Experiment 4 : Flask Application using GET and POST

Name of Student	**Aryan Patankar**
Class Roll No	**D15A_33**
D.O.P.	**27/02/2025**
D.O.S.	**06/03/2025**
Sign and Grade	

AIM : To design a Flask application that showcases URL building and demonstrates the use of HTTP methods (`GET` and `POST`) for handling user input and processing data.

PROBLEM STATEMENT :

Create a Flask application with the following requirements:

1. **A homepage (`/`) with links to a "Profile" page and a "Submit" page using the `url_for()` function.**
2. **The "Profile" page (`/profile/<username>`) dynamically displays a user's name passed in the URL.**
3. **A "Submit" page (`/submit`) displays a form to collect the user's name and age. The form uses the POST method to send the data, and the server displays a confirmation message with the input.**

Theory:

1. What is a route in Flask, and how is it defined?

In Flask, a **route** is a URL pattern that is associated with a specific function in a web application. When a user visits a particular URL, Flask executes the corresponding function and returns the response. Routes define how the application should respond to different URLs.
A route is defined using the `@app.route()` decorator. For
example: python
CopyEdit

```
from flask import Flask

app = Flask(_name_)
```

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

In this example, the root URL (`/`) is mapped to the `home` function, which returns a simple text response.

2. How can you pass parameters in a URL route?

In Flask, parameters can be passed in a URL route using angle brackets (`< >`). These parameters can be dynamic and help in passing values from the URL to the function.

For example:

```
@app.route('/user/<name>')
def greet_user(name):
    return f"Hello, {name}!"
```

Here, when a user visits `/user/Sanket`, the function receives `"Sanket"` as the `name` parameter and returns `"Hello, Sanket!"`.
Flask also allows specifying the data type of parameters, such as:

```
@app.route('/square/<int:num>') def square(num):
    return f"The square of {num} is {num**2}"
```

This ensures that `num` is treated as an integer.

3. What happens if two routes in a Flask application have the same URL pattern?

If two routes have the same URL pattern in a Flask application, only the last defined route will take effect, and the previous one will be overridden. Flask does not allow duplicate routes with the same URL pattern, as it would cause ambiguity.

Example of conflicting routes:

```
@app.route('/hello')
def hello1():
    return "Hello from function 1"

@app.route('/hello')
def hello2():
    return "Hello from function 2"
```

In this case, when a user visits `/hello`, Flask will only execute `hello2()`, and `hello1()` will be ignored.

4. What are the commonly used HTTP methods in web applications?

The most commonly used HTTP methods in web applications are:

1. **GET** – Requests data from the server (e.g., retrieving a webpage).
2. **POST** – Sends data to the server (e.g., submitting a form).
3. **PUT** – Updates existing data on the server.
4. **DELETE** – Removes a resource from the server.
5. **PATCH** – Partially updates a resource.

In Flask, these methods can be specified in the `methods` parameter of the `@app.route()` decorator:

```
@app.route('/submit', methods=['POST'])
def submit():
    return "Form submitted!"
```

5. What is a dynamic route in Flask?

A **dynamic route** in Flask is a route that contains variables, allowing it to handle multiple different URLs with a single function. Dynamic routes make the web application more flexible by enabling the use of parameters within the URL.

Example:

```
@app.route('/user/<username>')
def profile(username):
    return f"User Profile: {username}"
```

If a user visits `/user/Sanket`, the function receives `"Sanket"` as a parameter and responds accordingly.

6. Write an example of a dynamic route that accepts a username as a parameter.

```
from flask import Flask

app = Flask(__name__)

@app.route('/user/<username>')
def show_user(username):
    return f"Welcome, {username}!"

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

In this example, the route `/user/<username>` accepts a `username` parameter from the URL and returns a personalized welcome message.

7. What is the purpose of enabling debug mode in Flask?

Enabling **debug mode** in Flask is useful for development because it provides:

1. **Automatic Code Reloading** – The server automatically restarts when changes are made to the code.
2. **Detailed Error Messages** – Flask displays an interactive debugger when an error occurs, making it easier to identify and fix issues.

However, debug mode should **not** be enabled in a production environment due to security risks.

8. How do you enable debug mode in a Flask application?

Debug mode can be enabled in two ways:

1. **Setting `debug=True` in `app.run()`**

python
CopyEdi
t

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

2. **Using Environment Variables**
   In the terminal, set the environment variable before running the Flask app:

```
export
FLASK_ENV=development flask
run
```

On Windows (Command Prompt):

