AI ASSISTED CODING -14.3

NAME: NAREDDY JASMITHA

ENROLLMENT NO: 2403A51281

COURSE: AI ASSISTED CODING

SPECILIZATION: CSE-AIML

DATE :28-10-2025

Task Description #1 – AI-generated HTML Page
Task: Ask AI to generate a simple HTML homepage for a "Student Info
Portal" with a header, navigation menu, and footer

PROMPT:

Develop a code to generate aTask: Ask AI to generate a simple HTML homepage for a "Student Info Portal" with a header, navigation menu, and footer. Expected Output: • HTML code with <header>, <nav>, <footer>. • Clean indentation, proper tags, and comments

Expected Output:
- **HTML code with <header>, <nav>, <footer>.**
- **Clean indentation, proper tags, and comments**

```
<header>
    <h1>Welcome to the Student Info Portal</h1>
    <!-- Header content goes here -->
</header>

<nav>
    <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Students</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
    <!-- Navigation links go here -->
</nav>

<main>
    <!-- Main content of the page goes here -->
    <p>This is the homepage of the Student Info Portal.</p>
</main>

<footer>
    <p>&copy; 2023 Student Info Portal. All rights reserved.</p>
    <!-- Footer content goes here -->
</footer>
```

EXPLANATION:

- `<!DOCTYPE html>`: Declares the document type to be HTML5.
- `<html lang="en">`: The root element of the HTML page. The `lang="en"` attribute specifies the language of the document as English.
- `<head>`: Contains meta-information about the HTML document, such as character set, viewport settings, and the title.
    - `<meta charset="UTF-8">`: Specifies the character encoding for the document, supporting a wide range of characters.
    - `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Configures the viewport for responsive web design, ensuring the page scales correctly on different devices.
    - `<title>Student Info Portal</title>`: Sets the title of the HTML page, which appears in the browser's title bar or tab.
- `<body>`: Contains the visible content of the HTML page.
    - `<header>`: Represents the introductory content or a container

elements and a logo.

- `<h1>Welcome to the Student Info Portal</h1>`: The main heading of the page.
- `<!-- Header content goes here -->`: An HTML comment indicating where additional header content could be placed.

- `<nav>`: Represents a section of a page that links to other pages or parts within the page.
  - `<ul>`: An unordered list, used here for the navigation links.
  - `<li><a href="#">...</a></li>`: List items containing anchor tags (`<a>`) which create hyperlinks. The `#` as the `href` value is a placeholder.
  - `<!-- Navigation links go here -->`: An HTML comment indicating where more navigation links could be added.

- `<main>`: Represents the dominant content of the `<body>` of a document.
  - `<!-- Main content of the page goes here -->`: An HTML comment indicating where the primary content of the page should be placed.
  - `<p>This is the homepage of the Student Info Portal.</p>`: A paragraph of text within the main content area.

- `<footer>`: Represents a footer for its nearest sectioning content or the root element (`<html>`).
  - `<p>&copy; 2023 Student Info Portal. All rights reserved.</p>`: A paragraph containing copyright information. The `&copy;` is an HTML entity for the copyright symbol.
  - `<!-- Footer content goes here -->`: An HTML comment indicating where additional footer content

Task Description #2 – CSS Styling

Task:

Use AI to add CSS styling to Task #1 homepage for:

- **Responsive navigation bar.**
- **Centered content section.**
- **Footer with light gray background.**

