|  |  |  |
| --- | --- | --- |
| **Name : Harsh Jain** | **Class/Roll No. :D11AD/25** | **Grade :** |

**Title of Experiment :Learning ReactJs.**

**Objective of Experiment : To showcase the implementation of state management with the useState hook and data passing between components using props in a React application.**

**Outcome of Experiment : Enhanced interactivity and dynamic content rendering, resulting in optimized user experience and improved reactivity within the application.**

**Problem Statement : Enhance your application that you made in Exp 07 with state and props**

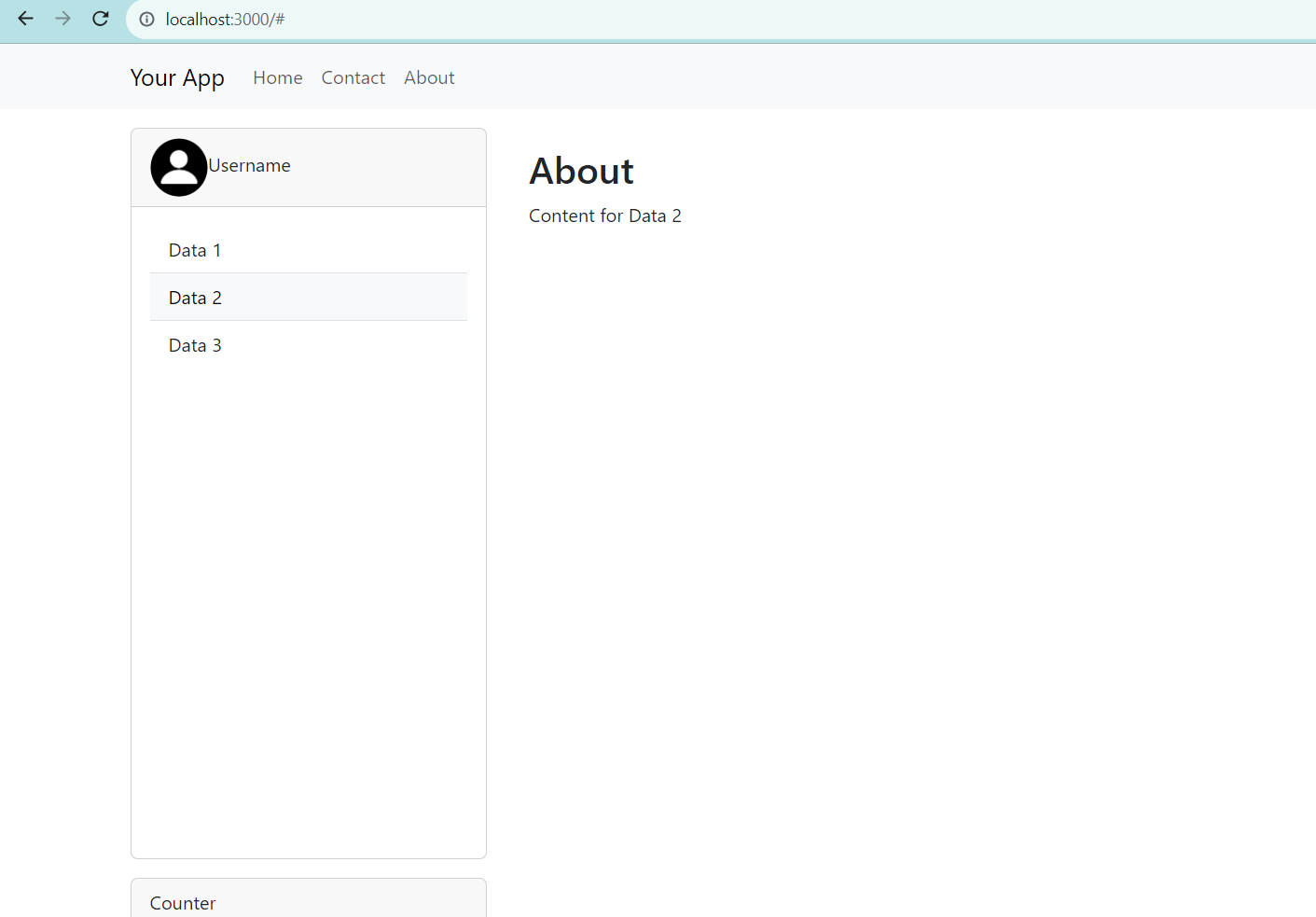
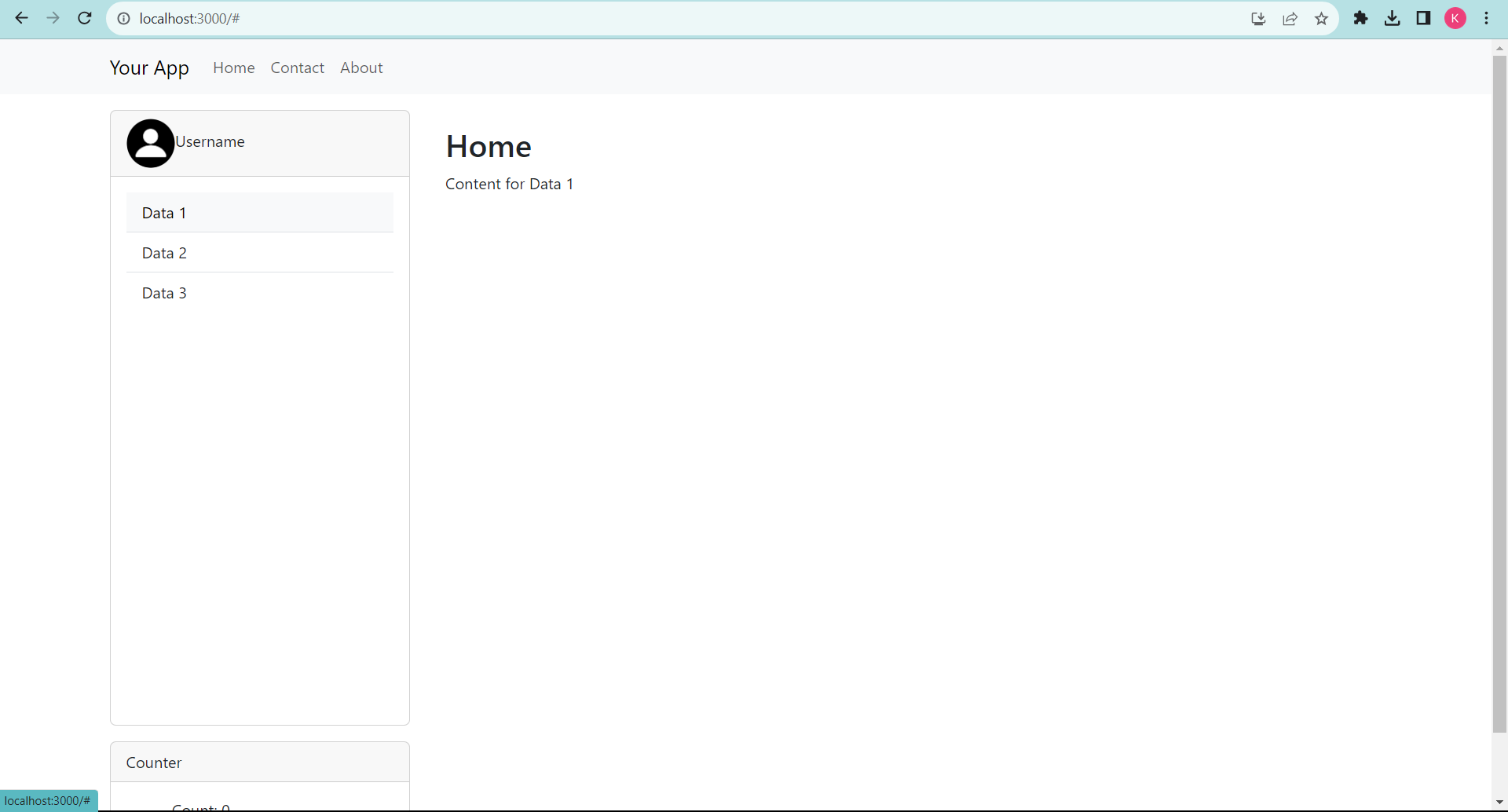
* **Implement a counter component that keeps track of a count using state. Include buttons to increment and decrement the count. Display the current count.**
* **Pass data from the Sidebar component to the MainContent component using props. Display the data in the MainContent component.**

