**The National Institute of Engineering, Mysuru**

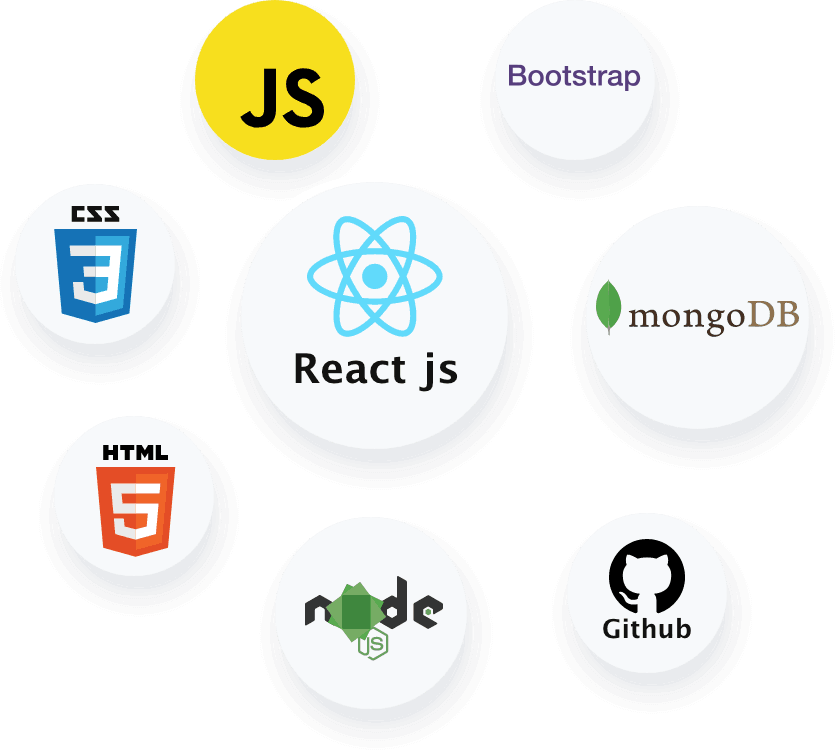
# Autonomous Institute, Affliated to VTU

North Campus No.50 (Part), Koorgalli village, Hootagalli Industrial Area, Mysuru-570018 Karnataka

**Department of Computer Science and Engineering**

**Laboratory Manual For**

Full Stack Development – BCSL504



**Coordinators:**

***Mrs. Padmini M S***

***Mrs. Mahe Mubeen Akhtar***

***Ms. Divyashree R***

BE (Ⅲ YEAR - Ⅴ SEM)

**2024-2025**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Expt. No** | **Experiment Title** | **Page No** |
| 1 | 1. a. Design a html page as shown below. (Book Information)     b. Write a HTML program for the demonstration of Lists.  (i)Unordered List.  (ii)Ordered List.  (iii)Descriptive List. | 1 - 2 |
| 2 | Create the below registration form using forms in HTML. | 3 - 5 |
| 3 | Build a responsive web application for shopping cart with Registration, login, catalog and cart pages using CSS features. | 6 - 14 |
| 4 | Write JavaScript to validate the following fields of the Registration page.   1. First Name (Name should contains alphabets and the length Should not be less than 6 characters). 2. Password (Password should not be less than 6 characters length). 3. E-mail id (should not contain any invalid and must follow the Standard pattern name@domain.com) 4. Mobile Number (Phone number should contain 10 digits only). 5. Last Name and Address (should not be Empty). | 15 – 21 |
| 5 | Develop and demonstrate JavaScript with POP-UP boxes and functions for the following problems:   1. Input: Click on Display Date button using on click( ) function Output: Display date in the textbox. 2. Input: A number n obtained using prompt Output: Factorial of n number using alert. | 22 – 27 |
| 6 | Write a JavaScript program to generate a for loop that will iterate from 0 to 15. For each iteration, it will check if the current number is odd or even, and display a message to the screen using map and filters. | 28 - 29 |

|  |  |  |
| --- | --- | --- |
| 7 | Write a JavaScript program that modifies the DOM to change the content of an HTML element. Add event listeners to handle button clicks and dynamically change text on the page | 30 - 32 |
| 8 | Write a JavaScript program that handles events triggered by the body, button, text box, and password field | 33 - 34 |
| 9 | Write a JavaScript program to find HTML elements using various methods in JavaScript, such as by id, tag name, class name, CSS selectors, and HTML object collections. | 35 - 36 |
| 10 | Create a React application that demonstrates the use of state and props. Implement a parent component that maintains state and passes it down to child components via props. Show how changes in state affect the rendering of the child components | 37 - 39 |
| 11 | Develop a React program that utilizes hooks, such as useState and useEffect, within a functional component. | 40 - 41 |
| 12 | Write a React program that demonstrates the component life cycle phases mounting, updating and unmounting. | 42 - 46 |
| 13 | Write a React program that implements error handling using error boundaries. | 47 - 49 |
| 14 | Write a React program that integrates React Router into a React application. | 50 - 52 |
| 15 | Write a program that connects to MongoDB using Nodejs and performs basic query. | 53 - 60 |
| 16 | Describe the following operations in MongoDB.  **a)**     switch to a new database called **companyDB.**  b)     Create a new collection called **employees.**  c)     Insert a few documents into the **employees** collection  d)     Query all documents in the **employees** collection  e)     Query documents where the age is greater than 30  f)      Insert a new document into the **employees** collection  g)     Find a document where the name is "Alice"  h)     Update the age of the employee named "Bob"  i)      Delete the employee named "Charlie"  j)      Retrieve only the name and department fields for all employees  k)     Calculate the average age of employees in each department | 61 - 64 |
| 17 | Demonstrate the following operations using  Mongo shell script  a)     switch to the schoolDB database, create a teachers collection, and insert documents with fields name, subject, and experience (in years).  b)     Use the Mongo shell to export the students collection to a JSON file.  Import the JSON file back into a new collection named studentsBackup. | 65 - 68 |
| 18 | Create and demonstrate the following in Mongo DB  a)     Projection operators ($, $elematch and $slice)  b)     Execute query selectors (Any one comparison selectors, Any one logical selectors)  c)     Execute Aggregation operations ($avg, $min, $max, $push, $addToSet.) | 69 - 74 |

# Create a html page as shown below. (Book Information)

**Book Information**

Author Title Price

Reset

Submit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width">

<title>BookInfo</title>

<style> label {

display: inline-block; width: 60px;

text-align: left;

}

</style>

</head>

<body>

<h1>Book Information</h1>

<form>

<label for="author">Author</label>

<input type="text" id="author">

<br><br>

<label for="title">Title</label>

<input type="text" id="title">

<br><br>

<label for="price">Price</label>

<input type="number" id="price" min="0" step="0.1">

<br><br>

<button type="submit" style="margin-right: 120px">Submit</button>

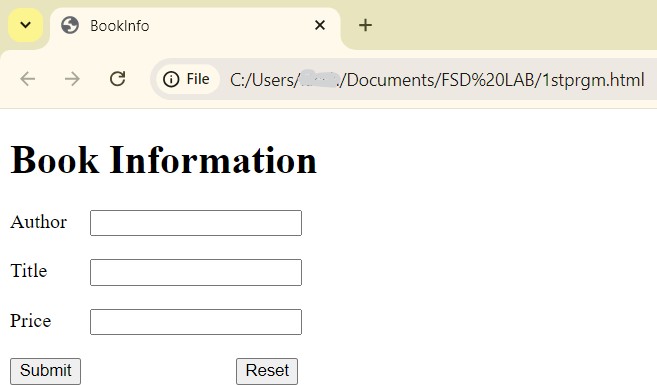
<button type="reset">Reset</button>

</form>

</body>

</html>

# OUTPUT:



1.b. Write a HTML program for the demonstration of Lists.

(i)Unordered List.

(ii)Ordered List.

(iii)Descriptive List.

<!DOCTYPE html>.

<html>

   <head>

      <title>HTML Unordered List</title>

   </head>

   <body>

   <h2>Unordered List </h2>

      <ul>

         <li>Apple</li>

         <li>Mango</li>

         <li>Banana</li>

         <li>Grapes</li>

         <li>Orange</li>

      </ul>

      <h2>Ordered List</h2>

      <ol type="i">

          <li>Apple</li>

          <li>Mango</li>

         <li>Banana</li>

     </ol>

     <h2>Descriptive List </h2>

     <dl>

        <dt><b>Apple</b></dt>

        <dd>A red colored fruit</dd>

        <dt><b>Honda</b></dt>

        <dd>A brand of a car</dd>

        <dt><b>Spinach</b></dt>

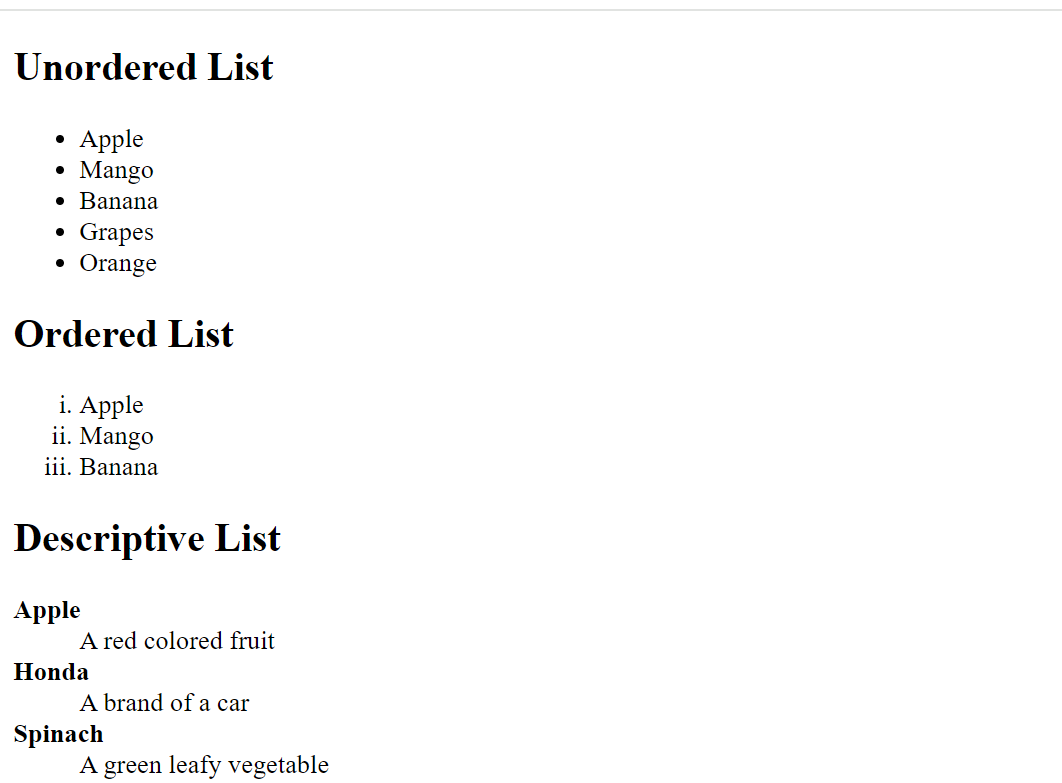
        <dd>A green leafy vegetable</dd>