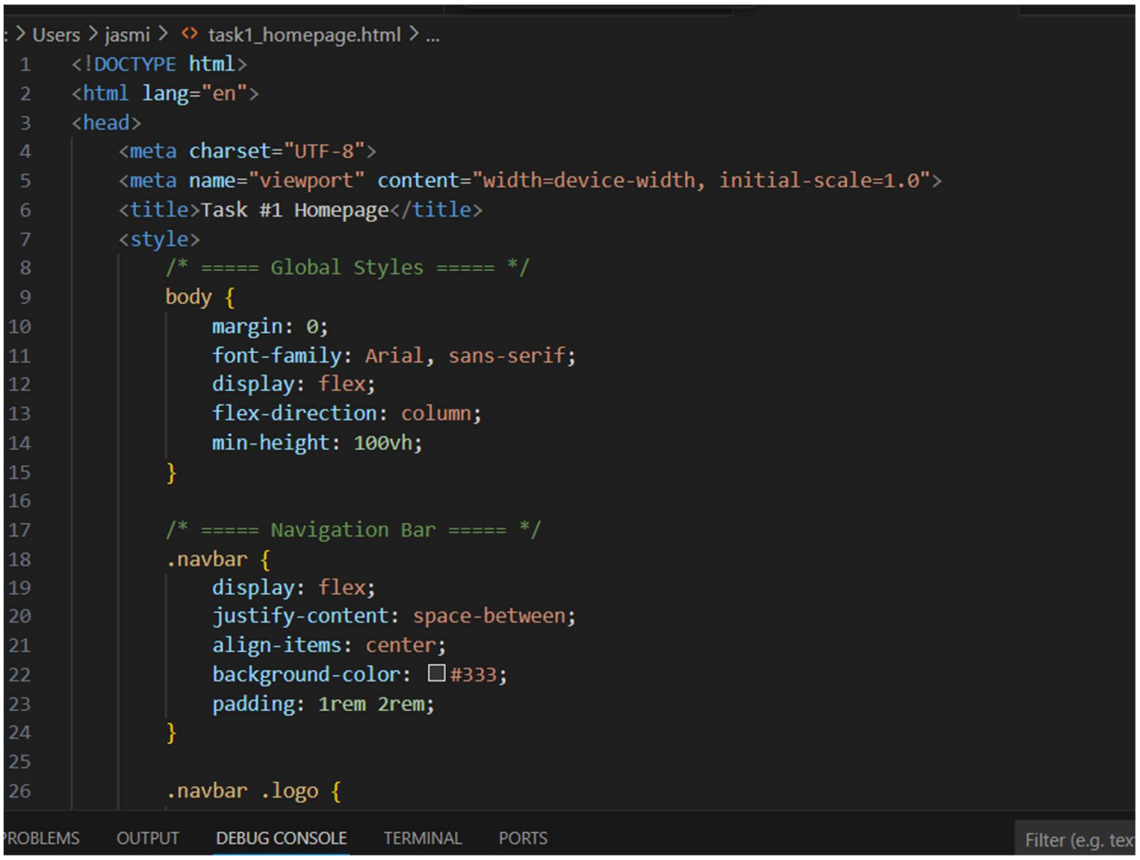
PROMPT:

Develop a python code to generate Use AI to add CSS styling to Task #1 homepage for: • Responsive navigation bar. • Centered content section. • Footer with light gray background. Expected Output: • HTML + CSS combined. • AI explains how CSS classes apply

Expected Output:

- **HTML + CSS combined.**
- **AI explains how CSS classes apply**

Expected Output: AI refactors with with open() and try-except:


: > Users > jasmi > task1_homepage.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Task #1 Homepage</title>
7 <style>
8 /* ===== Global Styles ===== */
9 body {
10 margin: 0;
11 font-family: Arial, sans-serif;
12 display: flex;
13 flex-direction: column;
14 min-height: 100vh;
15 }
16
17 /* ===== Navigation Bar ===== */
18 .navbar {
19 display: flex;
20 justify-content: space-between;
21 align-items: center;
22 background-color: #333;
23 padding: 1rem 2rem;
24 }
25
26 .navbar .logo {
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. tex

