Daniel W.

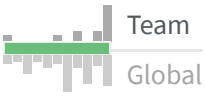
React: Fundamentals Results

Interact

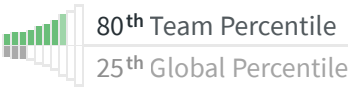
Interviewer: Dev Eval

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%	Team	5 m, 40 s
#2: React: Turbo Click Counter	100%	Team	10 m, 40 s
#3: React: Unique List	100%	Team	37 m, 35 s
#4: React: Ordered List	100%	Team	18 m, 24 s

#1: React: Click Counter

Scoring

100%

Team

16 / 16 Tests (1 Attempt)

Timing

5 m, 40 s Active Time

90th Team Percentile

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:

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

```jsx
import React from "react";

class Counter extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      counter: 0
    }
  }

  onIncrementClick = () => {
    this.setState({
      counter: this.state.counter + 1
    })
  }

  onDecrementCount = () => {
    this.setState({
      counter: this.state.counter - 1
    })
  }

  render() {
    const { counter } = this.state;

    return (
      <div>
        <h1 className="counter">{counter}</h1>
        <button type="button" onClick={this.onDecrementCount} className="decrement">
          Decrement
        </button>
        <button type="button" onClick={this.onIncrementClick} className="increment">
          Increment
        </button>
      </div>
    );
  }
}

export default Counter;

```

docs/environment.md

```md
# Execution Environment
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.

## Test Environment
```

```
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.

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.

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.

## Web Preview Environment
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.

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.

```

JSX __tests__/Counter.test.jsx

```jsx
import React from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import Counter from "../src/Counter";

describe("Counter", () => {
  test("has an element with a 'counter' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".counter")).toBe(true);
  });

  test("has an element with an 'increment' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".increment")).toBe(true);
  });

  test("has an element with a 'decrement' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".decrement")).toBe(true);
  });

  test("shows inital value", () => {
```

```
    const wrapper = mount(<Counter />);
    expect(wrapper.find('.counter').text()).toEqual("0");
  });

  it("increments counter by 1 when increment button is clicked", () => {
    const wrapper = mount(<Counter />);
    wrapper.find(".increment").simulate("click");
    expect(wrapper.find(".counter").text()).toEqual("1");
  });

  it("decrements counter by 1 when decrement button is clicked", () => {
    const wrapper = mount(<Counter />);
    wrapper.find(".decrement").simulate("click");
    expect(wrapper.find(".counter").text()).toEqual("-1");
  });
});
```

__tests__/Submission.test.jsx

```
import React from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import Counter from "../src/Counter";

describe("Counter", () => {
  test("has an element with a 'counter' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".counter")).toBe(true);
  });

  test("has an element with an 'increment' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".increment")).toBe(true);
  });

  test("has an element with a 'decrement' class", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.exists(".decrement")).toBe(true);
  });

  test("shows inital value of 0", () => {
    const wrapper = mount(<Counter />);
    expect(wrapper.find('.counter').text()).toEqual("0");
  });

  it("increments counter by 1 when increment button is clicked", () => {
    const wrapper = mount(<Counter />);
    wrapper.find(".increment").simulate("click");
    expect(wrapper.find(".counter").text()).toEqual("1");
  });
```

```
  it("decrements counter by 1 when decrement button is clicked", () => {
    const wrapper = mount(<Counter />);
    wrapper.find(".decrement").simulate("click");
    expect(wrapper.find(".counter").text()).toEqual("-1");
  });

  for (let n = 1; n <= 10; n++) {
    it(`works on a random increment/decrement test (${n} of 10)`, () => {
      const wrapper = mount(<Counter />);
      const decCount = Math.floor(Math.random() * 50)
      const incCount = Math.floor(Math.random() * 50)

      for (let i = 0; i < decCount; i++) {
        wrapper.find(".decrement").simulate("click");
      }

      for (let i = 0; i < incCount; i++) {
        wrapper.find(".increment").simulate("click");
      }

      expect(wrapper.find(".counter").text()).toEqual(incCount - decCount + "");
    });
  }
});
```

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Head content will be replaced by WebPack -->
  </head>
  <body>
    <!-- include any external stylesheets within the body instead
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
    />
    -->

    <!-- Bootstrap -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
crossorigin="anonymous">

