

1. React forms are used to handle user input, such as text, selections, and file uploads. Unlike traditional HTML forms where the browser manages the form data, in React, the state of the form is typically controlled by the component itself. This is achieved by using the component's state to store the input values and handling changes to those values with event handlers. This approach ensures that the single source of truth for the form data is the component's state, making the form behavior predictable and easy to manage.
2. A controlled component is an input form element whose value is controlled by React's state. The value of the input is always driven by the state of the component, and any change to the input (e.g., a user typing in a text field) is handled by an event handler (like onChange) which updates the component's state. This makes the state the "single source of truth" for the input's value. The component's state is then used to set the value prop of the input element, ensuring that the displayed value always reflects the current state.
3. React supports a variety of input controls, which can be managed as controlled components. These include:

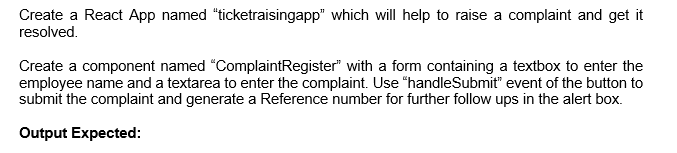
* <input type="text">: A single-line text input field. The value is controlled by the component's state.
* <textarea>: A multi-line text input area. In React, a <textarea> is handled similarly to an <input>, with its value being set by the value prop.
* <select>: A dropdown list. The selected option is controlled by the value prop on the <select> tag itself, rather than on the individual <option> tags.
* <input type="checkbox">: A checkbox that can be checked or unchecked. Its state is managed by the checked prop, and changes are handled by the onChange event.
* <input type="radio">: Radio buttons that allow a single selection from a group. The selected option is managed by the checked prop.

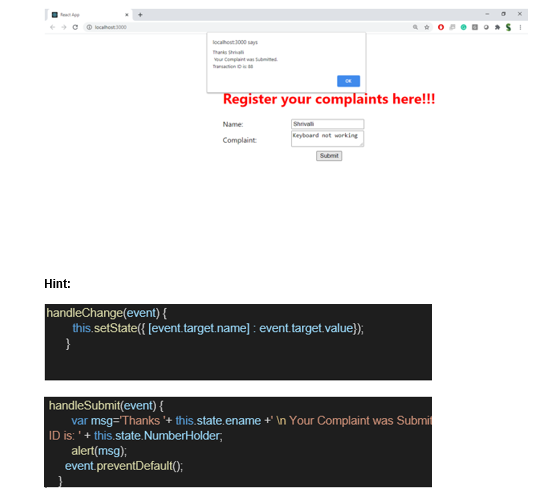
1. Handling forms in React involves managing the state of the form inputs and responding to user actions. The typical process is as follows:

* Initialize state: Set up the component's state to hold the initial values of all form inputs.
* Attach event handlers: For each input element, attach an onChange event handler. This function will be called whenever the input's value changes.
* Update state: Inside the onChange handler, update the component's state with the new value from the input. This is done by calling this.setState (in class components) or using the state setter function (in functional components).
* Bind value: Bind the value prop of the input element to the corresponding state variable. This ensures that the displayed value of the input always reflects the component's state.

1. Submitting a form in React is handled by a special event handler attached to the form element itself.

* Prevent default behavior: When a user submits a form, the default browser behavior is to refresh the page. To prevent this, you must call event.preventDefault() inside the onSubmit event handler.
* Define onSubmit handler: Attach an onSubmit event handler to the <form> element. This function will be triggered when the user clicks a submit button or presses Enter.
* Process form data: Inside the onSubmit handler, you can access the form data from the component's state. We can then use this data to perform actions like making an API call, updating a parent component's state, or navigating to a new page.
* Clear form (optional): After the form is successfully submitted, we can reset the component's state to clear the form fields for the next submission.





**Solution**

**App.js**

import logo from './logo.svg';

import React from 'react';

import './App.css';

import ComplaintRegister from './ComplaintRegister';

function App() {

return (

<div className="App">

<ComplaintRegister />

</div>

);

}

export default App;

**ComplaintRegister.js**

import React, { Component } from 'react';

import './ComplaintRegister.css';

class ComplaintRegister extends Component {

constructor(props) {

super(props);

this.state = {

ename: '',

complaint: '',

NumberHolder: Math.floor(Math.random() \* 1000)

};

}

handleChange = (event) => {

this.setState({ [event.target.name]: event.target.value });

}

handleSubmit = (event) => {

const { ename, complaint, NumberHolder } = this.state;

const msg = `Thanks ${ename}\nYour Complaint was Submitted.\nTransaction ID: ${NumberHolder}`;

alert(msg);

event.preventDefault();

}

render() {

return (

<div className="complaint-form">

<h2>Register your complaints here!!!</h2>

<form onSubmit={this.handleSubmit}>

<div>

<label>Name: </label>

<input

type="text"

name="ename"

value={this.state.ename}

onChange={this.handleChange}

required

/>

</div>

<div>

<label>Complaint: </label>

<textarea

name="complaint"

value={this.state.complaint}

onChange={this.handleChange}

required

/>

</div>

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

</form>

</div>

);

}

}

export default ComplaintRegister;

**ComplaintRegister.css**

.complaint-form {

text-align: center;

padding: 20px;

}

.complaint-form h2 {

color: red;

font-weight: bold;

}

.complaint-form form {

display: inline-block;

margin-top: 20px;

text-align: left;

}

.complaint-form input,

.complaint-form textarea {

width: 100%;

margin-top: 5px;

margin-bottom: 15px;

}

.complaint-form button {

width: 100%;

padding: 8px;

}

**Output**