     </dl>

   </body>

</html>

# OUTPUT:



1. **Create the below registration form using forms in HTML.**



**User Information Form**

Name Email Age Country

Password

Resume Hobbies Gender City

Address

Cricket

Male

No file chosen

Football Female

Reset

Submit

--Choose city--

Choose File

India **▼**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width">

<title>User Information</title>

<style> label {

display: inline-block; width: 70px;

}

</style>

</head>

<body>

<h1>User Information Form</h1>

<label for="name">Name</label>

<input type="text" id="name">

<br><br>

<label for="email">Email</label>

<input type="email" id="email">

<br><br>

<label for="age">Age</label>

<input type="number" id="age">

<br><br>

<label for="country">Country</label>

<select id="country">

<option selected>India</option>

<option>USA</option>

<option>UK</option>

</select>

<br><br>

<label for="password">Password</label>

<input type="password" id="password">

<br><br>

<label for="resume">Resume</label>

<input type="file" id="resume">

<br><br>

<label>Hobbies</label>

<input type="checkbox" id="cricket">

<label for="cricket">Cricket</label>

<input type="checkbox" id="football">

<label for="football">Football</label>

<br><br>

<label>Gender</label>

<input type="radio" id="male" name="gender">

<label for="male">Male</label>

<input type="radio" id="female" name="gender">

<label for="female">Female</label>

<br><br>

<label for="city">City</label>

<select id="city">

<option selected disabled hidden>--Choose city--</option>

<option>Mysuru</option>

<option>Bengaluru</option>

</select>

<br><br>

<label for="address" style="display: block">Address</label>

<textarea id="address" cols="30" rows="5"></textarea>

<br><br>

<button type="submit">Submit</button>

<button type="reset">Reset</button>

</body>

</html>

# OUTPUT:



1. **Build a responsive web application for shopping cart with Registration, login, catalog and cart pages using CSS features.**

**Index.html – *Catalog page***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>Catalog</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="index.html">Catalog</a></li>

<li><a href="cart.html">Cart</a></li>

<li><a href="login.html">Login</a></li>

<li><a href="register.html">Register</a></li>

</ul>

</nav>

</header>

<main>

<section class="catalog">

<h1>Catalog</h1>

<div class="product">

<img src="product1.jpg" alt="Product 1">

<h2>Product 1</h2>

<p>1000.00</p>

<button>Add to Cart</button>

</div>

<div class="product">

<img src="product2.jpg" alt="Product 2">

<h2>Product 2</h2>

<p>2000.00</p>

<button>Add to Cart</button>

</div>

<!-- Add more products here -->

</section>

</main>

<footer>

<p>&copy; 2024 My Shopping Cart</p>

</footer>

</body>

</html>

**Cart.html – *Cart page***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>Shopping Cart</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="index.html">Catalog</a></li>

<li><a href="cart.html">Cart</a></li>

<li><a href="login.html">Login</a></li>

<li><a href="register.html">Register</a></li>

</ul>

</nav>

</header>

<main>

<section class="cart">

<h1>Shopping Cart</h1>

<div class="cart-item">

<h2>Product 1</h2>

<p>100.00</p>

<button>Remove</button>

</div>

<div class="cart-item">

<h2>Product 2</h2>

<p>200.00</p>

<button>Remove</button>

</div>

<!-- Add more cart items here -->

<button class="checkout">Checkout</button>

</section>

</main>

<footer>

<p>&copy; 2024 My Shopping Cart</p>

</footer>

</body>

</html>

**Login.html – *Login page***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="index.html">Catalog</a></li>

<li><a href="cart.html">Cart</a></li>

<li><a href="login.html">Login</a></li>

<li><a href="register.html">Register</a></li>

</ul>

</nav>

</header>

<main>

<section class="login">

<h1>Login</h1>

<form>

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password"

required>

<br>

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

</form>

</section>

</main>

<footer>

<p>&copy; 2024 My Shopping Cart</p>

</footer>

</body>

</html>

**Register.html – *Registration page***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>Register</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="index.html">Catalog</a></li>

<li><a href="cart.html">Cart</a></li>

<li><a href="login.html">Login</a></li>

<li><a href="register.html">Register</a></li>

</ul>

</nav>

</header>

<main>

<section class="register">

<h1>Register</h1>

<form>

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password"

required>

<br>

<button type="submit">Register</button>

</form>

</section>

</main>

<footer>

<p>&copy; 2024 My Shopping Cart</p>

</footer>

</body>

</html>

**Style.css – *CSS file***

/\* Reset some default styles \*/ body, h1, h2, p, ul, li {

margin: 0;

padding: 0;

box-sizing: border-box;

}

/\* Basic styling \*/ body {

font-family: 'Times New Roman', Times, serif, sans-serif; line-height: 1.6;

}

header {

background: rgb(2,0,36);

background: linear-gradient(9deg, rgba(2,0,36,1) 0%, rgba(21,101,133,1) 46%, rgba(0,199,239,1) 100%);

color: #fff; padding: 1rem;

}

nav ul {

list-style: none; display: flex;

justify-content: center;

}

nav ul li {

margin: 0 1rem;

}

nav ul li a { color: #fff;

text-decoration: none;

}

main {

padding: 1rem;

}

.catalog, .cart, .login, .register { max-width: 1200px;

margin: 0 auto;

}

.product, .cart-item { border: 1px solid #ddd; padding: 1rem;

margin-bottom: 1rem;

}

.product img {

max-width: 12%; height: auto;

}

button {

background: #156585; color: #fff;

border: none;

padding: 0.5rem 1rem; cursor: pointer;

}

button:hover {

background: rgb(1, 151, 181);

}

form {

display: flex;

flex-direction: column;

}

form label {

margin-top: 1rem;

}

form input { padding: 0.5rem;

margin-top: 0.5rem;

}

footer {

background: rgb(2,0,36);

background: linear-gradient(9deg, rgba(2,0,36,1) 0%, rgba(21,101,133,1) 46%, rgba(0,199,239,1) 100%);

color: #fff;

text-align: center; padding: 1rem; position: absolute; bottom: 0;

width: 100%;

}

/\* Responsive Styles \*/ @media (max-width: 768px) {

nav ul {

flex-direction: column;

}

nav ul li {

margin: 0.5rem 0;

