

EXPERIMENT NO.3 : Flask

Name of Student	<u>Komal Milind Deolekar</u>
Class Roll No	<u>14</u>
D.O.P.	<u>04-02-2025</u>
D.O.S.	<u>11-02-2025</u>
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.

Github Link :

https://github.com/KomalDeolekar0607/Webx_Lab/tree/main/Webx_Lab_Exp_3

Theory:**A. List some of the core features of Flask**

- **Lightweight & Micro-framework** – Flask is minimalistic and does not include built-in tools like ORM or authentication.

- **Built-in Development Server** – Provides debugging support with an interactive debugger
- **Jinja2 Templating Engine** – Allows dynamic content rendering in HTML templates.
- **Routing** – Supports URL routing to map URLs to functions.
- **WSGI Support** – Works with WSGI (Web Server Gateway Interface) for better request handling.
- **RESTful Support** – Provides built-in support for creating RESTful APIs.
- **Extensibility** – Can integrate third-party libraries like Flask-SQLAlchemy and Flask-Login.
- **Session Management** – Supports secure client-side sessions using cookies.

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

- Flask(__name__) initializes a Flask application with the name of the current module.
- It helps Flask locate resources like templates, static files, and configurations.
- It ensures that Flask knows the application's context for proper URL routing and debugging.

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

- **Templates** in Flask use **Jinja2**, which allows embedding Python logic inside HTML files.
- **Template Inheritance** lets developers create a base template (base.html) with common layout elements (e.g., header, footer).
- Other templates (child.html) extend base.html using `{% block content %}...{% endblock %}` to define specific page content.
- This helps in maintaining **clean, reusable, and scalable** HTML structures.

D. What methods of HTTP are implemented in Flask.

- **GET** – Retrieves data from the server.
- **POST** – Sends data to the server (e.g., form submission).
- **PUT** – Updates existing resources.
- **DELETE** – Removes resources.
- **PATCH** – Partially updates resources.
- **HEAD** – Retrieves only headers, not the response body.
- **OPTIONS** – Returns allowed HTTP methods for a given URL.

E. What is difference between Flask and Django framework

Feature	Flask	Django
Routing	Manual, flexible with <code>@app.route()</code>	Automatic, uses <code>urls.py</code> for URL mapping
URL Building	Uses <code>url_for('function_name')</code>	Uses <code>reverse('app_name:view_name')</code>
GET Request	<code>request.args.get('key')</code>	Uses <code>request.GET['key']</code>
POST Request	<code>request.form['key']</code>	Uses <code>request.POST['key']</code>

Flask is more **lightweight** and allows developers to structure applications **freely**.

Django is more **feature-rich**, following a "**batteries-included**" approach.

Flask is ideal for **small to medium** projects, while Django is better for **large-scale** applications.

Routing

In Flask, routing refers to the process of mapping a URL to a specific function in your application. Routes determine how your app responds to different URL paths accessed by users.

Example:

```
@app.route('/')

```

```
def home():

```

```
    return "Welcome to the homepage!"

```

This code maps the / URL to the home() function, so when a user visits the root URL, the function is triggered.

URL building

URL building in Flask allows you to dynamically generate URLs using the `url_for()` function. It's useful when you want to avoid hardcoding URLs and keep your app flexible and maintainable.

Example:

```
@app.route('/about')

```

```
def about():

```

```
return "About Page"
```

```
@app.route('/')
```

```
def home():
```

```
    return f"<a href='{url_for('about')}'>Go to About</a>"
```

`url_for("about")` automatically builds the URL for the about route.

GET REQUEST

A GET request is used to retrieve data from the server. It's the default HTTP method when you open a URL in a browser. Query parameters can be passed in the URL for dynamic responses.

Example:

```
@app.route('/welcome')
```

```
def welcome():
```

```
    name = request.args.get('name', 'Guest')
```

```
    return f"Welcome, {name}!"
```

If the user visits `/welcome?name=Komal`, the page will display "Welcome, Komal!".

POST REQUEST

A POST request is used to submit data to the server, often through a form. It is more secure for sending sensitive data and does not show the information in the URL.

Example:

```
@app.route('/submit', methods=['GET', 'POST'])
```

```
def submit():
```

```
    if request.method == 'POST':
```

```

name = request.form['name']

return f'Thank you, {name}!'

return ""

<form method="post">

    Name: <input type="text" name="name">

    <input type="submit">

</form>

""

```

Here, the form accepts a name and sends it via POST. The server then displays a thank-you message.

Code :

app.py

```

from flask import Flask, render_template, request
app = Flask(__name__)
# @app.route('/', methods=['GET', 'POST'])
@app.route('/', methods=['GET'])
def home():
    # name = request.form.get('name', 'Guest') # Get name from form input
    name = request.args.get('name', 'Guest') # Get name from form input

    return f"""
    <html>
    <head>
        <title>Flask App</title>
        <style>
            body {{
                font-family: Arial, sans-serif;
                text-align: center;
                margin: 50px;
                background-color: #f4f4f4;
            }}
            h1 {{
                color: #333;
            }}
            form {{
                margin-top: 20px;
            }}
            input, button {{

```

```

        padding: 10px;
        font-size: 16px;
        margin: 5px;
    }}
    button {{
        background-color: #007BFF;
        color: white;
        border: none;
        cursor: pointer;
    }}
    button:hover {{
        background-color: #0056b3;
    }}
</style>
</head>
<body>
    <h1>Welcome, {name}!</h1>
    <form method="get">
        <input type="text" name="name" placeholder="Enter your name" required>
        <button type="submit">Submit</button>
    </form>
    <p><a href="/contact">Go to Contact Form</a></p>
</body>
</html>
'''

@app.route('/contact', methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        name = request.form.get('name')
        email = request.form.get('email')
        return f'''
<html>
    <head>
        <title>Thank You</title>
        <style>
            body {{ font-family: Arial, sans-serif; text-align: center; margin: 50px; }}
            h1 {{ color: #28a745; }}
            a {{ text-decoration: none; color: #007BFF; }}
            a:hover {{ color: #0056b3; }}
        </style>
    </head>
    <body>
        <h1>Thank You, {name}!</h1>
        <p>Your email: {email}</p>
        <a href="/">Back to Home</a>
    </body>
</html>
'''

    return '''
<html>
<head>
    <title>Contact Us</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; margin: 50px; background-color: #f4f4f4; }