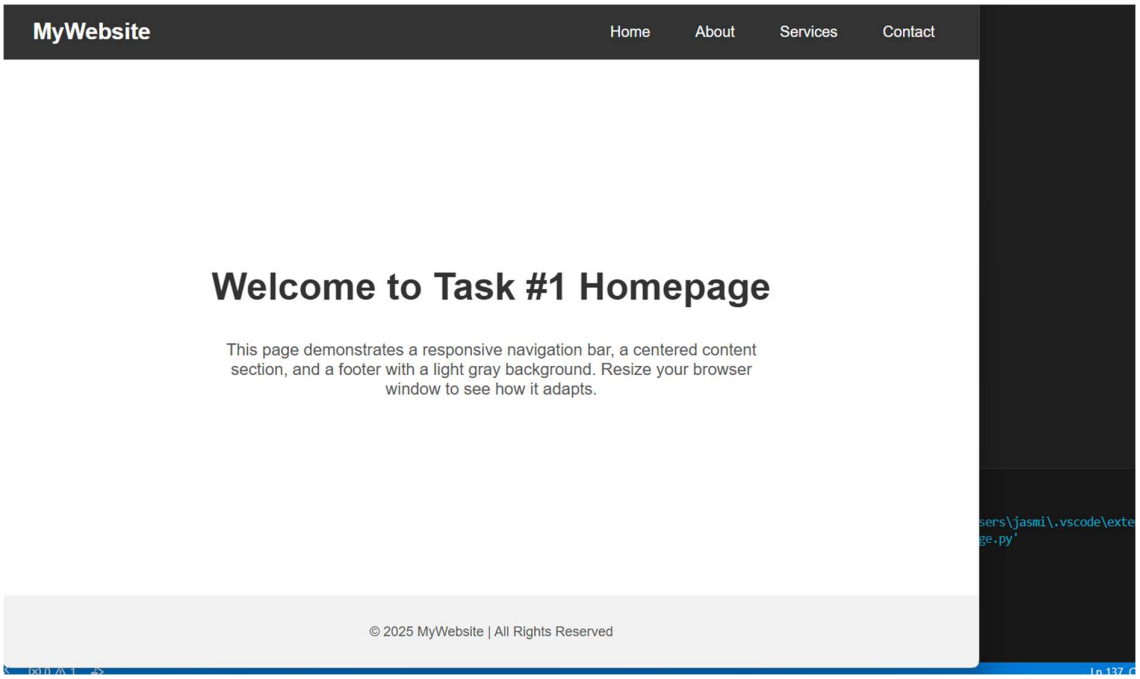
```
<html lang="en">
<head>
    <style>
        @media (max-width: 600px) {
            .navbar {
            }

            .navbar ul {
                flex-direction: column;
                width: 100%;
            }

            .navbar ul li {
                width: 100%;
            }

            .navbar ul li a {
                display: block;
                width: 100%;
                padding: 0.75rem;
            }
        }

        /* ===== Centered Content Section ===== */
        .content {
            flex: 1;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            text-align: center;
            padding: 2rem;
        }

        .content h1 {
            font-size: 2.5rem;
            color: #333;
```

```
    </style>
</head>
<body>

    <!-- ===== Navigation Bar ===== -->
    <nav class="navbar">
        <a href="#" class="logo">MyWebsite</a>
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Services</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>

    <!-- ===== Centered Content Section ===== -->
    <section class="content">
        <h1>Welcome to Task #1 Homepage</h1>
        <p>This page demonstrates a responsive navigation bar, a centered content section, and a
    </section>

    <!-- ===== Footer Section ===== -->
    <footer class="footer">
        <p>© 2025 MyWebsite | All Rights Reserved</p>
    </footer>

</body>
</html>
```

const express = require('express'); Untitled-1 ● | Release Notes: 1.105.1 | task1_homepage.html | generate_homepage.py 1 ×

C: > Users > jasmi > generate_homepage.py > ...

```python
try:
    with open("task1_homepage.html", "w", encoding="utf-8") as file:
        file.write(html_code)
    print("✅ task1_homepage.html created successfully!")
except Exception as e:
    print(f"❌ An error occurred while writing the file: {e}")
```



EXPLANATION :

- `html_code = """..."""`: This defines a multiline string variable named `html_code` which holds the entire HTML content for the homepage.
- `try:`: This block starts a `try` block, which is used to handle potential errors that might occur during the file writing process.
- `with open("task1_homepage.html", "w", encoding="utf-8") as file:`: This opens a file named `task1_homepage.html` in write mode (`"w"`). If the file doesn't exist, it will be created. If it exists, its content will be overwritten. The `encoding="utf-8"` ensures that the file is saved with UTF-8 encoding, which is a common and recommended encoding for web pages. The `with` statement ensures that the file is automatically closed even if errors occur. The opened file object is assigned to the variable `file`.
- `file.write(html_code)`: This line writes the content of the `html_code` variable into the opened file.
- `print("✅ task1_homepage.html created successfully!")`: If the file is written successfully without any errors, this line prints a

- `file.write(html_code)`: This line writes the content of the `html_code` variable into the opened file.
- `print("✅ task1_homepage.html created successfully!")`: If the file is written successfully without any errors, this line prints a success message.
- `except Exception as e:`: This block catches any exception that might occur during the file writing process.
- `print(f"❌ An error occurred while writing the file: {e}")`: If an error occurs, this line prints an error message indicating that there was an issue writing the file, along with the specific error message (`e`).

In summary, this code provides a robust way to generate an HTML file programmatically, including basic error handling.

Task Description #3 – JavaScript Interactivity
Task: Prompt AI to generate a JS script that validates a simple login
form (non-empty username/password)

PROMPT:

Develop a code to generate Task: Prompt AI to generate a JS script that validates a simple login form (non-empty username/password). Expected Output: Working on submit JS validation. Clear error messages if inputs are empty

Expected Output:
Working on submit JS validation.
Clear error messages if inputs are empty.

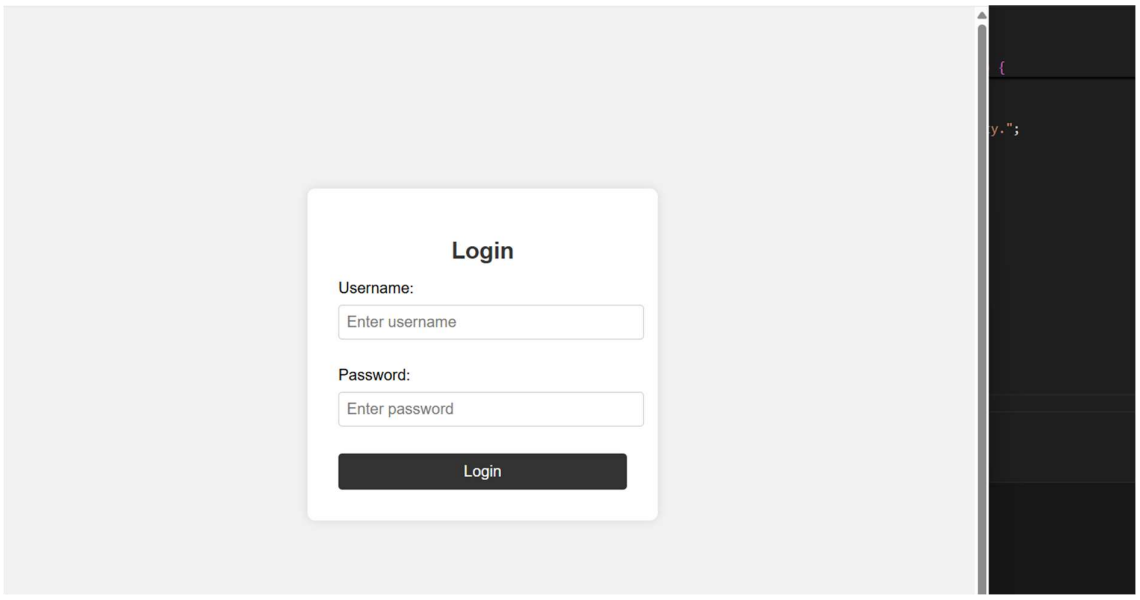
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login Form Validation</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      background-color: #f2f2f2;
    }
    .login-container {
      background: white;
      padding: 2rem;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
      width: 300px;
    }
    h2 {
      text-align: center;
      margin-bottom: 1rem;
      color: #333;
    }
    input[type="text"], input[type="password"] {
      width: 100%;
      padding: 0.5rem;
      margin: 0.5rem 0;
      border: 1px solid #ccc;
      border-radius: 4px;
      font-size: 1rem;
    }
    .error {
      color: red;
```

Spaces: 4 UTF-8 { } HTML

```
  <style>
    .error {
      color: red;
      font-size: 0.9rem;
      margin-top: -0.25rem;
      margin-bottom: 0.5rem;
      height: 1rem;
    }
    button {
      width: 100%;
      padding: 0.6rem;
      background-color: #333;
      color: white;
      border: none;
      border-radius: 4px;
      cursor: pointer;
      font-size: 1rem;
    }
    button:hover {
      background-color: #555;
    }
  </style>
</head>
<body>

  <div class="login-container">
    <h2>Login</h2>
    <form id="loginForm">
      <label for="username">Username:</label>
      <input type="text" id="username" placeholder="Enter username">
      <div id="userError" class="error"></div>