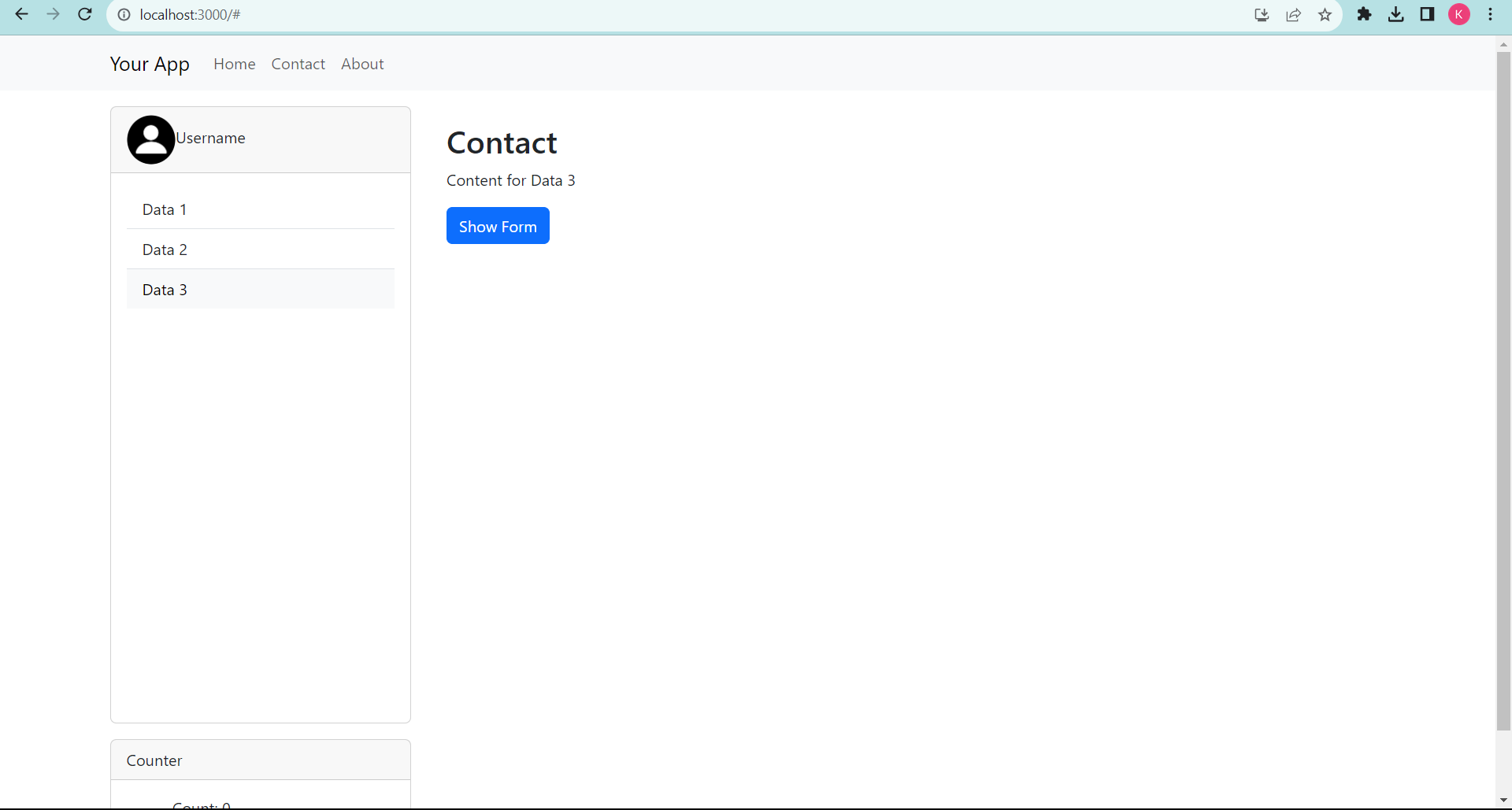
**Description / Theory :**

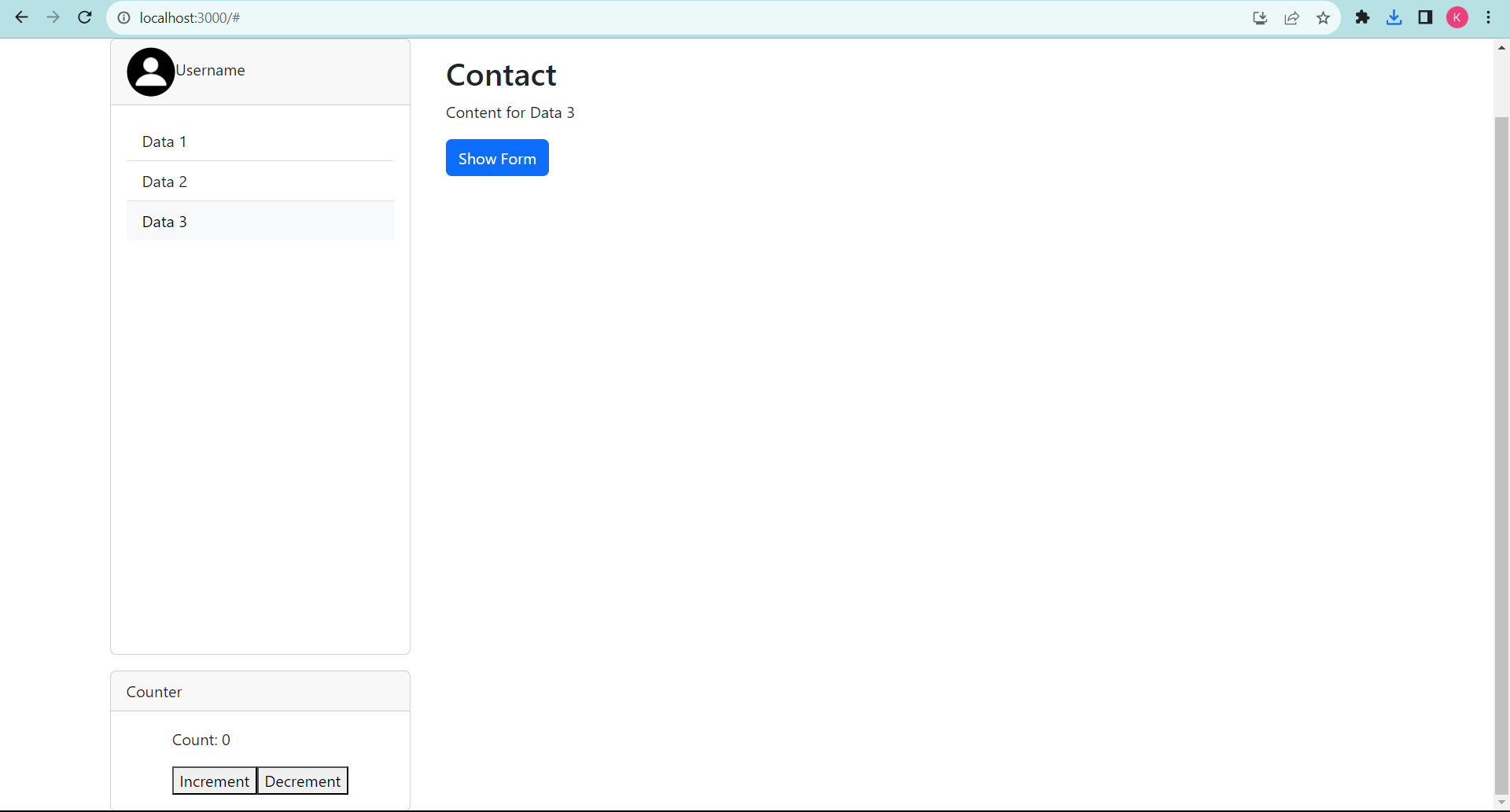
1. **State Management with useState Hook:**
   1. The **useState** hook is an essential part of React's state management, allowing functional components to manage stateful logic.
   2. It provides a way to declare state variables and update them, triggering a re-render of the component.
   3. State variables, when updated, cause React to re-evaluate the component, ensuring that the UI reflects the most recent changes in the application's data.
2. **Data Transfer and Rendering with Props:**
   1. The passing of data from the Sidebar component to the MainContent component demonstrates the use of props in React.
   2. Props allow for the transfer of data between components and are particularly useful for creating reusable and composable components.
   3. Components can access the data passed through props and use it to render dynamic content, providing flexibility in designing complex user interfaces.
3. **Interactive User Interfaces and Event Handling:**
   1. The buttons in the counter component showcase event handling in React.
   2. Event handlers are functions that are triggered in response to user actions such as clicks, mouse movements, or keyboard inputs.
   3. By attaching event handlers to specific elements, developers can enable interactivity and create responsive user interfaces that react to user actions in real time.

