React - Forms

Topics : React JS
Written on January 03, 2024

In React, forms are a crucial part of building interactive user interfaces. Handling form input involves capturing user input, managing the form state, and often making use of controlled components. Below, I'll provide an overview of how you can work with forms in React.

1. **Controlled Components:** React follows a unidirectional data flow, and controlled components are a key concept in handling form input. In a controlled component, form elements like input, textarea, and select are controlled by the state of the React component. This means that the value of the input is controlled by React state and is updated through a change handler.

   Here's an example of a controlled component:

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

   function ControlledForm() {
   const [inputValue, setInputValue] = useState('');

   const handleChange = (event) => {
   setInputValue(event.target.value);
   };

   return (
   <form>
   <label>
   Input:
   <input type="text" value={inputValue} onChange={handleChange} />
   </label>
   <p>Typed Value: {inputValue}</p>
   </form>
   );
   }
   ```

2. **Handling Form Submission:** To handle form submission, you typically attach an event handler to the `onSubmit` event of the form. This handler can prevent the default form submission behavior, perform any necessary actions (e.g., sending data to a server), and update the component state if needed.

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

```
function SubmitForm() {
const [inputValue, setInputValue] = useState('');

const handleSubmit = (event) => {
event.preventDefault();
// Perform actions with the form data, e.g., send it to a server
console.log('Form submitted with value:', inputValue);
};

const handleChange = (event) => {
setInputValue(event.target.value);
};

return (
<form onSubmit={handleSubmit}>
<label>
Input:
<input type="text" value={inputValue} onChange={handleChange} />
</label>
<button type="submit">Submit</button>
</form>
);
}
```

3. **Form Validation:** You can implement form validation by checking the input values against certain criteria within your change handler or submit handler. Additionally, you might want to provide feedback to users about validation errors.

```
// Example: Validating a simple email input
const [email, setEmail] = useState('');
const [emailError, setEmailError] = useState('');

const handleChange = (event) => {
const value = event.target.value;
setEmail(value);

// Validate email format
const isValidEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(value);
setEmailError(isValidEmail ? '' : 'Invalid email format');
};

// ...

return (
<form onSubmit={handleSubmit}>
<label>
Email:
<input
type="text"
value={email}
onChange={handleChange}
className={emailError ? 'error' : ''}
```

```
/>
</label>
{emailError && <p className="error-message">{emailError}</p>}
<button type="submit" disabled={emailError}>
Submit
</button>
</form>
);
```