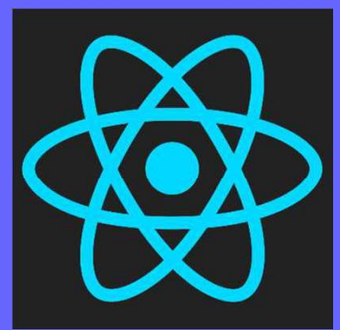
10

Styling React and Form Handling

Objectives

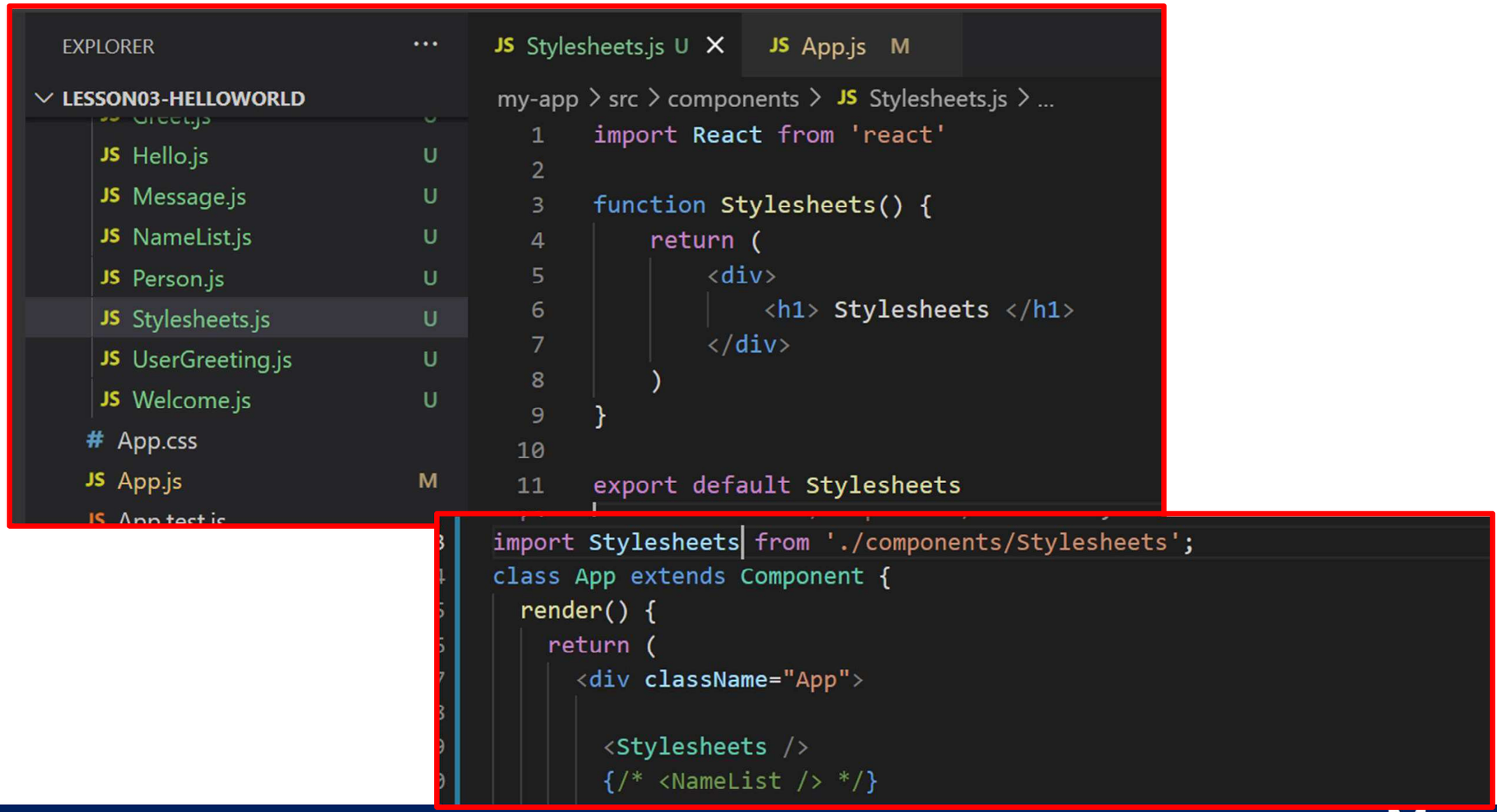
After completing this lesson, you should be able to do the following:

