Experiment 5 : Flask Application using render_template() function.

Name of the Student	**Ria Chaudhari**
Class and Roll No	**D15A07**
DOP	
DOS	
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?

The `render_template()` function in Flask is used to render HTML templates by dynamically inserting data and returning the final HTML content as a response to the client.

It allows Flask applications to separate logic from presentation by placing HTML templates in a dedicated `templates` folder.

Example:

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('index.html', title="Welcome Page")
```

- In this example, render_template('index.html', title="Welcome Page") loads index.html and passes the title variable to it.

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

- Flask looks for HTML templates in a default directory named `templates`.
- It is a convention in Flask to store all HTML files inside this folder so that `render_template()` can locate them automatically.
- It helps maintain a clean project structure by separating application logic (Python code) from the presentation layer (HTML files).

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

Jinja2 is a powerful templating engine used by Flask to dynamically generate HTML pages.
It allows embedding Python-like expressions inside HTML using curly braces (`{{ }}` for variables, `{% %}` for logic like loops and conditionals).
Integration with Flask: Flask uses Jinja2 by default to process templates when `render_template()` is called.
Example Usage in an HTML file:

```
<!DOCTYPE html>

<html>

<head>

    <title>{{ title }}</title>

</head>

<body>

    <h1>Hello, {{ username }}!</h1>

    {% if username == "Admin" %}

        <p>Welcome, Admin! You have full access.</p>

    {% else %}

        <p>Welcome, {{ username }}! You have limited access.</p>

    {% endif %}

</body>

</html>
```

Here,

- {{ title }} inserts a variable value.
- {% if %} and {% else %} demonstrate control structures for conditional rendering.

Code:

app.py

app.py > home C:\Users\Ria Chaudhari\webx\flask_contact_app\app.py • 1 problem in this file

```python
from flask import Flask, render_template

app = Flask(__name__)

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

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

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

index.html

templates > index.html > html > body > p > a

```html
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Flask App!</h1>
    <p>Visit <a href="/user/Ria">your profile</a>.</p>
</body>
</html>

```

user.html

templates > user.html > html > body > p

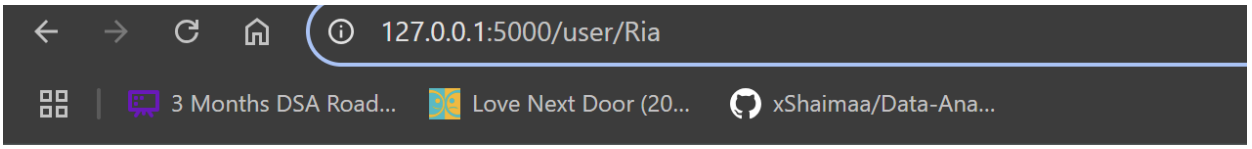
```
<!DOCTYPE html>
<html>
<head>
    <title>User Page</title>
</head>
<body>
    <h1>Hello, {{ username }}!</h1>
    <p>Welcome to your profile page.Hope you have a great time here!</p>
    <a href="/">Go back home</a>
</body>
</html>

```

Output:

Welcome to the Flask App!

Visit your profile.



Hello, Ria!

Welcome to your profile page.Hope you have a great time here!

Go back home

Conclusion:

In this experiment, I successfully developed a Flask application demonstrating template rendering using the render_template() function. By creating dynamic routes and using Jinja2 templates, I displayed personalized user greetings based on URL parameters. The separation of business logic and presentation using HTML templates ensured clean and maintainable code. Additionally, I applied CSS for styling, enhancing the visual appeal of the application. This experiment helped me understand how to build interactive and user-friendly web applications using Flask.