}}

# OUTPUT 1:

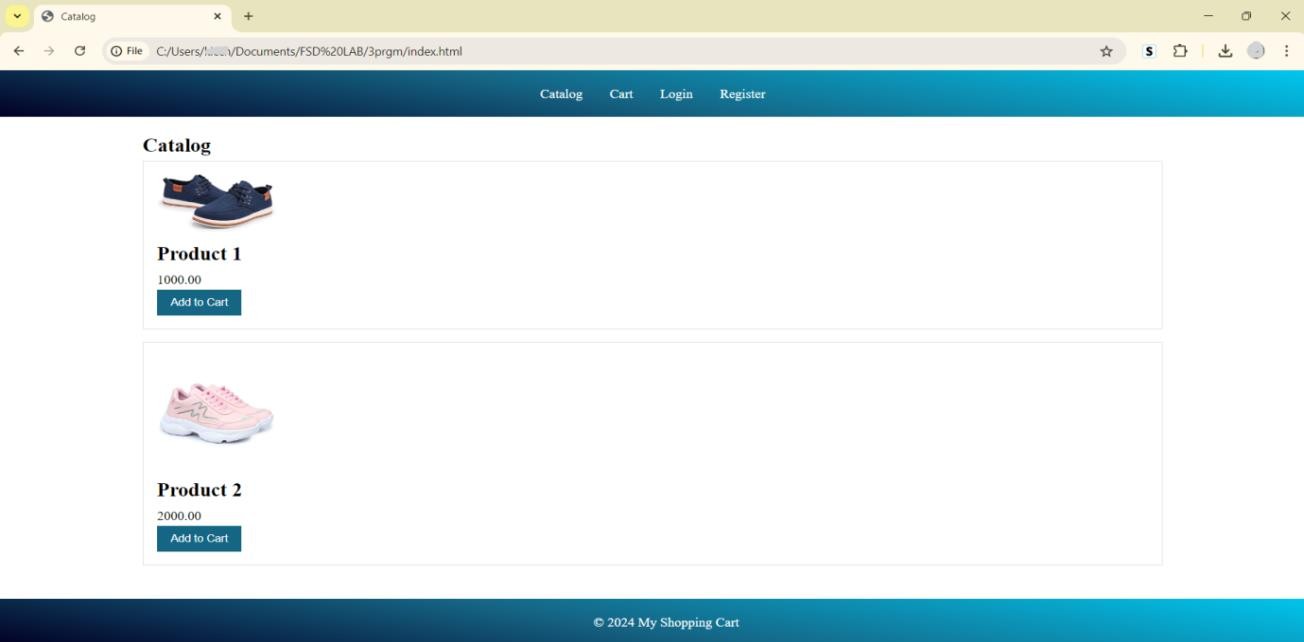


Fig: Catalog page

# OUTPUT 2:

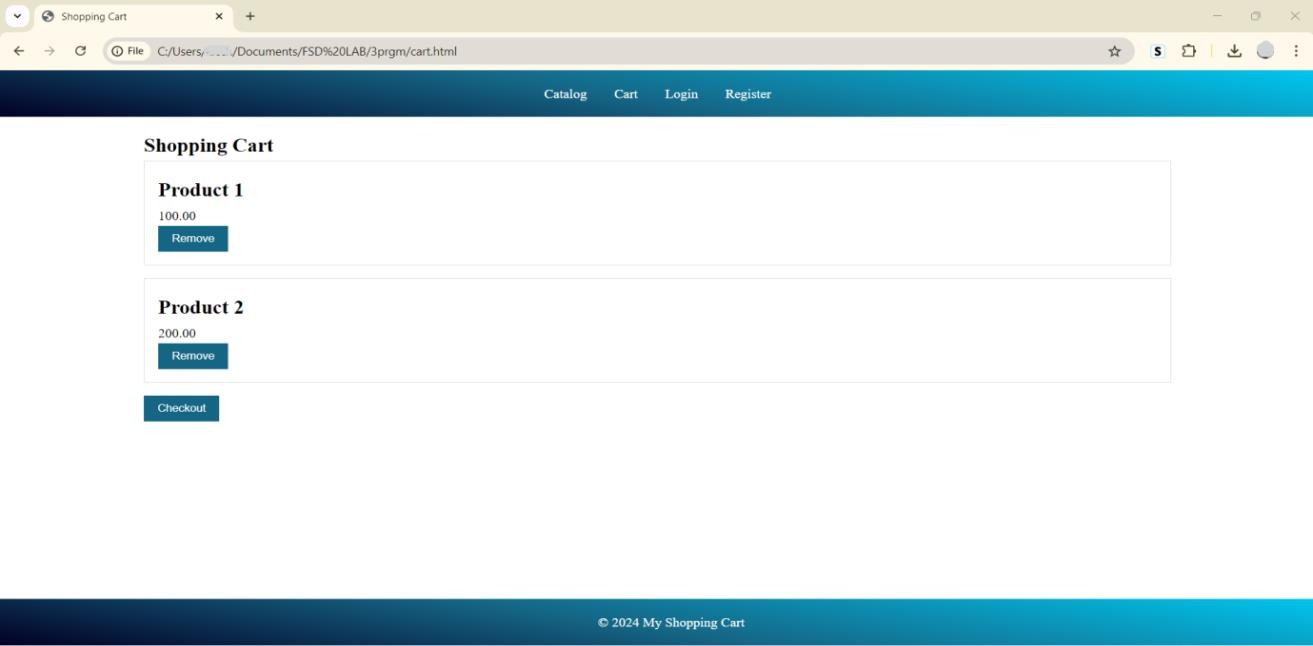


Fig: Cart page

# OUTPUT 3:

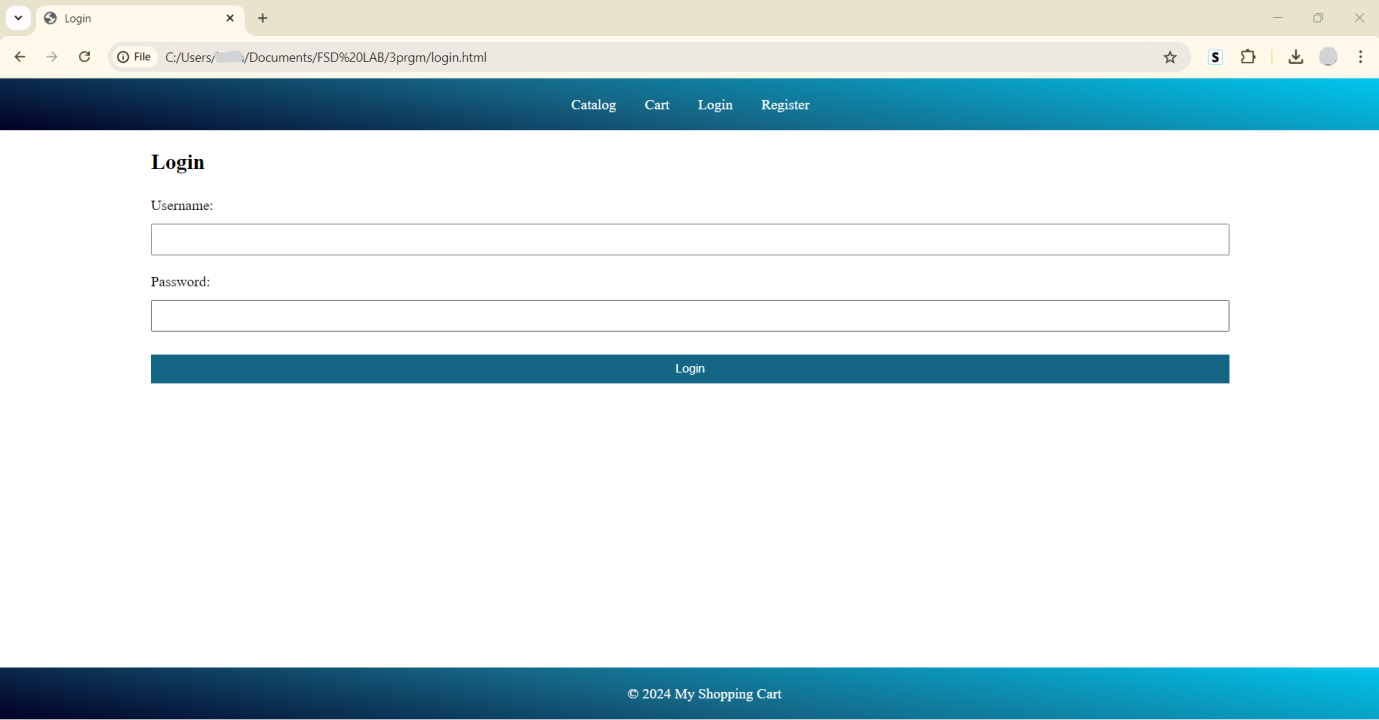


Fig: Login page

# OUTPUT 4:

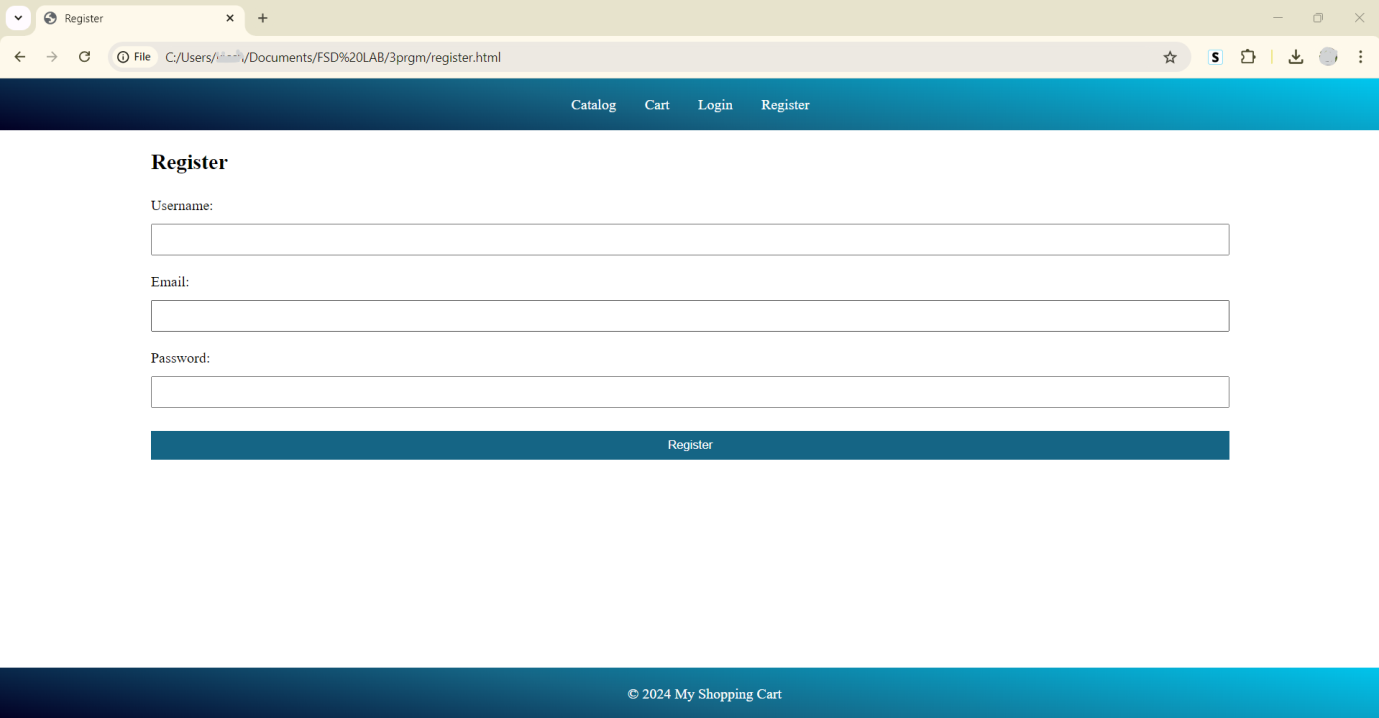


Fig: Registration page

# Write JavaScript to validate the following fields of the Registration page.

* 1. **First Name (Name should contains alphabets and the length Should not be less than 6 characters).**

# Password (Password should not be less than 6 characters length).

* 1. **E-mail id (should not contain any invalid and must follow the**

# Standard pattern name@domain.com)

* 1. **Mobile Number (Phone number should contain 10 digits only).**

# Last Name and Address (should not be Empty).

**Index.html – *Registration page***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>Registration Form</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Registration Form</h2>

<form id="registrationForm">

<div class="form-group">

<label for="firstName">First Name:</label>

<input type="text" id="firstName" name="firstName" placeholder="Enter your first name">

</div>

<div class="form-group">

<label for="lastName">Last Name:</label>

<input type="text" id="lastName" name="lastName" placeholder="Enter your last name">

</div>

<div class="form-group">

<label for="address">Address:</label>

<textarea id="address" name="address" placeholder="Enter your address"></textarea>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password" name="password" placeholder="Enter your password">

</div>

<div class="form-group">

<label for="email">E-mail:</label>

<input type="email" id="email" name="email" placeholder="Enter your email">

</div>

<div class="form-group">

<label for="mobileNumber">Mobile Number:</label>

<input type="text" id="mobileNumber" name="mobileNumber" placeholder="Enter your mobile number">

</div>

<button type="submit">Register</button>

<div id="errorMessages" class="error"></div>

</form>

</div>

<script src="validation.js"></script>

</body>

</html>

# Style.css

body {

font-family: Arial, sans-serif; background-color: #f4f4f4; display: flex;

justify-content: center; align-items: center; height: 100vh;

margin: 0;

}

.container {

background-color: #fff; padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); max-width: 500px;

width: 100%;

}

h2 {

margin-bottom: 20px; text-align: center;

}

.form-group {

margin-bottom: 15px;

}

.form-group label { display: block; margin-bottom: 5px; font-weight: bold;

}

.form-group input, .form-group textarea { width: 100%;

padding: 10px;

border: 1px solid #ddd; border-radius: 4px;

box-sizing: border-box;

}

.form-group textarea { resize: vertical; height: 100px;

}

button {

background-color: #28a745; color: white;

border: none; padding: 10px 20px; border-radius: 4px; cursor: pointer;

font-size: 16px;

}

button:hover {

background-color: #218838;

}

.error {

color: red;

margin-top: 10px;

}

# Validation.js

document.getElementById('registrationForm').addEventListener('submit', function(event) {

// Prevent form submission for validation event.preventDefault();

// Clear previous error messages

var errorDisplay = document.getElementById('errorMessages'); errorDisplay.innerHTML = '';

