Form Handling in React

Introduction

Forms in React are handled primarily based on the manner in which data is managed. Form handling deals with how to handle the data when the input values are changed or when the form is submitted. Control these changes by adding event handlers to onChange and onSubmit respectively.

HTML Form elements such as <input>, <textarea> and <select> typically maintain their own state and update it based on the user input. The DOM becomes responsible for handling the form data.

Ways to create forms in React

➢ **Uncontrolled Components:** These React Components are traditional HTML form inputs which remember what you typed. **Refs are used get the form values.** Use the **defaultValue** property to specify initial value in React.

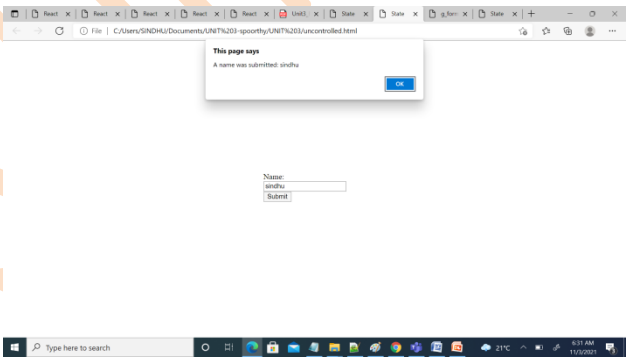
➢ **Controlled Components:** These are React components that render a form and also control what happens in that form on subsequent user input. This means that, as form value changes, the component that renders the form saves the value in its state. The controlled component is a way that you can handle the form input value using the **state** and changing the input value is possible using **setState.**

Coding example 1: Demo of uncontrolled components

```
<body>
  <div id="root"></div>
  <script type="text/babel">
      class UncontrolledForm extends React.Component {
          constructor() {
              super();  this.handleSubmit = this.handleSubmit.bind(this);
```

```
                this.input = React.createRef();
            }
            handleSubmit(event) {
                event.preventDefault();
                alert('A name was submitted: ' + this.input.current.value);
            }
            render() {
                return ( <form onSubmit={this.handleSubmit}>
                Name:<input className="input" type="text" ref={this.input} />
                <input type="submit" value="Submit" />
                </form>       );
                 }
          }
        ReactDOM.render(<UncontrolledForm/>, document.getElementById('root'));
</script>
</body>
```

Output: Input box is filled with a value and submit button is clicked

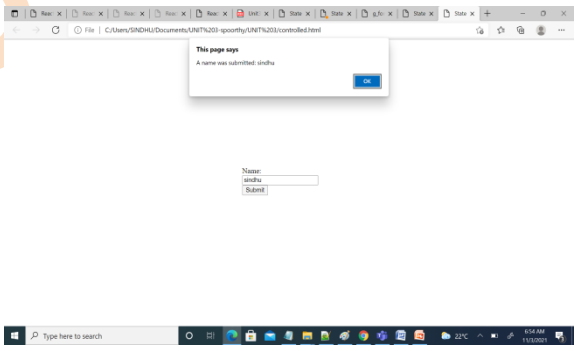


Coding example 2: Demo of controlled components

```
 <body>
   <div id="root"></div>
   <script type="text/babel">
    class ControlledForm extends React.Component {
        constructor(props) { super(props);     this.state = {value: ''};       }
```

```
handleChange=(event)=> {
  //        console.log(this.state.value)
  this.setState({value: event.target.value});        }
handleSubmit=(event)=> {
  event.preventDefault(); alert('A name was submitted: ' + this.state.value); }
render() {
  return (
  <form onSubmit={this.handleSubmit}>
     <label>
     Name:
     <input type="text" value={this.state.value} onChange={this.handleChange} />
     </label>
     <input type="submit" value="Submit" />
  </form> );
  }
}
ReactDOM.render(<ControlledForm/>, document.getElementById('root'));
</script>
</body>
```

Output: Input box is filled with a value and submit button is clicked

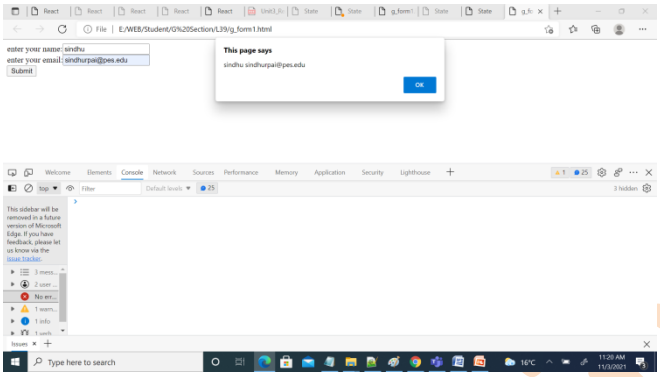


If there is more than one input field, can we have the same handler for onChange event? Explained through below example code

