EXPERIMENT NO. 3

Name-Spandan Deb
Class-D15A
Roll no-13

AIM: To develop a basic Flask application with multiple routes and demonstrate the handling of GET and POST requests.

PROBLEM STATEMENT:

Design a Flask web application with the following features:

1. A homepage (/) that provides a welcome message and a link to a contact form.
    a. Create routes for the homepage (/), contact form (/contact), and thank-you page (/thank_you).
2. A contact page (/contact) where users can fill out a form with their name and email.
3. Handle the form submission using the POST method and display the submitted data on a thank-you page (/thank_you).
    a. On the contact page, create a form to accept user details (name and email).
    b. Use the POST method to handle form submission and pass data to the thank-you page
4. Demonstrate the use of GET requests by showing a dynamic welcome message on the homepage when the user accesses it with a query parameter, e.g., /welcome?name=<user_name>.
    a. On the homepage (/), use a query parameter (name) to display a personalized welcome message.

Theory:

A) List some of the core features of Flask.

Flask is a lightweight and flexible web framework for Python. Some of its core features include:

1. Lightweight & Minimalistic – Flask is a microframework, meaning it provides only the essential tools needed for web development, making it easy to extend.
2. Built-in Development Server & Debugger – Comes with a built-in server for testing and an interactive debugger to identify errors efficiently.
3. Routing – Provides a simple way to map URLs to functions using decorators (@app.route()).
4. Template Engine (Jinja2) – Uses Jinja2 for dynamic HTML rendering with features like template inheritance and macros.
5. WSGI Compatibility – Works with WSGI (Web Server Gateway Interface) via Werkzeug, ensuring smooth request handling.
6. RESTful Request Handling – Supports handling different HTTP methods (GET, POST, PUT, DELETE) for building RESTful APIs.
7. Session Management – Supports secure client-side sessions using cookies.
8. Extensibility – Can be extended with third-party libraries for authentication, database integration (SQLAlchemy, MongoDB, etc.), and more.
9. Blueprints – Allows structuring large applications into smaller, reusable modules.
10. Integrated Support for Unit Testing – Provides tools to write and test applications effectively.

B) Why do we use Flask(__name__) in Flask?

In Flask, Flask(__name__) is used to create an instance of the Flask application. The __name__ variable, which represents the name of the current module, plays a crucial role in the framework's functionality.

Key Reasons:

1. Identifies the Application's Module – Helps Flask determine the application's root location.
2. Locates Static & Template Files – Ensures Flask can find static files (CSS, JS) and templates (HTML) in their respective directories.
3. Enables Debugging & Error Handling – Helps Flask differentiate between running as a script (__main__) or an imported module, allowing it to enable debugging features accordingly.
4. Supports Flask Extensions – Ensures proper integration of Flask extensions like Flask-SQLAlchemy and Flask-Login.

C)What is Template (Template Inheritance) in Flask?

A template in Flask refers to an HTML file that dynamically generates content using Jinja2, Flask's built-in templating engine. Template Inheritance allows multiple templates to share a common structure, reducing redundancy and improving code organization.

How Template Inheritance Works:

- A base template (base.html) contains the common layout (header, footer, navigation).
- Other templates extend ({% extends "base.html" %}) the base and override specific blocks ({% block content %}) to customize content.

Example: Parent template

```
<!DOCTYPE html>

<html>

<head>

   <title>{% block title %}My Website{% endblock %}</title>

</head>

<body>
```

```
    <header>Welcome to My Website</header>
    <div>{% block content %}{% endblock %}</div>
    <footer>Footer Section</footer>
</body>
</html>
```

Child Template

```
{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block content %}
    <h1>This is the Home Page</h1>
{% endblock %}
```

Template inheritance simplifies maintaining a consistent design across multiple pages

D)What methods of HTTP are implemented in Flask.

Flask supports multiple HTTP methods for handling different types of requests in a web application. The main methods implemented are:

1. Get – Retrieves data from the server (default method).

   ```
   @app.route('/data', methods=['GET'])
   def get_data():
   return "This is a GET request"
   ```

2. Post – Sends data to the server for processing.

   ```
   @app.route('/submit', methods=['POST'])
   def submit_form():
      return "This is a POST request"
   ```

3. Put-Updates existing data on the server

```
@app.route('/update', methods=['PUT'])
    def update_data():
     return "This is a PUT request"
```

4. Delete -Removes data from the server

```
@app.route('/delete', methods=['DELETE'])
def delete_data():
   return "This is a DELETE request"
```

5. Patch – Partially updates existing data.
6. Head – Similar to GET but without a response body.
7. Options – Returns allowed HTTP methods for a route.

E) What is difference between Flask and Django framework.

