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

**Title of Experiment :Learning ReactJs.**

**Objective of Experiment : To augment the application with a user-friendly form, including input fields for name, email, and message, implementing form validation for email, handling form submissions, and enhancing the visual appearance with CSS styles.**

**Outcome of Experiment : Improved user interaction facilitated by a comprehensive form, error-free input validation, real-time feedback on successful submissions, and an aesthetically appealing application interface.**

**Problem Statement : Enhance your application that you made in Exp 07 and Exp 08 by adding a form and event handling:**

* **Create a form component that includes input fields for a user's name, email, and a message.**
* **Implement form validation to ensure that the email address is valid.**
* **Handle form submission events and display a success message when the form is submitted successfully.**
* **Apply CSS styles to make your application visually appealing.**

**Description / Theory :**

1. **State Management with useState Hook:**
   1. **Form Component**:

Creating a form component involves structuring and rendering input fields for the user's name, email, and a message. This component captures user input and serves as a means for users to provide specific data, enabling data collection and interaction within the application.

* 1. **Form Validation**:

Implementing form validation is critical for ensuring data accuracy and integrity. In the context of email validation, the application checks if the email address provided by the user is in a valid format, preventing the submission of incorrect or improperly formatted email addresses. This process enhances data quality and reliability within the application.

* 1. **Event Handling**:

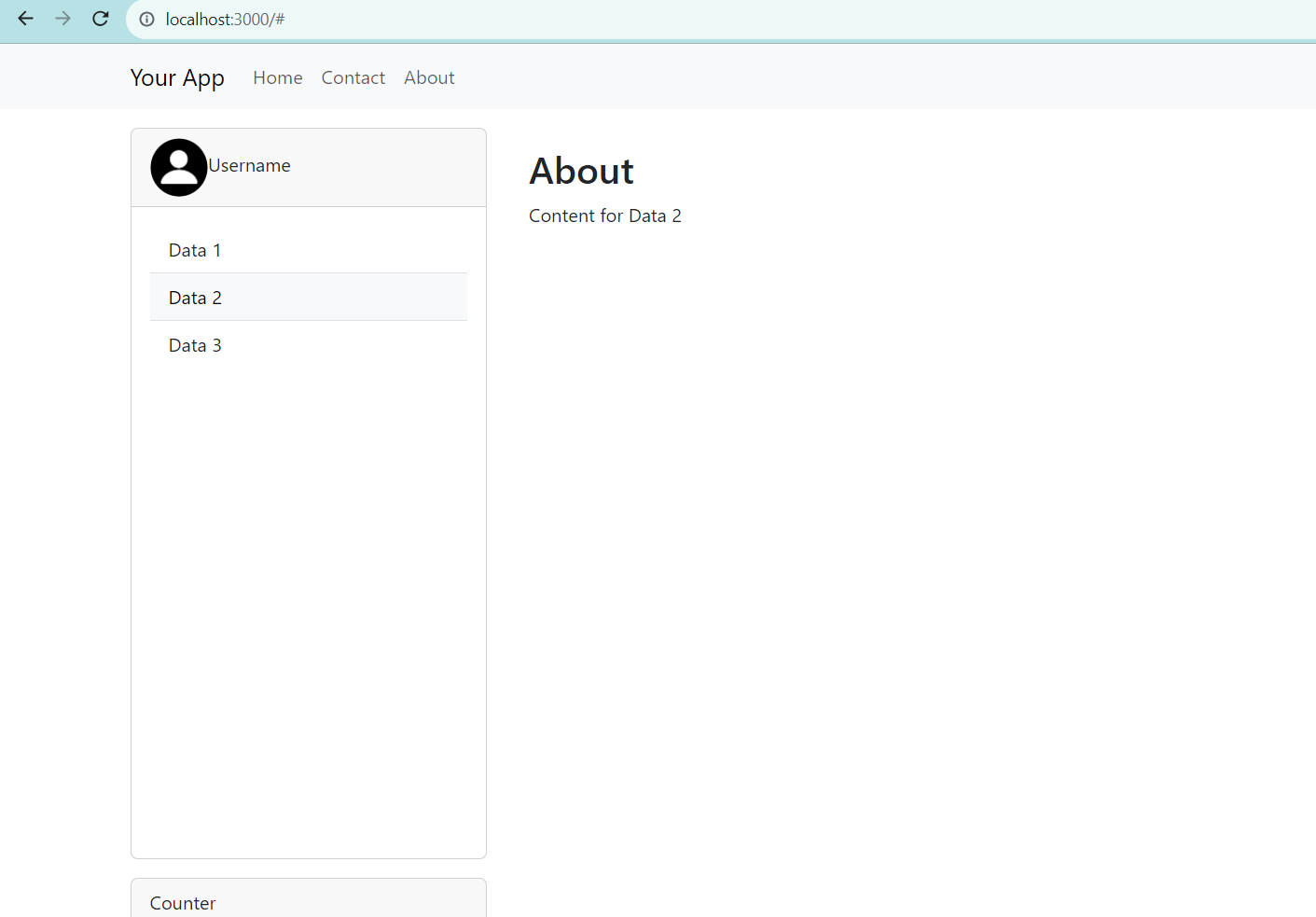
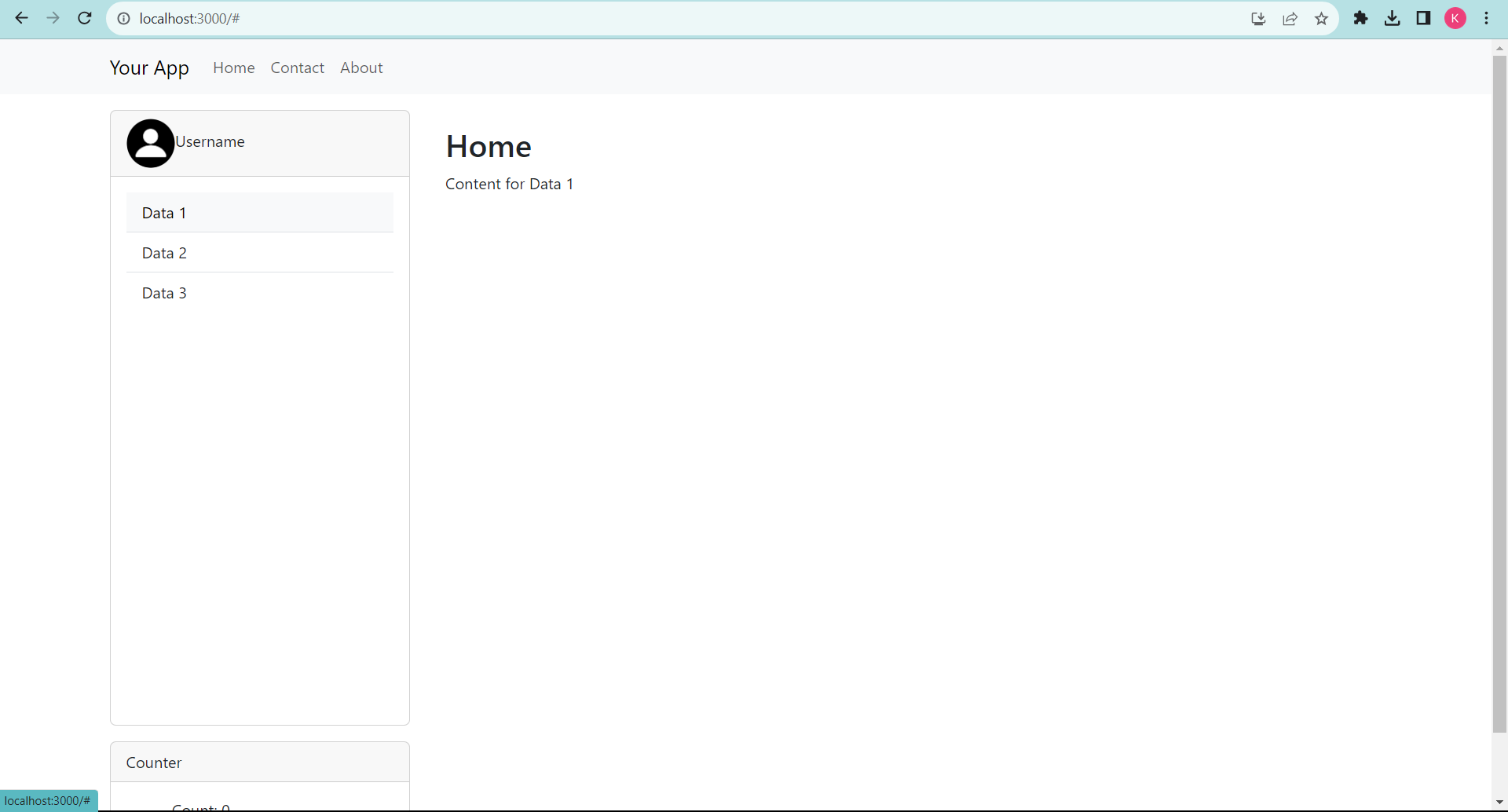
Managing form submission events is crucial for processing user-provided data and responding to user actions. The application includes event handlers that trigger specific actions upon successful form submissions. These handlers facilitate data processing, storage, and potential external operations, ensuring a seamless and responsive user experience.

