SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 1st semester

Web Application Development (PGI20D03J)

Lab Manual

Lab 1: Simple HTML Page Structure

Title: Creating a Basic HTML Page with Header, Footer, Navigation, and Main Content

Aim: To create a simple HTML page demonstrating the basic structure of a webpage, including a header, footer, navigation bar, and main content section with text, images, and links.

Procedure:

1. Open a text editor (e.g., VS Code, Sublime Text, Notepad++).
2. Create a new HTML file (e.g., `lab1.html`).
3. Add the basic HTML5 boilerplate (`<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`).
4. Inside the `<head>` section, add a `<title>` tag.
5. Inside the `<body>` section, create:
    - A `<header>` element for the page header.
    - A `<nav>` element for the navigation bar with some `<a>` tags for links.
    - A `<main>` element to contain the main content.
    - Inside `<main>`, add `<h2>` for a section heading, `<p>` tags for text, `<img>` tags for images (use placeholder images), and more `<a>` tags for links.
    - A `<footer>` element for the page footer.
6. Save the file and open it in a web browser to view the output.

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 1: Basic HTML Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            line-height: 1.6;
            color: #333;
        }
        header {
            background-color: #4CAF50;
            color: white;
            padding: 1rem 0;
            text-align: center;
        }
```

```
        nav {
            background-color: #333;
            overflow: hidden;
        }
        nav a {
            float: left;
            display: block;
            color: white;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
        }
        nav a:hover {
            background-color: #ddd;
            color: black;
        }
        main {
            padding: 20px;
            max-width: 960px;
            margin: 20px auto;
            background-color: #f4f4f4;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }
        footer {
            background-color: #333;
            color: white;
            text-align: center;
            padding: 1rem 0;
            position: relative;
            bottom: 0;
            width: 100%;
        }
        img {
            max-width: 100%;
            height: auto;
            display: block;
            margin: 15px 0;
            border-radius: 5px;
        }
    </style>
</head>
<body>
    <header>
        <h1>Welcome to My Webpage</h1>
    </header>

    <nav>
        <a href="#home">Home</a>
        <a href="#about">About</a>
        <a href="#services">Services</a>
        <a href="#contact">Contact</a>
    </nav>

    <main>
        <section id="home">
            <h2>About This Page</h2>
            <p>This is a simple HTML page created as part of Lab 1 for Web
Application Development. It demonstrates the fundamental structure of a
webpage.</p>
            <p>You will find a header at the top, a navigation bar, a main
content area, and a footer at the bottom.</p>
            <img
src="https://placehold.co/600x200/ADD8E6/000000?text=Placeholder+Image+1"
alt="Placeholder Image 1">
            <p>Click on the links below to navigate:</p>
            <ul>
```

```
                <li><a href="https://www.example.com" target="_blank">Visit
Example.com</a></li>
                <li><a href="https://www.google.com" target="_blank">Go to
Google</a></li>
            </ul>
        </section>

        <section id="about">
            <h2>More Information</h2>
            <p>This section can contain more detailed information about the
topic or purpose of the webpage.</p>
            <img
src="https://placehold.co/400x150/FFD700/000000?text=Placeholder+Image+2"
alt="Placeholder Image 2">
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</p>
        </section>
    </main>

    <footer>
        <p>&copy; 2025 Web Application Development Lab. All rights reserved.</p>
    </footer>
</body>
</html>
```

Input: No direct user input is required for this program. The content is static HTML.

Expected Output: A webpage displayed in the browser with:

- A green header displaying "Welcome to My Webpage".
- A black navigation bar with "Home", "About", "Services", and "Contact" links.
- A main content area with "About This Page" and "More Information" headings, paragraphs of text, two placeholder images, and two external links.
- A black footer displaying copyright information.
- The page elements should be visually distinct and well-arranged.

Lab 2: Tabular Data and Lists

Title: Building Tables and Experimenting with Lists and Link Styles

Aim: To display tabular data using HTML tables, experiment with different table attributes (`<thead>`, `<tbody>`, `<tfoot>`, `<th>`, `<td>`), and explore various types of lists (`<ul>`, `<ol>`, `<dl>`) and link styles.

Procedure:

1. Open a text editor and create a new HTML file (e.g., `lab2.html`).
2. Set up the basic HTML5 structure.
3. Inside the `<body>`, create a `<table>` element.
4. Within the table, define `<thead>`, `<tbody>`, and `<tfoot>` sections.
5. Populate `<thead>` with `<th>` (table header) elements for column titles.
6. Populate `<tbody>` with `<tr>` (table row) elements, each containing `<td>` (table data) elements.
7. Populate `<tfoot>` with summary `<td>` elements.
8. After the table, create examples of:
    - An unordered list (`<ul>`) with `<li>` items.
    - An ordered list (`<ol>`) with `<li>` items (experiment with `type` attribute).
    - A description list (`<dl>`) with `<dt>` (definition term) and `<dd>` (definition description) pairs.
9. Experiment with CSS to style the links (e.g., `text-decoration`, `color`, `hover` effects).
10. Save the file and view it in a browser.

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 2: Tables and Lists</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            line-height: 1.6;
            color: #333;
            background-color: #f9f9f9;
        }
        h1, h2 {
            color: #2c3e50;
        }
        table {
            width: 80%;
            border-collapse: collapse;
            margin: 20px 0;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
            background-color: white;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 12px;
            text-align: left;
```

```
        }
        th {
            background-color: #4CAF50;
            color: white;
            text-transform: uppercase;
        }
        tbody tr:nth-child(even) {
            background-color: #f2f2f2;
        }
        tbody tr:hover {
            background-color: #ddd;
        }
        tfoot {
            background-color: #e0e0e0;
            font-weight: bold;
        }
        ul, ol, dl {
            margin-bottom: 20px;
            padding-left: 25px;
        }
        ul {
            list-style-type: square;
        }
        ol {
            list-style-type: decimal;
        }
        dl {
            padding-left: 0;
        }
        dt {
            font-weight: bold;
            margin-top: 10px;
            color: #3498db;
        }
        dd {
            margin-left: 20px;
            margin-bottom: 5px;
        }
        a {
            color: #007bff;
            text-decoration: none;
            transition: color 0.3s ease;
        }
        a:hover {
            color: #0056b3;
            text-decoration: underline;
        }
        a.styled-link {
            background-color: #28a745;
            color: white;
            padding: 8px 15px;
            border-radius: 5px;
            text-decoration: none;
            display: inline-block;
            margin-top: 10px;
            transition: background-color 0.3s ease;
        }
        a.styled-link:hover {
            background-color: #218838;
            text-decoration: none;
        }
    </style>
</head>
<body>
    <h1>Lab 2: Tables and Lists</h1>

    <h2>Student Grades</h2>
```

```
    <table>
        <thead>
            <tr>
                <th>Student Name</th>
                <th>Subject</th>
                <th>Marks</th>
                <th>Grade</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Alice Smith</td>
                <td>Mathematics</td>
                <td>92</td>
                <td>A</td>
            </tr>
            <tr>
                <td>Bob Johnson</td>
                <td>Physics</td>
                <td>85</td>
                <td>B+</td>
            </tr>
            <tr>
                <td>Charlie Brown</td>
                <td>Chemistry</td>
                <td>78</td>
                <td>B</td>
            </tr>
            <tr>
                <td>Diana Prince</td>
                <td>Computer Science</td>
                <td>95</td>
                <td>A+</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td colspan="2">Total Students:</td>
                <td colspan="2">4</td>
            </tr>
        </tfoot>
    </table>

    <h2>Types of Lists</h2>

    <h3>Unordered List (Fruits)</h3>
    <ul>
        <li>Apple</li>
        <li>Banana</li>
        <li>Cherry</li>
        <li>Date</li>
    </ul>

    <h3>Ordered List (Steps to make coffee)</h3>
    <ol>
        <li>Boil water.</li>
        <li>Add coffee powder to a cup.</li>
        <li>Pour hot water into the cup.</li>
        <li>Add sugar and milk (optional).</li>
        <li>Stir and enjoy!</li>
    </ol>

    <h3>Description List (Web Technologies)</h3>
    <dl>
        <dt>HTML</dt>
        <dd>HyperText Markup Language, the standard markup language for
documents designed to be displayed in a web browser.</dd>
```

```
        <dt>CSS</dt>
        <dd>Cascading Style Sheets, a style sheet language used for describing
the presentation of a document written in HTML.</dd>

        <dt>JavaScript</dt>
        <dd>A programming language that is one of the core technologies of the
World Wide Web, alongside HTML and CSS.</dd>
    </dl>

    <h2>Link Styles</h2>
    <p>
        Here is a <a href="https://www.w3.org/" target="_blank">standard link to
W3C</a>.
    </p>
    <p>
        Here is a <a href="https://developer.mozilla.org/en-US/docs/Web/HTML"
class="styled-link" target="_blank">styled link to MDN HTML Docs</a>.
    </p>
</body>
</html>
```

Input: No direct user input is required. The program displays static data.

Expected Output: A webpage displaying:

- A well-formatted table with distinct header, body, and footer sections, showing student grades.
- An unordered list of fruits (e.g., with square bullet points).
- An ordered list of steps (e.g., numbered).
- A description list of web technologies with terms and their definitions.
- Two links, one with default browser styling and another with custom green background and white text.

Lab 3: Designing HTML Forms with Validation

Title: Designing HTML Forms and Implementing Basic Input Validation

Aim: To design a form using various HTML elements (`<form>`, `<input>`, `<select>`, `<textarea>`, `<button>`) and implement basic client-side validation for specific input formats (e.g., email, phone number).

Procedure:

1. Open a text editor and create a new HTML file (e.g., `lab3.html`).
2. Set up the basic HTML5 structure.
3. Inside the `<body>`, create a `<form>` element.
4. Add various input fields:
    - Text input for name (`<input type="text">`).
    - Email input for email address (`<input type="email">` with `required`).
    - Tel input for phone number (`<input type="tel">` with `pattern` and `title` for validation).
    - Number input for age (`<input type="number">`).
    - A dropdown (`<select>`) for a choice (e.g., country).
    - A multi-line text area (`<textarea>`) for comments.
    - A submit button (`<button type="submit">`).
5. Use `<label>` tags to associate labels with input fields for accessibility.
6. Utilize HTML5 attributes like `required`, `type="email"`, `type="tel"`, `pattern`, and `title` for built-in browser validation.
7. Add some basic CSS to style the form for better appearance.
8. Save the file and open it in a browser. Test the form by entering valid and invalid data to observe the validation messages.

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 3: HTML Forms with Validation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f7f6;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            margin: 0;
        }
        .form-container {
            background-color: white;
            padding: 30px 40px;
            border-radius: 10px;
            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 500px;
        }
        h1 {
```

```
            text-align: center;
            color: #333;
            margin-bottom: 30px;
        }
        .form-group {
            margin-bottom: 20px;
        }
        label {
            display: block;
            margin-bottom: 8px;
            color: #555;
            font-weight: bold;
        }
        input[type="text"],
        input[type="email"],
        input[type="tel"],
        input[type="number"],
        select,
        textarea {
            width: calc(100% - 22px); /* Account for padding and border */
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 16px;
            box-sizing: border-box; /* Include padding and border in the
element's total width and height */
        }
        textarea {
            resize: vertical; /* Allow vertical resizing */
            min-height: 80px;
        }
        input:focus,
        select:focus,
        textarea:focus {
            border-color: #4CAF50;
            outline: none;
            box-shadow: 0 0 5px rgba(76, 175, 80, 0.5);
        }
        button[type="submit"] {
            width: 100%;
            padding: 12px;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 5px;
            font-size: 18px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }
        button[type="submit"]:hover {
            background-color: #45a049;
        }
        /* Style for invalid input feedback */
        input:invalid:not(:placeholder-shown),
        input[type="email"]:invalid:not(:placeholder-shown),
        input[type="tel"]:invalid:not(:placeholder-shown) {
            border-color: #e74c3c; /* Red border for invalid */
            box-shadow: 0 0 5px rgba(231, 76, 60, 0.5);
        }
        input:invalid:not(:placeholder-shown) + .error-message {
            display: block;
            color: #e74c3c;
            font-size: 0.9em;
            margin-top: 5px;
        }
        .error-message {
            display: none; /* Hidden by default */
```

```
        }
    </style>
</head>
<body>
    <div class="form-container">
        <h1>User Registration Form</h1>
        <form action="#" method="post">
            <div class="form-group">
                <label for="name">Full Name:</label>
                <input type="text" id="name" name="name" required
placeholder="Enter your full name">
            </div>

            <div class="form-group">
                <label for="email">Email Address:</label>
                <input type="email" id="email" name="email" required
placeholder="name@example.com">
                <span class="error-message">Please enter a valid email
address.</span>
            </div>

            <div class="form-group">
                <label for="phone">Phone Number:</label>
                <input type="tel" id="phone" name="phone" pattern="[0-9]{10}"
title="Please enter a 10-digit phone number (e.g., 1234567890)" required
placeholder="e.g., 1234567890">
                <span class="error-message">Please enter a 10-digit phone
number.</span>
            </div>

            <div class="form-group">
                <label for="age">Age:</label>
                <input type="number" id="age" name="age" min="18" max="99"
required placeholder="Your age (18-99)">
            </div>

            <div class="form-group">
                <label for="country">Country:</label>
                <select id="country" name="country" required>
                    <option value="">--Select a country--</option>
                    <option value="usa">United States</option>
                    <option value="can">Canada</option>
                    <option value="uk">United Kingdom</option>
                    <option value="aus">Australia</option>
                    <option value="ind">India</option>
                </select>
            </div>

            <div class="form-group">
                <label for="comments">Comments:</label>
                <textarea id="comments" name="comments" rows="4"
placeholder="Any additional comments..."></textarea>
            </div>

            <button type="submit">Register</button>
        </form>
    </div>
</body>
</html>
```

Input: User will interact with the form by typing into input fields, selecting from dropdowns, and clicking the submit button.

- **Valid Input Example:**
    - Full Name: John Doe

    - Email Address: john.doe@example.com
    - Phone Number: 1234567890
    - Age: 30
    - Country: United States
    - Comments: This is a test comment.
- **Invalid Input Example (for email/phone validation):**
    - Email Address: invalid-email
    - Phone Number: 123 (less than 10 digits)

Expected Output: A visually appealing form with:

- Input fields for Full Name, Email Address, Phone Number, Age, Country, and Comments.
- The Email Address field will show a browser-level error message if an invalid email format is entered.
- The Phone Number field will show a browser-level error message if it's not a 10-digit number.
- The Age field will enforce a minimum of 18 and a maximum of 99.
- All required fields will trigger a browser validation message if left empty upon submission.
- Upon valid submission (though the `action="#"` means it won't go anywhere), the browser will typically indicate success or clear the form.

Lab 4: Applying CSS Styles to HTML Elements

Title: Styling HTML Elements with CSS Properties

Aim: To apply CSS styles to various HTML elements like headings, paragraphs, and links, experimenting with properties such as `color`, `font-family`, `font-size`, `text-align`, and `text-decoration`.

Procedure:

1. Open a text editor and create a new HTML file (e.g., `lab4.html`).
2. Set up the basic HTML5 structure.
3. Inside the `<head>` section, add a `<style>` tag for internal CSS.
4. Inside the `<body>`, add:
    - An `<h1>` heading.
    - A few `<p>` paragraphs.
    - Some `<a>` links.
    - A `<div>` or `<section>` to group elements and apply styles.
5. Within the `<style>` tag, write CSS rules to target these elements:
    - Apply `color` to headings and paragraphs.
    - Change `font-family` and `font-size` for various text elements.
    - Use `text-align` to center or justify text.
    - Modify `text-decoration` for links (e.g., remove underline, add underline on hover).
    - Experiment with `background-color`, `padding`, and `margin` for containers.
6. Save the file and open it in a browser to observe the applied styles.

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 4: Applying CSS Styles</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #eef2f7;
            color: #333;
            line-height: 1.6;
        }

        .container {
            max-width: 800px;
            margin: 30px auto;
            background-color: white;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
        }
```

```css
h1 {
    color: #2c3e50;
    font-size: 2.8em;
    text-align: center;
    margin-bottom: 25px;
    text-shadow: 1px 1px 2px rgba(0,0,0,0.1);
}

h2 {
    color: #3498db;
    font-size: 2em;
    margin-top: 30px;
    border-bottom: 2px solid #3498db;
    padding-bottom: 10px;
}

p {
    font-size: 1.1em;
    color: #555;
    margin-bottom: 15px;
    text-align: justify;
}

.highlight-text {
    color: #e74c3c;
    font-weight: bold;
    font-style: italic;
}

a {
    color: #27ae60;
    text-decoration: none;
    font-weight: bold;
    transition: color 0.3s ease, text-decoration 0.3s ease;
}

a:hover {
    color: #1e8449;
    text-decoration: underline;
}

.button-link {
    display: inline-block;
    background-color: #f39c12;
    color: white;
    padding: 10px 20px;
    border-radius: 5px;
    text-decoration: none;
    margin-top: 20px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.button-link:hover {
    background-color: #e67e22;
    transform: translateY(-2px);
    text-decoration: none;
}

/* Specific styling for a section */
.styled-section {
    background-color: #ecf0f1;
    padding: 20px;
    border-left: 5px solid #9b59b6;
    margin-top: 25px;
    border-radius: 5px;
}
```

```
    </style>
</head>
<body>
    <div class="container">
        <h1>Styling HTML Elements with CSS</h1>

        <p>This page demonstrates how to apply various CSS properties to
different HTML elements. We will explore properties like <span class="highlight-
text">color</span>, <span class="highlight-text">font-family</span>, <span
class="highlight-text">font-size</span>, <span class="highlight-text">text-
align</span>, and <span class="highlight-text">text-decoration</span>.</p>

        <h2>Paragraph Styling</h2>
        <p>This paragraph uses the default font-family 'Inter' and a font-size
of 1.1em. Its text is justified, providing a clean block-like appearance. The
text color is a soft grey for readability.</p>
        <p>Here is another paragraph to show more text. Notice how the line-
height property ensures good spacing between lines, making the content easy to
read. This is crucial for user experience.</p>

        <h2>Link Styling</h2>
        <p>
            Visit <a href="https://www.google.com" target="_blank">Google</a>
for search.
            <br>
            Explore <a href="https://developer.mozilla.org/" target="_blank">MDN
Web Docs</a> for comprehensive web development resources.
        </p>
        <p>
            You can also click this <a href="#" class="button-link">Styled
Button Link</a> to see a button-like link.
        </p>

        <div class="styled-section">
            <h2>Section with Custom Background</h2>
            <p>This section has a distinct background color and a left border,
demonstrating how to style container elements. The text within this section
maintains its own styling while inheriting some properties from the parent.</p>
            <p>CSS allows for incredible flexibility in design, enabling us to
create visually appealing and user-friendly interfaces.</p>
        </div>
    </div>
</body>
</html>
```

Input: No direct user input is required. The program displays static content with applied styles.

Expected Output: A webpage with:

- A centered, large, dark blue `<h1>` heading with a subtle text shadow.
- Paragraphs with a specific font, size, justified text alignment, and a grey color.
- A highlighted span of text within a paragraph in red, bold, and italic.
- `<h2>` headings in blue with an underline.
- Links in green, bold, and without an underline by default, turning darker green and underlined on hover.
- A button-like link with an orange background that changes on hover.
- A distinct section with a light grey background and a purple left border.

Lab 5: Responsive Web Design with CSS Layouts

Title: Designing Responsive Webpages with Media Queries and CSS Layouts

Aim: To design a webpage that adapts to different screen sizes using media queries and to create various layouts (fixed-width, fluid, responsive) using CSS techniques like floats, Flexbox, and CSS Grid.

Procedure:

1. Open a text editor and create a new HTML file (e.g., `lab5.html`).
2. Set up the basic HTML5 structure. Include the `viewport` meta tag in the `<head>`.
3. Inside the `<head>` section, add a `<style>` tag for internal CSS.
4. Inside the `<body>`, create several sections or `<div>` elements to demonstrate different layout techniques.
5. **Fixed-width Layout:** Create a container with a fixed `width` (e.g., `960px`) and `margin: auto` for centering.
6. **Fluid Layout:** Create a container with `width: 90%` or `max-width: 1200px` and `margin: auto`.
7. **Responsive Layout (using Flexbox):**
    - Create a `div` with `display: flex`.
    - Add child `div`s and use `flex-grow`, `flex-basis`, `flex-wrap` to make them responsive.
8. **Responsive Layout (using CSS Grid):**
    - Create a `div` with `display: grid`.
    - Use `grid-template-columns` with `fr` units or `repeat(auto-fit, minmax(250px, 1fr))` for responsive columns.
    - Add child `div`s as grid items.
9. **Media Queries:**
    - Define `@media` rules for different screen sizes (e.g., `max-width: 768px` for tablets, `max-width: 480px` for mobile).
    - Inside media queries, adjust layout properties (e.g., change `flex-direction`, `grid-template-columns`, `width`, `font-size`) to optimize for smaller screens.
10. Add some content (text, placeholder images) to each section to visualize the layout.
11. Save the file and open it in a browser. Resize the browser window to observe how the layout changes.

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 5: Responsive Web Design</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f0f2f5;
```

```
            color: #333;
            line-height: 1.6;
        }

        h1 {
            text-align: center;
            color: #2c3e50;
            margin-bottom: 40px;
            font-size: 2.5em;
        }

        section {
            background-color: white;
            padding: 30px;
            margin-bottom: 30px;
            border-radius: 10px;
            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.08);
        }

        h2 {
            color: #3498db;
            margin-top: 0;
            border-bottom: 2px solid #3498db;
            padding-bottom: 10px;
            margin-bottom: 20px;
        }

        p {
            margin-bottom: 15px;
        }

        .box {
            background-color: #ecf0f1;
            border: 1px solid #bdc3c7;
            padding: 20px;
            text-align: center;
            border-radius: 8px;
            margin: 10px;
        }

        /* Fixed-width Layout */
        .fixed-width-container {
            width: 960px; /* Fixed width */
            margin: 0 auto; /* Center the container */
            background-color: #d1e7dd;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }

        /* Fluid Layout */
        .fluid-container {
            width: 90%; /* Fluid width */
            max-width: 1200px; /* Max width to prevent too wide on large screens
*/
            margin: 0 auto;
            background-color: #f8d7da;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }

        /* Flexbox Layout */
        .flex-container {
            display: flex;
            flex-wrap: wrap; /* Allow items to wrap to the next line */
            justify-content: space-around;
```

```
            gap: 20px; /* Space between flex items */
        }

        .flex-item {
            flex: 1; /* Allow items to grow and shrink */
            min-width: 280px; /* Minimum width before wrapping */
            background-color: #e0f2f7;
            border: 1px solid #a7d9ed;
            padding: 15px;
            border-radius: 8px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.05);
            text-align: center;
        }

        /* CSS Grid Layout */
        .grid-container {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(280px, 1fr)); /*
Responsive columns */
            gap: 20px; /* Space between grid items */
        }

        .grid-item {
            background-color: #d4edda;
            border: 1px solid #8bd19d;
            padding: 15px;
            border-radius: 8px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.05);
            text-align: center;
        }

        img {
            max-width: 100%;
            height: auto;
            border-radius: 5px;
            margin-top: 10px;
        }

        /* Media Queries for Responsiveness */

        /* For screens smaller than 992px (e.g., tablets, smaller desktops) */
        @media (max-width: 992px) {
            .fixed-width-container {
                width: 90%; /* Make fixed-width container fluid on smaller
screens */
            }
        }

        /* For screens smaller than 768px (e.g., tablets in portrait, large
phones) */
        @media (max-width: 768px) {
            body {
                padding: 15px;
            }
            h1 {
                font-size: 2em;
            }
            section {
                padding: 20px;
            }
            .flex-item {
                min-width: 200px; /* Adjust min-width for smaller screens */
            }
            .grid-container {
                grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); /*
Adjust min-width for grid */
            }
```

```
        }

        /* For screens smaller than 480px (e.g., mobile phones) */
        @media (max-width: 480px) {
            body {
                padding: 10px;
            }
            h1 {
                font-size: 1.8em;
            }
            .flex-container {
                flex-direction: column; /* Stack flex items vertically on mobile
*/
            }
            .flex-item, .grid-item {
                margin: 0 0 15px 0; /* Adjust margins for stacked layout */
                width: auto; /* Full width */
            }
            .grid-container {
                grid-template-columns: 1fr; /* Single column on mobile */
            }
        }
    </style>
</head>
<body>
    <h1>Lab 5: Responsive Web Design</h1>

    <section>
        <h2>Fixed-Width Layout Example</h2>
        <div class="fixed-width-container">
            <p>This container has a fixed width of 960px on larger screens. As
the screen size reduces, it becomes fluid to adapt to smaller viewports due to
media queries.</p>
            <img
src="https://placehold.co/400x100/ADD8E6/000000?text=Fixed+Width" alt="Fixed
Width Example">
        </div>
    </section>

    <section>
        <h2>Fluid Layout Example</h2>
        <div class="fluid-container">
            <p>This container uses a fluid width (90%) and a maximum width to
ensure it scales nicely with the browser window, without becoming too wide on
very large screens.</p>
            <img
src="https://placehold.co/400x100/FFD700/000000?text=Fluid+Width" alt="Fluid
Width Example">
        </div>
    </section>

    <section>
        <h2>Responsive Layout with Flexbox</h2>
        <p>This section demonstrates a responsive layout using CSS Flexbox.
Items will wrap to the next line when there isn't enough horizontal space.</p>
        <div class="flex-container">
            <div class="flex-item">
                <h3>Flex Item 1</h3>
                <p>Content for flex item one.</p>
                <img
src="https://placehold.co/150x100/98FB98/000000?text=Flex+1" alt="Flex Item 1">
            </div>
            <div class="flex-item">
                <h3>Flex Item 2</h3>
                <p>Content for flex item two.</p>
                <img
src="https://placehold.co/150x100/87CEEB/000000?text=Flex+2" alt="Flex Item 2">
```