```
set
FLASK_ENV=development
flask run
```

This enables debug mode and allows for easier debugging during development.

OUTPUT:-

app.py

```
from flask import Flask, render_template, request, url_for, redirect

app = Flask(__name__)

# 1. Homepage Route (/)
@app.route('/')
def home():
    return render_template('home.html')

# 2. Profile Page Route (/profile/<username>)
@app.route('/profile/<username>')
def profile(username):
    return render_template('profile.html', username=username)

# 3. Submit Page Route (/submit)
@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        name = request.form['name']
        age = request.form['age']
        return render_template('confirmation.html', name=name, age=age)
    return render_template('submit.html')

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

Templates

1.home.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Homepage</title>
</head>
<body>
    <h1>Welcome to Our Site!</h1>
    <p>
        <a href="{{ url_for('profile', username='JohnDoe') }}">View JohnDoe's Profile</a>
    </p>
    <p>
        <a href="{{ url_for('submit') }}">Submit Your Information</a>
    </p>
</body>
</html>
```

2. profile.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Profile</title>
</head>
<body>
   <h1>User Profile</h1>
   <p>Username: {{ username }}</p>
</body>
</html>
```

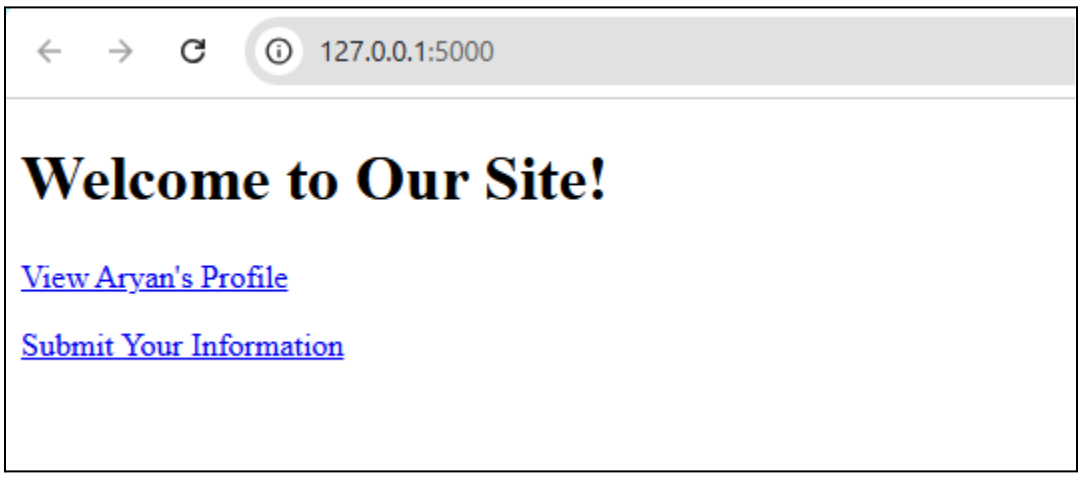
3. submit.html

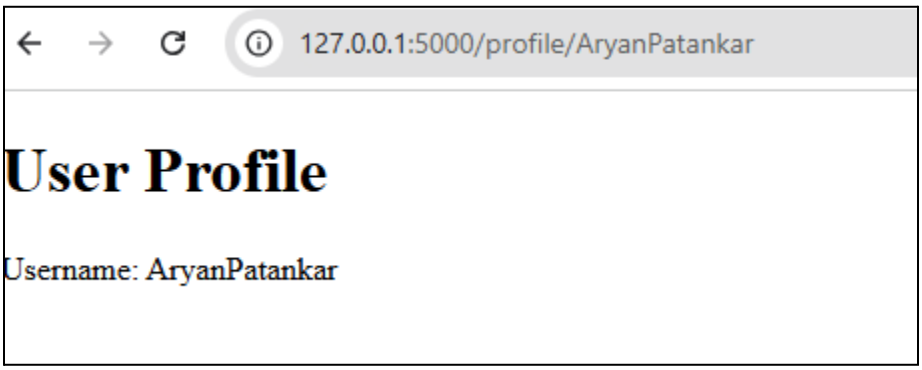
```
<!DOCTYPE html>
<html>
<head>
   <title>Submit Information</title>
</head>
<body>
   <h1>Submit Your Information</h1>
   <form method="POST" action="{{ url_for('submit') }}">
      <label for="name">Name:</label><br>
      <input type="text" id="name" name="name" required><br><br>
      <label for="age">Age:</label><br>
      <input type="number" id="age" name="age" required><br><br>
      <input type="submit" value="Submit">
   </form>
</body>
</html>
```