    <div id="root"></div>
  </body>
</html>

```

src/styles.scss

```scss
#app {
  margin: 2em;
}

```

__mocks__/fileMock.js

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

src/App.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom";
import Counter from "./Counter";

function App(props) {
  return (
    <div id="app">
      <Counter />
    </div>
  );
}

export default App;
```

package.json

```json
{
  "name": "react-app",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.jsx",
  "dependencies": {
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-scripts": "2.0.3",
    "axios":"0.18.0",
    "mobx":"5.9.4",
    "mobx-react":"5.4.3",
    "prop-types":"15.7.2",
    "recompose":"0.30.0",
    "redux":"4.0.1",
    "@babel/runtime": "7.4.4"
  },
  "devDependencies": {
    "sass": "1.20.1",
    "sinon": "7.3.1"
```

```
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

src/index.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom";
import "./styles.scss";
import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

__mocks__/styleMock.js

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

jest.config.js

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

```

#2: React: Turbo Click Counter

Scoring

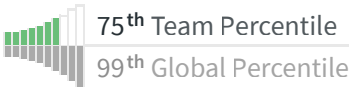
100%

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

```jsx
import React from "react";

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {counter: 0};
    this.incrementCount = this.incrementCount.bind(this);
    this.decrementCount = this.decrementCount.bind(this);
  }

  incrementCount() {
    this.setState(state => ({counter: state.counter + this.props.stepSize}));
  }

  decrementCount() {
    this.setState(state => ({counter: state.counter - this.props.stepSize}));
  }

  render() {
    return (
      <div>
        <h1 className="counter">{this.state.counter}</h1>
        <button
          type="button"
          className="increment"
```

```
          onClick={this.incrementCount}
        >Increment</button>
        <button
          type="button"
          className="decrement"
          onClick={this.decrementCount}
        >Decrement</button>
      </div>
    );
  }
}

export default Counter;

```

docs/environment.md

```
# Execution Environment
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.

## Test Environment
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.

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.

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.

## Web Preview Environment
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.

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.

```

__tests__/TurboCounter.test.jsx

```
import React from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import TurboCounter from "../src/TurboCounter";

let wrapper;

describe("TurboCounter", () => {
  describe("basic counter functionality", () => {
    beforeEach(() => {
      wrapper = mount(<TurboCounter />);
    });

    test("has an element with a '.counter' class", () => {
      expect(wrapper.exists(".counter")).toBe(true);
    });

    test("has an element with an '.increment' class", () => {
      expect(wrapper.exists(".increment")).toBe(true);
    });

    test("has an element with a '.decrement' class", () => {
      expect(wrapper.exists(".decrement")).toBe(true);
    });

    test("'.counter' shows an inital value of 0", () => {
      expect(wrapper.find('.counter').text()).toEqual("0");
    });

    test("increments counter by 1 when increment button is clicked", () => {
      wrapper.find(".increment").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("1");
    });

    test("decrements counter by 1 when decrement button is clicked", () => {
      wrapper.find(".decrement").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("-1");
    });
  });

  describe("step size controller functionality", () => {
    beforeEach(() => {
      wrapper = mount(<TurboCounter />);
    });

    test("has an element with a '.step-size' class", () => {
      expect(wrapper.exists(".step-size")).toBe(true);
    });

    test("has an element with an '.increment-step-size' class", () => {
      expect(wrapper.exists(".increment-step-size")).toBe(true);
```

```
    });

    test("has an element with a '.decrement-step-size' class", () => {
      expect(wrapper.exists(".decrement-step-size")).toBe(true);
    });

    test("'.step-size' shows an inital value of 1", () => {
      expect(wrapper.find('.step-size').text()).toEqual("1");
    });

    test("increments step size by 1 when '.increment-step-size' button is clicked", () => {
      wrapper.find(".increment-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("2");
    });

    test("decrements step size by 1 when '.decrement-step-size' button is clicked", () => {
      wrapper.find(".increment-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("2");
      wrapper.find(".decrement-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("1");
    });

    test("doesn't go below 1", () => {
      wrapper.find(".decrement-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("1");
    });

    test("increment works repeatedly, increasing the value to 5", () => {
      for (let i = 2; i < 6; i++) {
        wrapper.find(".increment-step-size").simulate("click");
        expect(wrapper.find(".step-size").text()).toEqual("" + i);
      }
    });
  });

  describe("step size and counter operate together", () => {
    beforeEach(() => {
      wrapper = mount(<TurboCounter />);
    });

    test("increments counter by 5 when increment button is clicked", () => {
      for (let i = 0; i < 4; i++) {
        wrapper.find(".increment-step-size").simulate("click");
      }

      expect(wrapper.find(".counter").text()).toEqual("0");
      wrapper.find(".increment").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("5");
    });

    test("decrements counter by 5 when decrement button is clicked", () => {
      for (let i = 0; i < 4; i++) {
        wrapper.find(".increment-step-size").simulate("click");
      }
```

```
      wrapper.find(".decrement").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("-5");
    });

    test("should work on multiple operations", () => {
      wrapper.find(".increment-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("2");
      wrapper.find(".increment").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("2");
      wrapper.find(".increment-step-size").simulate("click");
      expect(wrapper.find(".step-size").text()).toEqual("3");
      wrapper.find(".increment").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("5");
      wrapper.find(".decrement").simulate("click");
      expect(wrapper.find(".counter").text()).toEqual("2");
    });
  });
});
```