- Basics in State
- setState
- State, Events and Managed Controls

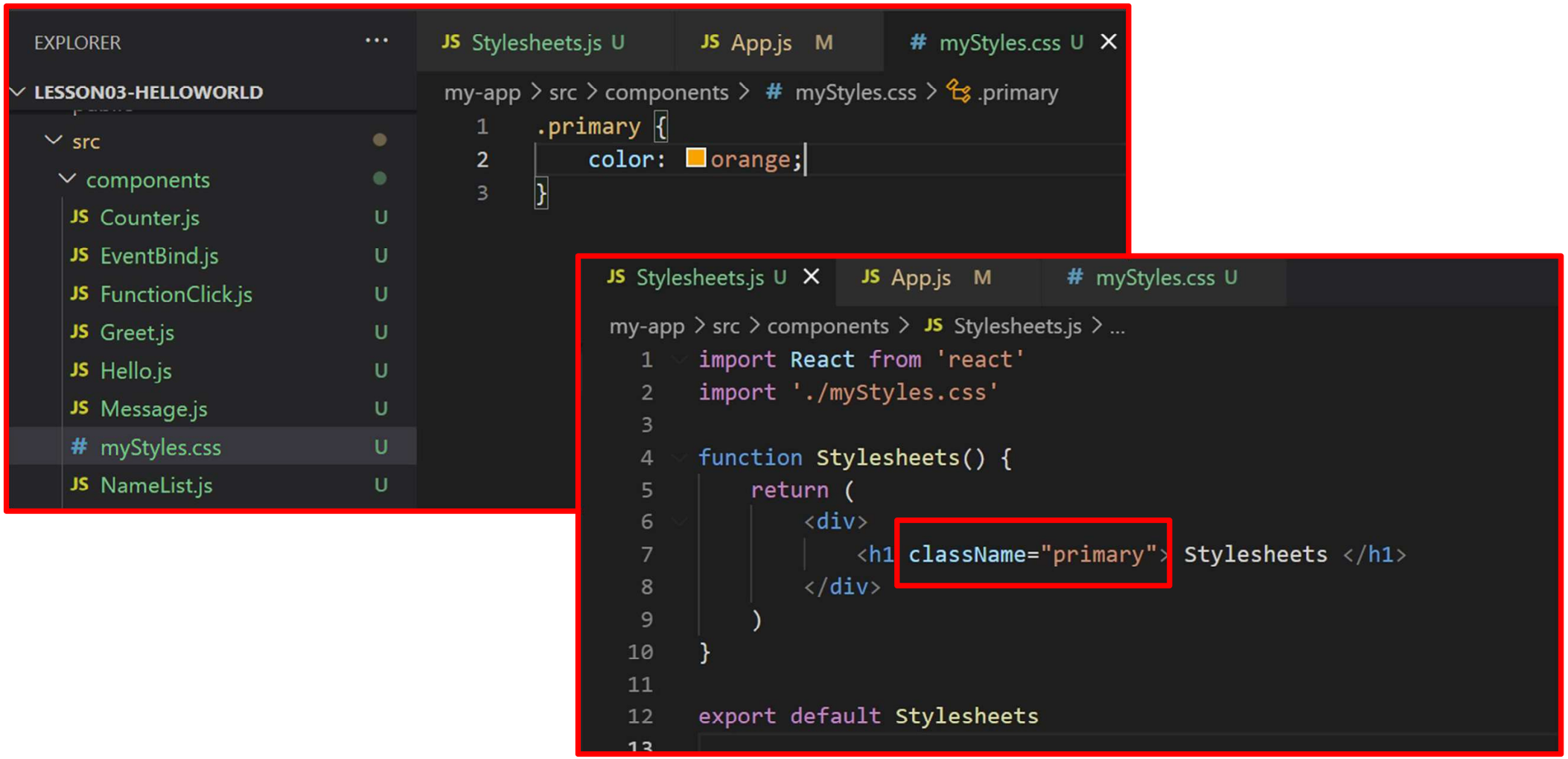


Styling React

- Styling React Components
  1. CSS Stylesheets
  2. Inline Styling
  3. CSS Modules
  4. CSS in JS Libraries




```
import Stylesheets from './components/Stylesheets';
class App extends Component {
  render() {
    return (
      <div className="App">

        <Stylesheets />
        {/* <NameList /> */}
```


EXPLORER
LESSON03-HELLOWORLD
src
components
Counter.js U
EventBind.js U
FunctionClick.js U
Greet.js U
Hello.js U
Message.js U
myStyles.css U
NameList.js U
Stylesheets.js U
App.js M
myStyles.css U
my-app > src > components > myStyles.css > .primary
1 .primary {
2 color: orange;
3 }
Stylesheets.js U
App.js M
myStyles.css U
my-app > src > components > Stylesheets.js > ...
1 import React from 'react'
2 import './myStyles.css'
3
4 function Stylesheets() {
5 return (
6
7 <h1 className="primary"> Stylesheets </h1>
8
9)
10 }
11
12 export default Stylesheets

Passing Style aspects Conditional Using props

JS Stylesheets.js U ✕ | JS App.js M | # myStyles.css U

my-app > src > components > JS Stylesheets.js > ...

```
1   import React from 'react'
2   import './myStyles.css'
3
4   function Stylesheets(props) {
5       let className = props.primary ? 'primary' : ''
6       return (
7           <div>
8               <h1 className={ className }> Stylesheets </h1>
9           </div>
10      )
11  }
12
13  export default Stylesheets
```

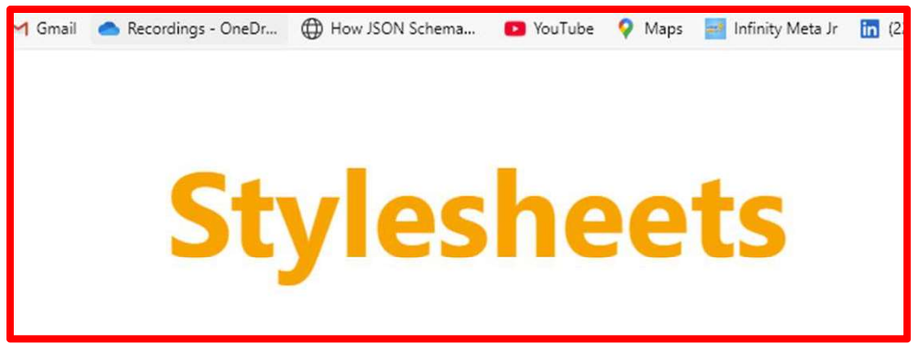
```
12  import NameList from './components/NameList';
13  import Stylesheets from './components/Stylesheets';
14  class App extends Component {
15    render() {
16      return (
17        <div className="App">
18
19          <Stylesheets primary= {true} />
```

Appling Two or More Classes

JS Stylesheets.js U | JS App.js M | # myStyles.css U

my-app > src > components > # myStyles.css > .font-xl

```
1   .primary {
2       color: orange;
3   }
4
5   .font-xl {
6       font-size: 72px;
7   }
```



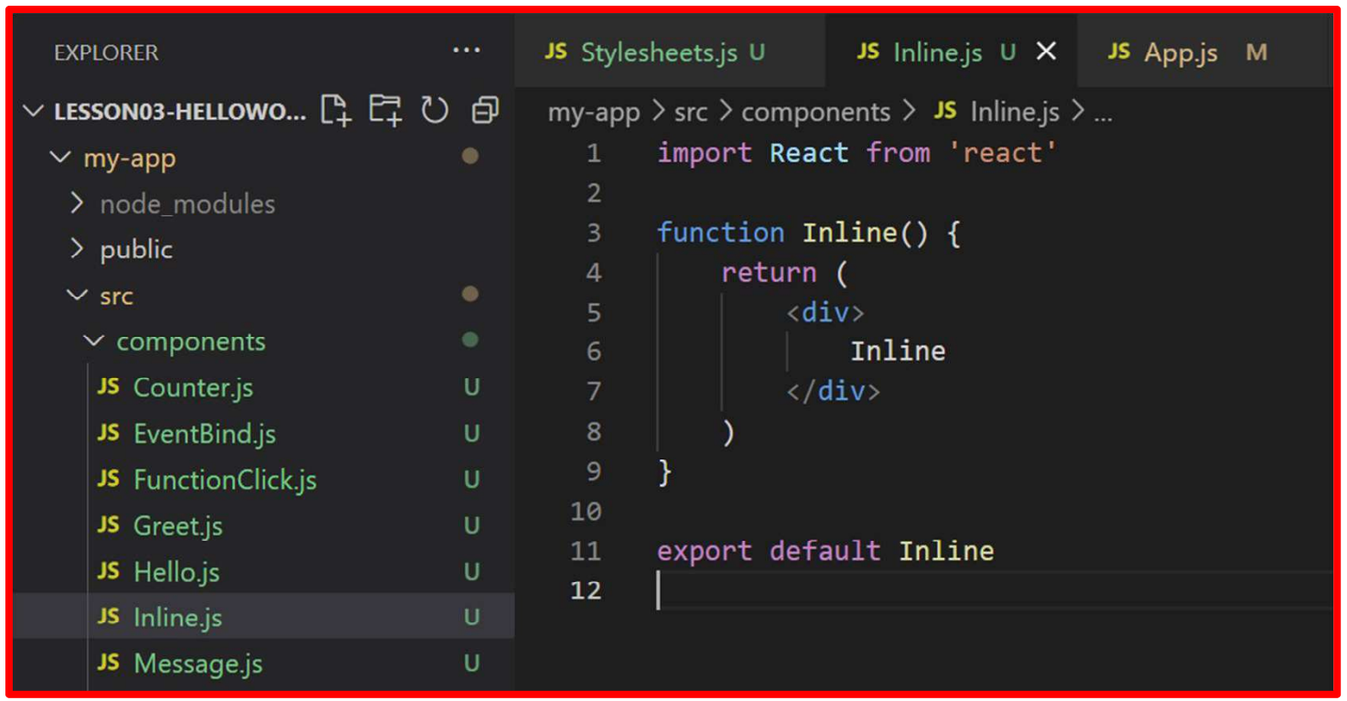
JS Stylesheets.js U | JS App.js M | # myStyles.css U

