EXPERIMENT NO. 3

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. Some of its core features include:

- **Minimalistic & Lightweight**: Flask is a micro-framework that provides only essential tools, making it easy to extend.
- **Built-in Development Server**: It comes with a built-in server for testing applications.
- **Routing**: Supports URL routing, allowing you to map URLs to specific functions.
- **Jinja2 Templating**: Uses Jinja2 for dynamic HTML generation.
- **Request Handling**: Supports handling GET, POST, and other HTTP requests.
- **Middleware Support**: Easily integrates with third-party extensions like authentication, database handling, and more.
- **RESTful API Support**: Ideal for building APIs due to its simplicity and flexibility.

Why do we use Flask(_name_) in Flask?

When creating a Flask application, we initialize it with `Flask(__name__)` where `__name__` refers to the name of the current module. This is important because:

- It helps Flask determine the root path of the application, which is essential for locating static files, templates, and configurations.
- It allows Flask to differentiate between the main module and imported modules, which helps with debugging and running the application correctly.

What is Template (Template Inheritance) in Flask?

Templates in Flask use **Jinja2**, a templating engine that allows dynamic content rendering.

- **Template Inheritance** is a feature that helps avoid repetitive code by using a base template (`base.html`) that child templates can extend.
- The base template contains common elements like headers, footers, and navigation bars.
- Child templates use `{% extends "base.html" %}` and override sections with `{% block content %} ... {% endblock %}`.

Example:
base.html

html

CopyEdit

```
<!DOCTYPE html>

<html>

<head><title>{% block title %}My Site{% endblock %}</title></head>

<body>

    <header>Welcome to My Website</header>

    <div>{% block content %}{% endblock %}</div>

    <footer>Copyright 2025</footer>

</body>
```

```
</html>
```

home.html (Child Template)

html

CopyEdi

t

```
{% extends "base.html" %}

{% block content %}

<h1>Home Page</h1>

<p>Welcome to the homepage!</p>

{% endblock %}
```

What methods of HTTP are implemented in Flask.

Flask supports multiple **HTTP methods**, including:

- **GET**: Used to retrieve data from the server.
- **POST**: Used to send data to the server (e.g., form submissions).
- **PUT**: Used to update existing resources.
- **DELETE**: Used to delete resources from the server.
- **PATCH**: Partially updates an existing resource.
- **OPTIONS**: Provides information about the available HTTP methods for a resource.

What is difference between Flask and Django framework

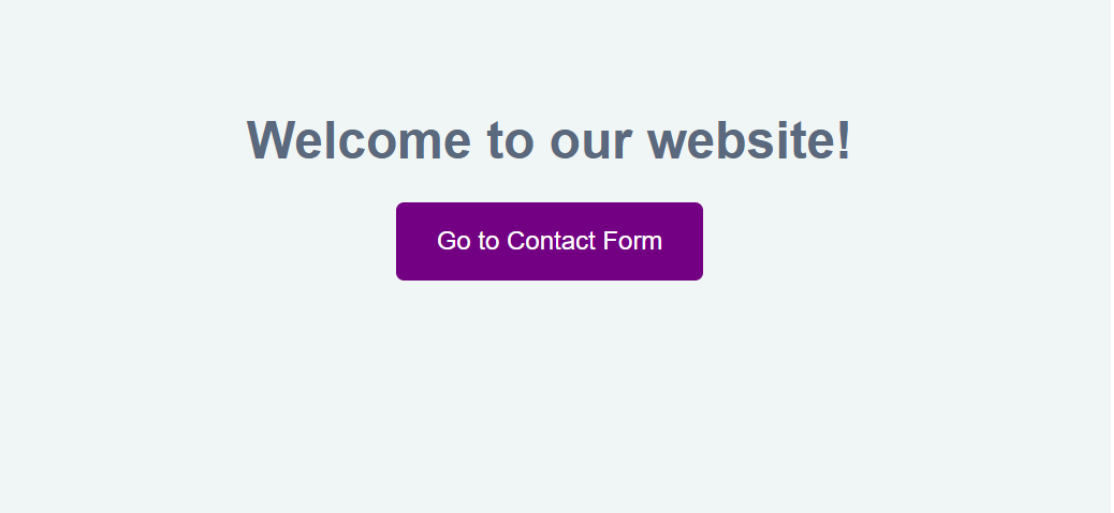
Feature	Flask	Django
Type	Micro-framework (lightweight)	Full-stack framework
Flexibility	Highly flexible; developers choose libraries	Opinionated; comes with built-in features

