Experiment 3 : Flask Application

Name of Student	**Sonam Chhabaidiya**
Class Roll No	**D15A_09**
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:-

1. Core Features of Flask

Flask is a lightweight and flexible web framework for Python. It is widely used for developing web applications due to its simplicity and scalability. The core features of Flask include:

1. **Lightweight and Modular** – Flask has a small core and allows developers to add extensions as needed.
2. **Built-in Development Server and Debugger** – It provides an interactive debugger and a development server for testing applications.
3. **Routing Mechanism** – It allows defining URL patterns for handling different types of requests.
4. **Jinja2 Templating Engine** – Flask supports Jinja2, which enables dynamic HTML rendering with the use of variables and logic.
5. **Integrated Unit Testing Support** – Flask includes features to test applications efficiently.
6. **Support for HTTP Methods** – Flask handles different HTTP methods such as GET, POST, PUT, and DELETE.
7. **Session and Cookie Management** – Flask allows managing user sessions and cookies securely.
8. **Blueprints for Modular Applications** – It enables breaking large applications into smaller, reusable modules.

---

2. Why do we use `Flask(__name__)` in Flask?

In Flask, the statement `Flask(__name__)` is used to create an instance of the Flask application. The `__name__` parameter is essential for the following reasons:

1. **Determining the Root Path** – Flask uses `__name__` to locate resources such as templates and static files.
2. **Enabling Debugging Features** – It helps in identifying the correct module name when debugging errors.
3. **Handling Routing Properly** – It ensures that Flask knows where the application is being executed from.

Thus, `Flask(__name__)` plays a crucial role in setting up a Flask application correctly.

---

3. What is Template and Template Inheritance in Flask?

Flask uses the **Jinja2 templating engine** to separate logic from presentation, making HTML files more dynamic and reusable.

Template Inheritance allows a developer to create a base template and extend it in child templates. This helps in maintaining a consistent layout across multiple pages.

Base Template (`base.html`)
html
CopyEdit

```
<!DOCTYPE html>
<html>
<head><title>{% block title %}My Website{% endblock
%}</title></head>
<body>
    <header>Header Section</header>
    <main>{% block content %}{% endblock %}</main>
</body>
</html>
```

-

Child Template (`index.html`)
html
CopyEdit

```
{% extends "base.html" %}
{% block title %}Home Page{% endblock %}
{% block content %}<p>Welcome to my website!</p>{% endblock %}
```

-

This mechanism ensures code reusability and efficient web page management.

---

4. What HTTP Methods are Implemented in Flask?

Flask supports multiple HTTP methods, primarily:

1. **GET** – Retrieves data from the server.

2. **POST** – Sends data to the server, often used for form submissions.
3. **PUT** – Updates existing resources on the server.
4. **DELETE** – Deletes a resource from the server.

Example in Flask:

python
CopyEdit

```
from flask import Flask, request

app = Flask(__name__)

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

---

5. Difference Between Flask and Django

Flask and Django are both popular Python web frameworks, but they have key differences:

Feature	Flask	Django
Type	Micro-framework	Full-stack framework
Flexibility	More flexible, requires external libraries	Comes with built-in features
Routing	Manually defined	Automatic routing support
ORM Support	Needs extensions like SQLAlchemy	Comes with Django ORM
Template Engine	Jinja2	Django Template Language (DTL)
Best For	Small to medium applications	Large-scale applications

Flask is preferred for lightweight applications, while Django is suitable for complex projects requiring built-in functionalities.

---

6. Routing in Flask

Routing in Flask refers to mapping a URL to a specific function. It helps in handling different requests and serving appropriate responses.

Example:

python
CopyEdit

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

This means that when a user visits `/home`, the `home()` function executes.

---

7. URL Building in Flask

Flask provides `url_for()` to dynamically generate URLs based on function names.

Example:

python
CopyEdit

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

# Generating URL
url_for('profile', username='JohnDoe')  # Output:
/profile/JohnDoe
```

This ensures flexibility and avoids hardcoding URLs.

---

8. GET Request in Flask

A **GET request** is used to fetch data from a server.

