

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

Name of Student	Ansh Sarfare
Class Roll No	D15A_49
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.

Code:

app.py:

```
from flask import Flask, render_template  
app = Flask(__name__)  
app.secret_key = 'supersecretkey123'  
@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

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flask Template Rendering</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
<style>
  body {
    font-family: 'Poppins', sans-serif;
    background-color: #e8f0fe;
    margin: 0;
    padding: 0;
  }
  .container {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: flex-start;
    text-align: center;
    margin-top: 50px; /* Moves the content closer to the top */
  }
  .card {
    background: white;
    padding: 30px;
    border-radius: 12px;
    box-shadow: 0px 6px 8px rgba(0, 0, 0, 0.15);
    width: 60%;
    max-width: 500px;
  }
  h1 {
    color: #1a73e8;
    font-size: 28px;
  }
  p {
    font-size: 16px;
    color: #555;
    margin-bottom: 20px;
  }
  .btn-custom {
    background-color: #1a73e8;
    color: white;
    padding: 12px 24px;
    text-decoration: none;
    border-radius: 8px;
  }
```

```

        font-weight: 500;
    }
    .btn-custom:hover {
        background-color: #1257b6;
    }
    .links {
        display: flex;
        flex-direction: column;
        gap: 12px;
        margin-top: 15px;
    }
</style>
</head>
<body>
    <div class="container">
        <div class="card">
            <h1>Welcome to Flask App 🚀</h1>
            <p>Discover personalized user pages!</p>
            <div class="links">
                <a href="{{ url_for('user', username='Ansh') }}" class="btn btn-custom">Visit Ansh's
Page</a>
                <a href="{{ url_for('user', username='Akshay') }}" class="btn btn-custom">Visit
Akshay's Page</a>
            </div>
        </div>
    </div>
</body>
</html>

```

User.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Page</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
    <style>
        body {
            font-family: 'Poppins', sans-serif;
            background-color: #f3f3f3;
            margin: 0;
            padding: 0;
        }
        .container {
            display: flex;

```

```

        justify-content: center;
        align-items: flex-start; /* Aligns content to the top */
        text-align: center;
        margin-top: 50px; /* Moves the content closer to the top */
    }
    .card {
        background: white;
        padding: 40px;
        border-radius: 15px;
        box-shadow: 0px 5px 10px rgba(0, 0, 0, 0.2);
        text-align: center;
        width: 50%;
        max-width: 450px;
    }
    h1 {
        color: #ff6f00;
        font-size: 26px;
    }
    p {
        font-size: 17px;
        color: #444;
        margin-bottom: 20px;
    }
    .btn-custom {
        background-color: #ff6f00;
        color: white;
        padding: 12px 22px;
        text-decoration: none;
        border-radius: 10px;
        font-weight: 600;
        display: inline-block;
    }
    .btn-custom:hover {
        background-color: #e65c00;
    }
</style>
</head>
<body>
    <div class="container">
        <div class="card">
            <h1>Hello, {{ username }}! 🙌</h1>
            <p>This is your personalized space.</p>
            <a href="{{ url_for('home') }}" class="btn btn-custom">⬅ Back to Home</a>
        </div>
    </div>
</body>
</html>

```

Output:

