

Unit 3 - ReactJS

Class notes by Vibha Masti

Feedback/corrections: vibha@pesu.pes.edu

0.0 Table of Contents

Unit 3 - ReactJS

- 0.0 Table of Contents
- 1.0 Introduction to MERN Stack
 - Why MERN and not MEAN?
- 1.1 Introduction to React
 - Babel and JSX
- 1.2 Getting Started with React
 - First Program
- 2.0 Introduction to Components
 - Simple Components
 - Parameterised Components (using this.props)
- 2.2 Styling Components
 - Hard-Coded Approach
 - Styling it the React Way
- 3.0 Complex Components
 - Hard-Coded
 - Better Composite Components
 - Transferring Properties Better
- 4.0 Stateful Components
 - Basic Counter Component
 - Component Life Cycle
- 5.0 Stateless Components
 - More on Stateless Components
- 6.0 Key Property
 - Hard-Coded Method
 - Using Map Method
 - Example of `map()`
 - Using key property
- 7.0 References
- 7.1 Passing References
- 8.0 Events
 - `SyntheticEvent` Object Characteristics
 - `SyntheticEvent` Object properties
- 9.0 Forms
 - Controlled Components

1.0 Introduction to MERN Stack

1. MongoDB

- Database server
- Bottom tier of MERN stack
- Application data store
- JSON docs can be stored
- Key-value pairs
- Non-SQL

2. ExpressJS

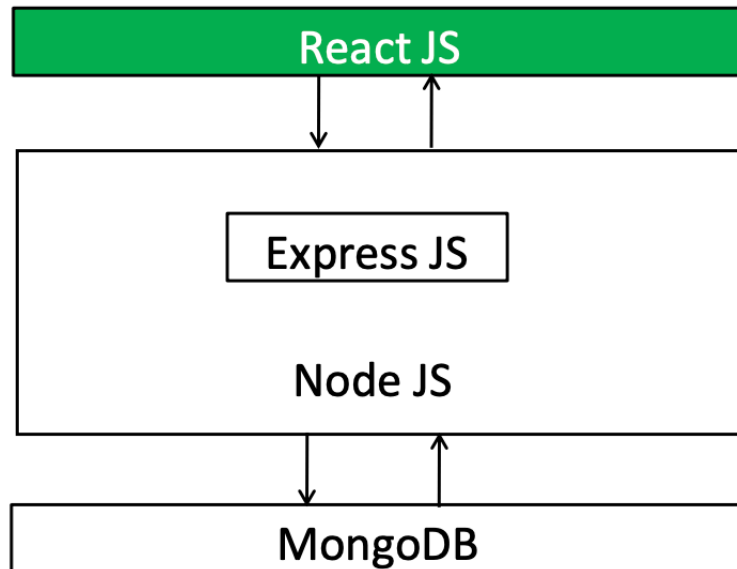
- ExpressJS and NodeJS - server side frameworks
- Middle of MERN stack
- Express: URL Routing
- Build APIs

3. ReactJS

- This unit
- Frontend JS Library (mainly SPA)
- Components
- Connect to backend server
- Render to HTML
- Stateful, data-driven interfaces
- Forms, error handling, events, lists

4. NodeJS

- Web server
- Use NodeJS MongoDB drivers
- Using callbacks and promises



Why MERN and not MEAN?

- Angular - MVC, heavy, learning curve
- React - easier, library

1.1 Introduction to React

- Released by Facebook and Instagram to help make building SPAs (single page applications) easier
- DOM manipulation is faster (using virtual DOM)
- Uses components for visual elements

Babel and JSX

- Babel is a JSX to JavaScript converter that allows us to create React elements in XML/HTML syntax
- JSX gets converted to `React.createElement()`

JSX

```

1 ReactDOM.render(
2   <div>
3     <h1>Captain America</h1>
4     <h1>Iron Man</h1>
5     <h1>Thor</h1>
6   </div>,
7   destination
8 );

```

JavaScript

```

1 ReactDOM.render(
2   React.createElement("div", null,
3     React.createElement( "h1", null, "Captain America" ),
4     React.createElement ( "h1", null, "Iron Man" ),
5     React.createElement ( "h1", null, "Thor" ),
6   ),
7   destination
8 );

```

1.2 Getting Started with React

First Program

Include the following libraries in the `<script></script>` tags

```

1 <!-- React JS libraries -->
2
3 <script src="https://unpkg.com/react@16/umd/react.development.js"
4   crossorigin></script>
5 <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
6   crossorigin></script>
7
8 <!-- Babel -->
9 <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

```

1. Tip: if you are using VSCode, type in `html:5` and press enter to get a basic HTML5 skeleton

2. Be sure to include the sources in the header `<script></script>` tags
3. Create an empty `<div></div>` tag to manipulate using `ReactDOM`
4. Add a script tag with text type `text/babel` and start writing your react code!