__tests__/Submission.test.jsx

```jsx
import React, { useState } from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import TurboCounter from "../src/TurboCounter";

let wrapper;
let wrapperRef;

describe("TurboCounter", () => {
  beforeEach(() => {
    wrapper = mount(<TurboCounter />);
    wrapperRef = mount(<TurboCounterRef />);
  });

  test("random test", () => {
    for (let i = 0; i < 50; i++) {
      const action = Math.random() * 4;

      if (action > 3) {
        wrapper.find(".increment-step-size").simulate("click");
        wrapperRef.find(".increment-step-size").simulate("click");
      }
      else if (action > 2) {
        wrapper.find(".decrement-step-size").simulate("click");
        wrapperRef.find(".decrement-step-size").simulate("click");
      }
      else if (action > 1) {
        wrapper.find(".increment").simulate("click");
```

```
        wrapperRef.find(".increment").simulate("click");
      }
      else {
        wrapper.find(".decrement").simulate("click");
        wrapperRef.find(".decrement").simulate("click");
      }

      const refStep = wrapperRef.find(".step-size").text();
      const refCounter = wrapperRef.find(".counter").text();
      expect(wrapper.find(".step-size").text()).toEqual(refStep);
      expect(wrapper.find(".counter").text()).toEqual(refCounter);
    }
  });
});

const CounterRef = ({stepSize}) => {
  const [count, setCount] = useState(0);

  return (
    <div className="cnt">
      <h1 className="counter">{count}</h1>
      <div>
        <button
          type="button"
          className="increment"
          onClick={() => setCount(count + stepSize)}
        >Increment</button>
      </div>
      <div>
        <button
          type="button"
          className="decrement"
          onClick={() => setCount(count - stepSize)}
        >Decrement</button>
      </div>
    </div>
  );
};

const IncrementCounterRef = ({stepSize, onInc, onDec}) =>
  <div className="inc">
    <h3 className="step-size">{stepSize}</h3>
    <div>
      <button
        type="button"
        className="increment-step-size"
        onClick={onInc}
      >Step size +</button>
    </div>
    <div>
      <button
        type="button"
        className="decrement-step-size"
        onClick={onDec}
```

```
      >Step size -</button>
    </div>
  </div>
;

const TurboCounterRef = () => {
  const [stepSize, setStepSize] = useState(1);

  return (
    <div className="turbo-counter">
      <CounterRef stepSize={stepSize} />
      <IncrementCounterRef
        stepSize={stepSize}
        onInc={() => setStepSize(stepSize + 1)}
        onDec={() => setStepSize(Math.max(1, stepSize - 1))}
      />
    </div>
  );
};
```

src/styles.scss

```
#app {
  margin: 2em;
}

```

src/App.jsx

```
import React from "react";
import ReactDOM from "react-dom";
import TurboCounter from "./TurboCounter";

function App(props) {
  return (
    <div id="app">
      <TurboCounter />
    </div>
  );
}

export default App;
```

__mocks__/fileMock.js

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

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Head content will be replaced by WebPack -->
  </head>
  <body>
    <!-- include any external stylesheets within the body instead
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
    />
    -->

    <!-- Bootstrap -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
crossorigin="anonymous">

