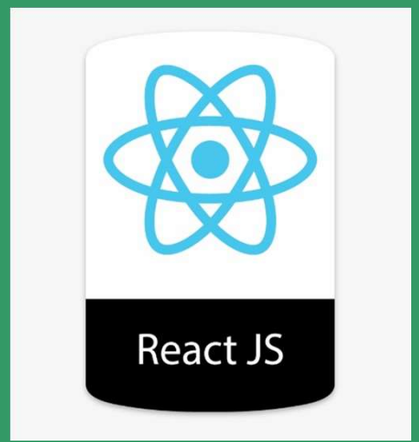
9

React JS



Styling Component

Styling Components

- No user-facing web application is complete without styles.
- There are different approaches to style a component:
  - CSS Stylesheet
    - Import the .css file and use the `className` property on elements to attach styles
  - Inline styles
    - Styles are created inline as JS objects and assigned to elements using the `style` property
  - CSS-in-JS
    - Similar to inline styles except that the style objects are in a separate JS module
  - CSS Modules

CSS Stylesheet

CSSDemo.css

```
.CSSDemo {
	margin: 40px;
	border: 5px dashed blue;
}

.CSSDemo_content {
	font-size: 15px;
	text-align: center;
}
```

CSSDemo.js

```
<div className="CSSDemo">
	<p className="CSSDemo_content">Content</p>
</div>
```

Inline styling

```
const CSSDemoStyle = {
      margin: '40px',
      border: '5px dashed blue'
};

<div style={CSSDemoStyle}>
    Content
</div>
```

CSSDemo1.js

CSS-in-JS

- This is a pattern where the CSS is composed using JavaScript instead of being defined in CSS files.
- This functionality is not a part of React but provided by third party libraries.

CSSinJS.js

```
export default {
CSSDemo_CSSInJS: {
margin: '40px',
border: '5px dashed blue'}
}
```

CSSDemo2.js

```
import CSSJS from './CSSinJS';

<div style={CSSJS.CSSDemo_CSSInJS}>
<p style={CSSJS.CSSDemo_Content_CSSInJS}>CSS in JS</p>
</div>
```

CSS Modules

- A CSS Module is a CSS file in which all class names are scoped locally by default.

CSSModule.css

```
:local(.CSSModule) {
margin: 40px;
border: 5px dashed blue;
}
```

CSSDemo3.js

```
import styles from './CSSModule.css';

<div className={styles.CSSModule}>Content</div>
```

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.

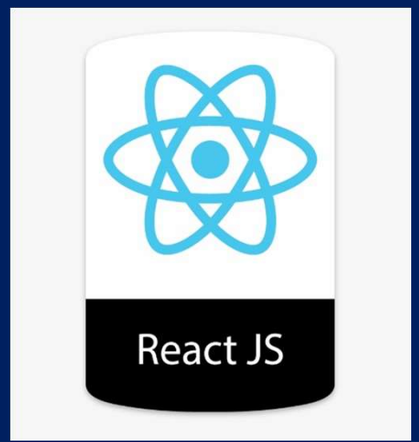
React Bootstrap Package

- 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';
```



Forms

Forms

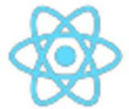
- Forms are integral to any modern application.
- They serve as a basic medium for users to interact with the application.
- Developers rely on forms for various capabilities such as securely logging in the user, building a cart, searching a product list etc.
- React does not provide comprehensive form 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.

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
```

