## 1. Explain about React Forms

React forms are used to capture user input and manage it using the component's state. Unlike traditional HTML forms, where the browser handles the form submission, React lets developers control every aspect of form behavior through JavaScript. This gives you full control over validation, formatting, and user interaction.

Forms in React are built using standard HTML form elements such as <input>, <textarea>, and <select>. The key difference is that in React, the form elements are typically tied to state variables. This two-way data binding ensures that any user input updates the state and any state update reflects on the form UI.

This approach provides great flexibility and ensures that the data is always in sync with the UI. It also allows the developer to perform custom validations, transformations, and interactions at any point during the user’s interaction with the form.

## 2. Define Controlled Components

Controlled components are form elements that are controlled by React's state. In this approach, the value of the form element is set by the state, and any change in the element updates the state using event handlers. This is the most common way of handling forms in React.

For example, if you use an <input> element, its value is not set independently, but by a state variable like name. When the user types into the input field, an onChange handler updates the state, and the input reflects the updated state. This ensures one source of truth for your form data — the component's state.

The advantage of controlled components is predictability. You can validate input, enforce formatting, and conditionally disable or enable fields based on state. Controlled components make it easier to implement form logic in a structured and testable way.

## 3. Explain about Various Input Controls

React supports several types of input controls to capture user input. These include <input type="text">, <input type="checkbox">, <input type="radio">, <textarea>, and <select>. Each of these inputs can be used as a controlled component by tying them to state and managing changes via event handlers.

Text inputs are used for short strings or single-line inputs and are typically managed using a state variable and onChange. Textareas allow multi-line text input and behave similarly to text inputs but use a separate tag. Checkbox and radio inputs are used for boolean or mutually exclusive options and require slightly different handling in their event functions.

Select dropdowns allow the user to choose from multiple options and can also be managed using state. By customizing these input controls with validation, error messages, and dynamic behavior, developers can create complex, user-friendly forms.

## 4. Explain About Handling Forms

Form handling in React starts with creating a form UI and connecting it to state variables. Each form element has an associated piece of state and an onChange event handler. This way, any user input directly updates the state, and the UI remains in sync with the data.

You can handle multiple inputs using a single onChange handler by using the name attribute on inputs and dynamically updating state based on it. This is especially useful in large forms where manually writing individual handlers becomes repetitive. Form handling also includes validating inputs before submission and showing relevant feedback to users.

React provides flexibility in form management — you can manually handle everything using pure React or integrate form libraries like Formik or React Hook Form for more complex scenarios. These libraries simplify the process of managing field states, validations, and form submission.

## 5. Explain About Submitting Forms

Form submission in React is usually handled by intercepting the default behavior and implementing custom logic. This is done by defining an onSubmit handler for the form, which receives the event object and calls event.preventDefault() to stop the page from refreshing.

Within the onSubmit handler, you can access all form values through the component’s state. You can then validate them, send them to a backend server, or display confirmation messages. This flexibility allows developers to fully control what happens when the user submits the form.

Once the submission is complete, the state can be reset to clear the form fields. This controlled submission approach ensures a seamless user experience and allows integration with any backend or API.

# Ticket Raising Application Code

## App.js

import React from 'react';  
import ComplaintRegister from './ComplaintRegister';  
  
function App() {  
 return (  
 <div className="App">  
 <ComplaintRegister />  
 </div>  
 );  
}  
  
export default App;

## ComplaintRegister.js

import React, { useState } from 'react';  
  
function ComplaintRegister() {  
 const [name, setName] = useState('');  
 const [complaint, setComplaint] = useState('');  
  
 const handleSubmit = (e) => {  
 e.preventDefault();  
 const txnId = Math.floor(Math.random() \* 100);  
 alert(\`Thanks \${name}\nYour Complaint was Submitted.\nTransaction ID is: \${txnId}\`);  
 };  
  
 return (  
 <div style={{ textAlign: 'center' }}>  
 <h2 style={{ color: 'red' }}>Register your complaints here!!!</h2>  
 <form onSubmit={handleSubmit}>  
 <div>  
 <label>Name: </label>  
 <input  
 type="text"  
 value={name}  
 onChange={(e) => setName(e.target.value)}  
 required  
 />  
 </div>  
 <br />  
 <div>  
 <label>Complaint: </label>  
 <textarea  
 value={complaint}  
 onChange={(e) => setComplaint(e.target.value)}  
 required  
 />  
 </div>  
 <br />  
 <button type="submit">Submit</button>  
 </form>  
 </div>  
 );  
}  
  
export default ComplaintRegister;

## Output