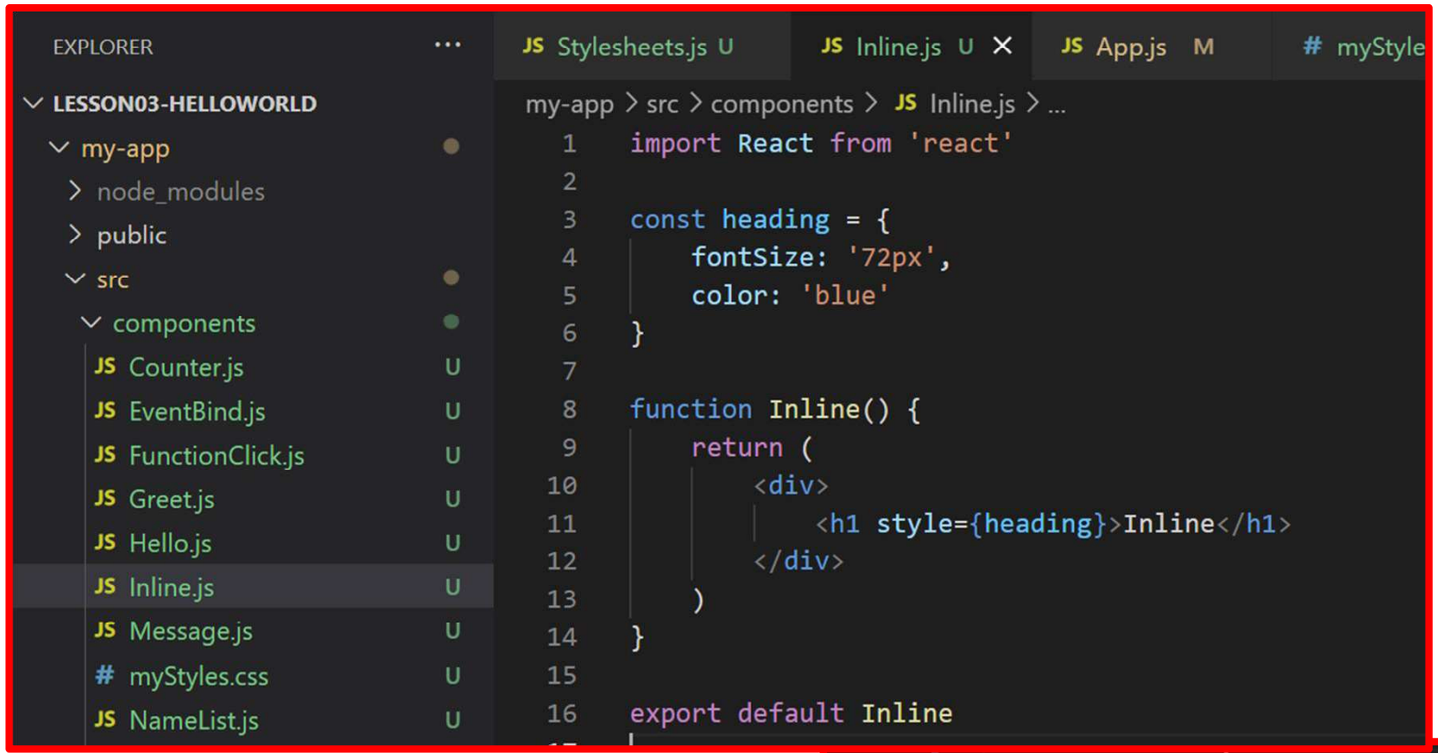
my-app > src > components > JS Stylesheets.js > ...

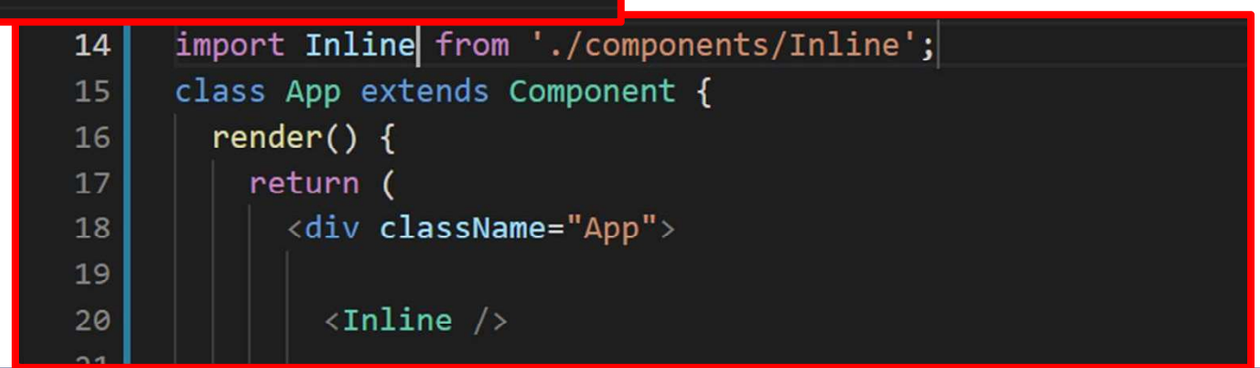
```
1   import React from 'react'
2   import './myStyles.css'
3
4   function Stylesheets(props) {
5       let className = props.primary ? 'primary' : ''
6       return (
7           <div>
8               <h1 className={ `${className} font-xl` }> Stylesheets </h1>
9           </div>
10      )
11  }
12
13  export default Stylesheets
14
```

Inline Styling

- In react inline styles are not specified as a string instead they are specified with an object whose key is the **Camel cased version of the style name** and the value is usually a string




EXPLORER
Stylesheets.js U
Inline.js U
App.js M
myStyle
LESSON03-HELLOWORLD
my-app
node_modules
public
src
components
Counter.js U
EventBind.js U
FunctionClick.js U
Greet.js U
Hello.js U
Inline.js U
Message.js U
myStyles.css U
NameList.js U
my-app > src > components > Inline.js > ...
1 import React from 'react'
2
3 const heading = {
4 fontSize: '72px',
5 color: 'blue'
6 }
7
8 function Inline() {
9 return (
10
11 <h1 style={heading}>Inline</h1>
12
13)
14 }
15
16 export default Inline


14 import Inline from './components/Inline';
15 class App extends Component {
16 render() {
17 return (
18
19
20 <Inline />

- Now there is a file naming convention **CSS Modules** to be used for CSS modules with create react app the file name must be suffixed with module.

