

Daniel W.



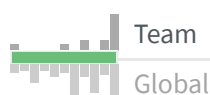
React: Fundamentals Results

Interact

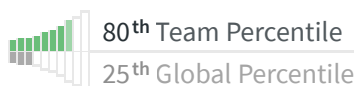
Interviewer: Dev Eval

Assessment Summary

100%



1 h, 12 m, 19 s Active Time



- 🕒 **Invited** by **Dev Eval** on Tuesday, July 7, 2020 3:06 PM
Sent once, and opened 5 times
- 🕒 Daniel Wallace **opened** this assessment on Tuesday, July 7, 2020 3:36 PM
- 🕒 Daniel Wallace **started** this assessment on Wednesday, July 8, 2020 2:20 PM
- 🕒 Daniel Wallace **submitted** this assessment after **1 hour and 18 minutes** on Wednesday, July 8, 2020 3:39 PM
- 🕒 This candidate spent **1 hour and 12 minutes** active in the browser working on the assessment which places them in the **80th Team Percentile** amongst candidates

Review Summary

- 🟢 Danny Brinlee seems to have a decent familiarity with react

Solutions Summary

Challenge	Score	Active Time
✅ #1: React: Click Counter	100%	5 m, 40 s
✅ #2: React: Turbo Click Counter	100%	10 m, 40 s
✅ #3: React: Unique List	100%	37 m, 35 s
✅ #4: React: Ordered List	100%	18 m, 24 s

#1: React: Click Counter



✓ Scoring

100%

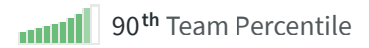
16 / 16 Tests (1 Attempt)



🕒 Timing

5 m, 40 s Active Time

6,093 ms Run Time



Challenge Reviews

● Danny Brinlee

Instructions

Task

Write a React component that will display the current value of an integer number counter.

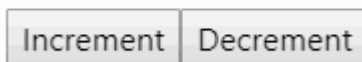
- The counter should start at 0.
- There should be a button to add 1 to the counter's value.
- There should be a button to subtract 1 from the counter's value.
- The counter value display should be rendered when the value changes.

Your class' `render` function should return:

- The counter display element with a class of `'counter'`, containing the counter value.
- An increment button with a class of `'increment'`.
- A decrement button with a class of `'decrement'`.

Here's a demo of the working component:

0



Feel free to consult React's [documentation on handling events](https://reactjs.org/docs/handling-events.html) if you get stuck.

Solution Code

 src/Counter.jsx

```

1  import React from "react";
2
3  class Counter extends React.Component {
4    constructor(props) {
5      super(props);
6
7      this.state = {
8        counter: 0
9      }
10   }
11
12   onIncrementClick = () => {
13     this.setState({
14       counter: this.state.counter + 1
15     })
16   }
17
18   onDecrementCount = () => {
19     this.setState({
20       counter: this.state.counter - 1
21     })
22   }
23
24   render() {
25     const { counter } = this.state;
26
27     return (
28       <div>
29         <h1 className="counter">{counter}</h1>
30         <button type="button" onClick={this.onDecrementCount} className="decrement">
31           Decrement
32         </button>
33         <button type="button" onClick={this.onIncrementClick} className="increment">
34           Increment
35         </button>
36       </div>
37     );
38   }
39 }
40
41 export default Counter;
42

```

docs/environment.md

```

1  # Execution Environment
2  Two separate environments exist for this challenge. The test environment, which is remotely
   executed within a NodeJS runtime, and the web preview environment, which is bundled using
   Webpack. The `package.json` does not control both environments.
3
4  ## Test Environment

```

5 The code for this project is tested using Node 10, with Jest as the test framework. JSX is supported, as well as ES6 style imports. The code is executed in a remote environment, not within your browser. Any `console.log` statements executed while running tests will not send logs to your browser, but the output will be shown at the bottom of your tests. Jest does not support inline `console.log` statements. This is why logs come after all test output.

6

7 Any package that is already loaded into the runner environment is able to be used; there is no `package.json` file that manages what is executed within tests. Candidates are not able to import packages that are not already installed.

8

9 Stylesheets and other assets can be imported. See `jest.config.js` and the `__mocks__` directory to see what is loaded in their place. By default, these imports have no impact on tests, only the web preview environment.

10

11 **## Web Preview Environment**

12 The web preview environment uses Sandpack to bundle assets. The `package.json` file is used to determine what gets loaded into this environment. There are no special config files; everything is inferred from package.json.

13

14 Customization is possible, but be careful not to use different versions within the web preview environment than what is loaded within the test environment. In some cases you can load additional dependencies if it is only cosmetic (styling) or meant for building preview assets, but those dependencies may not be available within the runner environment and can cause issues with your tests.

15


`__tests__/Counter.test.jsx`

```
1 import React from "react";
2 import Enzyme, { mount, shallow } from "enzyme";
3 import Adapter from "enzyme-adapter-react-16";
4 Enzyme.configure({ adapter: new Adapter() });
5 import sinon from "sinon";
6 import Counter from "../src/Counter";
7
8 describe("Counter", () => {
9   test("has an element with a 'counter' class", () => {
10     const wrapper = mount(<Counter />);
11     expect(wrapper.exists(".counter")).toBe(true);
12   });
13
14   test("has an element with an 'increment' class", () => {
15     const wrapper = mount(<Counter />);
16     expect(wrapper.exists(".increment")).toBe(true);
17   });
18
19   test("has an element with a 'decrement' class", () => {
20     const wrapper = mount(<Counter />);
21     expect(wrapper.exists(".decrement")).toBe(true);
22   });
23
24   test("shows initial value", () => {
```

```

25     const wrapper = mount(<Counter />);
26     expect(wrapper.find('.counter').text()).toEqual("0");
27   });
28
29   it("increments counter by 1 when increment button is clicked", () => {
30     const wrapper = mount(<Counter />);
31     wrapper.find(".increment").simulate("click");
32     expect(wrapper.find(".counter").text()).toEqual("1");
33   });
34
35   it("decrements counter by 1 when decrement button is clicked", () => {
36     const wrapper = mount(<Counter />);
37     wrapper.find(".decrement").simulate("click");
38     expect(wrapper.find(".counter").text()).toEqual("-1");
39   });
40 });

```

 __tests__/Submission.test.jsx

```

1  import React from "react";
2  import Enzyme, { mount, shallow } from "enzyme";
3  import Adapter from "enzyme-adapter-react-16";
4  Enzyme.configure({ adapter: new Adapter() });
5  import sinon from "sinon";
6  import Counter from "../src/Counter";
7
8  describe("Counter", () => {
9    test("has an element with a 'counter' class", () => {
10     const wrapper = mount(<Counter />);
11     expect(wrapper.exists(".counter")).toBe(true);
12   });
13
14   test("has an element with an 'increment' class", () => {
15     const wrapper = mount(<Counter />);
16     expect(wrapper.exists(".increment")).toBe(true);
17   });
18
19   test("has an element with a 'decrement' class", () => {
20     const wrapper = mount(<Counter />);
21     expect(wrapper.exists(".decrement")).toBe(true);
22   });
23
24   test("shows initial value of 0", () => {
25     const wrapper = mount(<Counter />);
26     expect(wrapper.find('.counter').text()).toEqual("0");
27   });
28
29   it("increments counter by 1 when increment button is clicked", () => {
30     const wrapper = mount(<Counter />);
31     wrapper.find(".increment").simulate("click");
32     expect(wrapper.find(".counter").text()).toEqual("1");
33   });

