**ReactJS: Uncontrolled Components vs Controlled Components**

Uncontrolled Components

With uncontrolled input, there is no updating or changing of any states. What you submit is what you get.

class Form extends React.Component {

formHandler(event){

event.preventDefault();

const name = this.input.value; //here we are accessing input elements value

}

render() {

return (

<div>

<form onSubmit={this.formHandler.bind(this)}>

<label>Name<label>

<input type='text' ref={(input) => this.input = input} />

<input type='submit'/>

</form>

</div>

)

}

}

The “ref” that we are using is a special attribute provided by Component – it takes the callback and when used on an HTML element, the argument it takes is the underlying DOM element (which in this case is input). Filling this form out with name “Peter”:

https://miro.medium.com/max/307/1*0qi9PgtTvttbefiSDmzO0Q.png



As you can see, the value of the input doesn’t change at all. Whatever you end up submitting will be the final value from the input field.

***Quick Note:*** *With uncontrolled values, the value attribute (value={value here}) inside the input element will override any values that are typed in so in order to work around that, React provides another attribute called defaultValue (defaultValue={value here}) that will pre-populate the input field with the defaultValue without overriding any value input by the user.*

This is most basic way of setting up a form. Although it is quicker to set this up, but it limits the potential of React.

Controlled Components

With Controlled inputs, there is always some sort of change going on the value of input field will always be a prop of class component (usually state). There will also be a callback that is necessary to handle the changes going on with the input field.

class Form extends React.Component {

constructor(props){

super(props);

this.state ={

name: ''

}

}

handleNameChange(event){

const inputName = event.target.value;

this.setState({

name: inputName;

})

}

render() {

return (

<div>

<form>

<label>Name<label>

<input type='text' value={this.state.name}

onChange = {this.handleNameChange.bind(this)} />

<input type='submit'/>

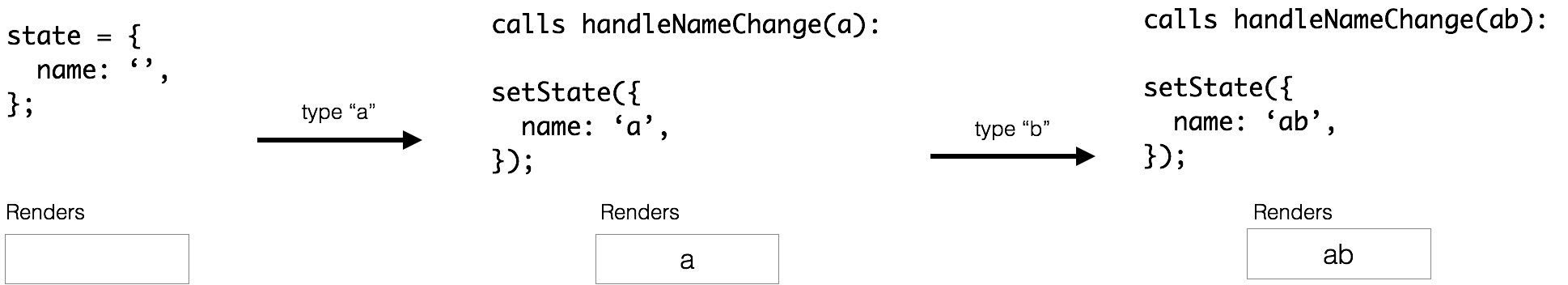
</form>

</div>

)

}

}



Here, we have the component’s state name as the value for the input field. The onChange handles the change in the input field and immediately updates the state of the component to the input of the user.



**When to use uncontrolled values vs controlled values**

Uncontrolled values are useful when the form is very basic with minimalistic features. They are also good options when dealing with libraries or other languages that don’t interact well with or don’t use React. The downside of using uncontrolled values is the fact that there is not much you can do with it — making it quite limited in functionality.

Controlled values are useful for things such as validation or instant user feedback. You can imagine writing an error code that alerts the user that the input from the user is invalid or doesn’t meet certain criteria (passwords for instance) which disappears once it is valid. The downside to using controlled values is that it requires much more code and there is a need to take care of the changes using callbacks. On top of that, every callback that you pass it through, needs to be a React component to process the changes. It’s possible to see how changing your pre-existing code framework to React would make this a tedious task.

There isn’t a right or wrong way to use these features. It all comes down to how the architect will use these forms and what will happen to the information that is being passed along. The design can get complicated when using controlled values but that allows for more functionality and a better user experience.