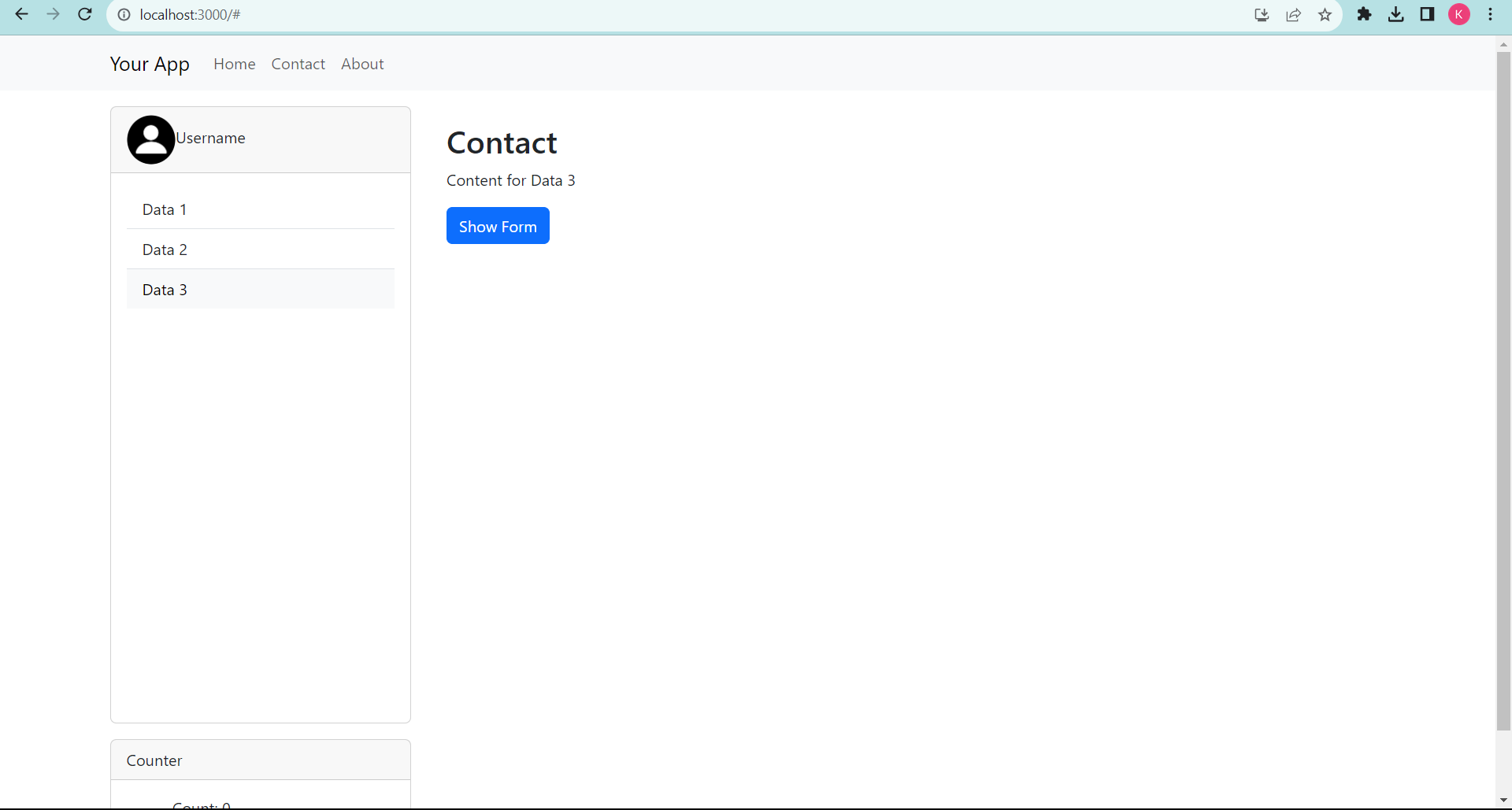
* 1. **CSS Styling**:

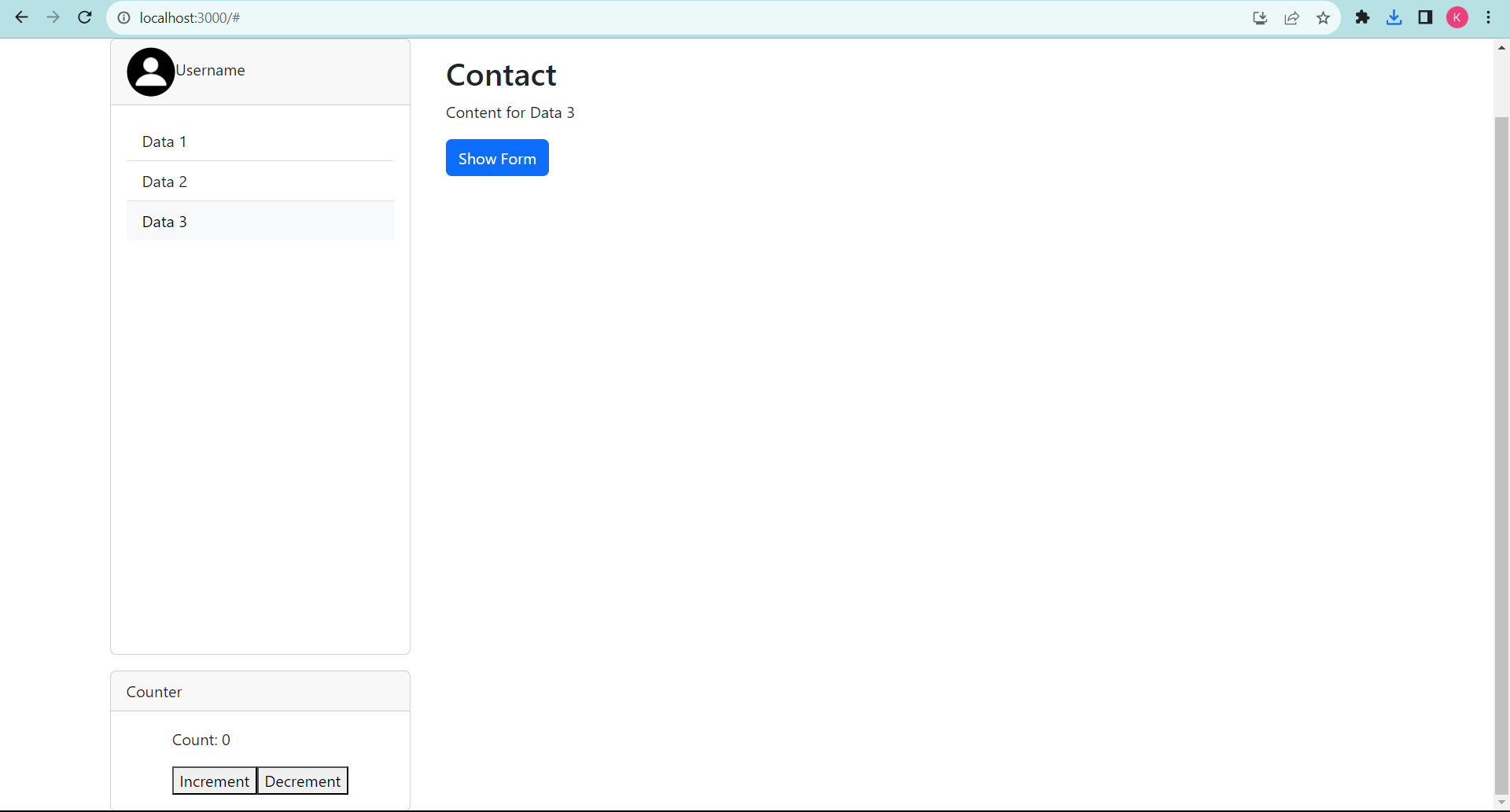
Enhancing the application's visual appeal through CSS styling is fundamental for creating an engaging and aesthetically pleasing user interface. By applying appropriate styling, such as color schemes, typography, and layout designs, the application becomes more visually appealing and user-friendly, resulting in an improved overall user experience and increased user engagement.