```

```

34
35 it("decrements counter by 1 when decrement button is clicked", () => {
36   const wrapper = mount(<Counter />);
37   wrapper.find(".decrement").simulate("click");
38   expect(wrapper.find(".counter").text()).toEqual("-1");
39 });
40
41 for (let n = 1; n <= 10; n++) {
42   it(`works on a random increment/decrement test (${n} of 10)`, () => {
43     const wrapper = mount(<Counter />);
44     const decCount = Math.floor(Math.random() * 50)
45     const incCount = Math.floor(Math.random() * 50)
46
47     for (let i = 0; i < decCount; i++) {
48       wrapper.find(".decrement").simulate("click");
49     }
50
51     for (let i = 0; i < incCount; i++) {
52       wrapper.find(".increment").simulate("click");
53     }
54
55     expect(wrapper.find(".counter").text()).toEqual(incCount - decCount + "");
56   });
57 }
58 });

```

public/index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <!-- Head content will be replaced by WebPack -->
5    </head>
6    <body>
7      <!-- include any external stylesheets within the body instead
8      <link
9        rel="stylesheet"
10       href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
11     />
12     -->
13
14     <!-- Bootstrap -->
15     <link rel="stylesheet"
16       href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
17       integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSFWFphJiwGPXr1jddIhOegiu1FwO5qRgvFX0dJZ4"
18       crossorigin="anonymous">
19
20     <div id="root"></div>
21   </body>
22 </html>

```

src/styles.scss

```
1 #app {
2   margin: 2em;
3 }
4
```

__mocks__/fileMock.js

```
1 module.exports = {}
```

src/App.jsx

```
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import Counter from "./Counter";
4
5 function App(props) {
6   return (
7     <div id="app">
8       <Counter />
9     </div>
10  );
11 }
12
13 export default App;
```

package.json

```
1 {
2   "name": "react-app",
3   "version": "1.0.0",
4   "description": "",
5   "main": "src/index.jsx",
6   "dependencies": {
7     "react": "16.8.3",
8     "react-dom": "16.8.3",
9     "react-scripts": "2.0.3",
10    "axios": "0.18.0",
11    "mobx": "5.9.4",
12    "mobx-react": "5.4.3",
13    "prop-types": "15.7.2",
14    "recompose": "0.30.0",
15    "redux": "4.0.1",
16    "@babel/runtime": "7.4.4"
17  },
18  "devDependencies": {
19    "sass": "1.20.1",
20    "sinon": "7.3.1"
```

```
21 },
22 "scripts": {
23   "start": "react-scripts start",
24   "build": "react-scripts build",
25   "test": "react-scripts test --env=jsdom",
26   "eject": "react-scripts eject"
27 }
28 }
```

src/index.jsx

```
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import "./styles.scss";
4 import App from "./App";
5
6 const rootElement = document.getElementById("root");
7 ReactDOM.render(<App />, rootElement);
```

__mocks__/styleMock.js

```
1 module.exports = {}
```

jest.config.js

```
1 module.exports = {
2   verbose: false,
3   testEnvironment: "jsdom",
4   reporters: [["jest-reporter", {}]],
5   moduleNameMapper: {
6     "\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$": "<rootDir>/__mocks__/fileMock.js",
7     "\\.(css|less|scss|sass)$": "<rootDir>/__mocks__/styleMock.js"
8   },
9 };
10
```


#2: React: Turbo Click Counter



✓ Scoring

100%

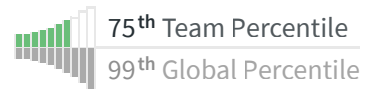
18 / 18 Tests (1 Attempt)



🕒 Timing

10 m, 40 s Active Time

6,452 ms Run Time



Challenge Reviews

● Danny Brinlee

Instructions

Task

This challenge is a boost on a basic Click Counter (the component code for which has been provided in the workspace). The `Counter` component offers a readout of its current value and increment/decrement buttons to adjust that value. You can run this component using Web Preview.

In this challenge, your task is to create a component called `TurboCounter`. This component will add a second set of controls which will enable the user to adjust the step size (the amount of increase or decrease) of the basic Click Counter's increment and decrement behavior.

Functional requirements

This section outlines the requirements for this challenge. You'll be evaluated on a test suite provided in `__tests__/TurboCounter.test.jsx` in the project directory. These Jest/Enzyme tests will verify that your component renders certain elements which exhibit the behavior described below.

Rendering

Your `TurboCounter`'s `render` function should return the controls and display for a basic `Counter`:

- An element with a class of `.counter` showing the counter's current value.
- A counter increment button with a class of `.increment`.
- A counter decrement button with a class of `.decrement`.

Additionally, expand the `render` function to also create:

- An element with a class of `.step-size` which displays the current step size.
- A step size increment button with a class of `.increment-step-size`.
- A step size decrement button with a class of `.decrement-step-size`.

The step size should start at 1 and should never be allowed to go below 1.

You may modify the provided `Counter` component as you see fit.

Button text throughout the challenge will not be tested. There is no predefined correct way of handling the CSS or design; the goal is to get a sense for your comfort level with both.

Here's a screen capture of the working `TurboCounter`.



Feel free to consult React's documentation for [handling events](https://reactjs.org/docs/handling-events.html) if you get stuck.

Solution Code

`src/Counter.jsx`

```
1  import React from "react";
2
3  class Counter extends React.Component {
4    constructor(props) {
5      super(props);
6      this.state = {counter: 0};
7      this.incrementCount = this.incrementCount.bind(this);
8      this.decrementCount = this.decrementCount.bind(this);
9    }
10
11   incrementCount() {
12     this.setState(state => ({counter: state.counter + this.props.stepSize}));
13   }
14
15   decrementCount() {
16     this.setState(state => ({counter: state.counter - this.props.stepSize}));
17   }
18
19   render() {
20     return (
21       <div>
22         <h1 className="counter">{this.state.counter}</h1>
23         <button
24           type="button"
25           className="increment"
```

```
26         onClick={this.incrementCount}
27       >Increment</button>
28     <button
29       type="button"
30       className="decrement"
31       onClick={this.decrementCount}
32     >Decrement</button>
33   </div>
34 );
35 }
36 }
37
38 export default Counter;
39
```

docs/environment.md

```
1 # Execution Environment
2 Two separate environments exist for this challenge. The test environment, which is remotely
3 executed within a NodeJS runtime, and the web preview environment, which is bundled using
4 Webpack. The `package.json` does not control both environments.
5
6 ## Test Environment
7 The code for this project is tested using Node 10, with Jest as the test framework. JSX is
8 supported, as well as ES6 style imports. The code is executed in a remote environment, not
9 within your browser. Any `console.log` statements executed while running tests will not send
10 logs to your browser, but the output will be shown at the bottom of your tests. Jest does not
11 support inline `console.log` statements. This is why logs come after all test output.
12
13 Any package that is already loaded into the runner environment is able to be used; there is no
14 `package.json` file that manages what is executed within tests. Candidates are not able to
15 import packages that are not already installed.
16
17 Stylesheets and other assets can be imported. See `jest.config.js` and the `__mocks__` directory
18 to see what is loaded in their place. By default, these imports have no impact on tests, only
19 the web preview environment.
20
21 ## Web Preview Environment
22 The web preview environment uses Sandpack to bundle assets. The `package.json` file is used to
23 determine what gets loaded into this environment. There are no special config files; everything
24 is inferred from package.json.
25
26 Customization is possible, but be careful not to use different versions within the web preview
27 environment than what is loaded within the test environment. In some cases you can load
28 additional dependencies if it is only cosmetic (styling) or meant for building preview assets,
29 but those dependencies may not be available within the runner environment and can cause issues
30 with your tests.
```

__tests__/TurboCounter.test.jsx