      <label for="password">Password:</label>
      <input type="password" id="password" placeholder="Enter password">
      <div id="passError" class="error"></div>
```

Spaces: 4 UTF-8 { } HTML

```
    // Validate username
    if (username === "") {
      document.getElementById("userError").textContent = "Username cannot be empty.";
      valid = false;
    }

    // Validate password
    if (password === "") {
      document.getElementById("passError").textContent = "Password cannot be empty.";
      valid = false;
    }

    // Prevent form submission if invalid
    if (!valid) {
      event.preventDefault();
    } else {
      alert("Login successful!"); // optional success feedback
    }
  });
</script>

</body>
</html>
```



EXPLANATION:

HTML Structure:

- `<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`: Standard HTML document structure.
- `<title>Login Form Validation</title>`: Sets the title of the web page.
- `<div class="login-container">`: A container `div` to hold the login form, styled to be centered and have a box shadow.
- `<h2>Login</h2>`: The heading for the login form.
- `<form id="loginForm">`: The form element with an ID `loginForm` which is used by the JavaScript to access the form.
- `<label for="username">Username:</label>` and `<input type="text" id="username" placeholder="Enter username">`: A label and input field for the username. The `id="username"` is used by the JavaScript to get the input value.
- `<div id="userError" class="error"></div>`: A `div` with an ID `userError` and class `error` to display username validation error messages.
- `<label for="password">Password:</label>` and `<input`

messages.

- `<label for="password">Password:</label>` and `<input type="password" id="password" placeholder="Enter password">`: A label and input field for the password. The `id="password"` is used by the JavaScript.
- `<div id="passError" class="error"></div>`: A `div` with an I `passError` and class `error` to display password validation error messages.
- `<button type="submit">Login</button>`: The submit button fo the form.

CSS Styling (within the `<style>` tags):

- Provides basic styling for the body, centering the content.
- Styles the `.login-container` to create a visually distinct box for the login form.
- Styles the heading (`h2`), input fields (`input[type="text"]`, `input[type="password"]`), error messages (`.error`), and the button.

red and have a fixed height to prevent layout shifts.