Stylesheets.js U | Inline.js U | appStyles.css U

my-app > src > components > # appStyles.css > .error

```
1  .error {
2      color: red
3  }
```

appStyles.css U | appStyles.module.css U | App.js M

my-app > src > components > # appStyles.module.css > .success

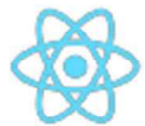
```
1  .success {
2      color: green
3  }
```

```
15  import './appStyle.css'
16  import styles from './appStyles.module.css'
17  class App extends Component {
18    render() {
19      return (
20        <div className="App">
21
22          <h1 className='error'>Error</h1>
23          <h1 className={styles.success}>Success </h1>
24          {/* <Inline /> */}
```

Error

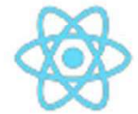
Success

- CSS kind of applies to every child component as well
- **CSS Modules** on the other hand because you reference the class names using the Styles variable it cannot be used in the children component



React and Bootstrap

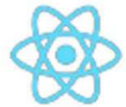
- React being a view library, does not have any built-in mechanism to help create designs that are responsive and intuitive.
- A front end design framework like Bootstrap can help alleviate these concerns.
- Integrating Bootstrap with React allows developers to use Bootstrap's grid system and various other components.



Adding Bootstrap for React

- Bootstrap can be added to the React application in three common ways:
  - Using the Bootstrap CDN
    - No installs required
  - Bootstrap as a dependency
    - A common option to add Bootstrap to the React application
    - Bootstrap, jquery and popper.js need to be installed using npm
  - React Bootstrap Package
    - A package that has rebuilt Bootstrap components to work as React components.

React Bootstrap Package

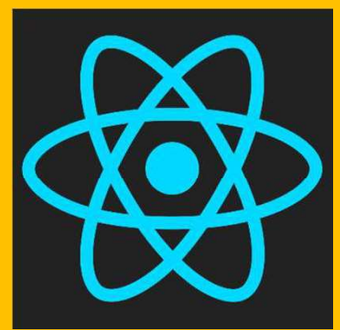


- There are a few libraries that create a React specific implementation of Bootstrap.
- `reactstrap` is a library that gives the ability to use Bootstrap components in React.
  - The module includes components for typography, icons, buttons etc.
- Install `bootstrap` and `reactstrap`

```
npm install --save bootstrap
npm install --save reactstrap
```

- Import Bootstrap CSS in `src/index.js`

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

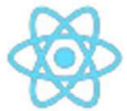


Form Handling in React

Introduction

- Basics of working with forms
- Let's capture input from form elements like the input tag text area tag and also a select tag and have the data available for form submission in regular HTML form

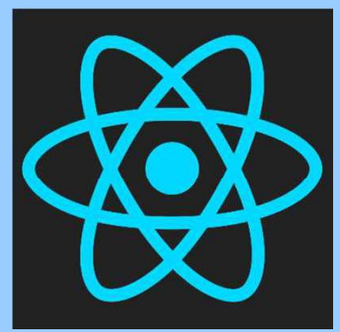
Forms



- Forms are integral to any modern application.
- They serve as a basic medium for users to interact with the application.
- Developers rely on forms for various capabilities such as securely logging in the user, building a cart, searching a product list etc.
- React does not provide comprehensive form validation support out of the box.
- Unlike other DOM elements, HTML form elements work differently in react.

Forms in React

- There are two types of form input in react.
- **Uncontrolled input**
  - Like traditional HTML form inputs, they remember what is typed.
  - `ref` is used to get the form values.
- **Controlled input**
  - This is when the react component that renders a form also controls what happens to the form on subsequent user input.
  - As the form value changes, the component that renders the form saves the value in its state.