Feature	Flask 🟢 (Micro-framework)	Django 🔵 (Full-stack framework)
Architecture	Lightweight and modular, gives developers more flexibility	Monolithic, follows MVC (Model-View-Controller) pattern
Flexibility	Highly customizable, requires external libraries for many features	Comes with built-in features like authentication, admin panel, ORM
Database Support	No default database, uses SQLAlchemy or other ORMs	Built-in ORM, supports multiple databases (PostgreSQL, MySQL, SQLite)
Performance	Faster for small applications due to minimal overhead	Can be slower due to built-in features but optimized for large apps
Learning Curve	Easier to learn, minimal setup, less boilerplate code	Steeper learning curve due to conventions and built-in features
Best Suited For	Small to medium apps, RESTful APIs, microservices	Large-scale, data-driven web applications and enterprise-level projects

Routing

Routing in Flask is the process of mapping URLs to specific functions. This is done using the @app.route() decorator.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')  # Maps the root URL to this function

def home():

    return "Welcome to Flask!"

if __name__ == '__main__':

    app.run(debug=True)
```

URL building

Flask allows dynamic URL building using the url_for() function. This helps in avoiding hardcoded URLs and improves maintainability.

```
from flask import Flask, url_for

app = Flask(__name__)

@app.route('/user/<name>')  # Dynamic route

def user_profile(name):

    return f"Hello, {name}!"


@app.route('/get-url')

def get_url():

    return f"Profile URL: {url_for('user_profile', name='Spandan')}"


if __name__ == '__main__':

    app.run(debug=True)
```

GET REQUEST

A GET request is used to retrieve data from the server. It is the default request method in Flask.

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/greet', methods=['GET'])

def greet():

    name = request.args.get('name', 'Guest')  # Gets 'name' from the URL

    return f"Hello, {name}!"

if __name__ == '__main__':

    app.run(debug=True)
```

POST REQUEST

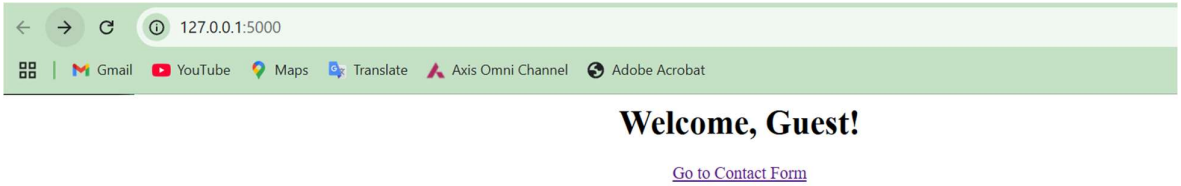
A POST request is used to send data to the server, typically in form submissions or API requests.

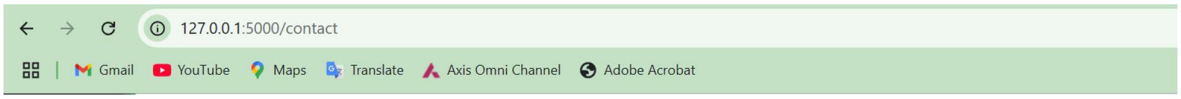
```
from flask import Flask, request

app = Flask(__name__)

@app.route('/greet', methods=['GET'])

def greet():

    name = request.args.get('name', 'Guest')  # Gets 'name' from the URL

    return f"Hello, {name}!"


if __name__ == '__main__':

    app.run(debug=True)
```

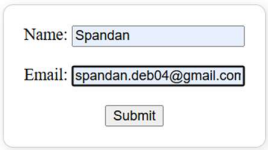
OUTPUT

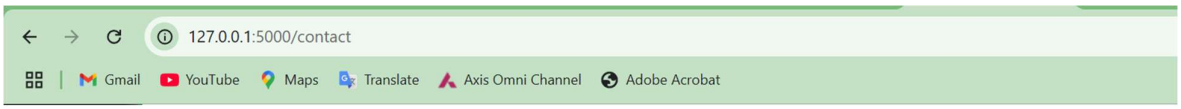
```
from flask import Flask, request, render_template

app = Flask(__name__)

# Homepage Route
@app.route('/')
def home():
    name = request.args.get('name', 'Guest')  # Get name from query parameter (default: Guest)
    return f"<h1 style='text-align: center;'>Welcome, {name}!</h1><p style='text-align: center;'><a href='/contact'>Go to Contact Form</a></p>"

# Contact Page Route
@app.route('/contact', methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        return f"<h1 style='text-align: center;'>Thank You!</h1><p style='text-align: center;'>Name: {name}</p><p style='text-align: center;'>Email: {email}</p><p style='text-align: center;'><a href='/'>Go Back</a></p>"
    return '''
    <div style='display: flex; justify-content: center; align-items: center; height: 100vh;'>
        <form action='/contact' method='post' style='text-align: center; padding: 20px; border: 1px solid #ccc; border-radius: 10px; box-shadow: 2px 2px 10px rgba(0,0,0,0.1);'>
            <label>Name:</label>
            <input type='text' name='name' required><br><br>
            <label>Email:</label>
            <input type='email' name='email' required><br><br>
            <button type='submit'>Submit</button>
        </form>
    </div>
    '''

# Thank You Page Route
```

```
@app.route('/thank_you')
def thank_you():
    return "<h1 style='text-align: center;'>Thank You for Your Submission!</h1><p style='text-align: center;'><a href='/'>Go Back</a></p>"

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









Thank You!

Name: Spandan

Email: spandan.deb04@gmail.com

Go Back