// Retrieve form values

var firstName = document.getElementById('firstName').value.trim(); var password = document.getElementById('password').value.trim(); var email = document.getElementById('email').value.trim();

var mobileNumber = document.getElementById('mobileNumber').value.trim();

var lastName = document.getElementById('lastName').value.trim(); var address = document.getElementById('address').value.trim();

// Regular expressions for validation var nameRegex = /^[A-Za-z]+$/;

var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; var phoneRegex = /^\d{10}$/;

// Validation messages var errors = [];

// Validate First Name

if (!nameRegex.test(firstName) || firstName.length < 6) { errors.push("First Name should contain only alphabets and be at least 6

characters long.");

}

// Validate Password

if (password.length < 6) {

errors.push("Password should be at least 6 characters long.");

}

// Validate Email

if (!emailRegex.test(email)) {

errors.push("Invalid email format. Please enter a valid email address (e.g., name@domain.com).");

}

// Validate Mobile Number

if (!phoneRegex.test(mobileNumber)) {

errors.push("Mobile Number should contain exactly 10 digits.");

}

// Validate Last Name and Address if (lastName === '') {

errors.push("Last Name should not be empty.");

}

if (address === '') {

errors.push("Address should not be empty.");

}

// Display errors if any if (errors.length > 0) {

errorDisplay.innerHTML = errors.join('<br>');

} else {

// Proceed with form submission or any further processing alert("Form submitted successfully!");

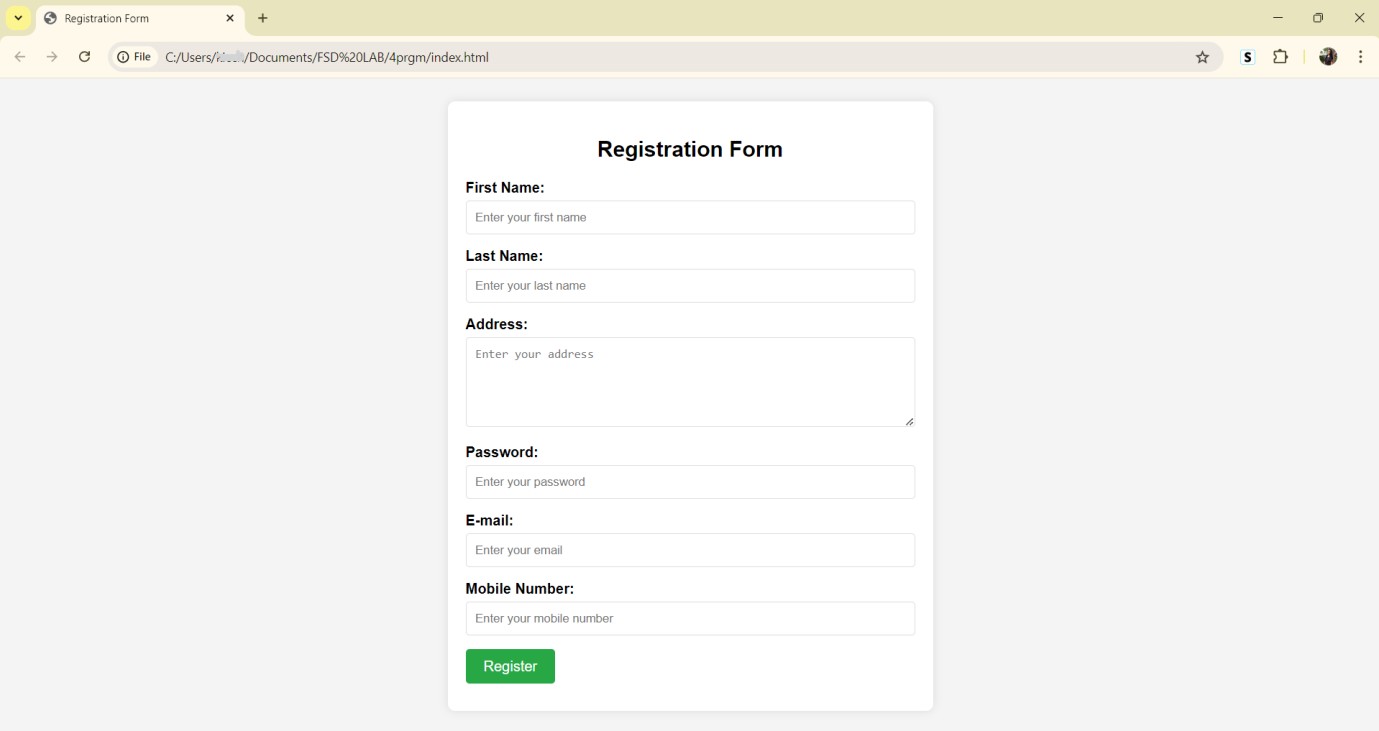
// Uncomment the following line to actually submit the form if desired

// this.submit();

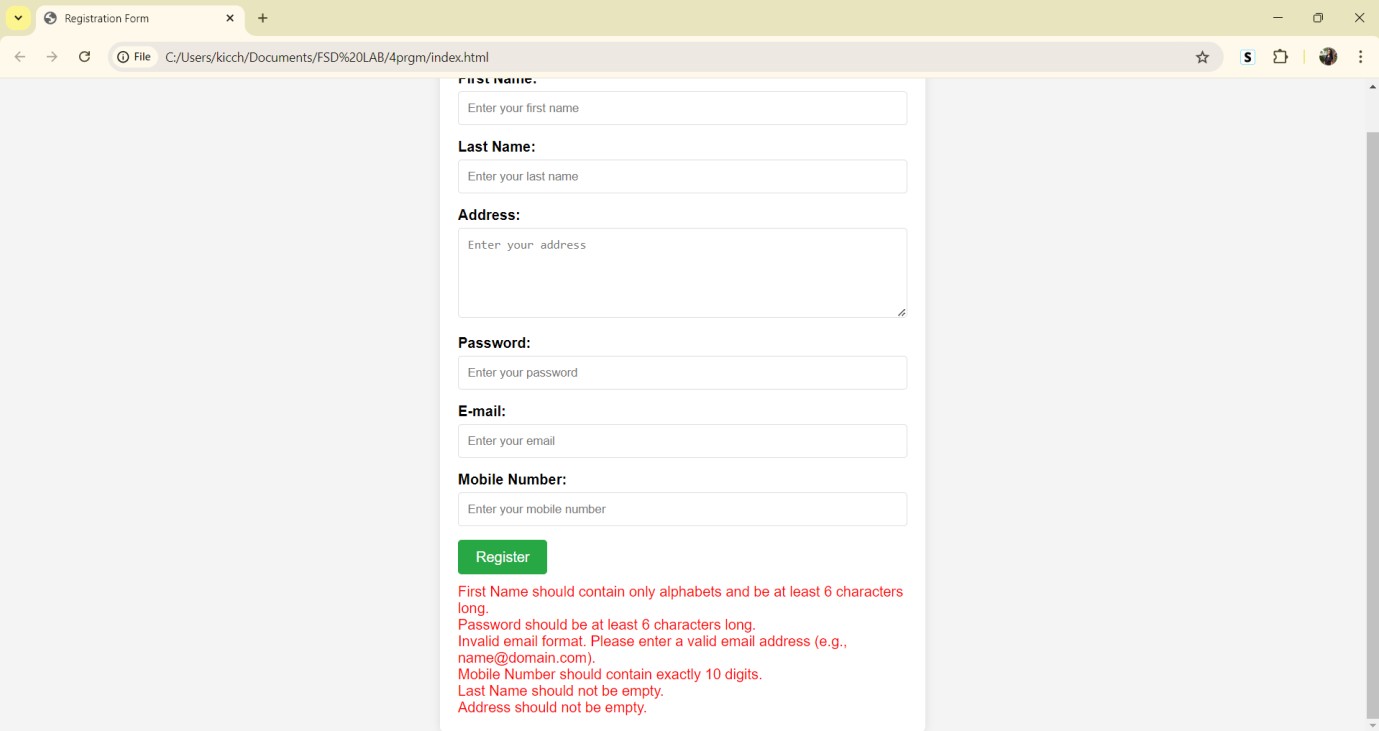
}

});

# OUTPUT 1:



**OUTPUT 2:**



# Develop and demonstrate JavaScript with POP-UP boxes and functions for the following problems:

1. **Input: Click on Display Date button using on click( ) function Output: Display date in the textbox.**

# Input: A number n obtained using prompt Output: Factorial of n number using alert.

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial- scale=1.0">

<title>JavaScript Pop-up Boxes and Functions</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Displaying Date</h2>

<!-- Section for Displaying Date -->

<div class="section">

<button class="btn" onclick="displayDate()">Display Date</button>

<input type="text" id="dateTextbox" class="textbox" readonly>

</div>

<hr>

<!-- Section for Factorial Calculation -->

<div>

<h2>Factorial Calculation</h2>

<div class="section">

<button class="btn" onclick="calculateFactorial()">Calculate Factorial</button>

</div>

</div>

</div>

<script>

// Function to display the current date in the textbox function displayDate() {

var today = new Date();

var dateString = today.toDateString(); // Formats date as "Day Mon Date Year"

document.getElementById('dateTextbox').value = dateString;

}

// Function to calculate the factorial of a number function calculateFactorial() {

var n = prompt("Enter a number to calculate its factorial:"); n = parseInt(n, 10); // Convert input to an integer

if (isNaN(n) || n < 0) {

alert("Please enter a valid non-negative integer."); return;

}

var factorial = 1;

for (var i = 1; i <= n; i++) { factorial \*= i;

}

alert("The factorial of " + n + " is " + factorial + ".");

}

</script>

</body>

</html>

# Styles.css

body {

font-family: Arial, sans-serif; background-color: #03203e; display: flex;

justify-content: center; align-items: center; height: 100vh; margin: 0;

}

.container {

background-color: #f0f2f6; border-radius: 8px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1); padding: 20px;

max-width: 500px; width: 100%;

}

h2 {

margin-bottom: 20px; text-align: center; color: #343a40;

}

.section {

margin-bottom: 20px; text-align: center;

}

.btn {

background-color: #007bff; color: white;

border: none; padding: 10px 20px; border-radius: 5px; font-size: 16px; cursor: pointer;

transition: background-color 0.3s, transform 0.2s;

}

.btn:hover {

background-color: #0056b3;

}

.btn:active {

transform: scale(0.98);

}

.textbox { width: 80%;

padding: 10px;

border: 1px solid #ced4da; border-radius: 5px;

font-size: 16px; margin-top: 10px;

box-sizing: border-box;