Example:

python
CopyEdit

```
@app.route('/user', methods=['GET'])
def get_user():
    return "User Information"
```

Visiting `/user` in a browser triggers the `get_user()` function, which returns user details.

---

9. POST Request in Flask

A **POST request** is used to send data to the server.

Example:

python
CopyEdit

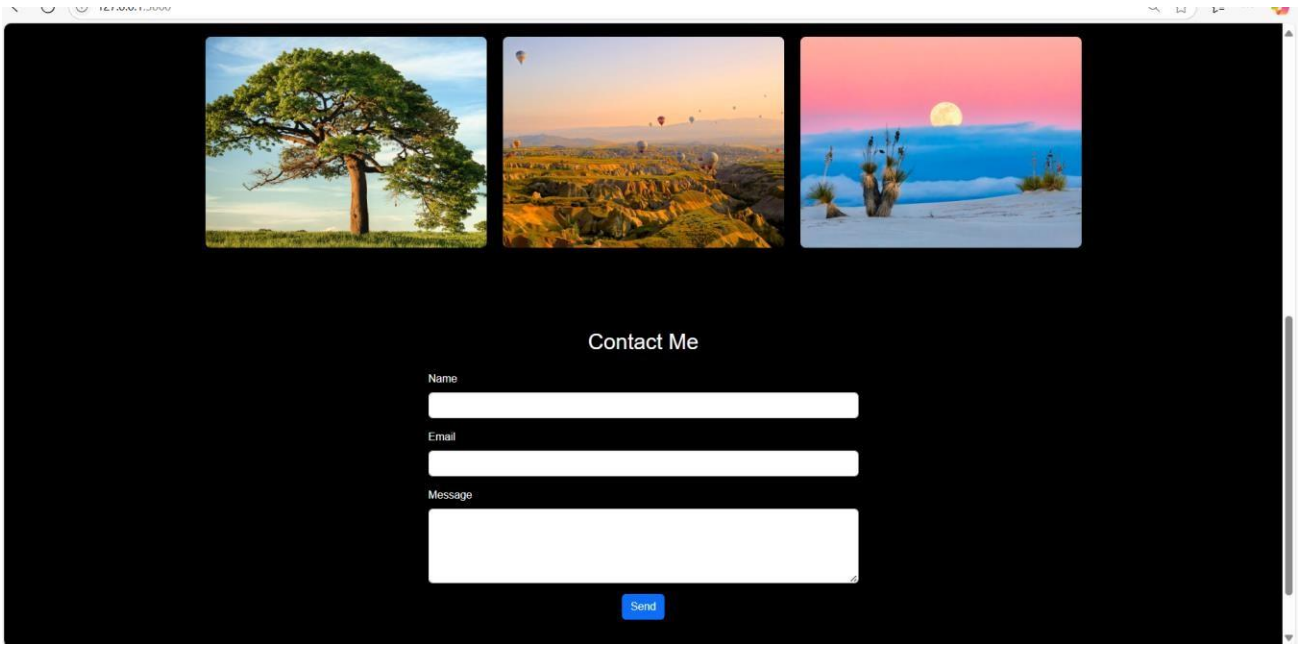
```
@app.route('/submit', methods=['POST'])
def submit_data():
    data = request.form['name']
    return f"Received: {data}"
```