```

1 import React from "react";
2 import Enzyme, { mount, shallow } from "enzyme";
3 import Adapter from "enzyme-adapter-react-16";
4 Enzyme.configure({ adapter: new Adapter() });
5 import sinon from "sinon";
6 import TurboCounter from "../src/TurboCounter";
7
8 let wrapper;
9
10 describe("TurboCounter", () => {
11   describe("basic counter functionality", () => {
12     beforeEach(() => {
13       wrapper = mount(<TurboCounter />);
14     });
15
16     test("has an element with a '.counter' class", () => {
17       expect(wrapper.exists(".counter")).toBe(true);
18     });
19
20     test("has an element with an '.increment' class", () => {
21       expect(wrapper.exists(".increment")).toBe(true);
22     });
23
24     test("has an element with a '.decrement' class", () => {
25       expect(wrapper.exists(".decrement")).toBe(true);
26     });
27
28     test("''.counter' shows an initial value of 0", () => {
29       expect(wrapper.find('.counter').text()).toEqual("0");
30     });
31
32     test("increments counter by 1 when increment button is clicked", () => {
33       wrapper.find(".increment").simulate("click");
34       expect(wrapper.find(".counter").text()).toEqual("1");
35     });
36
37     test("decrements counter by 1 when decrement button is clicked", () => {
38       wrapper.find(".decrement").simulate("click");
39       expect(wrapper.find(".counter").text()).toEqual("-1");
40     });
41   });
42
43   describe("step size controller functionality", () => {
44     beforeEach(() => {
45       wrapper = mount(<TurboCounter />);
46     });
47
48     test("has an element with a '.step-size' class", () => {
49       expect(wrapper.exists(".step-size")).toBe(true);
50     });
51
52     test("has an element with an '.increment-step-size' class", () => {
53       expect(wrapper.exists(".increment-step-size")).toBe(true);

```

```


54   });
55
56   test("has an element with a '.decrement-step-size' class", () => {
57     expect(wrapper.exists(".decrement-step-size")).toBe(true);
58   });
59
60   test("'step-size' shows an initial value of 1", () => {
61     expect(wrapper.find('step-size').text()).toEqual("1");
62   });
63
64   test("increments step size by 1 when '.increment-step-size' button is clicked", () => {
65     wrapper.find("increment-step-size").simulate("click");
66     expect(wrapper.find("step-size").text()).toEqual("2");
67   });
68
69   test("decrements step size by 1 when '.decrement-step-size' button is clicked", () => {
70     wrapper.find("increment-step-size").simulate("click");
71     expect(wrapper.find("step-size").text()).toEqual("2");
72     wrapper.find("decrement-step-size").simulate("click");
73     expect(wrapper.find("step-size").text()).toEqual("1");
74   });
75
76   test("doesn't go below 1", () => {
77     wrapper.find("decrement-step-size").simulate("click");
78     expect(wrapper.find("step-size").text()).toEqual("1");
79   });
80
81   test("increment works repeatedly, increasing the value to 5", () => {
82     for (let i = 2; i < 6; i++) {
83       wrapper.find("increment-step-size").simulate("click");
84       expect(wrapper.find("step-size").text()).toEqual("" + i);
85     }
86   });
87 });
88
89 describe("step size and counter operate together", () => {
90   beforeEach(() => {
91     wrapper = mount(<TurboCounter />);
92   });
93
94   test("increments counter by 5 when increment button is clicked", () => {
95     for (let i = 0; i < 4; i++) {
96       wrapper.find("increment-step-size").simulate("click");
97     }
98
99     expect(wrapper.find("counter").text()).toEqual("0");
100    wrapper.find("increment").simulate("click");
101    expect(wrapper.find("counter").text()).toEqual("5");
102  });
103
104  test("decrements counter by 5 when decrement button is clicked", () => {
105    for (let i = 0; i < 4; i++) {
106      wrapper.find("increment-step-size").simulate("click");
107    }

```

```

108
109     wrapper.find(".decrement").simulate("click");
110     expect(wrapper.find(".counter").text()).toEqual("-5");
111 });
112
113 test("should work on multiple operations", () => {
114     wrapper.find(".increment-step-size").simulate("click");
115     expect(wrapper.find(".step-size").text()).toEqual("2");
116     wrapper.find(".increment").simulate("click");
117     expect(wrapper.find(".counter").text()).toEqual("2");
118     wrapper.find(".increment-step-size").simulate("click");
119     expect(wrapper.find(".step-size").text()).toEqual("3");
120     wrapper.find(".increment").simulate("click");
121     expect(wrapper.find(".counter").text()).toEqual("5");
122     wrapper.find(".decrement").simulate("click");
123     expect(wrapper.find(".counter").text()).toEqual("2");
124 });
125 });
126 });

```

 __tests__/Submission.test.jsx

```

1  import React, { useState } from "react";
2  import Enzyme, { mount, shallow } from "enzyme";
3  import Adapter from "enzyme-adapter-react-16";
4  Enzyme.configure({ adapter: new Adapter() });
5  import sinon from "sinon";
6  import TurboCounter from "../src/TurboCounter";
7
8  let wrapper;
9  let wrapperRef;
10
11 describe("TurboCounter", () => {
12     beforeEach(() => {
13         wrapper = mount(<TurboCounter />);
14         wrapperRef = mount(<TurboCounterRef />);
15     });
16
17     test("random test", () => {
18         for (let i = 0; i < 50; i++) {
19             const action = Math.random() * 4;
20
21             if (action > 3) {
22                 wrapper.find(".increment-step-size").simulate("click");
23                 wrapperRef.find(".increment-step-size").simulate("click");
24             }
25             else if (action > 2) {
26                 wrapper.find(".decrement-step-size").simulate("click");
27                 wrapperRef.find(".decrement-step-size").simulate("click");
28             }
29             else if (action > 1) {
30                 wrapper.find(".increment").simulate("click");

```

```

31     wrapperRef.find(".increment").simulate("click");
32   }
33   else {
34     wrapper.find(".decrement").simulate("click");
35     wrapperRef.find(".decrement").simulate("click");
36   }
37
38   const refStep = wrapperRef.find(".step-size").text();
39   const refCounter = wrapperRef.find(".counter").text();
40   expect(wrapper.find(".step-size").text()).toEqual(refStep);
41   expect(wrapper.find(".counter").text()).toEqual(refCounter);
42 }
43 });
44 });
45
46 const CounterRef = ({stepSize}) => {
47   const [count, setCount] = useState(0);
48
49   return (
50     <div className="cnt">
51       <h1 className="counter">{count}</h1>
52       <div>
53         <button
54           type="button"
55           className="increment"
56           onClick={() => setCount(count + stepSize)}
57         >Increment</button>
58       </div>
59       <div>
60         <button
61           type="button"
62           className="decrement"
63           onClick={() => setCount(count - stepSize)}
64         >Decrement</button>
65       </div>
66     </div>
67   );
68 };
69
70 const IncrementCounterRef = ({stepSize, onInc, onDec}) =>
71   <div className="inc">
72     <h3 className="step-size">{stepSize}</h3>
73     <div>
74       <button
75         type="button"
76         className="increment-step-size"
77         onClick={onInc}
78       >Step size +</button>
79     </div>
80     <div>
81       <button
82         type="button"
83         className="decrement-step-size"
84         onClick={onDec}

```

```

85     >Step size -</button>
86   </div>
87 </div>
88 ;
89
90 const TurboCounterRef = () => {
91   const [stepSize, setStepSize] = useState(1);
92
93   return (
94     <div className="turbo-counter">
95       <CounterRef stepSize={stepSize} />
96       <IncrementCounterRef
97         stepSize={stepSize}
98         onInc={() => setStepSize(stepSize + 1)}
99         onDec={() => setStepSize(Math.max(1, stepSize - 1))}
100       />
101     </div>
102   );
103 };

```

src/styles.scss

```

1  #app {
2    margin: 2em;
3  }
4

```

src/App.jsx

```

1  import React from "react";
2  import ReactDOM from "react-dom";
3  import TurboCounter from "../TurboCounter";
4
5  function App(props) {
6    return (
7      <div id="app">
8        <TurboCounter />
9      </div>
10   );
11 }
12
13 export default App;

```

__mocks__/fileMock.js

