

React Forms

Forms are an integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users. Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

Creating Form

React offers a stateful, reactive approach to build a form. The component rather than the DOM usually handles the React form. In React, the form is usually implemented by using controlled components.

There are mainly two types of form input in React.

1. Uncontrolled component
2. Controlled component

Uncontrolled component

The uncontrolled input is similar to the traditional HTML form inputs. The DOM itself handles the form data. Here, the HTML elements maintain their own state that will be updated when the input value changes. To write an uncontrolled component, you need to use a ref to get form values from the DOM. In other words, there is no need to write an event handler for every state update. You can use a ref to access the input field value of the form from the DOM.

Example

In this example, the code accepts a field **username** and **company name** in an uncontrolled component.

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.updateSubmit = this.updateSubmit.bind(this);
    this.input = React.createRef();
  }
  updateSubmit(event) {
    alert('You have entered the UserName and CompanyName successfully.');
```

```

}
render() {
  return (
    <form onSubmit={this.updateSubmit}>
      <h1>Uncontrolled Form Example</h1>
      <label>Name:
        <input type="text" ref={this.input} />
      </label>
      <label>
        CompanyName:
        <input type="text" ref={this.input} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
}
export default App;

```

Output

When you execute the above code, you will see the following screen.



The screenshot shows a web browser window at localhost:8080. The page title is "Uncontrolled Form Example". Below the title, there are two text input fields. The first is labeled "Name:" and the second is labeled "CompanyName:". To the right of the "CompanyName" field is a "Submit" button.

After filling the data in the field, you get the message that can be seen in the below screen.



The screenshot shows the same web browser window, but now the "Name" field contains the text "Abhishek" and the "CompanyName" field contains "JavaTpoint". A modal dialog box is displayed over the form, indicating a successful submission. The dialog box has a title bar that says "localhost:8080 says" and the main text "You have entered the UserName and CompanyName successfully." with an "OK" button.

Controlled Component

In HTML, form elements typically maintain their own state and update it according to the user input. In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with **setState()** method.

Controlled components have functions that govern the data passing into them on every **onChange event**, rather than grabbing the data only once, e.g., when you click a **submit button**. This data is then saved to state and updated with `setState()` method. This makes component have better control over the form elements and data.

A controlled component takes its current value through **props** and notifies the changes through **callbacks** like an `onChange` event. A parent component "controls" this changes by handling the callback and managing its own state and then passing the new values as props to the controlled component. It is also called as a "dumb component."

Example

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ""};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value: event.target.value});
  }
  handleSubmit(event) {
    alert('You have submitted the input successfully: ' + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <h1>Controlled Form Example</h1>
        <label>
```

Name:

```
<input type="text" value={this.state.value} onChange={this.handleChange} />
```

```
</label>
```

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

```
</form>
```

```
);
```

```
}
```

```
}
```

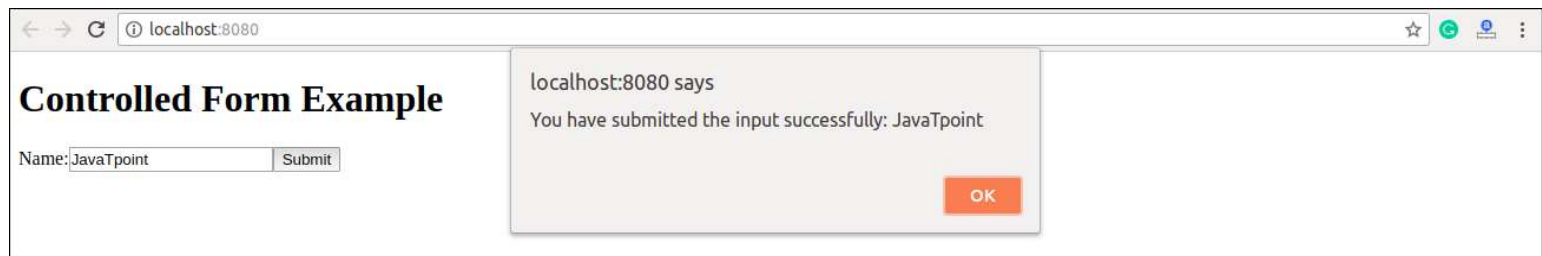
```
export default App;
```

Output

When you execute the above code, you will see the following screen.



After filling the data in the field, you get the message that can be seen in the below screen.



Handling Multiple Inputs in Controlled Component

If you want to handle multiple controlled input elements, add a **name** attribute to each element, and then the handler function decided what to do based on the value of **event.target.name**.

Example

```
import React, { Component } from 'react';
```

```
class App extends React.Component {
```

```
  constructor(props) {
```

```
    super(props);
```

```
    this.state = {
```

```

    personGoing: true,
    numberOfPersons: 5
  };
  this.handleInputChange = this.handleInputChange.bind(this);
}
handleInputChange(event) {
  const target = event.target;
  const value = target.type === 'checkbox' ? target.checked : target.value;
  const name = target.name;
  this.setState({
    [name]: value
  });
}
render() {
  return (
    <form>
      <h1>Multiple Input Controlled Form Example</h1>
      <label>
        Is Person going:
        <input
          name="personGoing"
          type="checkbox"
          checked={this.state.personGoing}
          onChange={this.handleInputChange} />
      </label>
      <br />
      <label>
        Number of persons:
        <input
          name="numberOfPersons"
          type="number"
          value={this.state.numberOfPersons}
          onChange={this.handleInputChange} />
      </label>
    </form>
  );
}

```

```
}  
}  
  
export default App;
```

Output



A screenshot of a web browser window with the address bar showing 'localhost:8080'. The page title is 'Multiple Input Controlled Form Example'. The form contains two labels: 'Is Person going:' followed by a checked checkbox, and 'Number of persons:' followed by a text input field containing the value '5'.

← Prev

Next →

 [For Videos Join Our Youtube Channel: Join Now](#)


Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share




Learn Latest Tutorials


 [Splunk tutorial](#)
Splunk

 [SPSS tutorial](#)
SPSS

 [Swagger tutorial](#)
Swagger

 [T-SQL tutorial](#)
Transact-SQL

 [Tumblr tutorial](#)
Tumblr

 [React tutorial](#)
ReactJS

 [Regex tutorial](#)
Regex

 [Reinforcement learning tutorial](#)
Reinforcement Learning