**Experiment – 3: Flask**

|  |  |
| --- | --- |
| **Name of Student** | **AKRUTI DABAS** |
| **Class Roll No** | **D15A / 11** |
| **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.

* Create routes for the homepage (/), contact form (/contact), and thank-you page (/thank\_you).

1. A contact page (/contact) where users can fill out a form with their name and email.
2. Handle the form submission using the POST method and display the submitted data on a thank-you page (/thank\_you).

* On the contact page, create a form to accept user details (name and email).
* Use the POST method to handle form submission and pass data to the thank-you page

1. 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>.

* On the homepage (/), use a query parameter (name) to display a personalized welcome message.

**Theory:**

1. **List some of the core features of Flask**

Flask is a lightweight and flexible web framework for Python, designed to be simple yet powerful. Some of its core features include:

1. Lightweight and Minimalistic – Flask provides only the essential tools for web development and lets developers choose additional libraries as needed.
2. Built-in Development Server & Debugger – Comes with a built-in server for testing applications and a debugger to track errors.
3. Jinja2 Templating Engine – Supports Jinja2 for dynamic HTML generation and template inheritance.
4. URL Routing – Allows easy mapping of URLs to specific functions using decorators (@app.route).
5. WSGI Support – Based on Werkzeug, which provides WSGI (Web Server Gateway Interface) support.
6. RESTful Request Handling – Supports GET, POST, PUT, DELETE, and other HTTP methods for building RESTful APIs.
7. Session and Cookie Management – Supports secure cookies for session management.
8. Extension Support – Can be extended using Flask extensions like Flask-SQLAlchemy (database support), Flask-WTF (forms), and Flask-Login (authentication).
9. **Why do we use Flask(\_\_name\_\_) in Flask?**

The Flask(\_\_name\_\_) statement is used to create a Flask application instance. The \_\_name\_\_ argument is important because:

1. Identifies the Application Module – \_\_name\_\_ refers to the name of the current module (file), which helps Flask locate resources such as templates and static files.
2. Helps in Debugging – It allows Flask to understand where the application is running and enables better debugging.
3. Essential for Routing – Since Flask applications rely on decorators like @app.route(), the app instance needs to be created first.
4. Allows Flask to Locate Files – Flask uses the \_\_name\_\_ variable to determine the root path of the application, making it easier to load templates, static files, and configuration settings.

Example:

from flask import Flask

app = Flask(\_\_name\_\_) # Creates a Flask application

@app.route('/')

def home():

return "Hello, Flask!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **What is Template (Template Inheritance) in Flask?**

In Flask, templates allow dynamic content rendering using the Jinja2 templating engine. Template Inheritance is a feature where a base template provides a common structure, and child templates extend it while modifying only specific sections.

Why Use Template Inheritance?

* Avoids code duplication by defining a base template for common elements like headers, footers, and navigation bars.
* Child templates only need to modify or add content to specific blocks, keeping the code clean and modular.

Example of Template Inheritance:

Base Template (base.html)

<!DOCTYPE html>

<html>

<head>

<title>{% block title %}Default Title{% endblock %}</title>

</head>

<body>

<header>My Website Header</header>

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

<footer>My Website Footer</footer>

</body>

</html>

Child Template (home.html)

html

CopyEdit

{% extends "base.html" %}

{% block title %}Home Page{% endblock %}

{% block content %}

<h1>Welcome to My Website</h1>

<p>This is the homepage.</p>

{% endblock %}

Here, home.html extends base.html, inheriting its structure but modifying the title and content blocks.

1. **What methods of HTTP are implemented in Flask.**

Flask supports multiple HTTP methods to handle different types of requests in web applications. Some of the common methods include:

1. GET – Used to request data from a server. (Default method if not specified)

@app.route('/home', methods=['GET'])

def home():

return "Welcome to Flask!"

1. POST – Used to send data to the server (e.g., submitting forms).

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

def submit():

return "Form Submitted!"

1. PUT – Used to update existing data on the server.

@app.route('/update', methods=['PUT'])

def update():

return "Data Updated!"