```

1  module.exports = {}

```

public/index.html


```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <!-- Head content will be replaced by WebPack -->
5   </head>
6   <body>
7     <!-- include any external stylesheets within the body instead
8     <link
9       rel="stylesheet"
10      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
11    />
12    -->
13
14    <!-- Bootstrap -->
15    <link rel="stylesheet"
16 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
17 integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSFWFphJiwGPXr1jddIhOegiu1Fw05qRQvFX0dJZ4"
18 crossorigin="anonymous">
19
20    <div id="root"></div>
21  </body>
22 </html>
```

src/index.jsx

```
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import "./styles.scss";
4 import App from "./App";
5
6 const rootElement = document.getElementById("root");
7 ReactDOM.render(<App />, rootElement);
```

__mocks__/styleMock.js

```
1 module.exports = {}
```

jest.config.js

```
1 module.exports = {
2   verbose: false,
3   testEnvironment: "jsdom",
4   reporters: [["jest-reporter", {}]],
5   moduleNameMapper: {
6     "\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$": "<rootDir>/__mocks__/fileMock.js",
```

```
7     "\\.(css|less|scss|sass)$": "<rootDir>/__mocks__/styleMock.js"
8   },
9 };
10
```

src/TurboCounter.jsx

```
1  import React from "react";
2  import Counter from "../Counter";
3
4  class TurboCounter extends React.Component {
5    constructor(props) {
6      super(props);
7
8      this.state = {
9        stepSize: 1
10     }
11
12     this.incrementStepSize = this.incrementStepSize.bind(this);
13     this.decrementStepSize = this.decrementStepSize.bind(this);
14   }
15
16   incrementStepSize() {
17     this.setState(state => ({stepSize: state.stepSize + 1}));
18   }
19
20   decrementStepSize() {
21     this.setState(state => ({stepSize: Math.max(state.stepSize - 1, 1)}));
22   }
23
24   render() {
25     const { stepSize } = this.state;
26
27     return (
28       <>
29         <Counter stepSize={stepSize} />
30         <h2 className="step-size">{stepSize}</h2>
31         <button
32           type="button"
33           className="increment-step-size"
34           onClick={this.incrementStepSize}
35         >Step size +</button>
36         <button
37           type="button"
38           className="decrement-step-size"
39           onClick={this.decrementStepSize}
40         >Step size -</button>
41       </>
42     );

```

```
43   }
44 }
45
46 export default TurboCounter;
```

package.json

```
1  {
2    "name": "react-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/index.jsx",
6    "dependencies": {
7      "react": "16.8.3",
8      "react-dom": "16.8.3",
9      "react-scripts": "2.0.3",
10     "axios": "0.18.0",
11     "mobx": "5.9.4",
12     "mobx-react": "5.4.3",
13     "prop-types": "15.7.2",
14     "recompose": "0.30.0",
15     "redux": "4.0.1",
16     "@babel/runtime": "7.4.4"
17   },
18   "devDependencies": {
19     "sass": "1.20.1",
20     "sinon": "7.3.1"
21   },
22   "scripts": {
23     "start": "react-scripts start",
24     "build": "react-scripts build",
25     "test": "react-scripts test --env=jsdom",
26     "eject": "react-scripts eject"
27   }
28 }
```

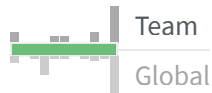
#3: React: Unique List



✓ Scoring

100%

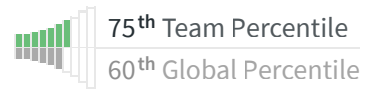
17 / 17 Tests (1 Attempt)



🕒 Timing

37 m, 35 s Active Time

5,628 ms Run Time



Challenge Reviews

● Danny Brinlee

Instructions

Task

Web applications often require keeping a unique list of items entered by a user. For example, an app containing a list of category tags for a post should prevent the user from entering duplicate tags.

For this challenge, you'll create a simple component `UniqueList` which renders the following skeleton:

```
<>
<div>
  <input
    className="item-input"
    type="text"
    value={text}
    onChange={onChangeHandler}
    onKeyDown={onKeyDownHandler}
  />
  <input
    className="add-button"
    type="button"
    value="Add Item"
    onClick={addHandler}
  />
  <input
    className="remove-button"
    type="button"
    value="Remove Item"
    onClick={removeHandler}
  />
  <input
    className="clear-button"
    type="button"
    value="Clear Items"
    onClick={clearHandler}
  />
</div>
```

```
    />
  </div>
  <ul className="items">
  </ul>
</>
```

Note that you may change the above function and variable names for your handlers, but make sure the DOM remains functionally intact for the test suite to use.

Functionality Specifications

Here are the requirements you'll be implementing for the `UniqueList` component.

Text input

The user will enter text into the `<input class="item-input" />` field. This input element should be a controlled component (<https://reactjs.org/docs/forms.html#controlled-components>) and have a listener for `Enter` key presses, which will activate the add item functionality as if the `<input class="add-button" />` button had been clicked. The input element should also respond to `onChange` for updating state. Upon submission of the input field's content, the value should have leading and trailing whitespace trimmed before any subsequent operation is performed.

Adding items

After whitespace trimming, an item from the submitted input using the add button (or, equivalently, pressing `Enter` when focused on the `<input class="item-input" />` element) will be added to the list and appended to the `<ul class="items">` as an `` child if it is not an empty string and it is not already present in the list (case sensitive).

Removing items

After whitespace trimming, an item from the submitted input using the remove item button (`<input class="remove-button" />`) will be removed from the list if it is not an empty string and it is not already present in the list (case sensitive).

Clearing items

The clear items button (`<input class="clear-button" />`) should clear the list completely, updating the state only if the list wasn't already empty.

Updating state and DOM

Only update the component's state if the add, remove or clear operation resulted in a change to the list contents. More specifically, if any of the add, remove or clear operations failed in response to an event (the text input element was empty or entirely whitespace, the item was already in the list upon add, the item wasn't in the list upon removal, or a clear operation was performed on an empty list), don't update the DOM or clear the input text field.

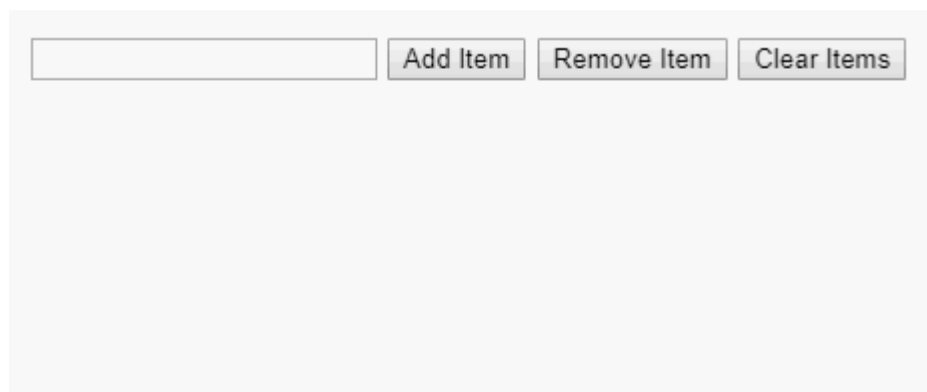
Conversely, if any action was successful in updating DOM state, clear the input element's text content.

Style

Include some simple CSS and design in your component's stylesheet. There is no predefined correct way of handling these; the goal is to get a sense for your comfort level with both.

Demo

This demo shows the finished app in action. Note cases when the input element is cleared and when it isn't, and the trimming behavior for all strings input via the text field.



Solution Code

 docs/environment.md

