EXPERIMENT-1

AIM

To design and implement a basic student registration form using HTML5 that collects user information such as name, email, and age, with built-in validation for each field.

OBJECTIVE

- To understand and apply basic HTML5 form elements.
- To use HTML5 input attributes for form validation (e.g., required, type, min, pattern).
- To build a responsive, user-friendly registration form.

THEORY

HTML5 provides new input types and attributes that simplify form validation and improve user experience. For instance:
- type="email" ensures a proper email format.
- type="number" with min and max restricts age input.
- required ensures that the field cannot be left empty.

These validations are performed client-side, reducing unnecessary server requests and improving performance.

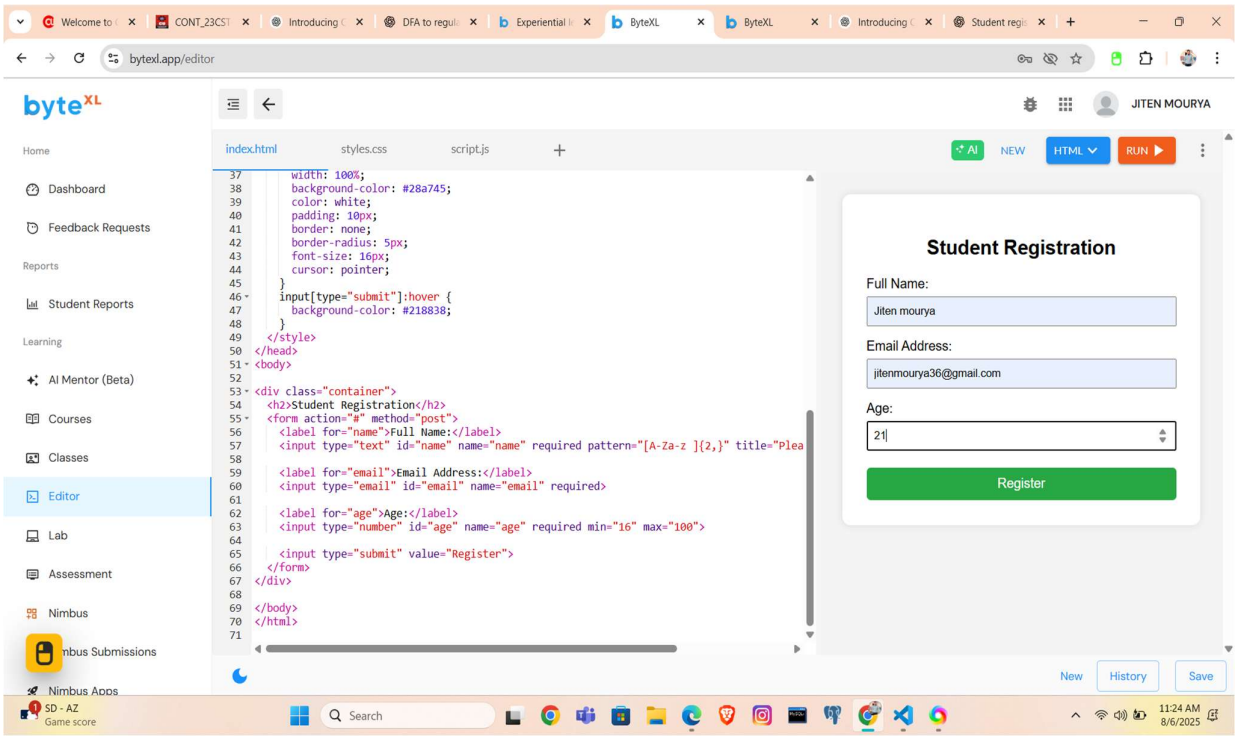
CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Registration Form</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f4f4f4;
      padding: 20px;
    }
    .container {
      background: #fff;
      max-width: 400px;
      margin: auto;
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
```

```
    }
    h2 {
      text-align: center;
      margin-bottom: 20px;
    }
    label {
      display: block;
      margin: 15px 0 5px;
    }
    input[type="text"],
    input[type="email"],
    input[type="number"] {
      width: 100%;
      padding: 8px;
      box-sizing: border-box;
    }
    input[type="submit"] {
      margin-top: 20px;
      width: 100%;
      background-color: #28a745;
      color: white;
      padding: 10px;
      border: none;
      border-radius: 5px;
      font-size: 16px;
      cursor: pointer;
    }
    input[type="submit"]:hover {
      background-color: #218838;
    }
  </style>
</head>
<body>