```

```

        form { margin-top: 20px; }
        input, button { padding: 10px; font-size: 16px; margin: 5px; }
        button { background-color: #28a745; color: white; border: none; cursor: pointer; }
        button:hover { background-color: #218838; }
    </style>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="post">
        <input type="text" name="name" placeholder="Your Name" required><br>
        <input type="email" name="email" placeholder="Your Email" required><br>
        <button type="submit">Submit</button>
    </form>
    <p><a href="/">Back to Home</a></p>
</body>
</html>
"""
if __name__ == '__main__':
    app.run(debug=True)

```

Output :

```

D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>py --version
Python 3.13.2

D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>py -m venv venv

D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>
D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>venv\Scripts\activate

(venv) D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>

```

```

(venv) D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\flask_lab_3>pip install flask
Collecting flask
  Downloading flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from flask)
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting itsdangerous>=2.2 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting blinker>=1.9 (from flask)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting colorama (from click>=8.1.3->flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)

```

127.0.0.1:5000

YouTube Maps WhatsApp My Applied Scheme... Love Food Hate Wa...

Welcome, Guest!

[Go to Contact Form](#)

127.0.0.1:5000

YouTube Maps WhatsApp My Applied Scheme... Love Food Hate Wa...

Welcome, Guest!

[Go to Contact Form](#)

127.0.0.1:5000/?name=Komal+Deolekar

YouTube Maps WhatsApp My Applied Scheme... Love Food Hate Wa...

Welcome, Komal Deolekar!

[Go to Contact Form](#)

127.0.0.1:5000/contact

YouTube Maps WhatsApp My Applied Scheme... Love Food Hate Wa...

Contact Us

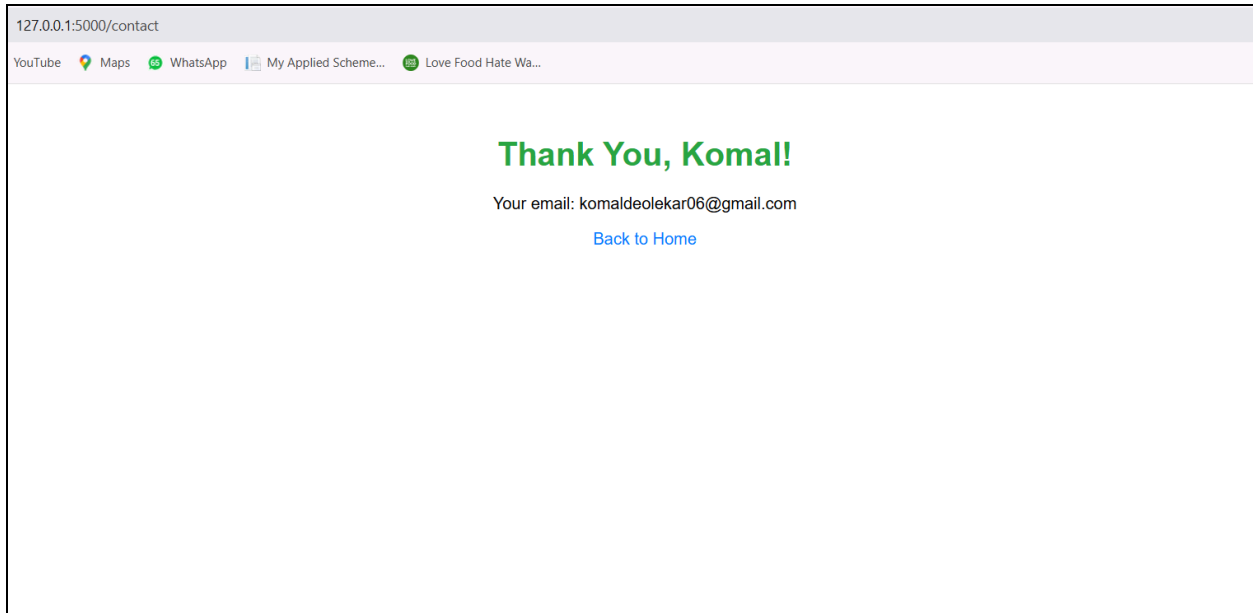
[Back to Home](#)

127.0.0.1:5000/contact

YouTube Maps WhatsApp My Applied Scheme... Love Food Hate Wa...

Contact Us

[Back to Home](#)



Conclusion :

In this Flask application, we successfully implemented multiple routes to handle both GET and POST requests, demonstrating the fundamental workings of a web application.

1. Dynamic GET Request Handling:
 - The homepage (/) dynamically greets users when accessed with a query parameter, e.g., /welcome?name=Komal, enhancing user experience with personalized messages.
2. Form Handling with POST Request:
 - The contact form (/contact) accepts user inputs (name and email) and submits data via the POST method.
 - The submitted details are then displayed on the thank-you page (/thank_you), confirming the correct data transfer.
3. Routing and Page Navigation:
 - We designed separate routes for the homepage, contact form, and thank-you page, ensuring smooth navigation within the application.

This project effectively demonstrates how Flask handles HTTP methods, form submissions, and dynamic query parameters, forming the basis of real-world web applications.