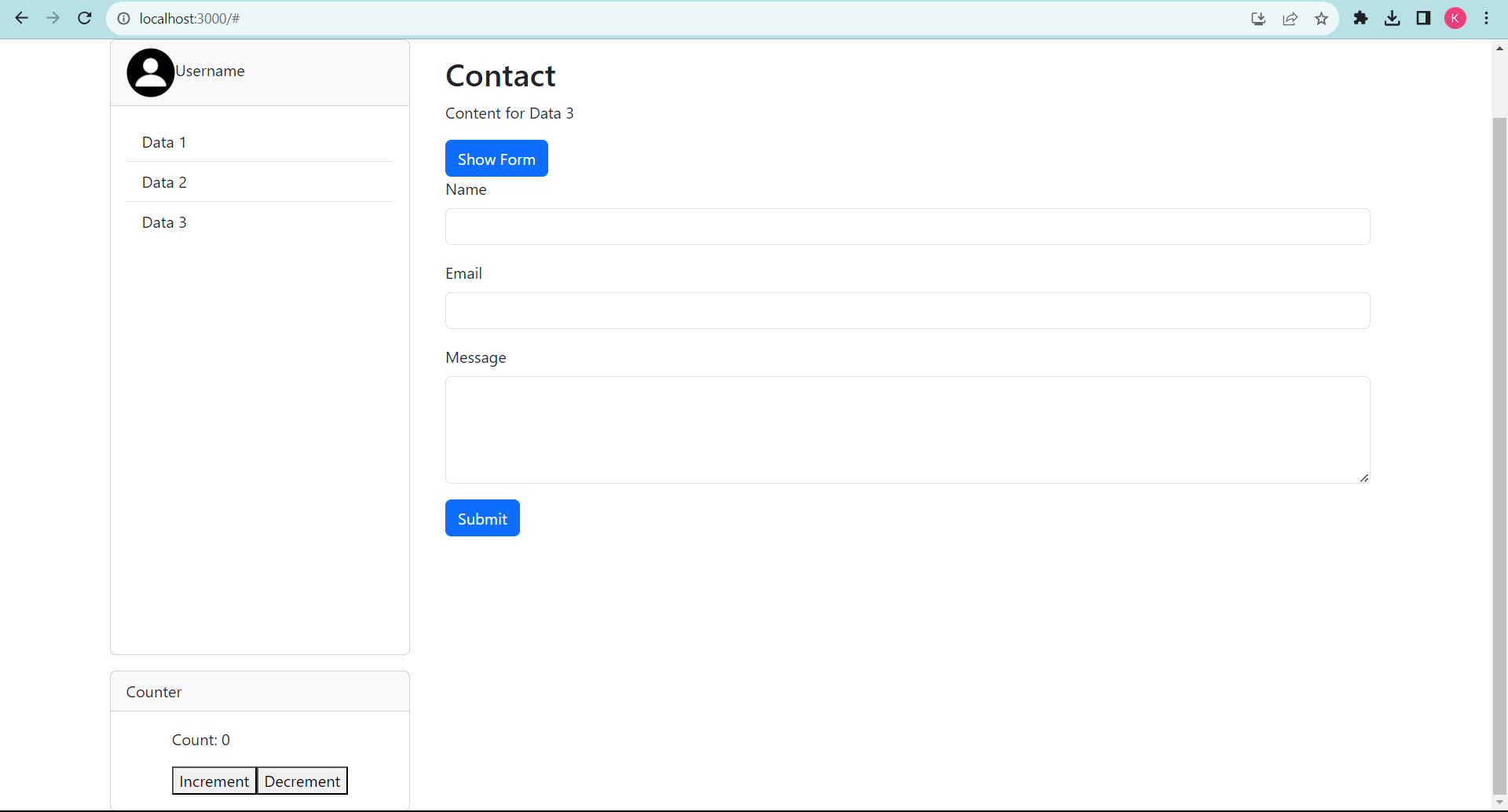
**Program :**

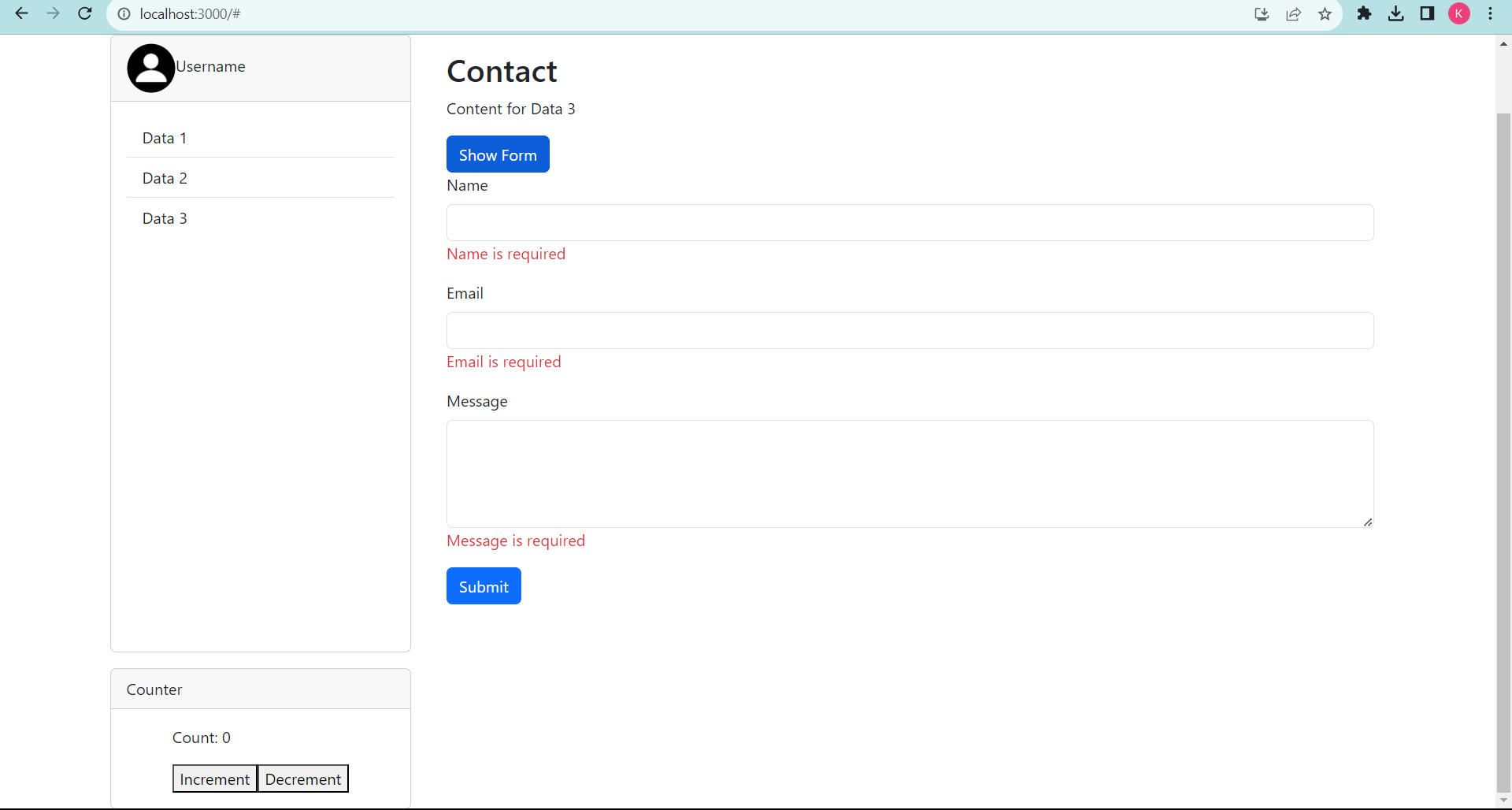
|  |
| --- |
| import React, { useState } from 'react';    const Counter = () => {  const [count, setCount] = useState(0);    const increment = () => {  setCount(count + 1);  };    const decrement = () => {  setCount(count - 1);  };    return (  <div>  <p>Count: {count}</p>  <button onClick={increment}>Increment</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; |

