EXPERIMENT NO. 5

Name of Student	Mahvish Siddiqui
Class Roll No	D15A 56
D.O.P.	
D.O.S.	
Sign and Grade	

AIM : To create a Flask application that demonstrates template rendering by dynamically generating HTML content using the `render_template()` function.

PROBLEM STATEMENT :

Develop a Flask application that includes:

1. A homepage route (`/`) displaying a welcome message with links to additional pages.
2. A dynamic route (`/user/<username>`) that renders an HTML template with a personalized greeting.
3. Use Jinja2 templating features, such as variables and control structures, to enhance the templates.

Theory:

1. What does the render_template() function do in a Flask application?

In a Flask application, the render_template() function is used to render HTML templates. It takes an HTML file (usually located in the templates folder) and returns it as a response to the client's browser. This allows you to dynamically generate content in your HTML files by passing data from the Flask application to the templates.

How it works:

- Dynamic HTML Rendering: render_template() renders HTML templates with dynamic content. You can pass variables to the template, and it will generate the final HTML by replacing placeholders (template expressions) with actual values.
- Template Files: The templates are usually HTML files with special syntax that allows you to insert dynamic data (like variables or control flow).

Syntax:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
```

```
    username = 'John Doe'
    return render_template('index.html', user=username)

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

In this example:

- The render_template('index.html', user=username) function tells Flask to look for an index.html file in the templates directory and pass the value of username to the template as a variable user.

In the template file (index.html), you can access the passed user variable as follows:

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Hello, {{ user }}!</h1>
</body>
</html>
```

The result in the browser would be:

Hello, John Doe!

2. What is the significance of the templates folder in a Flask project?

In a Flask project, the templates folder holds HTML files (or other template files) that Flask uses to render dynamic content. This is the default directory Flask looks for when calling render_template(). The templates folder is significant because it keeps the structure of the application organized by separating static content (like images, CSS, JavaScript) from dynamic content (like HTML templates).

Key Points:

- Location: The templates folder is typically located in the root of the Flask project. Flask automatically searches this folder for templates.
- Organization: Keeping your HTML templates in a dedicated folder (usually templates) is a convention in Flask. This promotes better organization and structure in your project, especially as the number of templates grows.

Example Project Structure:

```
my_flask_app/
│
├── app.py (Flask application file)
├── templates/
│   ├── index.html
```

```
│   └── about.html
├── static/
│   ├── style.css
│   └── logo.png
└── requirements.txt
```

Flask Behavior:

- When you call render_template(), Flask will search the templates folder for the template file you specify. If it's not found, Flask will raise a TemplateNotFound exception.

3. What is Jinja2, and how does it integrate with Flask?

Jinja2 is a powerful and flexible templating engine for Python. It is the default template engine used by Flask. Jinja2 allows you to write dynamic HTML pages by embedding Python-like expressions and control structures in HTML files. With Jinja2, you can insert variables, perform loops, and use conditionals directly inside your HTML templates.

Key Features of Jinja2:

- Variables: You can insert variables into your templates using the {{ variable_name }} syntax.
- Control Structures: Jinja2 supports control flow elements like loops and conditionals. You can use {% if %}, {% for %}, and other similar constructs.
- Filters: Jinja2 provides filters to format and manipulate variables before rendering, such as {{ name|capitalize }} to capitalize the first letter of the name.
- Macros: Jinja2 supports macros, which are reusable pieces of templates (like functions) that can be used to avoid repetition.
- Template Inheritance: Jinja2 allows you to create a base template that other templates can inherit, reducing code duplication.

How Jinja2 integrates with Flask:

- Automatic Integration: Flask uses Jinja2 internally to process templates. When you call render_template(), Flask automatically compiles the template using Jinja2 before rendering it to the client.
- Template Syntax: In the template files (usually .html files), you use Jinja2 syntax to embed Python-like expressions, loops, and conditionals inside HTML. These expressions are evaluated and replaced by the corresponding values at runtime.

Example of Jinja2 in a Flask Template:

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ page_title }}</title>
</head>
<body>
    <h1>Welcome, {{ username }}!</h1>

    {% if logged_in %}
        <p>You are logged in.</p>
```

```
    {% else %}
        <p>Please log in.</p>
    {% endif %}

    <ul>
    {% for item in items %}
        <li>{{ item }}</li>
    {% endfor %}
    </ul>
</body>
</html>
```

