TASK

React IV: Managing State

Visit our website

Introduction

WELCOME TO THE REACT STATE TASK!

We have already briefly discussed using state in React. However, there is a lot more to learn about this topic. In this task, we will answer the following questions about states: How do we keep the state of two or more components synced? When should you use an external state store (like Redux or Flux) to handle state?

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A FEW REMINDERS ABOUT STATE MANAGEMENT WITH REACT

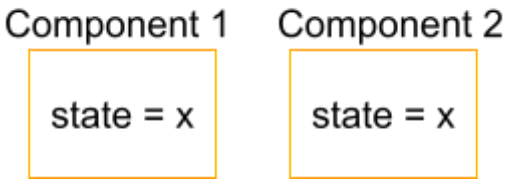
Remember that with React, you *use state when some data associated with a component changes over time*. For example, you would need to store information about whether a checkbox is checked or not in its state. In other words, if a component needs to change one of its attributes over time, that attribute should be part of its state.

Here are a few more important things you need to remember about state:

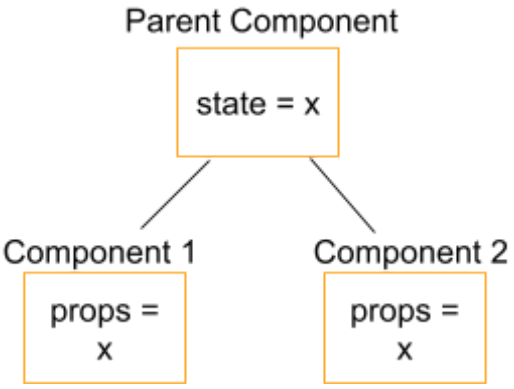
- Only components defined as classes can have state, i.e. components implemented using functions instead of a class cannot have state.
- A component can change its state (unlike props which cannot change).
- State is private to its component. State is not accessible to any component other than the one that owns and sets it. However, a component can choose to pass info stored in its state down to children using props.
- Have as few components that use state (stateful components) as possible in your application. Using too much state will make a program more complex and unpredictable.
- The state is user-defined, and it should be a plain JavaScript object.

KEEPING THE STATE OF TWO COMPONENTS SYNCED

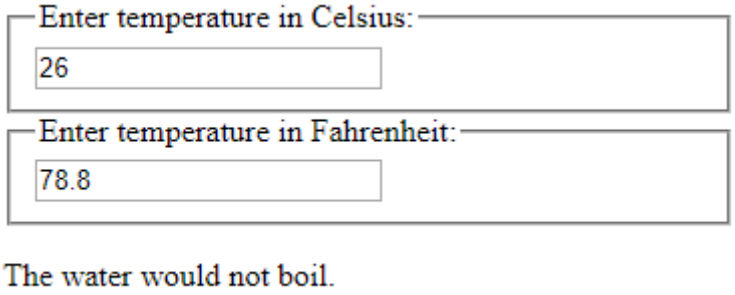
As you create more UIs, you are bound to want the state of two or more components to be synced at some stage. In other words, if the value in component 1 changes, you want the value in component 2 to change too.



In React, we accomplish this by *lifting state* up. This means that if two or more components need to have the same state at any given time, we store the state in the nearest shared parent component instead of in the separate children components. We then pass the value we want to keep synced down as props to the child components.



Let's look at an example of how this is done. In this example (provided by React), we create two components: a Calculator component and a TemperatureInput component. The Calculator component contains two TemperatureInput components (for inputting temperature in degrees Celsius or Fahrenheit) as shown below:



When a user enters a value in either of the two `TemperatureInput` components, the value in the other `TemperatureInput` component should automatically change. To implement this, follow these steps:

Step 1: Make the parent component a stateful component.

In this example, the Calculator component is the parent container since it contains the two TemperatureInput components. The Calculator component must, therefore, be a stateful component. You have already learnt how to create a stateful component.

The code for this example is shown below.

```
class Calculator extends React.Component {
   constructor(props) {
       super(props);
       this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
```

```
        this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
        this.state = {temperature: '', scale: 'c'};
    }

    handleCelsiusChange(temperature) {
        this.setState({scale: 'c', temperature});
    }

    handleFahrenheitChange(temperature) {
        this.setState({scale: 'f', temperature});
    }

    render() {
        const scale = this.state.scale;
        const temperature = this.state.temperature;
        const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) :
temperature;
        const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit)
: temperature;

        return (
            <div>
                <TemperatureInput
                    scale="c"
                    temperature={celsius}
                    onTemperatureChange={this.handleCelsiusChange} />

                <TemperatureInput
                    scale="f"
                    temperature={fahrenheit}
                    onTemperatureChange={this.handleFahrenheitChange} />

                <BoilingVerdict
                    celsius={parseFloat(celsius)} />

            </div>
        );
    }
}
```

Step 2: In the parent component, create and register event handlers that will respond to events triggered in the child components - i.e. the **`handle_Change()`** functions. These event handlers are used to change the state using **`this.setState()`**.