    <div id="root"></div>
  </body>
</html>

```

src/index.jsx

```
import React from "react";
import ReactDOM from "react-dom";
import "./styles.scss";
import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

__mocks__/styleMock.js

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

jest.config.js

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

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

```

src/TurboCounter.jsx

```jsx
import React from "react";
import Counter from "./Counter";

class TurboCounter extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      stepSize: 1
    }

    this.incrementStepSize = this.incrementStepSize.bind(this);
    this.decrementStepSize = this.decrementStepSize.bind(this);
  }

  incrementStepSize() {
    this.setState(state => ({stepSize: state.stepSize + 1}));
  }

  decrementStepSize() {
    this.setState(state => ({stepSize: Math.max(state.stepSize - 1, 1)}));
  }

  render() {
     const { stepSize } = this.state;

    return (
      <>
        <Counter stepSize={stepSize} />
        <h2 className="step-size">{stepSize}</h2>
        <button
          type="button"
          className="increment-step-size"
          onClick={this.incrementStepSize}
        >Step size +</button>
        <button
          type="button"
          className="decrement-step-size"
          onClick={this.decrementStepSize}
        >Step size -</button>
      </>
    );
```

```
  }
}

export default TurboCounter;
```

package.json

```
{
  "name": "react-app",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.jsx",
  "dependencies": {
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-scripts": "2.0.3",
    "axios":"0.18.0",
    "mobx":"5.9.4",
    "mobx-react":"5.4.3",
    "prop-types":"15.7.2",
    "recompose":"0.30.0",
    "redux":"4.0.1",
    "@babel/runtime": "7.4.4"
  },
  "devDependencies": {
    "sass": "1.20.1",
    "sinon": "7.3.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

#3: React: Unique List

Scoring

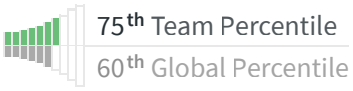
100%

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>
  <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 `<li>` 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.

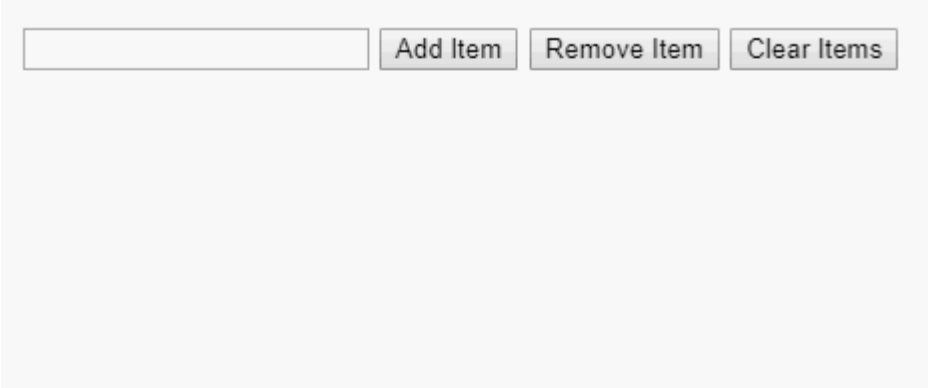
Conversely, if any action was successful in updating DOM state, clear the input element's text content.

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

```
# Execution Environment
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.

## Test Environment
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.

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.

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.

## Web Preview Environment
```

```
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.

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.

```

src/App.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom";
import "./style.scss";
import UniqueList from "./UniqueList";

/* Changes made to this file will not affect your tests.
 * This file is used to control the behavior of the web preview.
*/
function App() {
  return (
    <div className="App">
      <UniqueList />
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById("root"));
```

src/style.scss

```scss
html, body, #root {
  height: 100%;
}

body {
  background-color: #f8f8f8;
}

.App {
  font-family: sans-serif;
  padding: 1em;
  height: 100%;
}

.items {
  list-style: none;
  marginTop: 1rem;
  padding: 0;
```

```
  li {
    padding: 1rem 0;
    border-bottom: 1px solid grey;
    font-size: 14px;

    &:last-child {
      border-bottom: 0;
    }
  }
}

.toolbar {
  display: flex;
  flex-direction: row;

  > input[type="text"] {
    flex: 1;
    padding: .5rem;
    margin-right: .5rem;
  }