<div class="container">
  <h2>Student Registration</h2>
  <form action="#" method="post">
    <label for="name">Full Name:</label>
    <input type="text" id="name" name="name" required pattern="[A-Za-z ]{2,}"
title="Please enter a valid name.">

    <label for="email">Email Address:</label>
    <input type="email" id="email" name="email" required>

    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required min="16" max="100">

    <input type="submit" value="Register">
  </form>
</div>

</body>
```

```
</html>
```

OUTPUT



LEARNING OUTCOMES

- Create HTML forms with proper structure and styling.
- Implement HTML5 form validation without using JavaScript.
- Understand form submission and input data constraints.
- Enhance user experience through clean, validated forms.

EXPERIMENT-2

AIM

To design and develop a responsive banking user interface using only HTML and CSS, which includes a balance display, deposit button, and withdraw button.

OBJECTIVE

- Practice modern layout techniques using HTML and CSS.
- Create a responsive and visually clean UI that works on desktops, tablets, and mobile devices.
- Apply design principles like alignment, spacing, and centering using Flexbox and media queries.

THEORY

HTML and CSS provide powerful tools for building visually appealing and responsive interfaces. In this task:

- HTML structures the content (balance, buttons).
- CSS Flexbox centers content both vertically and horizontally and manages layout.
- Media queries make the layout responsive across different screen sizes.
- The UI must be static—no interactivity with JavaScript.

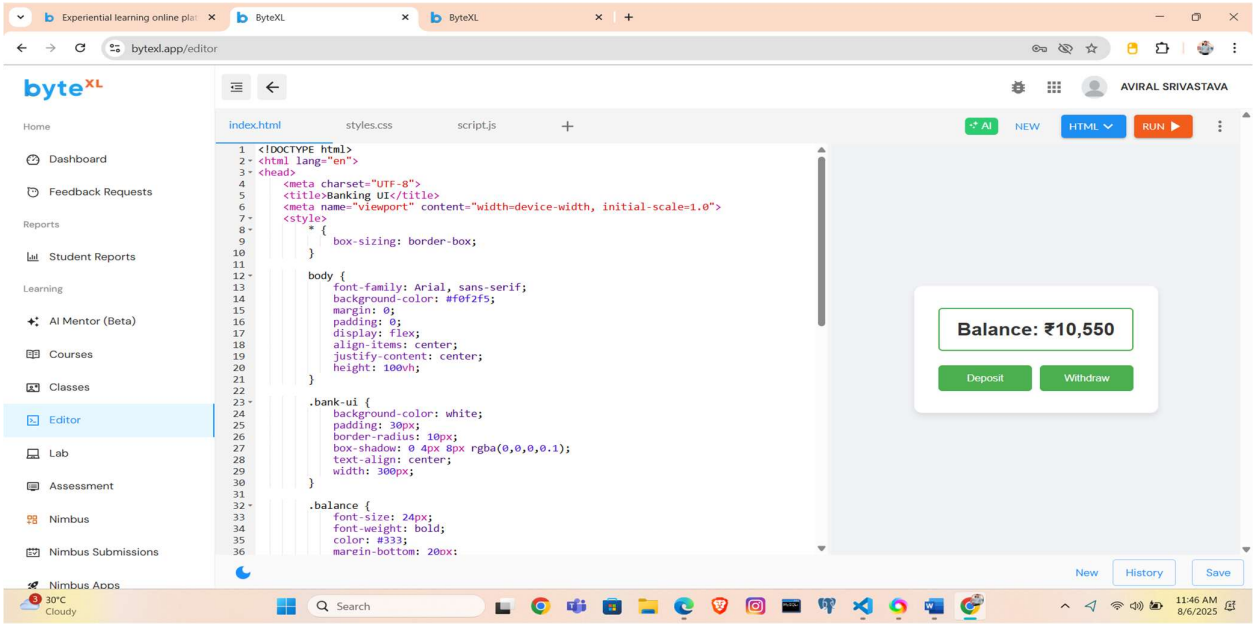
CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Banking UI</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        * {
            box-sizing: border-box;
        }
```

```
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f2f5;
            margin: 0;
            padding: 0;
            display: flex;
            align-items: center;
            justify-content: center;
            height: 100vh;
        }

        .bank-ui {
            background-color: white;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0,0,0,0.1);
            text-align: center;
            width: 300px;
        }

        .balance {
            font-size: 24px;
            font-weight: bold;
            color: #333;
            margin-bottom: 20px;
            padding: 15px;
            border: 2px solid #4CAF50;
            border-radius: 5px;
        }

        .button-group {
            display: flex;
            justify-content: space-between;
            gap: 10px;
        }

        button {
            padding: 10px;
            flex: 1;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: default;
        }

        @media (max-width: 500px) {
            .bank-ui {
                width: 90%;
            }

            .button-group {
```

```
                flex-direction: column;
            }
        }
    </style>
</head>
<body>

<div class="bank-ui">
    <div class="balance">Balance: ₹10,550</div>
    <div class="button-group">
        <button>Deposit</button>
        <button>Withdraw</button>
    </div>
</div>

</body>
</html>
```