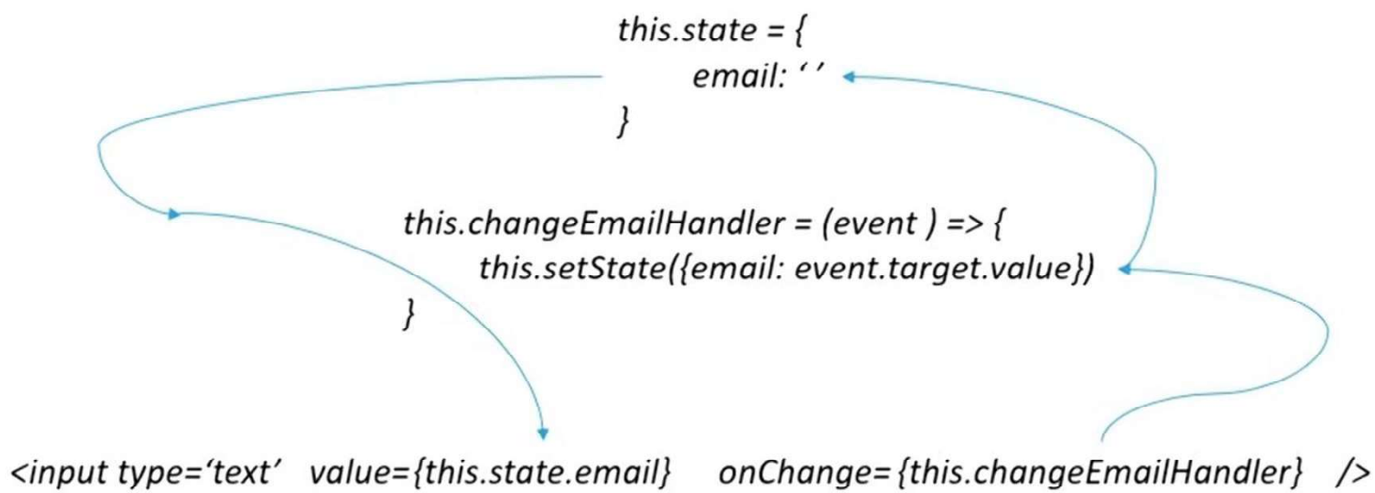


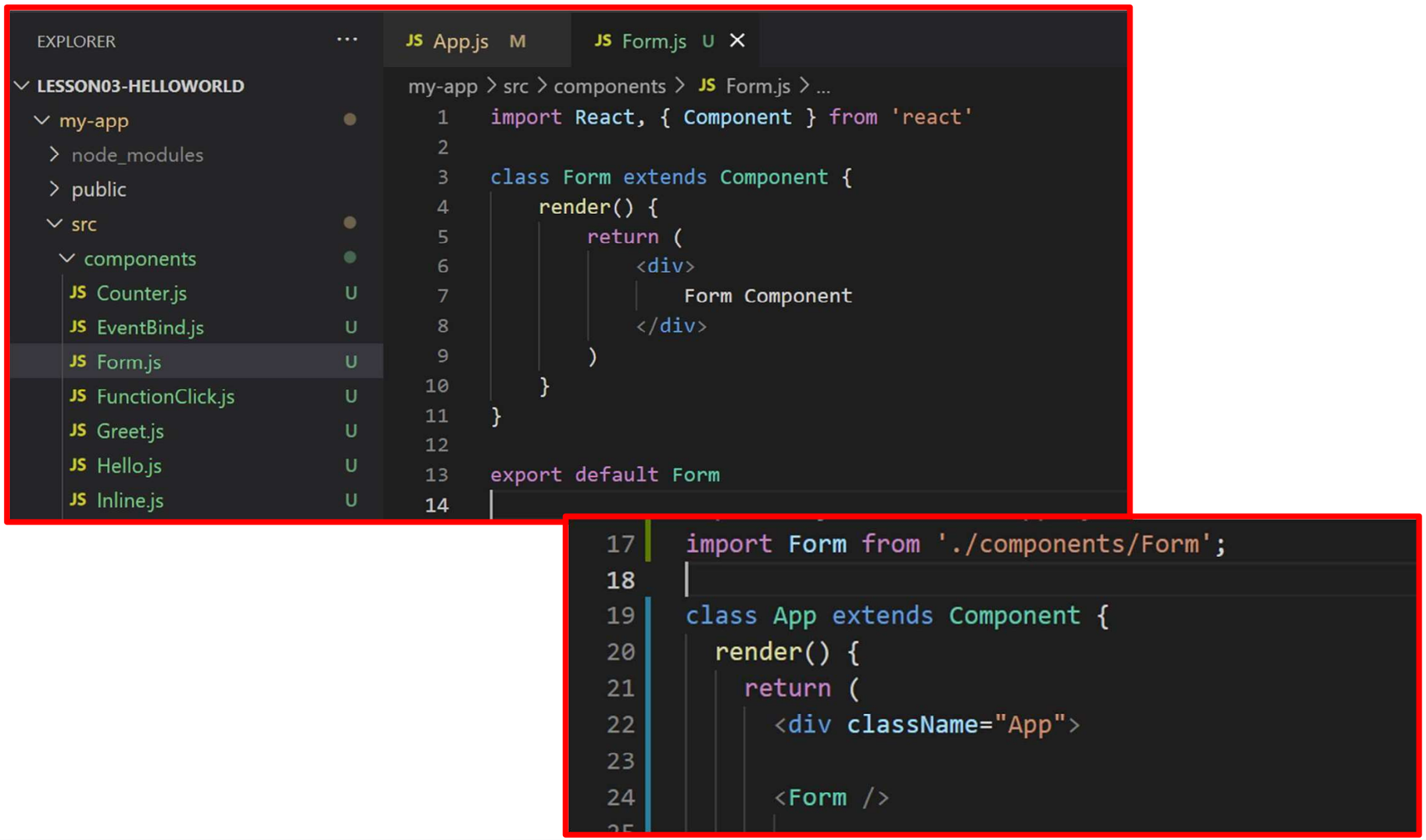
Controlled Components

Controlled Components

- Elements like input text area and so on are responsible on their own to handle the user input and update their respective values.
- But what we want is for react to control the form elements instead such form elements whose value is controlled by react is called a **Controlled Component**

Controlled components

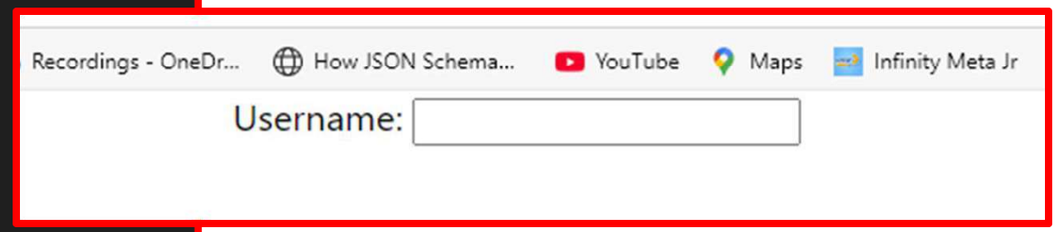



EXPLORER
App.js M
Form.js U
LESSON03-HELLOWORLD
my-app
node_modules
public
src
components
Counter.js U
EventBind.js U
Form.js U
FunctionClick.js U
Greet.js U
Hello.js U
Inline.js U
my-app > src > components > Form.js > ...
1 import React, { Component } from 'react'
2
3 class Form extends Component {
4 render() {
5 return (
6
7 Form Component
8
9)
10 }
11 }
12
13 export default Form
14
17 import Form from './components/Form';
18
19 class App extends Component {
20 render() {
21 return (
22
23
24 <Form />

JS App.js M JS Form.js U

my-app > src > components > JS Form.js > Form > render

```
1   import React, { Component } from 'react'
2
3   class Form extends Component {
4       render() {
5           return (
6               <form>
7                   <div>
8                       <label>Username: </label>
9                       <input type='text' />
10                  </div>
11              </form>
12          )
13      }
14  }
15
16  export default Form
17
```



- The label and the input field right now though the form is regular HTML it is not a controlled react component to convert this into a **Controlled Component**

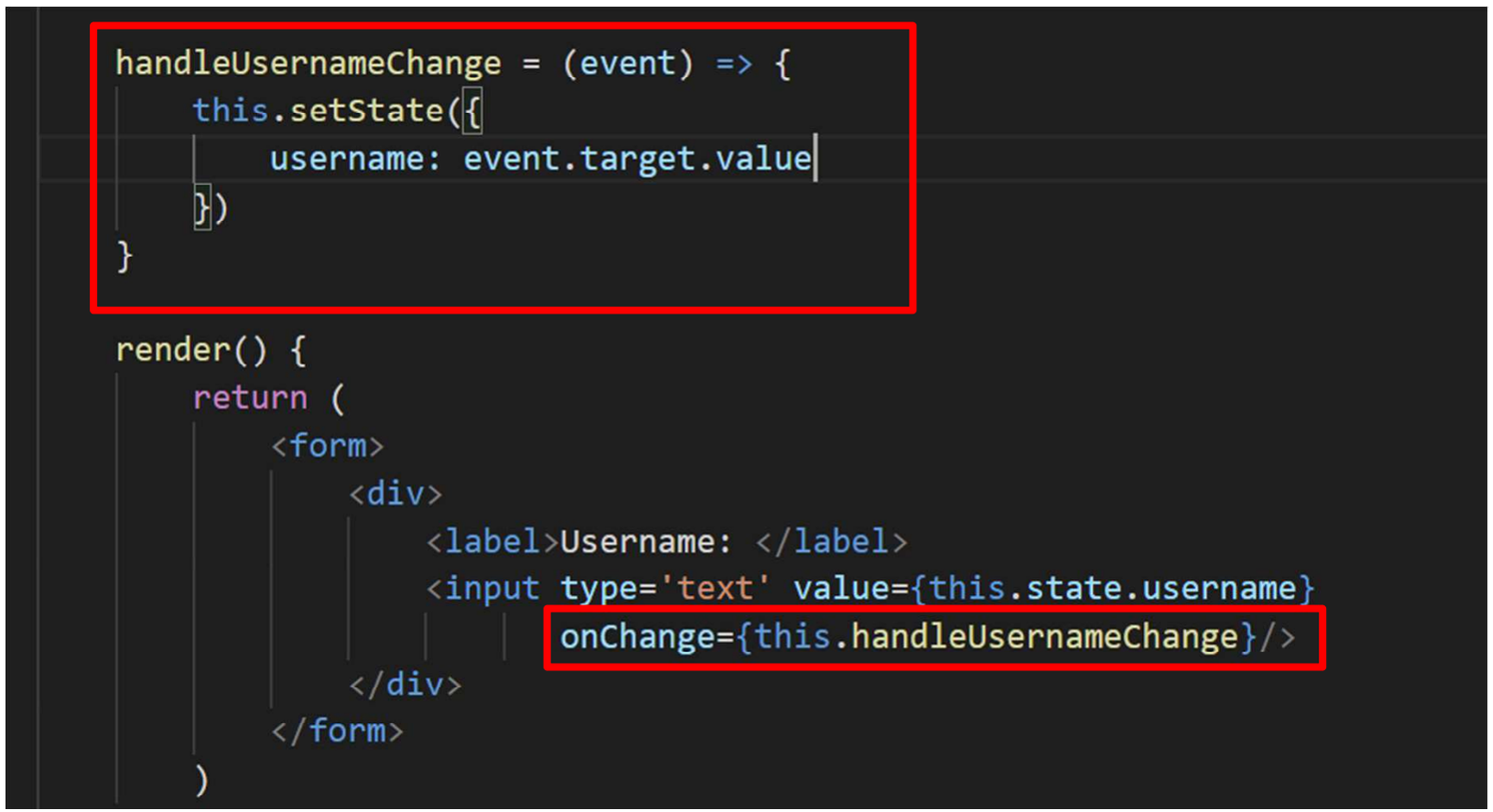
Step 1

- Create a component state that we control the value of the input element so within the component