  > input[type="button"] {
    flex: 0;
    margin-right: .5rem;
  }
}
```

package.json

```
{
  "name": "react-app",
  "version": "1.0.0",
  "description": "",
  "main": "src/App.jsx",
  "dependencies": {
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-scripts": "2.0.3",
    "axios":"0.18.0",
    "mobx":"5.9.4",
    "mobx-react":"5.4.3",
    "prop-types":"15.7.2",
    "recompose":"0.30.0",
    "redux":"4.0.1",
    "@babel/runtime": "7.4.4"
  },
  "devDependencies": {
    "sass": "1.20.1",
    "sinon": "7.3.1"
  },
  "scripts": {
    "start": "react-scripts start",
```

```
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

src/UniqueList.jsx

```jsx
import React from "react";

class UniqueList extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      inputValue: '',
      items: []
    }
  }

  onInputChange = (e) => {
    this.setState({
      inputValue: e.target.value
    })
  }

  checkValue = (value) => {
    return value.trim().length > 0
  }

  checkUnique = (value) => {
    return !this.state.items.find(item => item === value.trim())
  }

  addItem = () => {
    const { inputValue, items } = this.state;

    if (this.checkValue(inputValue) && this.checkUnique(inputValue)) {
      this.setState({
        items: [...items, inputValue.trim()],
        inputValue: ''
      })
    } else {
      // needs some error handling
    }
  }

  onAddItemClick = () => {
    this.addItem();
  }

  onClearItemsClick = () => {
```

```
    const { items, inputValue } = this.state;

    this.setState({
      items: [],
      inputValue: items.length ? '' : inputValue
    })
  }

  onRemoveItemClick = () => {
    const { inputValue, items } = this.state;
    const exists = !!items.find(item => item === inputValue.trim())

    if (this.checkValue(inputValue) && exists) {
      this.setState({
        items: items.filter(item => item !== inputValue.trim()),
        inputValue: ''
      })
    } else {
      // some error
    }
  }

  onInputKeyDown = (e) => {
    if (e.keyCode === 13) {
      this.addItem();
    }
  }

  render() {
    const { inputValue, items } = this.state;

    return (
      <>
        <div className="toolbar">
          <input
            className="item-input"
            type="text"
            value={inputValue}
            onChange={this.onInputChange}
            onKeyDown={this.onInputKeyDown}
          />
          <input
            className="add-button"
            type="button"
            value="Add Item"
            onClick={this.onAddItemClick}
          />
          <input
            className="remove-button"
            type="button"
            value="Remove Item"
            onClick={this.onRemoveItemClick}
          />
          <input
```

```
            className="clear-button"
            type="button"
            value="Clear Items"
            onClick={this.onClearItemsClick}
          />
        </div>
        <ul className="items">
          {items.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>
      </>
    );
  }
}

export default UniqueList;
```

