**Hands-on: 15. ReactJS-HOL**

**Introduction**

Handling user input is a critical part of any interactive application. In React, forms are used to collect data from users. Unlike traditional HTML forms, React treats forms as controlled components, where form data is handled by the component state. This gives developers more control over how input values are managed and validated. This section covers React forms, controlled components, different input types, handling form data, and form submission.

1. **React Forms**

React forms are used to gather input from users through fields like text boxes, checkboxes, and dropdowns. Unlike plain HTML, React forms are typically state-driven.

* Example:

function MyForm() {

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

return (

<form>

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

</form>

);

}

* In this example:
* The input’s value is controlled via state.
* onChange updates the state on every keystroke.

1. **Controlled Components**

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

* Example:

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

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

* Features:
* Keeps form input in sync with the component state.
* Allows validation, formatting, and transformation in real-time.
* Ensures consistent UI behavior and testability.

1. **Various Input Controls**

React supports all standard HTML form elements, all of which can be turned into controlled components.

Common input controls:

|  |  |  |
| --- | --- | --- |
| Input Type | Description | Example |
| text | Single-line input | <input type="text" /> |
| textarea | Multi-line text input | <textarea /> |
| checkbox | Boolean input | <input type="checkbox" /> |
| radio | Select one from multiple options | <input type="radio" /> |
| select | Drop-down menu | <select><option /></select> |
| password | Hidden characters | <input type="password" /> |
| file | Upload files | <input type="file" /> |

Each of these inputs can be controlled with corresponding state and event handlers.

1. **Handling Forms**

Handling forms involves managing user input, updating state, and optionally validating data in real time.

* Example:

function LoginForm() {

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

const handleChange = (e) => {

setUsername(e.target.value);

};

return (

<form>

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

</form>

);

}

* Key tasks:
* Use onChange to detect and update values.
* Keep input values in sync with component state.
* Optionally validate input on every change.

1. **Submitting Forms**

Form submission is handled using the onSubmit event of the <form> element.

Example:

function ContactForm() {

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

const handleSubmit = (e) => {

e.preventDefault(); // Prevent page reload

alert(`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>

);

}

* Best Practices:
* Always use e.preventDefault() to stop default behavior.
* Validate form data before processing.
* Use a state management pattern (like lifting state up or using context) if multiple components depend on form data.

**Conclusion**

React provides a structured and powerful way to manage form inputs using controlled components. By binding input fields to state and handling form events through event handlers like onChange and onSubmit, developers can maintain full control over form behavior. Understanding how to use different input controls and managing form submission ensures that forms remain reliable, scalable, and user-friendly in any React application.