App.js M | Form.js U

my-app > src > components > Form.js > Form > render

```
1   import React, { Component } from 'react'
2
3   class Form extends Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               username: ''
9           }
10      }
11
12      render() {
13          return (
14              <form>
15                  <div>
16                      <label>Username: </label>
17                      <input type='text' value={this.state.username} />
18                  </div>
19              </form>
20          )
21      }
22  }
```


handleUsernameChange = (event) => {
this.setState({
username: event.target.value
})
}
render() {
return (
<form>

<label>Username: </label>
<input type='text' value={this.state.username}
onChange={this.handleUsernameChange}/>

</form>
)

- Now we can see that we have user name as a state property which is supplied as a value to the value attribute of the input element
- Whenever there is a change that new value is propagated to handle user name change which sets back the state property user name to the updated value
- And when the state is said the render method is called again and the new value is available to the value property and that is how we have a **Controlled Component**
- Working with form elements there are three simple steps
  1. Add the element HTML
  2. Assign the component state to the element value
  3. assign an unchanged handler that updates

Step 1

```
    render() {
        return (
            <form>
                <div>
                    <label>Username: </label>
                    <input type='text' value={this.state.username}
                           onChange={this.handleUsernameChange}/>
                </div>

                <div>
                    <label>Comments</label>
                    <textarea></textarea>
                </div>
            </form>
        )
    }
}
```

Step 2 :

- Create a new state comments which is initialized to an empty string

```
class Form extends Component {
    constructor(props) {
        super(props)

        this.state = {
            username: '',
            comments: ''
        }
    }
```

```
            <div>
                <label>Comments</label>
                <textarea value={this.state.comments}></textarea>
            </div>
        </form>
```

Step 3 :

- Assign the change handler that updates the state on change

```
handleCommentsChange = (event) => {
    this.setState({
        comments: event.target.value
    })
}

render() {
    return (
        <form>
            <div>
                <label>Username: </label>
                <input type='text' value={this.state.username}
                       onChange={this.handleUsernameChange}/>
            </div>

            <div>
                <label>Comments</label>
                <textarea value={this.state.comments}
                          onChange={this.handleCommentsChange}></textarea>
            </div>
        </form>
```

Step 1

```
        <div>
            <label>Username: </label>
            <input type='text' value={this.state.username}
                   onChange={this.handleUsernameChange}/>
        </div>

        <div>
            <label>Comments</label>
            <textarea value={this.state.comments}
                      onChange={this.handleCommentsChange}></textarea>
        </div>

        <div>
            <label>Topic</label>
            <select>
                <option value="React">React</option>
                <option value="Angular">Angular</option>
                <option value="OJET">OJET</option>
            </select>
        </div>
    </form>
```

Step 2 :

- Create a new state topic which is initialized to an ‘React’

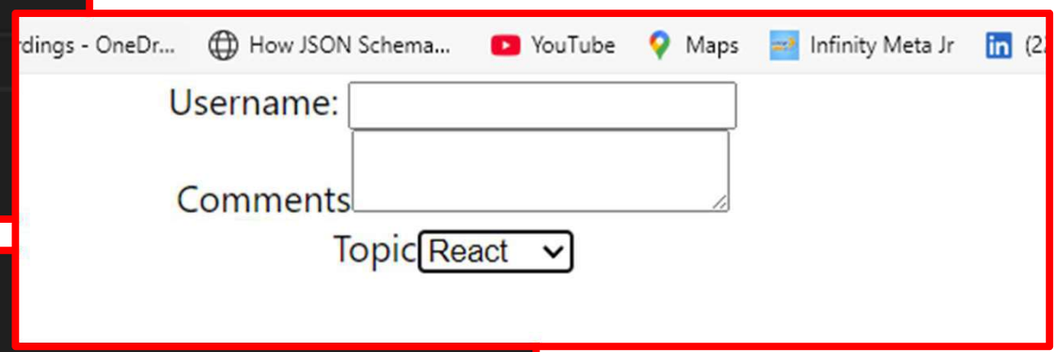
```
class Form extends Component {
    constructor(props) {
        super(props)

        this.state = {
            username: '',
            comments: '',
            topic: 'React'
        }
    }
```

```
<div>
    <label>Topic</label>
    <select value={this.state.topic}>
        <option value="React">React</option>
        <option value="Angular">Angular</option>
        <option value="OJET">OJET</option>
    </select>
</div>
```

Step 3 :

- Assign the change handler that updates the state on change

```
handleTopicChange = (event) => {
    this.setState({
        topic: event.target.value
    })
}
```



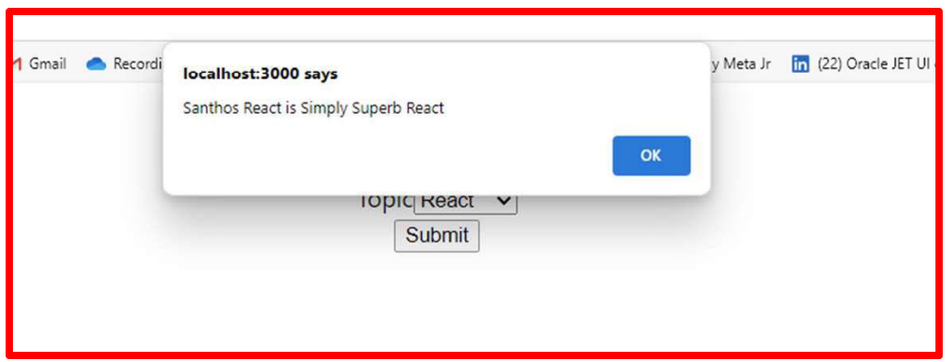

```
    <div>
        <label>Topic</label>
        <select value={this.state.topic} onChange={this.handleTopicChange}>
            <option value="React">React</option>
            <option value="Angular">Angular</option>
            <option value="OJET">OJET</option>
        </select>
    </div>
</form>
```

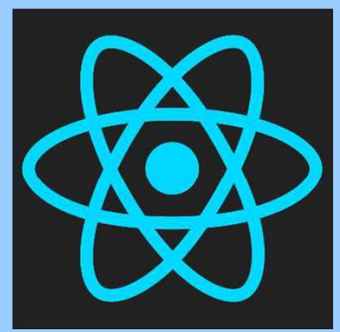
Finally Submitting Form

```
        <div>
            <button type="submit">Submit</button>
        </div>
    </form>
```

```
handleSubmit = event => {
    alert(`${this.state.username} ${this.state.comments} ${this.state.topic}`)
    event.preventDefault()
}

render() {
    return (
        <form onSubmit={this.handleSubmit}>
            <div>
```

After Submit Page Doesn't Refresh

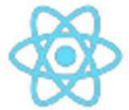




UnControlled Components

Uncontrolled Components

- Uncontrolled components are inputs that do not have a value property.
- It is the application's responsibility to keep the component state and the input value in sync.



