Name: Kaushik Kotian   Roll no.:30   Div : D15B   Batch : B

Experiment No. 6

Aim: Design Flask application to demonstrate the usage of routing and template

Theory:

Flask Web Framework
Flask is a micro web framework for Python based on Werkzeug and Jinja2. It is called a micro framework because it does not require particular tools or libraries. Despite its simplicity, Flask is extensible and can be used to build complex web applications. It provides the basics for web development, such as request handling, routing, and template rendering, while allowing developers to choose their tools for database integration, form validation, authentication, and more.

Project Structure
The project is structured into two main components: the Python script (app.py) that defines the Flask application and its behavior, and the HTML templates (index.html, home.html, login.html) that provide the user interface.

- app.py: This is the core of the Flask application. It sets up the Flask app, defines routes, handles requests, and renders templates. It includes essential Flask concepts like application initialization, route decorators, view functions, and the use of render_template for integrating Python logic with HTML content.

- templates/: A directory that stores HTML templates. Flask automatically looks for templates in a folder named templates within the application root. These templates use Jinja2 syntax to allow dynamic content generation based on the Python code and logic defined in app.py.

Core Concepts

- Application Setup and Configuration: The Flask application is initialized with app = Flask(__name__), and various configurations, including debug mode, are set to tailor the app's behavior.

- Routing and View Functions: Flask uses routes to bind URLs to Python functions, known as view functions. These functions define the responses for specific HTTP requests. The project uses decorators (@app.route()) to map view functions to routes.

- Template Rendering: Flask integrates with the Jinja2 templating engine, allowing Python variables and logic to be included in HTML templates. The render_template function is used to render templates and dynamically pass data from the Flask application to the HTML frontend.

- Request Handling: The project demonstrates handling GET and POST requests using the request object. This object provides access to request data, such as form fields submitted via a POST request.

- User Authentication: Basic user authentication is implemented to demonstrate form handling and conditional logic based on user input. It involves checking submitted credentials and redirecting users based on the authentication outcome.

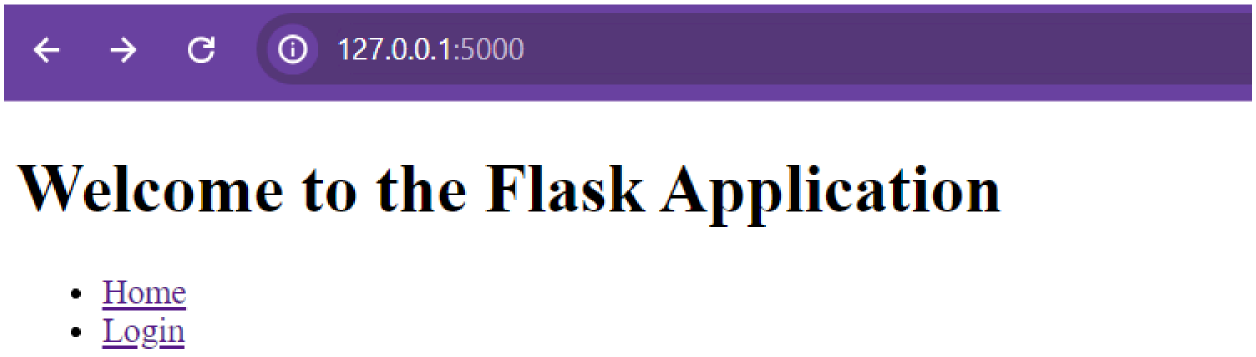
Development and Testing
The application is run in a development environment, facilitated by Flask's built-in development server. This server is suitable for development and testing but not for production use. Developers are advised to use a production WSGI server for deploying Flask applications.

Security Considerations
While this project illustrates basic concepts, it simplifies aspects like user authentication for educational purposes. In real-world applications, security measures such as password hashing, secure session management, and protection against common web vulnerabilities (e.g., SQL injection, cross-site scripting) are essential.