```
1 # Execution Environment
2 Two separate environments exist for this challenge. The test environment, which is remotely
  executed within a NodeJS runtime, and the web preview environment, which is bundled using
  Webpack. The `package.json` does not control both environments.
3
4 ## Test Environment
5 The code for this project is tested using Node 10, with Jest as the test framework. JSX is
  supported, as well as ES6 style imports. The code is executed in a remote environment, not
  within your browser. Any `console.log` statements executed while running tests will not send
  logs to your browser, but the output will be shown at the bottom of your tests. Jest does not
  support inline `console.log` statements. This is why logs come after all test output.
6
7 Any package that is already loaded into the runner environment is able to be used; there is no
  `package.json` file that manages what is executed within tests. Candidates are not able to
  import packages that are not already installed.
8
9 Stylesheets and other assets can be imported. See `jest.config.js` and the `__mocks__` directory
  to see what is loaded in their place. By default, these imports have no impact on tests, only
  the web preview environment.
10
11 ## Web Preview Environment
```

12 The web preview environment uses Sandpack to bundle assets. The `package.json` file is used to
determine what gets loaded into this environment. There are no special config files; everything
is inferred from package.json.

13

14 Customization is possible, but be careful not to use different versions within the web preview
environment than what is loaded within the test environment. In some cases you can load
additional dependencies if it is only cosmetic (styling) or meant for building preview assets,
but those dependencies may not be available within the runner environment and can cause issues
with your tests.

15

src/App.jsx

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3  import "./style.scss";
4  import UniqueList from "./UniqueList";
5
6  /* Changes made to this file will not affect your tests.
7   * This file is used to control the behavior of the web preview.
8   */
9  function App() {
10   return (
11     <div className="App">
12       <UniqueList />
13     </div>
14   );
15 }
16
17 ReactDOM.render(<App />, document.getElementById("root"));
```

src/style.scss

```
1  html, body, #root {
2    height: 100%;
3  }
4
5  body {
6    background-color: #f8f8f8;
7  }
8
9  .App {
10   font-family: sans-serif;
11   padding: 1em;
12   height: 100%;
13 }
14
15 .items {
16   list-style: none;
17   margin-top: 1rem;
18   padding: 0;
```

```

19   li {
20     padding: 1rem 0;
21     border-bottom: 1px solid grey;
22     font-size: 14px;
23
24     &:last-child {
25       border-bottom: 0;
26     }
27   }
28 }
29
30 .toolbar {
31   display: flex;
32   flex-direction: row;
33
34   > input[type="text"] {
35     flex: 1;
36     padding: .5rem;
37     margin-right: .5rem;
38   }
39
40   > input[type="button"] {
41     flex: 0;
42     margin-right: .5rem;
43   }
44 }

```

package.json

```

1  {
2    "name": "react-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/App.jsx",
6    "dependencies": {
7      "react": "16.8.3",
8      "react-dom": "16.8.3",
9      "react-scripts": "2.0.3",
10     "axios": "0.18.0",
11     "mobx": "5.9.4",
12     "mobx-react": "5.4.3",
13     "prop-types": "15.7.2",
14     "recompose": "0.30.0",
15     "redux": "4.0.1",
16     "@babel/runtime": "7.4.4"
17   },
18   "devDependencies": {
19     "sass": "1.20.1",
20     "sinon": "7.3.1"
21   },
22   "scripts": {
23     "start": "react-scripts start",

```



```
24   "build": "react-scripts build",
25   "test": "react-scripts test --env=jsdom",
26   "eject": "react-scripts eject"
27 }
28 }
```

src/UniqueList.jsx

```
1  import React from "react";
2
3  class UniqueList extends React.Component {
4    constructor(props) {
5      super(props);
6
7      this.state = {
8        inputValue: '',
9        items: []
10     }
11   }
12
13   onInputChange = (e) => {
14     this.setState({
15       inputValue: e.target.value
16     })
17   }
18
19   checkValue = (value) => {
20     return value.trim().length > 0
21   }
22
23   checkUnique = (value) => {
24     return !this.state.items.find(item => item === value.trim())
25   }
26
27   addItem = () => {
28     const { inputValue, items } = this.state;
29
30     if (this.checkValue(inputValue) && this.checkUnique(inputValue)) {
31       this.setState({
32         items: [...items, inputValue.trim()],
33         inputValue: ''
34       })
35     } else {
36       // needs some error handling
37     }
38   }
39
40   onAddItemClick = () => {
41     this.addItem();
42   }
43
44   onClearItemsClick = () => {
```

```

45     const { items, inputValue } = this.state;
46
47     this.setState({
48       items: [],
49       inputValue: items.length ? '' : inputValue
50     })
51   }
52
53   onRemoveItemClick = () => {
54     const { inputValue, items } = this.state;
55     const exists = !!items.find(item => item === inputValue.trim())
56
57     if (this.checkValue(inputValue) && exists) {
58       this.setState({
59         items: items.filter(item => item !== inputValue.trim()),
60         inputValue: ''
61       })
62     } else {
63       // some error
64     }
65   }
66
67   onInputKeyDown = (e) => {
68     if (e.keyCode === 13) {
69       this.addItem();
70     }
71   }
72
73   render() {
74     const { inputValue, items } = this.state;
75
76     return (
77       <>
78         <div className="toolbar">
79           <input
80             className="item-input"
81             type="text"
82             value={inputValue}
83             onChange={this.onInputChange}
84             onKeyDown={this.onInputKeyDown}
85           />
86           <input
87             className="add-button"
88             type="button"
89             value="Add Item"
90             onClick={this.onAddItemClick}
91           />
92           <input
93             className="remove-button"
94             type="button"
95             value="Remove Item"
96             onClick={this.onRemoveItemClick}
97           />
98           <input

```

```

99         className="clear-button"
100         type="button"
101         value="Clear Items"
102         onClick={this.onClearItemsClick}
103     />
104 </div>
105 <ul className="items">
106     {items.map(item => (
107         <li key={item}>{item}</li>
108     ))}
109 </ul>
110 </>
111 );
112 }
113 }
114
115 export default UniqueList;

```

__tests__/UniqueList.test.jsx

```

1  import React from "react";
2  import Enzyme, { mount, shallow } from "enzyme";
3  import Adapter from "enzyme-adapter-react-16";
4  Enzyme.configure({ adapter: new Adapter() });
5  import sinon from "sinon";
6  import UniqueList from "../src/UniqueList";
7  let ul;
8
9  describe("UniqueList Basic Functionality", () => {
10     beforeEach(() => {
11         ul = mount(<UniqueList />);
12     });
13
14     test("item input element exists", () => {
15         expect(ul.find(".item-input").exists()).toBe(true);
16         expect(ul.find(".item-input")).toHaveLength(1);
17     });
18
19     test("add item button element exists", () => {
20         expect(ul.find(".add-button").exists()).toBe(true);
21         expect(ul.find(".add-button")).toHaveLength(1);
22     });
23
24     test("remove item button element exists", () => {
25         expect(ul.find(".remove-button").exists()).toBe(true);
26         expect(ul.find(".remove-button")).toHaveLength(1);
27     });
28
29     test("clear item button element exists", () => {
30         expect(ul.find(".clear-button").exists()).toBe(true);
31         expect(ul.find(".clear-button")).toHaveLength(1);
32     });