**Program :**

|  |
| --- |
| import React, { useState } from 'react';    const Counter = () => {  const [count, setCount] = useState(0);    const increment = () => {  setCount(count + 1);  };    const decrement = () => {  setCount(count - 1);  };    return (  <div>  <p>Count: {count}</p>  <button onClick={increment}>Increment</button>  <button onClick={decrement}>Decrement</button>  </div>  );  };    export default Counter;      import React, { useState } from 'react';    const MainContent = ({ data }) => {  const [showForm, setShowForm] = useState(false);  const [formData, setFormData] = useState({  name: '',  email: '',  message: '',  });  const [formErrors, setFormErrors] = useState({});  const [isSubmitted, setIsSubmitted] = useState(false);    const handleInputChange = (event) => {  const { name, value } = event.target;  setFormData((prevData) => ({ ...prevData, [name]: value }));  };    const handleSubmit = (event) => {  event.preventDefault();  const errors = {};    if (formData.name.trim() === '') {  errors.name = 'Name is required';  }    if (formData.email.trim() === '') {  errors.email = 'Email is required';  }    if (formData.message.trim() === '') {  errors.message = 'Message is required';  }    if (Object.keys(errors).length === 0) {  setIsSubmitted(true);  // You can handle form submission logic here  } else {  setFormErrors(errors);  }  };    let content = <p>Select a feature to view data.</p>;    if (data) {  if (data === 'Data 1') {  content = (  <div>  <h2>Home</h2>  <p>Content for Data 1</p>  </div>  );  } else if (data === 'Data 2') {  content = (  <div>  <h2>About</h2>  <p>Content for Data 2</p>  </div>  );  } else if (data === 'Data 3') {  content = (  <div>  <h2>Contact</h2>  <p>Content for Data 3</p>  <button className="btn btn-primary" onClick={() => setShowForm(true)}>  Show Form  </button>  {showForm && (  <form onSubmit={handleSubmit}>  <div className="mb-3">  <label htmlFor="name" className="form-label">  Name  </label>  <input  type="text"  className="form-control"  id="name"  name="name"  value={formData.name}  onChange={handleInputChange}  />  {formErrors.name && <div className="text-danger">{formErrors.name}</div>}  </div>  <div className="mb-3">  <label htmlFor="email" className="form-label">  Email  </label>  <input  type="email"  className="form-control"  id="email"  name="email"  value={formData.email}  onChange={handleInputChange}  />  {formErrors.email && <div className="text-danger">{formErrors.email}</div>}  </div>  <div className="mb-3">  <label htmlFor="message" className="form-label">  Message  </label>  <textarea  className="form-control"  id="message"  name="message"  rows="4"  value={formData.message}  onChange={handleInputChange}  ></textarea>  {formErrors.message && <div className="text-danger">{formErrors.message}</div>}  </div>  <button type="submit" className="btn btn-primary">  Submit  </button>  </form>  )}  {isSubmitted && (  <div className="alert alert-success mt-3" role="alert">  Form submitted successfully!  </div>  )}  </div>  );  }  }    return (  <div className="col-md-9">  <div className="container mt-3">{content}</div>  </div>  );  };    export default MainContent; |

**Results and Discussion:**

****

****