}

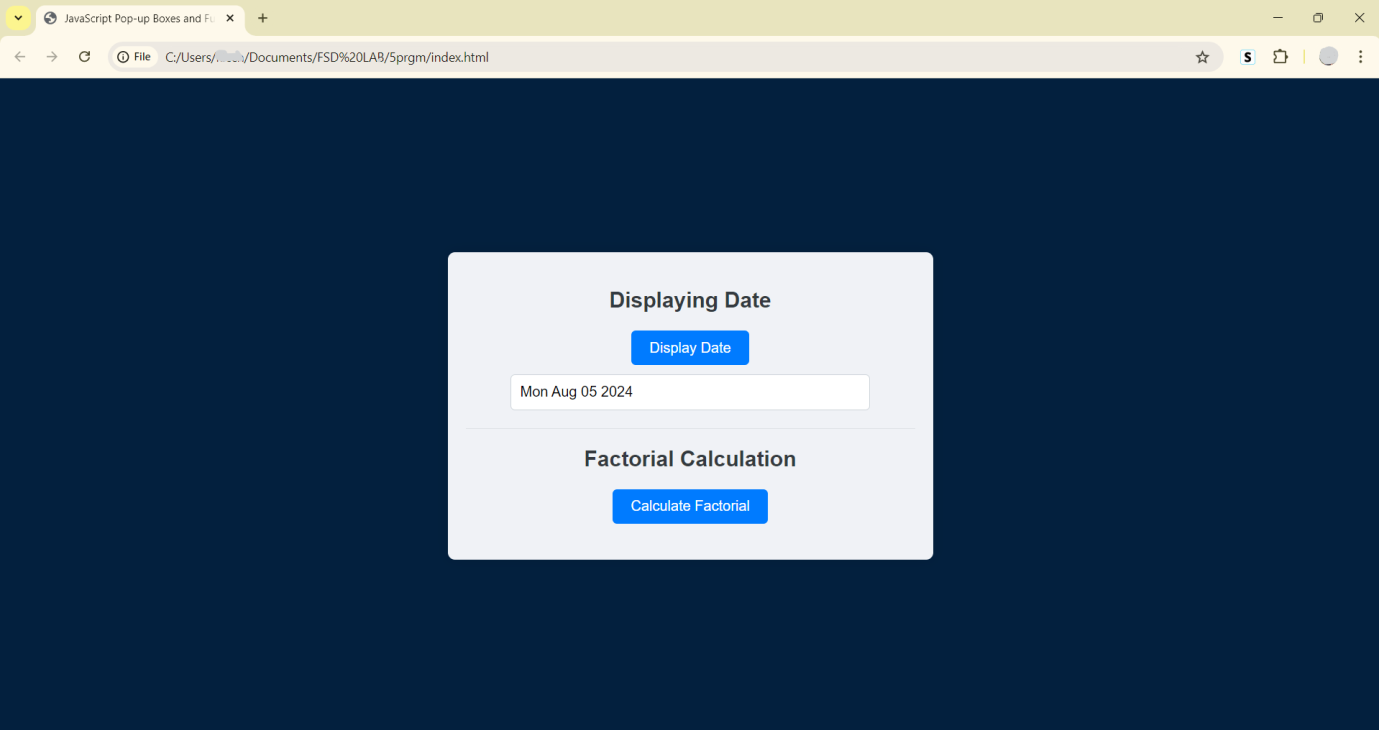
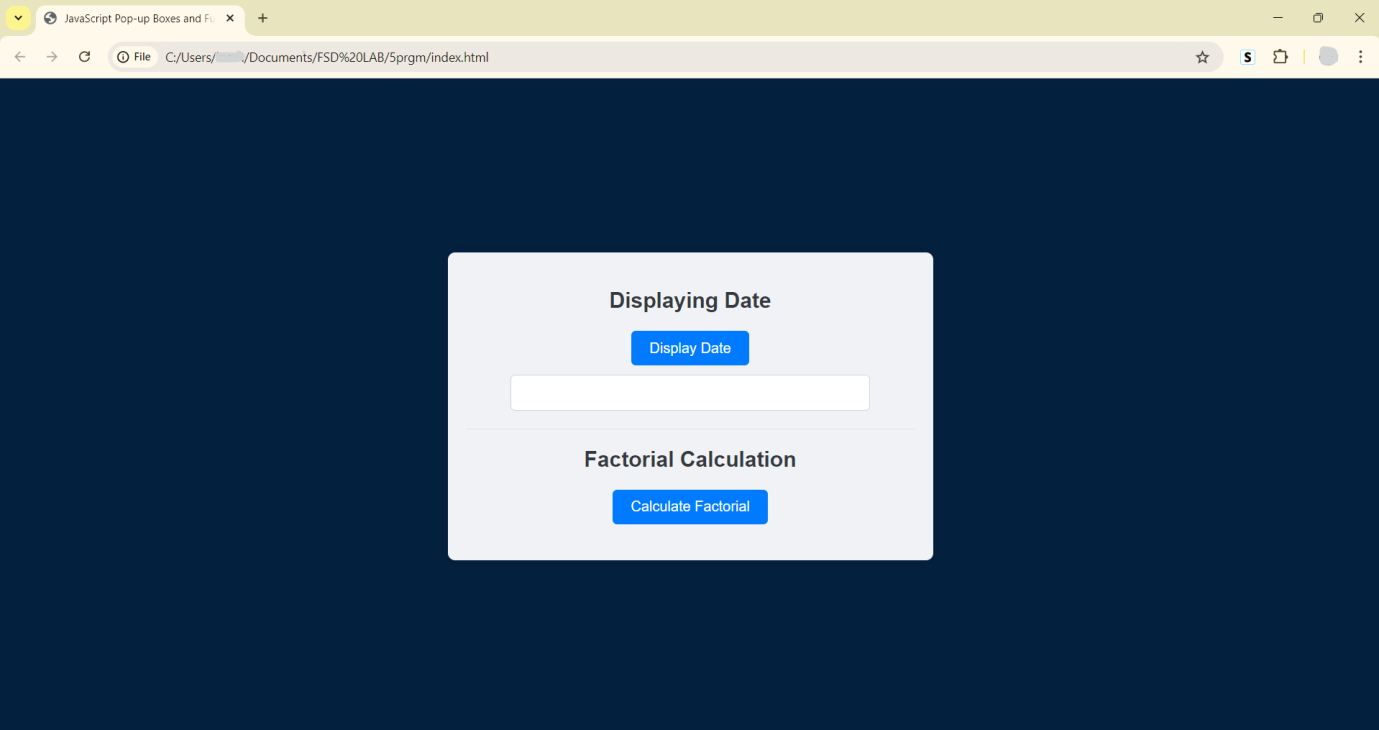
hr {

border: 0;

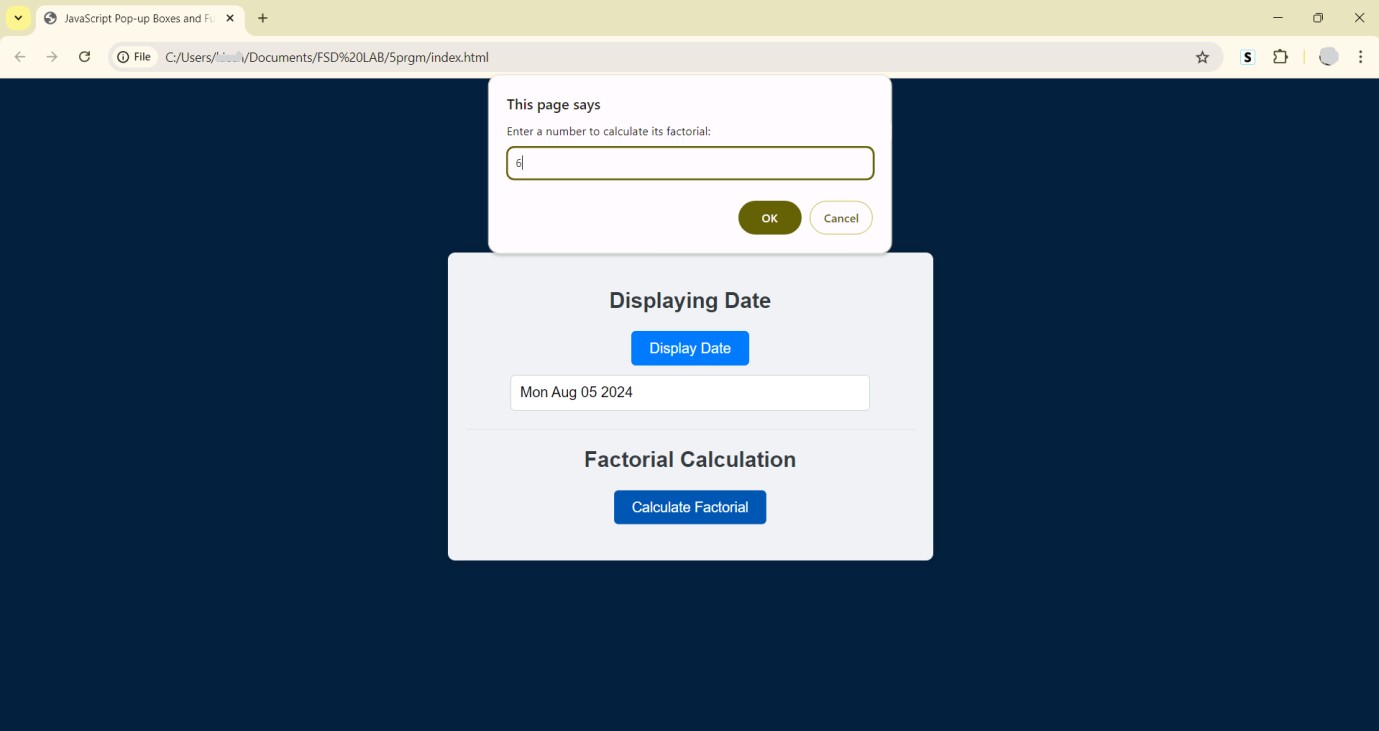
border-top: 1px solid #dee2e6; margin: 20px 0;

