Experiment 5 : Flask Application using `render_template()` function.

Name of Student	**Prathamesh Palve**
Class Roll No	**D15A_29**
D.O.P.	**06/03/2025**
D.O.S.	**13/03/2025**
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 and return them as responses to client requests. Instead of returning plain text or manually writing HTML inside the Python code, Flask allows the use of separate HTML files stored in the `templates` folder.

Usage Example:
python
CopyEdit

```
from flask import Flask,

render_template app = Flask(____name__)

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

```
    return render_template('index.html')
```

Here, `render_template('index.html')` loads the `index.html` file from the `templates` folder and sends it as a response. This helps in separating logic from presentation, making web applications more organized and maintainable.
Additionally, `render_template()` supports passing dynamic data to templates: python
CopyEdit

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

In `user.html`, we can access `username` using Jinja2 templating:
html
CopyEdit

```
<p>Hello, {{ username }}!</p>
```

---

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

The `templates` folder holds all the HTML files used for rendering web pages in a Flask application. Flask automatically looks for template files inside this directory, making it a convention that helps in maintaining a well-structured project.

Key Significance:

1. **Separation of Concerns** – Keeps the HTML structure separate from Python logic, improving code readability.
2. **Easy Management** – All templates are stored in one location, simplifying maintenance.
3. **Supports Jinja2** – Enables the use of dynamic content within HTML files through Jinja2 templating.
4. **Enables Code Reusability** – Common UI components, such as headers and footers, can be stored in separate template files and reused across multiple pages using template inheritance.

Project Structure Example:
bash
CopyEdit

```
/my_flask_app
|── app.py
|── /templates
|    |── index.html
|    |── about.html
|    |── base.html
|── /static
|    |── styles.css
|    |── script.js
```

Here, `index.html` and `about.html` are stored inside the `templates` folder and can be rendered using `render_template()`.

---

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

Jinja2 is a powerful templating engine used in Flask to generate dynamic HTML content. It allows embedding Python-like expressions inside HTML, making web pages more interactive and adaptable based on user input or backend data.

Integration with Flask:

Flask uses Jinja2 by default when rendering templates through `render_template()`. The syntax includes:

- **Variables** – `{{ variable_name }}`
- **Control Structures** – `{% if condition %} ... {% endif %}`
- **Loops** – `{% for item in list %} ... {% endfor %}`

Example Usage:

Python Code (Flask App)
python
CopyEdit

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

Jinja2 Template (`greet.html`)
html

CopyEdit

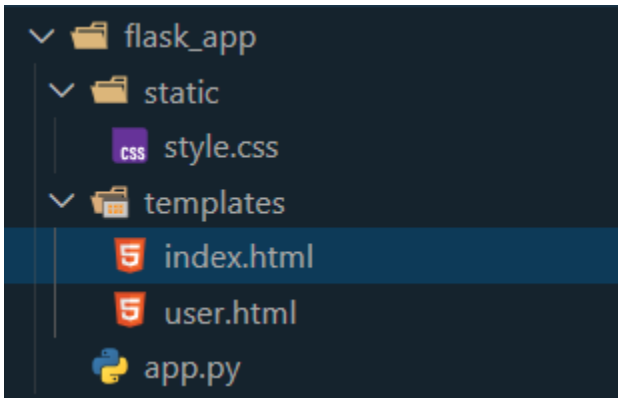
```
<!DOCTYPE html>
<html>
<head>
    <title>Greeting</title>
</head>
<body>
    <h1>Hello, {{ username }}!</h1>
</body>
</html>
```

Features of Jinja2 in Flask:

1. **Template Inheritance** – Allows reusing base layouts using `{% extends "base.html" %}` and `{% block content %} ... {% endblock %}`.
2. **Filters** – Modify data output (e.g., `{{ name.upper() }}` converts text to uppercase).
3. **Control Structures** – Supports conditionals and loops for dynamic content.

Jinja2 enhances the flexibility of Flask applications by enabling dynamic content generation within HTML templates.

OUTPUT:-



app.py