JavaScript (within the `<script>` tags):

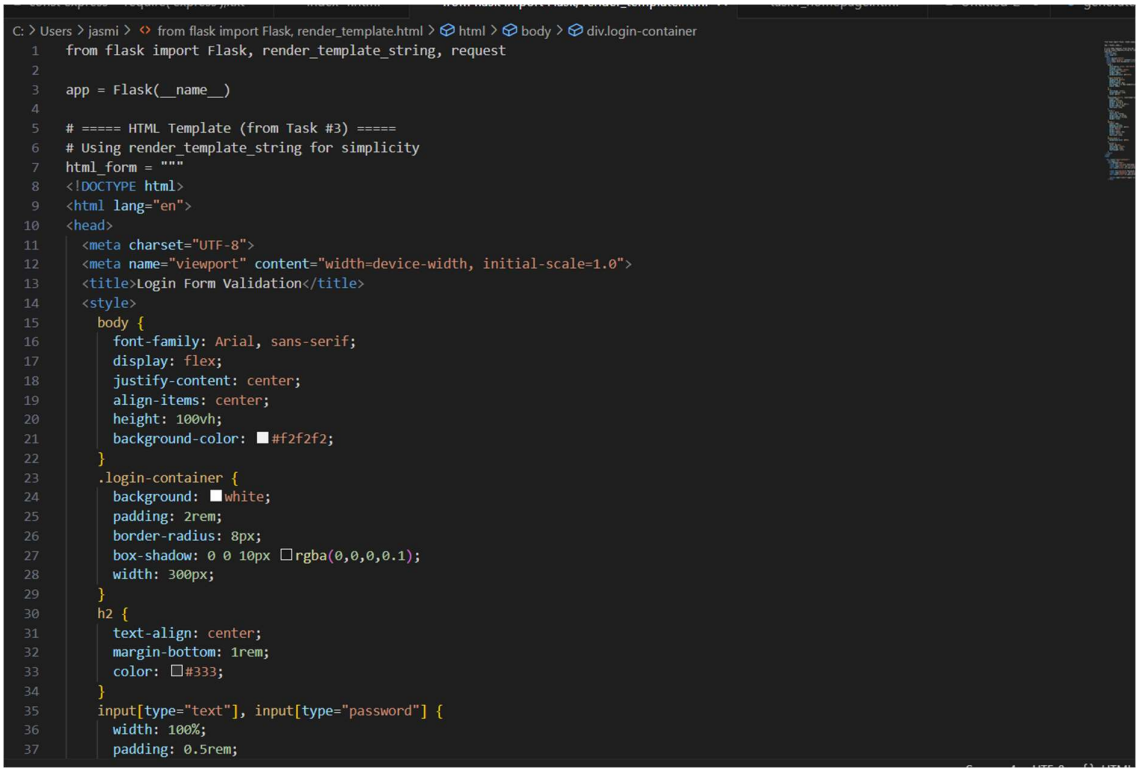
- `document.getElementById("loginForm").addEventListener("submit", function(event) { ... });`: This attaches an event listener to the form with the ID `loginForm`. When the form is submitted, the function inside the event listener is executed. The `event` object is passed to the function.
- `document.getElementById("userError").textContent = "";` and `document.getElementById("passError").textContent = "";`: These lines clear any previously displayed error messages when the form is submitted again.
- `const username = document.getElementById("username").value.trim();` and `const password = document.getElementById("password").value.trim();`: These lines get the values entered in the username and password input fields, and the `.trim()` method removes any leading or trailing whitespace.

Task Description #4 – Python Backend Integration
Task: Ask AI to generate a Flask app that serves the HTML form (Task
#3) and prints the username on successful login.

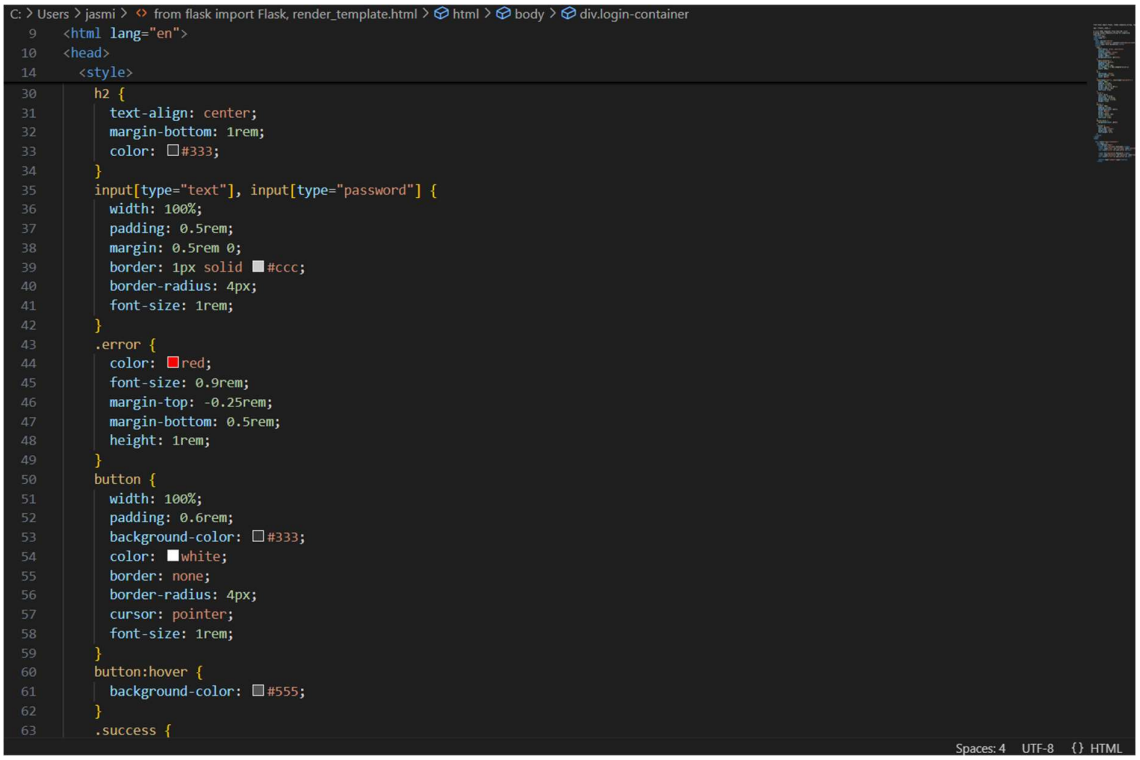
PROMPT:

Generate a code to develop Task: Ask AI to generate a Flask app that serves the HTML form (Task #3) and prints the username on successful login.

Expected output:



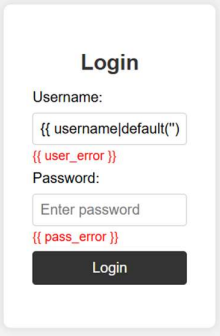
```
C: > Users > jasmi > from flask import Flask, render_template.html > html > body > div.login-container
1   from flask import Flask, render_template_string, request
2
3   app = Flask(__name__)
4
5   # ===== HTML Template (from Task #3) =====
6   # Using render_template_string for simplicity
7   html_form = """
8   <!DOCTYPE html>
9   <html lang="en">
10  <head>
11    <meta charset="UTF-8">
12    <meta name="viewport" content="width=device-width, initial-scale=1.0">
13    <title>Login Form Validation</title>
14    <style>
15      body {
16        font-family: Arial, sans-serif;
17        display: flex;
18        justify-content: center;
19        align-items: center;
20        height: 100vh;
21        background-color: #f2f2f2;
22      }
23      .login-container {
24        background: white;
25        padding: 2rem;
26        border-radius: 8px;
27        box-shadow: 0 0 10px rgba(0,0,0,0.1);
28        width: 300px;
29      }
30      h2 {
31        text-align: center;
32        margin-bottom: 1rem;
33        color: #333;
34      }
35      input[type="text"], input[type="password"] {
36        width: 100%;
37        padding: 0.5rem;
```



