EXPERIMENT NO. 3

Name of Student	**Aryan Dangat**
Class Roll No	**12**
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

Flask is a lightweight, **micro-framework** for Python that helps build web applications easily. It is designed to be simple yet powerful.

Key Features of Flask:

1. **Minimal and Fast** – Flask is designed to be lightweight and does not come with unnecessary components.
2. **Built-in Development Server** – It has an inbuilt server for testing applications locally.
3. **Flexible & Extensible** – Developers can integrate third-party libraries and extensions like Flask-SQLAlchemy, Flask-Login, etc.
4. **Jinja2 Templating Engine** – Supports dynamic HTML rendering.
5. **URL Routing System** – Maps URLs to Python functions using @app.route().
6. **Supports RESTful APIs** – Ideal for creating web services.
7. **Session and Cookie Management** – Supports secure user authentication.
8. **Integrated Debugging & Error Handling** – Comes with a built-in debugger.

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

When creating a Flask app, we initialize it using:

```
from flask import Flask
app = Flask(__name__)
```

__name___represents the current module, allowing Flask to correctly locate resources (templates, static files, etc.).
Distinguishes whether the script is being run directly or imported into another module.
Used for setting up configurations like file paths, logging, and debugging

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

Jinja is a web template engine for the Python programming language . We have seen that webpages of a website contains same footer , navigation bar etc. So instead of making same footer and navigation bar in all webpages separately , we make use of template inheritance , which allows us to create the part which is same in all webpages (eg. footer,navigation bar) only once and we also don’t need to write the html , head , title tag again and again .

D. What methods of HTTP are implemented in Flask.

Method	Description
GET	Fetches data from the server.
POST	Sends data to the server (e.g.,] form submission).
PUT	Updates an existing resource.
DELETE	Removes a resource from the server.
PATCH	Partially updates an existing resource.

E. What is difference between Flask and Django framework

Feature	**Flask**	**Django**
Framework Type	Micro-framework (lightweight)	Full-stack framework (batteries included)
Flexibility	More customizable	Comes with built-in features
Use Case	Best for small projects, APIs, and microservices	Best for large applications
Database Support	Uses SQLAlchemy or any other ORM	Has built-in Django ORM
Admin Panel	No built-in admin panel	Comes with an admin panel
Routing	Uses @app.route() decorators	Uses urls.py with path functions

Routing :- Routing is the process of mapping URLs to specific Python functions.

```
@app.route('/')
def home():
    return "Welcome to the Home Page!"
```

URL building : - Instead of hardcoding URLs, Flask provides the url_for() function to dynamically generate links.

```
from flask import url_for
@app.route('/profile/<username>')
def profile(username):
    return f"Welcome, {username}!"

with app.test_request_context():
    print(url_for('profile', username='Alex'))  # Output: /profile/Alex
```

GET REQUEST : - A GET request **retrieves data** from the server.

```
@app.route('/welcome')
def welcome():
    name = request.args.get('name', 'Guest')
    return f"'Welcome, {name}!"'
```

POST REQUEST : - A POST request **submits data** to the server.

```
@app.route('/submit', methods=['POST'])
def submit():
    name = request.form['name']
    return f"Thanks for submitting, {name}!"
```

CODE

App.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def home():
    name = request.args.get('name', 'Guest')
    return render_template('index.html', name=name)

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

@app.route('/thank_you')
def thank_you():
    return "Thank you for your submission!"

