FastAPI

## **1. Overview**

**Objectives:**

* Understand the basics of FastAPI and why it's popular.
* Learn about creating RESTful APIs with FastAPI.
* Get familiar with key concepts such as Pydantic models, CRUD endpoints, and middleware.
* Learn how to run and test a FastAPI application using Uvicorn and the auto-generated Swagger docs.

## **2. Introduction to FastAPI**

**Key Points:**

* **FastAPI Overview:** FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.6+ based on standard Python type hints. It provides automatic interactive API documentation using Swagger UI and Redoc.
* **Advantages:**
  + Easy to code and maintain.
  + Built-in data validation and serialization using Pydantic.
  + Asynchronous support for improved performance.
  + Automatic interactive API documentation.

## **3. Setting Up the Environment**

**Steps:**

**Installation:**  
pip install fastapi uvicorn

* **Project Structure:** Create a file called main.py for your FastAPI code.
* Requirements.txt file

## **4. Walkthrough of the Sample Code**

from fastapi import FastAPI, HTTPException, Request

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel

from typing import List, Optional

import time

import uvicorn

app = FastAPI()

# -----------------------------

# Middleware Setup

# -----------------------------

# CORS Middleware: Allow all origins (for demonstration)

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # In production, limit to specific origins

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# Custom Logging Middleware: Logs the request method, URL, and processing time.

@app.middleware("http")

async def log\_requests(request: Request, call\_next):

start\_time = time.time()

response = await call\_next(request)

process\_time = (time.time() - start\_time) \* 1000 # time in milliseconds

print(f"{request.method} {request.url} completed in {process\_time:.2f}ms")

return response

# -----------------------------

# Data Model and In-Memory "Database"

# -----------------------------

class Item(BaseModel):

id: int

name: str

description: Optional[str] = None

# Simulated database using a Python dictionary.

items\_db = {}

# -----------------------------

# CRUD Endpoints

# -----------------------------

# GET all items

@app.get("/items/", response\_model=List[Item])

def get\_items():

return list(items\_db.values())

# GET a specific item by id

@app.get("/items/{item\_id}", response\_model=Item)

def get\_item(item\_id: int):

if item\_id not in items\_db:

raise HTTPException(status\_code=404, detail="Item not found")

return items\_db[item\_id]

# POST a new item

@app.post("/items/", response\_model=Item)

def create\_item(item: Item):

if item.id in items\_db:

raise HTTPException(status\_code=400, detail="Item already exists")

items\_db[item.id] = item

return item

# PUT to update an existing item

@app.put("/items/{item\_id}", response\_model=Item)

def update\_item(item\_id: int, item: Item):

if item\_id not in items\_db:

raise HTTPException(status\_code=404, detail="Item not found")

items\_db[item\_id] = item

return item

# DELETE an item by id

@app.delete("/items/{item\_id}")

def delete\_item(item\_id: int):

if item\_id not in items\_db:

raise HTTPException(status\_code=404, detail="Item not found")

del items\_db[item\_id]

return {"detail": "Item deleted successfully"}

# -----------------------------

# Run the App

# -----------------------------

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run(app, host="0.0.0.0", port=8000)

**Explanation of the Code:**

* **Imports:** Explain each import and its role (e.g., FastAPI for the app, HTTPException for error handling, CORSMiddleware for CORS support, etc.).
* **Middleware Section:**
  + **CORS Middleware:** Discuss why CORS is important for web APIs.
  + **Custom Logging Middleware:** Explain how middleware can intercept requests and responses to log important details.
* **CRUD Endpoints:**
  + Walk through each endpoint, discussing:  
    - **GET:** Retrieving data.
    - **POST:** Creating new data (and handling duplicates).
    - **PUT:** Updating existing data.
    - **DELETE:** Removing data.
  + Emphasize error handling using HTTPException.
* **Running the App:** Explain the importance of the if \_\_name\_\_ == "\_\_main\_\_": block and using Uvicorn as the ASGI server.

## **5. Hands-On Exercise**

* **Task:** Ask students to run the sample application.
* **Testing:** Use a tool like cURL, Postman, or directly visit http://localhost:8000/docs in the browser to test the endpoints.
* **Challenge:** Have students add an extra endpoint (for example, a search endpoint) or modify the logging middleware to include more details (such as the client's IP address).

## 6. DIST Integration:

Import the required modules

* from fastapi.staticfiles import StaticFiles
* from fastapi.responses import FileResponse

Keep these methods in your code while adding dist in your application

* app.mount("/assets", StaticFiles(directory="dist/assets"), name="assets")
* @app.get("/{path:path}")

async def get\_index(request: Request, path: str):

return FileResponse("dist/index.html")