EXPERIMENT NO. 3

Name of Student	Shreya Sawant
Class Roll No	D15A 53
D.O.P.	
D.O.S.	
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:

A. List some of the core features of Flask
B. Why do we use Flask(__name__) in Flask?
C. What is Template (Template Inheritance) in Flask?
D. What methods of HTTP are implemented in Flask?
E. What is difference between Flask and Django framework

1. **List some of the core features of Flask**

Core Features of Flask:

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

- **Lightweight**: Flask is a micro-framework, meaning it does not include many built-in features, allowing for easy customization and flexibility.
- **Routing**: Flask uses decorators to handle HTTP requests and map them to Python functions (routes).
- **Template Engine (Jinja2)**: Flask integrates with Jinja2, a powerful template engine, to generate dynamic HTML content.
- **RESTful Request Handling**: Flask easily supports RESTful request handling for APIs.
- **Development Server**: It comes with an inbuilt development server, making it easy to test applications locally.
- **Extensible**: Flask is highly extensible, allowing for the addition of various libraries and tools for functionalities like database integration (e.g., SQLAlchemy).
- **Session Management**: Flask provides session management, allowing for persistent user data across requests.
- **Debug Mode**: Flask has a built-in debug mode that helps developers catch errors in development.

2. **Why do we use Flask(__name__) in Flask?**
    - Flask(__name__) is used to create an instance of the Flask class. It tells Flask where to find the application and its resources.
    - __name__ is passed to indicate where the application is defined (usually the main entry point). It helps Flask determine where to look for static files, templates, and other resources.
        - When __name__ is used, Flask knows whether it's being run directly or N imported as a module.
        - __name__ is used for resource loading and debugging.

3. **What is Template (Template Inheritance) in Flask?**

Templates in Flask are files that contain placeholders for dynamic content. They are typically written in HTML, with embedded Python code for dynamic content (using the Jinja2 template engine).

Template Inheritance: Flask allows you to create a base template that defines the structure of the page, and child templates can extend it and add their own content.

- **Base Template**: Defines common structure, like headers, footers, and navigation.
- **Child Templates**: Extend the base template and inject specific content into defined blocks.

4. **What methods of HTTP are implemented in Flask?**

HTTP Methods in Flask:

Flask, a lightweight web framework for Python, supports various HTTP methods that can be used to handle different types of HTTP requests. The primary HTTP methods are:

1. **GET** – Used to request data from a specified resource. This is the most common HTTP method.
2. **POST** – Used to send data to a server to create a resource.
3. **PUT** – Used to update a current resource with new data.
4. **DELETE** – Used to delete a resource.
5. **PATCH** – Used to apply partial modifications to a resource.
6. **OPTIONS** – Used to describe the communication options for the target resource.
7. **HEAD** – Similar to GET, but it only returns the headers and no body content.

In Flask, you can define route handlers for each of these methods using the `@app.route` decorator with the `methods` parameter:

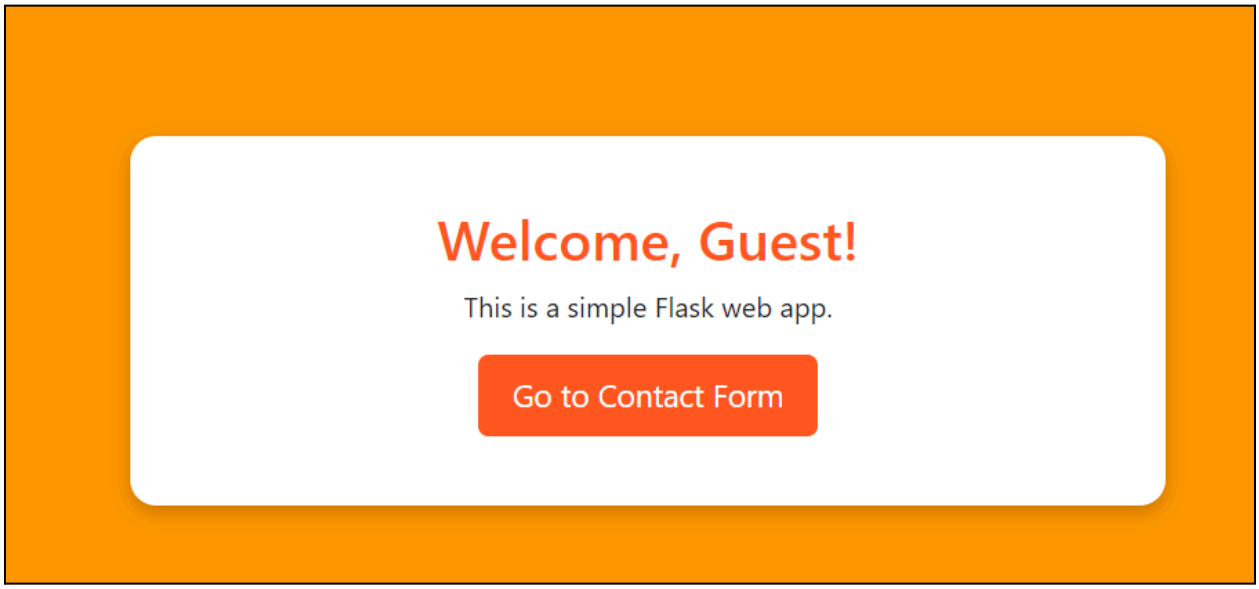
```
@app.route('/example', methods=['GET', 'POST'])
def example():
    if request.method == 'GET':
        return 'This is a GET request'
    elif request.method == 'POST':
        return 'This is a POST request'
```

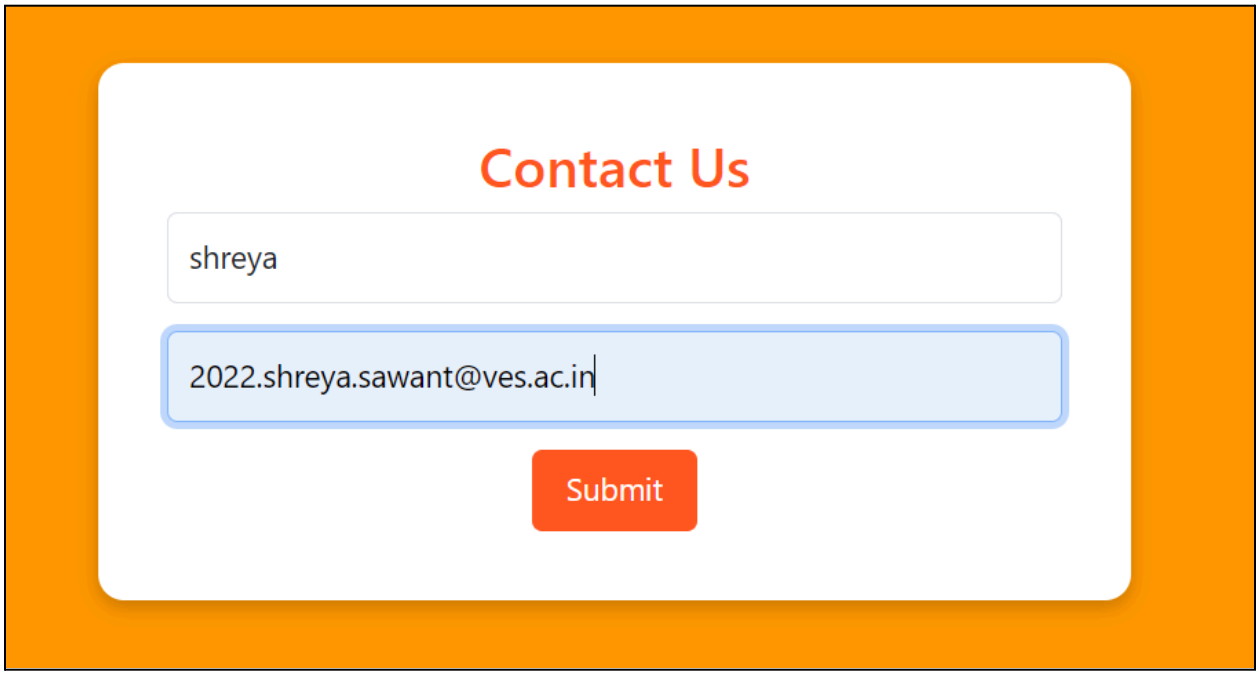
5. What is difference between Flask and Django framework