}

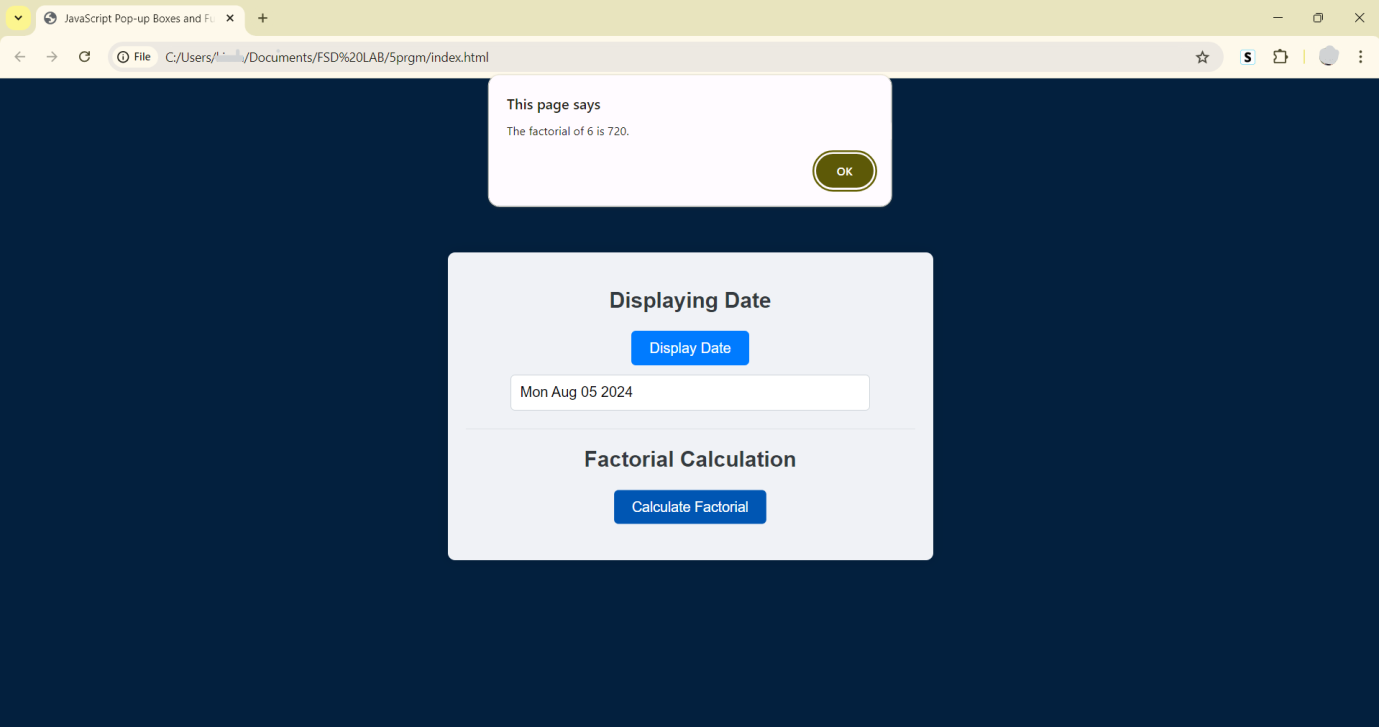
# OUTPUT 1:



**OUTPUT 3:**



# OUTPUT 4:



1. **Write a JavaScript program to generate a for loop that will iterate from 0 to 15. For each iteration, it will check if the current number is odd or even, and display a message to the screen using map and filters.**

**// Generate an array of numbers from 0 to 15**

**const numbers = Array.from({ length: 16 }, (\_, index) => index);**

**// Use map to check if each number is odd or even and create messages**

**const messages = numbers.map(num => {**

**if (num % 2 === 0) {**

**return `${num} is even`;**

**} else {**

**return `${num} is odd`;**

**}**

**});**

**// Generate an array of numbers from 0 to 15**

**const numbers = Array.from({ length: 16 }, (\_, index) => index);**

**// Filter even numbers**

**const evenNumbers = numbers.filter(num => num % 2 === 0);**

**const oddNumbers = numbers.filter(num => num % 2 !== 0);**

**// Create messages for even numbers**

**const evenMessages = evenNumbers.map(num => `${num} is even`);**

**// Create messages for odd numbers**

**const oddMessages = oddNumbers.map(num => `${num} is odd`);**

**// Combine messages**

**const allMessages = [...evenMessages, ...oddMessages];**

**// Display the messages**

**allMessages.forEach(message => console.log(message));**

**// Display the messages**

**messages.forEach(message => console.log(message));**

1. **Write a JavaScript program that modifies the DOM to change the content of an HTML element. Add event listeners to handle button clicks and dynamically change text on the page.**

<!DOCTYPE html>  
<html>  
<body>  
  <p id="demo">Hello World!</p>  
  <button id="changeTextButton">Change Text</button>  
  <script src="script.js"></script>  
</body>  
</html>

document.getElementById("changeTextButton").addEventListener("click", function() {  
  document.getElementById("demo").innerHTML = "Text changed!";  
});

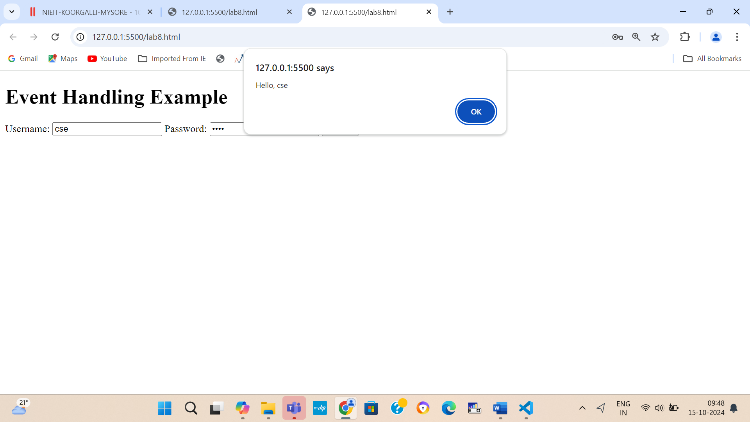
1. **Write a JavaScript program that handles events triggered by the body, button, text box, and password field**

<!DOCTYPE html>  
<html>  
<body onload="onPageLoad()">  
  <h1>Event Handling Example</h1>  
   
  <label for="username">Username:</label>  
  <input type="text" id="username" onfocus="onFocus()" onblur="onBlur()">  
   
  <label for="password">Password:</label>  
  <input type="password" id="password">  
   
  <button onclick="onButtonClick()">Submit</button>  
  
  <script src="events.js"></script>  
</body>  
</html>

# Events.js

function onPageLoad() {  
  console.log("Page loaded.");  
}  
  
function onFocus() {  
  console.log("Text box focused.");  
}  
  
function onBlur() {  
  console.log("Text box lost focus.");  
}  
  
function onButtonClick() {  
  let username = document.getElementById("username").value;  
  alert("Hello, " + username);  
}

**OUTPUT**



1. **Write a JavaScript program to find HTML elements using various methods in JavaScript, such as by id, tag name, class name, CSS selectors, and HTML object collections.**

# Example9.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Element Finder</title>

</head>

<body>

<h1 id="main-title">Element Finder</h1>

<div class="container">

<p class="description">This is a description paragraph.</p>

<p class="description">This is another description paragraph.</p>

<ul>

<li class="item">Item 1</li>

<li class="item">Item 2</li>

<li class="item">Item 3</li>

</ul>

</div>

<button id="find-elements">Find Elements</button>

<script src="Example9.js"></script>

</body>

</html>

# Example9.js

document.getElementById('find-elements').addEventListener('click', function() {

// 1. Find an element by ID

const title = document.getElementById('main-title');

console.log('Element by ID:', title.textContent);

// 2. Find elements by tag name

const paragraphs = document.getElementsByTagName('p');

console.log('Elements by tag name (p):', paragraphs);

for (let p of paragraphs) {

console.log('Paragraph:', p.textContent);

}

// 3. Find elements by class name

const items = document.getElementsByClassName('item');

console.log('Elements by class name (item):', items);

for (let item of items) {

console.log('List item:', item.textContent);

}

// 4. Find elements using querySelector

const firstDescription = document.querySelector('.description');

console.log('First description using querySelector:', firstDescription.textContent);

// 5. Find elements using querySelectorAll

const allDescriptions = document.querySelectorAll('.description');

console.log('All descriptions using querySelectorAll:');

allDescriptions.forEach(desc => {

console.log('Description:', desc.textContent);

});

// 6. Find elements using HTML object collections

const container = document.querySelector('.container');

const containerChildren = container.children;

console.log('Children of the container:', containerChildren);

for (let child of containerChildren) {

console.log('Container child:', child.tagName);

}

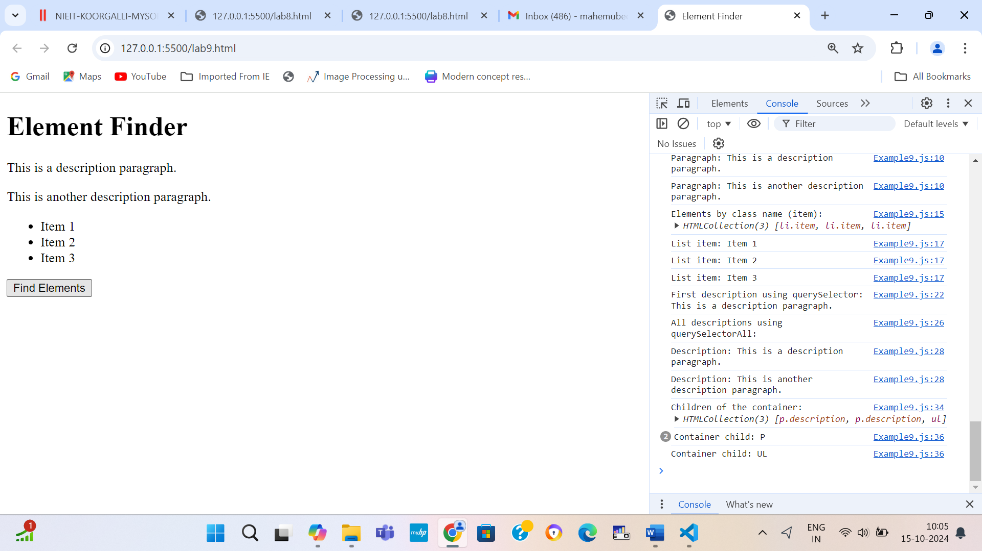
});

**Running the Code**

To see this in action:

1. Create an HTML file with the provided structure.
2. Create a JavaScript file named script.js with the provided code.
3. Open the HTML file in a web browser and click the "Find Elements" button. You will see the results logged in the console.