```
C: > Users > jasmi > from flask import Flask, render_template.html > html > body > div.login-container
9   <html lang="en">
10  <head>
14    <style>
30      h2 {
31        text-align: center;
32        margin-bottom: 1rem;
33        color: #333;
34      }
35      input[type="text"], input[type="password"] {
36        width: 100%;
37        padding: 0.5rem;
38        margin: 0.5rem 0;
39        border: 1px solid #ccc;
40        border-radius: 4px;
41        font-size: 1rem;
42      }
43      .error {
44        color: red;
45        font-size: 0.9rem;
46        margin-top: -0.25rem;
47        margin-bottom: 0.5rem;
48        height: 1rem;
49      }
50      button {
51        width: 100%;
52        padding: 0.6rem;
53        background-color: #333;
54        color: white;
55        border: none;
56        border-radius: 4px;
57        cursor: pointer;
58        font-size: 1rem;
59      }
60      button:hover {
61        background-color: #555;
62      }
63      .success {
                                                    Spaces: 4   UTF-8   {} HTML
```

```
<head>
  <style>
    .success {
      text-align: center;
      margin-top: 1rem;
      font-weight: bold;
    }
  </style>
</head>
<body>

  <div class="login-container">
    <h2>Login</h2>
    <form method="POST">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" placeholder="Enter username" value="{{ username|default('') }}">
      <div class="error">{{ user_error }}</div>

      <label for="password">Password:</label>
      <input type="password" id="password" name="password" placeholder="Enter password">
      <div class="error">{{ pass_error }}</div>

      <button type="submit">Login</button>
    </form>

```

```
from flask import Flask, render_template_string, request app = Flask(__name__) # ===== HTML Template (from Task #3) ===== # Using render_template_string for simplicity html_form = """
```




```
value="{{ username|default('
ord">
```

EXPLANATION:

HTML Structure:

- `<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`: Standard HTML document structure.
- `<title>Login Form Validation</title>`: Sets the title of the web page.
- `<div class="login-container">`: A container `div` to hold the login form, styled to be centered and have a box shadow.
- `<h2>Login</h2>`: The heading for the login form.
- `<form id="loginForm">`: The form element with an ID `loginForm` which is used by the JavaScript to access the form.
- `<label for="username">Username:</label>` and `<input type="text" id="username" placeholder="Enter username">`: A label and input field for the username. The `id="username"` is used by the JavaScript to get the input value.
- `<div id="userError" class="error"></div>`: A `div` with an ID `userError` and class `error` to display username validation error messages.
- `<label for="password">Password:</label>` and `<input`