Code:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Welcome Page</title>
</head>
<body>
    <h1>Welcome to the Flask Application</h1>
    <nav>
        <ul>
            <li><a href="{{ url_for('home') }}">Home</a></li>
            <li><a href="{{ url_for('login') }}">Login</a></li>
        </ul>
    </nav>
</body>
</html>
```

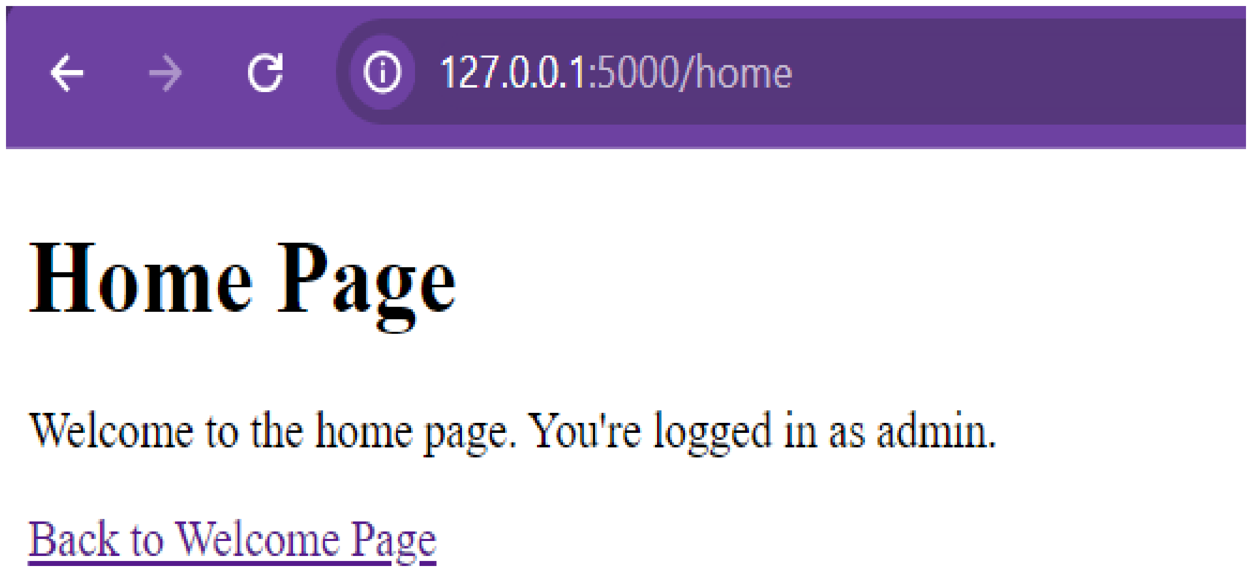


Welcome to the Flask Application

- Home
- Login

Home.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Home Page</title>
</head>
<body>
    <h1>Home Page</h1>
    <p>Welcome to the home page. You're logged in as admin.</p>
    <a href="{{ url_for('index') }}">Back to Welcome Page</a>
</body>
</html>
```



127.0.0.1:5000/home

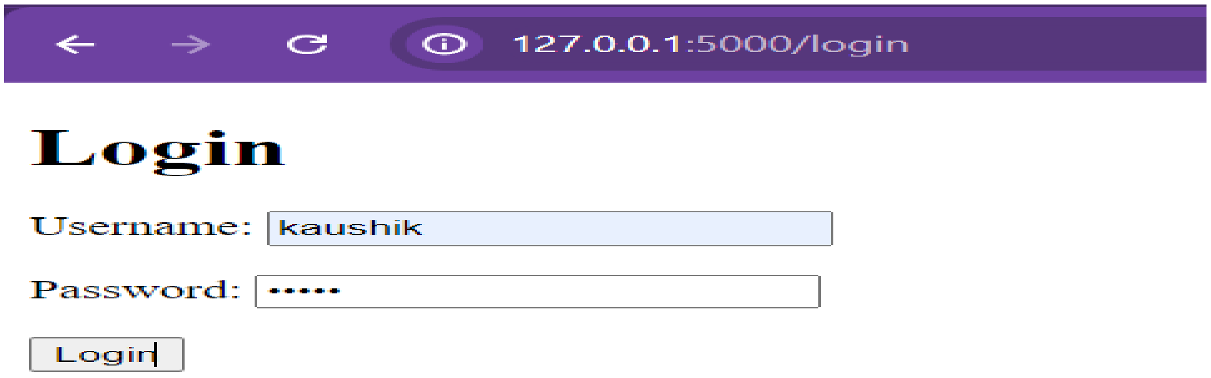
Home Page

Welcome to the home page. You're logged in as admin.

