MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)
Juhu Versova Link Road, Andheri (West), Mumbai-53

Experiment No:1

Aim: Design/Create static website including elements, attributes list, frameworks multimedia.

Theory:

HTML stands for Hyper Text Markup Language, which is the core language used to structure content on the web. It organizes text, images, links, and media using tags and elements that browsers can interpret. As of 2025, over 95% of websites rely on HTML alongside CSS and JavaScript, making it a fundamental tool in modern web development.

- It is a markup language, not a programming language. This means it annotates text to define how it is structured and displayed by web browsers.
- It is a static language, meaning it does not inherently provide interactive features but can be combined with CSS for styling and JavaScript for interactivity.

In a nutshell, HTML is all about organizing and displaying information on a webpage. We can think of it as the bones or structure of a webpage.

Program:

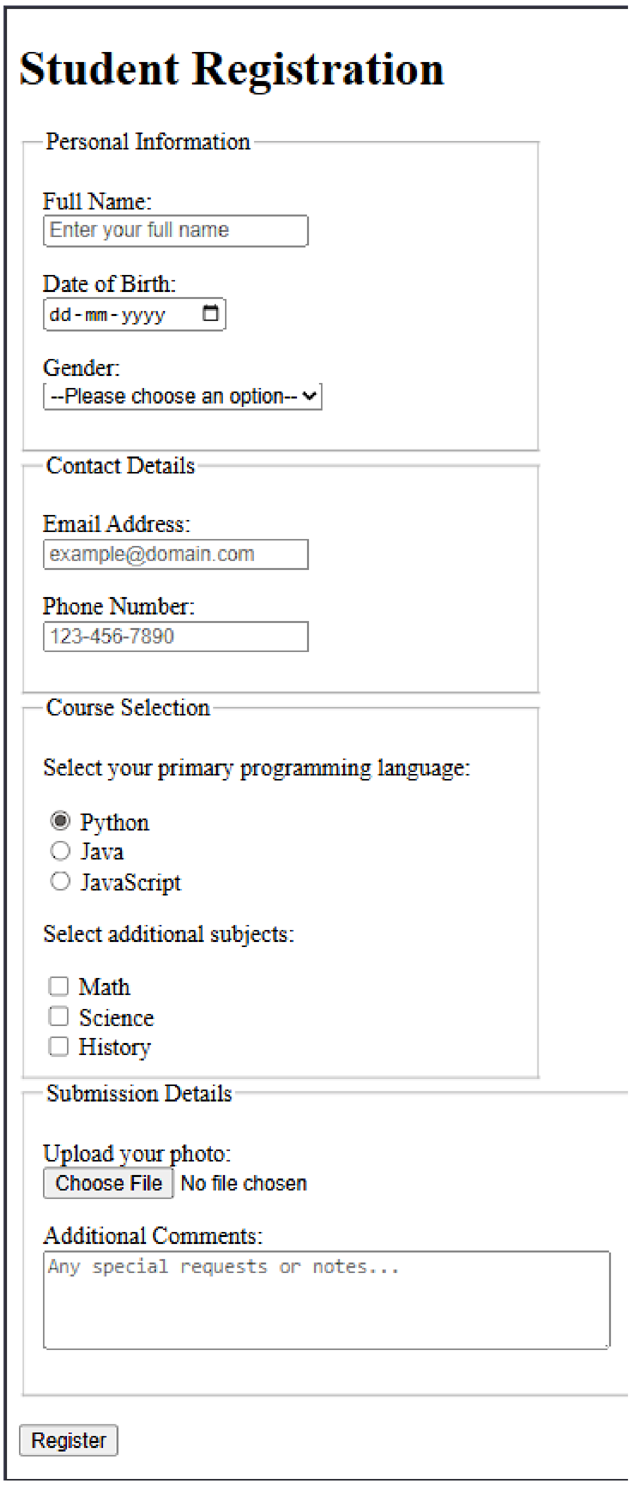
Index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Student Registration Form</title>
  </head>
  <body>
    <form id="studentRegistrationForm">
      <h1>Student Registration</h1>

      <fieldset>
        <legend>Personal Information</legend>

        <p>
          <label for="fullName">Full Name:</label><br />
          <input
            type="text"
            id="fullName"
            name="fullName"
            placeholder="Enter your full name"
            required
          />
        </p>

        <p>
          <label for="dob">Date of Birth:</label><br />
          <input type="date" id="dob" name="dob" required />
        </p>
      <p>
        <label for="gender">Gender:</label><br />
        <select id="gender" name="gender" required>
          <option value="">--Please choose an option--</option>
          <option value="male">Male</option>
          <option value="female">Female</option>
          <option value="other">Other</option>
        </select>
      </p>
    </fieldset>

    <fieldset>
      <legend>Contact Details</legend>

      <p>
        <label for="email">Email Address:</label><br />
        <input
          type="email"
          id="email"
          name="email"
          placeholder="example@domain.com"
          required
        />
      </p>

      <p>
        <label for="phone">Phone Number:</label><br />
        <input
          type="tel"
          id="phone"
          name="phone"
          placeholder="123-456-7890"
          pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"
        />
      </p>
    </fieldset>

    <fieldset>
      <legend>Course Selection</legend>

      <p>Select your primary programming language:</p>
      <input
        type="radio"
        id="python"
        name="primaryLanguage"
        value="Python"
        checked
```

```
  />
  <label for="python">Python</label><br />

  <input type="radio" id="java" name="primaryLanguage" value="Java" />
  <label for="java">Java</label><br />

  <input
    type="radio"
    id="javascript"
    name="primaryLanguage"
    value="JavaScript"
  />
  <label for="javascript">JavaScript</label><br />

  <p>Select additional subjects:</p>
  <input type="checkbox" id="math" name="subjects" value="math" />
  <label for="math">Math</label><br />

  <input type="checkbox" id="science" name="subjects" value="science" />
  <label for="science">Science</label><br />

  <input type="checkbox" id="history" name="subjects" value="history" />
  <label for="history">History</label><br />
</fieldset>

<fieldset>
  <legend>Submission Details</legend>

  <p>
    <label for="photo">Upload your photo:</label><br />
    <input
      type="file"
      id="photo"
      name="photo"
      accept="image/png, image/jpeg"
    />
  </p>

  <p>
    <label for="comments">Additional Comments:</label><br />
    <textarea
      id="comments"
      name="comments"
      rows="4"
      cols="50"
      placeholder="Any special requests or notes..."
    ></textarea>
```

```
      </p>
    </fieldset>

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

Output:



Student Registration

Personal Information

Full Name:
Enter your full name

Date of Birth:
dd-mm-yyyy

Gender:
--Please choose an option--

Contact Details

Email Address:
example@domain.com

Phone Number:
123-456-7890

Course Selection

Select your primary programming language:

- [✓ Python]
- Java
- JavaScript

Select additional subjects:

- Math
- Science
- History

Submission Details

Upload your photo:
Choose File No file chosen

Additional Comments:
Any special requests or notes...

Register

Conclusion:
The experiment involved designing a static website using HTML and CSS, incorporating key elements, attributes, multimedia (audio/video), and frameworks like Bootstrap. It demonstrated the effective use of web components to build a responsive and visually appealing site, reinforcing core web development skills.

Experiment No:2

Aim: Design/Create website including CSS3 syntax, inclusion color, background, fonts, tables, lists

Theory:

CSS (Cascading Style Sheets) is a language designed to simplify the process of making web pages presentable.

1. It allows you to apply styles to HTML documents by prescribing colors, fonts, spacing, and positioning.
2. The main advantages are the separation of content (in HTML) and styling (in CSS) and the same CSS rules can be used across all pages and not have to be rewritten.
3. HTML uses tags, and CSS uses rule sets.
4. CSS styles are applied to the HTML element using selectors.

CSS Syntax:

CSS consists of style rules that are interpreted by the browser and applied to the corresponding elements. A stylerule set includes a selector and a declaration block.

• Selector: Targets specific HTML elements to apply styles.
• Declaration: Combination of a property and its corresponding value.

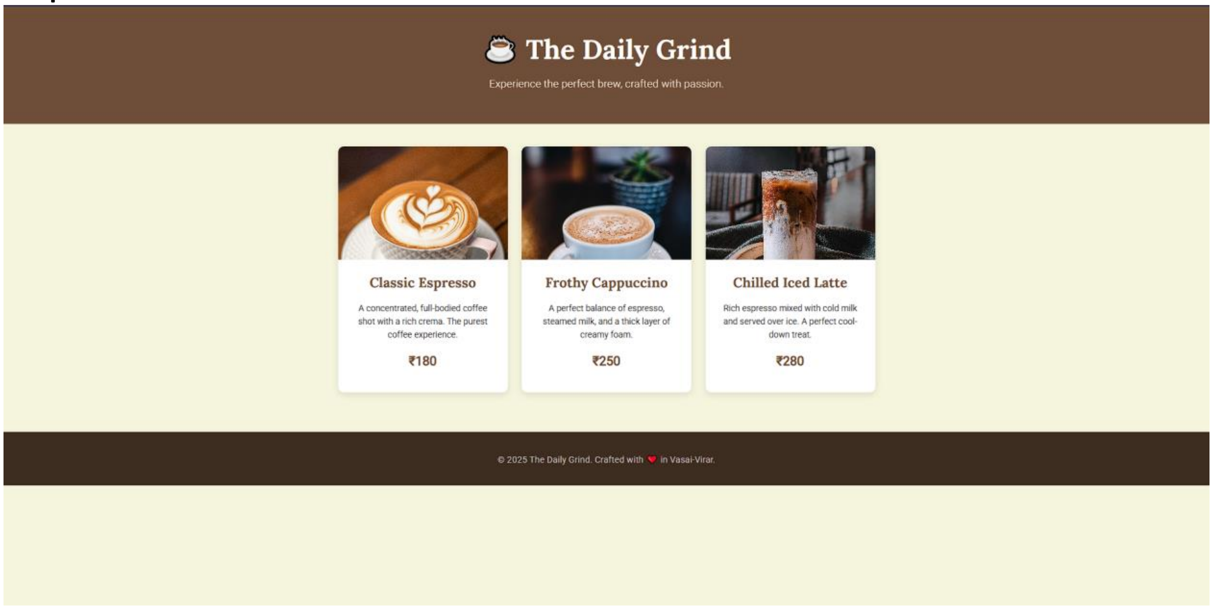
Program:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>The Daily Grind Coffee Shop</title>
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
      href="https://fonts.googleapis.com/css2?family=Lora:wght@400;700&family=Roboto:wght@3
00;400&display=swap"
      rel="stylesheet"
    />
    <style>
      /* General Styling */
      :root {
        --primary-color: #6f4e37; /* Coffee Brown */
        --secondary-color: #f5f5dc; /* Beige */
        --text-color: #333333;
        --card-bg: #ffffff;
      }

      body {
        font-family: "Roboto", sans-serif;
        margin: 0;
        background-color: var(--secondary-color);
        color: var(--text-color);
      }

      /* Header Styling */
      .main-header {
        background-color: var(--primary-color);
        color: white;
        text-align: center;
        padding: 40px 20px;
      }

      .main-header h1 {
        font-family: "Lora", serif;
        font-size: 3rem;
        margin: 0;
      }

      .main-header p {
        font-size: 1.2rem;
        font-weight: 300;
        opacity: 0.9;
      }

      /* Product Grid Styling */
      .product-grid {
        display: flex;
        flex-wrap: wrap;
        justify-content: center;
        gap: 25px;
        padding: 40px 20px;
        max-width: 1200px;
        margin: 0 auto;
      }