OUTPUT



LEARNING OUTCOME

- Use Flexbox for layout and centering.
- Build responsive web interfaces using media queries.
- Style HTML elements professionally using modern CSS practices.
- Create clean, structured UI components without using JavaScript.

EXPERIMENT-3

AIM

To design and build a professional, responsive admin dashboard interface using HTML, CSS Grid layout, and implement a theme switching feature (light and dark modes).

OBJECTIVE

This practice focuses on advanced layout skills using CSS Grid and demonstrates the adaptability of a user interface through theme switching without reloading the page.

THEORY

CSS Grid Layout allows web developers to create complex responsive web design layouts more easily and consistently across browsers. Theming enables switching between light and dark modes, improving accessibility and user experience. JavaScript is used to toggle classes or attributes that change the dashboard's appearance dynamically.

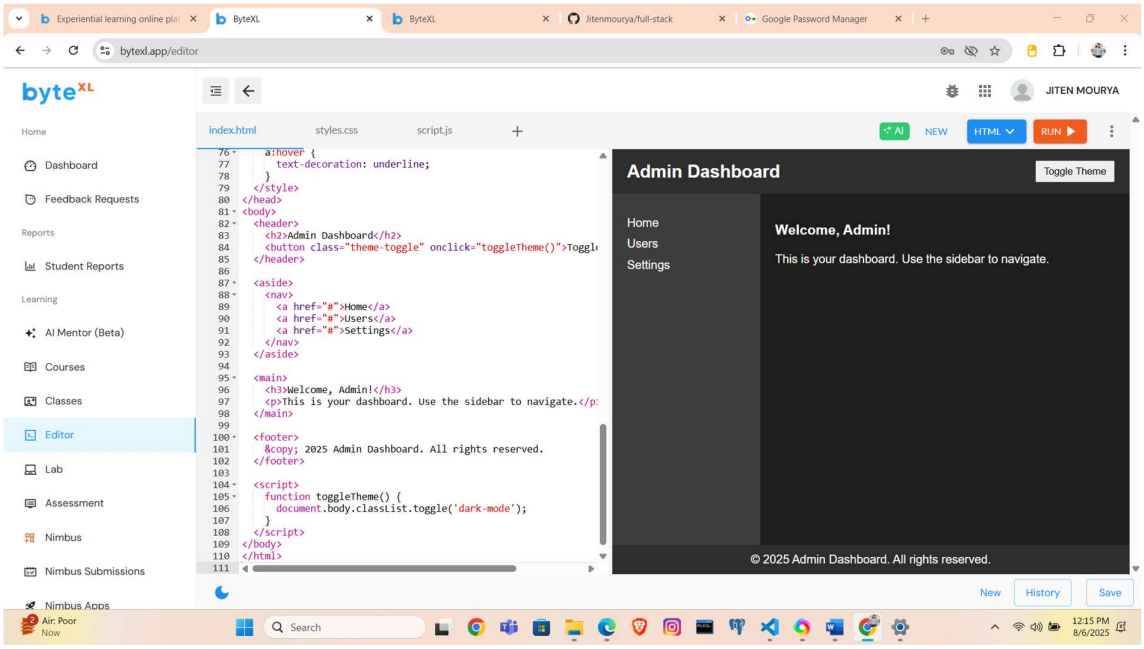
CODE

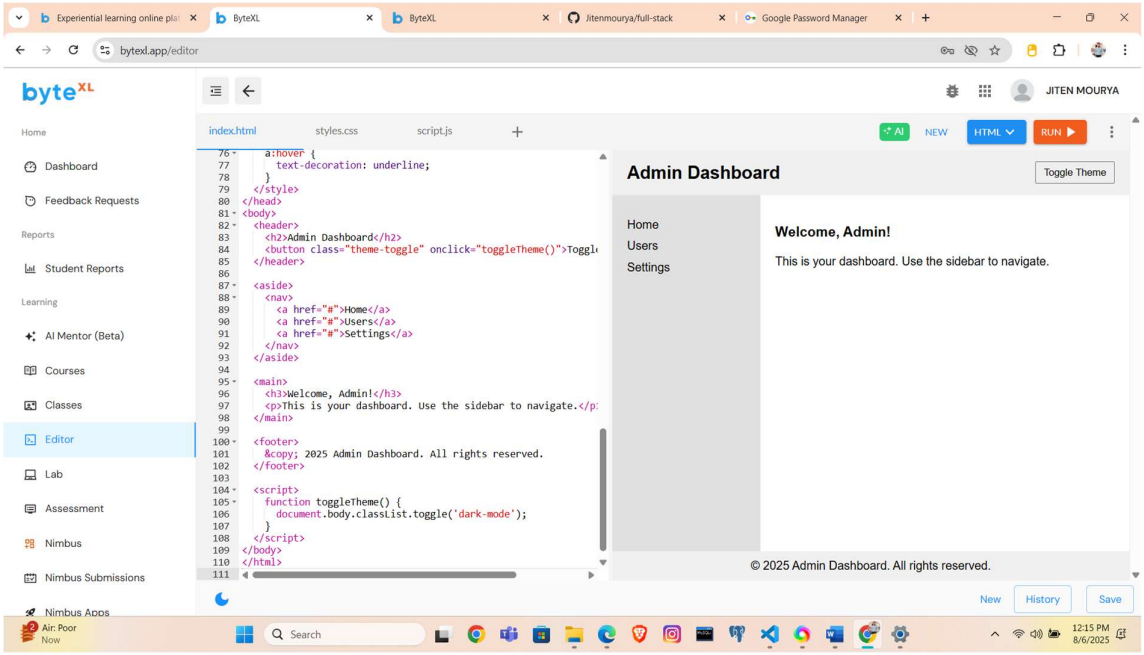
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Admin Dashboard</title>
  <style>
    :root {
      --bg-color: #ffffff;
      --text-color: #000000;
      --header-bg: #f0f0f0;
      --sidebar-bg: #e0e0e0;
    }

    .dark-mode {
      --bg-color: #1e1e1e;
      --text-color: #ffffff;
      --header-bg: #2e2e2e;
      --sidebar-bg: #3e3e3e;
    }

    body {
```

```
  margin: 0;
  font-family: Arial, sans-serif;
  background-color: var(--bg-color);
  color: var(--text-color);