```
            </div>
            <div class="flex-item">
                <h3>Flex Item 3</h3>
                <p>Content for flex item three.</p>
                <img
src="https://placehold.co/150x100/F08080/000000?text=Flex+3" alt="Flex Item 3">
            </div>
        </div>
    </section>

    <section>
        <h2>Responsive Layout with CSS Grid</h2>
        <p>This section showcases a responsive layout using CSS Grid. Columns
will automatically adjust and stack based on available screen space.</p>
        <div class="grid-container">
            <div class="grid-item">
                <h3>Grid Item A</h3>
                <p>Content for grid item A.</p>
                <img
src="https://placehold.co/150x100/DDA0DD/000000?text=Grid+A" alt="Grid Item A">
            </div>
            <div class="grid-item">
                <h3>Grid Item B</h3>
                <p>Content for grid item B.</p>
                <img
src="https://placehold.co/150x100/FFB6C1/000000?text=Grid+B" alt="Grid Item B">
            </div>
            <div class="grid-item">
                <h3>Grid Item C</h3>
                <p>Content for grid item C.</p>
                <img
src="https://placehold.co/150x100/B0E0E6/000000?text=Grid+C" alt="Grid Item C">
            </div>
            <div class="grid-item">
                <h3>Grid Item D</h3>
                <p>Content for grid item D.</p>
                <img
src="https://placehold.co/150x100/C0C0C0/000000?text=Grid+D" alt="Grid Item D">
            </div>
        </div>
    </section>
</body>
</html>
```

Input: No direct user input is required. The responsiveness is observed by resizing the browser window.

Expected Output: A webpage that visually adapts to different screen sizes:

- On large screens, the fixed-width container will be 960px wide and centered. The fluid container will be 90% wide (max 1200px). Flexbox and Grid items will appear in multiple columns.
- On medium screens (e.g., tablet size), the fixed-width container will become fluid (90% width). Flexbox and Grid items might re-arrange into fewer columns or adjust their sizes.
- On small screens (e.g., mobile phone size), Flexbox items will stack vertically, and Grid items will arrange into a single column, ensuring content remains readable and accessible without horizontal scrolling.

Lab 6: BEM Methodology and Sass Features

Title: Implementing BEM Methodology and Converting CSS to Sass

Aim: To understand and implement the BEM (Block, Element, Modifier) methodology for structuring CSS classes and to convert existing CSS code to Sass, utilizing Sass features like variables, nesting, and mixins.

Procedure:

1. **Set up Sass:**
    - Install Node.js (if not already installed).
    - Install Sass globally: `npm install -g sass`.
    - Create a project folder.
2. **Create HTML:**
    - Create an `index.html` file with a simple structure that will use BEM-styled classes. For example, a card component, a button, etc.
3. **Write Sass with BEM:**
    - Create a `styles.scss` file.
    - Define Sass variables for colors, fonts, etc.
    - Implement BEM naming conventions for your CSS classes (e.g., `.block`, `.block__element`, `.block--modifier`).
    - Use Sass nesting to organize your CSS rules, following the BEM structure.
    - (Optional but recommended) Create a simple mixin for common styles (e.g., button styling).
4. **Compile Sass:**
    - Open your terminal in the project folder.
    - Compile the Sass file to CSS: `sass styles.scss:styles.css`.
    - Set up a watch command for automatic compilation during development: `sass --watch styles.scss:styles.css`.
5. **Link CSS to HTML:**
    - In `index.html`, link the compiled `styles.css` file.
6. Open `index.html` in a browser to see the styled components. Observe the structured and maintainable CSS output.

Source Code (styles.scss):

```
/* Lab 6: BEM Methodology and Sass Features */

// 1. Sass Variables
$primary-color: #3498db;
$secondary-color: #2ecc71;
$text-color: #333;
$background-color: #f8f8f8;
$border-color: #ddd;
$font-stack: 'Inter', Arial, sans-serif;
$spacing-unit: 10px;

// 2. Sass Mixin for Button Styling
@mixin button-styles($bg-color, $text-color: white) {
    display: inline-block;
    padding: $spacing-unit * 1.2 $spacing-unit * 2;
    border: none;
    border-radius: 5px;
    background-color: $bg-color;
```

```
    color: $text-color;
    font-size: 1em;
    cursor: pointer;
    text-decoration: none;
    transition: background-color 0.3s ease, transform 0.2s ease;

    &:hover {
        background-color: darken($bg-color, 10%);
        transform: translateY(-2px);
    }

    &:active {
        transform: translateY(0);
    }
}

body {
    font-family: $font-stack;
    margin: 0;
    padding: 20px;
    background-color: $background-color;
    color: $text-color;
    line-height: 1.6;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    flex-direction: column;
}

h1 {
    color: $primary-color;
    margin-bottom: $spacing-unit * 3;
}

// BEM Block: .card
.card {
    background-color: white;
    border: 1px solid $border-color;
    border-radius: 8px;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.08);
    padding: $spacing-unit * 3;
    margin: $spacing-unit * 2;
    max-width: 400px;
    width: 100%;
    text-align: center;

    // BEM Element: .card__header
    &__header {
        font-size: 1.8em;
        color: $primary-color;
        margin-bottom: $spacing-unit * 1.5;
    }

    // BEM Element: .card__body
    &__body {
        font-size: 1.1em;
        color: $text-color;
        margin-bottom: $spacing-unit * 2;
    }

    // BEM Element: .card__button-group
    &__button-group {
        display: flex;
        justify-content: center;
        gap: $spacing-unit; // Space between buttons
    }
```

```
    // BEM Element: .card__button
    &__button {
        @include button-styles($secondary-color); // Using the mixin
    }

    // BEM Modifier: .card--featured
    &--featured {
        border-color: $primary-color;
        box-shadow: 0 6px 20px rgba($primary-color, 0.2);
        background-color: lighten($primary-color, 45%); // Lighter shade of
primary color
    }
}

// BEM Block: .button (standalone block, also used within .card)
.button {
    @include button-styles($primary-color);

    // BEM Modifier: .button--secondary
    &--secondary {
        @include button-styles(#e74c3c); // Override with a different color
    }
}
```

Source Code (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 6: BEM and Sass</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <link rel="stylesheet" href="styles.css"> </head>
<body>
    <h1>BEM Methodology and Sass Features</h1>

    <div class="card">
        <h2 class="card__header">Standard Card</h2>
        <p class="card__body">This is a standard card component demonstrating
BEM naming conventions and Sass styling.</p>
        <div class="card__button-group">
            <a href="#" class="card__button">Learn More</a>
            <a href="#" class="card__button button--secondary">Dismiss</a>
        </div>
    </div>

    <div class="card card--featured">
        <h2 class="card__header">Featured Card</h2>
        <p class="card__body">This card uses a modifier class to show a featured
style, showcasing how BEM modifiers work.</p>
        <div class="card__button-group">
            <a href="#" class="card__button">View Details</a>
            <a href="#" class="card__button button--secondary">Close</a>
        </div>
    </div>

    <p>Standalone buttons:</p>
    <a href="#" class="button">Primary Button</a>
    <a href="#" class="button button--secondary">Secondary Button</a>

</body>
```

```
</html>
```

Input:

- **Sass Compilation:** Run `sass --watch styles.scss:styles.css` in your terminal.
- No direct user input for the HTML page itself.

Expected Output:

- A `styles.css` file will be generated in the same directory as `styles.scss`, containing the compiled CSS.
- The `index.html` page, when opened in a browser, will display:
    - Two distinct card components.
    - One "Standard Card" with a light background and a green button.
    - One "Featured Card" with a slightly tinted background and a blue border, also with a green button.
    - Two standalone buttons, one blue and one red.
    - All elements should be styled according to the Sass variables, nesting, and mixin definitions, demonstrating clean, modular, and maintainable CSS.

Lab 7: XML Document Creation

Title: Creating a Simple XML Document

Aim: To create a well-formed XML document with a root element and several child elements, demonstrating the basic syntax and structure of XML.

Procedure:

1. Open a text editor.
2. Create a new file and save it with a `.xml` extension (e.g., `books.xml`).
3. Start with the XML declaration: `<?xml version="1.0" encoding="UTF-8"?>`.
4. Define a single root element (e.g., `<library>`).
5. Inside the root element, add multiple child elements (e.g., `<book>`).
6. Each child element can have its own nested elements (e.g., `<title>`, `<author>`, `<year>`).
7. Ensure all tags are properly closed and the XML is well-formed.
8. Save the file. You can open it in a web browser (which will typically display it as a collapsible tree structure) or an XML editor to verify its structure.

Source Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="bk001">
        <title>The Great Adventure</title>
        <author>Jane Doe</author>
        <year>2020</year>
        <genre>Fantasy</genre>
        <price>25.99</price>
    </book>

    <book id="bk002">
        <title>Introduction to XML</title>
        <author>John Smith</author>
        <year>2018</year>
        <genre>Technical</genre>
        <price>35.50</price>
    </book>

    <book id="bk003">
        <title>Cooking for Beginners</title>
        <author>Emily White</author>
        <year>2022</year>
        <genre>Cookbook</genre>
        <price>19.95</price>
    </book>

    <book id="bk004">
        <title>The Mystery of the Old House</title>
        <author>David Green</author>
        <year>2019</year>
        <genre>Mystery</genre>
        <price>22.00</price>
    </book>
</library>
```

Input: No user input is required. The program is the XML file itself.

Expected Output: When opened in a web browser or XML editor, the XML document will be displayed as a hierarchical tree structure, showing the `<library>` root element containing multiple `<book>` elements, each with its own `<title>`, `<author>`, `<year>`, `<genre>`, and `<price>` child elements. The browser might allow collapsing/expanding nodes.

Lab 8: XML Document for Products and XSLT Transformation

Title: Creating a Product XML Document and XSLT Transformation to HTML Table

Aim: To create an XML document representing a collection of products and to write an XSLT stylesheet to transform an XML document (e.g., a list of students) into an HTML table for display on a webpage.

Procedure:

1. **Create Products XML:**
    - Create `products.xml` with a root element (e.g., `<products>`) and multiple `<product>` elements, each having `<name>`, `<price>`, `<category>`, and `<quantity>` child elements.
2. **Create Students XML:**
    - Create `students.xml` with a root element (e.g., `<students>`) and multiple `<student>` elements, each having `<name>`, `<age>`, and `<grade>` child elements.
3. **Create XSLT Stylesheet:**
    - Create `students_to_html.xslt` file.
    - Define the XSLT stylesheet with `<xsl:stylesheet>` and `<xsl:template match="/">`.
    - Inside the template, construct the HTML structure for a table (`<table>`, `<thead>`, `<tbody>`, `<tr>`, `<th>`, `<td>`).
    - Use `<xsl:for-each select="students/student">` to iterate over each student.
    - Use `<xsl:value-of select="name"/>`, `<xsl:value-of select="age"/>`, etc., to extract data from the XML and place it into the HTML table cells.
4. **Link XSLT to XML:**
    - Add the `<?xml-stylesheet type="text/xsl" href="students_to_html.xslt"?>` processing instruction to the `students.xml` file, right after the XML declaration.
5. Save all files. Open `students.xml` in a web browser. The browser will automatically apply the XSLT transformation and display the HTML table.

Source Code (products.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
    <product id="P001">
        <name>Laptop Pro</name>
        <price>1200.00</price>
        <category>Electronics</category>
        <quantity>50</quantity>
    </product>
    <product id="P002">
        <name>Wireless Mouse</name>
        <price>25.50</price>
        <category>Accessories</category>
        <quantity>200</quantity>
    </product>
    <product id="P003">
        <name>Mechanical Keyboard</name>
        <price>80.00</price>
        <category>Accessories</category>
```

```
        <quantity>75</quantity>
    </product>
    <product id="P004">
        <name>Monitor 27-inch</name>
        <price>350.00</price>
        <category>Electronics</category>
        <quantity>30</quantity>
    </product>
</products>
```

