Student Worksheet: Building and Understanding Your Boilerplate Project for Flask Applications

Objective:

This worksheet will guide you through setting up a reusable Flask project boilerplate and explain how to adapt it for different scenarios. By the end of this activity, you will understand how to create, customise, and expand your project for scenarios such as interactive tools, booking systems, and user feedback features.

Step 1: Understanding the Boilerplate Structure

Every Flask project follows a structured approach to ensure scalability and readability. Here's the general structure you'll work with:

- **`app.py`**: The backend logic that connects routes, handles requests, and interacts with the database.
- **`templates/`**: Folder containing all HTML files for the web pages.
- **`static/`**: Folder for static assets like CSS, images, and JavaScript files.
- **Database (`app.db`)**: Stores all data relevant to your project.

Key Concept:

This structure ensures that your project is modular, making it easier to manage and extend for different scenarios. For example:

- In a **surfboard customisation tool**, the focus is on user input and saving design data.
- In a **booking system**, the focus shifts to form validation, data storage, and user interaction.

Step 2: Setting Up Your Boilerplate

1. **Create a New Project Folder:**
    a. Open your file explorer.
    b. Create a folder named `FlaskProject`.
    c. Open this folder in Visual Studio Code.
2. **Create Required Files and Folders:**
    a. Inside the `FlaskProject` folder, create:
        i. `app.py` (backend logic).
        ii. `static/` (folder for stylesheets and assets).
        iii. `templates/` (folder for HTML pages).
        iv. `app.db` (SQLite database, automatically created).
3. **Install Flask:**
    a. Open the terminal in Visual Studio Code.
    b. Run the command:

```
pip install flask
```

Step 3: Adapting the Boilerplate

This boilerplate is flexible and can be adapted to different scenarios. Below are some examples:

Scenario: Interactive Surfboard Customisation Tool

- **Routes**: Add a route for uploading designs and viewing saved customisations.
- **Database**: Store user-generated surfboard specifications.
- **Frontend**: Create a drag-and-drop interface for customisation.

Scenario: Workshop Booking System

- **Routes**: Include routes for booking forms and viewing availability.
- **Database**: Manage data for workshops, bookings, and users.
- **Frontend**: Add date selectors and filtering options.

Scenario: User Feedback Forum

- **Routes**: Create routes for submitting and displaying feedback.
- **Database**: Store user reviews and ratings.
- **Frontend**: Display feedback dynamically without refreshing the page.

Step 4: Step-by-Step Code Implementation

1. Backend (app.py):

The boilerplate provides basic routes and database handling. Copy the following code into `app.py`:

```
from flask import Flask, render_template, request, redirect, url_for
import sqlite3

app = Flask(__name__)

# Database connection function
def connect_db():
    conn = sqlite3.connect('app.db')
    return conn

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

@app.route('/form', methods=['GET', 'POST'])
def form():
    if request.method == 'POST':
        data = request.form['input_data']
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO example_table (data) VALUES (?)',
(data,))
        conn.commit()
        conn.close()
```

```
        return redirect(url_for('home'))
    return render_template('form.html')

@app.route('/data')
def data():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM example_table')
    rows = cursor.fetchall()
    conn.close()
    return render_template('data.html', rows=rows)

if __name__ == '__main__':
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS example_table (
                      id INTEGER PRIMARY KEY AUTOINCREMENT,
                      data TEXT NOT NULL)''')
    conn.commit()
    conn.close()
    app.run(debug=True)
```

Explanation:

- This boilerplate handles data input (`/form`), storage, and display (`/data`).
- Adapt the routes as needed for each scenario.

2. HTML Files (Frontend):

Place the following files in the `templates/` folder.

`index.html`: The homepage

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <title>Home</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <header>
        <h1>Welcome to Flask Project</h1>
    </header>
    <main>
        <nav>
            <a href="/form">Submit Data</a> | <a href="/data">View
Data</a>
        </nav>
    </main>
</body>
</html>
```

form.html: Form page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Form</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <header>
        <h1>Submit Your Data</h1>
    </header>
    <main>
        <form method="POST">
            <label for="input_data">Enter Data:</label><br>
            <input type="text" name="input_data" required><br>
            <button type="submit">Submit</button>
        </form>
```

```
    </main>
</body>
</html>
```

data.html: Data display page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Data</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <header>
        <h1>Submitted Data</h1>
    </header>
    <main>
        <table>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Data</th>
                </tr>
            </thead>
            <tbody>
                {% for row in rows %}
                <tr>
                    <td>{{ row[0] }}</td>
                    <td>{{ row[1] }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </main>
</body>
```

```
</html>
```

Step 5: Running Your Project

1. Run `app.py` in the terminal:

```
python app.py
```

2. Open your browser and go to `http://127.0.0.1:5000`.

Step 6: Key Concepts to Adapt the Boilerplate

- **Input/Output**: Modify form fields and database structure for each project.
- **Routes**: Create new routes for additional features.
- **Styling**: Update `styles.css` for branding and accessibility.

This boilerplate is the foundation for all future projects, ensuring consistency and scalability across tasks. Adapt it for each scenario by following the steps provided.