Day 5 Notes

API, FLASK AND FASTAPI

---

1. What is an API?

API (Application Programming Interface) acts as a **middleware** between the **frontend** and the **backend**.

How it works

- Frontend sends a **request (payload)**
- Backend receives the payload
- Backend processes the request
- Backend sends a **response (output)** back to frontend

APIs usually exchange data in **JSON format**.

---

2. HTTP Methods (Three main API Methods)

I. GET

- Used to **fetch/read data**
- Payload is sent through the **URL (endpoint)**

Example:

```
GET /users/1
```

II. POST

- Used to **send or create data**
- Payload is sent through the **request body**

Example:

```
POST /users
```

III. DELETE

- Used to **delete data**
- Payload is usually sent via **URL or ID**

Example:

```
DELETE /users/1
```

---

3. Flask API

What is Flask?

- A **lightweight Python web framework**
- Used to build **simple APIs**
- Suitable for:
  - Mini projects
  - Learning
  - Basic production-level apps
- Helps create **URLs (endpoints)** easily

Flask Imports & Setup

```
from flask import flask     #creates the app
from flask import request     #accept payload from frontend
from flask import request Jsonify   #convert response into JSON format
```

Flask App Initialization

```
app = Flask(__name__)
```

Creating an Endpoint

```
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    return jsonify({"message": "Login successful"})
```

`@app.route()` is used to:

- Construct an **endpoint**
- Inside() you can initialize the prefix and suffix eg: project name
- Define **URL path**
- Specify **HTTP method**

---

4. FastAPI

What is FastAPI?

- A **modern Python framework** for building APIs
- Faster than Flask
- Flask is simpler
- Supports **asynchronous (async) programming**
- Best for:
  - High performance APIs
  - Scalable systems
  - Real-time applications

FastAPI Imports & Setup

from fastapi import FastAPI

```
app = FastAPI()
```

Creating Endpoints in FastAPI

```
@app.get("/users")
def get_users():
    return {"users": []}

@app.post("/users")
def create_user(user: dict):
    return {"message": "User created", "user": user}
```

FastAPI automatically:

- Validates data
- Generates API documentation

- Handles JSON efficiently

---

5. Flask vs FastAPI

Feature	Flask	FastAPI
Simplicity	*Very simple*	*Slight learning curve*
Speed	*Slower*	*Faster*
Async support	*Limited*	*Built-in async*
Documentation	*Manual*	*Auto-generated*
Use case	*Small apps*	*Modern scalable APIs*

---

6. Postman

Postman is a tool used to:

- Test APIs
- Send GET, POST, DELETE requests
- View API responses
- Debug backend APIs

Used heavily during backend development.

---

7. Types of Requests

HTTP Request

- Standard request-response model
- Client sends request → server responds once

WebSocket Request

- Two-way communication
- Used for:
    - Chat apps

    - Live notifications
    - Real-time updates

Streaming

- Data is sent in **small chunks**
- Instead of sending everything at once
- Used in:
    - Video streaming
    - Live data feeds
    - AI responses

---

8. Key Takeaways

- API connects frontend and backend
- GET → read data
- POST → send/create data
- DELETE → remove data
- Flask → simple and beginner-friendly
- FastAPI → modern, fast, async-based

Assignment I
Create a python code which is using the fast/flask api
Anti gravity
Milestone 1:
Daily doc
25% of project- login, basic structure, dashboard, database -mysql