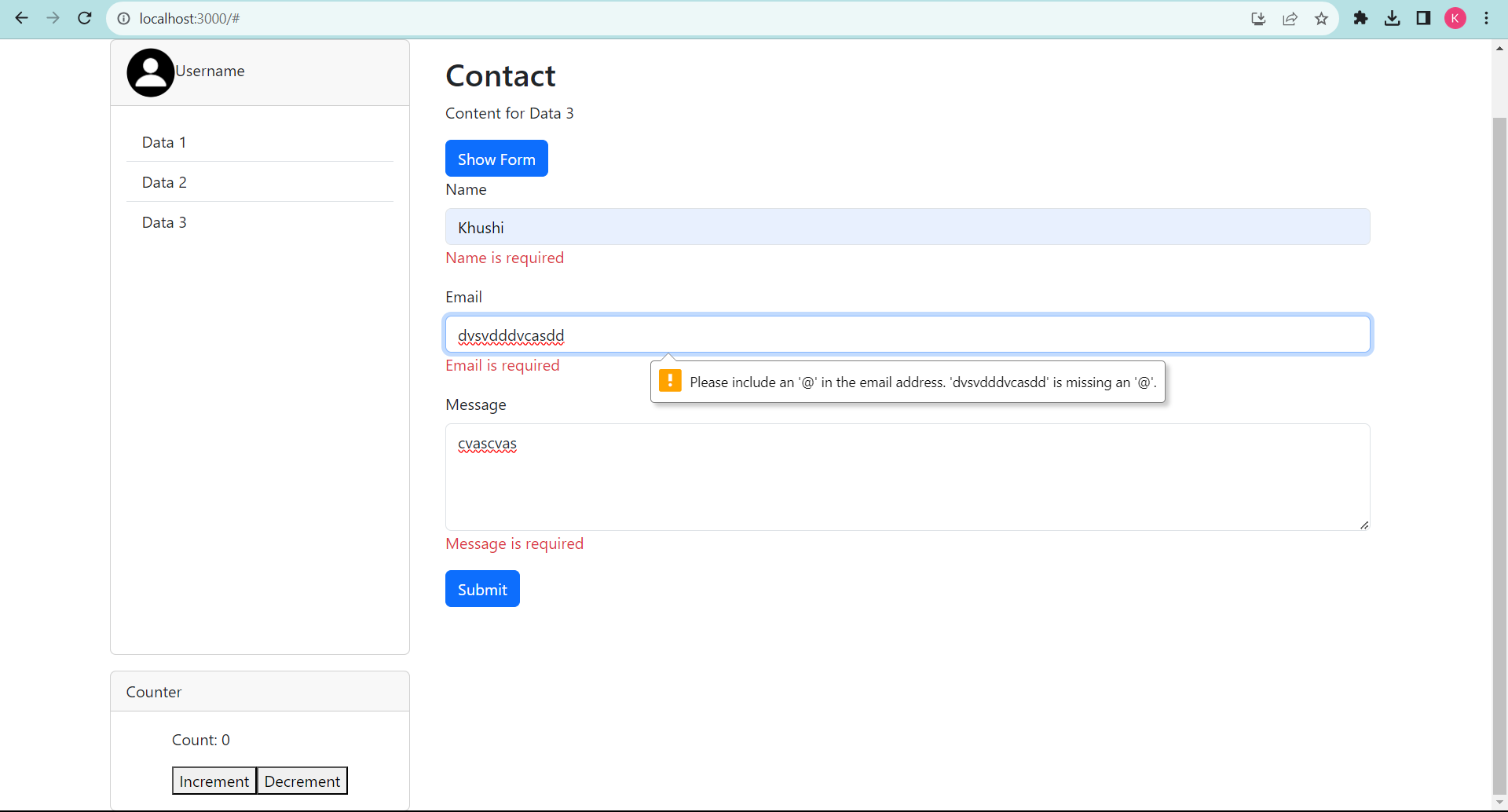
**Results and Discussion:**

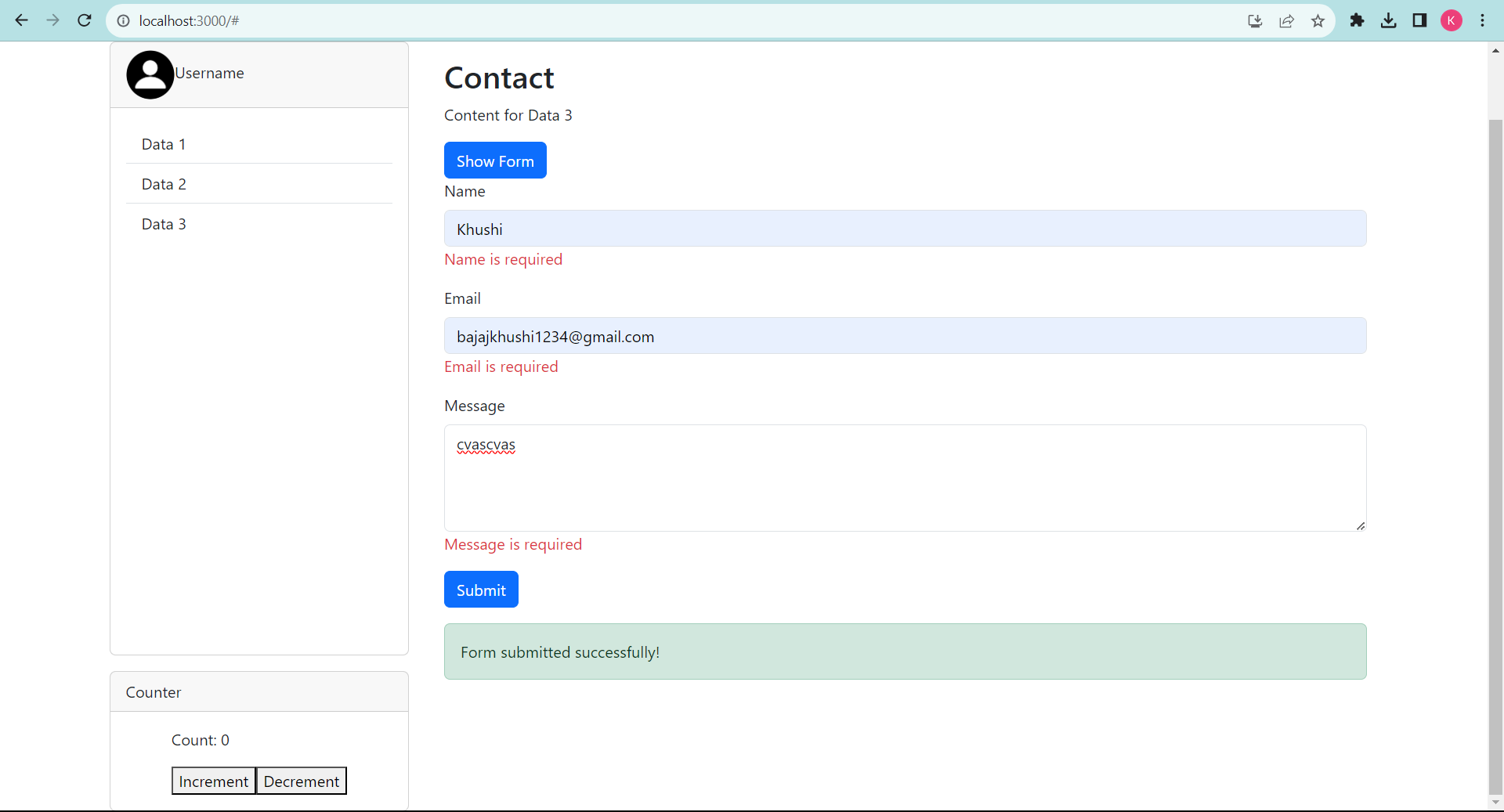
****

****

****

****

****

****