# React Forms - Theoretical Concepts

## 1. Explain React Forms Validation

React form validation ensures that user input meets specific criteria before submission. This is essential to ensure correct and secure data collection. In React, validations can be added using event handlers like onChange, onBlur, or onSubmit, often using JavaScript condition checks.

Validation logic can be written inline or extracted into helper functions. For example, to check if a field is empty or meets a specific length, developers use if conditions inside the form's submit handler. When an input doesn’t meet the criteria, feedback can be given using alerts, error messages, or by disabling the submit button.

Unlike plain HTML, React validation is usually integrated with state variables using useState. This makes it easy to track changes and show real-time validation messages. Libraries like Formik and Yup can also be used for advanced validation in larger apps.

## 2. Identify the Differences Between React Form and HTML Form

HTML forms are simple and depend heavily on the browser’s default behavior. They use basic attributes like required, minLength, and type to enforce rules. Validation and form data submission are often handled by the browser or server.

React forms, however, are JavaScript-driven and provide more control over how data is handled and validated. Each input field in a React form is usually a controlled component, where the value is stored in the component’s state. This allows for dynamic and real-time validation.

React also supports conditional rendering, making it easier to show/hide inputs based on other values. HTML forms are fine for simple cases, but React forms are essential for complex UIs, interactivity, and custom validations.

## 3. Explain About Controlled Components

A controlled component is an input element whose value is controlled by React state. In other words, any change made to the input is immediately reflected in the component’s state using useState. This allows React to manage form behavior.

Every input field, like <input>, <textarea>, and <select>, should have an onChange handler that updates the corresponding state variable. This way, the component has full control over the user input and can apply real-time validation or formatting rules.

Using controlled components helps in making the application predictable and easy to debug. It also allows developers to enable/disable buttons, show errors instantly, or manipulate the input values based on other UI conditions.

## 4. Identify Various React Form Input Controls

React supports various input controls like <input type="text">, <input type="password">, <input type="email">, <textarea>, <select>, and <checkbox>. Each of these can be turned into controlled components using useState.

These controls work similarly to their HTML counterparts but are enhanced in React with JavaScript state and event handling. For example, a checkbox can be checked or unchecked using a boolean state variable, and a dropdown can be dynamically populated based on props or external data.

Advanced form inputs such as date pickers, sliders, file uploads, and even rich-text editors can also be used. These are usually controlled using state and sometimes require additional libraries like react-select, react-datepicker, or formik.

## 5. Explain How to Handle React Forms

Handling forms in React involves tracking input changes using the onChange event and storing values in state using useState. This allows real-time updates and validations. Each input’s value should be tied to a state variable, making it a controlled component.

When the form is submitted, onSubmit is triggered. This function typically prevents the default form action using event.preventDefault() and performs validation before allowing any further action. The validated data can then be used, saved to a database, or sent to a server.

Handling forms also includes rendering error messages, resetting fields, and controlling submit button states. This makes user interaction smooth and responsive. React makes it easier to modularize these behaviors by splitting form logic into reusable components.

## 6. Explain About Submitting Forms in React

Form submission in React is usually done by attaching an onSubmit event handler to the <form> element. This handler typically prevents the default submission action using event.preventDefault(), allowing developers to handle the data using JavaScript.

Within the onSubmit function, validations are performed to ensure all input fields are valid. After successful validation, the form data can be logged, sent to an API, or used to trigger UI changes. This ensures that the form submission is fully controlled by React logic.

React provides flexibility in what happens after submission. For example, developers can clear the form, show a success message, or redirect the user. This approach avoids full-page reloads and results in a smoother user experience compared to traditional HTML forms.

**Register.js**

import React, { useState } from "react";

function Register() {

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

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

const [password, setPassword] = useState("");

const handleSubmit = (e) => {

e.preventDefault();

if (name.length < 5) {

alert("Full Name must be 5 characters long!");

return;

}

if (!email.includes("@") || !email.includes(".")) {

alert("Enter a valid email address!");

return;

}

if (password.length < 8) {

alert("Password must be at least 8 characters long!");

return;

}

alert("Registration successful!");

};

return (

<div style={{ textAlign: "center", marginTop: "50px" }}>

<h2 style={{ color: "red", fontWeight: "bold" }}>Register Here!!!</h2>

<form onSubmit={handleSubmit}>

<div>

<label>Name: </label>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

/>

</div>

<br />

<div>

<label>Email: </label>

<input

type="text"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

</div>

<br />

<div>

<label>Password: </label>

<input

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

/>

</div>

<br />

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

</form>

</div>

);

}

export default Register;

**App.js**

import React from "react";

import Register from "./Register";

function App() {

return (

<div>

<Register />

</div>

);

}

export default App;

**Output**





