Exercise 06 – Component Lifecycles

Objectives

To be able to use common lifecycle methods appropriately.

Overview

You have been asked to create an application that will provide a map of a specific location, using the Google Maps API. The specification for the application is as follows:

- There should be a heading for the page containing the text "Your Google Map Locations".
- There should be a map displayed from a hard-coded location that is the address of the training location you are currently in.
- There should be some text below the map that gives the address of the location that you are currently in.

To complete this exercise, you will need to create or use the following components:

- An **App** component that will act as a parent for the other components.
- A **MapContainer** component to display a Map in.
- A **Map** component is provided to enable the easy insertion of a Google Map (this uses the react-google-maps package already included in the project).
- A **CurrentLocation** component that will display the address that has been provided to display a map of.

Exercise Instructions

Check that the starter project runs correctly

1. In the command line, navigate into the **starter** folder for **EG06**.
2. Run the command:

```
npm install
```

3. Now run the command:

```
npm start
```

4. Verify that the browser outputs **Your Google Map Locations**.

Application boilerplate familiarity

1. Examine **index.html** that is found in the **public** folder.

You should notice that a stylesheet has already been provided, along with a link to a Bootstrap CDN. These are used for styling the page.

Completing the App Component

Constructing the component

1. Add a **constructor** to the class, including a super call and a message that logs out **"App constructor".**
2. Add **state** to the **component** that sets:
   - **currentAddress** to be a *string* containing the *office location* you are at (e.g Oxford Street, Manchester; St Katherine's Way, London; etc and addresses can be found at https://www.qa.com/training-locations).
   - **mapCoordinates** to an *object* that contains:
     - **lat** to be the *latitude of the address*
     - **lng** to be the *longitude of the address*

These can be found at http://www.latlong.net/

Inspecting Lifecycle Methods

These methods are being added to this component for demonstration purposes, they are not required for the functionality of the application.

1. Under the **constructor** add a *method* called **componentDidMount**
   - Make the method *log* out **"App componentDidMount"**
2. Under **componentDidMount** method, add another method called **componentDidUpdate**
   - Make the method *log* out **"App componentDidUpdate"**
3. Save the file and check the output.

The console should reveal that only the componentDidMount method has fired - we have not updated state to fire any other method.

4. Under **componentDidUpdate**, add **static getDerivedStateFromProps** method accepting **props** and **state** as arguments.
   - Make the method *log* out:
     - **"App getDerivedStateFromProps"**
     - **props** via a *dir*
     - **state** via a *dir*
5. Add a method**, getSnapshotBeforeUpdate**, accepting **prevProps** and **prevState** as argments.
   - Make the method log out:
     - **"App getSnapshotBeforeUpdate"**
     - **prevProps** via a *dir*
     - **prevState** via a *dir*
6. Add a method, **shouldComponentUpdate**, accepting **nextProps** and **nextState** as arguments
   - Make the method log out:

- **"App shouldComponentUpdate"**
- **nextProps** via a *dir*
- **nextState** via a *dir*

- The method should **return true** if *the NOT state* of **nextProps** AND **nextState** bring *equal* to the respective *current* value when both are converted to a string (using JSON.stringify).

```
return !(JSON.stringify(nextProps) === JSON.stringify(this.props)
&& JSON.stringify(nextState) === JSON.stringify(this.state));
```

7. Add a method, **componentWillUnmount**, making it method log out:
   - **"App componentWillUnmount"**
8. Save the file and check the output.

You should expect to see outputs from:

- App constructor
- App getDerivedStateFromProps (and two subsequent Objects)
- App render
- App componentDidMount

As we progress through the rest of this exercise, we will continue to inspect which methods are fired and when.

The CurrentLocation component

This component will be responsible for displaying the current location that is provided by address as a prop from its parent. This component could be a functional component, but since we want to examine lifecycles, we will create it as a class so that we can add the methods.

1. Create a new JSX file in the **src** folder called **CurrentLocation**.
2. Add the **import** for **React**.
3. Create a component as a **class** called **CurrentLocation**, it should be the **default export**.
4. Create a **constructor** with a **super** call and a **console.log** of **"CurrentLocation constructor"**.
5. Initialise **state** as an *empty object* in the **constructor**.
6. Add the lifecycle methods as in the **App** component - replacing any occurrence of **App** with **CurrentLocation**.
   - Note that the shouldComponentUpdate method should return:

```
return nextProps.address !== this.props.address;
```

7. Add a **render** method that logs out **"CurrentLocation render"**.
8. In **render**'s **return** add a **div** with it class defined as:

```
col-xs-12 col-md4 col-md-offset-3 current-location
```

9. Inside this **div** place a **h4** element with an **id** of **save-location** and content got from the *value* of **address** supplied by **props**.

