Exercise 07 – Forms and Events

Objectives

To be able to use a form in a React application and update its state appropriately.

To further explore lifecycle methods, in particular those invoked when updating the App.

Overview

Continuing the Your Google Map Locations application that was created earlier, there is now a requirement to have a search box within the app that will allow a user to search for maps of locations. The following specifications exist:

- The search box should allow text to be input and a search icon to activate the search
- When the search is run, the application will either:
  - Update the map display with a map of the location searched for and update the current location;
  - Display "Location not found…" in the current location component.

In addition this, we will examine the lifecycle methods that are invoked when state changes within the app.

Exercise Instructions

Check that the starter project runs correctly

1. In the command line, navigate into the **starter** folder for **EG07**.
2. Run the command:

```
npm install
```

3. Now run the command:

```
npm start
```

4. Verify that the browser outputs the **Your Google Map Locations** as it did at the end of the previous exercise.

Adding a Search Component

1. In the **src** folder, create a file called **Search.jsx**.
2. Declare **Search** as a **class** component

Constructing the component

1. Add a **constructor** to the class, including a super call and a message that logs out **"Search constructor".**
2. Add **state** to the **component** that sets:
   - **value** to be an empty *string*.

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 **"Search componentDidMount"**
2. Under **componentDidMount** method, add another method called **componentDidUpdate**
   - Make the method *log* out **"Search 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:
     - **"Search 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:
     - **"Search 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:
     - **"Search 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:
   - **"Search componentWillUnmount"**
8. Add a **render** method that initilly logs out **Search render**.
9. Add a **return** to the function that has the following structure:
   - A wrapping **form** component with:

- **id** of **geocoding_form**;
- **className** of **form-horizontal**;
- An inner **div** with the **className** of **form-group**;

10. Inside the **form-group div** add the following HTML:

- A wrapping **div** with the **className** of **col-xs-12 col-md-6 col-md-offset-3**
- An inner **div** with a **className** of **input-group** with:
  - An text **input** with **className** of **form-control**, **id** of **address** and **placeholder** of **'Find a location...'**
  - A **span** with a **className** of **input-group-btn** with an inner **span** with attributes:
    - **className** of **glyphicon glyphicon-search**
    - **aria-hidden** of **true**

11. Save the file.

We need to add the **Search** component to the **App** component.

12. In **App**, **import** the **Search** component.
13. Render the **Search** component *between* the **h1** and the **MapContainer**.
14. Save all files and check that the **Search** component is displayed on the page.

You should be able to type into the text input but nothing will happen because **Search** is currently an *UNCONTROLLED* component. To make it a *CONTROLLED* component and update the **currentAddress** we need to obtain the value searched for.

Controlling the Search Component

Firstly, we will put internal **state** on the **Search** component to update the **state** each time the **value** changes. This will allow us to pass this **value** back to the **App** component to update its **state**.

1. In the **input**, add:
   - An **onChange** event with a handler of **this.handleChange**;
   - A **value** attribute set to **this.state.value**.
2. In the **constructor** add **this.handleChange** *as an arrow function* that takes an **event** and *updates* **state.value** with the **value** of the *event's target*.
3. Save the file and inspect the React tab of the developer tools.
4. Type in the input and you should see that the **state** of **Search** updates with each keystroke.

Passing form data to the parent App component

Next we need to deal with the user either submitting the form (by pressing Enter on the keyboard) or clicking the Magnifying Glass glyphicon. Both these events will be dealt with the same event handler function.

1. Open **App.jsx**.
2. Add an **onSubmit** event to the **form** element with a handler of **this.handleSubmit**.
3. Add an **onClick** event to the *inner* **span** with a handler of **this.handleSubmit**.
4. In the **constructor** add **this.handleSubmit** as *an arrow function* that:

- Takes an **event** as an argument;
- Stops the default action of the event;
- Calls the (as yet unwritten) props function called **onSearch** with an argument of **this.state.value**.

5. Save the file.

Receiving data from the child component form

We need to modify the App component so that it takes the address from the form an updates its state with the value. We have already indicated that this will be done by a prop called onSearch that will be added to the rendering of the Search component. We need this onSearch to have a handling function that will:

- Fetch the map and data of the address searched for;
- Extract the lat and lng of the address searched for;
- Update state with the new address and lat/lng values;
- Catch any errors and display Location not found… and set the lat and lng to zero.

1. In **App.jsx**, add an attribute of **onSearch** to the **Search** component and set it to **this.searchForAddress**.
2. In the **constructor**, set a value **this.searchForAddress** to be **this.searchForAddress.bind(this)**.
3. Under the lifecycle methods (but before render) add an **async** function called **searchForAddress** which:
   - Takes an argument of **address** (i.e. the state from the form);
   - Declares a **const concatAddress** that *replaces* any **spaces** in **address** with a **'+'**;
   - Declares a local variable **currentState**, set to **this.state**;
   - *Tries* the following:
     - Set a **const url** to be the following:

```
const url =
`https://maps.googleapis.com/maps/api/geocode/json?address=${concaatAddress}`;
```

     - Declares **response** to be the result of the **fetch** of **url** (**await**);
     - Declares **responseJSON** to be the result of **response.json()** (also **await**);
     - Sets **currentState.mapCoordinates** to be:

```
await responseJSON.result[0].geometery.location;
```

     - Logs out the value of **currentState.mapCoordinates**.
   - *Catching* any **error** and:
     - Setting **currentState.currentAddress** to be **Location not found…**
     - Setting **lat** and **lng** in **currentState.mapCoordinates** to **0**.
   - Set the application **state** to **currentState**.
4. Save the file and observe the application.
5. Check to see that the map updates if you use a new address.

6. Check the lifecycle methods and note what happens when shouldComponentUpdate is true.
7. Try resubmitting the same text. What affect does this have?

Part of the advantages of React is that it doesn't update the actual DOM unless there are changes to make!

APPENDIX

Google Maps JavaScript API documentation:

https://developers.google.com/maps/documentation/javascript/