Step 3: In the children components (the **`TemperatureInput`** components), let the event handler that is triggered by the event that changes the state of the parent container respond by calling the event handler in the parent component.

```
class TemperatureInput extends React.Component {
   constructor(props) {
       super(props);
       this.handleChange = this.handleChange.bind(this);
   }

   handleChange(e) {
       this.props.onTemperatureChange(e.target.value);
   }

   render() {
       const temperature = this.props.temperature;
       const scale = this.props.scale;
       return (
           <fieldset>
               <legend>Enter temperature in {scaleNames[scale]}:</legend>
               <input value={temperature}
                      onChange={this.handleChange} />
           </fieldset>
       );
   }
}
```

Here, `onTemperatureChange` is both a function and a property (prop). It is a function from the parent passed to the child as a property.

This is an important concept and we recommend that you take the time to work through **this official React guide** to this example explained in more detail.

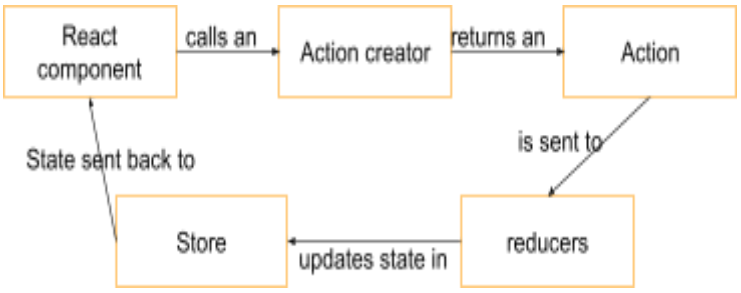
REDUX

Redux is a state container that can be used with React (and other view libraries) to assist with state management. The maker of Redux, Dan Abramov, recommends that you create React apps without Redux where possible and only use Redux when absolutely necessary. You can solve most frontend problems without Redux, and therefore you are not required to use it. If you are interested, read on to see how it works.

How does Redux work?

As stated earlier, Redux is a state container. It stores all state information for an app in one central place. The state information is stored in the store and the entire state of the application is stored in a single object known as the state tree.

We will now explain how state information is managed with Redux. A React component can call an *action creator* which creates an *action*. An action is an object that describes the change that is required. The action is sent to a *reducer*, which is a function that takes the previous state and the change (described in the action) and returns the next state. This is updated in the store and sent to the React component. When the React component's state is changed, it re-renders.



To learn more about Redux, you can do this **free tutorial** by the creator of Redux or visit the official Redux page **here** to see how to get started.

SPOT CHECK 1
Let's see what you can remember from this section.

1. What does it mean to "lift state up"?
2. What are the steps to "lift state up"?

Instructions

- Copy the 'example' directory that accompanies this task to your local computer and decompress it. Follow the instructions in the readme.md file before attempting this compulsory task.
- The React related tasks would see you creating apps that need some modules to run. These modules are located in a folder called 'node_modules', which is created when you run the following command from your command line or terminal: 'npx create-react-app my-app-name' or similar. Please note that this folder typically contains hundreds of files which, if you're working directly from Dropbox, has the potential to ***slow down Dropbox sync and possibly your computer***. As a result, please follow this process when creating/running such apps:
    - Create the app *on your local machine* (outside of Dropbox) by following the instructions in the compulsory task.

- When you're ready to have a code reviewer review the app, please *delete the node_modules folder*.
- Compress the folder and upload it to Dropbox.

The code reviewer who is reviewing your work will, in turn, decompress the folder, install the necessary modules and run the app from their local machine.

Compulsory Task 1

Follow these steps:

- Create a React app that:
  - Contains a dropdown that allows a user to select which functionality they want to use on your site. The options should be 'Currency converter' and 'Win!'.
  - Contains a component that converts currencies. This component should be displayed when the user selects 'Currency converter' from the Dropdown. The user should be able to enter a figure in dollars and the app should calculate and display the amount in South African Rands, UK pounds, and Euros. The components that display the converted values should be children of the conversion component.
  - Contains a component that allows the user to play a simple game. This component should be displayed when the user selects 'Win!' from the Dropdown. The user should be shown three 'Card' components that they can click. When the user clicks on a 'card' the card should be 'turned' to reveal a message which either indicates that they've won a prize or that they've lost. They should only be able to pick one 'card' with each try. Once they've picked a 'card' they should no longer be able to click the 'cards' but should either be able to 'Try again' or 'Quit'. The winning card should be different each time.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Things to look out for:

1. Create a components directory inside your src directory to contain all the components you create.

2. Remember, each component should be stored as a separate **.js** file inside the component directory and each of these files should start with a capital letter.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

Reference:

React.js. (2020). Getting Started – React. Retrieved 6 August 2020, from https://reactjs.org/docs/getting-started.html

SPOT CHECK 1 ANSWERS

1. If two or more components need to have the same state at any given time, we store the state in the nearest shared parent component instead of in the separate children components. We then pass the value we want to keep synced down as props to the child components.
2.
    a. Step 1: Make the parent component a stateful component.
    b. Step 2: In the parent component, create and register event handlers that will respond to events triggered in the child components. These event handlers are used to change the state using `this.setState()`.
    c. Step 3: In the children components, let the event handler that is triggered by the event that changes the state of the parent container respond by calling the event handler in the parent component.