Scenario Overview

In this task, you will develop a web-based solution using **Python and Flask** for EcoCycle Rentals, based on the project objectives outlined in your scenario. Your goal is to create a functional system that includes **user authentication**, **form validation**, **database integration**, **navigation**, **compatibility**, and **security**. Each section requires documentation, such as code explanations, screenshots, and testing outcomes.

Important Notes for Students

- This document serves as a **quick-reference guide** during the exam. Use it to stay on track and ensure you gather evidence for each section.
- **Document your progress** as you complete each part, using screenshots and brief explanations of your code and test results.
- **Time Management:** Follow the time breakdown provided to stay within the allocated 30 hours for Task 2.

Time Breakdown Overview

To help manage your time effectively, here's a suggested timeline for Task 2:

- **Day 1-2:** Flask app setup, routing, page templates, and CSS styling.
- **Day 3:** Form validation, login functionality, and database setup.
- **Day 4-5:** Backend logic for functional pages (FP1, FP2, FP3) and security measures.
- **Day 6:** Testing, debugging, and fault resolution.
- **Day 7:** Final adjustments, documentation, and final testing.

2. Functionality (8 Marks)

This section focuses on building the core features of your system in Flask. Follow each step carefully, ensuring all functions work as expected before proceeding to the next section.

2.1 Flask App Setup and Routes

- **Prompt:** Set up your Flask application and define routes for each page, forming the foundation of your app.
- **Evidence:** Screenshot of your `app.py` file with route definitions and a running instance of the app.
- **Time Guideline:** Spend up to 15 minutes on app setup and routing.

Example Code:

```
from flask import Flask, render_template
app = Flask(__name__)

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

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

2.2 Page Templates and CSS Styling

- **Prompt:** Create HTML templates and apply consistent CSS for styling.
- **Evidence:** Screenshots of the home page, login page, and key functional pages with CSS applied (if appropriate). A short discussion on how PEP-8 and other accessibility standards have been adapted into your code.
- **Time Guideline:** Allocate 30 minutes for template creation and styling.

2.3 Form Handling and Validation

- **Prompt:** Implement form validation using Flask's `request` object to process user inputs.
- **Evidence:** Screenshot of form validation in action with a brief explanation of the different validation checks that are appropriate for the task, either as Python code or SQL integration.
- **Time Guideline:** Spend 20 minutes implementing and testing form validation.

Example Code:

```
@app.route('/submit', methods=['POST'])
def submit():
    name = request.form['name']
    if not name:
        return "Error: Name is required"
    # Continue processing the form
```

2.4 Database Integration

- **Prompt:** Connect your Flask app to a database (e.g., SQLite) for secure data storage and retrieval, make sure you can see the database table content using database view.
- **Evidence:** Screenshots of your database schema and example queries, with a brief explanation.
- **Time Guideline:** Spend 45 minutes on database setup and integration.

Example Code:

```
import sqlite3
conn = sqlite3.connect('users.db')
c = conn.cursor()
c.execute('''CREATE TABLE users (id INTEGER PRIMARY KEY, name
TEXT)''')
```

2.5 Backend Logic for Login

- **Prompt:** Develop backend logic for user authentication (login and logout) using Flask sessions.
- **Evidence:** Screenshot of login page code and session management with an explanation.
- **Time Guideline:** Spend 30 minutes on login implementation.

Example Code:

```
from flask import session
session['user'] = username
```

2.6 Backend Logic for Functional Pages (FP1, FP2, FP3)

- **Prompt:** Implement backend functionality for key features like booking, GPS tracking, and loyalty rewards.
- **Evidence:** Provide screenshots of backend code and functionality output, and describe database interactions.
- **Time Guideline:** Spend 45 minutes per functional page.

2.7 Security Measures

- **Prompt:** Add security features, such as input sanitisation and encryption, to protect user data.
- **Evidence:** Explanation of security measures, with screenshots showing handling of sensitive data (e.g., password encryption).
- **Time Guideline:** Spend 20 minutes on security implementation.

3. Code Organisation (8 Marks)

This section assesses code structure and documentation.

3.1 Code Comments

- **Prompt:** Add clear comments explaining the purpose of functions and major code blocks.
- **Evidence:** Screenshot of commented code, highlighting complex sections.
- **Time Guideline:** Spend 10 minutes adding comments post-coding.

3.2 Naming Conventions

- **Prompt:** Use consistent, meaningful naming conventions for variables, functions, and files.
- **Evidence:** Screenshot of well-named variables and functions.
- **Time Guideline:** Apply naming conventions as you code.

3.3 File Structure

- **Prompt:** Organise files in directories (e.g., `templates`, `static`) for logical structure.
- **Evidence:** Screenshot of the project directory.
- **Time Guideline:** Spend 10 minutes structuring files.

4. User Experience (8 Marks)

Focuses on user interface and feedback mechanisms.

4.1 Form Validation and Feedback

- **Prompt:** Ensure users receive clear error messages when entering invalid data.
- **Evidence:** Screenshot of form feedback in action with a description.
- **Time Guideline:** Spend 10 minutes testing form feedback.

4.2 Navigation and Layout

- **Prompt:** Create a consistent layout across pages for a seamless user experience.
- **Evidence:** Screenshots of pages showing navigation and layout consistency.
- **Time Guideline:** Spend 10 minutes refining navigation.

5. Legal and Regulatory Compliance (8 Marks)

Ensures adherence to legal, regulatory, and compatibility standards.

5.1 Accessibility

- **Prompt:** Ensure your system complies with accessibility standards (e.g., WCAG) using ARIA labels, keyboard navigation, and other accessibility features.
- **Evidence:** Provide a brief explanation of your accessibility features and include screenshots demonstrating compliance.
- **Time Guideline:** Spend 10 minutes testing accessibility.

5.2 Compatibility

- **Prompt:** Ensure your system is compatible across major web browsers (e.g., Chrome, Firefox, Edge) and devices (e.g., desktops, tablets, mobile).
- **Evidence:** Test the system on different browsers and devices, and document the results with screenshots and a compatibility checklist.
- **Time Guideline:** Spend 15 minutes verifying compatibility on key platforms.

Example Compatibility Checklist:

Platform	Browser/Device	Status
Desktop	Chrome	Pass
Desktop	Firefox	Pass
Mobile	Safari (iOS)	Pass
Tablet	Chrome (Android)	Pass

5.3 Security and Privacy

- **Prompt:** Implement secure coding practices, such as data encryption and input validation, to protect user data and comply with regulations (e.g., GDPR).
- **Evidence:** Include a screenshot of secure form handling, encrypted data, or other security features with a brief explanation.
- **Time Guideline:** Spend 15 minutes documenting security measures.

6. Testing (6 Marks)

Evaluate testing thoroughness and documentation.

6.1 Testing for Different Data Types

- **Prompt:** Test normal, erroneous, extreme, boundary, and null data inputs.
- **Evidence:** Complete a test table with inputs, expected outcomes, actual outcomes, and status.
- **Time Guideline:** Spend 20 minutes testing and documenting.

Example Test Table:

Test Case	Input	Expected Outcome	Actual Outcome	Status
Normal Data	Valid username/password	User logs in successfully	User logs in successfully	Pass
Erroneous Data	Incorrect password	Display error message	Error message displayed	Pass
Extreme Data	Long username (100+ chars)	Display error or handle input	Error message displayed	Pass
Boundary Data	Password exactly 8 chars	User logs in successfully	User logs in successfully	Pass
Null Data	Empty fields	Display error message	Error message displayed	Pass

6.2 Debugging and Fault Resolution

- **Prompt:** Document bugs encountered and resolutions applied.
- **Evidence:** Screenshots of debugging steps or error messages and solutions.
- **Time Guideline:** Spend 15 minutes debugging and documenting.

7. Documentation (6 Marks)

Ensures clarity and completeness in project documentation.

7.1 Rationale for Technology Choices

- **Prompt:** Justify your choice of tools, including Flask and Python.
- **Evidence:** 2-3 sentences explaining your technology choices.
- **Time Guideline:** Spend 5 minutes on this section.

7.2 Version Control and Change Log

- **Prompt:** Use Git to track changes and maintain a change log.
- **Evidence:** Screenshot of Git history with key updates.
- **Time Guideline:** Spend 10 minutes maintaining the log.

Final Checklist for Task 2: Development (30 Marks)

- Have you created all necessary pages, routes, and back-end functionality in Flask?
- Have you tested your system with different types of inputs?
- Have you produced screenshots and brief explanations as evidence for each section?
- Is your code well-organised, commented, and documented?
- Have you ensured that your system is accessible, compatible, secure, and legally compliant?

Final Notes for Students

- **Take short breaks** after completing major milestones to stay focused and productive.
- **Debug immediately** if you encounter an issue and document the solution.
- **Test as you progress** to avoid accumulating issues at the end.
- Remember, **you've got this!** Stay calm, stay focused, and trust your skills. You're building something great.