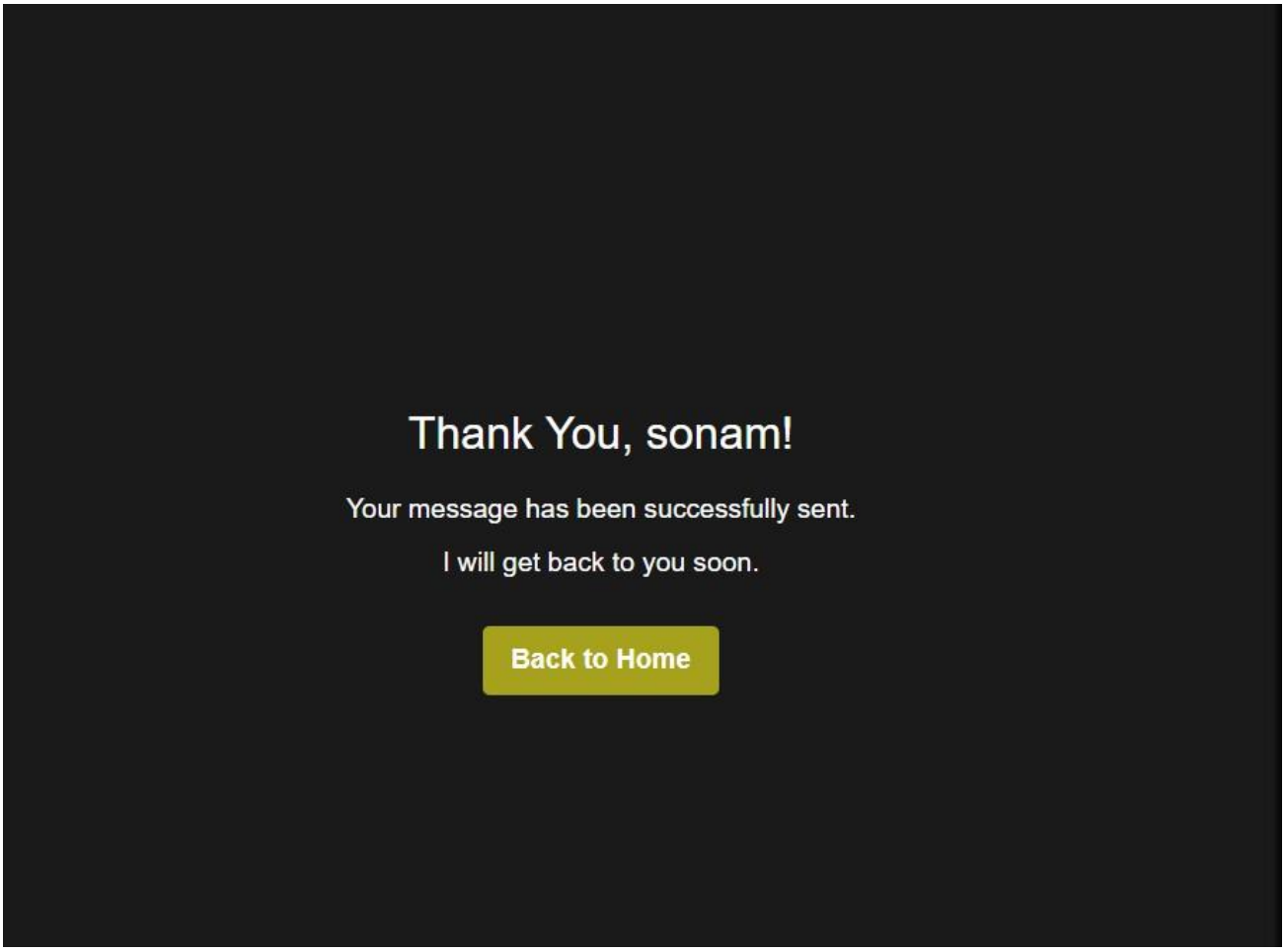
When a user submits a form, the server processes the data and returns a response.

GITHUB LINK: https://github.com/sonamcc/webbbx3

<

OUTPUT:






```
 * Debugger is active!
 * Debugger PIN: 127-349-387
127.0.0.1 - - [23/Mar/2025 18:01:23] "POST /submit HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2025 18:01:23] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [23/Mar/2025 18:01:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2025 18:01:45] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [23/Mar/2025 18:07:28] "POST /submit HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2025 18:07:28] "GET /static/styles.css HTTP/1.1" 304 -
```

CONCLUSION

In this experiment, a Flask web application was successfully developed to demonstrate the handling of GET and POST requests with multiple routes.
The application included:

- A homepage (/) that displayed a static or dynamic welcome message based on query parameters.
- A contact page (/contact) where users could submit their name and email via a form.
- A thank-you page (/thank_you) that dynamically displayed the submitted form data.

Through this implementation, key Flask features such as routing, request handling, form submission, template rendering, and URL parameter passing were explored. This experiment effectively demonstrated Flask's capability to create interactive and dynamic web applications with minimal code.