```

```

33
34 test("items list starts out empty", () => {
35     const items = ul.find(".items");
36     expect(items.text()).toEqual("");
37 });
38
39 test("input starts out empty", () => {
40     const input = ul.find(".item-input");
41     expect(input.text()).toEqual("");
42 });
43
44 test("input value changes on key presses", () => {
45     const input = ul.find(".item-input");
46     input.simulate("change", { target: { value: "apple" } });
47     expect(input.instance().value).toEqual("apple");
48 });
49
50 test("item is added after Enter keydown", () => {
51     const input = ul.find(".item-input");
52     const items = ul.find(".items");
53     input.simulate("change", { target: { value: "apple" } });
54     input.simulate("keydown", {
55         key: "Enter",
56         type: "keydown",
57         which: 13,
58         keyCode: 13,
59     });
60     expect(items.text()).toEqual("apple");
61 });
62
63 test("item is added after clicking the Add Item button", () => {
64     const add = ul.find(".add-button");
65     const input = ul.find(".item-input");
66     const items = ul.find(".items");
67     input.simulate("change", { target: { value: "apple" } });
68     add.simulate("click");
69     expect(items.text()).toEqual("apple");
70 });
71
72 test("duplicate item is not added", () => {
73     const input = ul.find(".item-input");
74     const items = ul.find(".items");
75     input.simulate("change", { target: { value: "apple" } });
76     input.simulate("keydown", {
77         key: "Enter",
78         type: "keydown",
79         which: 13,
80         keyCode: 13,
81     });
82     expect(items.text()).toEqual("apple");
83     input.simulate("change", { target: { value: "apple" } });
84     input.simulate("keydown", {
85         key: "Enter",
86         type: "keydown",

```

```

87     which: 13,
88     keyCode: 13,
89   });
90   expect(items.text()).toEqual("apple");
91 });
92
93 test("item is removed after clicking Remove Item button", () => {
94   const input = ul.find(".item-input");
95   const remove = ul.find(".remove-button");
96   const items = ul.find(".items");
97   input.simulate("change", { target: { value: "apple" } });
98   input.simulate("keydown", {
99     key: "Enter",
100    type: "keydown",
101    which: 13,
102    keyCode: 13,
103  });
104   expect(items.text()).toEqual("apple");
105   input.simulate("change", { target: { value: "apple" } });
106   remove.simulate("click");
107   expect(items.text()).toEqual("");
108   expect(input.instance().value).toEqual("");
109 });
110
111 test("list is cleared after clicking the Clear Items button", () => {
112   const input = ul.find(".item-input");
113   const clear = ul.find(".clear-button");
114   const items = ul.find(".items");
115   input.simulate("change", { target: { value: "apple" } });
116   input.simulate("keydown", {
117     key: "Enter",
118     type: "keydown",
119     which: 13,
120     keyCode: 13,
121   });
122   expect(items.text()).toEqual("apple");
123   input.simulate("change", { target: { value: "banana" } });
124   input.simulate("keydown", {
125     key: "Enter",
126     type: "keydown",
127     which: 13,
128     keyCode: 13,
129   });
130   expect(items.text()).toEqual("applebanana");
131   input.simulate("change", { target: { value: "orange" } });
132   input.simulate("keydown", {
133     key: "Enter",
134     type: "keydown",
135     which: 13,
136     keyCode: 13,
137   });
138   expect(items.text()).toEqual("applebananaorange");
139   clear.simulate("click");
140   expect(items.text()).toEqual("");

```

```

141     expect(input.instance().value).toEqual("");
142   });
143 });
144
145 describe("UniqueList Advanced Functionality", () => {
146   beforeEach(() => {
147     ul = mount(<UniqueList />);
148   });
149
150   test("input is not cleared attempting to clear an empty list", () => {
151     const input = ul.find(".item-input");
152     const clear = ul.find(".clear-button");
153     const items = ul.find(".items");
154     input.simulate("change", { target: { value: "apple" } });
155     clear.simulate("click");
156     expect(input.instance().value).toEqual("apple");
157     expect(items.text()).toEqual("");
158   });
159
160   test("input is cleared attempting to clear a list with items in it", () => {
161     const input = ul.find(".item-input");
162     const clear = ul.find(".clear-button");
163     const items = ul.find(".items");
164     input.simulate("change", { target: { value: "apple" } });
165     input.simulate("keydown", {
166       key: "Enter",
167       type: "keydown",
168       which: 13,
169       keyCode: 13,
170     });
171     expect(input.instance().value).toEqual("");
172     expect(items.text()).toEqual("apple");
173     input.simulate("change", { target: { value: "banana" } });
174     expect(input.instance().value).toEqual("banana");
175     clear.simulate("click");
176     expect(input.instance().value).toEqual("");
177     expect(items.text()).toEqual("");
178   });
179
180   test("input is not cleared attempting to remove something not in the list", () => {
181     const input = ul.find(".item-input");
182     const remove = ul.find(".remove-button");
183     const items = ul.find(".items");
184     input.simulate("change", { target: { value: "apple" } });
185     remove.simulate("click");
186     expect(input.instance().value).toEqual("apple");
187     expect(items.text()).toEqual("");
188     input.simulate("change", { target: { value: "apple" } });
189     expect(input.instance().value).toEqual("apple");
190     expect(items.text()).toEqual("");
191     input.simulate("keydown", {
192       key: "Enter",
193       type: "keydown",
194       which: 13,

```

```

195     keyCode: 13,
196   });
197   expect(input.instance().value).toEqual("");
198   expect(items.text()).toEqual("apple");
199   input.simulate("change", { target: { value: "apples" } });
200   remove.simulate("click");
201   expect(input.instance().value).toEqual("apples");
202   expect(items.text()).toEqual("apple");
203 });
204
205 test("input is not cleared attempting to add something already in the list", () => {
206   const input = ul.find(".item-input");
207   const add = ul.find(".add-button");
208   const items = ul.find(".items");
209   input.simulate("change", { target: { value: "apple" } });
210   add.simulate("click");
211   expect(input.instance().value).toEqual("");
212   expect(items.text()).toEqual("apple");
213   input.simulate("change", { target: { value: "apple" } });
214   add.simulate("click");
215   expect(input.instance().value).toEqual("apple");
216   expect(items.text()).toEqual("apple");
217 });
218
219 test("input is trimmed before all operations", () => {
220   const input = ul.find(".item-input");
221   const remove = ul.find(".remove-button");
222   const items = ul.find(".items");
223   input.simulate("change", { target: { value: "  apple  " } });
224   input.simulate("keydown", {
225     key: "Enter",
226     type: "keydown",
227     which: 13,
228     keyCode: 13,
229   });
230   expect(input.instance().value).toEqual("");
231   expect(items.text()).toEqual("apple");
232   input.simulate("change", { target: { value: "  apple  " } });
233   remove.simulate("click");
234   expect(input.instance().value).toEqual("");
235   expect(items.text()).toEqual("");
236 });
237 });

```

public/index.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <!-- Head content will be replaced by WebPack -->
5   </head>
6   <body>

```