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Introduction to React</title>
7
8   <script src="https://unpkg.com/react@16/umd/react.development.js"
crossorigin></script>
9   <script src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js" crossorigin></script>
10  <script src="https://unpkg.com/@babel/standalone/babel.min.js">
</script>
11
12 </head>
13 <body>
14   <div id="container"></div>
15   <script type="text/babel">
16     var des = document.querySelector("#container");
17     ReactDOM.render(
18       {/* JSX code */}
19       <h1> React world </h1>,
20       des
21     );
22   </script>
23 </body>
24 </html>
```

3. Note: the JSX code can be written as

```
1 | React.createElement("h1", {color: red}, "React world!!"),
```

2.0 Introduction to Components

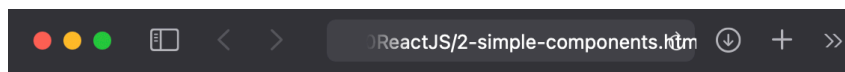
Simple Components

- Components are written as classes that inherit from `React.Component`
- A render method must be written that returns JSX
- Call the `ReactDOM.render()` method with an element of the newly-created component

Inside the body `<script type="text/babel"></script>` tags

```
1 var des = document.querySelector("#container");
2
3 class HelloWorld extends React.Component {
4   render() {
5     return <h1> Hello components! </h1>
6   }
7 }
8
9 ReactDOM.render(
10   <HelloWorld />,
11   des
12 );
```

Rendered in a browser



Hello components!

Parameterised Components (using `this.props`)

- JSX code within `{}` is JS
- `this.props` refers to the properties passed in the JSX from the `ReactDOM.render()` function

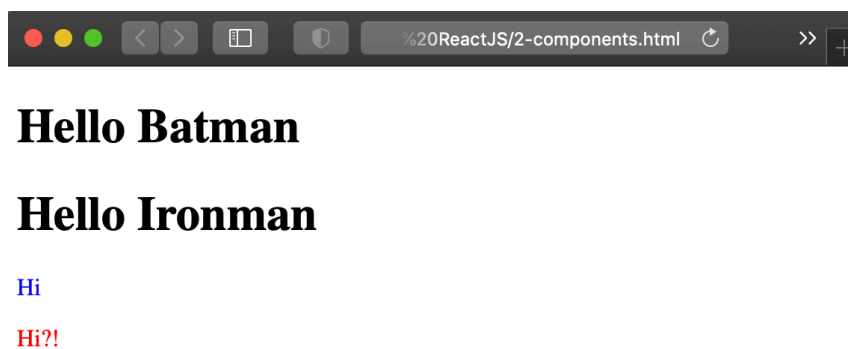
Inside the body `<script type="text/babel"></script>` tags

```
1  var des = document.querySelector("#container");
2
3  class HelloWorld extends React.Component {
4    render() {
5      /* This is a JSX comment */
6      /* {this.props.greet} is displayed in the <h1></h1>
7      tags on the browser*/
8      return <h1> Hello {this.props.greet} </h1>
9    }
10 }
11
12 class Hello extends React.Component {
13   render() {
14     return (
15       /* A component can have child elements like in HTML */
16       <div className={this.props.type}>
17         /* this.props.children contains the text within the tags,
18         not an array of children */
19         <p>{this.props.children}</p>
20       </div>
21     );
22   }
23 }
24
25 ReactDOM.render(
26   <div>
27     /* property greet is passed */
28     <HelloWorld greet="Batman"/>
29     <HelloWorld greet="Ironman"/>
30
31     <Hello type="greeting">
32       Hi
33     </Hello>
34     <Hello type="shocking">
35       Hi?!
36     </Hello>
37   </div>,
38   des
39 );
```

Inside `<style></style>` tags in the head of the page, we can add some basic styling

```
1 <style>
2   .greeting {
3     color: blue
4   }
5
6   .shocking {
7     color: red
8   }
9 </style>
```

Rendered in a browser



2.2 Styling Components

Hard-Coded Approach

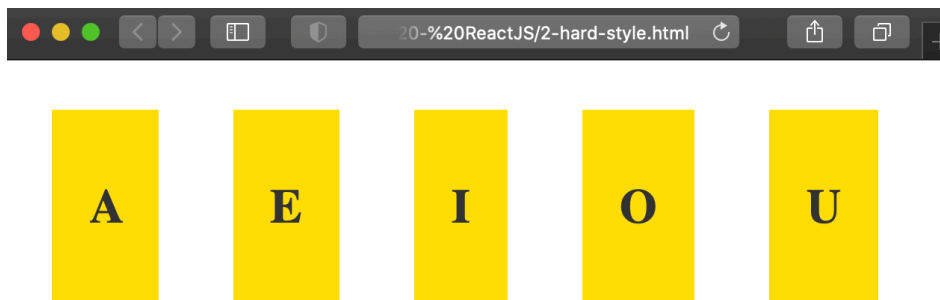
- To style various components, we can add CSS style to the `<style></style>` tags in the head
- This is not the best way to do it

```
1 .letter {
2   background-color: #ffde00;
3   color: #333;
4   display: inline-block;
5   padding: 25px;
6   margin: 25px;
7 }
```


Inside the body `<script type="text/babel"></script>` tags

```
1 var des = document.querySelector("#container");
2
3 class Letter extends React.Component {
4   render() {
5     return <div className="letter">
6       <h1>{this.props.children}</h1>
7     </div>;
8   }
9 }
10
11 ReactDOM.render(
12   <div>
13     <Letter> A </Letter>
14     <Letter> E </Letter>
15     <Letter> I </Letter>
16     <Letter> O </Letter>
17     <Letter> U </Letter>
18   </div>,
19   des
20 );
```

Rendered in a browser



Styling it the React Way

- Create a style object in the `Letter` component class
- Syntax similar to CSS but uses camelCase
- Reads backgroundColor property from `<Letter></Letter>` tags
- No `<style></style>` tags in the head

Inside the body `<script type="text/babel"></script>` tags

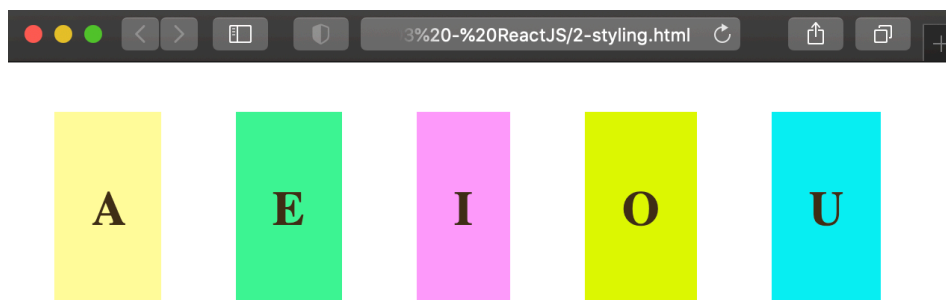
```
1 var des = document.querySelector("#container");
```

```

2
3 class Letter extends React.Component {
4   render() {
5     var letterStyle = {
6       backgroundColor: this.props.backgroundColor,
7       color: '#432e19',
8       display: 'inline-block',
9       padding: '25px',
10      margin: '25px'
11    };
12    return <div style={letterStyle}>
13      <h1>{this.props.children}</h1>
14    </div>;
15  }
16 }
17
18 ReactDOM.render(
19   <div>
20     <Letter backgroundColor='#ffff99'> A </Letter>
21     <Letter backgroundColor='#45f594'> E </Letter>
22     <Letter backgroundColor='#ff99ff'> I </Letter>
23     <Letter backgroundColor='#dbfb10'> O </Letter>
24     <Letter backgroundColor='#00f2f4'> U </Letter>
25   </div>,
26   des
27 );

```

Rendered in a browser



3.0 Complex Components

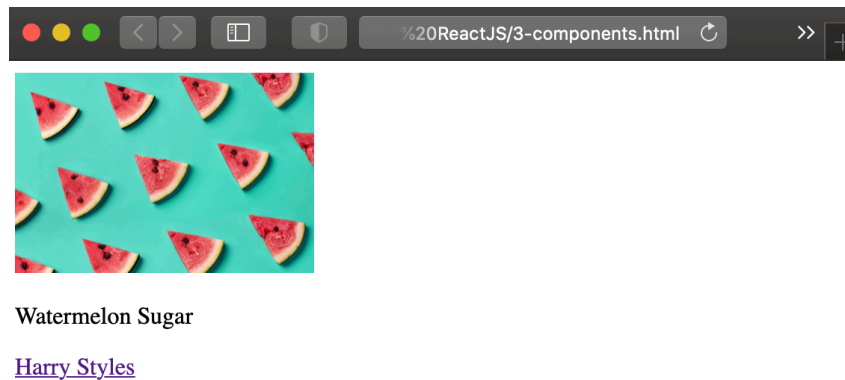
- Good for reusability and composability
- Components that contain components
- For example, a component called `SearchResult` that contains the components `ResultImage`, `ResultCaption` and `ResultLink`

Hard-Coded

Inside the body `<script type="text/babel"></script>` tags

```
1 class ResultImage extends React.Component {
2   render() {
3     return (
4       </img>
5     );
6   }
7 }
8
9 class ResultCaption extends React.Component {
10  render() {
11    return (
12      <p>Watermelon Sugar</p>
13    );
14  }
15 }
16
17 class ResultLink extends React.Component {
18  render() {
19    return (
20      <a href="https://www.youtube.com/watch?v=E07s5ZYygMg">Harry
21      styles</a>
22    );
23  }
24 }
25
26 class SearchResult extends React.Component {
27  render() {
28    return (
29      <div>
30        <ResultImage/>
31        <ResultCaption/>
32        <ResultLink/>
33      </div>
34    );
35  }
36 }
37 ReactDOM.render(
38   <SearchResult/>,
39   document.querySelector("#container")
40 );
```

Rendered in a browser



Better Composite Components

- To remove the hardcoding, the properties like `src`, `href` etc., have to be passed on using `this.props` from the parent Components to its sub components, as follows
- If desired, styling may be added as previously discussed

Inside the body `<script type="text/babel"></script>` tags

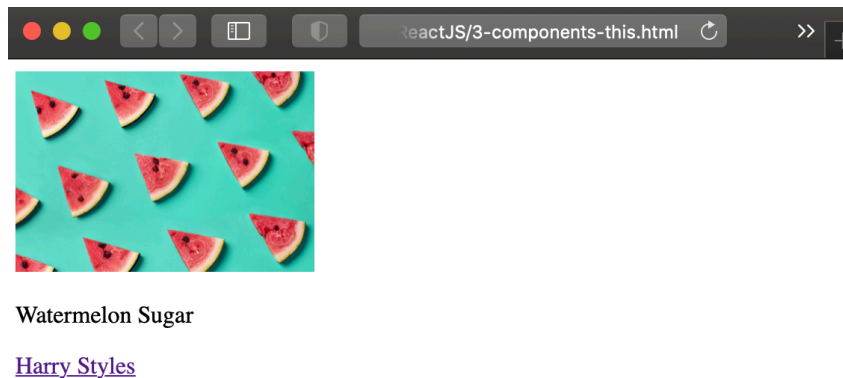
```
1 class ResultImage extends React.Component {
2   render() {
3     return (
4       <img src={this.props.src} width={this.props.width}></img>
5     );
6   }
7 }
8
9 class ResultCaption extends React.Component {
10  render() {
11    return (
12      <p>{this.props.children}</p>
13    );
14  }
15 }
16
17 class ResultLink extends React.Component {
18  render() {
19    return (
20      <a href={this.props.href}>Harry Styles</a>
21    );
22  }
23 }
```

