General notes

03/10/18

Searched for credible sources to learn react.

Progression will be logged in a git repo (Bitbucket).

Findings:

Research functional programming

Egghead has React tutorial with good references

Set a goal for myself. Create a mini app that calculates certain average values for special dice rolls.

Talked to React developer to get some inside knowledge.

03/11/18

Moved from personal bitbucket account to school github account.

Moved from egghead to advancedreact.

Advanced react set up and ready to go.

Learning about components

Comments are supposed to be put inside {} -> these put you into javascript land

No siblings in components

Learning about CSS in react

Created a layout using components

Learning about props

Learning about passing dynamic data

A component is basically just an object

Console gives comments about possible codesmells or unused pieces of code

Learning about stateless functional components

Learned about implicit returns:

**const** Header = (props) => (

Learned about react routing

Learned about page not found

Learned about helper functions & utility functions -> possible to only load in a single function

React rule: don’t touch the dom

Learned about event handlers

Functions are called inline, without parenthesis because else they will be called on page load

You can pass an event object to the called method

Refs allow you to reference dom nodes on the page

‘this’ does not always refer to the entire component. This has to do with binding. There is a whole bunch of lifecycle events for example componentDidMount() -> this will mount as soon as the component is loaded. So, all of the built-in methods of react bind this to the component object. When we add methods, they are not bound by default. That means its hard to reference the component inside of one of its own methods. The solution is to bind our own methods as well.

Example:

constructor() {  
 **super**();  
 **this**.goToStore = **this**.goToStore.bind(**this**);  
}

Side note:

super(); must always be called first in a constructor. This passes in the react parent.

This however might clutter your constructor when you have a lot of custom functions. If you find yourself having a lot of custom functions you can instead use them as a property. ‘this’ does bind to properties.

Example:

goToStore = (event) => {

Learning about state

State is an object that holds data that itself needs as well as some children may need. You can think of state as a single state of truth. Data sometimes lives in multiple places. For example, store data in a DOM element. With react the golden rule is don’t touch the dom. So, we just want to update the item that we want to change, not everything. So, state is a place where data lives. This way all the pieces of data that use this state get updated once the state is updated.

To pull values out of form fields, use refs

Example:

**import** React **from** 'react';  
  
**class** AddFishForm **extends** React.Component {  
  
 nameRef = React.createRef();  
 priceRef = React.createRef();  
 statusRef = React.createRef();  
 descRef = React.createRef();  
 imageRef = React.createRef();  
  
 createFish = (event) => {  
 event.preventDefault();  
  
 **const** fish = {  
 nameRef: **this**.nameRef.value.value,  
 priceRef: parseFloat(**this**.priceRef.value.value),  
 statusRef: **this**.statusRef.value.value,  
 descRef: **this**.descRef.value.value,  
 imageRef: **this**.imageRef.value.value,  
 };  
  
 console.log(fish);  
 };  
  
 render() {  
 **return** (  
 <form className={'fish-edit'} onSubmit={**this**.createFish}>  
 <input name="name" ref={**this**.nameRef} type={"text"} placeholder={"Name"} />  
 <input name="price" ref={**this**.priceRef} type={"text"} placeholder={"Price"} />  
 <select name="status" ref={**this**.statusRef}>  
 <option value="available">Fresh!</option>  
 <option value="unavailable">Sold Out!</option>  
 </select>  
 <textarea name="desc" ref={**this**.descRef} type={"text"} placeholder={"Desc"} />  
 <input name="image" ref={**this**.imageRef} type={"text"} placeholder={"Image"} />  
 <button type={"submit"}>Add Fish</button>  
 </form>  
 );  
 }  
}  
  
**export default** AddFishForm;

Now that we have the constant “fish” we have to get it into state, so that It can be used across multiple components. App is the parent that holds all components, so this is where we want the data to live. So, we need to go into our app component and create state. This is done by setting property on the highest parent component.

Example:

**class** App **extends** React.Component {  
 state = {  
 fishes: {},  
 order: {},  
 };

Now we have to be able to reach these states. This is done through a method on the same parent component. First, you have to pass down the method as a property.

Example:

Parent:

<Inventory addFish={**this**.addFish}/>

Child (here addFish is a property, that’s why you need to add .props):

<AddFishForm addFish={**this**.props.addFish}/>

Grandchild:

**this**.props.addFish(fish);

The lowest level component how has access to the state. Now we have to fill the state. This is never done directly, this I mutation and is wrong. First, we need to take a copy of the existing state. Then we add our new element to the copy. Finally, we have to set the new object to state.

Example:

addFish = fish => {  
 **const** fishes = { ...**this**.state.fishes };  
 fishes[`fish${Date.now()}`] = fish;  
 **this**.setState({ fishes });  
};

04/11/18

Went trough a recap of states to refresh and better understand them.

Always key objects so React can get to it faster. It will still find the object without, but much slower. This also defeats the purpose of using React.

Example (yellow letters):

{Object.keys(**this**.state.fishes).map(key => <p key={key}>{key}</p>)}

When accessing props inside a child the referencing can get quite long. For example:

**const** name = **this**.props.details.name;  
**const** image = **this**.props.details.image;

etc…

To put this into one line, use the code below:

**const** {name, image} = **this**.props.details;

This will take = **this**.props.details.name and **this**.props.details.image and put it into the variables inside the first brackets: name, image.

Learned how to loop over a load of data and output some nice-looking components. Getting more insight as to why components are used. They can not only be reused on the same spot, but for example also in e-mails, other pages, a modal. Kewl!

Did some more advanced updating of the order state where stuff was passed around trough different components.

It is possible to pass the entire state object to a component using:

{...**this**.state}

However, this is lazy and can get you into trouble. Seeing as we are working with modular components, we want to spread our data accordingly. So never pass down data it does not need.

Do not overload your render functions. When you notice that your render function is getting too long, you could probably shell off some of that functionality to a separate component. However if you feel that the logic is just part of this component, you can create multiple render functions with different names and call them from your original render function.

In order to save your states to a database you can use firebase. Setup example:

**import** Rebase **from** 're-base';  
**import** firebase **from** 'firebase';  
  
**const** firebaseApp = firebase.initializeApp({  
 apiKey: "AIzaSyAEGNW63-MiuHFPY-W-J72kKzXsf0YuJR8",  
 authDomain: "fishes-47936.firebaseapp.com",  
 databaseURL: "https://fishes-47936.firebaseio.com",  
});  
  
**const** base = Rebase.createClass(firebaseApp.database());  
  
**export** { firebaseApp };  
**export default** base;

In order to sync up your state with the database you will be using lifecycle methods. The one that we are going to be using is componentDidMount()

Example:

componentDidMount() {  
 **this**.ref = base.syncState(`${**this**.props.match.params.storeId}/fishes`, {  
 context: **this**,  
 state: 'fishes'  
 });  
}  
  
componentWillUnmount() {  
 base.removeBinding(**this**.ref);  
}

it is important to remove the binding afterwards to prevent memory leaks.

Now possess enough knowledge to create dice app, lets go!