```
from flask import Flask, render_template

app = Flask(__name__)

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

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

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

/templates/index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Profile - {{ username }}</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Hello, {{ username }}!</h1>
        <p>Welcome to your personalized page.</p>

        {% if username|length > 5 %}
            <p class="long-name">That's a long name you have there!</p>
        {% else %}
            <p class="short-name">Short and sweet name!</p>
        {% endif %}

        <a href="{{ url_for('home') }}">← Back to Home</a>
    </div>
</body>
</html>
```

/templates/user.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flask Template Demo</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Welcome to Our Flask Application</h1>
        <p>This is the homepage of our template rendering demo.</p>

        <h2>Available User Pages:</h2>
        <ul class="user-list">
            {% for name in [‘Prathamesh’, 'vedant', 'Charlie', 'Diana'] %}
                <li><a href="{{ url_for('user_profile', username=name) }}">{{ name }}'s Page</a></li>
            {% endfor %}
        </ul>

        <p>Or visit <a href="/user/YourName">your custom page</a> by changing the URL.</p>
    </div>
</body>
```

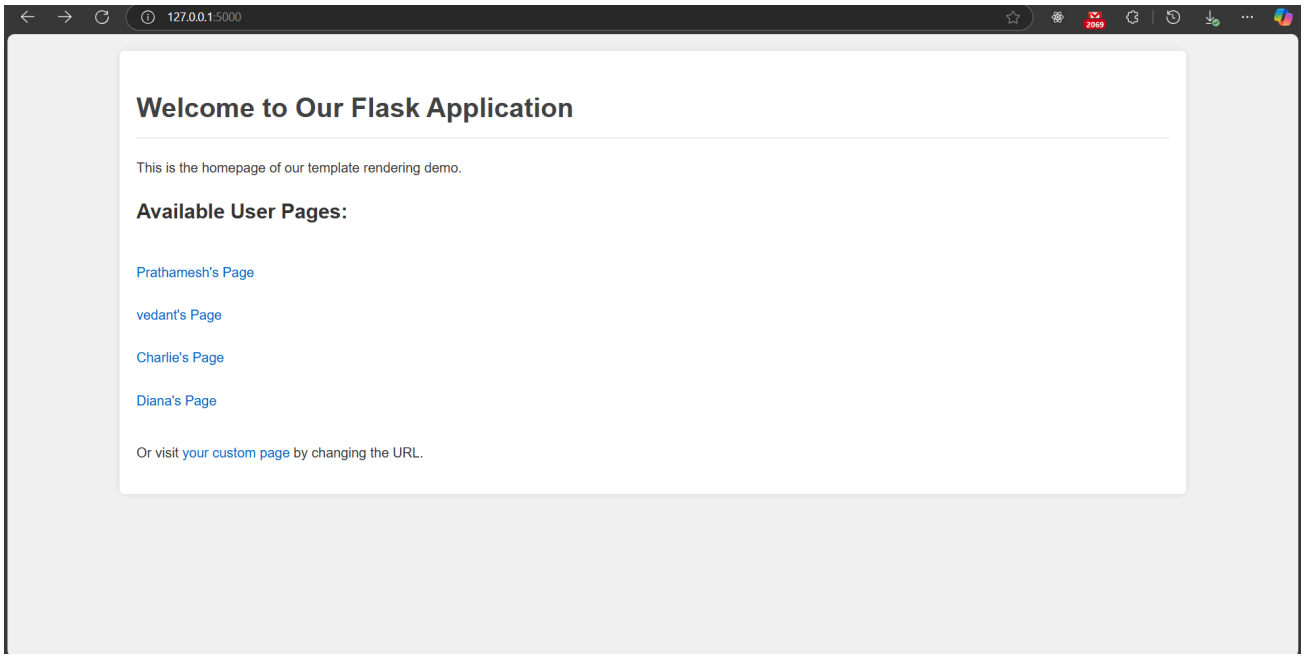
```
</html>
```

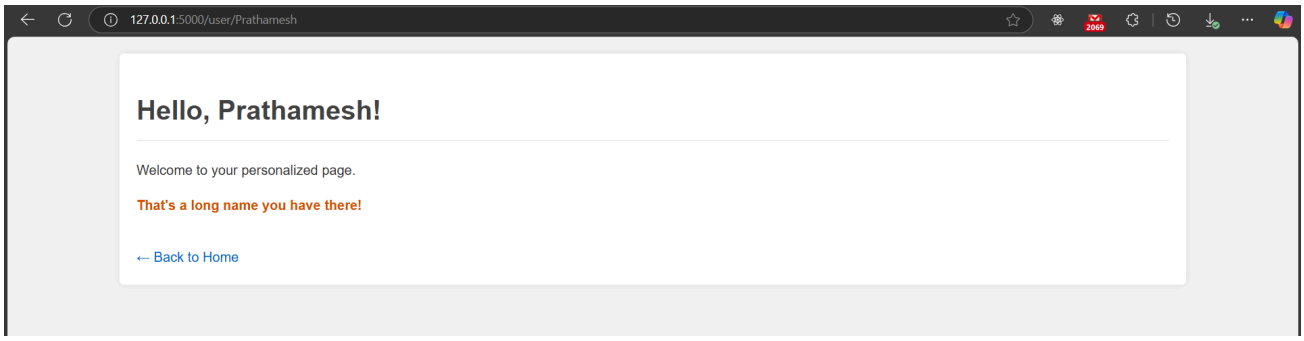
/static/style.css:

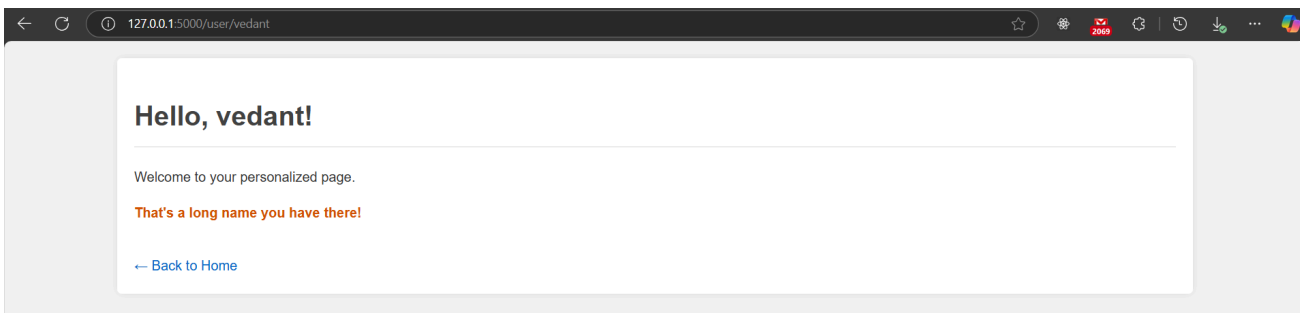
```
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
    color: #333;
}

.container {
    width: 80%;
    margin: 20px auto;
    padding: 20px;
    background: #fff;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    color: #444;
    border-bottom: 2px solid #eee;
    padding-bottom: 10px;
}

.user-list {
    list-style-type: none;
    padding: 0;
}

.user-list li {
    margin: 5px 0;
}

.user-list a {
    text-decoration: none;
    color: #0066cc;
}

.user-list a:hover {
    text-decoration: underline;
}

.long-name {
    color: #d35400;
    font-weight: bold;
}

.short-name {
    color: #27ae60;
    font-weight: bold;
```

```
}

a {
    display: inline-block;
    margin-top: 20px;
    color: #0066cc;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}
```

Results:







```
prath@prathmzzPC MINGW64 ~/Desktop/Hackathons/temp/temp4
$ 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 stat
 * Debugger is active!
 * Debugger PIN: 586-033-160
```