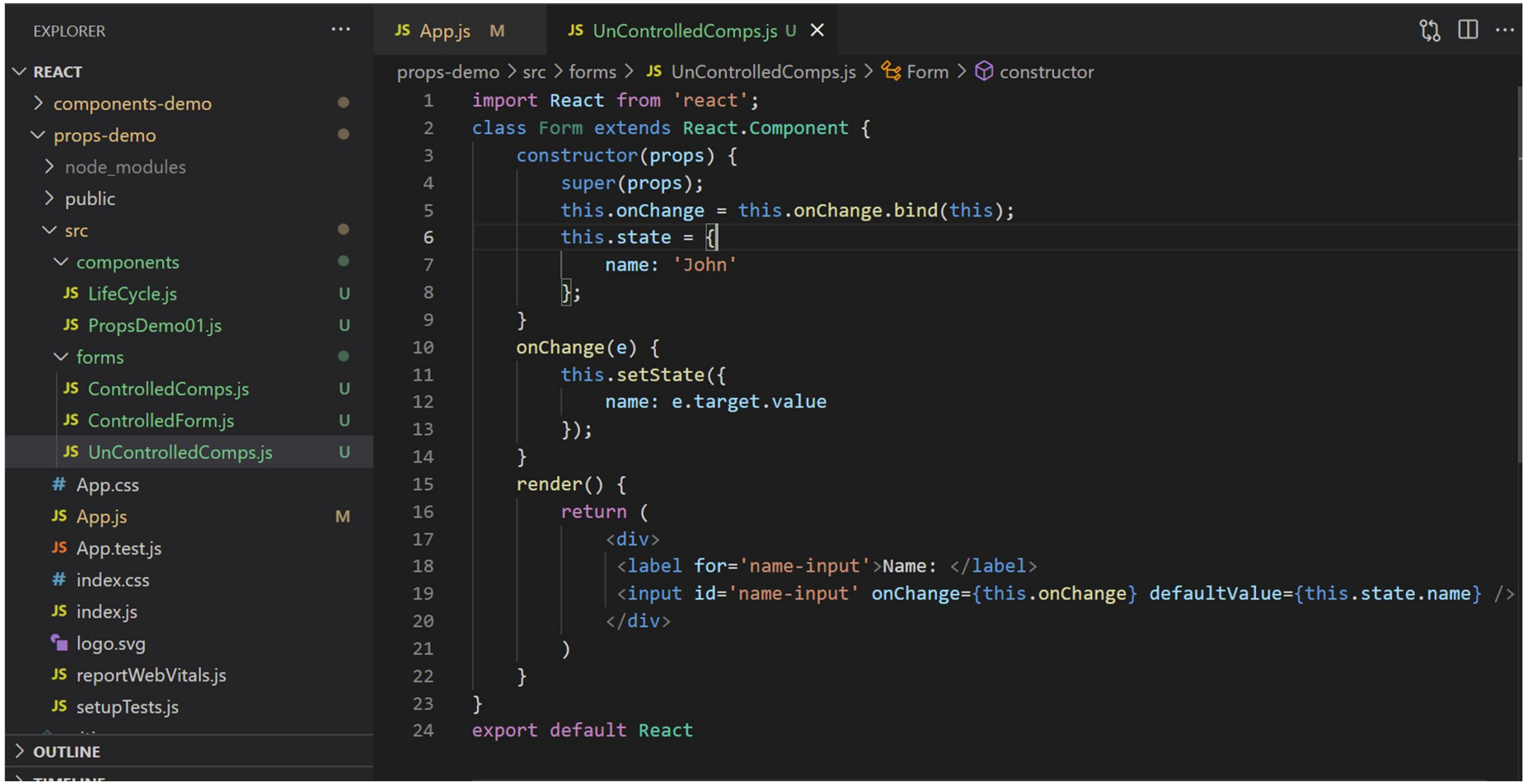
Refs

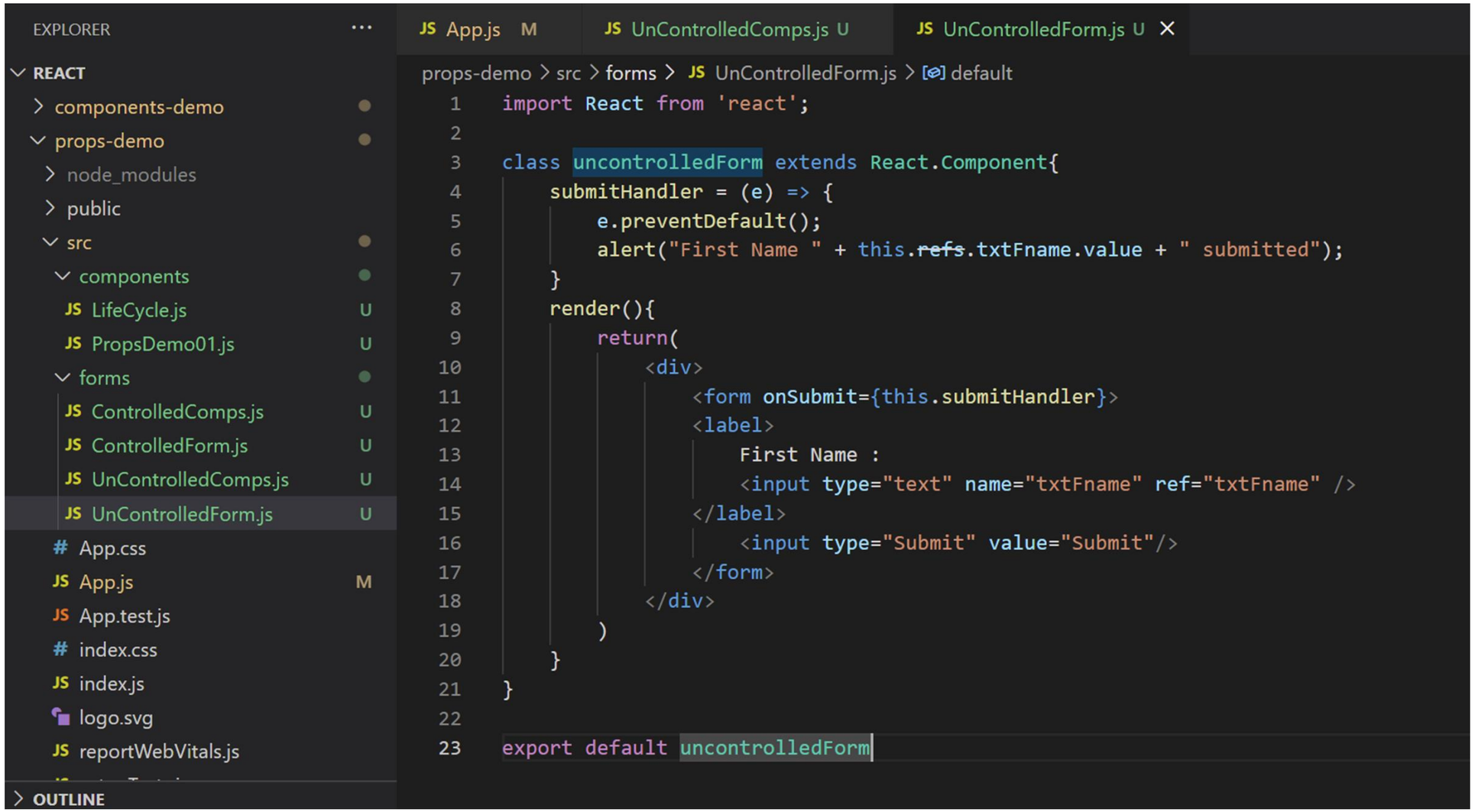
- Refs are similar to keys and are added to elements in the form of attributes.
- Props are generally the approach for parent components to interact with their children.
- Refs can be used when a child is to be modified without re-rendering it with the new props.
- ***Refs manipulate the actual DOM as 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
App.js M
UnControlledComps.js U
props-demo > src > forms > UnControlledComps.js > Form > constructor
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
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

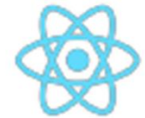
```
1   import React from 'react';
2
3   class uncontrolledForm extends React.Component{
4       submitHandler = (e) => {
5           e.preventDefault();
6           alert("First Name " + this.refs.txtFname.value + " submitted");
7       }
8       render(){
9           return(
10              <div>
11                  <form onSubmit={this.submitHandler}>
12                  <label>
13                      First Name :
14                      <input type="text" name="txtFname" ref="txtFname" defaultValue="Doe"/>
15                  </label>
16
17                  <p>Gender : </p>
18                  <label>
19                      Male :
20                      <input type="radio" name="optGender" ref="optGender" value="male" defaultChecked/>
21                  </label>
```


App.js M
UnControlledComps.js U
UnControlledForm.js U
UnControlledForm1.js U
props-demo > src > forms > UnControlledForm1.js > default
22 <label>
23 Female :
24 <input type="radio" name="optGender" ref="optGender" value="female" />
25 </label>
26 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
33 <input type="Submit" value="Submit"/>
34 </form>
35
36)
37 }
38 }
39
40 export default uncontrolledForm

Controlled Components

- Controlled form components are defined with a value property. The value of controlled inputs is managed by React, user inputs will not have any direct influence on the rendered input.
- Instead, a change to the value property needs to reflect this change.

- Form inputs should be defined as controlled components where possible.
- This ensures that the component state and the input value is in sync at all times, even if the value is changed by a trigger other than a user input.

