WORKSHOP #2

THE COLOR PICKER

STACKBLITZ.COM

▸ Lets get started by navigating to stackblitz.com

▸ Click on the **React** Javascript button to start a new project. You should find this as you scroll down the main page.
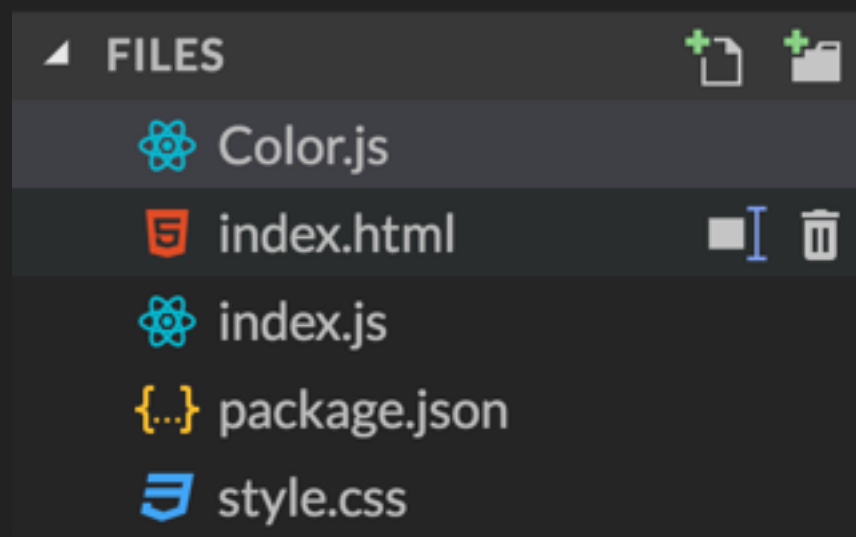
OUR GOAL

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

COLOR PICKER

▸ Delete all the code in your index.js file and lets start from scratch again. Delete the Hello.js file from our file list on the left side, and create a file named Color.js.

```
◢ FILES                        ⊕  ⊕
      ⚛ Color.js
      🛡 index.html           ▣I 🗑
      ⚛ index.js
      {..} package.json
      ⋥ style.css
```

This is what your Files tree should look.

▸ 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 colored circles.

▸ Our Picker application displays the color that is picked. Usually, and almost always, when you have values that are changing in your views you are going to need State. This should help you in making your decision between a pure functional component or a stateful class component.

▸ 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.

    ▸ github.com/voodoobrew/workshop-1

▸ In this workshop we will be using a functional component we defined as Color. Lets navigate to the Color.js file and start coding our functional component.

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

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

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

▸ Import React and the Component so we can create our stateful class component and use JSX. Then we want to import ReactDOM from the react-dom library so we can attach our component to the DOM. Finally import the Color component so we can use this in our main index.js file.

```
1    import React, {Component} from 'react';
2    import ReactDOM from 'react-dom';
3    import Color from './Color'
4    import './style.css'
```

Importing style.css file allows us to use that file to add styling to our JSX

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

```
 6  ⊟ class Picker extends Component {
 7  ⊟     render () {
 8  ⊟       return (
 9            <div></div>
10          )
11        }
12      }
13
```

▸ Render you Picker component to the DOM by declaring it as the first argument in your ReactDOM.render method. Note that we write the component as a self-closing JSX tag.

```
ReactDOM.render(<Picker />, document.getElementById('app'));
```

The first argument for the render method is the component you wish to attach to the DOM, and the second argument is how we attach it to a specific HTML element. In this case its the element with the id 'root'.

▸ At this point your index.js file should look exactly like this.

```
1    import React, {Component} from 'react';
2    import ReactDOM from 'react-dom';
3    import Color from './Color'
4    import './style.css'
5
6    class Picker extends Component {
7      render () {
8        return (
9          <div></div>
10       )
11     }
12   }
13
14   ReactDOM.render(<Picker />, document.getElementById('app'));
```

▸ 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.

Currently selected: red

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

Here we will want our colors to be displayed

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

```
14  ⊟    render() {
15  ⊟      return (
16  ⊟        <div id="container">
17  ⊟          <div id="navbar">
18              <div>Currently Selected: </div>
19              <div>? what goes here?</div>
20            </div>
21  ⊟          <div id="colors-list">
22              Add the Color Components here
23            </div>
24          </div>
25        )
26    }
```

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

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

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

```
<div id="colors-list">
  <Color />

  <Color />

  <Color />
</div>
```

You should see three red circles in line with one another

▸ Now that our Color component is setup to access the property of color we pass into it. Lets add the color property to our Color component with a specific color assigned.

```
<div id="colors-list">
  <Color color="red"/>

  <Color color="blue"/>

  <Color color="green"/>

</div>
```

You should now see the three circles with the passed in color values

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

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

▸ Remember that props are an object with key value pairs.

```
3   const Color = (props) => {
4     return(
5       <div className ={props.color} />
6     )
7   }
```

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

▸ The data that changes in our UI is the color 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 Picker class and define state on the component. 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 Picker class and before the render method.

```
 7  ⊟  class Picker extends Component {
 8  ⊟     constructor() {
 9           super();
10  ⊟        this.state = {
11             selectedColor: 'red'
12           }
13         }
```

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

```
selectColor = (colorName) => {

}
```

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

▸ We want to use the colorName argument and set the state of selectedColor with that chosen colorName

```
selectColor = (colorName) => {
  this.setState({
    selectedColor = colorName
  })
}
```

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

```
<div id="colors-list">
  <Color color="red" selectColor={this.selectColor}/>
  <Color color="blue" selectColor={this.selectColor}/>
  <Color color="green" selectColor={this.selectColor}/>
</div>
```

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

▸ To do this we need to attach a onClick listener to the html tag we set.

```
3  ⊟ const Color = (props) => {
4        const color = props.color
5        const selectColor = props.selectColor
6  ⊟    return(
7          <div className ={props.color} onClick={() => selectColor(color)} />
8        )
9    }
```

▸ 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 color argument into the selectColor method.

```
3   □ const Color = (props) => {
4         const color = props.color
5         const selectColor = props.selectColor
6   □     return(
7           <div className ={props.color} onClick={() => selectColor(color)} />
8         )
9     }
```

▸ Lets give it a try now.   Wait!!! the color on the nav bar isn't changing. Try and figure out how to make the nav bar display the color that was selected.   Then set the className for the <div> tag containing the color that was picked to display the color as a background as well.

▸ Then add all the colors instead of three circles… or you could create and add in your own custom colors.

▸ If you look in styles.css you'll see a class name called "selected" that will give a black border if used. Using your new knowledge, try to make it so that when you click on a Color, it will not only cause the nav bar to show the selected color, but it will also give that color the "selected" class.

▸ 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