```

24
25 class SearchResult extends React.Component {
26   render() {
27     return (
28       <div>
29         {/* Tedious way to transfer properties */}
30         <ResultImage src={this.props.src} width=
{this.props.width}/>
31         <ResultCaption>{this.props.children}</ResultCaption>
32         <ResultLink href={this.props.href}/>
33       </div>
34     );
35   }
36 }
37
38
39 ReactDOM.render(
40   <SearchResult src="images/watermelon.jpg" width="200px"
href="https://www.youtube.com/watch?v=E07s5ZYyMg">Watermelon
Sugar</SearchResult>,
41   document.querySelector("#container")
42 );

```

Rendered in a browser



Transferring Properties Better

- To simplify the tedious process of transferring components, we use the spread operator -



Inside the body `<script type="text/babel"></script>` tags

```
1  class ResultImage extends React.Component {
2    render() {
3      return (
4        <img src={this.props.src} width={this.props.width}></img>
5      );
6    }
7  }
8
9  class ResultCaption extends React.Component {
10   render() {
11     return (
12       <p>{this.props.caption}</p>
13     );
14   }
15 }
16
17 class ResultLink extends React.Component {
18   render() {
19     return (
20       <a href={this.props.href}>{this.props.children}</a>
21     );
22   }
23 }
24
25 class SearchResult extends React.Component {
26   render() {
27     return (
28       <div className="result">
29         <ResultImage {...this.props}/>
30         <ResultCaption {...this.props}/>
31         <ResultLink {...this.props}/>
32       </div>
33     );
34   }
35 }
36
37 ReactDOM.render(
38   <SearchResult src="images/watermelon.jpg"
39     width="200px" caption="watermelon sugar"
40     href="https://www.youtube.com/watch?v=E07s5ZYyMg">
41     Harry Styles
42   </SearchResult>,
43
44   document.querySelector("#container")
45 );
```

Rendered in a browser



4.0 Stateful Components

- Up until now, we have only dealt with static/stateless components that do not undergo any state changes
- Components may need to change based on user actions, timers, responses from servers etc.

Basic Counter Component

- Shows number of seconds that the user has been on the page for
- We create a `Component` `Counter` with a `constructor()`, `componentDidUpdate()` method, `componentDidMount()` method, `timer()` method and a `render()` method
- The `constructor(props, context)` is to initialise the counter and `this.state` and should always call the `super(props, context)` constructor
- The method `componentDidMount()` can be used to start the counter and set the timer
- The `timer()` method calls the `setState()` function, which is what makes the component stateful
-

Inside the body `<script type="text/babel"></script>` tags

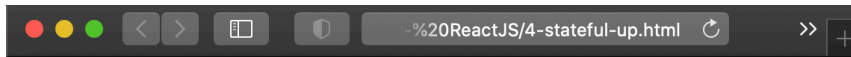
```
1 class Counter extends React.Component {
2   constructor (props, context) {
3     super(props, context);
4     this.state = {
5       seconds: 0
6     };
7   }
8 }
```