```
7 <!-- include any external stylesheets within the body instead
8 <link
9   rel="stylesheet"
10  href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
11 />
12 -->
13 <div id="root"></div>
14 </body>
15 </html>
16
```

public

1

__tests__

1

src

1

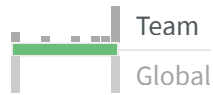
#4: React: Ordered List



✓ Scoring

100%

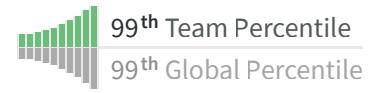
16 / 16 Tests (1 Attempt)



🕒 Timing

18 m, 24 s Active Time

5,580 ms Run Time



Challenge Reviews

● Danny Brinlee

📝 Candidate Notes:

I could add some css to this. My thoughts are the test isn't timed but I got a little behind trying to do some styling on the last question. Unsure how much to do.

Instructions

Task

Let's write a little component in React, `OrderedList`, which implements an alphabetically sorted list. The component will include a button to enable the user to sort either in ascending or descending order and a second button to permit the list to be cleared.

Your component should render specific elements for the testing suite to hook onto. The test suite will evaluate that the elements behave according to the following specifications:

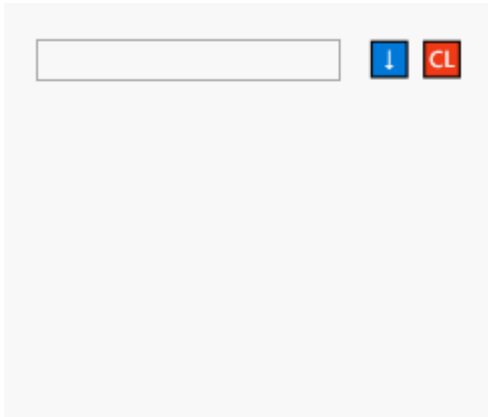
- An `<input class="add-item" />` element which the user can use to add items to the list. This input field should listen for `Enter` keydown events (the test suite triggers the `onKeyDown` event handler specifically) to add the current contents (if nonempty) to the list. After adding an item, the input box should be cleared.
- A `<button class="sort-direction">` element which the user can click to change the direction of the sort. Initially, the button should display text such as the `↓` emoji or the text `down`. When changed to a descending sort, the button should change to a `↑` emoji (or text such as `up`). The test suite will only test that pushing the `sort-direction` button toggles between two nonempty strings when clicked rather than checking that the string matches something in particular, so pick a string or icon that carries the most semantic meaning to you.
- A `<button class="clear-list">` element which the user can click to clear the list as well as any contents in the input box (essentially reverting to the component's default state). Use any text you'd like for this button.
- A `<ul class="items-list">` element, which should contain a series of `` elements that represent the sorted list contents.

You can assume all input typed into the box consists only of lowercase alphabetical characters for the purposes of this challenge.

There is no predetermined correct answer for style and CSS. Your solution need not look like the demo below but it should demonstrate understanding of basic CSS and user experience principles.

Demo

Here's a screen capture showing the component in action. Note the icon (button text) changes for up/down sorting.



Notes and guidance

Feel free to consult documentation at [MDN](https://developer.mozilla.org/) (https://developer.mozilla.org/) and [ReactJS](https://reactjs.org/docs/) (https://reactjs.org/docs/) if you get stuck.

If you want, you can copy emoji [up](https://emojipedia.org/upwards-black-arrow/) (https://emojipedia.org/upwards-black-arrow/) and [down](https://emojipedia.org/downwards-black-arrow/) (https://emojipedia.org/downwards-black-arrow/) arrows and other buttons at [emojipedia](https://emojipedia.org/) (https://emojipedia.org/).

Solution Code

 public/index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <!-- Head content will be replaced by WebPack -->
5   </head>
6   <body>
7     <!-- include any external stylesheets within the body instead
8     <link
9       rel="stylesheet"
10      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
11    />
12    -->
13    <div id="root"></div>
14  </body>
15 </html>
16
```

```
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import "./style.scss";
4 import OrderedList from "./OrderedList";
5
6 /* Changes made to this file will not affect your tests.
7  * This file is used to control the behavior of the web preview.
8  */
9 function App() {
10   return (
11     <div className="App">
12       <OrderedList />
13     </div>
14   );
15 }
16
17 ReactDOM.render(<App />, document.getElementById("root"));
```

```
1 html, body, #root {
2   height: 100%;
3 }
4
5 body {
6   background-color: #f8f8f8;
7 }
8
9 .App {
10  font-family: sans-serif;
11  margin: 2em;
12  height: 100%;
13 }
```

public

1

src

1

```

1 import React from "react";
2 import Enzyme, { mount, shallow } from "enzyme";
3 import Adapter from "enzyme-adapter-react-16";
4 Enzyme.configure({ adapter: new Adapter() });
5 import sinon from "sinon";
6 import OrderedList from "../src/OrderedList";
7
8 let wrapper;
9 let asc;
10 let desc;
11
12 const addAll = items => {
13   items.forEach(e => {
14     const addItem = wrapper.find(".add-item");
15     addItem.simulate("change", { target: { value: e } });
16     addItem.simulate("keydown", {
17       event: "keydown",
18       key: "Enter",
19       keyCode: 13,
20       which: 13
21     });
22   });
23 };
24
25 describe("OrderedList", () => {
26   beforeEach(() => {
27     wrapper = mount(<OrderedList />);
28   });
29   afterEach(() => wrapper.unmount());
30
31   describe("renders correct elements", () => {
32     test('should have a ".add-item" input', () => {
33       const addItem = wrapper.find(".add-item");
34       expect(addItem.exists()).toBe(true);
35       expect(addItem).toHaveLength(1);
36     });
37
38     test('should have a ".sort-direction" button with some text content', () => {
39       const sortDir = wrapper.find(".sort-direction");
40       expect(sortDir.exists()).toBe(true);
41       expect(sortDir).toHaveLength(1);
42       expect(sortDir.text()).not.toBe("");
43     });
44
45     test('should have a ".clear-list" button', () => {
46       const clear = wrapper.find(".clear-list");
47       expect(clear.exists()).toBe(true);
48       expect(clear).toHaveLength(1);
49     });
50
51     test('should have an ".items-list" element', () => {
52       const items = wrapper.find('.items-list');
53       expect(items.exists()).toBe(true);

```

```

54     expect(items).toHaveLength(1);
55   });
56 });
57
58 describe("responds correctly to all input behaviors", () => {
59   test('should allow input on ".add-item"', () => {
60     const addItem = wrapper.find(".add-item");
61     addItem.simulate("change", { target: { value: "pear" } });
62     expect(addItem.instance().value).toEqual("pear");
63   });
64
65   test('should reject empty input on ".add-item"', () => {
66     const addItem = wrapper.find(".add-item");
67     addItem.simulate("change", { target: { value: "" } });
68     addItem.simulate("keydown", {
69       event: "keydown",
70       key: "Enter",
71       keyCode: 13,
72       which: 13
73     });
74     const items = wrapper.find(".items-list");
75     expect(items.children()).toHaveLength(0);
76   });
77
78   test('should add an item when the Enter key is pressed on ".add-item"', () => {
79     addAll(["pear"]);
80     expect(wrapper.find(".add-item").instance().value).toEqual("");
81     const items = wrapper.find(".items-list");
82     expect(items.exists()).toBe(true);
83     expect(items.children()).toHaveLength(1);
84     expect(items.children().at(0).text()).toEqual("pear");
85   });
86
87   test('should add a second item that comes before the first alphabetically', () => {
88     addAll(["pear", "banana"]);
89     const items = wrapper.find(".items-list");
90     expect(items.exists()).toBe(true);
91     expect(items.children()).toHaveLength(2);
92     expect(items.children().at(0).text()).toEqual("banana");
93     expect(items.children().at(1).text()).toEqual("pear");
94   });
95
96   test('should add a third item that comes between the existing items', () => {
97     addAll(["pear", "banana", "peach"]);
98     const items = wrapper.find(".items-list");
99     expect(items.exists()).toBe(true);
100    expect(items.children()).toHaveLength(3);
101    [
102      "banana",
103      "peach",
104      "pear",
105    ].forEach((e, i) =>
106      expect(items.children().at(i).text()).toEqual(e)
107    );