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  grid-template-columns: 200px 1fr;
  grid-template-rows: 60px 1fr 40px;
  height: 100vh;
}

header {
  grid-area: header;
  background-color: var(--header-bg);
  padding: 10px 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

aside {
  grid-area: sidebar;
  background-color: var(--sidebar-bg);
  padding: 20px;
}

main {
  grid-area: main;
  padding: 20px;
}

footer {
  grid-area: footer;
  background-color: var(--header-bg);
  text-align: center;
  padding: 10px;
}

button.theme-toggle {
  padding: 5px 10px;
  cursor: pointer;
}

a {
  display: block;
  margin: 10px 0;
  color: var(--text-color);
  text-decoration: none;
}
```

```
    a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <header>
    <h2>Admin Dashboard</h2>
    <button class="theme-toggle" onclick="toggleTheme()">Toggle Theme</button>
  </header>

  <aside>
    <nav>
      <a href="#">Home</a>
      <a href="#">Users</a>
      <a href="#">Settings</a>
    </nav>
  </aside>

  <main>
    <h3>Welcome, Admin!</h3>
    <p>This is your dashboard. Use the sidebar to navigate.</p>
  </main>

  <footer>
    &copy; 2025 Admin Dashboard. All rights reserved.
  </footer>

  <script>
    function toggleTheme() {
      document.body.classList.toggle('dark-mode');
    }
  </script>
</body>
</html>
```

OUTPUT





LEARNING OUTCOME

After completing this practice, students will be able to: - Use CSS Grid to design a professional layout. - Implement responsive web components. - Add a dynamic theme switching feature using JavaScript. - Enhance user experience with adaptable design elements.