      /* Individual Card Styling */
      .product-card {
        background-color: var(--card-bg);
        border-radius: 10px;
        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
        overflow: hidden;
        width: 300px;
        text-align: center;
        transition: transform 0.3s ease, box-shadow 0.3s ease;
      }

      .product-card:hover {
        transform: translateY(-5px);
```

```
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
    }

    .product-card img {
      width: 100%;
      height: 200px;
      object-fit: cover; /* Ensures image covers the area without distortion */
    }

    .product-card-content {
      padding: 20px;
    }

    .product-card h2 {
      font-family: "Lora", serif;
      font-size: 1.5rem;
      margin-top: 0;
      color: var(--primary-color);
    }

    .product-card .description {
      font-size: 0.95rem;
      line-height: 1.5;
      margin-bottom: 20px;
    }

    .product-card .price {
      font-size: 1.4rem;
      font-weight: 700;
      color: var(--primary-color);
    }

    /* Footer Styling */
    .main-footer {
      text-align: center;
      padding: 20px;
      background-color: #3d2c1f;
      color: #ccc;
      margin-top: 30px;
    }
  </style>
</head>
<body>
  <header class="main-header">
    <h1>☕ The Daily Grind</h1>
    <p>Experience the perfect brew, crafted with passion.</p>
  </header>
  <main>
    <section class="product-grid">
      <article class="product-card">
        <img
          src="https://images.unsplash.com/photo-1630040995437-
80b01c5dd52d?fm=jpg&q=60&w=3000&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxzZWFyY2h8MTF8fGNvZmZlfGVufDB8fDB8fHww"
          alt="A rich cup of espresso"
        />
        <div class="product-card-content">
          <h2>Classic Espresso</h2>
          <p class="description">
            A concentrated, full-bodied coffee shot with a rich crema. The
            purest coffee experience.
          </p>
          <p class="price">₹180</p>
        </div>
      </article>

      <article class="product-card">
        <img
          src="https://images.unsplash.com/photo-1517256064527-09c73fc73e38?w=500"
          alt="A frothy cappuccino with latte art"
        />
        <div class="product-card-content">
          <h2>Frothy Cappuccino</h2>
          <p class="description">
            A perfect balance of espresso, steamed milk, and a thick layer of
            creamy foam.
          </p>
          <p class="price">₹250</p>
        </div>
      </article>

      <article class="product-card">
        <img
          src="https://images.unsplash.com/photo-1527156231393-
7023794f363c?fm=jpg&q=60&w=3000&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxzZWFyY2h8OXx8aWNlZCUyMGNvZmZlZXxlbnwwfHwwfHx8MA%3
D%3D"
          alt="A refreshing iced latte in a tall glass"
        />
        <div class="product-card-content">
          <h2>Chilled Iced Latte</h2>
          <p class="description">
            Rich espresso mixed with cold milk and served over ice. A perfect
            cool-down treat.
          </p>
          <p class="price">₹280</p>
        </div>
      </article>
    </section>
  </main>

  <footer class="main-footer">
    <p>&copy; 2025 The Daily Grind. Crafted with ❤️ in Vasai-Virar.</p>
  </footer>
</body>
</html>
```

Output:



☕ The Daily Grind

Experience the perfect brew, crafted with passion.

Classic Espresso

A concentrated, full-bodied coffee shot with a rich crema. The purest coffee experience.

₹180

Frothy Cappuccino

A perfect balance of espresso, steamed milk, and a thick layer of creamy foam.

₹250

Chilled Iced Latte

Rich espresso mixed with cold milk and served over ice. A perfect cool-down treat.

₹280

© 2025 The Daily Grind. Crafted with ❤ in Vasai-Virar.

Conclusion:

The experiment is focused on designing a static website using HTML and CSS3, demonstrating the use of various CSS3 properties such as color schemes, background customization, font styling, and layout control. Key HTML elements like tables and lists were also incorporated to organize content effectively. Through this experiment, the practical implementation of CSS3 syntax was explored to enhance both the functionality and aesthetic appeal of the website, providing a solid foundation in modern web design principles.

Experiment No:3

Aim: Design/Create website using Bootstrap.

Theory:

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. It solves many problems which we had once, one of which is the cross-browser compatibility issue.

Nowadays, the websites are perfect for all the browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones).

All thanks to Bootstrap developers -Mark Otto and Jacob Thornton of Twitter, though it was later declared to be an open-source project. Bootstrap has evolved many versions and every time when we want to use this framework we can select the version which we want to use.

Program:

Index.html:

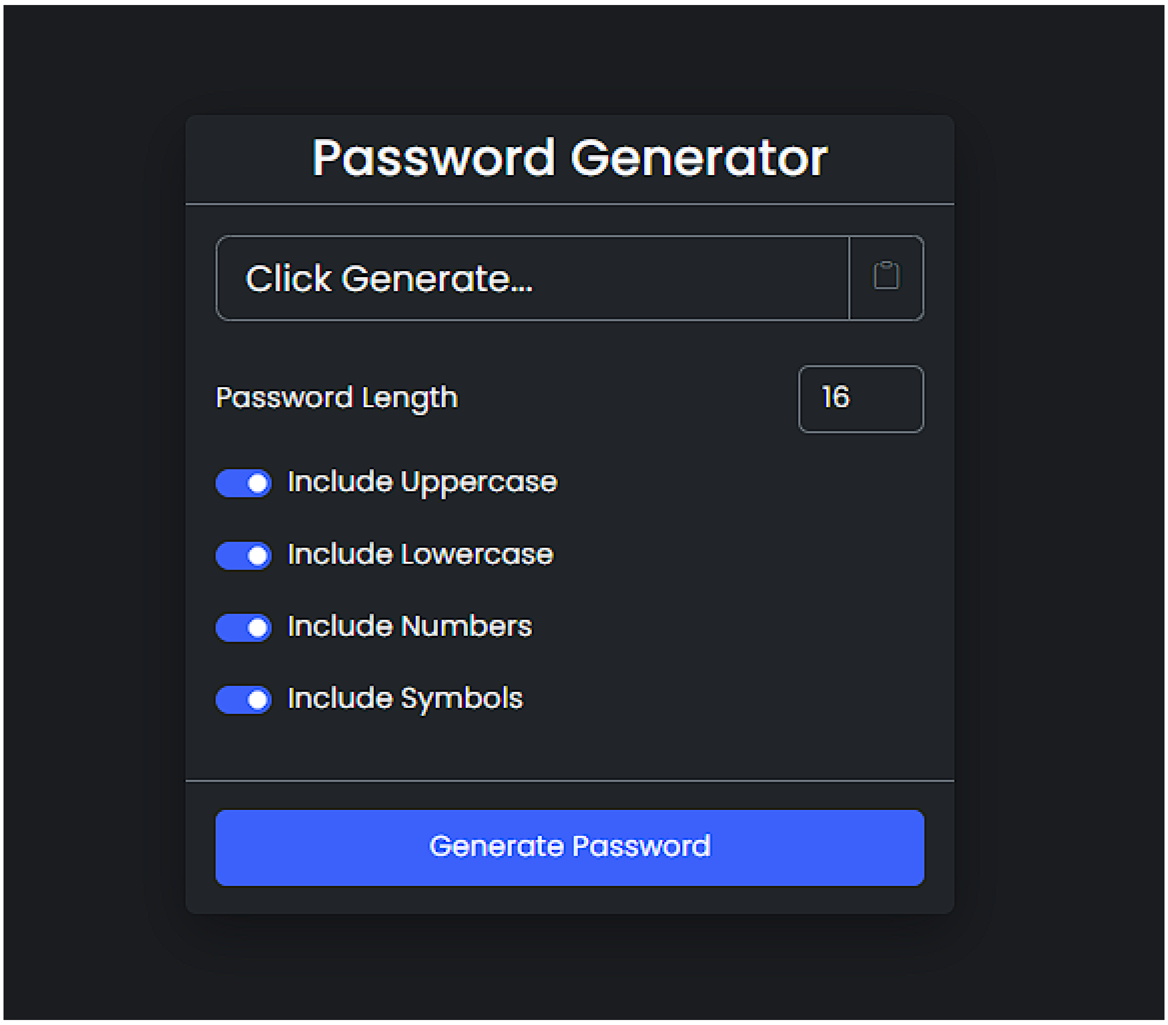
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Secure Password Generator</title>
   <!-- Bootstrap CSS -->
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">
   <!-- Bootstrap Icons -->
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.3/font/bootstrap-icons.min.css">
   <style>
      @import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600&display=swap');

      :root {
         --bg-color: #1a1c20;
         --container-bg: #25282f;
         --primary-color: #3d5aff;
         --text-color: #f0f0f0;
      }

      body {
         font-family: 'Poppins', sans-serif;
         background-color: var(--bg-color);
      }

      .card {
         border: none;
         box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
```

```
        }

        .form-control:focus, .form-check-input:focus {
            box-shadow: 0 0 0 0.25rem rgba(var(--bs-primary-rgb), 0.4);
        }

        .form-control[readonly] {
            background-color: #1a1c20;
        }

        .form-switch .form-check-input:checked {
            background-color: var(--primary-color);
            border-color: var(--primary-color);
        }

        .btn-primary {
            background-color: var(--primary-color);
            border-color: var(--primary-color);
        }
        .btn-primary:hover {
            background-color: #5d78ff;
            border-color: #5d78ff;
        }
    </style>
</head>
<body class="d-flex align-items-center justify-content-center min-vh-100">

    <div class="container" style="max-width: 450px;">
        <div class="card bg-dark text-light border-secondary">
            <div class="card-header text-center border-secondary">
                <h2 class="h3 mb-0">Password Generator</h2>
            </div>
            <div class="card-body">
                <div class="input-group mb-4">
                    <input type="text" id="passwordDisplay" class="form-control form-control-lg bg-dark
text-light border-secondary" value="Click Generate..." readonly>
                    <button class="btn btn-outline-secondary" type="button" id="copyBtn" title="Copy to
clipboard">
                        <i class="bi bi-clipboard"></i>
                    </button>
                </div>

                <div class="settings">
                    <div class="d-flex justify-content-between align-items-center mb-3">
                        <label for="length" class="form-label mb-0">Password Length</label>
                        <input type="number" id="length" min="8" max="32" value="16" class="form-
control bg-dark text-light border-secondary" style="width: 70px;">
```

```
          </div>
          <div class="form-check form-switch mb-3">
            <input class="form-check-input" type="checkbox" role="switch" id="uppercase"
checked>
            <label class="form-check-label" for="uppercase">Include Uppercase</label>
          </div>
          <div class="form-check form-switch mb-3">
            <input class="form-check-input" type="checkbox" role="switch" id="lowercase"
checked>
            <label class="form-check-label" for="lowercase">Include Lowercase</label>
          </div>
          <div class="form-check form-switch mb-3">
            <input class="form-check-input" type="checkbox" role="switch" id="numbers"
checked>
            <label class="form-check-label" for="numbers">Include Numbers</label>
          </div>
          <div class="form-check form-switch mb-3">
            <input class="form-check-input" type="checkbox" role="switch" id="symbols"
checked>
            <label class="form-check-label" for="symbols">Include Symbols</label>
          </div>
        </div>
      </div>
      <div class="card-footer p-3 border-secondary">
        <button id="generateBtn" class="btn btn-primary w-100 py-2">Generate
Password</button>
      </div>
    </div>
  </div>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
  <script>
    // DOM Elements
    const passwordDisplay = document.getElementById('passwordDisplay');
    const lengthEl = document.getElementById('length');
    const uppercaseEl = document.getElementById('uppercase');
    const lowercaseEl = document.getElementById('lowercase');
    const numbersEl = document.getElementById('numbers');
    const symbolsEl = document.getElementById('symbols');
    const generateBtn = document.getElementById('generateBtn');
    const copyBtn = document.getElementById('copyBtn');
    const copyBtnIcon = copyBtn.querySelector('i');

    // Character sets
    const charSets = {
      upper: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
      lower: 'abcdefghijklmnopqrstuvwxyz',
      number: '0123456789',
      symbol: '!@#$%^&*()_+~`|}{[]:;?><,./-='
    };

    // Generate Password Function
    function generatePassword() {
      const length = +lengthEl.value;
      let charset = '';
      let password = '';

      if (uppercaseEl.checked) charset += charSets.upper;
      if (lowercaseEl.checked) charset += charSets.lower;
      if (numbersEl.checked) charset += charSets.number;
      if (symbolsEl.checked) charset += charSets.symbol;

      if (charset === '') {
        passwordDisplay.value = 'Select options!';
        return;
      }

      for (let i = 0; i < length; i++) {
        const randomIndex = Math.floor(Math.random() * charset.length);
        password += charset[randomIndex];
      }

      passwordDisplay.value = password;
    }

    // Copy to Clipboard Function with visual feedback
    function copyToClipboard() {
      const password = passwordDisplay.value;
      if (!password || password === 'Click Generate...' || password === 'Select options!') return;

      navigator.clipboard.writeText(password).then(() => {
        // Change icon to a checkmark
        copyBtnIcon.classList.remove('bi-clipboard');
        copyBtnIcon.classList.add('bi-clipboard-check-fill');

        // Change back to original icon after 2 seconds
        setTimeout(() => {
          copyBtnIcon.classList.remove('bi-clipboard-check-fill');
          copyBtnIcon.classList.add('bi-clipboard');
        }, 2000);
      });
    }
```

```
        // Event Listeners
        generateBtn.addEventListener('click', generatePassword);
        copyBtn.addEventListener('click', copyToClipboard);

    </script>

</body>
</html>
```

