WORKSHOP #2

THE SELECTOR

CREATE-REACT-APP

▸ Now that we have installed and used create-react-app. Lets create a project called Selector.

▸ Type - " npx create-react-app selector" in our console.

▸ Follow the suggested prompt of running the commands

cd selector
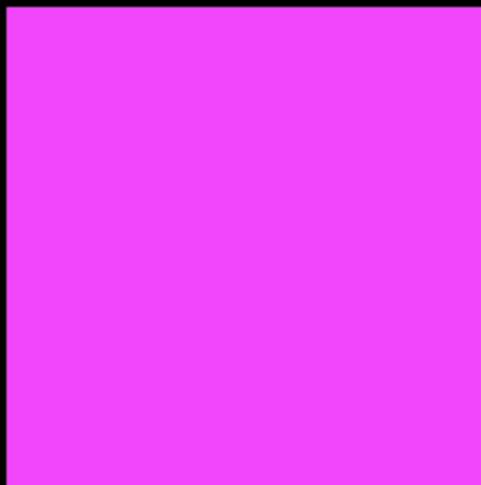npm start

▸ Now we should see our server running and the react-app boilerplate site being displayed.

OUR GOAL

Our goal is to build an application that when a selected shape is clicked, it will display the name and the shape in the nav bar.

SELECTOR

▸ Delete all the code in your App.js file and lets start from scratch again.

▸ So now that we have an idea on what we want to create. Take a moment and think about where should we start. See if you can answer these questions.

▸ What is the major functionality of the application?

▸ What type of components should we implement based on the functionality of the application? I.E. Stateful class component, or a pure functional component?

▸ How will you structure the JSX and styles to accomplish this view or UI?

▸ We know that this application will display three shapes.

▸ Our Selector application displays the shape name that is selected. Usually, and almost always, when you have values that are changing in your views you are going to need State.

▸ I'll provide the CSS and guide you along the way on how to add in styling.

▸ Lets start by navigating to the github and pulling the CSS that is prebuilt for this workshop.

▸ https://github.com/CaptainKRS/lessons/blob/master/React%20Workshops/workshop-2/CSS/style.css

▸ From here copy the index.css file from GitHub and copy it into your index.css file in your selector project.

▸ Lets start by importing the proper libraries into your **App.js** file.

▸ Import React and the Component so we can create our stateful class component and use JSX.

```
1  import React, {Component} from 'react';
```

▸ Now that we have our imports complete we need to create our Selector class component.

```
 1   import React, {Component} from 'react';
 2
 3   class Selector extends Component {
 4
 5     render () {
 6       return (
 7         <div></div>
 8       )
 9     }
10   }
11
12   export default Selector;
```

Remember when creating components in our create-react-app projects. We need to be sure to export these components so throughout our Application we can use them.

▸ Now that we have our component built, we should think about what we want our app to look like so we can start building the JSX. Lets take a look at the end result again.

Here we will need a nav bar that displays the current selected color

Here we will want our shapes to be displayed

▸ To help get you started lets go ahead and build out what we know so far.

```
 4 ⌄    render() {
 5 ⌄      return(
 6 ⌄        <div className="container">
 7 ⌄          <div className="navbar">
 8            <div>Selected: <span> What goes here!?!?</span> </div>
 9            </div>
10 ⌄          <div className="shape-list">
11 ⌄          Add the shape component here!!
12            </div>
13          </div>
14        )
15    }
```

In your return statements you can only have one parent JSX element. A common pattern is to wrap everything in a div.

▸ In this workshop we will be using a functional component we defined as Shape. Lets create the **Shape.js** file in our src directory (the same directory that your App.js file is in) and start coding our functional component.

▸ This component is going to display our shapes that we're going to be selecting.

```
1    import React from 'react';
2
3  ⊟ const Shape = () => {
4  ⊟   return (
5        <div className="square"/>
6      )
7    }
8
9    export default Shape
```

SELECTOR

▸ Remember that we want our shapes to be represented in our JSX. How do we do this?

▸ Remember we need to Import this Shape component into our App.js file. Example: Import OurFancyComponentName from "./SomeFile.js"

▸ For now lets just add in three instances of our Shape Component.