Source Code (students.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="students_to_html.xslt"?>
<students>
    <student id="S001">
        <name>Alice Wonderland</name>
        <age>20</age>
        <grade>A</grade>
    </student>
    <student id="S002">
        <name>Bob The Builder</name>
        <age>21</age>
        <grade>B+</grade>
    </student>
    <student id="S003">
        <name>Charlie Chaplin</name>
        <age>19</age>
        <grade>B</grade>
    </student>
    <student id="S004">
        <name>Diana Ross</name>
        <age>22</age>
        <grade>A-</grade>
    </student>
</students>
```

Source Code (students_to_html.xslt):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
  <html>
  <head>
    <title>Student List</title>
    <style>
      body { font-family: Arial, sans-serif; margin: 20px; background-color:
#f4f7f6; }
      h1 { color: #2c3e50; text-align: center; }
      table {
        width: 80%;
        border-collapse: collapse;
        margin: 25px auto;
        box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
        background-color: white;
        border-radius: 8px;
        overflow: hidden;
      }
      th, td {
        border: 1px solid #ddd;
        padding: 12px 15px;
        text-align: left;
```

```
      }
      th {
        background-color: #4CAF50;
        color: white;
        text-transform: uppercase;
        font-size: 0.9em;
      }
      tr:nth-child(even) {
        background-color: #f2f2f2;
      }
      tr:hover {
        background-color: #e9e9e9;
      }
    </style>
  </head>
  <body>
    <h1>Student Grades</h1>
    <table>
      <thead>
        <tr>
          <th>Student ID</th>
          <th>Name</th>
          <th>Age</th>
          <th>Grade</th>
        </tr>
      </thead>
      <tbody>
        <xsl:for-each select="students/student">
          <tr>
            <td><xsl:value-of select="@id"/></td>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="age"/></td>
            <td><xsl:value-of select="grade"/></td>
          </tr>
        </xsl:for-each>
      </tbody>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Input:

- `products.xml` (static content).
- `students.xml` (static content, linked to XSLT).
- `students_to_html.xslt` (static content).

Expected Output:

- When `products.xml` is opened, it will display the raw XML structure of products in the browser.
- When `students.xml` is opened, the browser will apply `students_to_html.xslt` and display an HTML table containing the student data (ID, Name, Age, Grade) in a well-formatted, styled table.

Lab 9: XQuery Transformation and Customer Profile XML

Title: XQuery Transformation for Movie Titles and Customer Profile XML Document

Aim: To write an XQuery to transform an XML document representing a list of movies into a new XML document containing only movie titles and release years, and to develop an XML document representing a customer profile.

Procedure:

1. **Create Movies XML:**
    - Create `movies.xml` with a root element (e.g., `<movies>`) and multiple `<movie>` elements, each having `<title>`, `<director>`, `<year>`, and `<genre>` child elements.
2. **Write XQuery:**
    - Create `movie_titles_years.xqy` file.
    - Write an XQuery expression to:
        - Load the `movies.xml` document.
        - Iterate through each `<movie>` element.
        - Construct a new XML structure (e.g., `<movie_list>`) containing `<movie_info>` elements.
        - For each `<movie_info>`, extract the `<title>` and `<year>` from the original movie element.
3. **Execute XQuery:**
    - You will typically need an XQuery processor (e.g., Saxon-HE, BaseX, or an online XQuery tester) to execute this. For the purpose of this manual, the output will be shown directly.
4. **Create Customer Profile XML:**
    - Create `customer_profile.xml` with a root element (e.g., `<customer>`) and elements like `<name>`, `<email>`, `<address>` (with nested elements like `<street>`, `<city>`, `<zip>`), and `<phone_number>`.
5. Save all files.

Source Code (movies.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
    <movie id="M001">
        <title>The Sci-Fi Odyssey</title>
        <director>Ava Director</director>
        <year>2010</year>
        <genre>Science Fiction</genre>
    </movie>
    <movie id="M002">
        <title>Romantic Comedy Nights</title>
        <director>Ben Writer</director>
        <year>2015</year>
        <genre>Comedy</genre>
    </movie>
    <movie id="M003">
        <title>Historical Drama</title>
        <director>Chris Vision</director>
        <year>2018</year>
        <genre>Drama</genre>
    </movie>
    <movie id="M004">
```

```
        <title>Animated Adventures</title>
        <director>Dana Animator</director>
        <year>2022</year>
        <genre>Animation</genre>
    </movie>
</movies>
```

Source Code (movie_titles_years.xqy):

```
(: Lab 9: XQuery to transform movies.xml to movie titles and years :)

<movie_list>
{
    for $movie in doc("movies.xml")/movies/movie
    return
        <movie_info>
            <title>{ $movie/title/text() }</title>
            <release_year>{ $movie/year/text() }</release_year>
        </movie_info>
}
</movie_list>
```

Source Code (customer_profile.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<customer id="CUST001">
    <name>
        <first_name>Alice</first_name>
        <last_name>Johnson</last_name>
    </name>
    <email>alice.johnson@example.com</email>
    <address>
        <street>123 Main St</street>
        <city>Anytown</city>
        <state>CA</state>
        <zip_code>90210</zip_code>
        <country>USA</country>
    </address>
    <phone_numbers>
        <phone type="mobile">123-456-7890</phone>
        <phone type="home">987-654-3210</phone>
    </phone_numbers>
</customer>
```

Input:

- `movies.xml` (static content for XQuery).
- `customer_profile.xml` (static content).
- `movie_titles_years.xqy` (the XQuery script).

Expected Output:

- **For `movie_titles_years.xqy` execution (requires an XQuery processor):** A new XML document structured as follows:
- `<movie_list>`
- `    <movie_info>`
- `        <title>The Sci-Fi Odyssey</title>`
- `        <release_year>2010</release_year>`
- `    </movie_info>`

```
    <movie_info>
        <title>Romantic Comedy Nights</title>
        <release_year>2015</release_year>
    </movie_info>
    <movie_info>
        <title>Historical Drama</title>
        <release_year>2018</release_year>
    </movie_info>
    <movie_info>
        <title>Animated Adventures</title>
        <release_year>2022</release_year>
    </movie_info>
</movie_list>
```

- **For `customer_profile.xml`:** When opened in a web browser or XML editor, it will display the raw XML structure of the customer profile, showing nested elements for name, address, and multiple phone numbers.

Lab 10: PHP Server Info and Data Types

Title: Displaying PHP Configuration and Working with PHP Data Types

Aim: To create a simple PHP page that outputs the current server's PHP configuration using `phpinfo()` and to demonstrate the creation and usage of PHP variables of different data types (string, integer, float, boolean, array).

Procedure:

1. **Set up PHP Environment:** Ensure you have a web server with PHP installed (e.g., Apache with PHP, Nginx with PHP-FPM, or use a local development server like XAMPP/WAMP).
2. **Create `phpinfo.php`:**
    - Create a new file named `phpinfo.php` in your web server's document root (e.g., `htdocs` for Apache).
    - Add the basic PHP opening and closing tags (`<?php ... ?>`).
    - Inside the tags, simply call the `phpinfo()` function.
3. **Create `datatypes.php`:**
    - Create another file named `datatypes.php` in the same directory.
    - Declare variables of different data types:
        - A string variable.
        - An integer variable.
        - A float (or double) variable.
        - A boolean variable.
        - An array variable (both indexed and associative).
    - Use `echo` or `print_r` (for arrays) to display the values and their types (using `var_dump()` for detailed info).
4. Save both files.
5. Open your web browser and navigate to `http://localhost/phpinfo.php` and `http://localhost/datatypes.php` (adjust URL based on your server setup).

Source Code (phpinfo.php):

```
<?php
// Lab 10: Displaying PHP Configuration
// This script outputs detailed information about the PHP installation.
phpinfo();
?>
```

Source Code (datatypes.php):

```
<?php
// Lab 10: Working with PHP Data Types
// This script demonstrates different PHP data types and their usage.

// 1. String Data Type
$name = "John Doe";
$greeting = 'Hello, ' . $name . '!';
echo "<h2>String Data Type</h2>";
echo "Name: " . $name . "<br>";
echo "Greeting: " . $greeting . "<br>";
echo "Type of \$name: " . gettype($name) . "<br><br>";

// 2. Integer Data Type
```

```
$age = 30;
$year = 2025;
$sum = $age + $year;
echo "<h2>Integer Data Type</h2>";
echo "Age: " . $age . "<br>";
echo "Year: " . $year . "<br>";
echo "Sum of Age and Year: " . $sum . "<br>";
echo "Type of \$age: " . gettype($age) . "<br><br>";

// 3. Float (Double) Data Type
$price = 19.99;
$tax_rate = 0.05;
$total_price = $price * (1 + $tax_rate);
echo "<h2>Float Data Type</h2>";
echo "Price: " . $price . "<br>";
echo "Tax Rate: " . $tax_rate . "<br>";
echo "Total Price (with tax): " . sprintf("%.2f", $total_price) . "<br>"; //
Format to 2 decimal places
echo "Type of \$price: " . gettype($price) . "<br><br>";

// 4. Boolean Data Type
$is_admin = true;
$is_guest = false;
echo "<h2>Boolean Data Type</h2>";
echo "Is Admin: " . ($is_admin ? "Yes" : "No") . "<br>"; // Ternary operator to
display Yes/No
echo "Is Guest: " . ($is_guest ? "Yes" : "No") . "<br>";
echo "Type of \$is_admin: " . gettype($is_admin) . "<br><br>";

// 5. Array Data Type (Indexed Array)
$fruits = array("Apple", "Banana", "Cherry");
echo "<h2>Array Data Type (Indexed)</h2>";
echo "First fruit: " . $fruits[0] . "<br>";
echo "All fruits: ";
print_r($fruits); // print_r is good for arrays
echo "<br>";
echo "Type of \$fruits: " . gettype($fruits) . "<br><br>";

// 6. Array Data Type (Associative Array)
$person = [
    "first_name" => "Jane",
    "last_name" => "Smith",
    "city" => "New York"
];
echo "<h2>Array Data Type (Associative)</h2>";
echo "Person's first name: " . $person["first_name"] . "<br>";
echo "Person's city: " . $person["city"] . "<br>";
echo "Full person details: ";
var_dump($person); // var_dump provides more detailed info including type and
length
echo "<br>";
echo "Type of \$person: " . gettype($person) . "<br><br>";

// 7. Null Data Type
$null_var = null;
echo "<h2>Null Data Type</h2>";
echo "Value of \$null_var: " . var_export($null_var, true) . "<br>"; //
var_export for null
echo "Is \$null_var null? " . (is_null($null_var) ? "Yes" : "No") . "<br>";
echo "Type of \$null_var: " . gettype($null_var) . "<br><br>";
?>
```

Input: No direct user input is required for these PHP scripts.

Expected Output:

- **For `phpinfo.php`:** A comprehensive HTML page displaying all configuration details of the PHP installation on the server, including PHP version, build date, loaded modules, environment variables, etc.
- **For `datatypes.php`:** A webpage showing:
    - Headings for each data type.
    - The values of the declared variables.
    - The `gettype()` output for each variable, confirming its data type (e.g., "string", "integer", "double", "boolean", "array", "NULL").
    - For arrays, `print_r` or `var_dump` output showing the structure and contents of the arrays.

Lab 11: PHP Conditional Statements and Loops

Title: Implementing PHP Conditional Statements and Loops

Aim: To implement a PHP script to check if a number is even or odd using an `if-else` statement and to write a PHP program to print the multiplication table of a given number using a `for` loop.

Procedure:

1. **Set up PHP Environment:** Ensure your web server with PHP is running.
2. **Create `even_odd.php`:**
   - Create a new file named `even_odd.php`.
   - Define a PHP variable `$number` and assign an integer value to it.
   - Use an `if-else` statement to check if `$number` is divisible by 2 (using the modulo operator `%`).
   - `echo` a message indicating whether the number is even or odd.
   - (Optional) Add a form to allow user input for the number.
3. **Create `multiplication_table.php`:**
   - Create a new file named `multiplication_table.php`.
   - Define a PHP variable `$num_to_multiply` and assign an integer value.
   - Use a `for` loop to iterate from 1 to 10 (or any desired range).
   - Inside the loop, calculate the product of `$num_to_multiply` and the loop counter.
   - `echo` the multiplication expression and its result for each iteration.
   - (Optional) Add a form to allow user input for the number.
4. Save both files.
5. Open your web browser and navigate to `http://localhost/even_odd.php` and `http://localhost/multiplication_table.php`.

Source Code (even_odd.php):

```
<?php
// Lab 11: Check if a number is Even or Odd
// This script demonstrates the use of if-else statements.

$number = 15; // You can change this number to test different values

echo "<h1>Even or Odd Checker</h1>";
echo "<p>The number to check is: <strong>" . $number . "</strong></p>";

if ($number % 2 == 0) {
    echo "<p>Result: <strong>" . $number . "</strong> is an EVEN number.</p>";
} else {
    echo "<p>Result: <strong>" . $number . "</strong> is an ODD number.</p>";
}

// Optional: Add a form to allow user input
echo "<h2>Test with your own number:</h2>";
echo '<form method="GET" action="">
        <label for="input_number">Enter a number:</label>
        <input type="number" id="input_number" name="input_number" required>
        <button type="submit">Check</button>
      </form>';

if (isset($_GET['input_number']) && is_numeric($_GET['input_number'])) {
    $user_number = (int)$_GET['input_number'];
    echo "<p>Checking your number: <strong>" . $user_number . "</strong></p>";
```

```
    if ($user_number % 2 == 0) {
        echo "<p>Result: <strong>" . $user_number . "</strong> is an EVEN
number.</p>";
    } else {
        echo "<p>Result: <strong>" . $user_number . "</strong> is an ODD
number.</p>";
    }
} elseif (isset($_GET['input_number']) && !is_numeric($_GET['input_number'])) {
    echo "<p style='color: red;'>Please enter a valid number.</p>";
}
?>
```

Source Code (multiplication_table.php):

```
<?php
// Lab 11: Print Multiplication Table
// This script demonstrates the use of a for loop.