__tests__/UniqueList.test.jsx

```
import React from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import UniqueList from "../src/UniqueList";
let ul;

describe("UniqueList Basic Functionality", () => {
  beforeEach(() => {
    ul = mount(<UniqueList />);
  });

  test("item input element exists", () => {
    expect(ul.find(".item-input").exists()).toBe(true);
    expect(ul.find(".item-input")).toHaveLength(1);
  });

  test("add item button element exists", () => {
    expect(ul.find(".add-button").exists()).toBe(true);
    expect(ul.find(".add-button")).toHaveLength(1);
  });

  test("remove item button element exists", () => {
    expect(ul.find(".remove-button").exists()).toBe(true);
    expect(ul.find(".remove-button")).toHaveLength(1);
  });

  test("clear item button element exists", () => {
    expect(ul.find(".clear-button").exists()).toBe(true);
    expect(ul.find(".clear-button")).toHaveLength(1);
  });
```

```
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",
```

```
      which: 13,
      keyCode: 13,
    });
    expect(items.text()).toEqual("apple");
  });

  test("item is removed after clicking Remove Item button", () => {
    const input = ul.find(".item-input");
    const remove = ul.find(".remove-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "apple" } });
    remove.simulate("click");
    expect(items.text()).toEqual("");
    expect(input.instance().value).toEqual("");
  });

  test("list is cleared after clicking the Clear Items button", () => {
    const input = ul.find(".item-input");
    const clear = ul.find(".clear-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "banana" } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(items.text()).toEqual("applebanana");
    input.simulate("change", { target: { value: "orange" } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(items.text()).toEqual("applebananaorange");
    clear.simulate("click");
    expect(items.text()).toEqual("");
```

```
    expect(input.instance().value).toEqual("");
  });
});

describe("UniqueList Advanced Functionality", () => {
  beforeEach(() => {
    ul = mount(<UniqueList />);
  });

  test("input is not cleared attempting to clear an empty list", () => {
    const input = ul.find(".item-input");
    const clear = ul.find(".clear-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    clear.simulate("click");
    expect(input.instance().value).toEqual("apple");
    expect(items.text()).toEqual("");
  });

  test("input is cleared attempting to clear a list with items in it", () => {
    const input = ul.find(".item-input");
    const clear = ul.find(".clear-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "banana" } });
    expect(input.instance().value).toEqual("banana");
    clear.simulate("click");
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("");
  });

  test("input is not cleared attempting to remove something not in the list", () => {
    const input = ul.find(".item-input");
    const remove = ul.find(".remove-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    remove.simulate("click");
    expect(input.instance().value).toEqual("apple");
    expect(items.text()).toEqual("");
    input.simulate("change", { target: { value: "apple" } });
    expect(input.instance().value).toEqual("apple");
    expect(items.text()).toEqual("");
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
```

```
      keyCode: 13,
    });
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "apples" } });
    remove.simulate("click");
    expect(input.instance().value).toEqual("apples");
    expect(items.text()).toEqual("apple");
  });

  test("input is not cleared attempting to add something already in the list", () => {
    const input = ul.find(".item-input");
    const add = ul.find(".add-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "apple" } });
    add.simulate("click");
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "apple" } });
    add.simulate("click");
    expect(input.instance().value).toEqual("apple");
    expect(items.text()).toEqual("apple");
  });

  test("input is trimmed before all operations", () => {
    const input = ul.find(".item-input");
    const remove = ul.find(".remove-button");
    const items = ul.find(".items");
    input.simulate("change", { target: { value: "   apple   " } });
    input.simulate("keydown", {
      key: "Enter",
      type: "keydown",
      which: 13,
      keyCode: 13,
    });
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("apple");
    input.simulate("change", { target: { value: "  apple  " } });
    remove.simulate("click");
    expect(input.instance().value).toEqual("");
    expect(items.text()).toEqual("");
  });
});
```

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Head content will be replaced by WebPack -->
  </head>
  <body>
```

```
    <!-- include any external stylesheets within the body instead
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
    />
    -->
    <div id="root"></div>
  </body>
</html>

```

public

```

```

__tests__

```

```

src

```

```

#4: React: Ordered List

Scoring

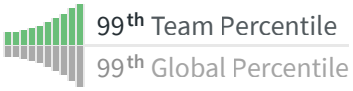
100%

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:

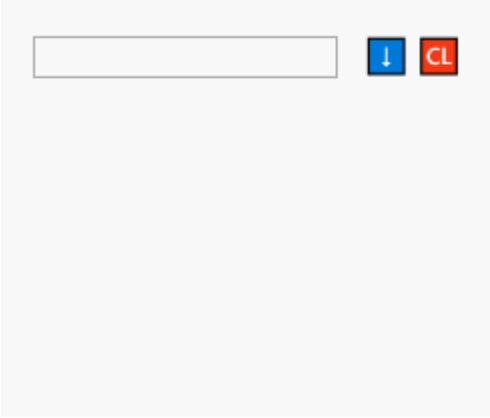
- An `<input class="add-item" />` element which the user can use to add items to the list. This input field should listen for `Enter` keydown events (the test suite triggers the `onKeyDown` event handler specifically) to add the current contents (if nonempty) to the list. After adding an item, the input 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 `<li>` 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/) and ReactJS (https://reactjs.org/docs/) if you get stuck.

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

Solution Code

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Head content will be replaced by WebPack -->
  </head>
  <body>
    <!-- include any external stylesheets within the body instead
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.css"
    />
    -->
    <div id="root"></div>
  </body>
</html>

```

src/App.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom";
import "./style.scss";
import OrderedList from "./OrderedList";

/* Changes made to this file will not affect your tests.
 * This file is used to control the behavior of the web preview.
*/
function App() {
  return (
    <div className="App">
      <OrderedList />
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById("root"));
```

src/style.scss