Coding Example 3:

```
<body>
    <div id = "root"> </div>
    <script type = "text/babel">
        class My_form extends React.Component
        {   constructor(){ super();
                this.state = {name:"",email:""  }
            }
            render() {
                return (<form onSubmit = {this.handleSubmit}>
                    <label>enter your name:<input type = "text" name =
"names" onChange = {this.handleChange}/></label><br/>
                    <label>enter your email:<input type = "email" name
= "email" onChange = {this.handleChange}/></label><br/>
                    <input type = "submit" value = "Submit" />
                </form>)                }
            handleChange=(event)=>
            {   var name1 = event.target.name
                var value1 = event.target.value
                if(name1 == "names")
                    this.setState({name:value1})
                if(name1 == "email")
                    this.setState({email:value1})
            }
            handleSubmit=(event)=>
            {   event.preventDefault();
                alert(this.state.name+" "+this.state.email)     }
        }
        ReactDOM.render(<My_form />, document.getElementById("root"))
</script>       </body>
```

Output:



Observation: The above code works absolutely fine. But if there are more number of fields in the form, those many key value pairs must be there in state object. Also, inside the handler, so many times if condition must be used to check for the equality of event.target.name. To avoid this, we use **...this.state.form** as shown in the below code.

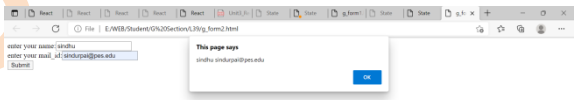
Handling Multiple inputs using React Way

Coding Example 4:Change in the creation of state object and accessing the setState function

```
<body>
<div id = "root"></div>
<script type = "text/babel">
        class My_form extends React.Component{
                constructor(){
                        super();
                        this.state = {form: {names:"", email:""}}
                }
                render(){
                        return (<form onSubmit = {this.handleSubmit}>
                        <label>enter your name:<input type = "text" name = "names"
onChange = {this.handleChange}/></label><br/>
                        <label>enter your mail_id:<input type = "email" name =
```

```
"email" onChange = {this.handleChange}/></label> <br/>
                    <input type = "submit" value = "Submit" />
                    </form> )                    }
                handleChange=(event)=>
                {       var name1 = event.target.name
                        var value1 = event.target.value
                        this.setState({
                                ...this.state.form,
                                form:{
                                        ...this.state.form,
                                        [name1]:[value1]
                                }
                        })                      }
                        handleSubmit=(event)=>{
                        event.preventDefault()
                        alert(this.state.form.names+ " "+this.state.form.email)   }
        }
        ReactDOM.render(<My_form />, document.getElementById("root"))
    </script>
</body>
```

Output:



- **Coding example 5:** Calculate the Body Mass Index of a person, given the height in meters and weight in kilograms. Also display appropriate message.

- bmi = weight/(height*height)
  - If bmi < 19, display “underweight”
  - If bmi is between 20 and 24, display “Normal”
  - Else display “overweight”

```
<body>
    <div id="root"></div>
    <script type="text/babel">
        class BMICalc extends React.Component
        {   constructor(){ super()
                this.setHRef=(el)=>{this.heightinput=el}
                this.setWRef=(el)=>{this.weightinput=el}
                this.setStatRef=(el)=>{this.statusoutput=el}
            }
            render()
            {   return(<div>  <form onSubmit = {this.handleSubmit}>
                <label>enter the height in meters:<input type = "text" ref
={this.setHRef}/></label> <br/>
                <label>enter the weight:<input type = "text" ref
={this.setWRef}/></label><br/>
                <input type = "submit" value = "calculate bmi"/>
                </form>
                <h2 ref={this.setStatRef}></h2>
                </div>
            }
            handleSubmit=(event)=>
            {
                event.preventDefault()
                var h = parseFloat(this.heightinput.value)
                var w = parseInt(this.weightinput.value)
                var bmistat;
                var bmi = w/(h*h);
```

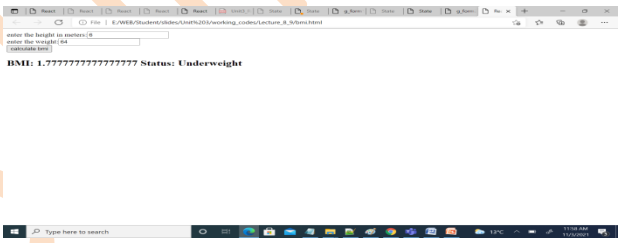
```
                    if(bmi<19)
                        bmistat="Underweight";
                    else if(bmi<25)
                        bmistat="Normal"
                    else
                        bmistat="Overweight"
                    this.statusoutput.innerHTML = "BMI: "+bmi+" Status:
"+bmistat
                }

            }
            ReactDOM.render(<BMICalc/>,document.getElementById("root"))

        </script>

</body>
```

Output:



Practice Problem

1. With the help of a form, accept first and last names from the user separately and display full name. Use one handler function for both the inputs. Additionally, display an error message on entering numerical characters.