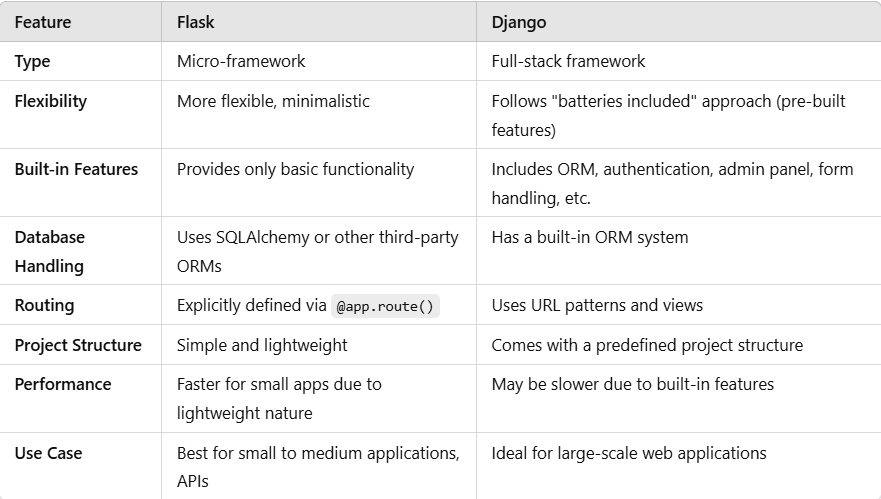
1. DELETE – Used to delete data from the server.

@app.route('/delete', methods=['DELETE'])

def delete():

return "Data Deleted!"

1. PATCH – Used to partially update data on the server.
2. HEAD – Similar to GET but only retrieves the headers and not the body.
3. OPTIONS – Used to check the HTTP methods supported by a server.
4. **What is difference between Flask and Django framework.**

****

**Output:**

* **App.py**

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

    name = request.args.get('name', 'Guest')  # Get name from query parameter

    return render\_template('home.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 submitting the form!"

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

* **home.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Home</title>

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    <style>

        body {

            background: linear-gradient(to right,rgb(0, 0, 0),rgb(3, 57, 54));

            height: 100vh;

            display: flex;

            align-items: center;

            justify-content: center;

        }

        .container {

            background: white;

            padding: 30px;

            border-radius: 15px;

            box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

            text-align: center;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1 class="text-info">Welcome, {{ name }}!</h1>

        <p class="lead">This is the homepage of our beautiful Flask application.</p>

        <a href="/contact" class="btn btn-outline-info">Go to Contact Form</a>

    </div>

</body>

</html>

* **contact.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Contact</title>

 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    <style>

        body {

            background: linear-gradient(to right, rgb(0, 0, 0),rgb(3, 57, 54));

            height: 100vh;

            display: flex;

            align-items: center;

            justify-content: center;

        }

        .form-container {

            background: white;

            padding: 40px;

            border-radius: 15px;

            box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

            width: 40%;

        }

    </style>

</head>

<body>

    <div class="form-container">

        <h2 class="text-center text-info">Contact Us</h2>

        <form method="POST">

            <div class="mb-3">

                <label class="form-label">Name:</label>

                <input type="text" name="name" class="form-control" required>

            </div>

            <div class="mb-3">

                <label class="form-label">Email:</label>

                <input type="email" name="email" class="form-control" required>

            </div>

            <button type="submit" class="btn btn-info w-100">Submit</button>

        </form>

    </div>

</body>

</html>

* **thank\_you.html**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Thank You</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    <style>

        body {

            background: linear-gradient(to right,rgb(0, 0, 0),rgb(3, 57, 54));

            height: 100vh;

            display: flex;

            align-items: center;

            justify-content: center;

        }

        .thank-you-box {

            background: white;

            padding: 30px;

            border-radius: 15px;

            box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

            text-align: center;

        }

    </style>

</head>

<body>

    <div class="thank-you-box">

        <h2 class="text-success">Thank You, {{ name }}!</h2>

        <p class="lead">We have received your email: <strong>{{ email }}</strong></p>

        <a href="/" class="btn btn-outline-success">Go Back Home</a>

    </div>

</body>

</html>