10. Save the file.

Now we need to get the **App** component to display our new component.

11. In the **render** method of **App**, add a *wrapping* **div** tag around the **h1**.
12. Display the address defined by adding a **CurrentLocation** component that has an *attribute* of **address** set to its *current state* under the h1.
13. Add an **import** statement for **CurrentLocation** and then save the file.
14. Save the file and observe the output on the console.

You should see that the App lifecycle methods fire as before, with the exception of **componentDidMount**, which waits until the CurrentLocation component has mounted.

As we haven't given our app a mechanism to change state, we do not fire any of the update methods. That will happen in a later exercise. For now, we are going to add a Map component, which will display the map for the address and co-ordinates supplied.

The MapContainer Component

To see if the Map will update, we are going to use a containment component. The reason for this is that the Map component has to be structured in a certain way and cannot easily provide lifecycle methods. We can assume that if the MapContainer component is going to update, then its children will also update.

1. Create a new JSX file in the **src** folder called **MapContainer**.
2. Add the **import** for **React**.
3. Create a component as a **class** called **MapContainer**, it should be the **default export**.
4. Create a **constructor** with a **super** call and a **console.log** of **"MapContainer constructor"**.
5. Initialise **state** as an *empty object* in the **constructor**.
6. Add the lifecycle methods as in the **App** component - replacing any occurrence of **App** with **MapContainer**.
   - Note that the shouldComponentUpdate method should return:

```
return (nextProps.coords.lat !== this.props.coords.lat &&
nextProps.coords.lng !== this.props.coords.lng);
```

7. Add a **render** method that logs out **"MapCOntainer render"**.
8. In **render**'s **return** add a **div** with its class defined as:

```
map-overlay
```

9. Inside this **div** place another **div** with an **id** of map.
10. Inside the second div, place the following code:

```
<Map
    isMarkerShown
    googleMapURL = "https://maps.googleapis.com/maps/
api/js?v=3.exp&libraries=geometry,drawing,places"
```

```
    loadingElement={<div style={{ height: `100%` }} />}
    containerElement={<div style={{ height: `350px` }} />}
    mapElement={<div style={{ height: `100%`, maxWidth: `500px`,
margin: `auto 0` }} />}
    coords={this.props.coords}
/>
```

This is to create a **Map** component that uses a pre-prepared **GoogleMap** component from **react-google-maps** available on **NPM**. This install has already been done as the **Map** component uses it. The first five attributes are required by the **GoogleMap** component. The sixth is added by us to supply the coordinates for the map.

11. Add an **import** for the **Map** component (this component has been provided for you).
12. Save the file.

Now we need to get the **App** component to display our new component.

13. Display the map defined by adding a **MapContainer** component that has an *attribute* of **coords** set to its *current state* of mapCoordinates between the h1 and CurrentLocation component.
14. Add an **import** statement for **MapContainer** and then save the file.

The Map Component

This component has been provided for you, make sure you take a look at its code. It is a functional component that is wrapped in a *Higher-Order Component* called **withGoogleMap**, which itself is wrapped in **withScriptjs**. These are needed by the **react-google-maps GoogleMap** component to be able to display the map. You can read more about this package at:

https://tomchentw.github.io/react-google-maps

A Higher-Order Component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature.

Concretely, a higher-order component is a function that takes a component and returns a new component.

The attributes used for the **GoogleMap** component are as follows:

- **defaultZoom** sets the zoom level that is used by the displayed map;
- **center** requires an object that has the *latitude* and *longitude* of the place to display, ensuring that the center of the map is this location.

The content of the **GoogleMap** (apart from the actual map) can be described as:

- **props.isMarkerShown** specifies whether our required location is marked on the map with a pin;
- The **Marker** component itself and the position to display it, supplied as an object.

1. Make sure that all of the files are saved and view the output.

You should see a map of the location that you defined as state in the App component. Make sure that you examine the console to see the order of lifecycle methods called. Notice that none of the updating methods are ever called. We will examine these when we add a search box to our application in the next chapter.