```

```

108 });
109
110 test('should add a fourth item that comes after the existing items', () => {
111   addAll(["pear", "banana", "peach", "watermelon"]);
112   const items = wrapper.find(".items-list");
113   expect(items.exists()).toBe(true);
114   expect(items.children()).toHaveLength(4);
115   [
116     "banana",
117     "peach",
118     "pear",
119     "watermelon"
120   ].forEach((e, i) =>
121     expect(items.children().at(i).text()).toEqual(e)
122   );
123 });
124 });
125
126 describe("responds to changes in the sort direction", () => {
127   test('should reverse all of the items currently in the list', () => {
128     addAll(["pear", "banana", "peach", "watermelon"]);
129     const dir = wrapper.find(".sort-direction");
130     dir.simulate("click");
131     const items = wrapper.find(".items-list");
132     expect(items.exists()).toBe(true);
133     expect(items.children()).toHaveLength(4);
134     [
135       "watermelon",
136       "pear",
137       "peach",
138       "banana",
139     ].forEach((e, i) =>
140       expect(items.children().at(i).text()).toEqual(e)
141     );
142   });
143 });
144
145 test('should show a different button string when ordered descending', () => {
146   addAll(["pear", "banana", "peach", "watermelon"]);
147   const dir = wrapper.find(".sort-direction");
148   const asc = dir.text();
149   dir.simulate("click");
150   expect(dir.text()).not.toEqual(asc);
151   expect(dir.text()).not.toEqual("");
152   dir.simulate("click");
153   expect(dir.text()).toEqual(asc);
154 });
155
156 test('should still add items correctly when ordered descending', () => {
157   addAll(["pear", "banana", "peach", "watermelon"]);
158   wrapper.find(".sort-direction").simulate("click");
159   addAll(["nectarine"]);
160   const items = wrapper.find(".items-list");
161   expect(items.exists()).toBe(true);
162   expect(items.children()).toHaveLength(5);

```

```

162     [
163         "watermelon",
164         "pear",
165         "peach",
166         "nectarine",
167         "banana"
168     ].forEach((e, i) =>
169         expect(items.children().at(i).text()).toEqual(e)
170     );
171 });
172
173 test('should reverse all of the items currently in the list back to ascending', () => {
174     addAll(["pear", "banana", "peach", "watermelon", "nectarine"]);
175     const dir = wrapper.find(".sort-direction");
176     dir.simulate("click");
177     dir.simulate("click");
178     const items = wrapper.find(".items-list");
179     expect(items.exists()).toBe(true);
180     expect(items.children()).toHaveLength(5);
181     [
182         "banana",
183         "nectarine",
184         "peach",
185         "pear",
186         "watermelon",
187     ].forEach((e, i) =>
188         expect(items.children().at(i).text()).toEqual(e)
189     );
190 });
191
192 test('should still add items correctly when re-ordered ascending', () => {
193     addAll(["pear", "banana", "watermelon", "nectarine"]);
194     const dir = wrapper.find(".sort-direction");
195     dir.simulate("click");
196     addAll(["peach"]);
197     dir.simulate("click");
198     const addItem = wrapper.find(".add-item");
199     addItem.simulate("change", { target: { value: "plum" } });
200     expect(addItem.instance().value).toEqual("plum");
201     addItem.simulate("keydown", {
202         event: "keydown",
203         key: "Enter",
204         keyCode: 13,
205         which: 13
206     });
207     expect(addItem.instance().value).toEqual("");
208     const items = wrapper.find(".items-list");
209     expect(items.exists()).toBe(true);
210     expect(items.children()).toHaveLength(6);
211     [
212         "banana",
213         "nectarine",
214         "peach",
215         "pear",

```

```

216     "plum",
217     "watermelon",
218   ].forEach((e, i) =>
219     expect(items.children().at(i).text()).toEqual(e)
220   );
221 });
222 });
223
224 describe("clear button works", () => {
225   test('should clear all of the items currently in the list as well as any value in ".add-
item"', () => {
226     const addItem = wrapper.find(".add-item");
227     addItem.simulate("change", { target: { value: "grape" } });
228     expect(addItem.instance().value).toEqual("grape");
229     const clear = wrapper.find(".clear-list");
230     clear.simulate("click");
231     const items = wrapper.find(".items-list");
232     expect(items.exists()).toBe(true);
233     expect(items.children()).toHaveLength(0);
234     expect(addItem.instance().value).toEqual("");
235   });
236 });
237 });
238
239

```

package.json

```

1  {
2    "name": "react-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/App.jsx",
6    "dependencies": {
7      "escape-html": "1.0.3",
8      "react": "16.8.3",
9      "react-dom": "16.8.3",
10     "react-scripts": "2.0.3",
11     "axios": "0.18.0",
12     "mobx": "5.9.4",
13     "mobx-react": "5.4.3",
14     "prop-types": "15.7.2",
15     "recompose": "0.30.0",
16     "redux": "4.0.1",
17     "@babel/runtime": "7.4.4"
18   },
19   "devDependencies": {
20     "sass": "1.20.1",
21     "sinon": "7.3.1"
22   },
23   "scripts": {
24     "start": "react-scripts start",

```



```
25     "build": "react-scripts build",
26     "test": "react-scripts test --env=jsdom",
27     "eject": "react-scripts eject"
28   }
29 }
```

docs/environment.md

```
1  # Execution Environment
2  Two separate environments exist for this challenge. The test environment, which is remotely
3  executed within a NodeJS runtime, and the web preview environment, which is bundled using
4  Webpack. The `package.json` does not control both environments.
5
6  ## Test Environment
7  The code for this project is tested using Node 10, with Jest as the test framework. JSX is
8  supported, as well as ES6 style imports. The code is executed in a remote environment, not
9  within your browser. Any `console.log` statements executed while running tests will not send
10 logs to your browser, but the output will be shown at the bottom of your tests. Jest does not
11 support inline `console.log` statements. This is why logs come after all test output.
12
13 Any package that is already loaded into the runner environment is able to be used; there is no
14 `package.json` file that manages what is executed within tests. Candidates are not able to
15 import packages that are not already installed.
16
17 Stylesheets and other assets can be imported. See `jest.config.js` and the `__mocks__` directory
18 to see what is loaded in their place. By default, these imports have no impact on tests, only
19 the web preview environment.
20
21 ## Web Preview Environment
22 The web preview environment uses Sandpack to bundle assets. The `package.json` file is used to
23 determine what gets loaded into this environment. There are no special config files; everything
24 is inferred from package.json.
25
26 Customization is possible, but be careful not to use different versions within the web preview
27 environment than what is loaded within the test environment. In some cases you can load
28 additional dependencies if it is only cosmetic (styling) or meant for building preview assets,
29 but those dependencies may not be available within the runner environment and can cause issues
30 with your tests.
```

src/OrderedList.jsx

```
1  import React from "react";
2
3  export default class OrderedList extends React.Component {
4    constructor(props) {
5      super(props);
6
7      this.state = {
8        items: [],
9        inputValue: '',
```

```

10     direction: 'DESC'
11   }
12 }
13
14 sortItems = (items) => {
15   const { direction } = this.state
16   let ordered = items.sort()
17   if (direction === 'ASC') {
18     ordered = items.reverse()
19   }
20
21   return ordered;
22 }
23
24 onInputKeyDown = (e) => {
25   const { value } = e.target;
26   if (e.keyCode === 13 && value.length) {
27     this.setState({
28       items: [...this.state.items, value],
29       inputValue: ''
30     })
31   }
32 }
33
34 onInputChange = (e) => {
35   this.setState({
36     inputValue: e.target.value
37   })
38 }
39
40 onClearList = () => {
41   this.setState({
42     items: [],
43     inputValue: ''
44   })
45 }
46
47 onSortToggle = () => {
48   const { direction } = this.state;
49   this.setState({
50     direction: direction !== 'DESC' ? 'DESC' : 'ASC'
51   })
52 }
53
54 render() {
55   const { inputValue, items, direction } = this.state;
56   return (
57     <>
58     <input onChange={this.onInputChange} onKeyDown={this.onInputKeyDown} value={inputValue}
59     className="add-item"/>
60     <button onClick={this.onSortToggle} className="sort-direction">{direction === 'DESC' ?
61     '↓' : '↑' }</button>
62     <button onClick={this.onClearList} className="clear-list">☐</button>

```

```
62     <ul className="items-list">
63         {this.sortItems(items).map(item => (
64             <li>{item}</li>
65         ))}
66     </ul>
67 </>
68 );
69 }
70 }
71
```

__tests__

1