Unit 3 Sprint 1

**Class Components**

**React.Component**

* A base class that allows us to use some of the methods that the React team has curated to tap into what we call the Component Lifecycle
* Life cycle hooks are methods that give us control over how our components work and can be used by building out a class component that extends the React.Component parent class
* By creating components as classes, you can set up a data object that your component is concerned with by using state and setting up that object on our constructor method

Text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Use CCR acronym to remember all of the steps ( Class Constructor Render )

A screenshot of a computer

Description automatically generated with medium confidence

Now that we have constructed a skeleton for our Class component it can be more dynamic by using some data that we pre-define as some information we’d like our component to display. We will take that data and present it to the DOM within some text with **interpolation**

A list of benefits to what we get out of the Component class can be found [here](https://reactjs.org/docs/react-component.html)

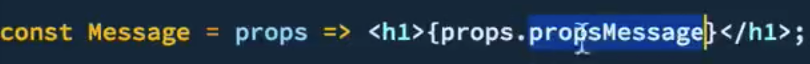
The *state* object that gets set up on the *constructor* allows us to drive our UI using data; like Facebook, the UI represents the data close to real-time.

Graphical user interface, text, application

Description automatically generated

Remember that what you define here is what needs to be referenced on the props object

Text

Description automatically generated 

**Reactivity** – the process in which changes to state immediately re-render the parts of the components effected by that change of state

~Executing functions in our child components will cause state to change at our parent level components, resulting in reactive rendering throughout all of our application~

When using **event handlers** within class components, just like props and state, they are bound to the instance of this class and are accessed through *.this*

class Button extends React.Component {

handleButton = (e)=> {

console.log(e); }

render() {

return <button onClick={this.handleButton}>Click Me</button>;

}

}

The “e” parameter is known in React as a **synthetic event** object inside of which we will have access to various pieces of information regarding what the event triggered, including the target DOM element, the type of event, and methods that control the propagation of that event (like preventDefault)

* One of the most useful properties attached to synthetic events is target which provides info on the text, value, style, attached attrubutes and other useful data within our DOM element

Building out some event handler functions using the following event listeners (onClick, onDoubleClick, onMouseEnter, onChange):

Graphical user interface, text, application, email

Description automatically generated

This saves the message in state:

Text

Description automatically generatedText

Description automatically generated

**The React Lifestyle**

Diagram, timeline

Description automatically generated

Birth/Mounting Phase:

* Component is being built out from the ground up; whatever initial data you want access to will be defined on the constructor of this phase
* Render method is invoked
* componentDidMount gets called as well

Growth/Updating Phase:

* This is where you are updating component data
* setState can be used to change the component’s state data, forcing a call to render
* shouldComponentUpdate is a method that could be used here to stop a component from calling render

Death/Un-mounting Phase

* This phase includes removing the component from the screen
* componentWillUnmount is called and can be used for any cleanup you may need to do

Graphical user interface, text

Description automatically generated

Constructor()

* If you don’t initialize state and you don’t bind methods, you don’t need to implement a constructor for your React component
* The constructor’s purpose in React is to create components with inciting state data for the initial render
* When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor
* You should not call setState(\*) in the constructor(). Instead, if your components needs to use local state, assign the initial state to this.state directly in the constructor
* This is the only place where you should assign this.state directly. In all other methods, you need to use this.setState() instead

import React from 'react';

import { data } from './extraneousSource.js';

class MyComponent extends React.Component {

constructor() { //if I wanted to receive some props here I could pass them in through constructor(props)

super(); //if I receive props through the constructor will need to pass them back through super(props);

this.state = {

arbitraryStateData: data,

}

}

}

Render()

* The render() method is the only required method in a class component
* The render()function should be pure, meaning it does not modify component state, it returns the same result each time it’s invoked, and it does not directly interact with the browser
* When called, it should examine this.props and this.state and return one of the following types:
  + **React elements.** Typically created via [JSX](https://reactjs.org/docs/introducing-jsx.html). For example, <div /> and <MyComponent /> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.
  + **Arrays and fragments.** Let you return multiple elements from render. See the documentation on [fragments](https://reactjs.org/docs/fragments.html) for more details.
  + **Portals**. Let you render children into a different DOM subtree.
  + **String and numbers.** These are rendered as text nodes in the DOM.
  + **Booleans or null**. Render nothing. (Mostly exists to support return test && <Child /> pattern, where test is boolean.)

render () {

return (

<div>

// map returns an array remember? So lets give react an array of JSX elements and let it perform it's magic. For this example we'll assume this list is a string.

{this.state.arbitraryStateData.map(data => <div>{data}</div>)}

</div>

);

}

ComponentDidMount()

* Invoked immediately after a component is mounted/ guaranteed to be called only once in the whole lifecycle
* Initialization that requires DOM nodes should go here
* If you need to load data from a remote endpoint, this is a good place to instantiate the network request (used for any async data loading that you’d need to have on state during the render)

A picture containing text, electronics, computer, projector

Description automatically generated

ComponentDidUpdate()

* Invoked immediately after updating occurs
* Not called for the initial render
* Use this as an opportunity to operate on the DOM when the component has been updated

ComponentWillUnmount()

* Invoked immediately before a component is unmounted and destroued
* Perform any necessary cleanup in this method, such as invalidating timers, caneling network requests, or cleaning up any subscriptions that were created in componentDidMount()
* You should not call setState() in componentWillUnmount() because the component will never be re-rendered. Once a component instance in unmounted, it will never be mounted again.

[Article](https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d) about the component lifecycle