

1. React forms validation is the process of ensuring that the data submitted by a user meets specific criteria before it is processed. In React, this is typically handled by the component itself, often within the onChange or onSubmit event handlers. The validation logic can be implemented in a few ways:

* Inline Validation: Validation checks are performed as the user types, providing immediate feedback. This is done inside the onChange handler, where the input value is checked against a set of rules.
* On Submit Validation: All validation checks are performed at once when the form is submitted. This is done inside the onSubmit handler, where the component's state is checked for validity before proceeding.
* External Libraries: For complex validation, developers often use third-party libraries like Formik, React Hook Form, or Yup, which provide pre-built solutions for managing form state, validation rules, and error messages.

|  |  |
| --- | --- |
| React Form | HTML Form |
| The state of the form (i.e., the values of its inputs) is managed by the React component. | The state of the form is managed by the browser's DOM. |
| The data flow is unidirectional. User input triggers an event handler that updates the component's state, and the state then dictates the value of the input element. | Data flow is bidirectional. User input directly changes the value of the input element in the DOM. |
| Validation logic is typically implemented within the React component's code, either inline or on submit, and error messages are rendered based on the component's state. | Validation is handled by built-in HTML attributes (e.g., required, pattern) or by client-side JavaScript that manipulates the DOM |
| Submission is handled by an onSubmit event handler that calls event.preventDefault() to stop the browser's default behavior. The form data is accessed from the component's state. | Form submission by default causes a full page reload, and the data is sent to a server. |
| Forms are typically "controlled," meaning the value of each input is explicitly set by the component's state. | Forms are "uncontrolled" by default, meaning the browser maintains the state of the inputs internally. |

1. A controlled component is an input element (like <input>, <textarea>, or <select>) whose value is controlled by React's state. This means that the value displayed in the input field is always determined by a state variable.
2. React forms can use a variety of input controls, which are managed as controlled components:

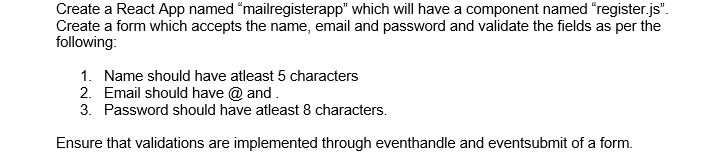
* Text Inputs: <input type="text"> for single-line text and <textarea> for multi-line text.
* Checkboxes: <input type="checkbox"> for boolean values, where the state is managed by the checked prop.
* Radio Buttons: <input type="radio"> for selecting one option from a group.
* Select Lists: <select> and its child <option> elements, where the selected value is managed by the value prop on the <select> tag.
* File Input: <input type="file"> is an exception and is typically an uncontrolled component because its value is read-only.

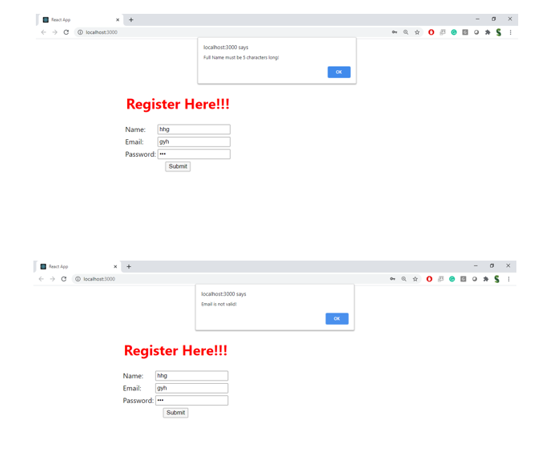
1. Handling a form in React involves managing the state of its input elements. The process generally follows these steps:

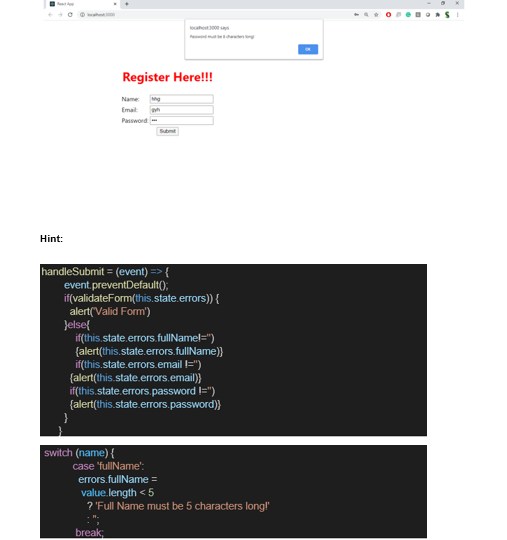
* Define Initial State: Use useState in a functional component or this.state in a class component to create state variables for each form input.
* Create an onChange Handler: Write a function that will be called whenever an input value changes. This function will take the event object as an argument.
* Update State: Inside the onChange handler, use the event object to get the new value (event.target.value) and update the corresponding state variable using the state setter function or this.setState.
* Bind Value: In the JSX, set the value prop of each input element to its corresponding state variable.