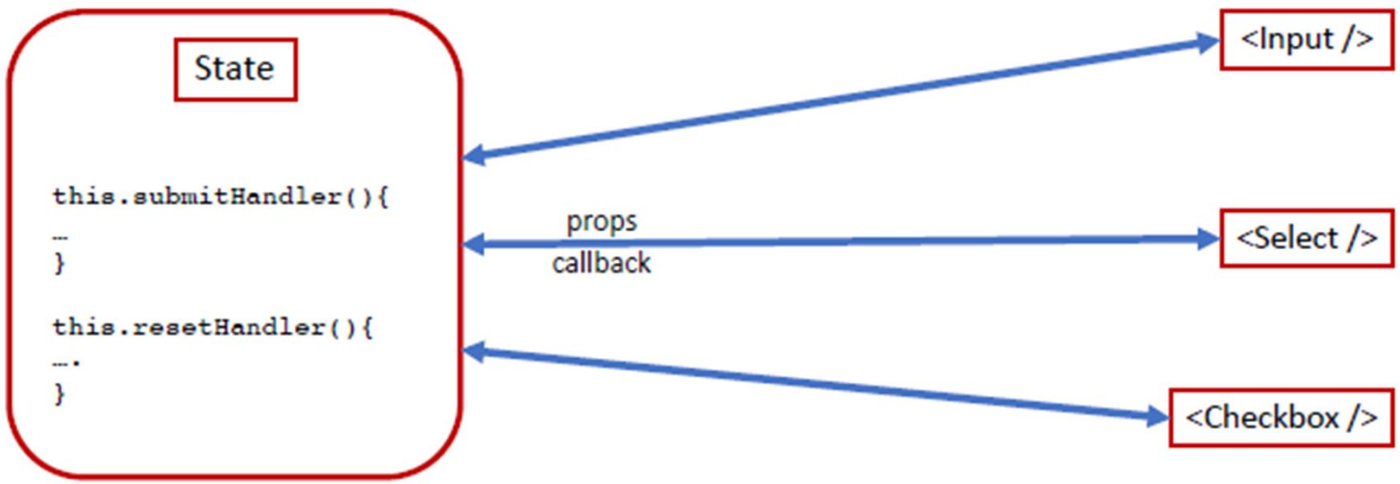


Controlled components

- The form data is handled by the component (React) rather than the DOM.
- The internal component state is always in sync with the view.
- The best practice includes:
  - Define elements in the `render()` using values from the state
  - Capture changes of a form element using `onChange()` as they happen.
  - Update the internal state in the event handler
  - New values are saved in state and the view is updated by a new `render()`