$num_to_multiply = 7; // You can change this number

echo "<h1>Multiplication Table for " . $num_to_multiply . "</h1>";
echo "<table border='1' cellpadding='10' cellspacing='0'>";
echo "<thead><tr><th>Expression</th><th>Result</th></tr></thead>";
echo "<tbody>";

for ($i = 1; $i <= 10; $i++) {
    $product = $num_to_multiply * $i;
    echo "<tr>";
    echo "<td>" . $num_to_multiply . " x " . $i . "</td>";
    echo "<td>" . $product . "</td>";
    echo "</tr>";
}

echo "</tbody>";
echo "</table>";

// Optional: Add a form to allow user input
echo "<h2>Generate table for your own number:</h2>";
echo '<form method="POST" action="">
        <label for="table_number">Enter a number:</label>
        <input type="number" id="table_number" name="table_number" required>
        <button type="submit">Generate Table</button>
      </form>';

if (isset($_POST['table_number']) && is_numeric($_POST['table_number'])) {
    $user_table_number = (int)$_POST['table_number'];
    echo "<h3>Multiplication Table for " . $user_table_number . "</h3>";
    echo "<table border='1' cellpadding='10' cellspacing='0'>";
    echo "<thead><tr><th>Expression</th><th>Result</th></tr></thead>";
    echo "<tbody>";
    for ($j = 1; $j <= 10; $j++) {
        $product_user = $user_table_number * $j;
        echo "<tr>";
        echo "<td>" . $user_table_number . " x " . $j . "</td>";
        echo "<td>" . $product_user . "</td>";
        echo "</tr>";
    }
    echo "</tbody>";
    echo "</table>";
} elseif (isset($_POST['table_number']) && !is_numeric($_POST['table_number']))
{
    echo "<p style='color: red;'>Please enter a valid number for the
table.</p>";
}
?>
```

Input:

- **For `even_odd.php`:**
  - Initial `$number` is `15`.
  - User can input a number via the form (e.g., `20` or `7`).
- **For `multiplication_table.php`:**
  - Initial `$num_to_multiply` is `7`.
  - User can input a number via the form (e.g., `9`).

Expected Output:

- **For `even_odd.php`:**
  - Initially, it will display: "The number to check is: 15. Result: 15 is an ODD number."
  - If the user enters `20` in the form and submits, it will display: "Checking your number: 20. Result: 20 is an EVEN number."
- **For `multiplication_table.php`:**
  - Initially, it will display a table showing the multiplication table for `7` (7x1=7, 7x2=14, ..., 7x10=70).
  - If the user enters `9` in the form and submits, it will display a new table showing the multiplication table for `9`.

Lab 12: PHP Includes and Form Handling

Title: Implementing PHP Includes for Page Structure and Basic Form Handling

Aim: To implement PHP includes to separate header, footer, and navigation sections in a webpage and to create a PHP form to accept user input for basic information (name, email, age) and display the entered data.

Procedure:

1. **Set up PHP Environment:** Ensure your web server with PHP is running.
2. **Create `header.php`:**
    - Create a file `header.php`.
    - Include the `<!DOCTYPE html>`, `<html>`, `<head>`, and opening `<body>` tags.
    - Add a simple header and navigation bar HTML.
3. **Create `footer.php`:**
    - Create a file `footer.php`.
    - Include the closing `</body>` and `</html>` tags.
    - Add a simple footer HTML.
4. **Create `index.php` (Main Page):**
    - Create `index.php`.
    - At the top, use `include 'header.php';`.
    - Add the main content of the page (e.g., a welcome message).
    - At the bottom, use `include 'footer.php';`.
5. **Create `register.php` (Form Handling):**
    - Create `register.php`.
    - Include `header.php` and `footer.php`.
    - Create an HTML form with input fields for name, email, and age. Set the form's `method` to `POST` and `action` to the same file (`register.php`).
    - At the top of `register.php` (before the HTML form), add PHP code to:
        - Check if the form has been submitted (`if ($_SERVER["REQUEST_METHOD"] == "POST")`).
        - Retrieve the submitted data using `$_POST`.
        - Sanitize and validate the input (basic checks).
        - Display the entered data back to the user.
6. Save all files.
7. Open your web browser and navigate to `http://localhost/index.php` and `http://localhost/register.php`.

Source Code (header.php):

```
<?php
// header.php - Contains the common header and navigation for the website
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web App Lab</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 0; padding: 0;
background-color: #f4f7f6; color: #333; }
```

```
        .container { max-width: 960px; margin: 20px auto; padding: 20px;
background-color: white; border-radius: 8px; box-shadow: 0 2px 10px
rgba(0,0,0,0.1); }
        header { background-color: #2c3e50; color: white; padding: 15px 0; text-
align: center; }
        header h1 { margin: 0; }
        nav { background-color: #34495e; overflow: hidden; }
        nav a { float: left; display: block; color: white; text-align: center;
padding: 14px 16px; text-decoration: none; }
        nav a:hover { background-color: #55708a; }
        .content { padding: 20px 0; }
        .form-group { margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; font-weight: bold; }
        input[type="text"], input[type="email"], input[type="number"] { width:
calc(100% - 22px); padding: 10px; border: 1px solid #ddd; border-radius: 4px; }
        button { background-color: #28a745; color: white; padding: 10px 20px;
border: none; border-radius: 5px; cursor: pointer; font-size: 1em; transition:
background-color 0.3s ease; }
        button:hover { background-color: #218838; }
        .data-display { background-color: #e9f5ee; border: 1px solid #d4edda;
padding: 15px; margin-top: 20px; border-radius: 5px; }
        .data-display p { margin: 5px 0; }
    </style>
</head>
<body>
    <header>
        <h1>My Awesome Website</h1>
    </header>
    <nav>
        <a href="index.php">Home</a>
        <a href="register.php">Register</a>
        <a href="#">About</a>
        <a href="#">Contact</a>
    </nav>
    <div class="container">
        <div class="content">
```

Source Code (footer.php):

```
<?php
// footer.php - Contains the common footer for the website
?>
        </div> </div> <footer>
        <p style="text-align: center; background-color: #2c3e50; color: white;
padding: 15px 0; margin-top: 20px;">
            &copy; <?php echo date("Y"); ?> Web App Lab. All rights reserved.
        </p>
    </footer>
</body>
</html>
```

Source Code (index.php):

```
<?php
// index.php - Main page using includes
include 'header.php';
?>

        <h2>Welcome to Our Homepage!</h2>
        <p>This is the main content of our website. It demonstrates how to
structure a webpage using PHP includes for common sections like the header and
footer.</p>
        <p>Navigate to the "Register" page to see a simple form handling
example.</p>
```

```
<?php
include 'footer.php';
?>
```

Source Code (register.php):

```
<?php
// register.php - Handles user registration form
include 'header.php';

$name = $email = $age = "";
$name_err = $email_err = $age_err = "";
$form_submitted = false;

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $form_submitted = true;

    // Validate Name
    if (empty(trim($_POST["name"]))) {
        $name_err = "Please enter your name.";
    } else {
        $name = htmlspecialchars(trim($_POST["name"]));
    }

    // Validate Email
    if (empty(trim($_POST["email"]))) {
        $email_err = "Please enter your email address.";
    } elseif (!filter_var(trim($_POST["email"]), FILTER_VALIDATE_EMAIL)) {
        $email_err = "Please enter a valid email format.";
    } else {
        $email = htmlspecialchars(trim($_POST["email"]));
    }

    // Validate Age
    if (empty(trim($_POST["age"]))) {
        $age_err = "Please enter your age.";
    } elseif (!filter_var(trim($_POST["age"]), FILTER_VALIDATE_INT,
array("options" => array("min_range"=>1, "max_range"=>120)))) {
        $age_err = "Please enter a valid age (1-120).";
    } else {
        $age = htmlspecialchars(trim($_POST["age"]));
    }

    // If no errors, display submitted data
    if (empty($name_err) && empty($email_err) && empty($age_err)) {
        echo "<div class='data-display'>";
        echo "<h2>Registration Successful!</h2>";
        echo "<p><strong>Name:</strong> " . $name . "</p>";
        echo "<p><strong>Email:</strong> " . $email . "</p>";
        echo "<p><strong>Age:</strong> " . $age . "</p>";
        echo "<p>Thank you for registering!</p>";
        echo "</div>";
    }
}
?>

        <h2>User Registration Form</h2>
        <?php if ($form_submitted && (!empty($name_err) || !empty($email_err) ||
!empty($age_err))): ?>
            <p style="color: red;">Please correct the errors below:</p>
        <?php endif; ?>

        <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="post">
```

```
            <div class="form-group">
                <label for="name">Name:</label>
                <input type="text" id="name" name="name" value="<?php echo
$name; ?>">
                <span style="color: red; font-size: 0.9em;"><?php echo
$name_err; ?></span>
            </div>
            <div class="form-group">
                <label for="email">Email:</label>
                <input type="email" id="email" name="email" value="<?php echo
$email; ?>">
                <span style="color: red; font-size: 0.9em;"><?php echo
$email_err; ?></span>
            </div>
            <div class="form-group">
                <label for="age">Age:</label>
                <input type="number" id="age" name="age" value="<?php echo $age;
?>">
                <span style="color: red; font-size: 0.9em;"><?php echo $age_err;
?></span>
            </div>
            <button type="submit">Submit</button>
        </form>

<?php
include 'footer.php';
?>
```

Input:

- **For `index.php`:** No direct input.
- **For `register.php`:** User inputs name, email, and age into the form fields.
    - **Valid Input Example:**
        - Name: Jane Doe
        - Email: jane.doe@example.com
        - Age: 25
    - **Invalid Input Example:**
        - Name: (empty)
        - Email: invalid-email
        - Age: abc

Expected Output:

- **For `index.php`:** A webpage displaying "Welcome to Our Homepage!" within the main content area, surrounded by the common header, navigation, and footer.
- **For `register.php`:**
    - Initially, it will display an empty registration form.
    - If valid data is submitted, a "Registration Successful!" message will appear above the form, displaying the submitted Name, Email, and Age.
    - If invalid data is submitted (e.g., empty fields, invalid email format, non-numeric age), error messages will appear next to the respective input fields, and the form will remain visible.

Lab 13: AJAX Form Submission

Title: Implementing AJAX for Asynchronous Form Data Submission

Aim: To create a form with input fields and use AJAX (Asynchronous JavaScript and XML) to send form data to a server-side script for processing, displaying the response without refreshing the entire page.

Procedure:

1. **Set up PHP Environment:** Ensure your web server with PHP is running.
2. **Create `process_form.php` (Server-side script):**
    - This PHP file will receive the AJAX request.
    - It should check if the request method is `POST`.
    - Retrieve the data sent from the client.
    - Perform some basic processing (e.g., validation, simple message generation).
    - Echo a JSON response back to the client (e.g., `{"status": "success", "message": "Data received!"}`).
3. **Create `index.html` (Client-side with AJAX):**
    - Create an HTML form with input fields (e.g., name, message).
    - Add a `<script>` tag for JavaScript.
    - Attach an event listener to the form's `submit` event.
    - Inside the event listener:
        - Prevent the default form submission (`event.preventDefault()`).
        - Collect form data using `FormData` or manually.
        - Use the `fetch` API (modern AJAX) to send a `POST` request to `process_form.php`.
        - Handle the promise: parse the JSON response.
        - Update a `div` on the HTML page with the server's response message.
        - Add a loading indicator and error handling.
4. Save both files.
5. Open `index.html` in your web browser. Fill the form, submit it, and observe the response updating on the page without a full refresh.

Source Code (process_form.php):

```
<?php
// Lab 13: Server-side script to process AJAX form data
header('Content-Type: application/json'); // Respond with JSON

$response = array();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Get raw POST data
    $input = file_get_contents('php://input');
    $data = json_decode($input, true); // Decode JSON string to associative
array

    if ($data === null) {
        // Fallback for non-JSON POST (e.g., FormData)
        $data = $_POST;
    }

    $name = isset($data['name']) ? htmlspecialchars(trim($data['name'])) : '';
```

```
    $message = isset($data['message']) ?
htmlspecialchars(trim($data['message'])) : '';

    if (empty($name) || empty($message)) {
        $response['status'] = 'error';
        $response['message'] = 'Name and message cannot be empty.';
    } else {
        // Simulate some processing delay
        sleep(1); // Wait for 1 second

