EXPERIMENT NO. 3- Flask

Name of Student	**Sneha Patra**
Class Roll No	**D15A_40**
D.O.P.	**20/2/2025**
D.O.S.	**27/2/2025**
Sign and Grade	

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:

List some of the core features of Flask

Flask is a lightweight and flexible web framework for Python, often described as "micro" because it provides the essentials to get a web application up and running without imposing unnecessary restrictions. Some core features of Flask include its minimalist design, which allows developers to add only what they need, built-in support for routing, templates, and handling HTTP requests and

responses. It uses the Jinja2 templating engine to generate dynamic HTML content and can be extended with various extensions like database support, authentication, and form handling.

Why do we use Flask(__name__) in Flask?

The reason we use Flask(__name__) is to create an instance of the Flask class. __name__ tells Flask if the script is being run directly or if it's being imported as a module into another script. When running directly, Flask knows to start the app, and when imported, it avoids running the app code unnecessarily.

What is Template (Template Inheritance) in Flask?

In Flask, a template is an HTML file with placeholders for dynamic content, usually populated with data passed from the Flask view functions. Template inheritance allows you to create a base template (with common elements like headers or footers) and extend it in other templates. This promotes reusability and helps keep your code DRY (Don't Repeat Yourself).

What methods of HTTP are implemented in Flask.

Flask supports several HTTP methods like GET (to retrieve data), POST (to send data), PUT (to update existing data), DELETE (to remove data), and OPTIONS (to describe communication options for the resource). These methods allow Flask to handle different types of interactions with the server.

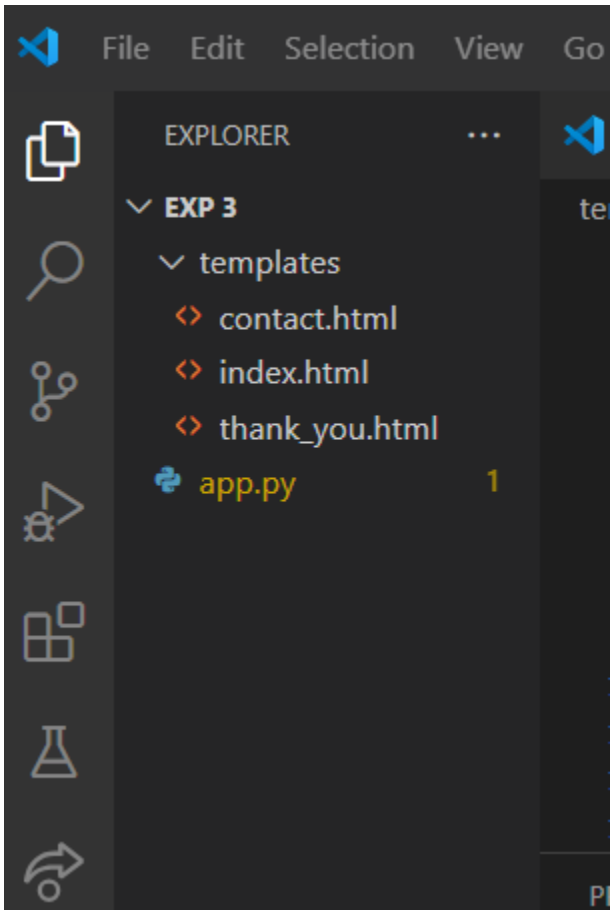
What is difference between Flask and Django framework

Feature	**Flask**	**Django**
Type	Lightweight, micro-framework	Full-stack, "batteries-included" framework
Flexibility	Highly flexible	Less flexible
Best For	Small to medium-sized projects	Larger, more complex projects
Database Integration	No built-in ORM	Built-in ORM
Community	Growing community	Larger community

GitHub Link: https://github.com/Sneha0321/WebX_EXP3

CODE:

Folder Structure:




```python
# app.py
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

@app.route('/')
def homepage():
    name = request.args.get('name')  # Get the name from query parameter
    if name:
        message = f"Welcome, {name}!"
    else:
        message = "Welcome to the homepage!"
```

```
    return render_template('index.html', message=message)

@app.route('/contact', methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        return redirect(url_for('thank_you', name=name, email=email))
    return render_template('contact.html')

@app.route('/thank_you')
def thank_you():
    name = request.args.get('name')
    email = request.args.get('email')
    return render_template('thank_you.html', name=name, email=email)

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

```
<!-- contact.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contact Form</title>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
        <button type="submit">Submit</button>
    </form>
```

```
    <p><a href="{{ url_for('homepage') }}">Back to homepage</a></p>
</body>
</html>
```

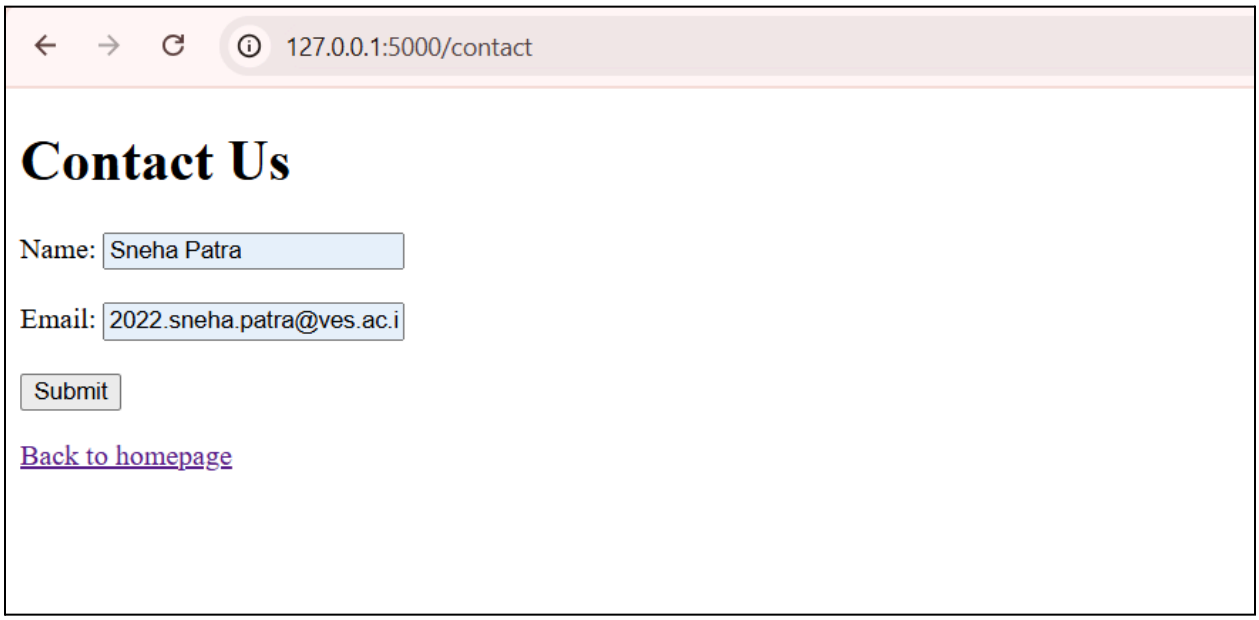
```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Homepage</title>
</head>
<body>
    <h1>{{ message }}</h1>
    <p><a href="{{ url_for('contact') }}">Go to the contact form</a></p>
</body>
</html>
```

```
<!-- thank_you.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Thank You</title>
</head>
<body>
    <h1>Thank You for Contacting Us!</h1>
    <p>Name: {{ name }}</p>
    <p>Email: {{ email }}</p>
    <p><a href="{{ url_for('homepage') }}">Back to homepage</a></p>
</body>
</html>
```

OUTPUT :

Successfully displayed the homepage with a personalized welcome message.

Displayed the contact form for user input.



Redirected to the thank-you page with user-submitted data.

CONCLUSION:

The Flask application was successfully developed to demonstrate GET and POST request handling. The implementation included multiple routes, form submission, and dynamic rendering using templates. This experiment provided hands-on experience with Flask's core functionalities like routing, request handling, and template inheritance.