```

7     this.timer = this.timer.bind(this);
8   }
9
10  componentDidMount() {
11    console.log("Updating!!");
12  }
13
14  componentDidMount() {
15    this.t = setInterval(this.timer, 1000);
16  }
17
18  componentWillUnmount() {
19    clearInterval(this.t);
20    console.log("Stopped!!!");
21  }
22
23  timer () {
24    this.setState((prevState) => {
25      return {
26        seconds: prevState.seconds + 1
27      }
28    })
29  }
30 }
31
32 render() {
33   return (
34     <h1>{this.state.seconds}</h1>
35   )
36 }
37 }
38
39 class CounterDisplay extends React.Component {
40   render() {
41     return (
42       <div>
43         <Counter/>
44         <h2> seconds </h2>
45         <h2> since the page loaded </h2>
46       </div>
47     )
48   }
49 }
50
51
52 ReactDOM.render(
53   <CounterDisplay/>,
54   document.querySelector("#container")
55 );

```

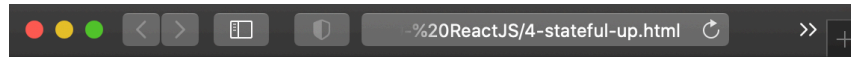

Rendered in a browser



8

seconds

since the page loaded

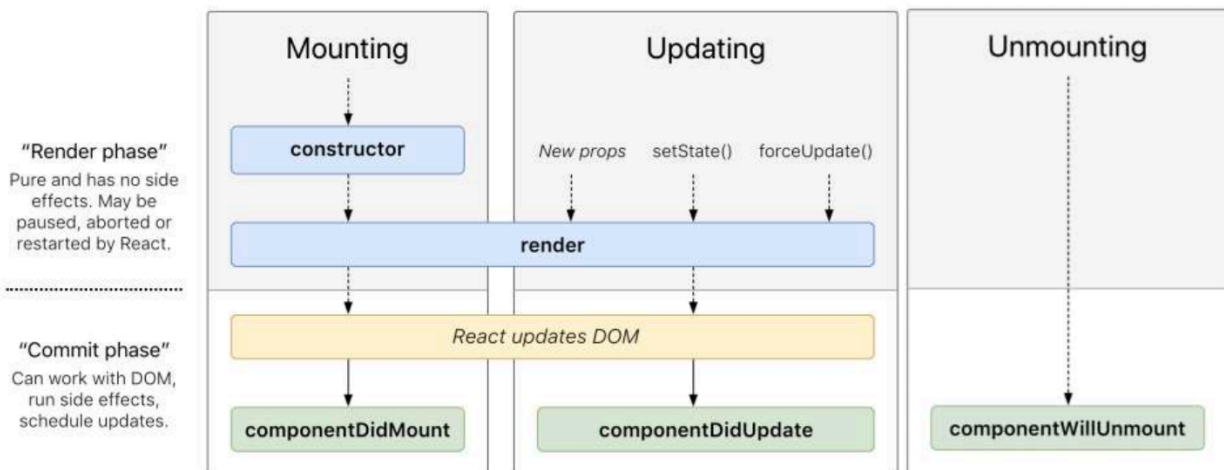


25

seconds

since the page loaded

Component Life Cycle



5.0 Stateless Components

- Class-based components are quite heavy
- For some stateless components, function-based components can be used
- Properties can be passed as parameters, not using `this`

Inside the body `<script type="text/babel"></script>` tags

```
1 function Stuff(props) {
2   return (
3     <div>
4       <h1>Hello {props.name}</h1>
5     </div>
6   );
7 }
8
9 ReactDOM.render(
10  <Stuff name="Thor"/>,
11  document.querySelector('#container')
12 );
```

Rendered in a browser



Hello Thor

More on Stateless Components

Inside the body `<script type="text/babel"></script>` tags

```
1 var des = document.querySelector("#container");
2
3 function HelloWorld(props) {
4   return <h1> Hello {props.greet} </h1>
5 }
6
7 function Hello(props) {
8   return (
9     <div className={props.type}>
10      <p>{props.children}</p>
11    </div>
12  );
13 }
14
```

```

15
16 ReactDOM.render(
17   <div>
18     <Hello world greet="Batman"/>
19     <Hello world greet="Ironman"/>
20     <Hello type="greeting">
21       Hi
22     </Hello>
23     <Hello type="shocking">
24       Hi?!
25     </Hello>
26   </div>,
27   des
28 );

```

Rendered in a browser



Hello Batman

Hello Ironman

Hi

Hi?!

6.0 Key Property

- When returning an array or list of elements, the individual elements should be uniquely identified by a key property
- Helps React identify each element in the list
- Unique key properties for each child in an array or iterator

Hard-Coded Method

Inside the body `<script type="text/babel"></script>` tags

```

1 function Stuff() {
2   return (
3     [
4       <p key="1">Batman</p>,
5       <p key="2">Superman</p>,
6       <p key="3">Joker</p>,
7     ]
8   )
9 }

```

Rendered in a browser



Using Map Method

- Map method on an array or list calls a callback function for each element of the array

Example of `map()`

```

1 const numbers = [1, 2, 3, 4, 5];
2 const doubled = numbers.map((number) => number * 2);
3 console.log(doubled);

```

Output

```

[2, 4, 6, 8, 10] (5)

```

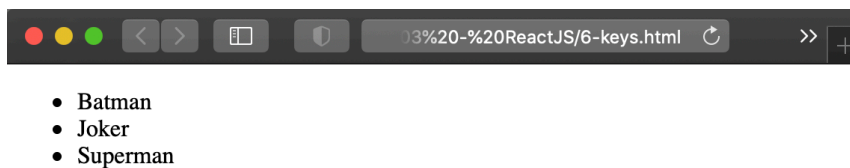
Using key property

- Return a `` element for each item
- If `listItems` does not have a `key` property, error

Inside the body `<script type="text/babel"></script>` tags