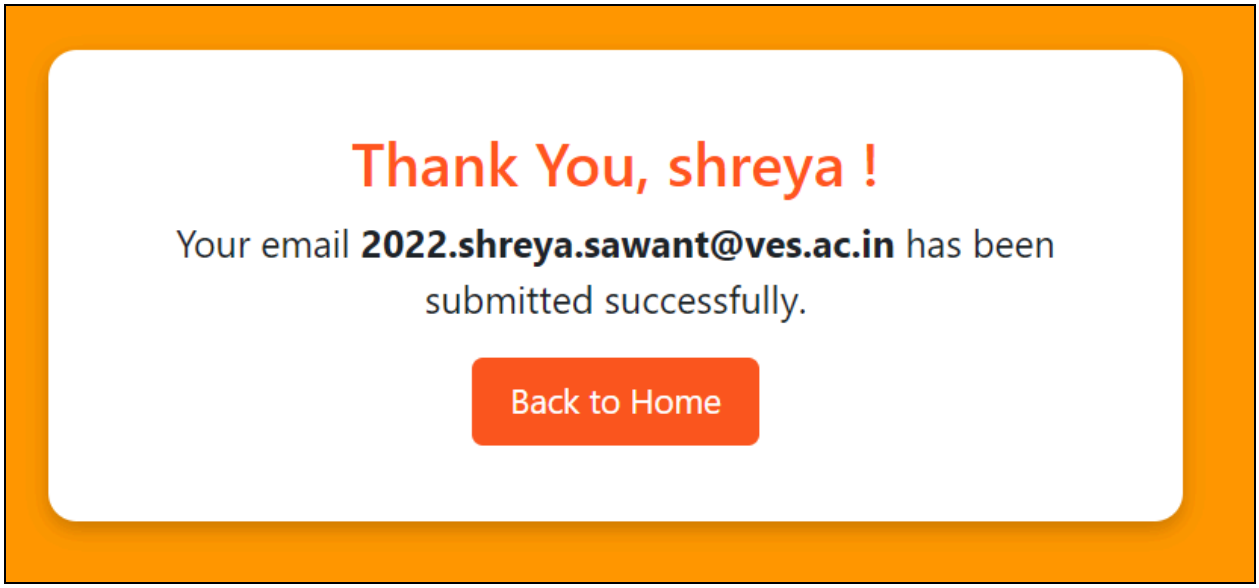
Feature	Flask	Django
Framework Type	Micro-framework	Full-stack web framework
Complexity	Lightweight, minimal structure	Comes with many built-in features
Flexibility	High – Developers choose components	Low – Follows "Django way" of doing things
Admin Interface	Not included (requires manual setup)	Built-in admin panel
Database Support	No built-in ORM (commonly uses SQLAlchemy)	Built-in ORM with migrations
Routing	Simple and manual routing	URL routing with patterns (`urls.py`)
Template Engine	Jinja2	Django Template Language (DTL)
Scalability	Suitable for small to medium apps	Suitable for medium to large applications
Learning Curve	Gentle – Easy to learn and get started	Steep – More features to understand initially
Community Support	Good, but smaller than Django	Large and mature community
Development Speed	Quick for small apps and APIs	Fast for large apps due to built-in tools
Best For	Microservices, REST APIs, quick prototypes	CMS, full-scale web apps, admin dashboards

Github link - https://github.com/sawantshreya11/WEBX-EX.3_53

OUTPUT :







Conclusion -

In this experiment, we developed a basic Flask web application with multiple routes and demonstrated the handling of both GET and POST requests. The application consisted of a homepage, a contact form, and a thank-you page.

We created a contact form that accepted user inputs like name and email, and used the POST method to process and display the submitted data on a separate page. Furthermore, we implemented the use of GET requests with a query parameter to display a personalized welcome message on the homepage.