Output:


Password Generator

Click Generate...

Password Length 16

Include Uppercase

Include Lowercase

Include Numbers

Include Symbols

Generate Password

Conclusion:
The website was successfully created using Bootstrap, demonstrating the use of its responsive grid system, components, and utility classes. By integrating Bootstrap's pre-designed elements like navigation bars, buttons, cards, and forms, the development process was efficient and visually consistent across devices. This project enhanced understanding of responsive web design and showcased the power of using a front-end framework to build modern, mobile-friendly websites.

Experiment No:4

Aim: Write a JavaScript program to validate a Student Registration Form.

Theory:

Validation is the process of checking whether the input data meets the required rules before it is processed or stored.
It ensures that the data is accurate, complete, and in the correct format.
In the Student Registration Form, HTML provides the structure while JavaScript performs client-side validation.
JavaScript checks required fields such as name, email, phone number, gender, address, branch, and student photo.
If any input is invalid, a red error message is displayed, and the form cannot be submitted until all errors are corrected.
This improves user experience and ensures reliable data entry.

Program:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Modern Student Registration Form</title>
  <style>
   body {
    font-family: "Segoe UI", sans-serif;
    background: linear-gradient(120deg, #74ebd5, #acb6e5);
    margin: 0;
    padding: 20px;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    box-sizing: border-box;
   }

   .container {
    max-width: 600px;
    width: 100%;
    margin: 20px auto;
    background-color: #ffffff;
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.1);
   }

   h2 {
```

```
  text-align: center;
  color: #333;
  margin-bottom: 25px;
}

label {
  display: block;
  margin: 15px 0 5px;
  color: #555;
  font-weight: 600;
}

input[type="text"],
input[type="email"],
input[type="tel"],
input[type="password"],
input[type="date"],
select {
  width: 100%;
  padding: 12px;
  margin-top: 5px;
  border-radius: 5px;
  border: 1px solid #ccc;
  box-sizing: border-box;
  transition: border-color 0.3s ease;
}

input:focus,
select:focus {
  outline: none;
  border-color: #74ebd5;
}

.gender-container {
  margin-top: 10px;
  display: flex;
  gap: 20px;
}

.gender-container label {
  display: inline-block;
  margin: 0;
  font-weight: normal;
}

.error {
  color: #e74c3c;
```

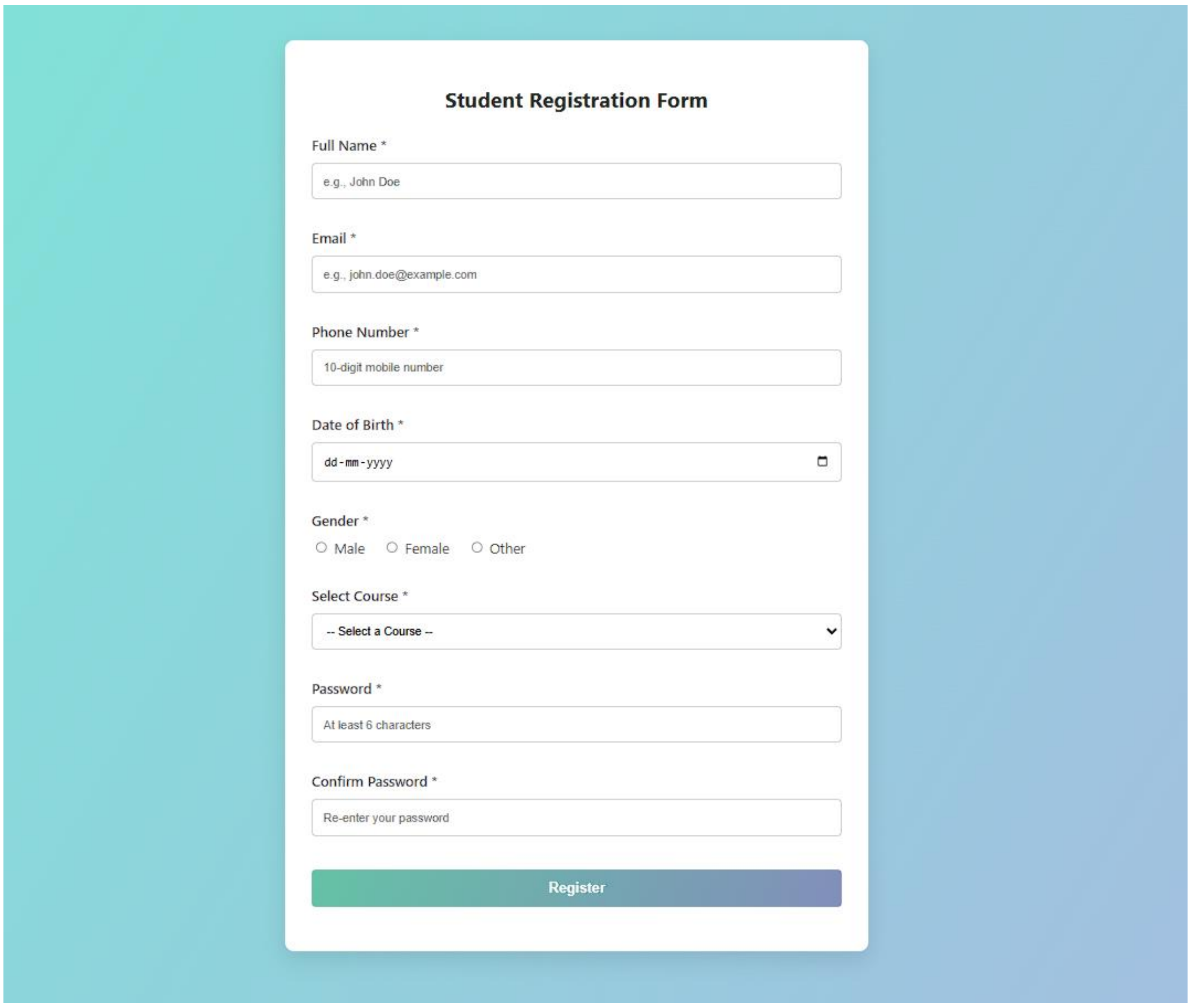
```
      font-size: 0.85em;
      height: 1em; /* Reserve space to prevent layout shifts */
      margin-top: 4px;
    }

    .success {
      color: #2ecc71;
      text-align: center;
      margin-top: 20px;
      font-weight: bold;
    }

    button {
      margin-top: 20px;
      width: 100%;
      padding: 12px;
      background: linear-gradient(120deg, #62c3a5, #828dbc);
      color: white;
      border: none;
      font-size: 16px;
      font-weight: bold;
      border-radius: 5px;
      cursor: pointer;
      transition: opacity 0.3s ease;
    }

    button:hover {
      opacity: 0.9;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>Student Registration Form</h2>
    <form id="registrationForm" novalidate>
      <label for="name">Full Name *</label>
      <input
        type="text"
        id="name"
        name="name"
        required
        placeholder="e.g., John Doe"
      />
      <div class="error" id="nameError"></div>

      <label for="email">Email *</label>
      <input
  type="email"
  id="email"
  name="email"
  required
  placeholder="e.g., john.doe@example.com"
/>
<div class="error" id="emailError"></div>

<label for="phone">Phone Number *</label>
<input
  type="tel"
  id="phone"
  name="phone"
  required
  placeholder="10-digit mobile number"
/>
<div class="error" id="phoneError"></div>

<label for="dob">Date of Birth *</label>
<input type="date" id="dob" name="dob" required />
<div class="error" id="dobError"></div>

<label>Gender *</label>
<div class="gender-container">
  <label><input type="radio" name="gender" value="Male" /> Male</label>
  <label
    ><input type="radio" name="gender" value="Female" /> Female</label
  >
  <label
    ><input type="radio" name="gender" value="Other" /> Other</label
  >
</div>
<div class="error" id="genderError"></div>

<label for="course">Select Course *</label>
<select id="course" name="course" required>
  <option value="">-- Select a Course --</option>
  <option value="B.Tech">B.Tech</option>
  <option value="B.Sc">B.Sc</option>
  <option value="B.Com">B.Com</option>
  <option value="MBA">MBA</option>
</select>
<div class="error" id="courseError"></div>

<label for="password">Password *</label>
<input
  type="password"
```

```
      id="password"
      name="password"
      required
      placeholder="At least 6 characters"
    />
    <div class="error" id="passwordError"></div>

    <label for="confirmPassword">Confirm Password *</label>
    <input
      type="password"
      id="confirmPassword"
      name="confirmPassword"
      required
      placeholder="Re-enter your password"
    />
    <div class="error" id="confirmPasswordError"></div>

    <button type="submit">Register</button>
    <div class="success" id="successMessage"></div>
  </form>
</div>

<script>
  const form = document.getElementById("registrationForm");
  const successMessage = document.getElementById("successMessage");

  form.addEventListener("submit", function (e) {
    e.preventDefault();

    // Clear previous messages
    const errors = document.querySelectorAll(".error");
    errors.forEach((el) => (el.textContent = ""));
    successMessage.textContent = "";

    let isValid = true;

    const name = form.name.value.trim();
    const email = form.email.value.trim();
    const phone = form.phone.value.trim();
    const dob = form.dob.value;
    const gender = form.gender.value;
    const course = form.course.value;
    const password = form.password.value;
    const confirmPassword = form.confirmPassword.value;

    // Name validation
    if (name === "" || !/^[A-Za-z\s]+$/.test(name)) {
```

```
  document.getElementById("nameError").textContent =
    "Please enter a valid name (letters and spaces only).";
  isValid = false;
}

// Email validation
if (email === "" || !/^\S+@\S+\.\S+$/.test(email)) {
  document.getElementById("emailError").textContent =
    "Please enter a valid email address.";
  isValid = false;
}

