EXPERIMENT NO. 4

Name of Student	Mahvish Siddiqui
Class Roll No	D15A 56
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 mapped to a function (also called a **view function**). When a user accesses a specific URL on the web server, Flask uses routes to determine which function should handle the request.

A route is defined by using the @app.route() decorator followed by the URL pattern and the HTTP methods that it should respond to. Here is an example of defining a simple route in Flask:

```
from flask import Flask

app = Flask(__name__)



@app.route('/')

def home():

    return 'Hello, World!'
```

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

In this example, the route '/' is defined for the function home(). When a user accesses the root URL, the function will return the message "Hello, World!".

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

You can pass parameters in a URL route by using **variable parts** in the URL pattern. These parameters are specified using <parameter_name> in the route definition. Flask automatically captures values in the URL and passes them as arguments to the view function.

For example:

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

In this case, when a user accesses a URL like /greet/John, the value "John" is passed to the name parameter in the greet function, and the response will be "Hello, John!".

You can also specify a **type** for the parameter:

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    return f'Post ID is {post_id}'
```

Here, the post_id parameter is restricted to an integer, and Flask will return a 404 error if the URL doesn’t match this format.

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

If two routes in a Flask application have the same URL pattern, Flask will raise an error, specifically an **AssertionError**, because Flask cannot differentiate between the two routes. A route pattern must be unique in the app.

For example:

```
@app.route('/example')
def route_one():
```

```
    return 'First Route'


@app.route('/example')

def route_two():

    return 'Second Route'
```

In this case, Flask will raise an error because both routes are mapped to the same URL '/example'.

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

The most commonly used HTTP methods in web applications are:

- **GET**: Used to request data from the server. It is the most common HTTP method used in browsing and accessing web pages.
- **POST**: Used to submit data to the server, often in the form of a form submission. It is commonly used for creating new resources or actions.
- **PUT**: Used to update existing resources on the server.
- **DELETE**: Used to remove a resource from the server.
- **PATCH**: Used to apply partial updates to a resource.
- **HEAD**: Similar to GET, but does not return a message body. It is used to retrieve metadata.
- **OPTIONS**: Used to retrieve the allowed HTTP methods for a particular resource.

In Flask, you can specify which HTTP methods a route should handle using the methods argument:

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

def submit_form():

    return 'Form submitted successfully!'
```

5. What is a dynamic route in Flask?

A **dynamic route** in Flask is a route where part of the URL is variable and can accept different values. This is achieved by using **dynamic URL segments** that are enclosed in angle brackets (< >), which act as placeholders for values that will be passed into the view function as parameters.

For example:

```
@app.route('/item/<item_id>')

def get_item(item_id):

    return f'Item ID: {item_id}'
```

Here, <item_id> is a dynamic part of the URL, and the value that the user provides in the URL will be passed as the item_id parameter to the get_item() function.

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

Here’s an example of a dynamic route that accepts a **username** as a parameter:

```
@app.route('/user/<username>')

def show_user_profile(username):

    return f'User Profile for {username}'
```

If a user accesses the URL /user/john, the response will be "User Profile for john". Flask automatically captures the value of the <username> parameter from the URL.

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

Enabling **debug mode** in Flask provides a set of helpful features for development, such as:

- **Automatic reloading**: The server will automatically reload whenever you make changes to your code, so you don’t have to manually restart the server.
- **Detailed error messages**: If an error occurs, Flask will show a detailed traceback in the browser, making it easier to diagnose and fix issues.
- **Better logging**: Debug mode enables verbose logging to provide more insight into requests and responses.

This feature should only be enabled during development, as it can expose sensitive information in production environments.

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

You can enable debug mode in Flask by setting the debug parameter to True in the app.run() method or by setting the FLASK_ENV environment variable to development. Here's how you can enable it:

Method 1: Using app.run(debug=True)

```
if __name__ == '__main__':

    app.run(debug=True)
```