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

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home</title>
   <link rel="stylesheet" href="/static/styles.css">
</head>
<body>
   <div class="container">
      <h1>Welcome, {{ name }}!</h1>
      <p>This is a simple Flask Applicationn.</p>
      <a href="/contact" class="btn">Contact Us</a>
   </div>

   <div class="footer">&copy; 2025 Flask Contact App</div>
</body>
</html>
```

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 Us</title>
   <link rel="stylesheet" href="/static/styles.css">
</head>
<body>
    <div class="container">
       <h1>Contact Us</h1>
       <form action="/contact" method="post">
          <label for="name">Name:</label>
          <input type="text" id="name" name="name" required>

          <label for="email">Email:</label>
          <input type="email" id="email" name="email" required>

          <button type="submit" class="btn">Submit</button>
       </form>
   </div>

   <div class="footer">&copy; 2025 Flask Contact App</div>
</body>
</html>
```

Thank_you.html

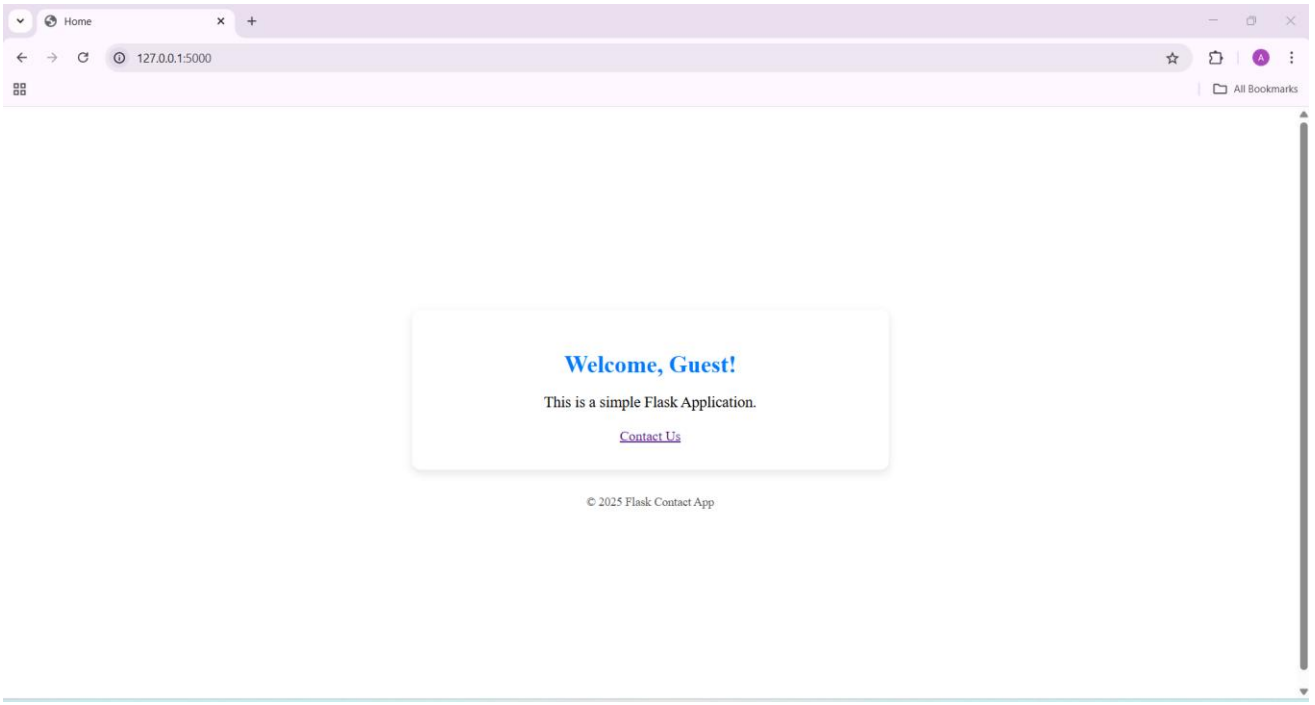
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Thank You</title>
    <link rel="stylesheet" href="/static/styles.css">
</head>
<body>
    <div class="container">
        <h1>Thank You!</h1>
        <p>We appreciate your message, <strong>{{ name }}</strong>!</p>
        <p>We will get back to you at <strong>{{ email }}</strong> soon.</p>
        <a href="/" class="btn">Back to Home</a>
    </div>

