**Hands-on: 16. ReactJS-HOL**

**Introduction**

Forms are essential for gathering user input in any web application. In React, form handling is more powerful and flexible than in plain HTML due to the use of state and controlled components. React also enables dynamic form validation, making it easy to check and guide user input in real time. This section covers React form validation, key differences from HTML forms, controlled components, various input controls, and how to handle and submit forms effectively.

1. **React Forms Validation**

Validation ensures that users provide correct and complete input. React allows both manual and library-based validation (e.g., using Formik, Yup, or React Hook Form).

* Manual Validation Example:

function MyForm() {

const [email, setEmail] = useState('');

const [error, setError] = useState('');

const handleSubmit = (e) => {

e.preventDefault();

if (!email.includes('@')) {

setError("Invalid email address");

} else {

setError("");

alert("Form submitted");

}

};

return (

<form onSubmit={handleSubmit}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />

{error && <p style={{color: 'red'}}>{error}</p>}

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

</form>

);

}

* Benefits of React Form Validation:
* Instant feedback
* Custom error messages
* Flexible rules (min length, regex, etc.)

1. **React Form vs. HTML Form**

|  |  |  |
| --- | --- | --- |
| Feature | HTML Form | React Form |
| Input management | Browser-controlled (value changes automatically) | Developer-controlled using state |
| Validation | Built-in (e.g., required, pattern) | Custom or library-driven (manual control) |
| State tracking | Not tracked in code | Fully controlled using state |
| Data submission | Typically reloads page | Handled manually with onSubmit and preventDefault() |
| Flexibility | Limited | Highly flexible and dynamic |

1. **Controlled Components (Recap)**

A controlled component is a form input whose value is controlled by React state.

const [username, setUsername] = useState('');

<input type="text" value={username} onChange={(e) => setUsername(e.target.value)} />

* Key Features:
* Input value is always in sync with state.
* Enables validation, formatting, and transformations.
* Commonly used in forms for full control over data flow.

1. **Various React Form Input Controls**

React supports a variety of HTML input types, all of which can be made controlled:

|  |  |  |
| --- | --- | --- |
| Input Type | Description | JSX Example |
| text | Single-line text | <input type="text" value={val} onChange={...} /> |
| textarea | Multi-line text | <textarea value={val} onChange={...} /> |
| checkbox | True/false values | <input type="checkbox" checked={val} onChange={...} /> |
| radio | Single selection from group | <input type="radio" value="A" checked={...} /> |
| select | Drop-down selection | <select value={val} onChange={...}>...</select> |
| file | Upload file | <input type="file" onChange={handleFile} /> |
| number | Numeric input | <input type="number" value={val} onChange={...} /> |

1. **Handling React Forms**

* Form handling includes:
* Setting up state for each input.
* Updating values on onChange.
* Validating data before submission.
* Example:

function UserForm() {

const [name, setName] = useState('');

const handleChange = (e) => {

setName(e.target.value);

};

return (

<form>

<input type="text" value={name} onChange={handleChange} />

</form>

);

}

1. **Submitting Forms in React**

React forms are submitted using the onSubmit event, and default form behavior is prevented to stop page reloads.

* Example:

function ContactForm() {

const [email, setEmail] = useState('');

const handleSubmit = (e) => {

e.preventDefault();

console.log("Form Submitted with email:", email);

};

return (

<form onSubmit={handleSubmit}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />

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

</form>

);

}

Optional: Combine validation inside handleSubmit before processing form data.

**Conclusion**

React forms offer a high level of flexibility and control compared to traditional HTML forms. By using controlled components, React ensures form inputs are tied to component state, allowing for real-time validation, better form handling, and customizable form submission logic. Understanding how to manage different input controls and handle events makes building interactive and user-friendly forms in React efficient and scalable.