Method 2: Using the FLASK_ENV environment variable

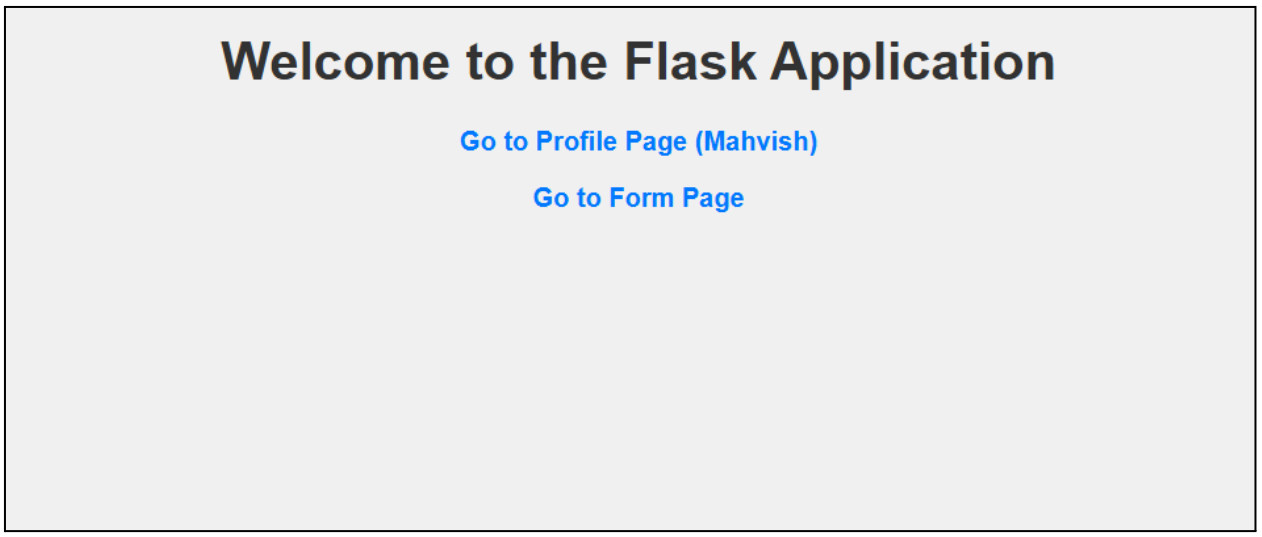
Set the FLASK_ENV environment variable to development before running the app:

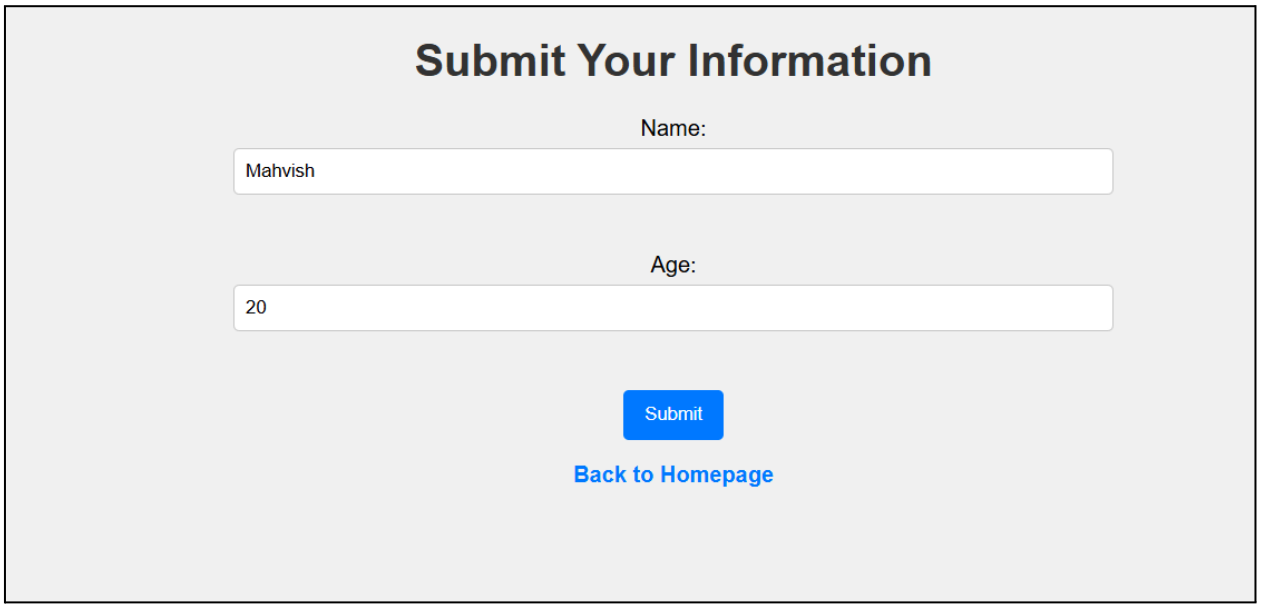
```
export FLASK_ENV=development

flask run
```

In this case, Flask will automatically enable debug mode when the app is running in the development environment.

OUTPUT


Welcome to the Flask Application
Go to Profile Page (Mahvish)
Go to Form Page


Submit Your Information
Name:
Mahvish
Age:
20
Submit
Back to Homepage


Confirmation
Thank you for your submission!
Your name is Mahvish and your age is 20.
Back to Homepage

CONCLUSION

In this experiment, we successfully designed and implemented a Flask application that demonstrates URL building and the handling of HTTP methods (GET and POST).

1. **Dynamic URL Handling:** The application uses the url_for() function to navigate between pages efficiently.

2. **GET Request Usage:** The "Profile" page dynamically extracts the username from the URL and displays it.

3. **POST Request Implementation:** The "Submit" page collects user input via an HTML form and sends it to the server, where it is processed and displayed on a confirmation page.

This project highlights key Flask concepts, such as routing, rendering templates, and managing user input using HTTP methods. By implementing these techniques, we have laid the foundation for developing more interactive and data-driven web applications.