```scss
html, body, #root {
  height: 100%;
}

body {
  background-color: #f8f8f8;
}

.App {
  font-family: sans-serif;
  margin: 2em;
  height: 100%;
}
```

public

src

__tests__/OrderedList.test.jsx

```
import React from "react";
import Enzyme, { mount, shallow } from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({ adapter: new Adapter() });
import sinon from "sinon";
import OrderedList from "../src/OrderedList";

let wrapper;
let asc;
let desc;

const addAll = items => {
  items.forEach(e => {
    const addItem = wrapper.find(".add-item");
    addItem.simulate("change", { target: { value: e } });
    addItem.simulate("keydown", {
      event: "keydown",
      key: "Enter",
      keyCode: 13,
      which: 13
    });
  });
};

describe("OrderedList", () => {
  beforeEach(() => {
    wrapper = mount(<OrderedList />);
  });
  afterEach(() => wrapper.unmount());

  describe("renders correct elements", () => {
    test('should have a ".add-item" input', () => {
      const addItem = wrapper.find(".add-item");
      expect(addItem.exists()).toBe(true);
      expect(addItem).toHaveLength(1);
    });

    test('should have a ".sort-direction" button with some text content', () => {
      const sortDir = wrapper.find(".sort-direction");
      expect(sortDir.exists()).toBe(true);
      expect(sortDir).toHaveLength(1);
      expect(sortDir.text()).not.toBe("");
    });

    test('should have a ".clear-list" button', () => {
      const clear = wrapper.find(".clear-list");
      expect(clear.exists()).toBe(true);
      expect(clear).toHaveLength(1);
    });

    test('should have an ".items-list" element', () => {
      const items = wrapper.find('.items-list');
      expect(items.exists()).toBe(true);
```

```
      expect(items).toHaveLength(1);
    });
  });

  describe("responds correctly to all input behaviors", () => {
    test('should allow input on ".add-item"', () => {
      const addItem = wrapper.find(".add-item");
      addItem.simulate("change", { target: { value: "pear" } });
      expect(addItem.instance().value).toEqual("pear");
    });

    test('should reject empty input on ".add-item"', () => {
      const addItem = wrapper.find(".add-item");
      addItem.simulate("change", { target: { value: "" } });
      addItem.simulate("keydown", {
        event: "keydown",
        key: "Enter",
        keyCode: 13,
        which: 13
      });
      const items = wrapper.find(".items-list");
      expect(items.children()).toHaveLength(0);
    });

    test('should add an item when the Enter key is pressed on ".add-item"', () => {
      addAll(["pear"]);
      expect(wrapper.find(".add-item").instance().value).toEqual("");
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(1);
      expect(items.children().at(0).text()).toEqual("pear");
    });

    test('should add a second item that comes before the first alphabetically', () => {
      addAll(["pear", "banana"]);
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(2);
      expect(items.children().at(0).text()).toEqual("banana");
      expect(items.children().at(1).text()).toEqual("pear");
    });

    test('should add a third item that comes between the existing items', () => {
      addAll(["pear", "banana", "peach"]);
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(3);
      [
        "banana",
        "peach",
        "pear",
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
```

```
    });

    test('should add a fourth item that comes after the existing items', () => {
      addAll(["pear", "banana", "peach", "watermelon"]);
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(4);
      [
        "banana",
        "peach",
        "pear",
        "watermelon"
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
    });
  });

  describe("responds to changes in the sort direction", () => {
    test('should reverse all of the items currently in the list', () => {
      addAll(["pear", "banana", "peach", "watermelon"]);
      const dir = wrapper.find(".sort-direction");
      dir.simulate("click");
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(4);
      [
        "watermelon",
        "pear",
        "peach",
        "banana",
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
    });

    test('should show a different button string when ordered descending', () => {
      addAll(["pear", "banana", "peach", "watermelon"]);
      const dir = wrapper.find(".sort-direction");
      const asc = dir.text();
      dir.simulate("click");
      expect(dir.text()).not.toEqual(asc);
      expect(dir.text()).not.toEqual("");
      dir.simulate("click");
      expect(dir.text()).toEqual(asc);
    });

    test('should still add items correctly when ordered descending', () => {
      addAll(["pear", "banana", "peach", "watermelon"]);
      wrapper.find(".sort-direction").simulate("click");
      addAll(["nectarine"]);
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(5);
```