# OUTPUT:



1. **Create a React application that demonstrates the use of state and props. Implement a parent component that maintains state and passes it down to child components via props. Show how changes in state affect the rendering of the child components.**

npx create-react-app counter-app cd counter-app

npm start

# App.js

import React, { useState } from 'react';

import ChildComponent from './ChildComponent';

function App() {

const [count, setCount] = useState(0);

const increment = () => {

setCount(count + 1);

};

const decrement = () => {

setCount(count - 1);

};

return (

<div style={{ textAlign: 'center', marginTop: '50px' }}>

<h1>State and Props Demo</h1>

<h2>Count: {count}</h2>

<button onClick={increment}>Increment</button>

<button onClick={decrement}>Decrement</button>

<ChildComponent count={count} />

</div>

);

}

export default App;

ChildComponent.js

import React from 'react';

function ChildComponent({ count }) {

return (

<div style={{ marginTop: '20px' }}>

<h3>Child Component</h3>

<p>The count passed from the parent is: {count}</p>

{count % 2 === 0 ? (

<p>The count is even!</p>

) : (

<p>The count is odd!</p>

)}

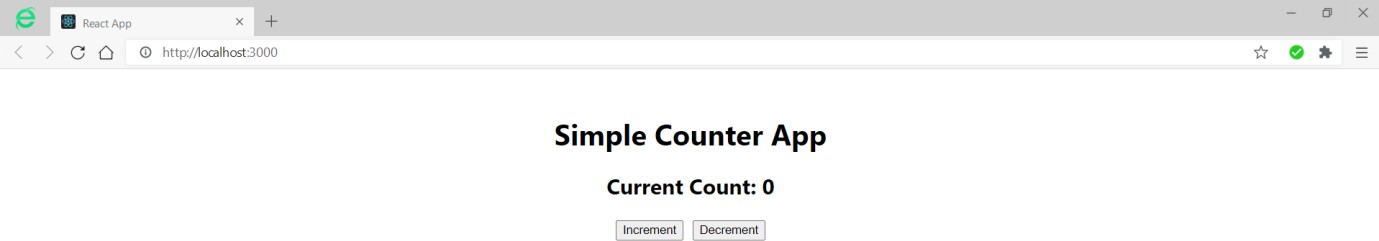
</div>

);

}

export default ChildComponent;

# OUTPUT 1:



**OUTPUT 2:** *click on increment*



**OUTPUT 3:** *click on decrement*



1. **Develop a React program that utilizes hooks, such as useState and useEffect, within a component.**

**npx create-react-app hooks-demo**

**cd hooks-demo**

**npm start**

**src/App.js**

import React, { useState, useEffect } from 'react';

function App() {

const [count, setCount] = useState(0);

const [isRunning, setIsRunning] = useState(false);

useEffect(() => {

let timer;

if (isRunning) {

timer = setInterval(() => {

setCount(prevCount => prevCount + 1);

}, 1000);

}

// Cleanup the timer when the component unmounts or isRunning changes

return () => clearInterval(timer);

}, [isRunning]);

const handleStart = () => {

setIsRunning(true);

};

const handleStop = () => {

setIsRunning(false);

};

const handleReset = () => {

setCount(0);

setIsRunning(false);

};

return (

<div style={{ textAlign: 'center', marginTop: '50px' }}>

<h1>Counter with Hooks</h1>

<h2>Count: {count}</h2>

<button onClick={handleStart} disabled={isRunning}>Start</button>

<button onClick={handleStop} disabled={!isRunning}>Stop</button>

<button onClick={handleReset}>Reset</button>

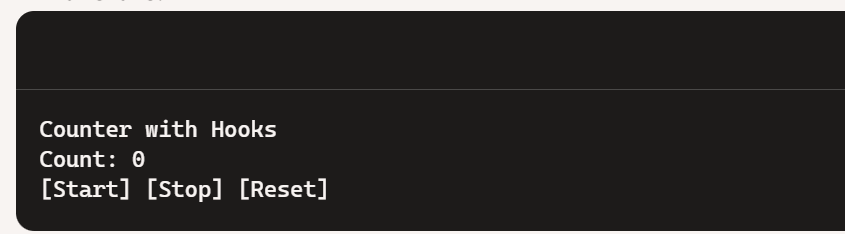
</div>

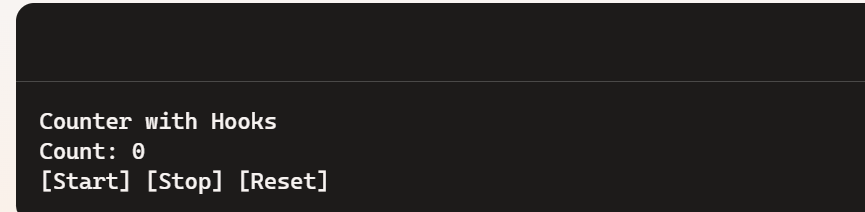
);

}

export default App;

**OUTPUT**

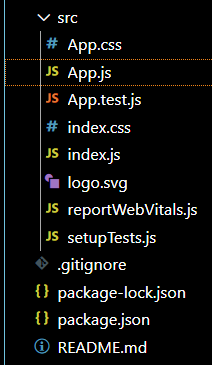
****

****

1. **Write a React program that demonstrates the component life cycle phases mounting, updating and unmounting.**

Lifecycle Methods in a Class Component

# File structure:



**App.js**

import React, { Component } from 'react';

class LifecycleDemo extends Component { constructor(props) {

super(props);

this.state = { count: 0 };

console.log('Constructor: Component is being initialized.');

}

componentDidMount() {

console.log('ComponentDidMount: Component has been mounted.');

}

componentDidUpdate(prevProps, prevState) { console.log('ComponentDidUpdate: Component has been updated.'); console.log('Previous State:', prevState);

console.log('Current State:', this.state);

}

componentWillUnmount() { console.log('ComponentWillUnmount: Component is about to be

unmounted.');

}

incrementCount = () => {

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

};

render() {

console.log('Render: Component is rendering.'); return (

<div style={{ textAlign: 'center', marginTop: '50px' }}>

<h1>React Component Lifecycle</h1>

<p>Count: {this.state.count}</p>

<button onClick={this.incrementCount}>Increment</button>

</div>

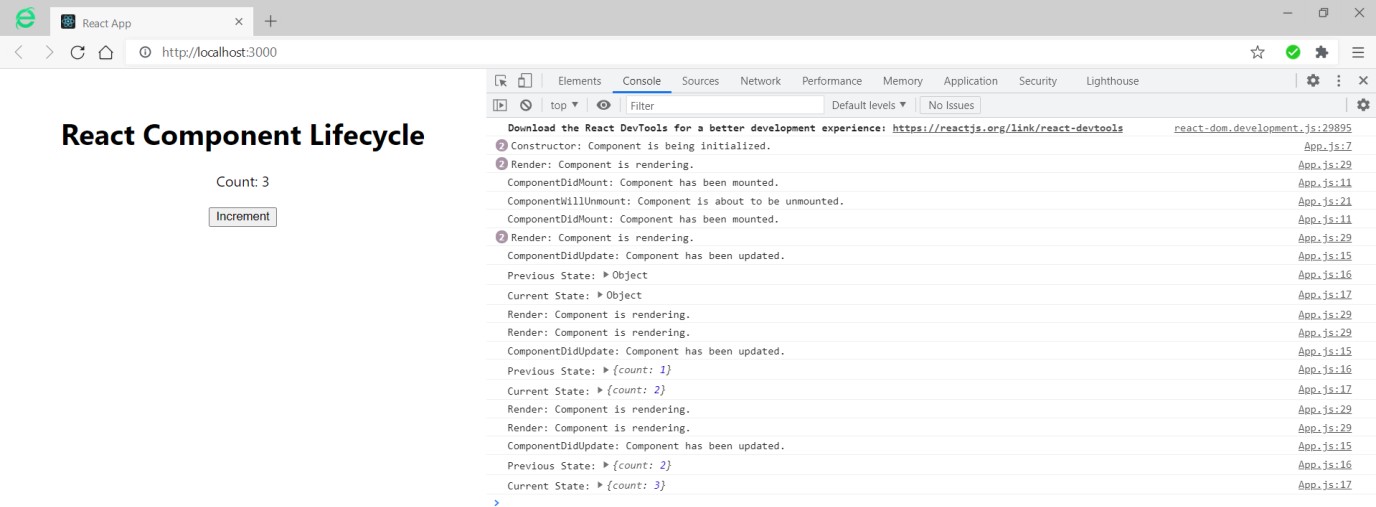
);

}

}

export default LifecycleDemo;

# OUTPUT:



**13.Write a React program that implements error handling using error boundaries.**

import React, { Component } from 'react';

class ErrorBoundary extends Component { constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

// Update state so the next render shows the fallback UI. return { hasError: true };

}

componentDidCatch(error, errorInfo) {

// You can also log the error to an error reporting service console.error("Error caught in ErrorBoundary:", error, errorInfo);

}

render() {

if (this.state.hasError) {

// You can render any custom fallback UI return <h1>Something went wrong.</h1>;

}

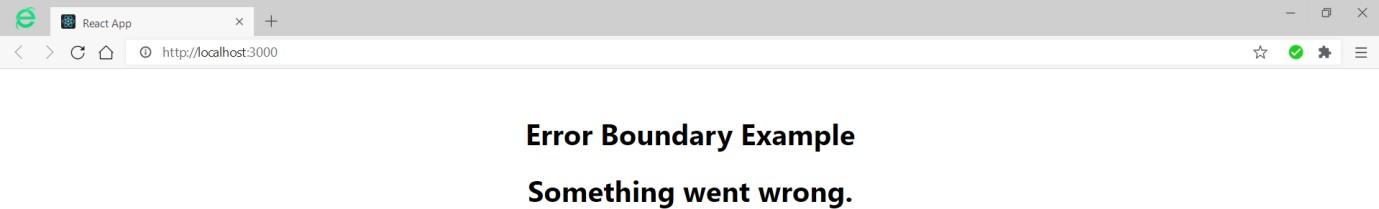
return this.props.children;

}

}

export default ErrorBoundary;

# OUTPUT:



**14.Write a React program that integrates React Router into a React application.**

Install React Router: **npm install react-router-dom App.js**

import React from "react"; import {

BrowserRouter as Router, Routes,

Route, Link,

useNavigate, Outlet,

} from "react-router-dom";

const Home = () => {

const navigate = useNavigate();

return (

<div>

<h2>Home Page</h2>

<button onClick={() => navigate("/contact")}>Go to Contact</button>

</div>

);

};

const About = () => (

<div>

<h2>About Page</h2>

<h3>Welcome to About page</h3>

<nav>

<ul>

<li>

<Link to="team">Our Team</Link>

</li>

<li>

<Link to="company">Our Company</Link>

</li>

</ul>

</nav>

<Outlet />

</div>

);

const Contact = () => <h2>Contact Page</h2>; const Team = () => <h2>Team Page</h2>;

const Company = () => <h2>Company Page</h2>;

function App() { return (

<Router>

<nav>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About</Link>

</li>

<li>

<Link to="/contact">Contact</Link>

</li>

</ul>

</nav>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />}>

<Route path="team" element={<Team />} />

<Route path="company" element={<Company />} />

</Route>

<Route path="/contact" element={<Contact />} />

</Routes>

</Router>

);

}

export default App;

# index.css

body {

font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 0;

padding: 0;

}

h2 {

text-align: center; color: #333;

}

h3 {

text-align: center; color: #333;

}

nav ul { display: flex;

justify-content: center; list-style: none; padding: 0;

}

nav li {

margin: 0 10px;

}

nav a {

text-decoration: none; color: #333;