// Phone validation
if (phone === "" || !/^[0-9]{10}$/.test(phone)) {
  document.getElementById("phoneError").textContent =
    "Please enter a valid 10-digit phone number.";
  isValid = false;
}

// DOB validation
if (!dob) {
  document.getElementById("dobError").textContent =
    "Please select your date of birth.";
  isValid = false;
}

// Gender validation
if (!gender) {
  document.getElementById("genderError").textContent =
    "Please select your gender.";
  isValid = false;
}

// Course validation
if (course === "") {
  document.getElementById("courseError").textContent =
    "Please select a course.";
  isValid = false;
}

// Password validation
if (password.length < 6) {
  document.getElementById("passwordError").textContent =
    "Password must be at least 6 characters long.";
  isValid = false;
}
```

```
    // Confirm Password validation
    if (password !== confirmPassword) {
      document.getElementById("confirmPasswordError").textContent =
        "Passwords do not match.";
      isValid = false;
    }

    if (isValid) {
      successMessage.textContent = "Registration successful!";
      form.reset();
      // Optionally, clear the success message after a few seconds
      setTimeout(() => {
        successMessage.textContent = "";
      }, 5000);
    }
  });
 </script>
</body>
</html>
```

Output:



Student Registration Form

Full Name *

e.g., John Doe

Email *

e.g., john.doe@example.com

Phone Number *

10-digit mobile number

Date of Birth *

dd-mm-yyyy

Gender *

○ Male ○ Female ○ Other

Select Course *

-- Select a Course --

Password *

At least 6 characters

Confirm Password *

Re-enter your password

Register

Conclusion:
The JavaScript-based validation in the Student Registration Form ensures that only correct and complete data is submitted.
It prevents invalid inputs, reduces user errors, and improves data accuracy.
By displaying red error messages instantly, it guides users to correct mistakes before submission, making the form more user-friendly and reliable.

Experiment No:5

Aim: Write a JavaScript program to ES6 features in function using map( ), reduce( ), filter( ).

Theory:

ES6, or ECMAScript 2015, is the 6th version of the ECMAScript programming language. ECMAScript is the standardization of JavaScript, which was released in 2015 and subsequently renamed as ECMAScript 2015.

An array is a homogeneous collection of values. It is a single variable that is used to store different elements. It is often used when we want to store a list of elements and access them by a single variable. Unlike most languages where the array is a reference to the multiple variables, an ES6 array is a single variable that stores multiple elements.

There are lots of array methods introduced in ES6. These are some important methods in ES6.

Methods	**Description**
concat( )	This method is used to merge two or more arrays together
filter( )	This method creates a new array with elements that follow or pass the given criteria and conditions.
find( )	Filter elements through the function, and return first/all values that make it return true.
forEach( )	This method is used to iterate through its elements and manipulate them.
Array.from( )	This change all thing that is array-like or iterable into true array especially when working with DOM, so that you can use other array methods like reduce, map, filter, and so on.
Indexof( )	This method is used to find the index of the first occurrence of the search element provided as the argument to the method.
join( )	This method is used to join the elements of an array into a string.
lastIndexOf()	Look for an item starting from position pos, and return the index or -1 if not found.
map()	This method creates a new array by calling the provided function in every element.
reduce()	The reduce() method applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value - MDN
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.

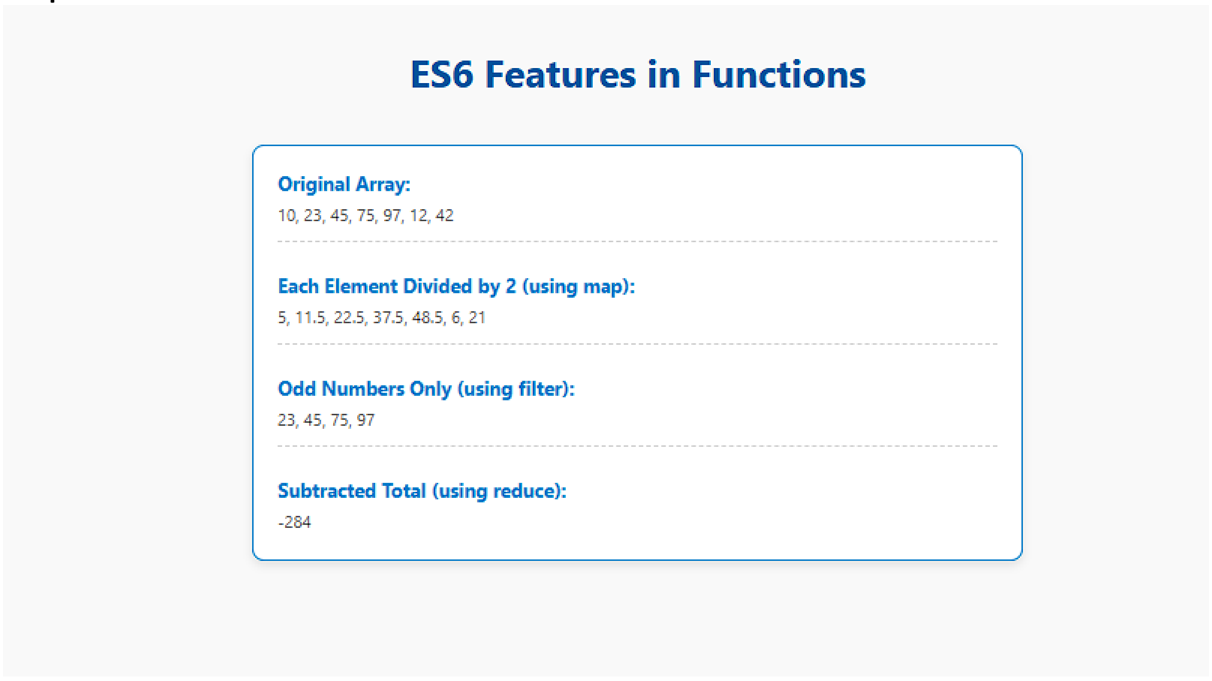
Program:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ES6 Function Features</title>
    <style>
      body {
        font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
        background-color: #f9f9f9;
        margin: 40px;
        padding: 0;
        color: #333;
      }

      h1 {
        text-align: center;
        color: #00509e;
        margin-bottom: 40px;
      }

      #ES6-Feature {
        max-width: 600px;
        margin: 0 auto;
        padding: 20px;
        border: 2px solid #0077cc;
        border-radius: 10px;
        background-color: #ffffff;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      }

      .result-block {
        margin-bottom: 25px;
        padding-bottom: 10px;
        border-bottom: 1px dashed #ccc;
      }

      .result-block:last-child {
        border-bottom: none;
        margin-bottom: 0;
        padding-bottom: 0;
      }

      .result-block h2 {
```

```
      font-size: 16px;
      color: #0077cc;
      margin: 0 0 5px 0;
    }

    .result-block p {
      font-size: 14px;
      line-height: 1.6;
      color: #444;
      margin: 0;
      word-break: break-word;
    }
  </style>
</head>
<body>
  <h1>ES6 Features in Functions</h1>
  <div id="ES6-Feature"></div>
  <script>
    const arr1 = [10, 23, 45, 75, 97, 12, 42];

    // 1. map() with arrow function
    const division = arr1.map((val) => val / 2);

    // 2. filter() with arrow function
    const odds = arr1.filter((val) => val % 2 !== 0);

    // 3. reduce() with arrow function
    const subtract = arr1.reduce(
      (accumulator, currentValue) => accumulator - currentValue
    );

    const outputDiv = document.getElementById("ES6-Feature");

    // Helper function to display results in the UI
    const createSection = (title, data) => {
      const section = document.createElement("div");
      section.className = "result-block";

      const heading = document.createElement("h2");
      heading.textContent = title;

      const content = document.createElement("p");
      content.textContent = Array.isArray(data) ? data.join(", ") : data;

      section.appendChild(heading);
      section.appendChild(content);
      outputDiv.appendChild(section);
    };

    // Displaying the results
    createSection("Original Array:", arr1);
    createSection("Each Element Divided by 2 (using map):", division);
    createSection("Odd Numbers Only (using filter):", odds);
    createSection("Subtracted Total (using reduce):", subtract);
  </script>
 </body>
</html>
```

Output:



ES6 Features in Functions

```
Original Array:
10, 23, 45, 75, 97, 12, 42

Each Element Divided by 2 (using map):
5, 11.5, 22.5, 37.5, 48.5, 6, 21

Odd Numbers Only (using filter):
23, 45, 75, 97

Subtracted Total (using reduce):
-284
```

Conclusion:
ES6 features like map, filter, and reduce make array operations simpler and cleaner. They help write concise, readable code by transforming, filtering, and reducing data efficiently without complex loops. Using these with arrow functions improves JavaScript’s expressiveness and maintainability.

Experiment No: 6

Aim: Write a JS program to demonstrate Promise and async / await.

Theory:

JavaScript is a single-threaded, non-blocking, asynchronous programming language. To manage asynchronous operations (like fetching data from an API or reading files), JavaScript provides mechanisms such as Callbacks, Promises, and Async/Await.

Promise:

A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. It can be in one of three states:

- Pending – Initial state, neither fulfilled nor rejected.
- Fulfilled – Operation completed successfully.
- Rejected – Operation failed.

Async/Await:

Async and **await** are syntactic sugar built on top of Promises, introduced in ES2017, to write cleaner and more readable asynchronous code.

- The async keyword is used to define an asynchronous function.
- The await keyword is used to pause the execution of an async function un-til a Promise is resolved or rejected.

Program:

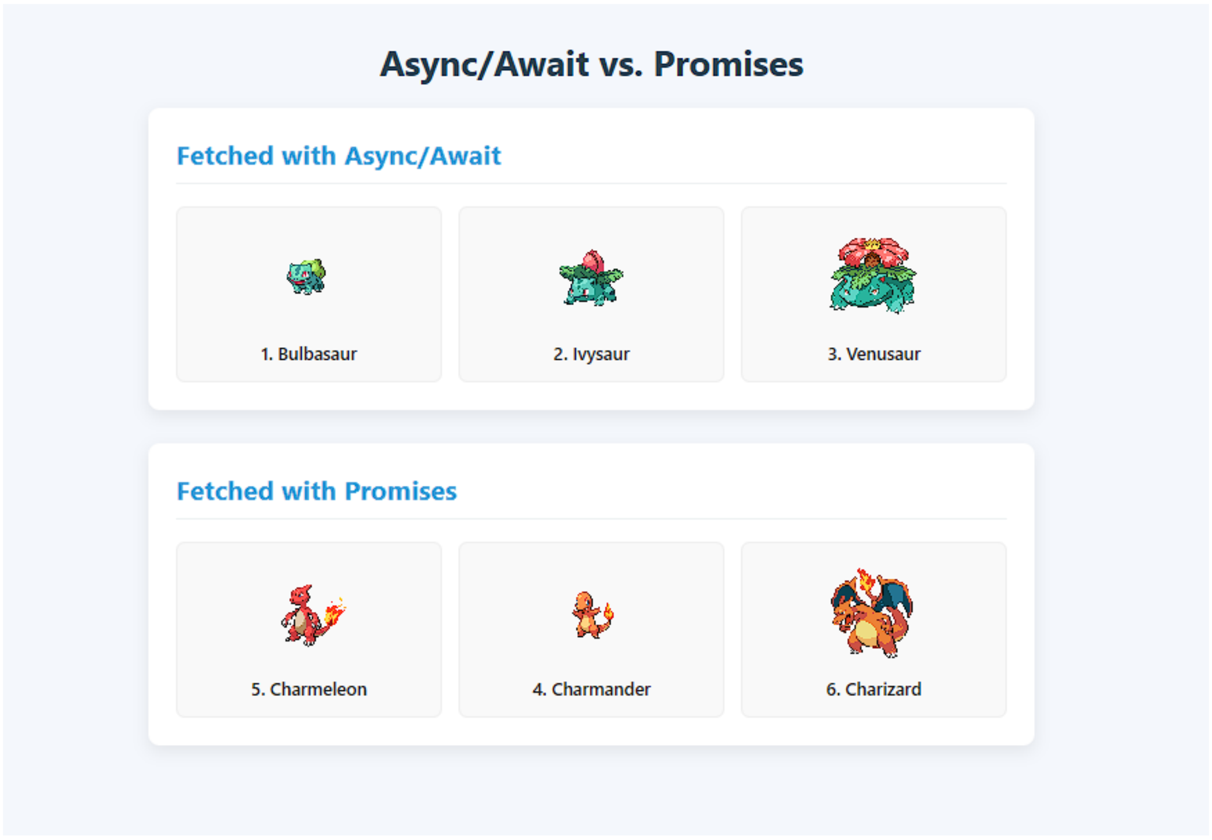
Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Async/Await vs Promises Demo</title>
  <style>
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background-color: #f4f7fc;
      margin: 0;
      padding: 20px;
      color: #333;
    }

    h1 {
      text-align: center;
      color: #2c3e50;
    }

    .container {
      max-width: 800px;
      margin: 20px auto;
```

```
            display: grid;
            grid-template-columns: 1fr;
            gap: 30px;
        }

