Snehal A. Patil D15A 38

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:

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

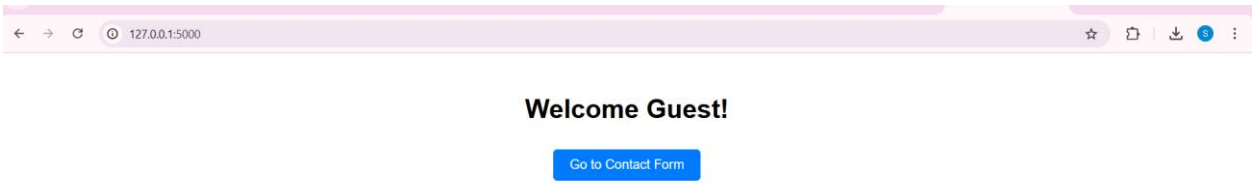
Routing

URL building

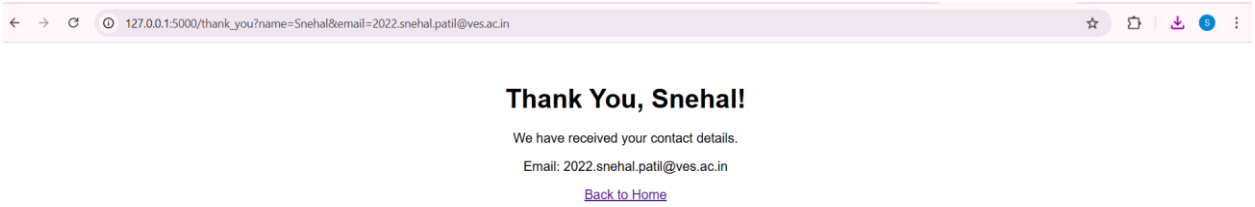
GET REQUEST

POST REQUEST

OUTPUT


127.0.0.1:5000
Welcome Guest!
Go to Contact Form


127.0.0.1:5000/contact
Contact Us
Name: Snehal
Email: 2022.snehal.patil@ves.ac.in
Submit


127.0.0.1:5000/thank_you?name=Snehal&email=2022.snehal.patil@ves.ac.in
Thank You, Snehal!
We have received your contact details.
Email: 2022.snehal.patil@ves.ac.in
Back to Home

A. List some of the core features of Flask

Flask is a micro-framework for web development with several powerful features:

1. Lightweight & Minimalistic: Flask is simple and does not enforce specific project structures.
2. Built-in Development Server & Debugger: Helps developers test applications in real time.
3. Routing: Allows defining URL rules for navigation between different pages.
4. Jinja2 Template Engine: Supports dynamic HTML rendering with Python logic.
5. Request Handling: Supports multiple HTTP methods (GET, POST, PUT, DELETE, etc.).
6. RESTful Support: Makes it easy to build APIs with JSON responses.
7. Extensibility: Can be integrated with third-party libraries like SQLAlchemy (for databases).

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

- Flask(__name__) initializes a Flask application.
- The __name__ variable represents the current module, helping Flask determine the root path of the application.
- This is essential for locating static files, templates, and other resources.
- Example:

python

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

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

- Flask uses Jinja2, a templating engine, to separate logic from presentation.
- Template Inheritance allows a base template to be extended by child templates, ensuring a consistent layout.
- Example:

```
<!-- base.html -->
<html>
<body>
   <header>My Website</header>
   {% block content %}{% endblock %}
</body>
</html>
```

html

CopyEdit

```
<!-- child.html -->
{% extends "base.html" %}
{% block content %}
   <h1>Welcome to My Page!</h1>
{% endblock %}
```

D. What methods of HTTP are implemented in Flask?

1. GET: Used to retrieve data from the server.
2. POST: Used to send data to the server (e.g., form submissions).
3. PUT: Used to update existing data.
4. DELETE: Removes data from the server.
5. PATCH: Partially updates a resource.

Example in Flask:

python

```
@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        return "Data received!"
    return "Send a POST request to submit data."
```

E. What is the difference between Flask and Django framework?

Feature	Flask	Django
Type	Micro-framework	Full-stack framework
Flexibility	Highly flexible, allows choosing libraries	Comes with built-in features
Learning Curve	Easier to learn	Steeper due to built-in tools
Performance	Faster due to minimalism	Heavier due to additional components
Best Use Case	Small to medium applications, APIs	Large-scale applications, CMS