```
      [
        "watermelon",
        "pear",
        "peach",
        "nectarine",
        "banana"
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
    });

    test('should reverse all of the items currently in the list back to ascending', () => {
      addAll(["pear", "banana", "peach", "watermelon", "nectarine"]);
      const dir = wrapper.find(".sort-direction");
      dir.simulate("click");
      dir.simulate("click");
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(5);
      [
        "banana",
        "nectarine",
        "peach",
        "pear",
        "watermelon",
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
    });

    test('should still add items correctly when re-ordered ascending', () => {
      addAll(["pear", "banana", "watermelon", "nectarine"]);
      const dir = wrapper.find(".sort-direction");
      dir.simulate("click");
      addAll(["peach"]);
      dir.simulate("click");
      const addItem = wrapper.find(".add-item");
      addItem.simulate("change", { target: { value: "plum" } });
      expect(addItem.instance().value).toEqual("plum");
      addItem.simulate("keydown", {
        event: "keydown",
        key: "Enter",
        keyCode: 13,
        which: 13
      });
      expect(addItem.instance().value).toEqual("");
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(6);
      [
        "banana",
        "nectarine",
        "peach",
        "pear",
```

```
        "plum",
        "watermelon",
      ].forEach((e, i) =>
        expect(items.children().at(i).text()).toEqual(e)
      );
    });
  });

  describe("clear button works", () => {
    test('should clear all of the items currently in the list as well as any value in ".add-item"', () => {
      const addItem = wrapper.find(".add-item");
      addItem.simulate("change", { target: { value: "grape" } });
      expect(addItem.instance().value).toEqual("grape");
      const clear = wrapper.find(".clear-list");
      clear.simulate("click");
      const items = wrapper.find(".items-list");
      expect(items.exists()).toBe(true);
      expect(items.children()).toHaveLength(0);
      expect(addItem.instance().value).toEqual("");
    });
  });
});
```

package.json

```
{
  "name": "react-app",
  "version": "1.0.0",
  "description": "",
  "main": "src/App.jsx",
  "dependencies": {
    "escape-html": "1.0.3",
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-scripts": "2.0.3",
    "axios":"0.18.0",
    "mobx":"5.9.4",
    "mobx-react":"5.4.3",
    "prop-types":"15.7.2",
    "recompose":"0.30.0",
    "redux":"4.0.1",
    "@babel/runtime": "7.4.4"
  },
  "devDependencies": {
    "sass": "1.20.1",
    "sinon": "7.3.1"
  },
  "scripts": {
    "start": "react-scripts start",
```

```
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

docs/environment.md

```
# Execution Environment
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.

## Test Environment
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.

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.

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.

## Web Preview Environment
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.

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.

```

src/OrderedList.jsx

```
import React from "react";

export default class OrderedList extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      items: [],
      inputValue: '',
```

```
      direction: 'DESC'
    }
  }

  sortItems = (items) => {
    const { direction } = this.state
    let ordered = items.sort()
    if (direction === 'ASC') {
      ordered = items.reverse()
    }

    return ordered;
  }

  onInputKeyDown = (e) => {
    const { value } = e.target;
    if (e.keyCode === 13 && value.length) {
       this.setState({
         items: [...this.state.items, value],
         inputValue: ''
       })
    }
  }

  onInputChange = (e) => {
    this.setState({
      inputValue: e.target.value
    })
  }

  onClearList = () => {
    this.setState({
      items: [],
      inputValue: ''
    })
  }

  onSortToggle = () => {
    const { direction } = this.state;
    this.setState({
      direction: direction !== 'DESC' ? 'DESC' : 'ASC'
    })
  }

  render() {
    const { inputValue, items, direction } = this.state;
    return (
      <>
        <input onChange={this.onInputChange} onKeyDown={this.onInputKeyDown} value={inputValue}
classNam="add-item"/>
        <button onClick={this.onSortToggle} className="sort-direction">{direction === 'DESC' ?
'↓' : '↑' }</button>
        <button onClick={this.onClearList} className="clear-list">CL</button>

```

```
        <ul className="items-list">
          {this.sortItems(items).map(item => (
            <li>{item}</li>
          ))}
        </ul>
      </>
    );
  }
}

```

__tests__

```

```