        $response['status'] = 'success';
        $response['message'] = 'Hello, ' . $name . '! Your message "' . $message
. '" was received successfully!';
        // In a real application, you would save this data to a database, etc.
    }
} else {
    $response['status'] = 'error';
    $response['message'] = 'Invalid request method.';
}

echo json_encode($response);
?>
```

Source Code (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 13: AJAX Form Submission</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            background-color: #f0f2f5;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            margin: 0;
            padding: 20px;
        }
        .container {
            background-color: white;
            padding: 40px;
            border-radius: 10px;
            box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 500px;
            text-align: center;
        }
        h1 {
            color: #2c3e50;
            margin-bottom: 30px;
        }
        .form-group {
            margin-bottom: 20px;
            text-align: left;
        }
        label {
            display: block;
            margin-bottom: 8px;
```

```
    color: #555;
    font-weight: bold;
}
input[type="text"],
textarea {
    width: calc(100% - 22px);
    padding: 12px;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 1em;
    box-sizing: border-box;
    transition: border-color 0.3s ease;
}
input[type="text"]:focus,
textarea:focus {
    border-color: #3498db;
    outline: none;
    box-shadow: 0 0 5px rgba(52, 152, 219, 0.3);
}
textarea {
    resize: vertical;
    min-height: 100px;
}
button {
    background-color: #28a745;
    color: white;
    padding: 12px 25px;
    border: none;
    border-radius: 5px;
    font-size: 1.1em;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
}
button:hover {
    background-color: #218838;
    transform: translateY(-1px);
}
button:disabled {
    background-color: #cccccc;
    cursor: not-allowed;
}
#response-message {
    margin-top: 30px;
    padding: 15px;
    border-radius: 5px;
    font-weight: bold;
    display: none; /* Hidden by default */
}
#response-message.success {
    background-color: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
}
#response-message.error {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
.loading-spinner {
    display: none;
    border: 4px solid #f3f3f3;
    border-top: 4px solid #3498db;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    animation: spin 1s linear infinite;
    margin: 20px auto 0;
```

```
        }
        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Contact Us (AJAX Demo)</h1>
        <form id="contactForm">
            <div class="form-group">
                <label for="name">Your Name:</label>
                <input type="text" id="name" name="name" required
placeholder="Enter your name">
            </div>
            <div class="form-group">
                <label for="message">Your Message:</label>
                <textarea id="message" name="message" required placeholder="Type
your message here..."></textarea>
            </div>
            <button type="submit" id="submitBtn">Send Message</button>
        </form>

        <div id="loadingSpinner" class="loading-spinner"></div>
        <div id="response-message"></div>

        <script>
            document.getElementById('contactForm').addEventListener('submit',
async function(event) {
                event.preventDefault(); // Prevent default form submission

                const nameInput = document.getElementById('name');
                const messageInput = document.getElementById('message');
                const submitBtn = document.getElementById('submitBtn');
                const loadingSpinner =
document.getElementById('loadingSpinner');
                const responseMessageDiv = document.getElementById('response-
message');

                // Clear previous messages and hide
                responseMessageDiv.style.display = 'none';
                responseMessageDiv.className = '';
                responseMessageDiv.textContent = '';

                // Show loading spinner and disable button
                loadingSpinner.style.display = 'block';
                submitBtn.disabled = true;

                const formData = {
                    name: nameInput.value,
                    message: messageInput.value
                };

                try {
                    const response = await fetch('process_form.php', {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json' // Indicate
sending JSON
                        },
                        body: JSON.stringify(formData) // Send data as JSON
string
                    });

                    if (!response.ok) {
```

```
                        throw new Error(`HTTP error! status:
${response.status}`);
                    }

                    const result = await response.json(); // Parse JSON response

                    responseMessageDiv.textContent = result.message;
                    if (result.status === 'success') {
                        responseMessageDiv.classList.add('success');
                        // Optionally clear form fields on success
                        nameInput.value = '';
                        messageInput.value = '';
                    } else {
                        responseMessageDiv.classList.add('error');
                    }
                    responseMessageDiv.style.display = 'block';

                } catch (error) {
                    console.error('Error submitting form:', error);
                    responseMessageDiv.textContent = 'An error occurred. Please
try again.';
                    responseMessageDiv.classList.add('error');
                    responseMessageDiv.style.display = 'block';
                } finally {
                    // Hide loading spinner and enable button
                    loadingSpinner.style.display = 'none';
                    submitBtn.disabled = false;
                }
            });
        </script>
    </div>
</body>
</html>
```

Input: User inputs their name and a message into the form fields.

- **Valid Input Example:**
  - Your Name: Alice
  - Your Message: This is a test message.
- **Invalid Input Example (empty fields):**
  - Your Name: (empty)
  - Your Message: (empty)

Expected Output:

- A webpage with a "Contact Us" form.
- Upon clicking "Send Message":
  - A loading spinner will appear, and the button will be disabled.
  - After a short delay (simulated by `sleep(1)` in PHP), a message will appear below the form without the page refreshing.
  - If valid input: "Hello, [Your Name]! Your message "[Your Message]" was received successfully!" in a green success box.
  - If empty input: "Name and message cannot be empty." in a red error box.

Lab 14: AJAX with External APIs (GitHub & Currency Converter)

Title: Building Webpages with AJAX and External APIs (GitHub & Currency Conversion)

Aim: To build a webpage that retrieves user information from the GitHub API using AJAX and displays user details, and to develop a currency converter application that fetches exchange rates from a currency exchange API using AJAX, allowing dynamic conversion.

Procedure:

1. **GitHub User Info (Client-side only):**
    - Create `github_viewer.html`.
    - Add an input field for a GitHub username and a button to fetch data.
    - Add a `div` to display the results.
    - In JavaScript:
        - Attach an event listener to the button.
        - Get the username from the input.
        - Use `fetch` to make a `GET` request to `https://api.github.com/users/{username}`.
        - Handle the promise: parse the JSON response.
        - Extract `name`, `avatar_url`, `public_repos`, etc.
        - Dynamically update the `results` div with the fetched information, including an `<img>` tag for the profile picture and links to repositories.
        - Include loading indicators and error handling for non-existent users or API issues.
2. **Currency Converter (Client-side only):**
    - Create `currency_converter.html`.
    - Add input fields for amount, dropdowns for source and target currencies, and a `div` for the result.
    - In JavaScript:
        - Identify a free currency exchange API (e.g., ExchangeRate-API, Open Exchange Rates - often require API keys and have rate limits; for this example, we'll use a simplified mock API or a public one if available without strict keys for demonstration). **Note: Public APIs often require registration and API keys. For a simple lab, you might need to use a mock API or instruct students to get their own keys.** I will use a simplified mock API structure for demonstration.
        - Attach event listeners to currency changes or an explicit convert button.
        - Use `fetch` to get exchange rates.
        - Perform the conversion calculation.
        - Display the converted amount dynamically.
        - Include error handling.

Source Code (github_viewer.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 14: GitHub User Viewer</title>
```

```
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            background-color: #f0f2f5;
            display: flex;
            justify-content: center;
            align-items: flex-start; /* Align to top */
            min-height: 100vh;
            margin: 0;
            padding: 40px 20px;
        }
        .container {
            background-color: white;
            padding: 30px 40px;
            border-radius: 10px;
            box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 600px;
            text-align: center;
        }
        h1 {
            color: #2c3e50;
            margin-bottom: 30px;
        }
        .input-group {
            margin-bottom: 25px;
            display: flex;
            justify-content: center;
            align-items: center;
            gap: 10px;
        }
        input[type="text"] {
            flex-grow: 1;
            padding: 12px;
            border: 1px solid #ddd;
            border-radius: 5px;
            font-size: 1.05em;
            max-width: 300px;
        }
        button {
            background-color: #28a745;
            color: white;
            padding: 12px 20px;
            border: none;
            border-radius: 5px;
            font-size: 1em;
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.2s ease;
        }
        button:hover {
            background-color: #218838;
            transform: translateY(-1px);
        }
        button:disabled {
            background-color: #cccccc;
            cursor: not-allowed;
        }
        #user-profile {
            margin-top: 30px;
            text-align: left;
            border-top: 1px solid #eee;
            padding-top: 20px;
        }
        #user-profile img {
```

```
    width: 120px;
    height: 120px;
    border-radius: 50%;
    object-fit: cover;
    margin-bottom: 15px;
    border: 3px solid #3498db;
    box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}
#user-profile h2 {
    color: #3498db;
    margin-bottom: 10px;
}
#user-profile p {
    margin-bottom: 8px;
    color: #555;
}
#user-profile a {
    color: #27ae60;
    text-decoration: none;
    font-weight: bold;
}
#user-profile a:hover {
    text-decoration: underline;
}
.repo-list {
    list-style: none;
    padding: 0;
    margin-top: 15px;
}
.repo-list li {
    background-color: #ecf0f1;
    padding: 10px 15px;
    margin-bottom: 8px;
    border-radius: 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-size: 0.95em;
}
.repo-list li a {
    color: #2c3e50;
}
.repo-list li span {
    font-weight: bold;
    color: #3498db;
}
.message {
    margin-top: 20px;
    padding: 15px;
    border-radius: 5px;
    font-weight: bold;
}
.message.error {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
.loading-spinner {
    display: none;
    border: 4px solid #f3f3f3;
    border-top: 4px solid #3498db;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    animation: spin 1s linear infinite;
    margin: 20px auto;
}
```

```
        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>GitHub User Profile Viewer</h1>
        <div class="input-group">
            <input type="text" id="github-username" placeholder="Enter GitHub
username (e.g., octocat)">
            <button id="fetch-user-btn">Fetch User</button>
        </div>

        <div id="loading-spinner" class="loading-spinner"></div>
        <div id="message" class="message" style="display: none;"></div>
        <div id="user-profile" style="display: none;">
            </div>

        <script>
            document.getElementById('fetch-user-btn').addEventListener('click',
async () => {
                const username = document.getElementById('github-
username').value.trim();
                const userProfileDiv = document.getElementById('user-profile');
                const loadingSpinner = document.getElementById('loading-
spinner');
                const messageDiv = document.getElementById('message');
                const fetchBtn = document.getElementById('fetch-user-btn');

                // Clear previous results and messages
                userProfileDiv.innerHTML = '';
                userProfileDiv.style.display = 'none';
                messageDiv.style.display = 'none';
                messageDiv.className = 'message';
                messageDiv.textContent = '';

                if (!username) {
                    messageDiv.textContent = 'Please enter a GitHub username.';
                    messageDiv.classList.add('error');
                    messageDiv.style.display = 'block';
                    return;
                }

                // Show loading spinner and disable button
                loadingSpinner.style.display = 'block';
                fetchBtn.disabled = true;

                try {
                    const userResponse = await
fetch(`https://api.github.com/users/${username}`);
                    if (!userResponse.ok) {
                        if (userResponse.status === 404) {
                            throw new Error('User not found.');
                        }
                        throw new Error(`GitHub API error:
${userResponse.status}`);
                    }
                    const userData = await userResponse.json();

                    const reposResponse = await fetch(userData.repos_url);
                    if (!reposResponse.ok) {
                        throw new Error(`GitHub API error fetching repos:
${reposResponse.status}`);
                    }
                    const reposData = await reposResponse.json();
```

```
                    // Display user profile
                    userProfileDiv.innerHTML = `
                        <img src="${userData.avatar_url}"
alt="${userData.login}'s avatar">
                        <h2>${userData.name || userData.login}</h2>
                        <p><strong>Username:</strong> ${userData.login}</p>
                        <p><strong>Bio:</strong> ${userData.bio || 'N/A'}</p>
                        <p><strong>Followers:</strong> ${userData.followers}</p>
                        <p><strong>Following:</strong> ${userData.following}</p>
                        <p><strong>Public Repos:</strong>
${userData.public_repos}</p>
                        <p><strong>Profile URL:</strong> <a
href="${userData.html_url}" target="_blank">${userData.html_url}</a></p>
                        <h3>Public Repositories (${reposData.length})</h3>
                        <ul class="repo-list">
                            ${reposData.map(repo => `
                                <li>
                                    <a href="${repo.html_url}"
target="_blank">${repo.name}</a>
                                    <span>⭐ ${repo.stargazers_count}</span>
                                </li>
                            `).join('')}
                        </ul>
                    `;
                    userProfileDiv.style.display = 'block';