```
1 function NameList(props) {
2   const names = props.names;
3   const listItems = names.map((name, index) => <li key={index}>{name}
  </li>);
4
5   return (
6     <ul>{listItems}</ul>
7   )
8 }
9
10 const names = ["Batman", "Joker", "Superman"];
11
12 ReactDOM.render(
13   <NameList names={names}/>,
14   document.querySelector('#container')
15 );
```

Rendered in a browser



7.0 References

- Instead of accessing DOM elements in JSX, can use `refs`
- Can be used to induce changes in Components or Elements after they are rendered
- `ref`s are callback functions passed as properties

Inside the body `<script type="text/babel"></script>` tags

```
1 var colors = ['yellow', 'red', 'green', 'blue', 'orange']
2 class CustomText extends React.Component {
3   constructor(props, context) {
4     super(props, context)
5     this.myText = null;
6     this.setTextRef = element => {
```

```

7      /* this.myText is a reference to the element
8      and can be used to perform raw DOM operations */
9      this.myText = element;
10     }
11     this.changeText = event => {
12         this.myText.innerHTML = "Changed";
13         var i = Math.floor(Math.random()*5);
14         this.myText.innerHTML = colors[i];
15         this.myText.style.color = colors[i];
16     }
17 }
18
19
20 render() {
21     return (
22         <div>
23             <h1 ref={this.setTextRef}>This is my text</h1>
24             <input type="button" value="Change the text" onClick=
{this.changeText}/>
25
26         </div>
27     )
28 }
29 }
30
31 ReactDOM.render(
32     <CustomText/>,
33     document.querySelector('#container')
34 );

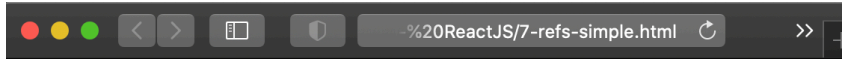
```

Rendered in a browser



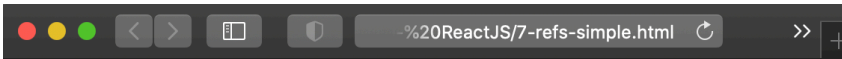
This is my text

Change the text



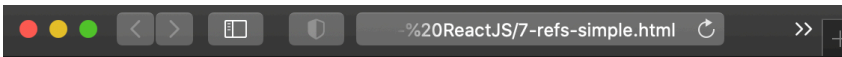
red

Change the text



green

Change the text



orange

Change the text



blue

Change the text



yellow

Change the text

7.1 Passing References

- Parents can pass a `ref` callback to its child element to get a reference to the child element

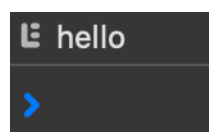
Inside the body `<script type="text/babel"></script>` tags

```
1 function CustomInput (props) {
2   return (
3     <div>
4       <input ref={props.inputRef} type="text"/>
5     </div>
6   )
7 }
8
9 class Parent extends React.Component {
10  constructor(props, context) {
11    super(props, context);
12
13    this.handleClick = event => {
14      this.inputText.focus();
15      console.log(this.inputText.value);
16    }
17  }
18
19  render() {
20    return (
21      <div>
22        <CustomInput inputRef={e1 => {this.inputText = e1}}/>
23        <button onClick={this.handleClick}>Click</button>
24      </div>
25    )
26  }
27 }
28
29 ReactDOM.render(
30   <Parent/>,
31   document.querySelector('#container')
32 );
```


Rendered in a browser



Console



8.0 Events

- Similar to DOM events, but with some syntax differences
- React events use camelCase and not lowercase (as in DOM events)
- With JSX, event handler is a function and not a string
- Event objects are of type `SyntheticEvent` object

`SyntheticEvent` Object Characteristics

- `SyntheticEvent` is a wrapper around the `DOMEvent` object
- Event handlers are registered at the time of rendering, rather than using `addEventListener` after the element has been created
- Returning false does not prevent the default browser behaviour
- `e.preventDefault()` or `e.stopPropagation()` should be used

`SyntheticEvent` Object properties

- `boolean bubbles`
- `boolean cancelable`
- `DOMEventTarget currentTarget`
- `boolean defaultPrevented`
- `number eventPhase`

- booleanisTrusted
- DOMEventnativeEvent
- voidpreventDefault()
- booleanisDefaultPrevented()
- voidstopPropagation()
- booleanisPropagationStopped()
- voidpersist()
- DOMEventTargettarget
- numbertimeStamp
- stringtype

Inside the body `<script type="text/babel"></script>` tags

```

1  class MyDiv extends React.Component {
2      constructor(props) {
3          super(props);
4          this.myText = null;
5          this.showChar = event => {
6              var txt;
7              this.myText.innerHTML = event.target.value + ' ';
8
9              if (event.shiftkey) {
10                 txt = '<span style="color:red">'+event.key+'</span>';
11             }
12             else {
13                 txt = event.key;
14             }
15             this.myText.innerHTML += txt;
16         }
17         this.setTextRef = element => {
18             this.myText = element;
19         }
20     }
21     render() {
22         return (
23             <div>
24
25                 <input onKeyPress={this.showChar} type='text' />
26                 <h1 ref={this.setTextRef}/>
27             </div>
28         );
29     }
30 }
31
32 ReactDOM.render(
33     <MyDiv />,