▸ Navigate to your index.js file and add the Shape Component between the div with a className of shape-list.

```
<div className="shape-list">
  <Shape/>
  <Shape/>
  <Shape/>
</div>
```

You should see three pink squares in line with one another

▸ Lets declare the shape property to be passed into Shape component with a specific shape assigned.

```
<div className="shape-list">
  <Shape shape="square"/>
  <Shape shape="circle"/>
  <Shape shape="triangle"/>
</div>
```

The shape property we are passing to our component is going to be passed in as a key on the props object.

▸ Now this is where passing props into Components comes in handy.

▸ Lets first modify the Shape component so that we can accept dynamic properties into it.

▸ Remember that props are an object with key value pairs so we can use the dot notation to access the values within the object.

```
const Shape = (props) => {
  return(
    <div className={props.shape} />
  )
}
```

▸ Remember, that we need a stateful component. Think for a moment: what should change about the view of our Selector app as user interact with it, and how could we represent that data?

▸ The data that changes in our UI is the shape that is picked being displayed in our nav bar. We can represent this as a JavaScript string. Let's add state to our component, and some behavior that will cause that state to change.

▸ To do this we need to add a constructor to your Selector class component and define state. Note that state is always defined as a Javascript object that have key value pairs.

▸ Make sure to add your constructor directly under where we create the Selector class and before the render method.

```
4   class Selector extends Component {
5     constructor() {
6       super();
7       this.state = {
8         selectedShape: 'square',
9       };
10    };
```

We initialize our state by setting this.state to a Javascript object and the key and value we want our component to be initialized to.

▸ Now that we have our state correctly declared. We can now use that data to be represented in our application.

▸ Where did we want to have the name of the shape selected represented? We wanted it to represent the clicked shape in the nav bar. So here is where we will represent our state.

```
23  ⊟    <div className="navbar">
24         <div>Selected: </div>
25         <div>{this.state.selectedShape}</div>
26       </div>
```

Remember, this.state is an **object** with a key named selectedShape.

▸ Now we need to add some interactivity. We already added state to our Selector component that represents our "selectedShape". Now we need to write a method in our Selector component that will change the state based off of what shape is clicked and then we will pass it down to our Shape Component, so that the shape components can attach it to their click handlers.

```
selectShape = (shapeName) => {

}
```

Take a moment and think about what this method is aiming to accomplish. We want to be able to pass in a shapeName and use that to set the new state to the shapeName that was passed in.

▸ We want to use the shapeName argument and set the state of selectedShape with that chosen shapeName.

```
selectShape = (shapeName) => {
  this.setState({
    selectedShape: shapeName
  })
}
```

this.setState() is a React method that allows us to create methods that will be able to change the state and cause our application to re-render the changes

▸ Now that we have our selectShape method. We need to pass this method as a prop into our Shape component. We will use this method to be invoked in our Shape component so we can change the state to represent what shape was clicked.

```jsx
<div className="shape-list">
  <Shape shape="square" selectShape={this.selectShape} />
  <Shape shape="circle" selectShape={this.selectShape} />
  <Shape shape="triangle" selectShape={this.selectShape} />
</div>
```

▸ Now that we passed our selectShape method into our Shape Component, lets use it when someone clicks on our component.

▸ To do this we need to attach a onClick listener to the div tag in our return statement.

```
3    const Shape = (props) => {
4        const shape = props.shape
5        const selectShape = props.selectShape
6
7        return(
8          <div className={shape} onClick={() => selectShape(shape)} />
9        )
10   }
```

▸ Creating variables from the props that we pass in is a common strategy.

▸ Also, passing arguments to event handlers in React, instead of passing the function directly to the click handler, we wrap it in another function. This give us room to pass in arguments to the function. In this instance we want to pass the shape argument into the selectShape method.

```
3   const Shape = (props) => {
4      const shape = props.shape
5      const selectShape = props.selectShape
6
7      return(
8        <div className={shape} onClick={() => selectShape(shape)} />
9      )
10  }
```

SELECTOR EDGE CASES

▸ You should now be able to click on the shapes and see the name that corresponds to the correct shape in the nav bar.

▸ EDGE CASE TIME!

▸ If you look in styles.css you'll see tons of different shapes you can use. Try using several of these and adding them to your program.

▸ Make a Navbar component that you pass the correct props into to represent the shape that is selected.

▸ Make a Footer component that is styled, & displays how many shapes are clicked.

▸ If you got that far great!... Now show me a list of how many times each individual shape was clicked and displayed in the footer as well.

▸ If you've made it this far you're doing great!

▸ Here are some materials you should look at before our next workshop.
https://reactjs.org/docs/components-and-props.html