                } catch (error) {
                    console.error('Error fetching GitHub user data:', error);
                    messageDiv.textContent = `Error: ${error.message}`;
                    messageDiv.classList.add('error');
                    messageDiv.style.display = 'block';
                } finally {
                    loadingSpinner.style.display = 'none';
                    fetchBtn.disabled = false;
                }
            });
        </script>
    </div>
</body>
</html>
```

Source Code (currency_converter.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 14: Currency Converter</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            background-color: #f0f2f5;
            display: flex;
            justify-content: center;
            align-items: flex-start;
            min-height: 100vh;
            margin: 0;
            padding: 40px 20px;
        }
        .container {
```

```
    background-color: white;
    padding: 30px 40px;
    border-radius: 10px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 500px;
    text-align: center;
}
h1 {
    color: #2c3e50;
    margin-bottom: 30px;
}
.form-group {
    margin-bottom: 20px;
    text-align: left;
}
label {
    display: block;
    margin-bottom: 8px;
    color: #555;
    font-weight: bold;
}
input[type="number"],
select {
    width: calc(100% - 22px);
    padding: 12px;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 1em;
    box-sizing: border-box;
    transition: border-color 0.3s ease;
}
input[type="number"]:focus,
select:focus {
    border-color: #3498db;
    outline: none;
    box-shadow: 0 0 5px rgba(52, 152, 219, 0.3);
}
button {
    background-color: #28a745;
    color: white;
    padding: 12px 25px;
    border: none;
    border-radius: 5px;
    font-size: 1.1em;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    margin-top: 15px;
}
button:hover {
    background-color: #218838;
    transform: translateY(-1px);
}
button:disabled {
    background-color: #cccccc;
    cursor: not-allowed;
}
#conversion-result {
    margin-top: 30px;
    padding: 20px;
    background-color: #e9f5ee;
    border: 1px solid #d4edda;
    border-radius: 8px;
    font-size: 1.3em;
    font-weight: bold;
    color: #155724;
    display: none;
```

```
        }
        .message {
            margin-top: 20px;
            padding: 15px;
            border-radius: 5px;
            font-weight: bold;
            display: none;
        }
        .message.error {
            background-color: #f8d7da;
            color: #721c24;
            border: 1px solid #f5c6cb;
        }
        .loading-spinner {
            display: none;
            border: 4px solid #f3f3f3;
            border-top: 4px solid #3498db;
            border-radius: 50%;
            width: 30px;
            height: 30px;
            animation: spin 1s linear infinite;
            margin: 20px auto;
        }
        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Currency Converter</h1>
        <div class="form-group">
            <label for="amount">Amount:</label>
            <input type="number" id="amount" value="1" min="0.01" step="0.01"
required>
        </div>
        <div class="form-group">
            <label for="from-currency">From Currency:</label>
            <select id="from-currency">
                <option value="USD">USD - United States Dollar</option>
                <option value="EUR">EUR - Euro</option>
                <option value="GBP">GBP - British Pound</option>
                <option value="JPY">JPY - Japanese Yen</option>
                <option value="AUD">AUD - Australian Dollar</option>
                <option value="CAD">CAD - Canadian Dollar</option>
                <option value="CHF">CHF - Swiss Franc</option>
                <option value="CNY">CNY - Chinese Yuan</option>
                <option value="INR" selected>INR - Indian Rupee</option>
                <option value="BRL">BRL - Brazilian Real</option>
            </select>
        </div>
        <div class="form-group">
            <label for="to-currency">To Currency:</label>
            <select id="to-currency">
                <option value="USD">USD - United States Dollar</option>
                <option value="EUR">EUR - Euro</option>
                <option value="GBP">GBP - British Pound</option>
                <option value="JPY">JPY - Japanese Yen</option>
                <option value="AUD">AUD - Australian Dollar</option>
                <option value="CAD">CAD - Canadian Dollar</option>
                <option value="CHF">CHF - Swiss Franc</option>
                <option value="CNY">CNY - Chinese Yuan</option>
                <option value="INR">INR - Indian Rupee</option>
                <option value="BRL" selected>BRL - Brazilian Real</option>
            </select>
        </div>
```

```
        <button id="convert-btn">Convert</button>

        <div id="loading-spinner" class="loading-spinner"></div>
        <div id="message" class="message" style="display: none;"></div>
        <div id="conversion-result" style="display: none;"></div>

        <script>
            // Mock API for demonstration. In a real app, use a reliable
currency API.
            // Example: https://www.exchangerate-api.com/ (requires API key)
            // Or https://open.er-api.com/v6/latest/{base_currency} (free, but
check limits)
            const MOCK_EXCHANGE_RATES = {
                "USD": { "EUR": 0.92, "GBP": 0.79, "JPY": 156.90, "INR": 83.50,
"BRL": 5.10, "USD": 1 },
                "EUR": { "USD": 1.09, "GBP": 0.86, "JPY": 170.50, "INR": 90.70,
"BRL": 5.54, "EUR": 1 },
                "GBP": { "USD": 1.26, "EUR": 1.16, "JPY": 198.50, "INR": 105.70,
"BRL": 6.45, "GBP": 1 },
                "JPY": { "USD": 0.0064, "EUR": 0.0059, "GBP": 0.0050, "INR":
0.53, "BRL": 0.032, "JPY": 1 },
                "INR": { "USD": 0.012, "EUR": 0.011, "GBP": 0.0095, "JPY": 1.88,
"BRL": 0.061, "INR": 1 },
                "BRL": { "USD": 0.196, "EUR": 0.18, "GBP": 0.155, "JPY": 31.25,
"INR": 16.40, "BRL": 1 }
            };

            document.getElementById('convert-btn').addEventListener('click',
convertCurrency);
            // Optionally, convert on input/select change
            // document.getElementById('amount').addEventListener('input',
convertCurrency);
            // document.getElementById('from-
currency').addEventListener('change', convertCurrency);
            // document.getElementById('to-currency').addEventListener('change',
convertCurrency);

            async function convertCurrency() {
                const amount =
parseFloat(document.getElementById('amount').value);
                const fromCurrency = document.getElementById('from-
currency').value;
                const toCurrency = document.getElementById('to-currency').value;
                const resultDiv = document.getElementById('conversion-result');
                const loadingSpinner = document.getElementById('loading-
spinner');
                const messageDiv = document.getElementById('message');
                const convertBtn = document.getElementById('convert-btn');

                // Clear previous results and messages
                resultDiv.style.display = 'none';
                messageDiv.style.display = 'none';
                messageDiv.className = 'message';
                messageDiv.textContent = '';

                if (isNaN(amount) || amount <= 0) {
                    messageDiv.textContent = 'Please enter a valid amount
greater than zero.';
                    messageDiv.classList.add('error');
                    messageDiv.style.display = 'block';
                    return;
                }

                if (fromCurrency === toCurrency) {
                    resultDiv.textContent = `${amount.toFixed(2)}
${fromCurrency} = ${amount.toFixed(2)} ${toCurrency}`;
                    resultDiv.style.display = 'block';
```

```
                    return;
                }

                // Show loading spinner and disable button
                loadingSpinner.style.display = 'block';
                convertBtn.disabled = true;

                try {
                    // In a real application, you would fetch from a real API
like:
                    // const response = await fetch(`https://open.er-
api.com/v6/latest/${fromCurrency}`);
                    // if (!response.ok) {
                    //     throw new Error(`API error: ${response.status}`);
                    // }
                    // const data = await response.json();
                    // const rate = data.rates[toCurrency];

                    // Using mock data for demonstration
                    const rate =
MOCK_EXCHANGE_RATES[fromCurrency]?.[toCurrency];

                    if (rate === undefined) {
                        throw new Error(`Exchange rate for ${fromCurrency} to
${toCurrency} not available.`);
                    }

                    const convertedAmount = amount * rate;
                    resultDiv.textContent = `${amount.toFixed(2)}
${fromCurrency} = ${convertedAmount.toFixed(2)} ${toCurrency}`;
                    resultDiv.style.display = 'block';

                } catch (error) {
                    console.error('Error fetching exchange rates:', error);
                    messageDiv.textContent = `Error: ${error.message}`;
                    messageDiv.classList.add('error');
                    messageDiv.style.display = 'block';
                } finally {
                    loadingSpinner.style.display = 'none';
                    convertBtn.disabled = false;
                }
            }

            // Initial conversion on page load (optional)
            convertCurrency();
        </script>
    </div>
</body>
</html>
```

Input:

- **For `github_viewer.html`:** User enters a GitHub username (e.g., `octocat`, `google`).
- **For `currency_converter.html`:** User enters an amount, selects "From" and "To" currencies.

Expected Output:

- **For `github_viewer.html`:**
  - An input field for username and a "Fetch User" button.
  - Upon fetching a valid username, a profile card will appear with the user's avatar, name, username, bio, followers, following, public repos count, profile URL, and a list of their public repositories with star counts.

    - If the username is not found or an API error occurs, an error message will be displayed.
- **For `currency_converter.html`:**
    - Input for amount, dropdowns for "From" and "To" currencies, and a "Convert" button.
    - Upon conversion, a green box will appear displaying the converted amount (e.g., "10.00 USD = 9.20 EUR").
    - If an invalid amount is entered or a rate is unavailable, an error message will be displayed.

Lab 15: RESTful API (CRUD & JWT Authentication)

Title: Designing RESTful API with CRUD Operations and JWT Authentication

Aim: To design and implement a simple RESTful API for managing a list of products, including endpoints for CRUD (Create, Read, Update, Delete) operations using AJAX to interact with the API, and to extend the API to include user authentication using JWT (JSON Web Tokens), implementing AJAX-based login and registration functionality on a web page.

Procedure: This lab is quite extensive and typically involves a server-side language (like PHP, Node.js, Python Flask/Django) for the API and a client-side (HTML/JS) for interaction. For the purpose of this manual, I will provide a conceptual outline and a simplified client-side HTML/JS example that *mocks* the API interactions for the CRUD part and conceptually outlines JWT. A full, runnable JWT implementation would require a robust server-side setup beyond a single HTML file.

Conceptual Server-Side API (e.g., in PHP):

- **`api/products.php` (for CRUD):**
  - Handle `GET` requests: Return all products or a specific product by ID.
  - Handle `POST` requests: Create a new product.
  - Handle `PUT` requests: Update an existing product.
  - Handle `DELETE` requests: Delete a product by ID.
  - Products can be stored in a simple JSON file on the server for this lab, or a database for a more robust solution.
- **`api/auth.php` (for JWT):**
  - Handle `POST` to `/register`: Create a new user, hash password, store user.
  - Handle `POST` to `/login`: Authenticate user, generate JWT, return token.
  - Implement middleware/logic to validate JWT for protected routes (e.g., for `PUT`/`DELETE` on products).

Client-Side (HTML/JS):

1. **Create `index.html`:**
   - **Product Management Section:**
     - Form for creating/updating products (input fields for name, price, etc.).
     - List/table to display products with "Edit" and "Delete" buttons.
     - JavaScript to:
       - Fetch all products on page load (`GET /api/products.php`).
       - Handle form submission for Create/Update (`POST`/`PUT` to `/api/products.php`).
       - Handle "Delete" button clicks (`DELETE` to `/api/products.php`).
       - Update the product list dynamically.
   - **Authentication Section:**
     - Login form (username/password).
     - Registration form (username/password).
     - A display area for messages (login success/failure, registration status).
     - JavaScript to:
       - Handle login form submission (`POST` to `/api/auth.php/login`). Store the received JWT in `localStorage`.
       - Handle registration form submission (`POST` to `/api/auth.php/register`).

- Include the JWT in `Authorization: Bearer <token>` header for protected API calls (e.g., PUT/DELETE products).
- Implement a "Logout" button to clear the token.

Simplified Client-Side Source Code (HTML/JS - Mocking API):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab 15: RESTful API (Mock) & Auth</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Inter', sans-serif;
            background-color: #f0f2f5;
            margin: 0;
            padding: 40px 20px;
            display: flex;
            flex-direction: column;
            align-items: center;
            min-height: 100vh;
        }
        .container {
            background-color: white;
            padding: 30px 40px;
            border-radius: 10px;
            box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 800px;
            margin-bottom: 30px;
        }
        h1, h2 {
            color: #2c3e50;
            text-align: center;
            margin-bottom: 25px;
        }
        .form-group {
            margin-bottom: 15px;
        }
        label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
            color: #555;
        }
        input[type="text"],
        input[type="number"],
        input[type="email"],
        input[type="password"] {
            width: calc(100% - 22px);
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 5px;
            font-size: 1em;
            box-sizing: border-box;
        }
        button {
            background-color: #28a745;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 5px;
```

```
    font-size: 1em;
    cursor: pointer;
    transition: background-color 0.3s ease;
    margin-right: 10px;
}
button:hover {
    background-color: #218838;
}
button.delete-btn {
    background-color: #e74c3c;
}
button.delete-btn:hover {
    background-color: #c0392b;
}
button.edit-btn {
    background-color: #f39c12;
}
button.edit-btn:hover {
    background-color: #e67e22;
}
button:disabled {
    background-color: #cccccc;
    cursor: not-allowed;
}
.message {
    padding: 10px;
    margin-top: 15px;
    border-radius: 5px;
    font-weight: bold;
    display: none;
}
.message.success {
    background-color: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
}
.message.error {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}
th, td {
    border: 1px solid #ddd;
    padding: 10px;
    text-align: left;
}
th {
    background-color: #4CAF50;
    color: white;
}
tr:nth-child(even) {
    background-color: #f2f2f2;
}
.auth-section {
    display: flex;
    gap: 30px;
    justify-content: center;
    margin-top: 30px;
}
.auth-form {
    flex: 1;
    max-width: 350px;
```

```
            padding: 25px;
            border: 1px solid #eee;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.05);
        }
        #auth-status {
            text-align: center;
            margin-top: 20px;
            font-weight: bold;
            color: #3498db;
        }
        #auth-status.logged-in {
            color: #28a745;
        }
        #auth-status.logged-out {
            color: #e74c3c;
        }
        .hidden {
            display: none;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Product Management (Mock API)</h1>
        <div id="auth-status" class="logged-out">Not logged in.</div>
        <button id="logout-btn" class="hidden">Logout</button>