        .fetch-section {
            background-color: #ffffff;
            padding: 25px;
            border-radius: 10px;
            box-shadow: 0 5px 15px rgba(0, 0, 0, 0.08);
        }

        .fetch-section h2 {
            margin-top: 0;
            color: #3498db;
            border-bottom: 2px solid #ecf0f1;
            padding-bottom: 10px;
        }

        .results-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
            gap: 15px;
        }

        .pokemon-card {
            text-align: center;
            background-color: #f9f9f9;
            padding: 15px;
            border-radius: 8px;
            border: 1px solid #eee;
        }

        .pokemon-card img {
            width: 96px;
            height: 96px;
            image-rendering: pixelated; /* For crisp pixel art */
        }

        .pokemon-card span {
            display: block;
            margin-top: 5px;
            font-weight: 500;
            text-transform: capitalize;
        }
    </style>
</head>
```

```
<body>
  <h1>Async/Await vs. Promises</h1>

  <div class="container" id="app-container">
    <!-- Content will be generated by JavaScript -->
  </div>

  <script>
    const appContainer = document.getElementById('app-container');

    // --- Helper Function to Create UI ---
    const createSection = (id, title) => {
      const sectionDiv = document.createElement('div');
      sectionDiv.className = 'fetch-section';

      const heading = document.createElement('h2');
      heading.textContent = title;

      const resultsGrid = document.createElement('div');
      resultsGrid.id = id;
      resultsGrid.className = 'results-grid';

      sectionDiv.appendChild(heading);
      sectionDiv.appendChild(resultsGrid);
      appContainer.appendChild(sectionDiv);

      return resultsGrid;
    };

    const createPokemonCard = (parent, pokemon) => {
      const card = document.createElement('div');
      card.className = 'pokemon-card';

      const image = document.createElement('img');
      image.src = pokemon.sprites.front_default;
      image.alt = pokemon.name;

      const name = document.createElement('span');
      name.textContent = `${pokemon.id}. ${pokemon.name}`;

      card.appendChild(image);
      card.appendChild(name);
      parent.appendChild(card);
    };

    // --- Main Execution Logic ---
    (async () => {
      try {
        // 1. Fetching Data using async/await
        const asyncResultsContainer = createSection('async-results', 'Fetched with Async/Await');
        for (let i = 1; i <= 3; i++) {
          const url = `https://pokeapi.co/api/v2/pokemon/${i}`;
          const response = await fetch(url);
          const pokemon = await response.json();
          createPokemonCard(asyncResultsContainer, pokemon);
        }

        // 2. Fetching Data using Promises (.then)
        const promiseResultsContainer = createSection('promise-results', 'Fetched with
Promises');
        for (let i = 4; i <= 6; i++) {
          const url = `https://pokeapi.co/api/v2/pokemon/${i}`;
          fetch(url)
            .then(res => res.json())
            .then(pokemon => {
              createPokemonCard(promiseResultsContainer, pokemon);
            });
        }

      } catch (error) {
        console.error("Error fetching data:", error);
        appContainer.innerHTML = '<p style="color: red; text-align: center;">Failed to fetch
Pokémon data. Please try again later.</p>';
      }
    })();
  </script>
</body>
</html>
```

Output:


Async/Await vs. Promises

Fetched with Async/Await

1. Bulbasaur
2. Ivysaur
3. Venusaur

Fetched with Promises

5. Charmeleon
4. Charmander
6. Charizard

Conclusion:
From this experiment, we learnt how to make Api calls using JavaScript Async/Await and Promises. In this experiment you have successfully explored the asynchronous functionality/feature of JavaScript.

Experiment No:7

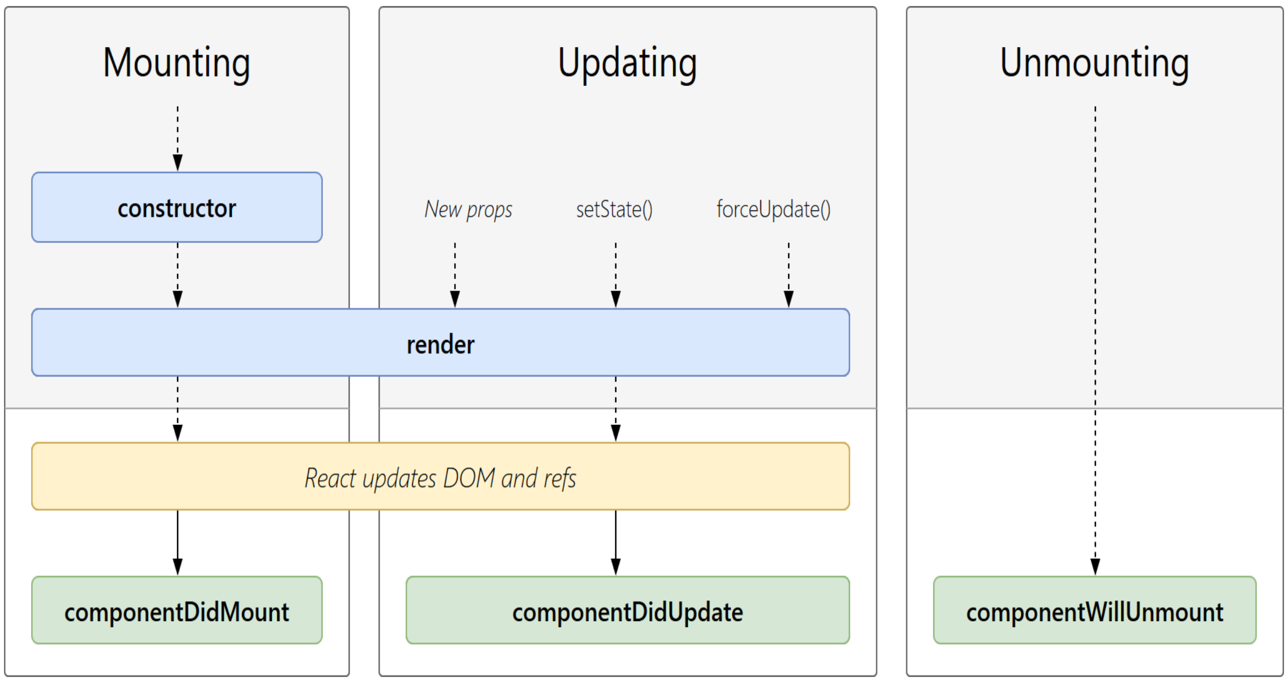
Aim: Implement of React and display “Welcome to TE IT”.

Theory:

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library that aims to make building user interfaces based on components more "seamless". It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js and Remix. Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality. A key advantage of React is that it only re-renders those parts of the page that have changed, avoiding unnecessary re-rendering of unchanged DOM elements.

React Lifecycle:



- **Mounting:** Initializes, renders, and mounts the component (componentDidMount()).
- **Updating**: Handles state/prop changes, re-renders, and updates (componentDidUpdate()).
- **Unmounting:** Cleans up before removal (componentWillUnmount()).

Phases of Lifecycle in React Components

1. Mounting

Mounting refers to the process of creating and inserting a component into the DOM for the first time in a React application. During mounting, React
initializes the component, sets up its internal state (if any), and inserts it into the DOM.

- **constructor()**

Method to initialize state and bind methods. Executed before the component is mounted.

- **getDerivedStateFromProps(props, state)**

Used for updating the state based on props. Executed before every render.

• render() method
Responsible for rendering JSX and updating the DOM.

• componentDidMount() Function

This function is invoked right after the component is mounted on the DOM, i.e. this function gets invoked once after the render() function is executed for the first time.

2. Updation
Updating refers to the process of a component being re-rendered due to changes in its state or props. This phase occurs whenever a component's internal state is modified or its parent component passes new props. When an update happens, React re-renders the component to reflect the changes and ensures that the DOM is updated accordingly.

• getDerivedStateFromProps
getDerivedStateFromProps(props, state) is a static method that is called just before the render() method in both the mounting and updating phase in React. It takes updated props and the current state as arguments.

• setState()
This is not particularly a Lifecycle function and can be invoked explicitly at any instant. This function is used to update the state of a component. You may refer to this article for detailed information.

• shouldComponentUpdate()
shouldComponentUpdate() Is a lifecycle method in React class components that determines whether a component should re-render. It compares the current and next props/states and returns true if the component should update or false if it should not.

• shouldComponentUpdate(nextProps, nextState)
It returns true or false, if false, then render(), componentWillUpdate(), and componentDidUpdate() method does not get invoked.

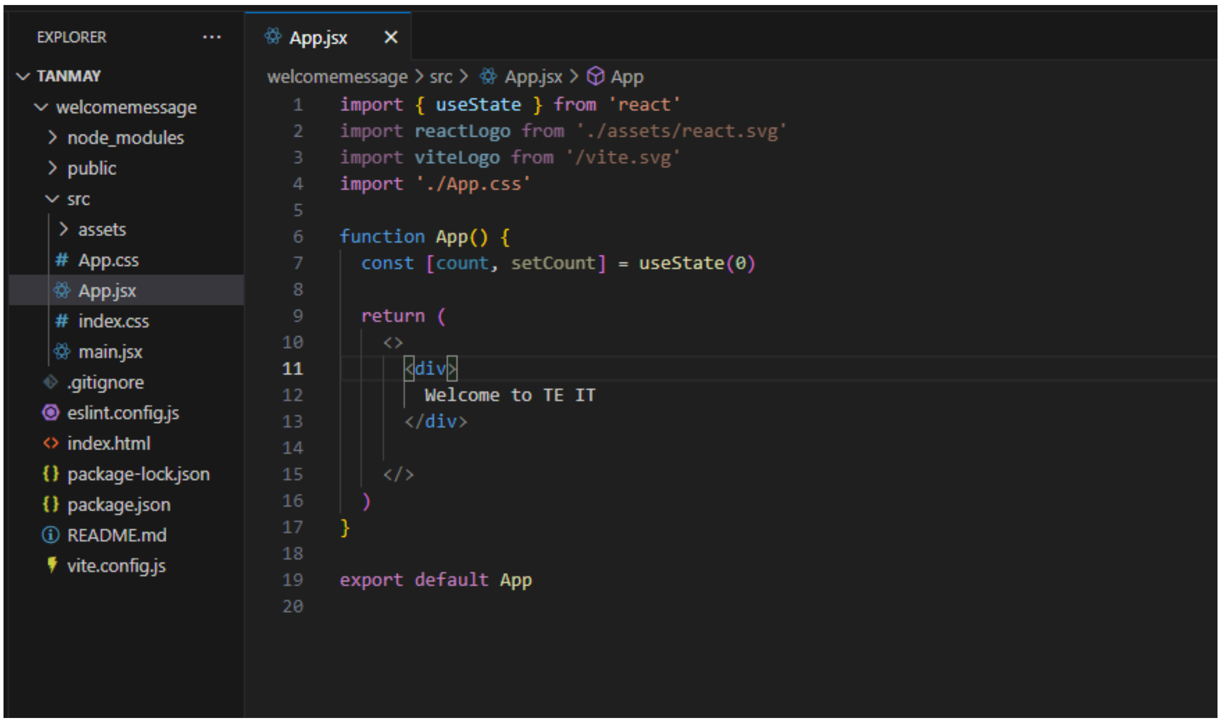
• getSnapshotBeforeUpdate() Method
The getSnapshotBeforeUpdate() method is invoked just before the DOM is being rendered. It is used to store the previous values of the state after the DOM is updated.

• componentDidUpdate()
Similarly, this function is invoked after the component is rendered, i.e., this function gets invoked once after the render() function is executed after the updation of State or Props.

3. Unmounting
This is the final phase of the lifecycle of the component, which is the phase of unmounting the component from the DOM. The following function is the sole member of this phase.

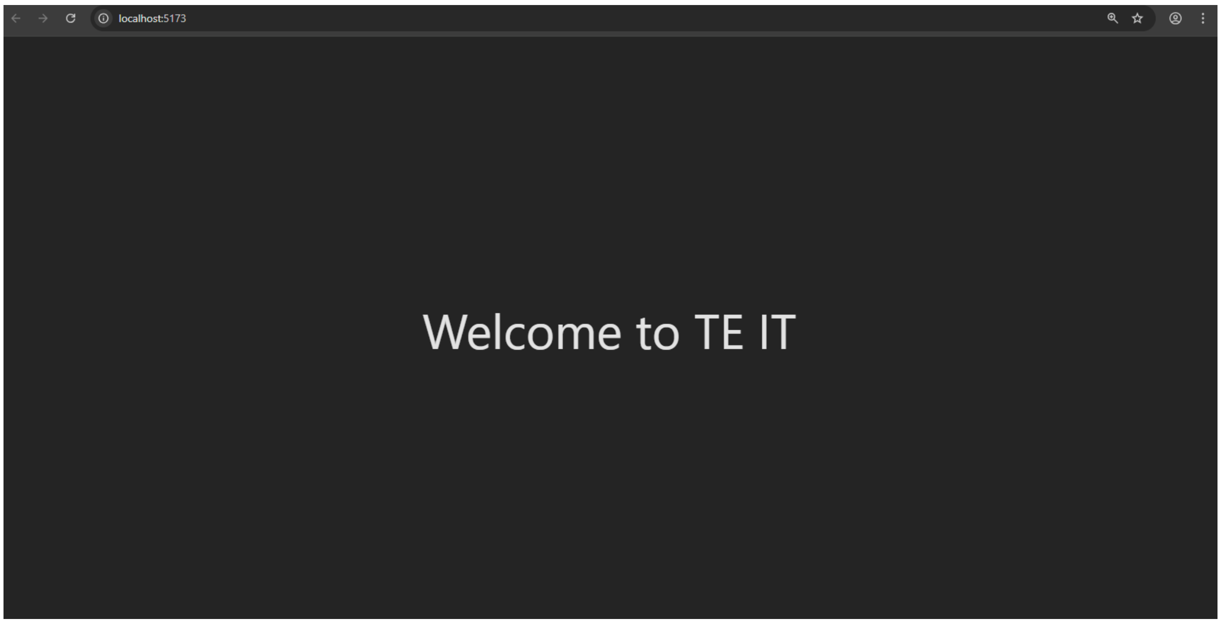
• componentWillUnmount()
This function is invoked before the component is finally unmounted from the DOM, i.e., this function gets invoked once before the component is removed from the page, and this denotes the end of the lifecycle.

Program:



```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        Welcome to TE IT
      </div>

    </>
  )
}

export default App

```

Output:



```
Welcome to TE IT
```

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)
Juhu Versova Link Road, Andheri (West), Mumbai-53

Experiment No:8

Aim: Write a program to demonstrate functional Component and class component.

Theory:

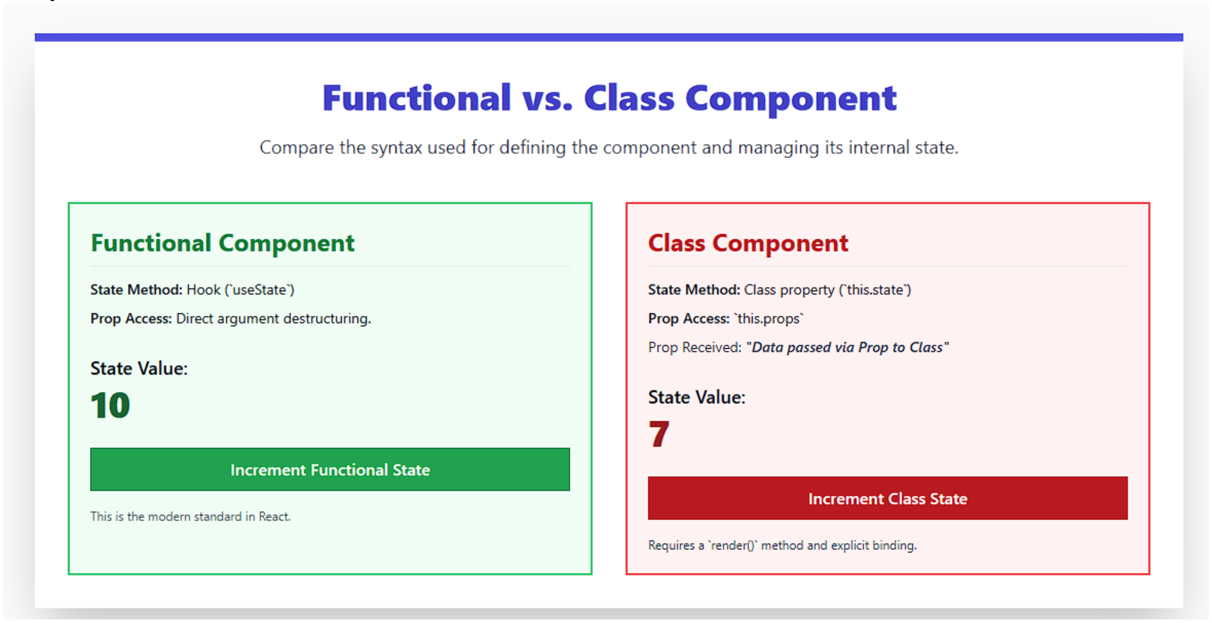
1. **Functional Component**
    - **Definition:** Simple JavaScript functions that accept props and return JSX.
    - **Modern State/Logic:** Use **Hooks** (useState, useEffect) to manage state and side effects.
    - **Pros:** Recommended for new development, less code, easier to read, and avoids the this keyword.
2. **Class Component**
    - **Definition:** ES6 classes that extend React.Component and require a render() method.
    - **State/Logic:** Use this.state, this.setState(), and traditional **Lifecycle Methods** (e.g., componentDidMount).
    - **Cons:** More complex, more verbose, and requires managing the this keyword.

Program:

App.js:

```
import React, { useState, Component } from 'react';

const FunctionalComponentDemo = ({ initialValue = 100 }) => {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);

  return (
    <div className="p-5 border-2 border-green-500 bg-green-50 min-h-[300px]">
      <h2 className="text-2xl font-bold mb-3 text-green-700 border-b pb-2">Functional
Component</h2>
      <p className="mb-2 text-sm">
        <span className="font-semibold">State Method:</span> Hook (`useState`)
      </p>
      <p className="mb-2 text-sm">
        <span className="font-semibold">Prop Access:</span> Direct argument destructuring.
      </p>

      <div className="my-6">
        <div className="text-lg font-medium">State Value:</div>
        <div className="text-4xl font-extrabold text-green-800">{count}</div>
      </div>

      <button
        onClick={increment}
        className="w-full bg-green-600 hover:bg-green-700 text-white font-medium py-2 px-4 border
border-green-800 transition"
      >
        Increment Functional State
      </button>
      <p className="mt-4 text-xs text-gray-600">
        This is the modern standard in React.
      </p>
    </div>
  );
};

class ClassComponentDemo extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
      message: this.props.initialMessage || "Initial Message"
    };
    this.increment = this.increment.bind(this);
  }

  increment() {
    this.setState({
      count: this.state.count + 1
    });
  }

  render() {
    const { initialMessage } = this.props;
    const { count } = this.state;

    return (
      <div className="p-5 border-2 border-red-500 bg-red-50 min-h-[300px]">
        <h2 className="text-2xl font-bold mb-3 text-red-700 border-b pb-2">Class Component</h2>
        <p className="mb-2 text-sm">
          <span className="font-semibold">State Method:</span> Class property (`this.state`)
        </p>
        <p className="mb-2 text-sm">
          <span className="font-semibold">Prop Access:</span> `this.props`
        </p>

        <p className="mb-2 text-sm text-gray-700">
          Prop Received: <span className="font-medium italic">"{initialMessage}"</span>
        </p>

        <div className="my-6">
          <div className="text-lg font-medium">State Value:</div>
          <div className="text-4xl font-extrabold text-red-800">{count}</div>
        </div>
        <button
          onClick={this.increment}
          className="w-full bg-red-600 hover:bg-red-700 text-white font-medium py-2 px-4 border
border-red-800 transition"
        >
          Increment Class State
        </button>
        <p className="mt-4 text-xs text-gray-600">
          Requires a `render()` method and explicit binding.
        </p>
      </div>
    );
  }
}

const App = () => {
  return (
    <div className="p-8 font-sans bg-gray-50 min-h-screen">
      <div className="max-w-6xl mx-auto bg-white p-8 shadow-2xl border-t-8 border-indigo-600">
        <h1 className="text-4xl font-extrabold mb-4 text-center text-indigo-700">
          Functional vs. Class Component
        </h1>
        <p className="mb-10 text-lg text-gray-700 text-center">
          Compare the syntax used for defining the component and managing its internal state.
        </p>