CODE

app.py

```
from flask import Flask, render_template


app = Flask(__name__)


@app.route('/')
def home():
    return render_template('home.html')


@app.route('/user/<username>')
def user_profile(username):
    return render_template('user.html', username=username)


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

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Flask App{% endblock %}</title>
```

```
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

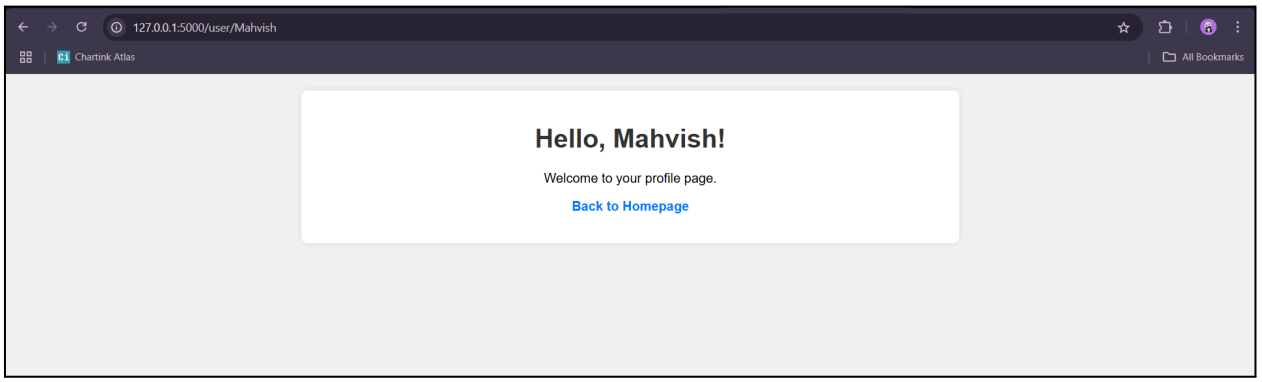
user.html

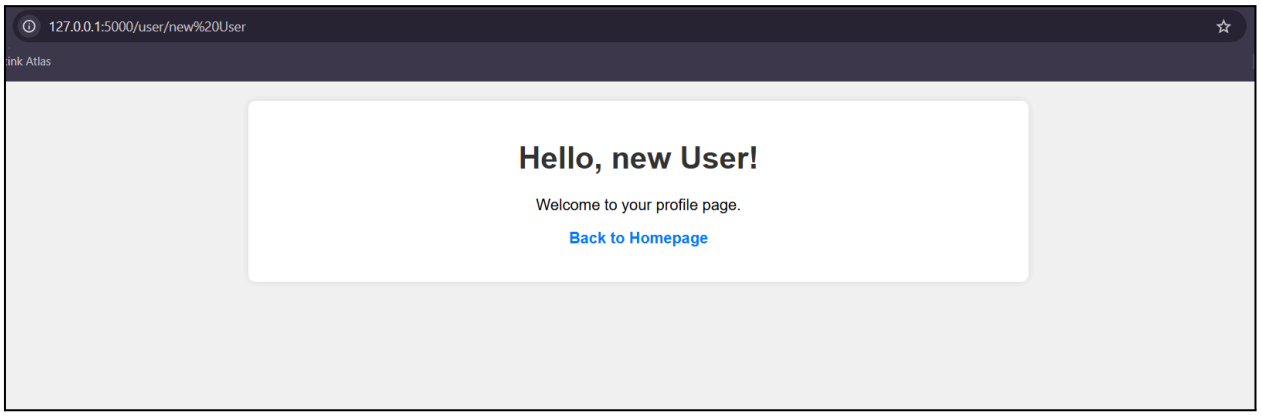
```
{% extends 'base.html' %}

{% block title %}User Profile{% endblock %}

{% block content %}
    <h1>Hello, {{ username }}!</h1>
    <p>Welcome to your profile page.</p>
    <p><a href="{{ url_for('home') }}">Back to Homepage</a></p>
{% endblock %}
```

OUTPUT





CONCLUSION

In this experiment, we successfully developed a Flask application that demonstrates template rendering using the render_template() function. We implemented dynamic routing to personalize user content, utilized Jinja2 templating for efficient HTML generation, and applied CSS for a structured and visually appealing layout. This project highlights the power of Flask in creating dynamic and reusable web applications.