EXPERIMENT NO. 4

Name of Student	MANAV PUNJABI
Class Roll No	**D15A 44**
D.O.P.	
D.O.S.	
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, known as a view function, which handles requests made to that URL.
    - Routes are defined using the `@app.route()` decorator, allowing developers to map URLs to functions easily

2. How can you pass parameters in a URL route?
    - Passing Parameters in URL Routes
    - Parameters can be passed in a URL route by including them within angle brackets in the route definition.
    - For example, a route can be defined as `@app.route('/user/<username>')`, where `username` is a variable that will capture the part of the URL that corresponds to it

3. What happens if two routes in a Flask application have the same URL pattern?
    - Handling Duplicate Routes
    - If two routes in a Flask application have the same URL pattern, Flask will raise an error.
    - This is because each route must be unique; having duplicate routes can lead to ambiguity regarding which function should handle the request[4]

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

    Commonly Used HTTP Methods

    The most commonly used HTTP methods in web applications include:

    - GET: Retrieve data from the server.
    - POST: Submit data to be processed.
    - PUT: Update existing data.
    - DELETE: Remove data from the server.
    - PATCH: Partially update existing data

5. What is a dynamic route in Flask?
    - Dynamic Routes
    - A dynamic route in Flask allows you to define routes that can accept variable parts in the URL. This enables more flexible and parameterized routing, making it possible to create URLs that change based on user input or other variables[6].

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

    Example of a Dynamic Route

    Here's 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_profile(username):
    return f'User: {username}' if
    __name__ == '__main__
    ': app.run(debug=True)
    ```

    In this example, accessing `/user/john` would return `User: john`

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

- Purpose of Enabling Debug Mode
- Enabling debug mode in Flask allows for automatic code reloading and provides detailed error messages. This is particularly useful during development, as it
  eliminates the need to manually restart the server after making changes to the code and helps identify issues quickly

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

- Enabling Debug Mode
- To enable debug mode in a Flask application, you can set `debug=True` when running the app. This can be done as follows:

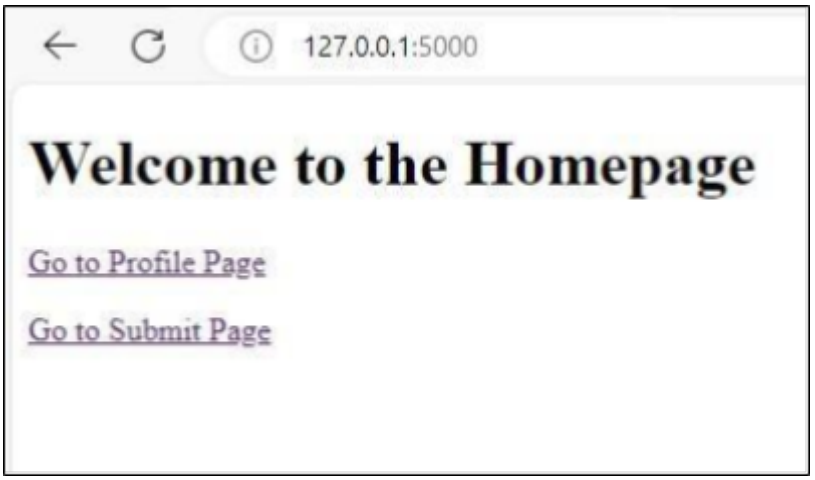
```
if __name__ == '__main__':
    app.run(debug=True)
```

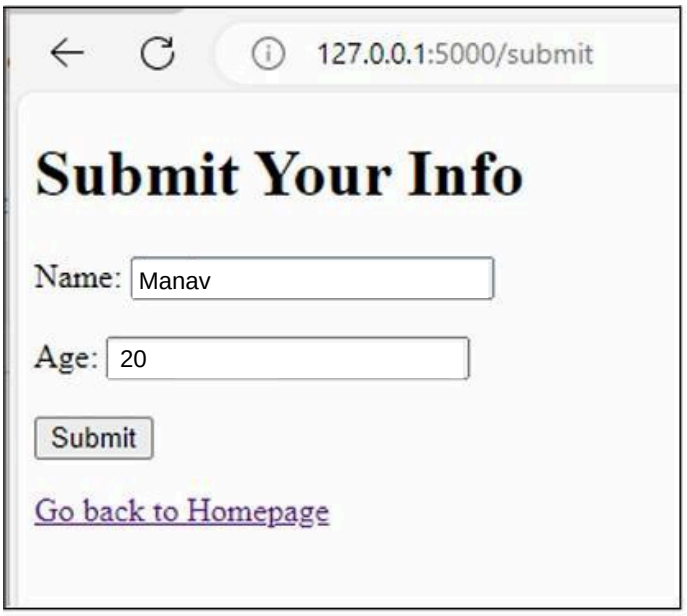
Code:

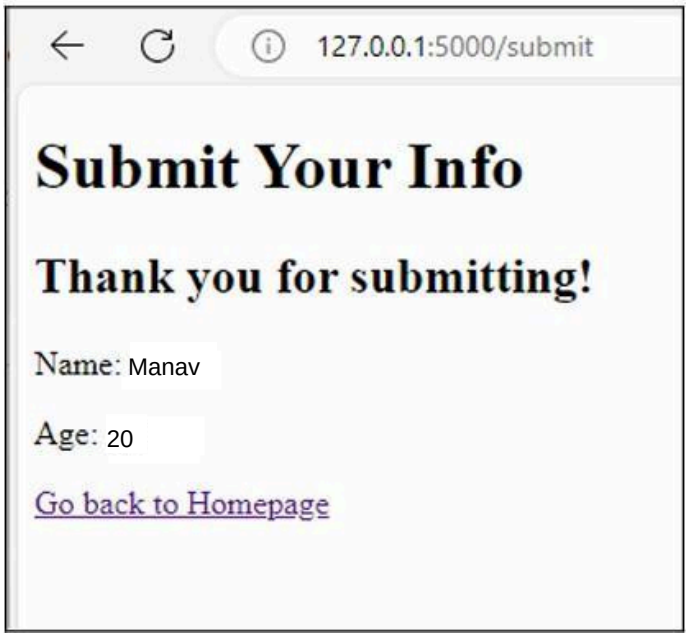
```
from flask import Flask, render_template, request, redirect, url_for


app = Flask(_name_)


@app.route('/') def

home():

   return '''<h1>Welcome to the Homepage</h1>

          <p><a href="/submit">Go to Submit Page</a></p>

          <p><a href="/profile/JohnDoe">Go to Profile Page</a></p>'''


@app.route('/profile/<username>') def

profile(username):

   return f'<h1>Profile Page</h1><p>Welcome, {username}!</p>'


@app.route('/submit', methods=['GET', 'POST']) def submit():     if request.method

== 'POST':         name = request.form['name']         age = request.form['age']
```

```
return f'<h1>Form Submitted</h1><p>Name: {name}</p><p>Age: {age}</p>'

return '''<form method="post">
                Name: <input type="text" name="name"><br>
                Age: <input type="text" name="age"><br>
                <input type="submit" value="Submit">
            </form>'''

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

OUTPUT:







Conclusion:

In this experiment, we successfully designed a Flask application that demonstrated URL building, dynamic routing, and the use of GET and POST methods. We used url_for() to generate dynamic links, created a profile route that accepts username as a URL parameter, and handled form submission through POST method on the submit page. This helped us understand how Flask efficiently processes user input and manages navigation between routes.