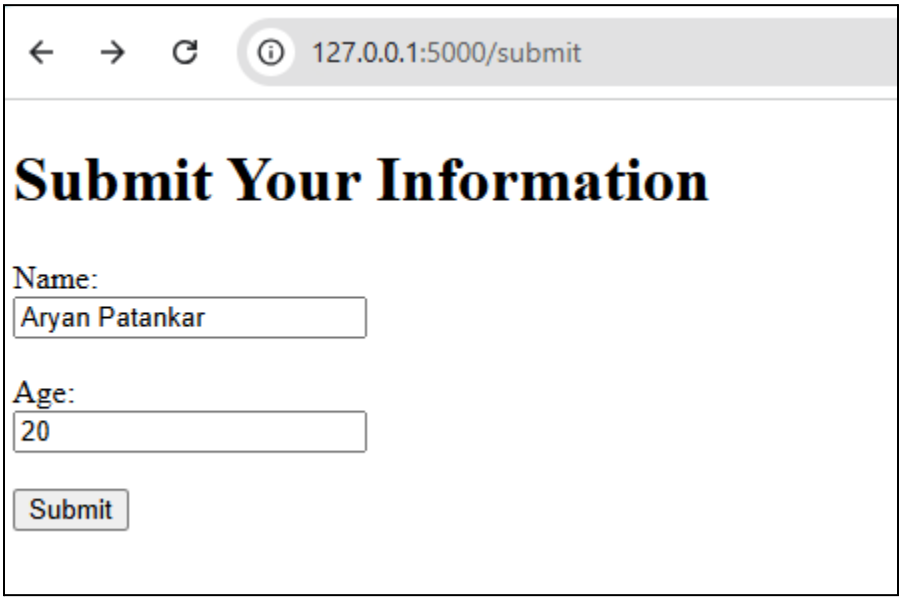
4. confirmation.html

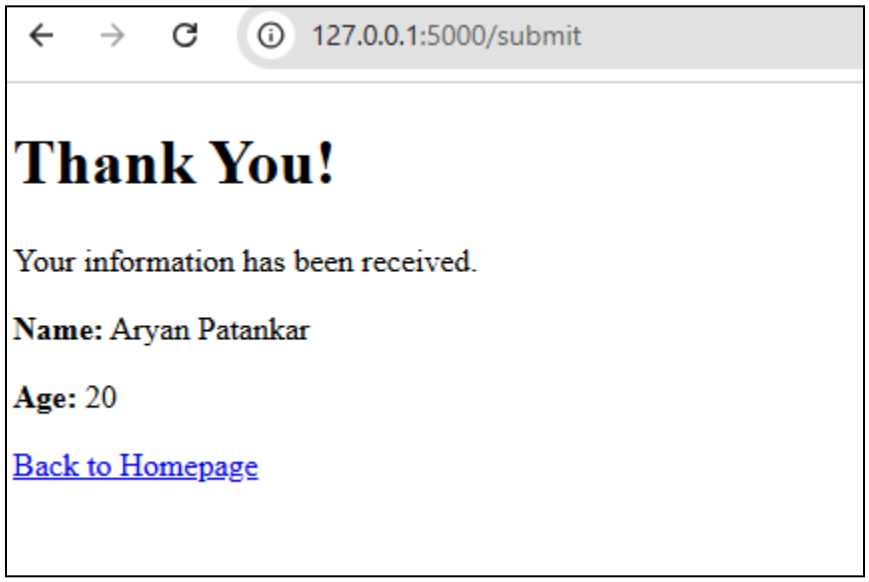
```
<!DOCTYPE html>
<html>
<head>
   <title>Confirmation</title>
</head>
<body>
   <h1>Thank You!</h1>
   <p>Your information has been received.</p>
   <p>Name: {{ name }}</p>
   <p>Age: {{ age }}</p>
</body>
</html>
```

Result:-


127.0.0.1:5000
Welcome to Our Site!
View Aryan's Profile
Submit Your Information


127.0.0.1:5000/profile/AryanPatankar
User Profile
Username: AryanPatankar


127.0.0.1:5000/submit
Submit Your Information
Name:
Aryan Patankar
Age:
20
Submit




```
PS C:\Users\aryan\WebX\Exp4> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 507-892-756
127.0.0.1 - - [02/Apr/2025 19:26:28] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Apr/2025 19:26:35] "GET / HTTP/1.1" 200 -
```