Sheryians Coding School

Live Cohort

React week

Topic Name: Introduction to Forms in React

Definition

Forms in React are used to collect user input and handle events. They are controlled using component state, allowing React to manage input values and user interactions programmatically.

Code (Example)

```
function BasicForm() {
  return (
    <form>
      <input type="text" placeholder="Name" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Use Case

deleteAtIndex() removes a node from a given index in a singly linked list. It handles empty list checks, boundary validation, and link updates.

Interview Q&A

Q: How are forms managed in React compared to vanilla HTML?

A: In React, form inputs are usually controlled components tied to the component's state, whereas in HTML, form elements manage their own state.

Topic Name: Creating Form Elements

Definition

React supports common form elements like input, textarea, and select. These are used to gather different types of user data.

Code (Example)

```
<form>
  <input type="text" />
  <textarea />
  <select>
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
  </select>
</form>
```

🔷 Use Case

- Creating flexible and reusable form layouts for surveys, preferences, or feedback collection.

🔷 Interview Q&A

Q: What are controlled vs uncontrolled components?

A: Controlled components tie form input values to React state, while uncontrolled components use refs to access values directly from the DOM.

Topic Name: Creating Form Elements

🔷 Definition

- Two-way binding means the form input reflects state, and state updates when the input changes — keeping them in sync.

🔹 Code (Example)

```
function TwoWayBinding() {
  const [value, setValue] = React.useState("");

  return (
    <input
      type="text"
      value={value}
      onChange={(e) => setValue(e.target.value)}
    />
  );
}
```

🔹 Use Case

- Used when inputs need to reflect dynamic changes, such as updating a live preview or syncing form values in real time.

🔹 Interview Q&A

Q: What are controlled vs uncontrolled components?

A: Using useState for the input value and updating it using the onChange event.

Topic Name: Creating Form Elements

🔷 Definition

- React uses event handlers like onChange and onSubmit to manage user interactions with forms.

🔷 Code (Example)

```
function Form() {
  const [name, setName] = React.useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name submitted: ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={name} onChange={e => setName(e.target.value)} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

🔷 Use Case

Allows for real-time data handling, client-side validation, and sending data to APIs.

🔷 Interview Q&A

Q: Why use e.preventDefault() in React forms?

A: It prevents the default browser behavior of form submission (which reloads the page), allowing React to handle the data instead.

Topic Name: Validation in React Forms

🔷 Definition

- Validation ensures the data entered in the form is correct before submission. It improves data quality and user experience.

Code (Example)

```
function FormValidation() {
  const [email, setEmail] = React.useState("");
  const [error, setError] = React.useState("");

  const validateEmail = (value) => {
    const regex = /\S+@\S+\.\S+/;
    return regex.test(value);
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!validateEmail(email)) {
      setError("Invalid email address");
    } else {
      setError("");
    }
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Enter email"
      />
      {error && <p style={{ color: "red" }}>{error}</p>}
      <button type="submit">Submit</button>
    </form>
  );
}
```

🔷 Use Case

- Used in forms to check required fields, email format, password strength, etc., before submitting data to backend.

🔷 Interview Q&A

Q: What are common techniques for form validation in React?

A: Manual validation using regex and logic, or using libraries like Formik or Yup for more complex validations.

Topic Name: Validation in React Forms

Definition

- Forms can send collected data to a server via HTTP requests. APIs process the input and return success or error messages.

Code (Example)

- Using fetch

```
function handleSubmit(data) {
  fetch("https://api.example.com/submit", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  })
    .then(res => res.json())
    .then(response => console.log("Success:", response))
    .catch(error => console.error("Error:", error));
}
```

- Using axios:

```
import axios from "axios";

function handleSubmit(data) {
  axios.post("https://api.example.com/submit", data)
    .then(response => console.log("Success:", response))
    .catch(error => console.error("Error:", error));
}
```

🔷 Use Case

- Sending user registration data, contact messages, or feedback to servers for processing.

🔷 Interview Q&A

Q: How can you handle form submission in React with API integration?

A: Use fetch or axios to send a POST request to the backend with the form data.

Topic Name: Handling Loading States and Success/ Error Feedback

Definition

- It's important to give users feedback when submitting forms—e.g., showing loading indicators, success messages, or error alerts.

Code (Example)

```
function FormWithLoading() {
  const [loading, setLoading] = React.useState(false);
  const [success, setSuccess] = React.useState("");
  const [error, setError] = React.useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);
    setSuccess("");
    setError("");
```

```
  fetch("https://api.example.com/submit", { method: "POST" })
    .then(() => {
      setSuccess("Form submitted successfully!");
    })
    .catch(() => {
      setError("Submission failed. Try again.");
    })
    .finally(() => {
      setLoading(false);
    });
};
```

```
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit" disabled={loading}>
        {loading ? "Submitting..." : "Submit"}
      </button>
      {success && <p style={{ color: "green" }}>{success}</p>}
      {error && <p style={{ color: "red" }}>{error}</p>}
    </form>
  );
}
```

🔷 Use Case

- Improves UX by informing users about the state of their action and handling asynchronous submission results gracefully

Interview Q&A

Q: How do you handle loading and error states in form submission?

A: Use useState to track loading, error, and success, and update the UI accordingly based on form submission results.