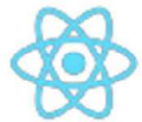
Uncontrolled components

- The form data is stored in the DOM and not within the component.
- With uncontrolled input values, there is no updating or changing of any states.
  - What you submit is what you get.
- Elements like **<input>** and **<textarea>** maintain their own state and update them when the input value changes.

```
<input type="text" name="name" ref="name" />
```

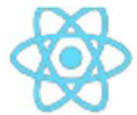
- The DOM can be queried for the value of an input field using a **ref**.
- The value needs to be pulled from the field when the form is submitted.

```
this.refs.name.value
```



Refs

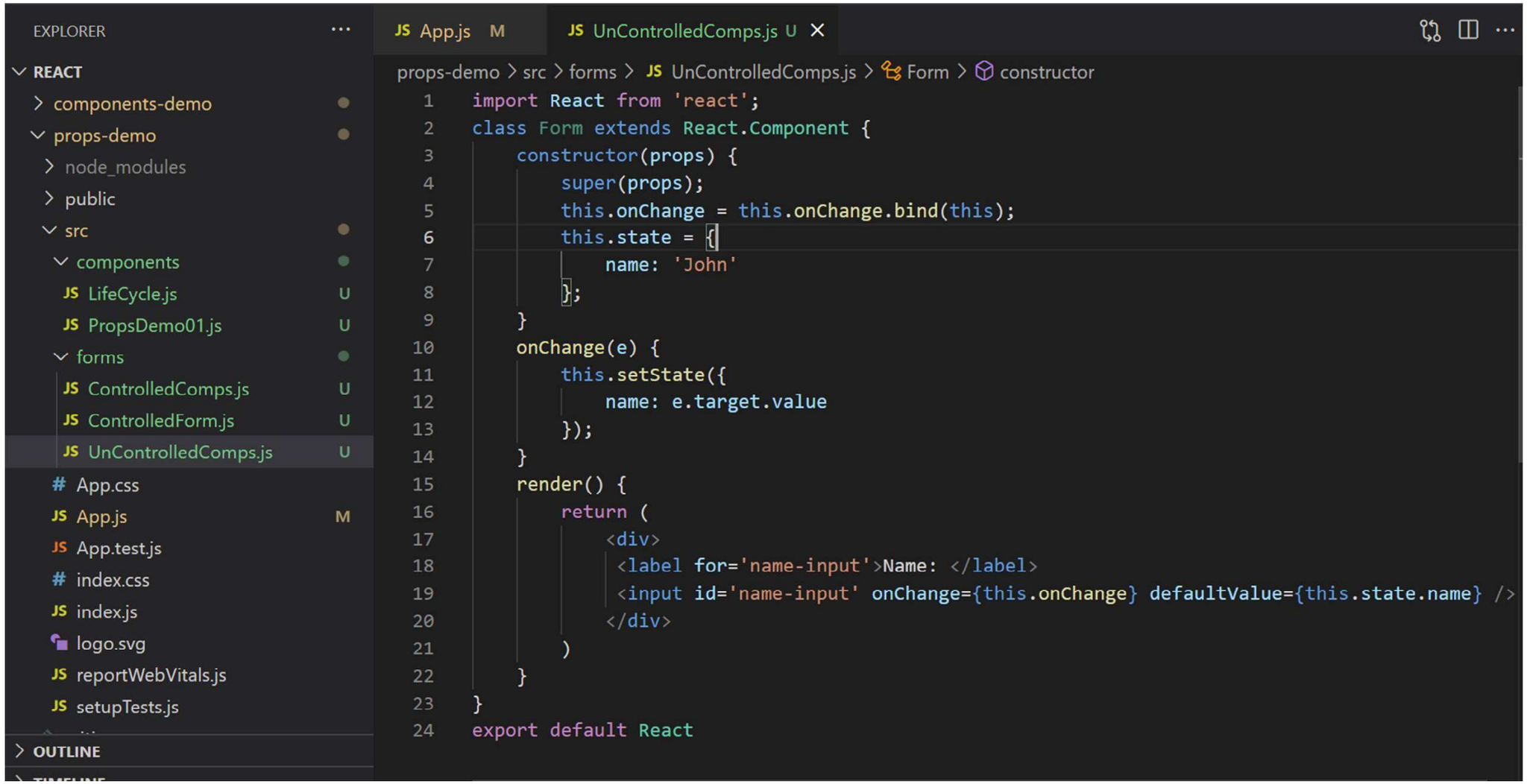
- Refs are similar to keys and are added to elements in the form of attributes.
- Props are generally the approach for parent components to interact with their children.
- Refs can be used when a child is to be modified without re-rendering it with the new props.
- ***Refs manipulate the actual DOM as opposed to the virtual DOM.***