        <div className="grid md:grid-cols-2 gap-8">
          <FunctionalComponentDemo initialValue={42} />
          <ClassComponentDemo initialMessage="Data passed via Prop to Class" />
        </div>
      </div>
    </div>
  );
};

export default App;
```

Output:



Functional vs. Class Component

Compare the syntax used for defining the component and managing its internal state.

Functional Component

State Method: Hook (`useState`)

Prop Access: Direct argument destructuring.

State Value:

10

This is the modern standard in React.

Class Component

State Method: Class property (`this.state`)

Prop Access: `this.props`

Prop Received: *"Data passed via Prop to Class"*

State Value:

7

Requires a `render()` method and explicit binding.

Conclusion:

The key takeaway is that **Functional Components with Hooks** are the preferred and modern standard in React. They offer a simpler, cleaner syntax and now possess the full capability to manage **state** and **side effects** (useState, useEffect), which was once exclusive to Class Components. While **Class Components** use this.state and **Lifecycle Methods**, they are more verbose and are generally relegated to maintaining legacy code or specific use cases like **Error Boundaries**. The demonstration ultimately proves that the future of React development is overwhelmingly **functional**.

Experiment No:9

Aim: Write a program for State and Props.

Theory:

1. **State**
   In React, state is an internal data store that a component manages and can update over time, triggering the component to re-render and reflect the latest data. It is mutable and primarily used for data that changes due to user interaction or events.
2. **Props**
   In contrast, props are external inputs passed from a parent component to a child component. Props are immutable inside the child and allow components to be dynamic and reusable by rendering different data based on what they receive. Together, state and props are essential concepts for creating interactive and flexible React applications.

Program:

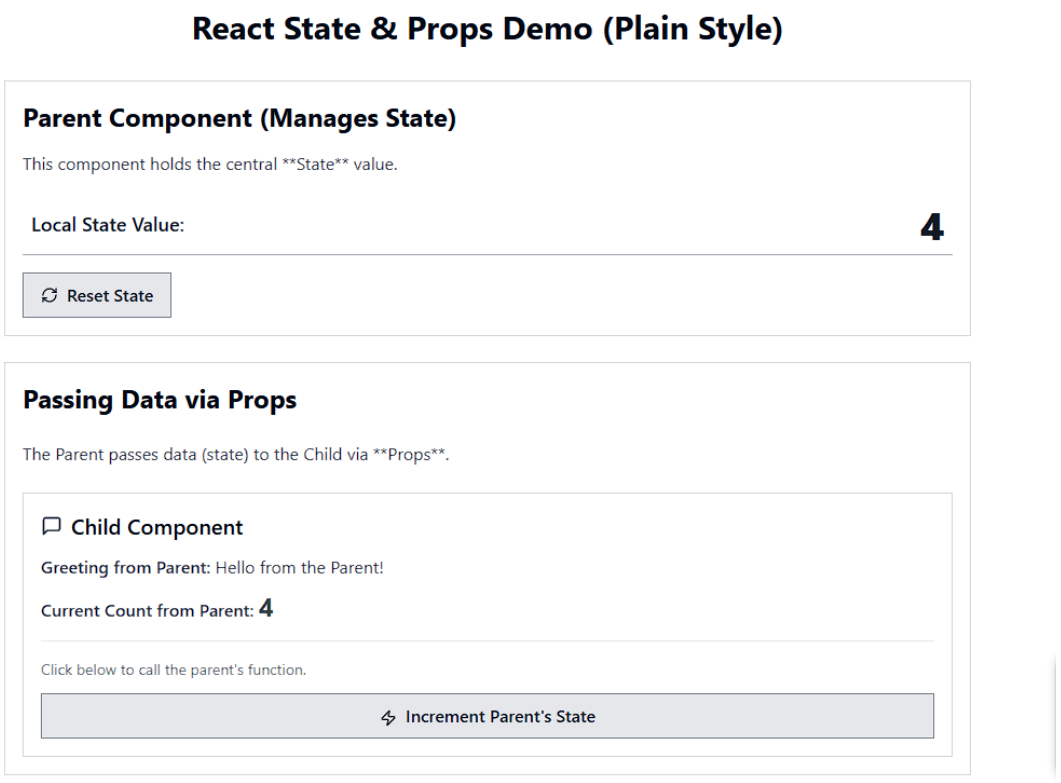
App.js:

```
import React, { useState } from "react";
const ChildComponent = (props) => {
  const { currentCount, greetingMessage, onIncrement } = props;

  return (
    <div className="p-4 border border-gray-400 border-2">
      <h2 className="text-xl font-medium mb-3">Child Component</h2>

      <p className="text-gray-700 mb-2">
        <span className="font-semibold">Greeting from Parent:</span>{" "}
        {greetingMessage}
      </p>
      <p className="text-gray-700 mb-4">
        <span className="font-semibold">Current Count from Parent:</span>{" "}
        <span className="text-2xl font-bold">{currentCount}</span>
      </p>

      <hr className="my-4" />

      <p className="text-sm text-gray-500 mb-3">
        Click below to call the parent's function.
      </p>
      <button
        onClick={onIncrement}
        className="w-full bg-gray-200 hover:bg-gray-300 text-gray-800 font-medium py-2 px-4 border
border-gray-500"
      >
        → Increment Parent's State
      </button>
    </div>
  );
};
```

```
const App = () => {
 const [count, setCount] = useState(0);

 const [message] = useState("Hello from the Parent!");

 const incrementCount = () => {
  setCount((prevCount) => prevCount + 1);
 };

 const resetCount = () => {
  setCount(0);
 };

 return (
  <div className="p-6 font-sans bg-blue-50 min-h-screen">
   <div className="max-w-4xl mx-auto bg-white p-6 border border-blue-500 border-4 shadow-
md">
    <h1 className="text-3xl font-bold mb-8 text-center">
     React State & Props Demo (Plain Style)
    </h1>

    <div className="p-4 mb-6 border border-gray-300">
     <h2 className="text-2xl font-bold mb-4">
      Parent Component (Manages State)
     </h2>
     <p className="text-gray-700 mb-4">
      This component holds the central **State** value.
     </p>

     <div className="flex items-center justify-between p-2 mb-4 border-b border-gray-400">
      <span className="text-lg font-medium text-gray-800">
       Local State Value:
      </span>
      <span className="text-4xl font-extrabold text-gray-900">
       {count}
      </span>
     </div>

     <button
      onClick={resetCount}
      className="bg-gray-200 hover:bg-gray-300 text-gray-800 font-medium py-2 px-4 border
border-gray-500"
     >
      ↺ Reset State
     </button>
    </div>
```

```
        <div className="p-4 border border-gray-300">
          <h2 className="text-2xl font-bold mb-6">Passing Data via Props</h2>

          <p className="text-gray-700 mb-6">
            The Parent passes data (state) to the Child via **Props**.
          </p>

          <div className="grid md:grid-cols-1 gap-6">
            <ChildComponent
              currentCount={count}
              greetingMessage={message}
              onIncrement={incrementCount}
            />
          </div>
        </div>
      </div>
    </div>
  );
};

export default App;
```

Output:



React State & Props Demo (Plain Style)

Parent Component (Manages State)

This component holds the central **State** value.

Local State Value: 4

Passing Data via Props

The Parent passes data (state) to the Child via **Props**.

Child Component

Greeting from Parent: Hello from the Parent!

Current Count from Parent: 4

Click below to call the parent's function.

Conclusion:

This program shows how React uses both class and functional components to build UI. Functional components are simpler and preferred today, while class components offer built-in support for state and lifecycle methods. Both can receive props and render dynamic content, making them essential for creating reusable and flexible React applications.

Experiment No. 10

Aim : Write a program for Login System by using Node.js with Express and MongoDB.

Theory :

Node.js is a JavaScript runtime environment that allows developers to run JavaScript on the server side, outside the browser. It is event-driven and non-blocking, meaning it can handle multiple requests simultaneously without slowing down the server. This makes Node.js highly efficient for building scalable web applications and APIs.

Express is a web application framework for Node.js that simplifies the process of creating web servers and APIs. It provides features for handling routes, HTTP methods (GET, POST, PUT, DELETE), and middleware, which can be used for authentication, validation, logging, and other tasks.

MongoDB is a NoSQL database that stores data in flexible JSON-like documents instead of traditional tables. This allows storing complex and dynamic user information such as name, email, and password. Passwords are stored in a hashed format using encryption libraries like bcrypt, ensuring security. MongoDB is highly scalable and works seamlessly with Node.js using Mongoose, which provides an Object Data Modeling (ODM) layer for easier interaction with the database. In a login system, MongoDB stores all registered users and their credentials, and retrieves user data during login and verification.

Installation steps of MongoDB :

Step 1: Download MongoDB Community Server
Go to the MongoDB Download Center to download the MongoDB Community Server.

Step 2: Install MongoDB
When the download is complete open the msi file and click the next button in the startup screen.
Now accept the End-User License Agreement and click the next button.
Now select the complete option to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the Custom option.

Step 3: Configure MongoDB Service
Select "Run service as Network Service user" and copy the path of the data directory. Click Next.
Click the Install button to start the MongoDB installation process.
After clicking on the install button installation of MongoDB begins.

Step 4: Complete Installation
Now click the Finish button to complete the MongoDB installation process.

Step 5: Set Environment Variables

Run MongoDB Server (mongod)

To run the MongoDB server (mongod) on Windows, follow these steps:

Step 1. Start MongoDB Service

Open the command prompt and run the following command:

- mongod

Program :

- **server.js**

```
const express = require("express");
const mongoose = require("mongoose");
const dotenv = require("dotenv");
const bodyParser = require("body-parser");
const cors = require("cors");

dotenv.config();

const app = express();

// Middleware
app.use(bodyParser.json());
app.use(cors());
const path = require("path");

// Serve static frontend files
app.use(express.static(path.join(__dirname, "public")));

// Routes
const authRoutes = require("./routes/auth");
app.use("/api/auth", authRoutes);

// Connect to MongoDB
mongoose.connect(process.env.MONGO_URI)
.then(() => console.log("MongoDB Connected"))
  .catch(err => console.log(err));
```

```
const PORT = process.env.PORT || 5000;
    app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