        <h2 id="product-form-title">Add New Product</h2>
        <form id="product-form">
            <input type="hidden" id="product-id">
            <div class="form-group">
                <label for="product-name">Product Name:</label>
                <input type="text" id="product-name" required>
            </div>
            <div class="form-group">
                <label for="product-price">Price:</label>
                <input type="number" id="product-price" step="0.01" min="0.01"
required>
            </div>
            <div class="form-group">
                <label for="product-category">Category:</label>
                <input type="text" id="product-category" required>
            </div>
            <div class="form-group">
                <label for="product-quantity">Quantity:</label>
                <input type="number" id="product-quantity" min="0" required>
            </div>
            <button type="submit" id="submit-product-btn">Add Product</button>
            <button type="button" id="cancel-edit-btn" class="hidden">Cancel
Edit</button>
            <div id="product-message" class="message"></div>
        </form>

        <h2>Product List</h2>
        <table id="product-table">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Category</th>
                    <th>Quantity</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody id="product-list">
```

```
                </tbody>
            </table>
        </div>

        <div class="container auth-section">
            <div class="auth-form">
                <h2>Login</h2>
                <form id="login-form">
                    <div class="form-group">
                        <label for="login-username">Username:</label>
                        <input type="text" id="login-username" required>
                    </div>
                    <div class="form-group">
                        <label for="login-password">Password:</label>
                        <input type="password" id="login-password" required>
                    </div>
                    <button type="submit">Login</button>
                    <div id="login-message" class="message"></div>
                </form>
            </div>

            <div class="auth-form">
                <h2>Register</h2>
                <form id="register-form">
                    <div class="form-group">
                        <label for="register-username">Username:</label>
                        <input type="text" id="register-username" required>
                    </div>
                    <div class="form-group">
                        <label for="register-password">Password:</label>
                        <input type="password" id="register-password" required>
                    </div>
                    <button type="submit">Register</button>
                    <div id="register-message" class="message"></div>
                </form>
            </div>
        </div>

        <script>
            // Mock API Base URL (replace with your actual backend API)
            const MOCK_API_BASE_URL = 'http://localhost/api'; // Example, will not
work without a backend

            // --- Mock Data and Functions for Demonstration ---
            let products = [
                { id: 1, name: 'Laptop', price: 1200.00, category: 'Electronics',
quantity: 15 },
                { id: 2, name: 'Mouse', price: 25.00, category: 'Accessories',
quantity: 100 },
                { id: 3, name: 'Keyboard', price: 75.00, category: 'Accessories',
quantity: 50 }
            ];
            let nextProductId = products.length > 0 ? Math.max(...products.map(p =>
p.id)) + 1 : 1;
            let authToken = localStorage.getItem('jwt_token'); // Simulate JWT
storage

            function getAuthHeader() {
                return authToken ? { 'Authorization': `Bearer ${authToken}` } : {};
            }

            function updateAuthStatus() {
                const authStatusDiv = document.getElementById('auth-status');
                const logoutBtn = document.getElementById('logout-btn');
                if (authToken) {
                    authStatusDiv.textContent = 'Logged in!';
                    authStatusDiv.classList.remove('logged-out');
```

```
                authStatusDiv.classList.add('logged-in');
                logoutBtn.classList.remove('hidden');
            } else {
                authStatusDiv.textContent = 'Not logged in.';
                authStatusDiv.classList.remove('logged-in');
                authStatusDiv.classList.add('logged-out');
                logoutBtn.classList.add('hidden');
            }
        }

        // --- Product CRUD Operations (Mocked) ---
        async function fetchProducts() {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/products`, { headers:
getAuthHeader() })
            // For mock, just return local data
            return new Promise(resolve => setTimeout(() => resolve(products),
300));
        }

        async function createProduct(product) {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/products`, { method:
'POST', body: JSON.stringify(product), headers: { 'Content-Type':
'application/json', ...getAuthHeader() } })
            return new Promise(resolve => {
                setTimeout(() => {
                    const newProduct = { ...product, id: nextProductId++ };
                    products.push(newProduct);
                    resolve({ status: 'success', message: 'Product added!',
product: newProduct });
                }, 300);
            });
        }

        async function updateProduct(id, updatedProduct) {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/products/${id}`, {
method: 'PUT', body: JSON.stringify(updatedProduct), headers: { 'Content-Type':
'application/json', ...getAuthHeader() } })
            return new Promise(resolve => {
                setTimeout(() => {
                    const index = products.findIndex(p => p.id === id);
                    if (index !== -1) {
                        products[index] = { ...products[index],
...updatedProduct };
                        resolve({ status: 'success', message: 'Product
updated!', product: products[index] });
                    } else {
                        resolve({ status: 'error', message: 'Product not found.'
});
                    }
                }, 300);
            });
        }

        async function deleteProduct(id) {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/products/${id}`, {
method: 'DELETE', headers: getAuthHeader() })
            return new Promise(resolve => {
                setTimeout(() => {
                    const initialLength = products.length;
                    products = products.filter(p => p.id !== id);
                    if (products.length < initialLength) {
                        resolve({ status: 'success', message: 'Product deleted!'
});
                    } else {
                        resolve({ status: 'error', message: 'Product not found.'
});
                    }
```

```
                }, 300);
            });
        }

        // --- Authentication Functions (Mocked JWT) ---
        async function registerUser(username, password) {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/auth/register`, {
method: 'POST', body: JSON.stringify({ username, password }), headers: {
'Content-Type': 'application/json' } })
            return new Promise(resolve => {
                setTimeout(() => {
                    // Simulate user registration
                    if (username === 'testuser') { // Simple mock conflict
                        resolve({ status: 'error', message: 'Username already
exists.' });
                    } else {
                        // In a real app, hash password and save to DB
                        resolve({ status: 'success', message: 'Registration
successful! You can now log in.' });
                    }
                }, 500);
            });
        }

        async function loginUser(username, password) {
            // In a real app: fetch(`${MOCK_API_BASE_URL}/auth/login`, { method:
'POST', body: JSON.stringify({ username, password }), headers: { 'Content-Type':
'application/json' } })
            return new Promise(resolve => {
                setTimeout(() => {
                    // Simulate authentication and JWT generation
                    if (username === 'user' && password === 'pass') {
                        const mockJwt = 'mock_jwt_token_for_user_pass_12345'; //
Dummy JWT
                        authToken = mockJwt;
                        localStorage.setItem('jwt_token', mockJwt);
                        resolve({ status: 'success', message: 'Login
successful!', token: mockJwt });
                    } else {
                        resolve({ status: 'error', message: 'Invalid username or
password.' });
                    }
                }, 500);
            });
        }

        function logoutUser() {
            authToken = null;
            localStorage.removeItem('jwt_token');
            updateAuthStatus();
            displayMessage('login-message', 'Logged out successfully.',
'success');
        }

        // --- UI Update Functions ---
        function displayProducts() {
            const productList = document.getElementById('product-list');
            productList.innerHTML = '';
            products.forEach(product => {
                const row = productList.insertRow();
                row.innerHTML = `
                    <td>${product.id}</td>
                    <td>${product.name}</td>
                    <td>$${product.price.toFixed(2)}</td>
                    <td>${product.category}</td>
                    <td>${product.quantity}</td>
                    <td>
```

```
                        <button class="edit-btn" data-
id="${product.id}">Edit</button>
                        <button class="delete-btn" data-
id="${product.id}">Delete</button>
                    </td>
                `;
            });
        }

        function displayMessage(elementId, message, type) {
            const msgDiv = document.getElementById(elementId);
            msgDiv.textContent = message;
            msgDiv.className = `message ${type}`;
            msgDiv.style.display = 'block';
            setTimeout(() => { msgDiv.style.display = 'none'; }, 5000); // Hide
after 5 seconds
        }

        function clearProductForm() {
            document.getElementById('product-id').value = '';
            document.getElementById('product-name').value = '';
            document.getElementById('product-price').value = '';
            document.getElementById('product-category').value = '';
            document.getElementById('product-quantity').value = '';
            document.getElementById('product-form-title').textContent = 'Add New
Product';
            document.getElementById('submit-product-btn').textContent = 'Add
Product';
            document.getElementById('cancel-edit-btn').classList.add('hidden');
        }

        // --- Event Listeners ---
        document.addEventListener('DOMContentLoaded', () => {
            updateAuthStatus();
            displayProducts(); // Load products on page load
        });

        document.getElementById('product-form').addEventListener('submit', async
(event) => {
            event.preventDefault();
            const id = document.getElementById('product-id').value;
            const name = document.getElementById('product-name').value;
            const price = parseFloat(document.getElementById('product-
price').value);
            const category = document.getElementById('product-category').value;
            const quantity = parseInt(document.getElementById('product-
quantity').value);

            if (!authToken) {
                displayMessage('product-message', 'You must be logged in to
add/update products.', 'error');
                return;
            }

            const productData = { name, price, category, quantity };
            let result;
            if (id) {
                result = await updateProduct(parseInt(id), productData);
            } else {
                result = await createProduct(productData);
            }

            if (result.status === 'success') {
                displayProducts();
                clearProductForm();
                displayMessage('product-message', result.message, 'success');
            } else {
```

```
                displayMessage('product-message', result.message, 'error');
            }
        });

        document.getElementById('product-list').addEventListener('click', async
(event) => {
            if (!authToken) {
                displayMessage('product-message', 'You must be logged in to
edit/delete products.', 'error');
                return;
            }

            const target = event.target;
            const productId = parseInt(target.dataset.id);

            if (target.classList.contains('delete-btn')) {
                if (confirm('Are you sure you want to delete this product?')) {
                    const result = await deleteProduct(productId);
                    if (result.status === 'success') {
                        displayProducts();
                        displayMessage('product-message', result.message,
'success');
                    } else {
                        displayMessage('product-message', result.message,
'error');
                    }
                }
            } else if (target.classList.contains('edit-btn')) {
                const productToEdit = products.find(p => p.id === productId);
                if (productToEdit) {
                    document.getElementById('product-id').value =
productToEdit.id;
                    document.getElementById('product-name').value =
productToEdit.name;
                    document.getElementById('product-price').value =
productToEdit.price;
                    document.getElementById('product-category').value =
productToEdit.category;
                    document.getElementById('product-quantity').value =
productToEdit.quantity;

                    document.getElementById('product-form-title').textContent =
`Edit Product (ID: ${productToEdit.id})`;
                    document.getElementById('submit-product-btn').textContent =
'Update Product';
                    document.getElementById('cancel-edit-
btn').classList.remove('hidden');
                }
            }
        });

        document.getElementById('cancel-edit-btn').addEventListener('click',
clearProductForm);

        document.getElementById('register-form').addEventListener('submit',
async (event) => {
            event.preventDefault();
            const username = document.getElementById('register-username').value;
            const password = document.getElementById('register-password').value;
            const result = await registerUser(username, password);
            displayMessage('register-message', result.message, result.status);
        });

        document.getElementById('login-form').addEventListener('submit', async
(event) => {
            event.preventDefault();
            const username = document.getElementById('login-username').value;
```

```
            const password = document.getElementById('login-password').value;
            const result = await loginUser(username, password);
            displayMessage('login-message', result.message, result.status);
            if (result.status === 'success') {
                updateAuthStatus();
                // Clear login form
                document.getElementById('login-username').value = '';
                document.getElementById('login-password').value = '';
            }
        });

        document.getElementById('logout-btn').addEventListener('click',
logoutUser);
    </script>
</body>
</html>
```

Input:

- **Registration:** Enter a username and password in the "Register" form.
- **Login:** Enter a username and password (e.g., `user` and `pass` for mock login) in the "Login" form.
- **Product CRUD (after login):**
    - **Create:** Fill the "Add New Product" form and click "Add Product".
    - **Read:** Products are displayed automatically.
    - **Update:** Click "Edit" next to a product, modify details in the form, and click "Update Product".
    - **Delete:** Click "Delete" next to a product and confirm.

Expected Output:

- A single HTML page with two main sections: Product Management and Authentication.
- **Authentication Section:**
    - Initially, "Not logged in." status.
    - Upon successful registration (mocked), a success message will appear.
    - Upon successful login (using `user`/`pass` for mock), the status will change to "Logged in!", and a "Logout" button will appear.
    - Invalid login attempts will show an error message.
- **Product Management Section:**
    - A form to add/update products.
    - A table displaying a list of mock products.
    - "Edit" and "Delete" buttons next to each product.
    - **Without login:** Attempts to add, edit, or delete products will result in an "You must be logged in..." error message.
    - **After successful login:**
        - Adding a new product will dynamically update the table.
        - Clicking "Edit" will populate the form with product details, changing the form title and button text to "Update Product". Submitting will update the product in the table.
        - Clicking "Delete" will remove the product from the table after confirmation.
    - All operations will show success or error messages dynamically without page refresh.