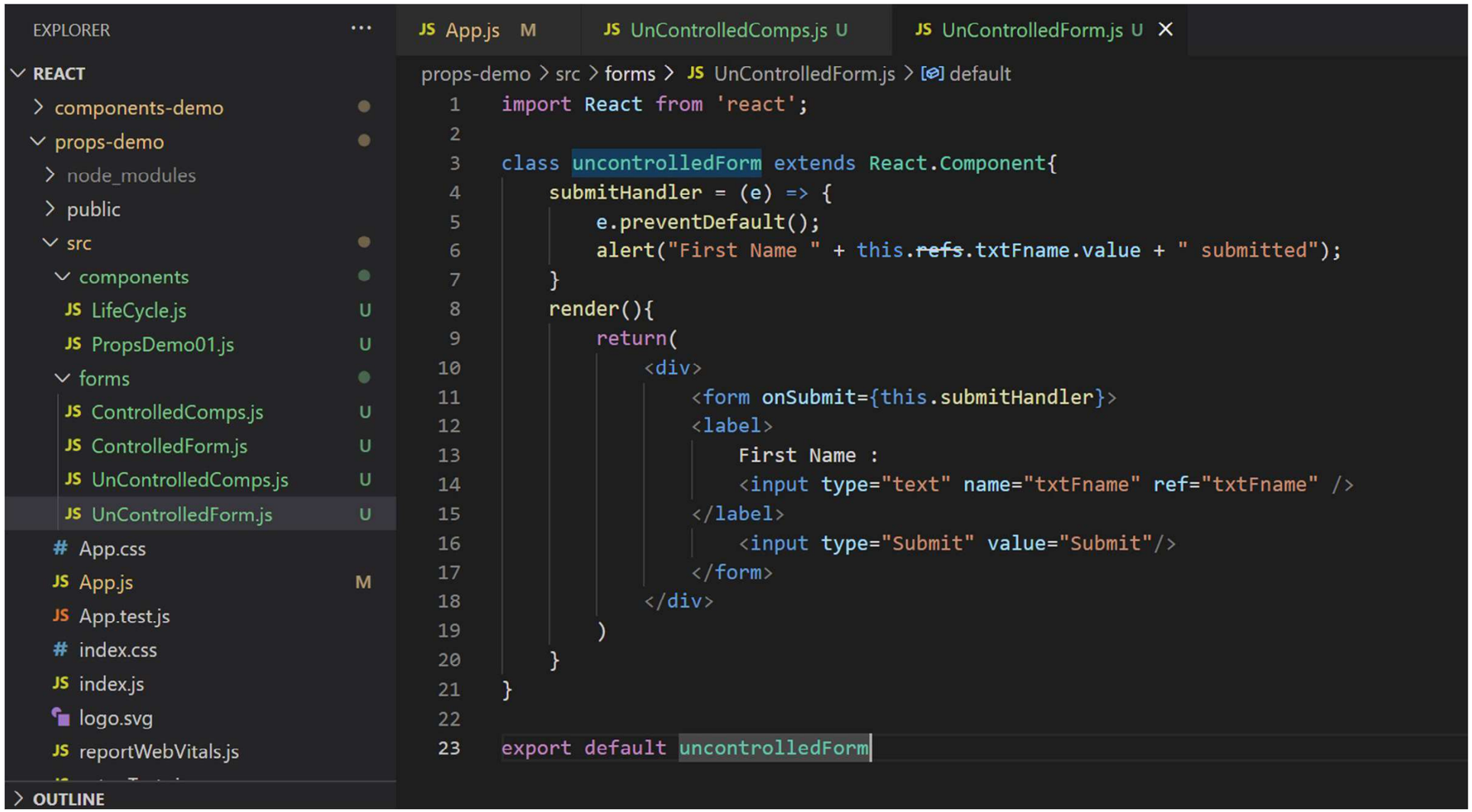


Uncontrolled components

- Default Values
  - In an uncontrolled component, React can be used to specify the initial value and leave the subsequent updates uncontrolled.
  - A `defaultValue` attribute can be used instead of the value
  - `checkbox` and `radio` button support `defaultChecked`, `select` and `textarea` support `defaultValue`

```
<input type="text" name="txtFname" ref="txtFname"
                                    defaultValue="Doe"/>
<input type="radio" name="optGender" ref="optGender"
                     value="male" defaultChecked/>
```


EXPLORER
REACT
components-demo
props-demo
node_modules
public
src
components
LifeCycle.js U
PropsDemo01.js U
forms
ControlledComps.js U
ControlledForm.js U
UnControlledComps.js U
App.css
App.js M
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
OUTLINE
App.js M
UnControlledComps.js U
props-demo > src > forms > UnControlledComps.js > Form > constructor
1 import React from 'react';
2 class Form extends React.Component {
3 constructor(props) {
4 super(props);
5 this.onChange = this.onChange.bind(this);
6 this.state = {
7 name: 'John'
8 };
9 }
10 onChange(e) {
11 this.setState({
12 name: e.target.value
13 });
14 }
15 render() {
16 return (
17
18 <label for='name-input'>Name: </label>
19 <input id='name-input' onChange={this.onChange} defaultValue={this.state.name} />
20
21)
22 }
23 }
24 export default React


EXPLORER
App.js M
UnControlledComps.js U
UnControlledForm.js U
props-demo > src > forms > UnControlledForm.js > default
REACT
components-demo
props-demo
node_modules
public
src
components
LifeCycle.js U
PropsDemo01.js U
forms
ControlledComps.js U
ControlledForm.js U
UnControlledComps.js U
UnControlledForm.js U
App.css
App.js M
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
OUTLINE
1 import React from 'react';
2
3 class uncontrolledForm extends React.Component{
4 submitHandler = (e) => {
5 e.preventDefault();
6 alert("First Name " + this.refs.txtFname.value + " submitted");
7 }
8 render(){
9 return(
10
11 <form onSubmit={this.submitHandler}>
12 <label>
13 First Name :
14 <input type="text" name="txtFname" ref="txtFname" />
15 </label>
16 <input type="Submit" value="Submit"/>
17 </form>
18
19)
20 }
21 }
22
23 export default uncontrolledForm

App.js M | UnControlledComps.js U | UnControlledForm.js U | UnControlledForm1.js U

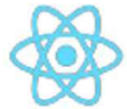
props-demo > src > forms > UnControlledForm1.js > default

```
import React from 'react';

class uncontrolledForm extends React.Component{
    submitHandler = (e) => {
        e.preventDefault();
        alert("First Name " + this.refs.txtFname.value + " submitted");
    }
    render(){
        return(
            <div>
                <form onSubmit={this.submitHandler}>
                <label>
                    First Name :
                    <input type="text" name="txtFname" ref="txtFname" defaultValue="Doe"/>
                </label>

                <p>Gender : </p>
                <label>
                    Male :
                    <input type="radio" name="optGender" ref="optGender" value="male" defaultChecked/>
                </label>
```

JS App.js M | JS UnControlledComps.js U | JS UnControlledForm.js U | JS UnControlledForm1.js U

props-demo > src > forms > JS UnControlledForm1.js > default

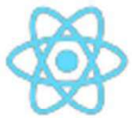
```
22                  <label>
23                      Female :
24                      <input type="radio" name="optGender" ref="optGender" value="female" />
25                  </label>
26                  <p>City :
27                  <select name="selCity" defaultValue="LON">
28                      <option value="BLR">Bemgaluru</option>
29                      <option value="NYC">New York</option>
30                      <option value="LON">London</option>
31                  </select>
32                  </p>
33                     <p> <input type="Submit" value="Submit"/></p>
34                  </form>
35              </div>
36          )
37      }
38  }
39
40  export default uncontrolledForm
```



Controlled components - validation

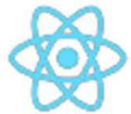
- React being a front-end library can facilitate building instant validation into a form component.
- State can also be used to help implement validation.
- Validation can be custom built or can be implemented with the help of pre-built packages such as 'validator'

```
import fieldValidator from 'validator'
fieldValidator.isEmpty('')  //true
fieldValidator.isEmail('smith@home.net')    //true
```



Uncontrolled vs Controlled values

- Uncontrolled values are useful when the form is basic with minimal features.
  - This is limited in functionality
- Controlled values are useful to facilitate validation, user feedback.
- Controlled values require more effort in terms of code.
- A form element becomes “controlled” if its value is set using state or prop.
  - Hence the state and UI are always in sync.



Summary

- Both controlled and uncontrolled form elements have their own merit.

Feature	Uncontrolled	Controlled
One-time value retrieval (submit)	Yes	Yes
Validating on submit	Yes	Yes
Instant field validation	No	Yes
Enforcing input format	No	Yes
Multiple inputs for one piece of data	No	Yes

Summary

In this lesson, you should have learned how to:

- Basics in State
- setState
- State, Events and Managed Controls