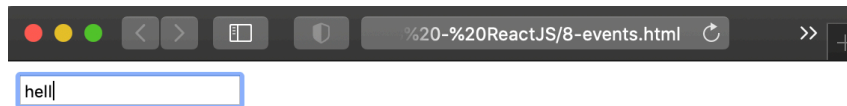
```

```
34 | document.querySelector('#container')
35 | );
```

Rendered in a browser



h



hel l



hello

9.0 Forms

- Two main functionalities: when input value is change (`onChange` event) and when the form is submitted (`onSubmit` event)
- Form Data in React is usually handled by Components by storing them in `state` object, such Form components are called **Controlled Components**

Controlled Components

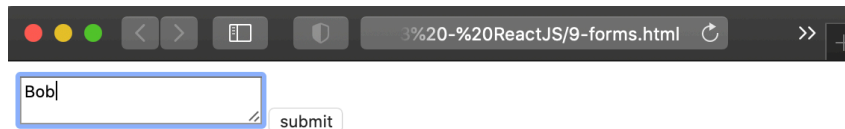
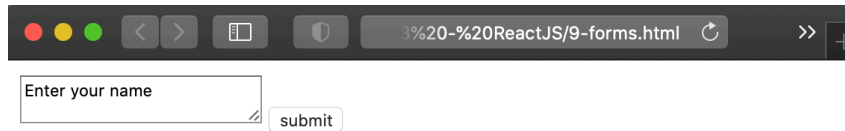
- The value property of the three types of form elements `<input>`, `<textarea>` and `<select>` are controlled by React using the `state` and updated only using `setState`
- The value is updated in the state when `onChange` event is triggered on the form element (which calls `setState`)
- The value is also set to the state property to keep it updated at all times (single source of truth)

Inside the body `<script type="text/babel"></script>` tags

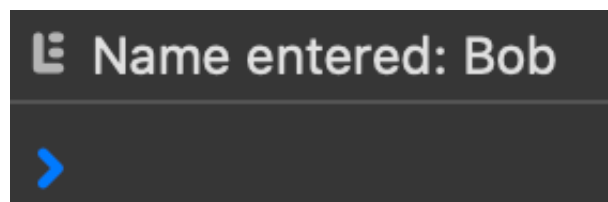
```
1  var txt, rv;
2
3  class ReadName extends React.Component {
4    constructor(props) {
5      super(props);
6      this.state = {
7        value: 'Enter your name'
8      };
9
10     /* Instead of defining
11     functions in constructor */
12     this.handleChange = this.handleChange.bind(this);
13     this.handleSubmit = this.handleSubmit.bind(this);
14   }
15
16   handleChange = event => {
17     this.setState({
18       value: event.target.value
19     });
20   }
21
22   handleSubmit = function (event) {
23     console.log("Name entered: " + this.state.value);
24     event.preventDefault();
25   }
26
27   render() {
28     return (
29       <form onSubmit={this.handleSubmit}>
30         <textarea value={this.state.value} onChange=
31 {this.handleChange}></textarea>
32         <input type="submit" value="submit"/>
33       </form>
34     )
35   }
36 }
```

```
35 }
36
37 ReactDOM.render(
38   <ReadName/>,
39   document.querySelector('#container')
40 );
```

Rendered in a browser



Console



More on Forms

Inside the body `<script type="text/babel"></script>` tags

```
1 class BMICalc extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       height: 'in cms',
6       weight: 'in kgs',
7       bmi: '0',
```

```

8         bmistat: '0'
9     };
10
11     /* Instead of defining
12     functions in constructor */
13     this.handleChangeHeight = this.handleChangeHeight.bind(this);
14     this.handleChangeWeight = this.handleChangeWeight.bind(this);
15     this.handleSubmit = this.handleSubmit.bind(this);
16 }
17
18 handleChangeHeight = event => {
19     var height_mtrs = event.target.value/100;
20     var bmi = this.state.weight/(height_mtrs*height_mtrs);
21     var bmistat = null;
22     if (bmi < 19) {
23         bmistat = 'Underweight'
24     }
25     else if (bmi < 26) {
26         bmistat = 'Normal'
27     }
28     else {
29         bmistat = 'Overweight'
30     }
31     this.setState({
32         height: event.target.value,
33         weight: this.state.weight,
34         bmi: bmi,
35         bmistat: bmistat
36     });
37 }
38
39 handleChangeWeight = event => {
40     var height_mtrs = this.state.height/100;
41     var bmi = event.target.value/(height_mtrs*height_mtrs);
42     var bmistat = null;
43     if (bmi < 19) {
44         bmistat = 'Underweight'
45     }
46     else if (bmi < 26) {
47         bmistat = 'Normal'
48     }
49     else {
50         bmistat = 'Overweight'
51     }
52     this.setState({
53         height: this.state.height,
54         weight: event.target.value,
55         bmi: bmi,
56         bmistat: bmistat