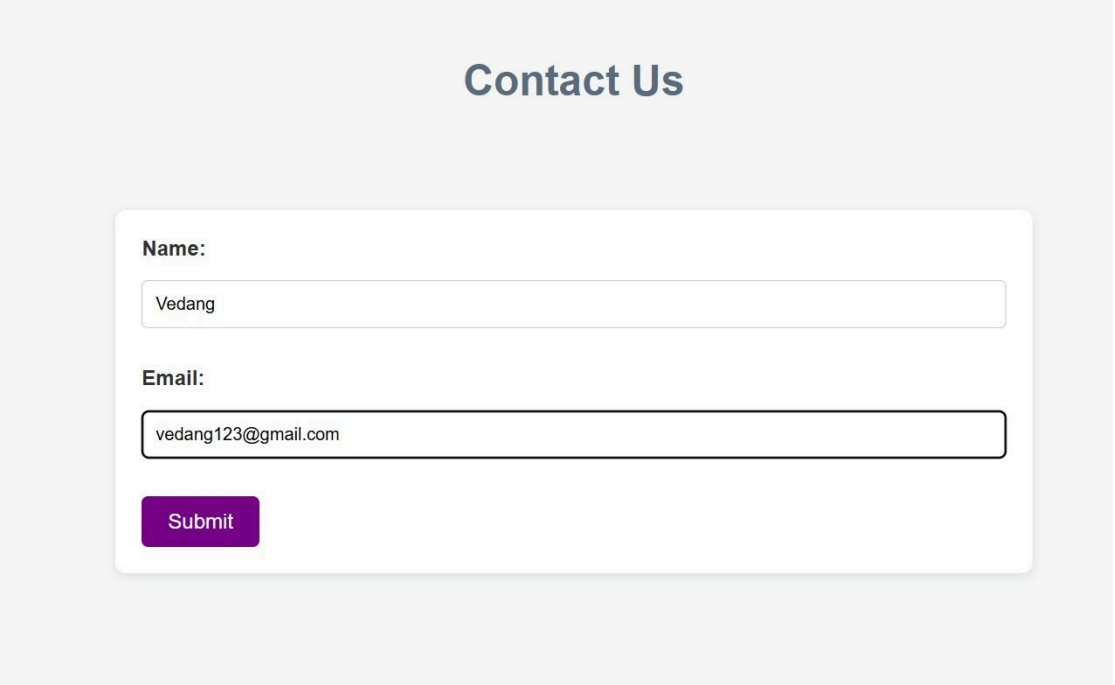
Learning Curve	Easier to learn, minimal setup	Steeper learning curve due to built features
Database Suppor	No built-in ORM, but supports SQLAlchem	Comes with Django ORM for databa management
Template Engine	Jinja2	Django's templating engine
Use Case	Best for small to medium projects and APIs	Ideal for large, enterprise-level applicatio

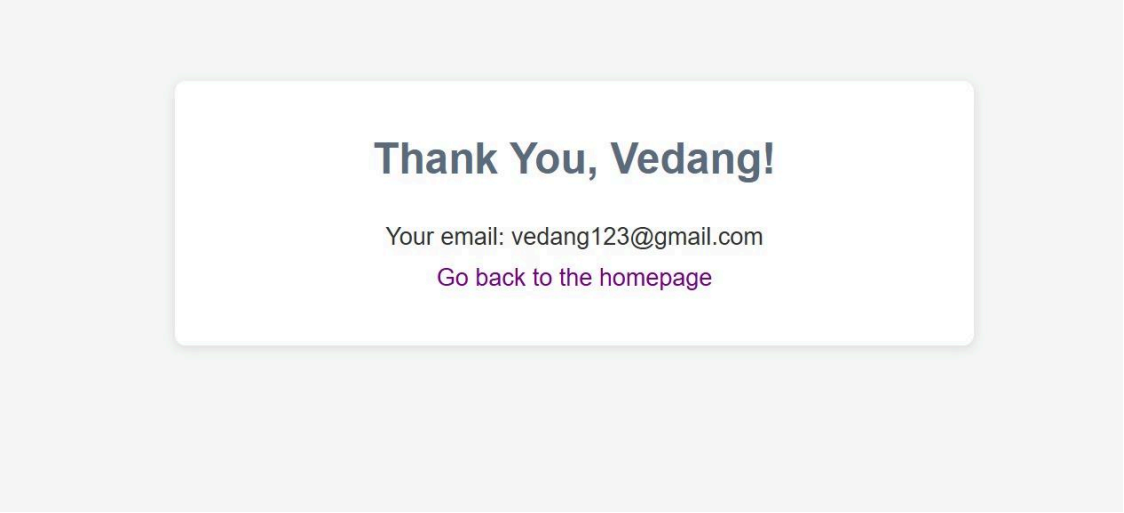
app.py

```
from flask import Flask, render_template, request, redirect, url_for
app = Flask(_name_)
@app.route('/')
def home():
  name = request.args.get('name', '')
  if name:
    message = f"Welcome, {name}!"
  else:
    message = "Welcome to our website!"

  return render_template('home.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)
```

OUTPUT


Welcome to our website!
Go to Contact Form


Contact Us
Name:
Vedang
Email:
vedang123@gmail.com
Submit


Thank You, Vedang!
Your email: vedang123@gmail.com
Go back to the homepage