    <div class="footer">&copy; 2025 Flask Contact App</div>
</body>
</html>
```

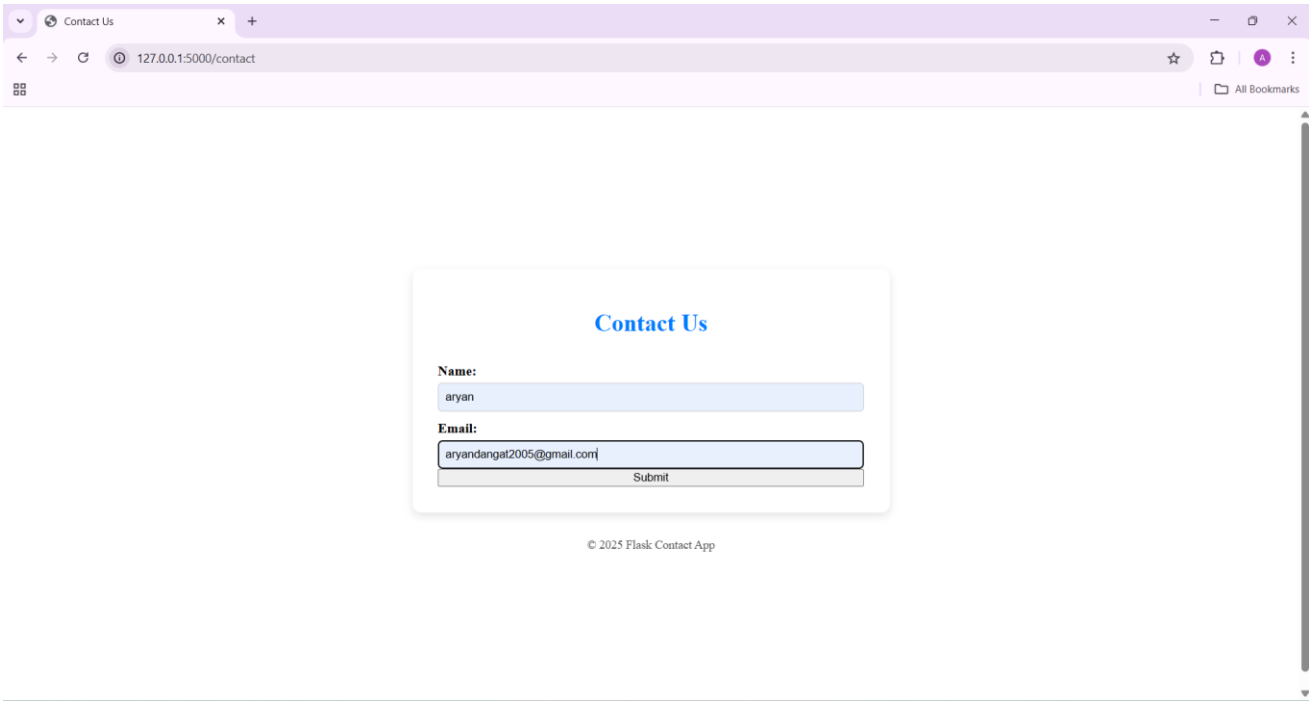
Styles.css

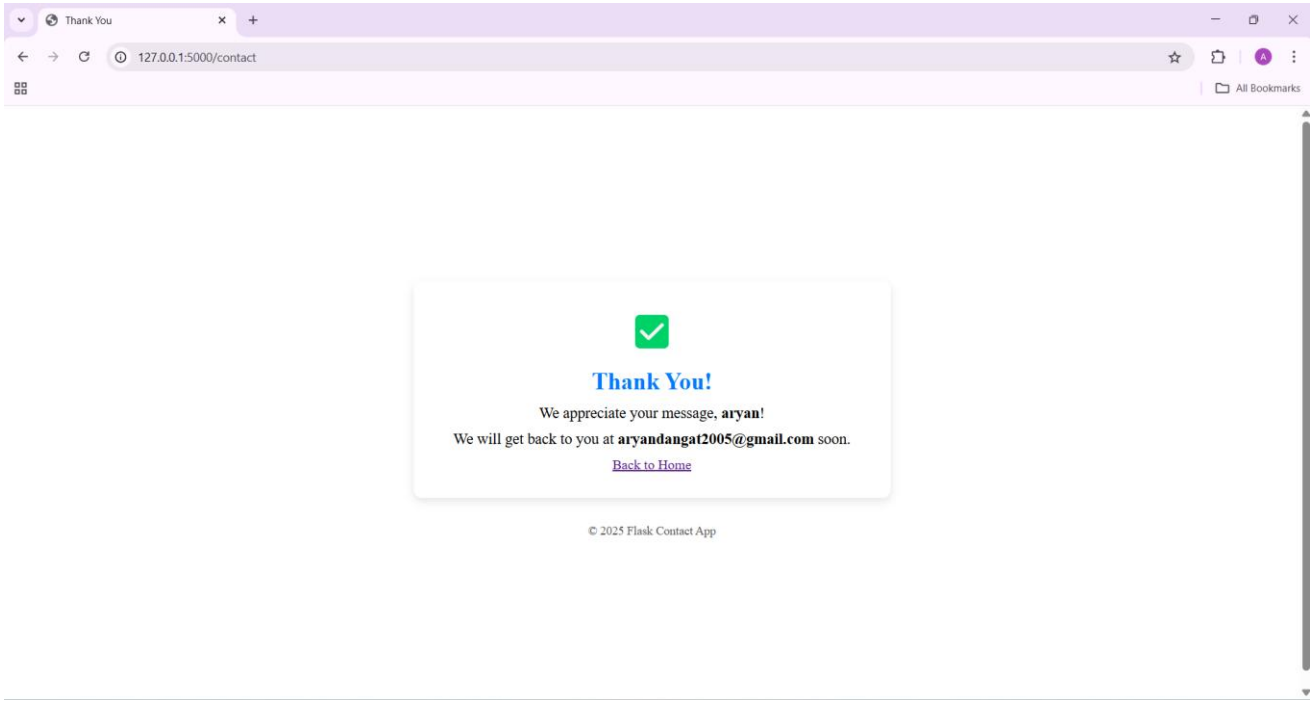
```
/* General Styling */
body {
    font-family: 'Arial', sans-serif;
    background: linear-gradient(to right, #e0f7fa, #80deea);
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    flex-direction: column;
}