Back to Welcome Page

Login.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Login Page</title>
</head>
<body style="display: flex; justify-content: center; align-items:
center; flex-direction: column; height: 100vh;">
    <h1>Login</h1>
    <form method="POST" action="{{ url_for('login') }}">
        <div>
            <label for="user">Username:</label>
            <input type="text" id="user" name="user" required>
        </div><br/>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password"
required>
        </div><br/>
        <button type="submit" >Login</button>
    </form>
</body>
</html>
```


127.0.0.1:5000/login

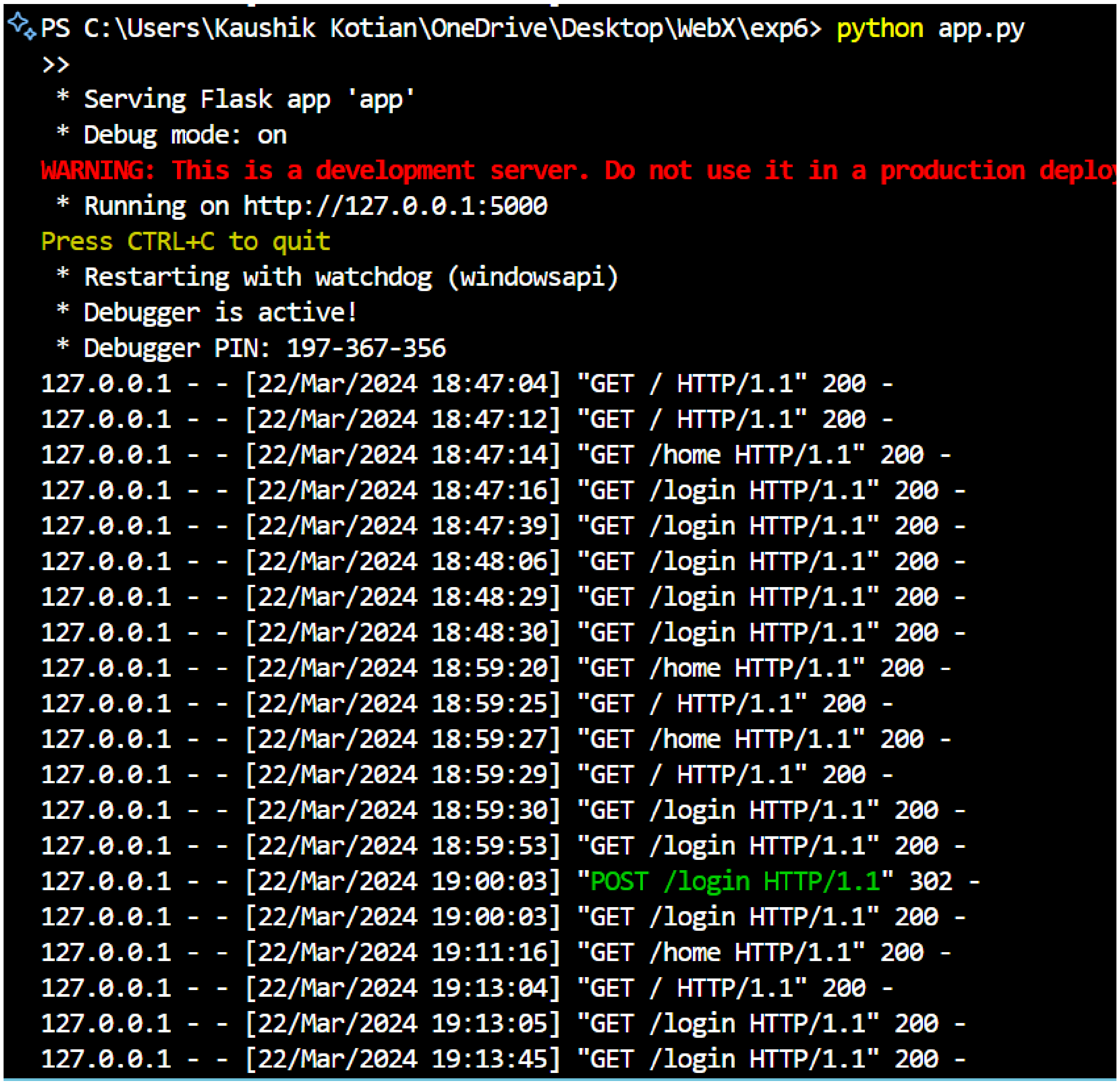
Login

Username: kaushik

Password: •••••

app.py:

```python
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

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

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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user = request.form['user']
        password = request.form['password']
        if user == 'admin' and password == 'admin':
            return redirect(url_for('home'))
        else:
            return redirect(url_for('login'))
    else:
        return render_template('login.html')

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



```
PS C:\Users\Kaushik Kotian\OneDrive\Desktop\WebX\exp6> python app.py
>>
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deplo
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 197-367-356
127.0.0.1 - - [22/Mar/2024 18:47:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:14] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:16] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:39] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:06] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:29] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:30] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:20] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:27] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:30] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:53] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:00:03] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [22/Mar/2024 19:00:03] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:11:16] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:05] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:45] "GET /login HTTP/1.1" 200 -
```

Conclusion:

This Flask project provides a foundation for understanding web application development with Flask, from setting up the application and handling HTTP requests to rendering dynamic content and implementing simple authentication. It serves as a stepping stone towards building more complex and secure web applications with Flask.