}

button { display: block;

margin: 20px auto; padding: 10px 20px;

background-color: #007BFF;

color: white; border: none; border-radius: 5px; cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

# index.js

import React from "react";

import ReactDOM from "react-dom/client"; import "./index.css";

import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root")); root.render(

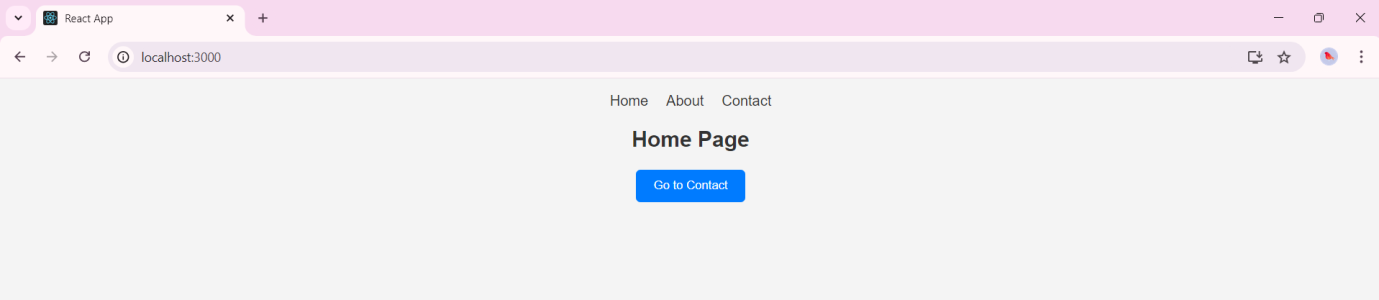
<React.StrictMode>

<App />

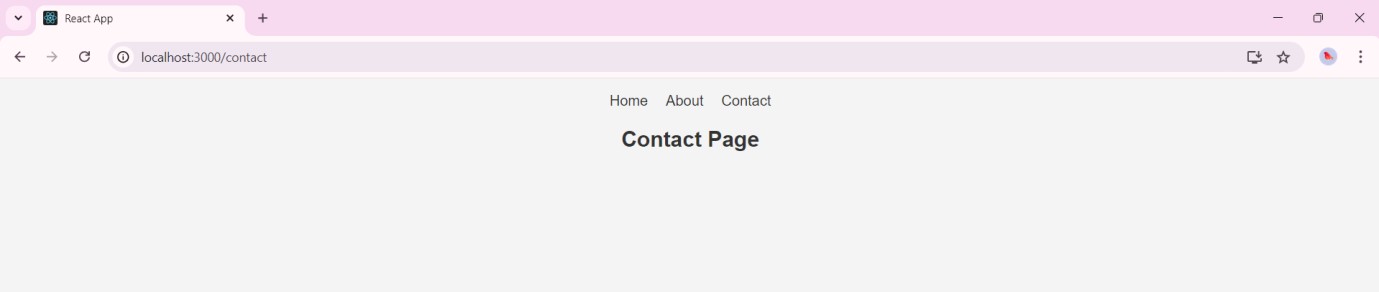
</React.StrictMode>

);

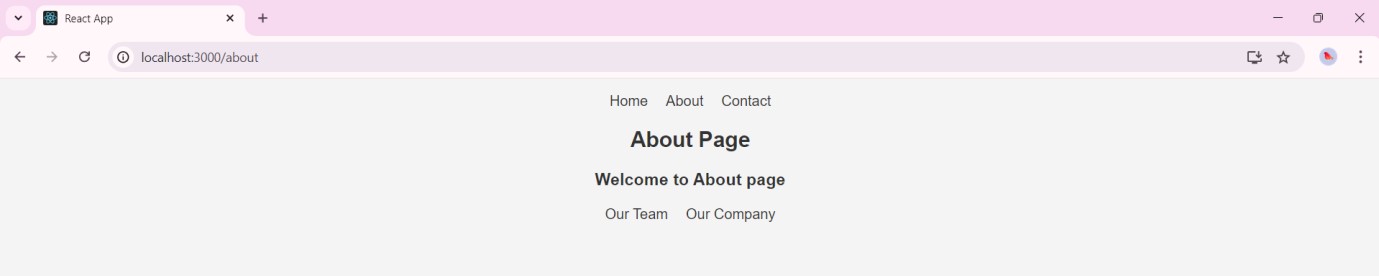
# OUTPUT 1:



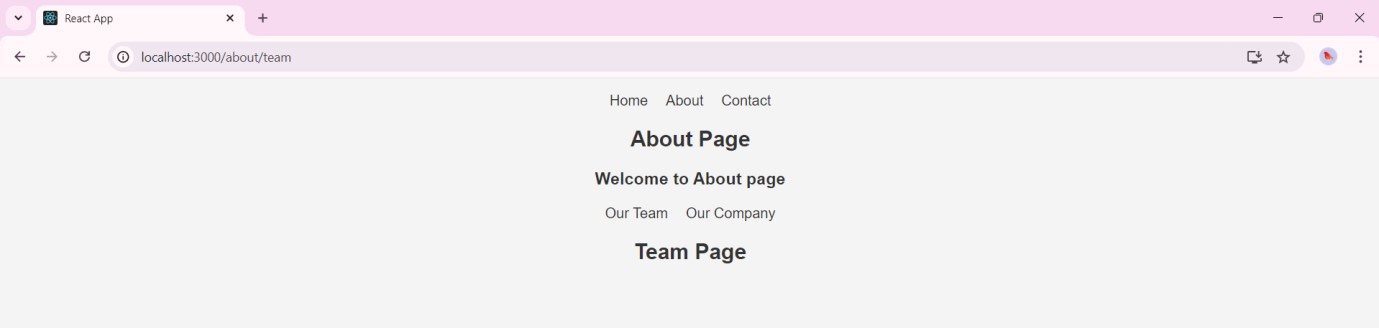
**OUTPUT 2:**



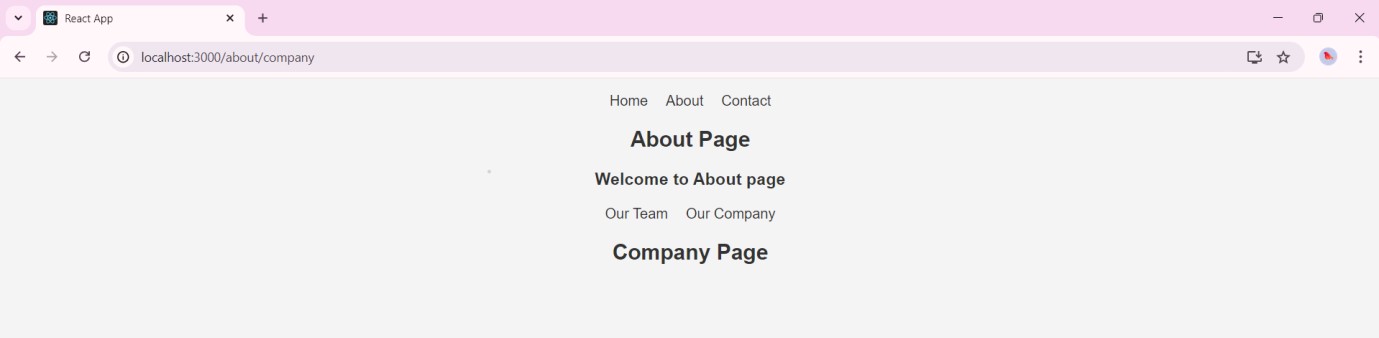
# OUTPUT 3:



**OUTPUT 4:**



# OUTPUT 5:



**15. Write a program that connects to MongoDB using Nodejs and performs basic query.**

// mongo\_demo.js  
const { MongoClient } = require('mongodb');  
  
const url = 'mongodb://localhost:27017';  
const dbName = 'mydatabase';  
  
MongoClient.connect(url, { useNewUrlParser: true, useUnifiedTopology: true }, (err, client) => {  
    if (err) throw err;  
     
    console.log("Connected successfully to the database");  
  
    const db = client.db(dbName);  
    const collection = db.collection('users');  
  
    // Insert a document  
    collection.insertOne({ name: "John", age: 30 }, (err, result) => {  
        if (err) throw err;  
        console.log("Document inserted");  
        client.close();  
    });  
});

**16. Describe the following operations in MongoDB.**

**a)     switch to a new database called companyDB.**

**b)     Create a new collection called employees.**

**c)     Insert a few documents into the employees collection**

**d)     Query all documents in the employees collection**

**e)     Query documents where the age is greater than 30**

**f)      Insert a new document into the employees collection**

**g)     Find a document where the name is "Alice"**

**h)     Update the age of the employee named "Bob"**

**i)      Delete the employee named "Charlie"**

**j)      Retrieve only the name and department fields for all employees**

**k)     Calculate the average age of employees in each department**

a. use companyDB

b. db.createCollection("employees")

c. db.employees.insertMany([

{ name: "Alice", age: 28, department: "HR" },

{ name: "Bob", age: 35, department: "Engineering" },

{ name: "Charlie", age: 32, department: "Finance" }

])

d. db.employees.find()

e. db.employees.find({ age: { $gt: 30 } })

f. db.employees.insertOne({ name: "David", age: 29, department: "Marketing" })

g. db.employees.findOne({ name: "Alice" })

h. db.employees.updateOne({ name: "Bob" }, { $set: { age: 36 } })

i. db.employees.deleteOne({ name: "Charlie" })

j. db.employees.find({}, { name: 1, department: 1 })

k. db.employees.aggregate([

{ $group: { \_id: "$department", averageAge: { $avg: "$age" } } }

])

**17. Demonstrate the following operations using  Mongo shell script**

**a)     switch to the schoolDB database, create a teachers collection, and insert documents with fields name, subject, and experience (in years).**

**b)     Use the Mongo shell to export the students collection to a JSON file.**

**Import the JSON file back into a new collection named studentsBackup.**

a. use schoolDB

db.teachers.insertMany([

{ name: "John Doe", subject: "Mathematics", experience: 10 },

{ name: "Jane Smith", subject: "English", experience: 8 },

{ name: "Mike Johnson", subject: "History", experience: 5 }

])

b. mongoexport --db=schoolDB --collection=students --out=students.json

**18. Create and demonstrate the following in Mongo DB**

**a)     Projection operators ($, $elematch and $slice)**

**b)     Execute query selectors (Any one comparison selectors, Any one logical selectors)**

**c)     Execute Aggregation operations ($avg, $min, $max, $push, $addToSet.)**

* 1. db.employees.find({}, { name: 1, department: 1 })
  2. db.students.find({ name: "Alice" }, { subjects: { $elemMatch: { score: { $gt: 90 } } } })
  3. db.students.find({ name: "Alice" }, { subjects: { $slice: 1 } })