RESTFUL API & FLASK

Q1 What is a RESTful API?
A **RESTful API** is a way for software systems to communicate over the internet using a set of simple, consistent rules based on **REST** (Representational State Transfer), an architectural style for designing networked applications.

Q2What is Flask, and why is it popular for building APIs?
Flask is a **lightweight Python web framework** used to build web applications and APIs. It gives you the essentials to handle HTTP requests, routing, and responses—without forcing a lot of structure or extra features on you.

Why Flask is popular for building APIs

1. Minimal and lightweight
2. **Easy to learn and use**
3. Perfect match for RESTful APIs
4. Highly flexible
5. Strong ecosystem and community

Q3. What are HTTP methods used in RESTful APIs?
In RESTful APIs, **HTTP methods** (also called verbs) define the action a client wants to perform on a resource. The most commonly used methods are **GET**, which retrieves data without modifying it; **POST**, which creates a new resource; **PUT**, which completely replaces an existing resource; **PATCH**, which partially updates a resource; and **DELETE**, which removes a resource. Additional methods like **HEAD** (to retrieve metadata without a response body) and **OPTIONS** (to discover supported methods or handle CORS preflight requests) are used less frequently. By combining clear resource-oriented URLs with these standardized HTTP methods,

RESTful APIs achieve predictable, scalable, and easy-to-understand communication between clients and servers.

Q4 What is the purpose of the @app.route() decorator in Flask?

The `@app.route()` **decorator** in Flask connects a URL endpoint to a Python function, telling Flask which function should handle requests made to that route.

Q5 What is the role of Flask-SQLAlchemy?

Flask-SQLAlchemy acts as the **database integration layer** for Flask applications. It simplifies working with databases by connecting Flask to **SQLAlchemy**, allowing you to define database tables as Python classes, perform queries using Python code instead of raw SQL, and manage database sessions efficiently within a Flask app.

Q6 How do you create a basic Flask application?

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

if __name__ == "__main__":
    app.run(debug=True)
```

Q7 :How do you return JSON responses in Flask?

```
from flask import jsonify

@app.route("/user")
def user():
    return jsonify(name="Alice", age=25)
```

Q8 :How do you handle POST requests in Flask?

```
from flask import request, jsonify

@app.route("/users", methods=["POST"])
def create_user():
    data = request.get_json()
    return jsonify(message="User created", user=data), 201
```

Q9 How do you handle errors in Flask (e.g., 404)?

```
from flask import jsonify

@app.errorhandler(404)
def not_found(error):
    return jsonify(error="Resource not found"), 404
```

Q10 How do you structure a Flask app using Blueprints?

```
# users/routes.py
from flask import Blueprint

users_bp = Blueprint("users", __name__)

@users_bp.route("/users")
def get_users():
    return "Users list"
```