2. Handout

Flask Framework, Routes, and Templates: Student Handout

Overview

This handout provides a concise guide to building a basic web application using the Flask framework in Python. You will learn about Flask, routes, templates, and how to create a simple To-Do List application.

---

1. Flask Framework

Flask is a micro web framework for Python, known for its simplicity and flexibility. It allows you to build web applications with minimal setup.

Key Features:

- **Lightweight**: Minimal built-in features.
- **Flexible**: Easily extendable with additional libraries.
- **Simple**: Beginner-friendly.

Examples:

1. **Basic Flask App**:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Flask!"
```

2. **Running the App**:

```
$ flask run
```

3. **Debug Mode**:

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

---

2. Routes

Routes define the URLs that users can visit in your web application. Each route is linked to a function that determines the response.

Examples:

1. **Home and About Routes**:

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

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

2. **Dynamic Route**:

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

3. **HTTP Methods**:

```
@app.route('/submit', methods=['POST'])
def submit():
    return "Form Submitted"
```

---

3. Templates

Templates allow you to create dynamic web pages by inserting Python code into HTML files using the Jinja2 templating engine.

Examples:

1. **Basic Template** (`home.html`):

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome, {{ name }}!</h1>
</body>
</html>
```

2. **Rendering a Template**:

```
from flask import render_template

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

3. **Looping in Templates**:

```
<ul>
    {% for item in items %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>
```

---

4. Activity: To-Do List Application

Steps:

1. **Set Up Flask**:
    - Install Flask: `pip install flask`
    - Create `app.py` and import Flask.
2. **Define Routes**:

- Create routes for displaying and adding tasks.

3. **Create Templates**:
   - Use HTML templates for task display and input form.
4. **Handle User Input**:
   - Use Flask's `request` object to handle form submissions.
5. **Display Tasks**:
   - Loop through tasks in the template to display them.

Code Example:

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

app = Flask(__name__)

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>
```

---

5. Challenges

1. **Understanding Routes**: Mapping URLs to functions.
2. **Working with Templates**: Integrating Python code in HTML.
3. **Handling User Input**: Using the `request` object for form data.

---

6. Conclusion

You have learned how to set up a Flask project, define routes, use templates, and handle user input. Practice by building your own projects to reinforce these concepts.

---

7. Next Steps

- Add features to the To-Do List app, like deleting tasks.
- Explore Flask's documentation for advanced features like database integration.

Happy coding! If you have questions, feel free to ask.