3. Activity

Student Activity: Hands-on Guided Project Using Python - Flask Framework, Routes, Templates

---

Welcome to this hands-on guided project using Python! In this activity, you will explore the **Flask Framework**, learn about **Routes**, work with **Templates**, and develop a simple web project. This guide assumes no prior knowledge of web development or Python frameworks. By the end of this session, you will have a solid understanding of how to build a basic web application using Flask.

---

1. Flask Framework: What is it?

Flask is a **micro web framework** written in Python. It is lightweight and flexible, allowing you to add only the components you need for your project.

Key Features of Flask:

- **Lightweight**: It doesn't force you to use specific tools or libraries.
- **Flexible**: You can easily extend it with additional libraries.
- **Simple**: It's easy to learn and use, especially for beginners.

Examples:

1. **Basic Flask Application**:

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

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

2. **Flask with Debug Mode**:

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Debug Mode is On!'

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

3. **Flask with Custom Port**:

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def custom_port():
    return 'Running on a custom port!'

if __name__ == '__main__':
    app.run(port=8080)
```

---

2. Routes: The Backbone of Web Applications

In Flask, **routes** are the URLs that users can visit in your web application. Each route is associated with a specific function in your Python code, which determines what the user will see when they visit that URL.

Examples:

1. **Basic Route**:

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
```

```
def home():
    return "Welcome to the Home Page!"
```

2. **Multiple Routes**:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Home Page"

@app.route('/about')
def about():
    return "About Page"
```

3. **Dynamic Route**:

```
from flask import Flask

app = Flask(__name__)

@app.route('/user/<username>')
def show_user_profile(username):
    return f'User {username}'
```

---

3. Templates: Making Your Web Pages Dynamic

In Flask, **templates** are used to create dynamic web pages. Instead of writing static HTML code for every page, you can use templates to insert dynamic content into your web pages.

Examples:

1. **Basic Template**:

```
<!-- home.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Home Page</title>
</head>
```

```
<body>
    <h1>Welcome, {{ name }}!</h1>
</body>
</html>
```

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html', name="Alice")
```

2. **Template with Loop**:

```
<!-- tasks.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Tasks</title>
</head>
<body>
    <h1>Task List</h1>
    <ul>
        {% for task in tasks %}
            <li>{{ task }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/tasks')
def tasks():
    task_list = ["Task 1", "Task 2", "Task 3"]
    return render_template('tasks.html', tasks=task_list)
```

3. **Template with Conditional**:

```
<!-- status.html -->
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Status</title>
</head>
<body>
    <h1>Status</h1>
    {% if status == 'active' %}
        <p>The user is active.</p>
    {% else %}
        <p>The user is inactive.</p>
    {% endif %}
</body>
</html>
```

```python
from flask import Flask, render_template

app = Flask(__name__)

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

---

4. Activity: Develop a Web Project

Now that we have covered the basics of Flask, routes, and templates, let’s put everything together and build a simple web project. We will create a **To-Do List Application** where users can add tasks, view them, and mark them as completed.

Step-by-Step Approach:

1. **Set Up Flask**:
    - Install Flask using pip: `pip install flask`
    - Create a new Python file (e.g., `app.py`) and import Flask.
2. **Define Routes**:
    - Create routes for the home page (to display tasks) and a route to add new tasks.
3. **Create Templates**:
    - Use HTML templates to display the list of tasks and a form to add new tasks.
4. **Handle User Input**:
    - Use Flask’s `request` object to handle form submissions and add new tasks to the list.
5. **Display Tasks**:

- Use a template to loop through the list of tasks and display them on the home page.

---

5. Project Example: To-Do List Application

Code Walkthrough:

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

app = Flask(__name__)

# List to store tasks
tasks = []

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

@app.route('/add', methods=['POST'])
def add_task():
    task = request.form['task']
    tasks.append(task)
    return redirect(url_for('home'))

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

Template (home.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>To-Do List</title>
</head>
<body>
    <h1>To-Do List</h1>
    <ul>
        {% for task in tasks %}
            <li>{{ task }}</li>
        {% endfor %}
    </ul>

    <form action="/add" method="POST">
        <input type="text" name="task" placeholder="Enter a new task">
        <button type="submit">Add Task</button>
    </form>
</body>
</html>
```

Explanation:

- The `home` route renders the `home.html` template and passes the list of tasks to it.
- The `add_task` route handles form submissions. When the user submits a new task, it is added to the `tasks` list, and the user is redirected back to the home page.

---

6. Challenges You Might Face

1. Understanding Routes:

- It might be confusing at first to understand how routes work and how they map to functions in your code. Practice by creating multiple routes and associating them with different functions.

2. Working with Templates:

- If you are new to HTML, working with templates might feel overwhelming. Start by creating simple static HTML pages and gradually introduce dynamic content using Jinja2.

3. Handling User Input:

- Handling form submissions and user input can be tricky. Make sure you understand how Flask's `request` object works and how to use it to capture data from forms.

---

7. Conclusion

In this session, we covered the basics of the Flask framework, routes, templates, and how to build a simple web project. By now, you should have a good understanding of how to:

- Set up a Flask project.
- Define routes to handle different URLs.
- Use templates to create dynamic web pages.

- Handle user input and display data on your web pages.

I encourage you to experiment with the code and try building your own projects, such as a **simple calculator** or a **file organizer**. The more you practice, the more comfortable you will become with Flask and web development in general.

---

8. Next Steps

- Try adding more features to the To-Do List application, such as the ability to delete tasks or mark them as completed.
- Explore Flask’s documentation to learn about more advanced features like database integration and user authentication.

Thank you for participating in this activity! If you have any questions or need further clarification, feel free to ask. Happy coding!