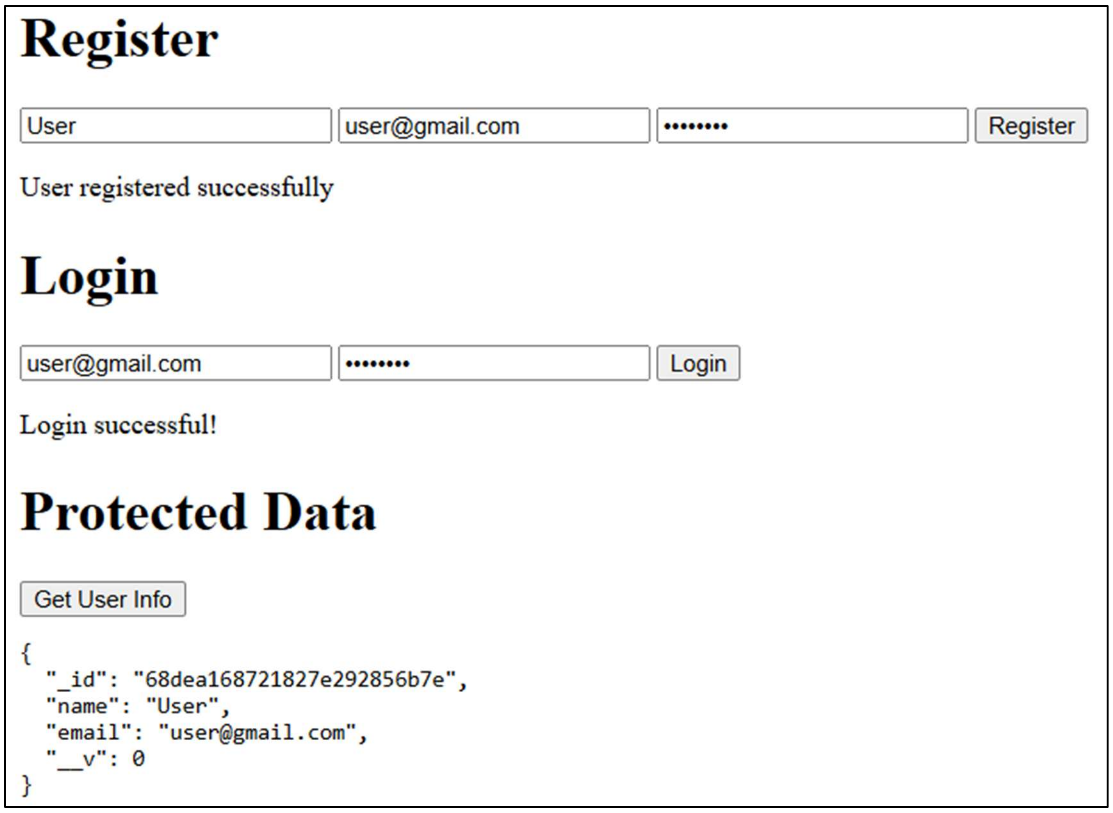
- **auth.js**

```
const express = require("express");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/User");

const router = express.Router();

// Register
router.post("/register", async (req, res) => {
   try {
      const { name, email, password } = req.body;

      let user = await User.findOne({ email });
      if (user) return res.status(400).json({ msg: "User already exists" });

      const hashedPassword = await bcrypt.hash(password, 10);

      user = new User({ name, email, password: hashedPassword });
      await user.save();

      res.json({ msg: "User registered successfully" });
   } catch (err) {
      res.status(500).send("Server error");
   }
});

// Login
router.post("/login", async (req, res) => {
   try {
      const { email, password } = req.body;

      const user = await User.findOne({ email });
      if (!user) return res.status(400).json({ msg: "Invalid credentials" });

      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) return res.status(400).json({ msg: "Invalid credentials" });

      const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: "1h" });

      res.json({ token });
   } catch (err) {
    res.status(500).send("Server error");
  }
});

const authMiddleware = require("../middleware/auth");

// Protected route example
router.get("/me", authMiddleware, async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select("-password"); // Exclude password
    res.json(user);
  } catch (err) {
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

- **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Login Demo</title>
</head>
<body>
  <h1>Register</h1>
  <input type="text" id="regName" placeholder="Name">
  <input type="email" id="regEmail" placeholder="Email">
  <input type="password" id="regPassword" placeholder="Password">
  <button onclick="register()">Register</button>
  <p id="regMsg"></p>

  <h1>Login</h1>
  <input type="email" id="loginEmail" placeholder="Email">
  <input type="password" id="loginPassword" placeholder="Password">
  <button onclick="login()">Login</button>
  <p id="loginMsg"></p>

  <h1>Protected Data</h1>
  <button onclick="getProtected()">Get User Info</button>
  <pre id="userData"></pre>
  <script>
let token = "
    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ZGU5NDdkMjg2M2Y5YmM
    yZjcxODk3ZSIsImlhdCI6MTc1OTQxNzUzNiwiZXhwIjoxNzU5NDIxMTM2fQ.hjCyH
    2zOhkk-g--zPdyf1oWOEM7rdQxTxCnwtdE1UjI"; // Store JWT token here

async function register() {
  const name = document.getElementById("regName").value;
  const email = document.getElementById("regEmail").value;
  const password = document.getElementById("regPassword").value;

  const res = await fetch("http://localhost:5000/api/auth/register", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name, email, password })
  });

  const data = await res.json();
  document.getElementById("regMsg").innerText = data.msg || JSON.stringify(data);
}

async function login() {
  const email = document.getElementById("loginEmail").value;
  const password = document.getElementById("loginPassword").value;

  const res = await fetch("http://localhost:5000/api/auth/login", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ email, password })
  });

  const data = await res.json();
  if (data.token) {
    token = data.token;
    document.getElementById("loginMsg").innerText = "Login successful!";
  } else {
    document.getElementById("loginMsg").innerText = data.msg || "Login failed";
  }
}

async function getProtected() {
  if (!token) {

 document.getElementById("userData").innerText = "Please login first!";
    return;
  }
 const res = await fetch("http://localhost:5000/api/auth/me", {
        method: "GET",
        headers: { "Authorization": "Bearer " + token }
    });

    const data = await res.json();
    document.getElementById("userData").innerText = JSON.stringify(data, null, 2);
      }</script>
</body>
</html>
```

Output :



Register

User | user@gmail.com | •••••••• | Register

User registered successfully

Login

user@gmail.com | •••••••• | Login

Login successful!

Protected Data

Get User Info

```
{
  "_id": "68dea168721827e292856b7e",
  "name": "User",
  "email": "user@gmail.com",
  "__v": 0
}
```

Conclusion :

By combining Node.js, Express, and MongoDB, a developer can build a secure, scalable, and efficient login system. Node.js provides the runtime for server-side JavaScript, Express simplifies server logic and routing, and MongoDB handles flexible storage of user data. Together, these technologies form a modern full-stack backend solution, capable of handling user registration, authentication, and secure access to protected resources.

Experiment No. 11

Aim : Write a program to implement Node.js using CRUD operations in REST API .

Theory :

CRUD stands for Create, Read, Update, and Delete. These are the four fundamental operations that are used to manage data in any application. For example, creating a new user account corresponds to Create, viewing user details corresponds to Read, updating a user's profile corresponds to Update, and removing a user corresponds to Delete.

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and principles that allows two systems to communicate over the internet. It uses standard HTTP methods such as POST for creating data, GET for reading data, PUT/PATCH for updating data, and DELETE for deleting data. REST APIs are widely used because they are lightweight, scalable, and easy to use across different platforms like web, mobile, or desktop applications.

Node.js is a powerful JavaScript runtime that allows developers to run JavaScript on the server side. Express.js is a popular web framework for Node.js that makes it easier to create REST APIs. MongoDB is a NoSQL database that stores data in a flexible, JSON-like format, making it a perfect choice to work with Node.js applications.

The aim of this project is to implement CRUD operations using Node.js, Express, and MongoDB in a REST API. This allows developers to perform operations such as creating, retrieving, updating, and deleting user information through HTTP requests.

Program :

- **server.js**

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();

// Middleware
app.use(bodyParser.json());

const path = require('path');
app.use(express.static(path.join(__dirname)));
// MongoDB connection
mongoose.connect('mongodb://127.0.0.1:27017/nodecrud', {
```

```
useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('MongoDB connected'))
.catch(err => console.log(err));

// Import User model
const User = require('./models/User');

// Routes
app.post('/users', async (req, res) => {
  try {

    const user = new User(req.body);
    await user.save();
    res.status(201).send(user);
  } catch (err) {
    res.status(400).send(err);
  }
});

app.get('/users', async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).send(users);
  } catch (err) {
    res.status(500).send(err);
  }
});
app.get('/users/:id', async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) return res.status(404).send({ message: 'User not found' });
    res.status(200).send(user);
  } catch (err) {
    res.status(500).send(err);
  }
});

app.put('/users/:id', async (req, res) => {
  try {
    const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true,
   runValidators: true });
    if (!user) return res.status(404).send({ message: 'User not found' });
    res.status(200).send(user);
  } catch (err) {
    res.status(400).send(err);
    }
});

app.delete('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndDelete(req.params.id);
        if (!user) return res.status(404).send({ message: 'User not found' });
        res.status(200).send({ message: 'User deleted successfully' });
    } catch (err) {
        res.status(500).send(err);
    }
});

const PORT = 3000;
      app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

- **User.js**

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    email: {
        type: String,
        required: true,
        unique: true,
        lowercase: true
    },
    age: {
        type: Number,
        required: true
    }
}, { timestamps: true });
      module.exports = mongoose.model('User', userSchema);
```

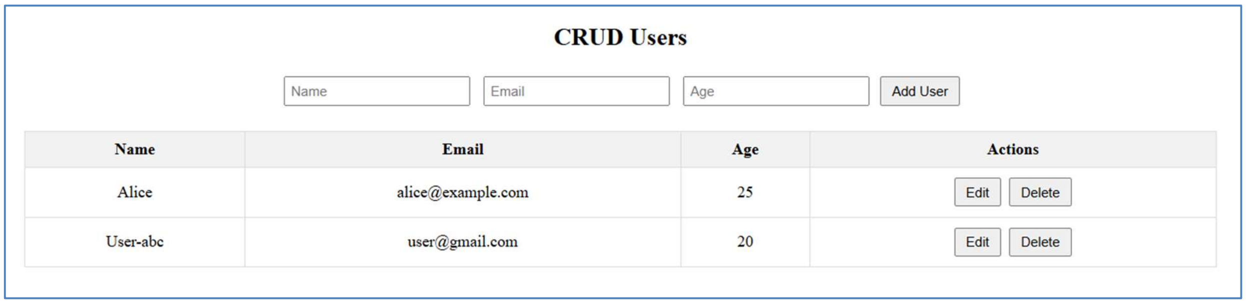
- **index.html**

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

```
    table { border-collapse: collapse; width: 70%; margin: 20px auto; }
    th, td { border: 1px solid #ddd; padding: 8px; text-align: center; }
    th { background-color: #f2f2f2; }
    input { padding: 5px; margin: 5px; }
    button { padding: 5px 10px; margin: 2px; }
    #formContainer { text-align:center; margin-bottom:20px; }
</style>
</head>
<body>

<h2 style="text-align:center">CRUD Users</h2>

<div id="formContainer">
    <input type="text" id="name" placeholder="Name">
    <input type="email" id="email" placeholder="Email">
    <input type="number" id="age" placeholder="Age">
    <button onclick="createUser()">Add User</button>
</div>

<table>
    <thead>
        <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Age</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody id="usersTable">
        <!-- Users will appear here -->
    </tbody>
</table>

<script>
const apiUrl = 'http://localhost:3000/users';

// Fetch and display all users
function fetchUsers() {
    fetch(apiUrl)
        .then(res => res.json())
        .then(users => {
            const table = document.getElementById('usersTable');
            table.innerHTML = '';
            users.forEach(user => {
                const row = document.createElement('tr');
                row.innerHTML = `
                <td>${user.name}</td>
                <td>${user.email}</td>
                <td>${user.age}</td>
                <td>
                    <button onclick="editUser('${user._id}', '${user.name}', '${user.email}',
        ${user.age})">Edit</button>
                    <button onclick="deleteUser('${user._id}')">Delete</button>
                </td>
            `;
            table.appendChild(row);
        });
    })
    .catch(err => console.error(err));
}

// Create a new user
function createUser() {
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;
    const age = Number(document.getElementById('age').value);

    fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ name, email, age })
    })
    .then(res => res.json())
    .then(() => {


        fetchUsers();
        document.getElementById('name').value = '';
        document.getElementById('email').value = '';
        document.getElementById('age').value = '';
    })
    .catch(err => console.error(err));
}

// Delete a user
function deleteUser(id) {
    fetch(`${apiUrl}/${id}`, { method: 'DELETE' })
        .then(() => fetchUsers())
        .catch(err => console.error(err));
}
```

```
// Edit a user
function editUser(id, name, email, age) {
   const newName = prompt("Enter new name:", name);
   const newEmail = prompt("Enter new email:", email);
   const newAge = prompt("Enter new age:", age);

   if (newName && newEmail && newAge) {
      fetch(`${apiUrl}/${id}`, {
         method: 'PUT',
         headers: { 'Content-Type': 'application/json' },
         body: JSON.stringify({ name: newName, email: newEmail, age: Number(newAge) })
      })
      .then(() => fetchUsers())
      .catch(err => console.error(err));
   }
}

// Initial fetch
fetchUsers();
</script>
</body>
</html>
```

Output :



CRUD Users

Name	Email	Age	Actions
Alice	alice@example.com	25	Edit Delete
User-abc	user@gmail.com	20	Edit Delete

Conclusion :

In conclusion, CRUD operations are the essential building blocks for managing data in applications. REST APIs provide a standard way of implementing these operations using HTTP methods, making them widely accepted in modern software development. By combining Node.js, Express, and MongoDB, we can build a backend system that is fast, scalable, and efficient for handling data. The above program demonstrates a simple REST API where we can create, read, update, and delete user records in a MongoDB database.