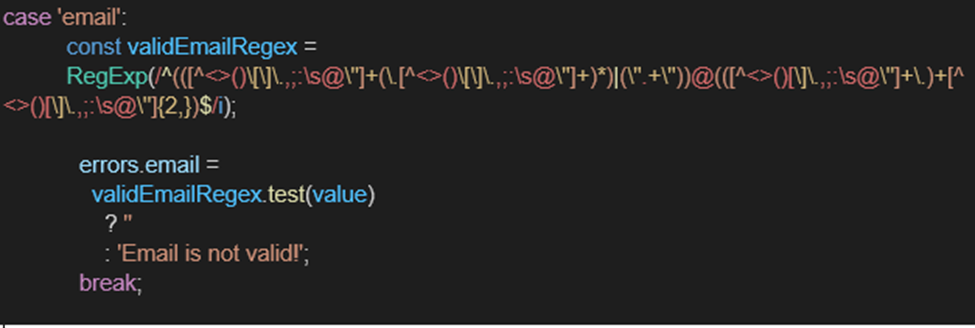
1. Submitting a form in React involves handling the onSubmit event of the <form> element. The process is as follows:

* Create an onSubmit Handler: Define a function that will be executed when the form is submitted.
* Prevent Default Behavior: The very first line of the onSubmit handler should be event.preventDefault(). This is crucial because it stops the browser from performing its default action of refreshing the page, which would cause the component's state to be lost.
* Access Form Data: The form's data is readily available in the component's state. We can now access this data to perform a desired action.
* Perform Actions: The data can be used to make an API call to a server, update a global state management system, or perform any other necessary logic.
* Reset Form (Optional): After the form has been successfully submitted, we can reset the component's state to clear the form fields for the next submission.









**Solution**

**register.js**

import React, { Component } from 'react';

import './register.css';

class Register extends Component {

constructor(props) {

super(props);

this.state = {

fullName: '',

email: '',

password: '',

errors: {

fullName: '',

email: '',

password: ''

}

};

}

handleChange = (event) => {

event.preventDefault();

const { name, value } = event.target;

let errors = this.state.errors;

switch (name) {

case 'fullName':

errors.fullName = value.length < 5

? 'Full Name must be 5 characters long!'

: '';

break;

case 'email':

const validEmailRegex = RegExp(/^[^\s@]+@[^\s@]+\.[^\s@]+$/);

errors.email = validEmailRegex.test(value)

? ''

: 'Email is not valid!';

break;

case 'password':

errors.password = value.length < 8

? 'Password must be 8 characters long!'

: '';

break;

default:

break;

}

this.setState({ errors, [name]: value });

};

validateForm = (errors) => {

let valid = true;

Object.values(errors).forEach((val) => {

if (val.length > 0) valid = false;

});

return valid;

};

handleSubmit = (event) => {

event.preventDefault();

const { fullName, email, password } = this.state;

if (fullName.length < 5) {

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

return;

}

const validEmailRegex = RegExp(/^[^\s@]+@[^\s@]+\.[^\s@]+$/i);

if (!validEmailRegex.test(email)) {

alert("Email is not valid!");

return;

}

if (password.length < 8) {

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

return;

}

alert("Valid Form");

};

render() {

return (

<div className="form-container">

<h2 className="form-title">Register Here!!!</h2>

<form onSubmit={this.handleSubmit}>

<div>

Name: <input type="text" name="fullName" onChange={this.handleChange} />

</div>

<div>

Email: <input type="text" name="email" onChange={this.handleChange} />

</div>

<div>

Password: <input type="password" name="password" onChange={this.handleChange} />

</div>

<div>

<input type="submit" value="Submit" />

</div>

</form>

</div>

);

}

}

export default Register;

**register.css**

.form-container {

width: 400px; /\* Increased width \*/

margin: 50px auto; /\* Center the form and add vertical spacing \*/

padding: 30px;

font-family: Arial, sans-serif;

}

.form-title {

color: red;

text-align: center;

margin-bottom: 20px;

}

input[type="text"],

input[type="password"] {

width: 50%;

padding: 10px;

margin: 10px 0 20px 0;

align-items: baseline;

text-align: justify;

}

input[type="submit"] {

width: 50%;

padding: 10px;

cursor: pointer;

align-items: baseline;

}

**App.js**

//import logo from './logo.svg';

import React from 'react';

import Register from './register';

import './App.css';

function App() {

return (

<div className="App">

<Register />

</div>

);

}

export default App;

**Output**