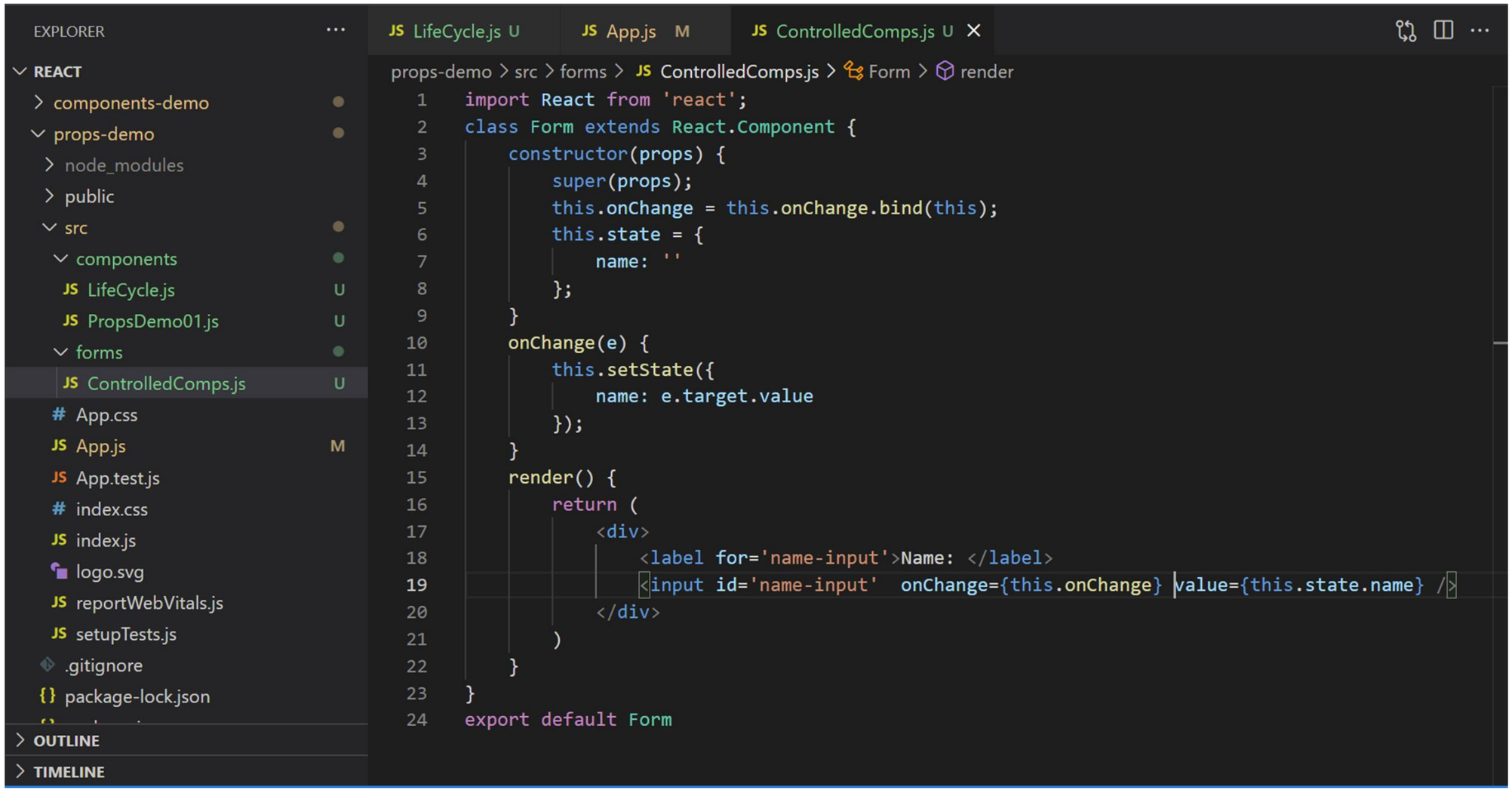
Controlled components

- Each form element gets a component of its own called dumb component(s).
- The container component maintains the state.
- This approach provides the component better control over form control elements and form data.



Controlled components

- React's composition model allows React components into smaller reusable code.
  - Each component represents an independent functional unit.
- Form container can be class based component and form elements can be functional/stateless components.
  - Container component handles state management, form submission.
- The dumb components are presentational components that contain the actual DOM markup.
  - These components receive data and callbacks as props


EXPLORER
REACT
components-demo
props-demo
node_modules
public
src
components
LifeCycle.js U
PropsDemo01.js U
forms
ControlledComps.js U
App.css
App.js M
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
OUTLINE
TIMELINE
LifeCycle.js U
App.js M
ControlledComps.js U
props-demo > src > forms > ControlledComps.js > Form > render
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: ''
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} value={this.state.name} />
20
21)
22 }
23 }
24 export default Form

JS LifeCycle.js U | JS App.js M | JS ControlledComps.js U | JS ControlledForm.js U

props-demo > src > forms > JS ControlledForm.js > controlledForm > changeHandler

```
1   import React from 'react';
2
3   class controlledForm extends React.Component{
4       state = {
5           fname : '',
6           lname : '',
7           email : ''
8       }
9       constructor(props){
10          super(props)
11      }
12
13      changeHandler = (e) => {
14          let name = e.target.name;
15          let value = e.target.value;
16          this.setState({
17              [name] : value
18          })
19      }
20
```

JS App.js M | JS ControlledComps.js U | JS ControlledForm.js U

props-demo > src > forms > JS ControlledForm.js > controlledForm

```
20
21      render(){
22          return(
23              <div>
24                  <form>
25                      <label>First Name: </label>
26                      <input type='text' name='fname' value={this.state.fname}
27                      onChange={this.changeHandler.bind(this)} /> <br/>
28                      <label>Last Name: </label>
29                      <input type='text' name='lname' value={this.state.lname}
30                      onChange={this.changeHandler.bind(this)} /> <br/>
31                      <label>Email: </label>
32                      <input type='text' name='email' value={this.state.email}
33                      onChange={this.changeHandler.bind(this)} /> <br/>
34                      <input type='Submit' value='Submit'/>
35                  </form>
36              </div>
37          )
38      }
39  }
40
41  export default controlledForm
```

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