Lecture 23 – HTTP Methods and API Routers in FastAPI

As we near the project phase of this course, it is important to establish a solid understanding of two fundamental aspects of API development with FastAPI: **HTTP methods** and **API routers**. These concepts form the backbone of designing, organizing, and managing scalable web applications.

This lecture is divided into two sections:

1. **HTTP Methods** – GET, POST, PUT, DELETE
2. **API Routers** – Their role in organizing FastAPI applications

By mastering these concepts, you will be well-prepared to structure and implement larger projects in a clear and maintainable way.

---

1. HTTP Methods in FastAPI

HTTP methods define the actions that can be performed on resources in a RESTful API. FastAPI supports all standard HTTP methods, but four are most commonly used in web development: **GET, POST, PUT, and DELETE**.

1.1 GET Method – Retrieve Data

- **Purpose**: Fetch data from the server.
- **Analogy**: Asking a waiter for the restaurant menu.
- **Usage in APIs**: Retrieve items, records, or any resource without modifying data.

Example:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    """
    GET request at the root path ('/').
    Returns a welcome message.
    """
    return {"message": "Hello World - This is a GET request example!"}
```

When executed, this endpoint returns:

```
{"message": "Hello World - This is a GET request example!"}
```

1.2 POST Method – Create Resource

- **Purpose**: Send data to the server to create a new resource.
- **Analogy**: Placing an order with a waiter after selecting from the menu.
- **Usage in APIs**: Create new items, users, or records in a database.

Example:

```
from pydantic import BaseModel

items_db = {}

class Item(BaseModel):
    name: str
    price: float

@app.post("/items/")
def create_item(item: Item):
    """
    POST request to create a new item.
    Expects item data in the request body.
    Returns the item data with a new ID.
    """
    item_id = len(items_db) + 1
    items_db[item_id] = item.dict()
    return {"item_id": item_id, **item.dict()}
```

1.3 PUT Method – Update Resource

- **Purpose**: Modify or replace an existing resource.
- **Analogy**: Asking the waiter to change your order.
- **Usage in APIs**: Update resource details, such as editing an item's price.

Example:

```
from fastapi import HTTPException

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    """
    PUT request to update an item.
    If the item exists, it is replaced with new data.
    Otherwise, a 404 error is returned.
    """
    if item_id not in items_db:
        raise HTTPException(status_code=404, detail="Item not found")
```

```
    items_db[item_id] = item.dict()
    return {"item_id": item_id, **item.dict()}
```

1.4 DELETE Method – Delete Resource

- **Purpose**: Remove a resource from the server.
- **Analogy**: Canceling an order with the waiter.
- **Usage in APIs**: Delete items, users, or records.

Example:

```
@app.delete("/items/{item_id}")
def delete_item(item_id: int):
    """
    DELETE request to remove an item.
    Returns a confirmation message if successful,
    or 404 if the item does not exist.
    """
    if item_id not in items_db:
        raise HTTPException(status_code=404, detail="Item not found")

    del items_db[item_id]
    return {"message": f"Item {item_id} deleted successfully"}
```

Practical Example of CRUD with HTTP Methods

- **GET** → Retrieve all items
- **POST** → Add a new item
- **PUT** → Update an item by ID
- **DELETE** → Remove an item by ID

Together, these methods implement the full **CRUD** (Create, Read, Update, Delete) functionality, which is fundamental in API development.

---

2. Using API Routers in FastAPI

As applications grow larger, managing all endpoints in a single file becomes difficult. FastAPI provides **API routers** to modularize and organize endpoints.

2.1 What Are API Routers?

Routers allow developers to group related endpoints together. Instead of defining all routes in `main.py`, we can create smaller modules for different parts of the application.

Benefits:

- Better code organization
- Easier maintenance in large projects
- Logical separation of functionality

2.2 Example of API Router

```
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
from typing import Dict, List, Union

items_router = APIRouter()

items_data: Dict[int, Dict] = {
    101: {"name": "Laptop", "price": 1200},
    102: {"name": "Headphones", "price": 150},
    103: {"name": "Mouse", "price": 30},
}

class Item(BaseModel):
    name: str
    price: float

class ItemWithId(Item):
    item_id: int

@items_router.get("/", response_model=Dict[str, Union[str,
List[ItemWithId]]])
def get_all_items():
    items_list_with_ids = [
        {"item_id": item_id, **item_data} for item_id, item_data in
items_data.items()
    ]
    return {"message": "Here are all the items in our shop:", "items":
items_list_with_ids}

@items_router.get("/{item_id}", response_model=ItemWithId)
def get_item(item_id: int):
    if item_id in items_data:
        return {"item_id": item_id, **items_data[item_id]}
    raise HTTPException(status_code=404, detail="Item not found in this
department!")

@items_router.post("/", response_model=Dict[str, Union[str, int, Item]])
def create_new_item(item: Item):
    new_id = max(items_data.keys()) + 1 if items_data else 201
    items_data[new_id] = item.dict()
```

```
    return {"message": f"Item '{item.name}' added successfully!", "item_id":
new_id, **item.dict()}
```

The router (`items_router`) can then be included in the main application with:

```
from fastapi import FastAPI
from items import items_router   # assuming saved in items.py

app = FastAPI()
app.include_router(items_router, prefix="/items", tags=["Items"])
```

This way, all item-related endpoints are grouped under `/items`.

Final Thoughts

In this lecture, we covered:

- The four primary **HTTP methods** (GET, POST, PUT, DELETE) and their role in CRUD operations.
- How to implement these methods in FastAPI with practical examples.
- The concept of **API routers**, which provide better structure and scalability for applications.

Understanding these concepts is essential as we move toward building a **complete FastAPI project**. With HTTP methods and routers as your foundation, you will be able to design well-structured, scalable, and professional APIs.