/* Container */
.container {
    background: white;
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.1);
    width: 400px;
    text-align: center;
}
```

```
/* Buttons */
.btn {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 10px 15px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
    transition: 0.3s;
}

.btn:hover {
    background-color: #0056b3;
}


a {
    text-decoration: none; /* Removes underline */
    color: white; /* Ensures link color matches button */
}

a:hover {
    opacity: 0.8; /* Optional: Adds hover effect */
}

/* Form Styling */
form {
    text-align: left;
}

input {
    width: 100%;
    padding: 8px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 5px;
}

/* Footer */
.footer {
    margin-top: 20px;
    font-size: 14px;
    color: #555;
}
```

OUTPUT : -


Home
127.0.0.1:5000
All Bookmarks
Welcome, Guest!
This is a simple Flask Application.
Contact Us
© 2025 Flask Contact App


Contact Us
127.0.0.1:5000/contact
All Bookmarks
Contact Us
Name:
aryan
Email:
aryandangat2005@gmail.com
Submit
© 2025 Flask Contact App



Conclusion: -

The Flask application successfully demonstrates handling multiple routes and processing both GET and POST requests. The homepage dynamically personalizes the welcome message using query parameters, while the contact form collects user details and submits them via POST. The submitted data is then displayed on a thank-you page, showcasing form handling and data transfer between routes. This implementation highlights Flask's ability to create interactive web applications with efficient request handling and user input management.