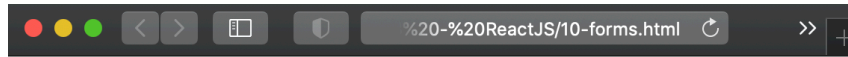
```

```

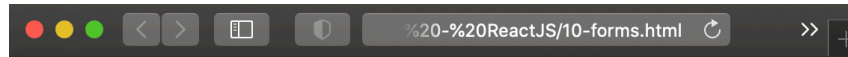
57     });
58   }
59
60   handleSubmit = function (event) {
61     console.log("Height entered: " + this.state.height);
62     console.log("Weight entered: " + this.state.weight);
63     console.log("BMI: " + this.state.bmi);
64     console.log("BMI status: " + this.state.bmistat);
65     event.preventDefault();
66   }
67
68   render() {
69     return (
70       <form onSubmit={this.handleSubmit}>
71         <label>
72           Height:
73         </label>
74         <input value={this.state.height}
75           onChange={this.handleChangeHeight} type="text">
76       </input>
77
78         <label>
79           weight:</label>
80         <input value={this.state.weight}
81           onChange={this.handleChangeweight} type="text">
82       </input>
83
84         <input type="submit" value="submit"/>
85       </form>
86     )
87   }
88 }
89
90 ReactDOM.render(
91   <BMICalc/>,
92   document.querySelector('#container')
93 );

```

Rendered in a browser

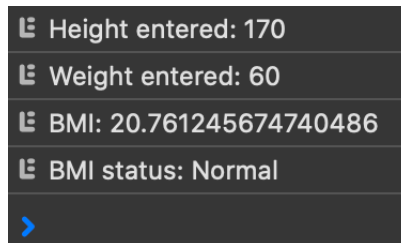


Height: Weight:



Height: Weight:

Console



Uncontrolled Components

- To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM
- Additionally, use the `defaultValue` property to specify initial value in React

```
1 | <input defaultValue="Bob" type="text" ref={this.input} />
```

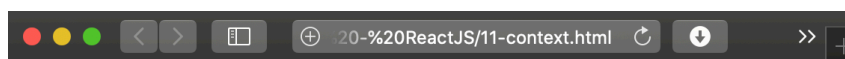
10.0 Context

- While passing `props` from parent to child, `...props` needs to be passed down through all the children
- Becomes tedious
- For example here, `App` needs to pass the props down to `Dashboard` and then finally to `Profile`, even though `Dashboard` does not use it

Inside the body `<script type="text/babel"></script>` tags

```
1 class Profile extends React.Component {
2   render() {
3     return (
4       <h1>Hello, {this.props.username} </h1>
5     )
6   }
7 }
8
9
10 function Dashboard(props) {
11   return <Profile {...props}/>
12 }
13
14 class App extends React.Component {
15   constructor(props) {
16     super(props);
17   }
18
19   render() {
20     return (
21       <Dashboard {...this.props}/>
22     );
23   }
24 }
25
26
27 ReactDOM.render(
28   <App username="thor"/>,
29   document.querySelector('#container')
30 );
```

Rendered in a browser



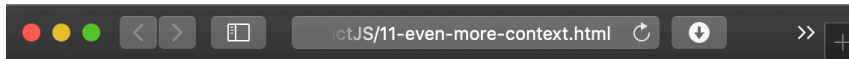
Hello, thor

- To make this using context, we create a context object by calling `React.createContext()`, which has two keys: `Provider` and `Consumer`
- The property (state) passes on from the `Provider` to the `Consumer`, not having to be passed through every level

Inside the body `<script type="text/babel"></script>` tags

```
1  const UContext = React.createContext();
2
3  class Profile extends React.Component {
4    static contextType = UContext;
5    render() {
6      return (
7        <h1>Hello, {this.context} </h1>
8      );
9    }
10 }
11
12
13 function Dashboard(props) {
14   return <Profile/>
15 }
16
17 class App extends React.Component {
18   render() {
19     return (
20       <UContext.Provider value={this.props.username}>
21         <Dashboard/>
22       </UContext.Provider>
23     );
24   }
25 }
26
27
28 ReactDOM.render(
29   <App username="thor"/>,
30   document.querySelector('#container')
31 );
```

Rendered in a browser



Hello, thor

Without defining a `static contextType`

Inside the body `<script type="text/babel"></script>` tags

```
1  const {Provider, Consumer} = React.createContext();
2
3  class Profile extends React.Component {
4    render() {
5      return (
6        <Consumer>
7          {function (user) {
8            return (
9              <h1>Hello, {user.username} </h1>
10             )
11           }}
12        </Consumer>
13      )
14    }
15  }
16
17
18  function Dashboard(props) {
19    return <Profile/>
20  }
21
22  class App extends React.Component {
23    constructor(props) {
24      super(props);
25
26      this.state = {
27        username: ''
28      }
29    }
30
31    componentDidMount() {
32      this.setState({
33        username: this.props.username
34      })
35    }
36  }
```

```
35     }
36
37     render() {
38         return (
39             <Provider value={this.state}>
40                 <DashBoard/>
41             </Provider>
42         );
43     }
44 }
45
46
47 ReactDOM.render(
48     <App uname="thor"/>,
49     document.querySelector('#container')
50 );
```

Rendered in a browser



Hello, thor