

WORKSHOP #1

THE COUNTER

CREATE-REACT-APP

- ▶ Now that we have installed and used create-react-app. Lets create a project called Counter.
- ▶ Type - "create-react-app Counter" in our console.
- ▶ Follow the suggested prompt of running the commands

```
cd Counter
```

```
npm start
```

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

HELLO WORLD

- ▶ Now we have to honor the tradition of first saying “Hello World”
- ▶ Lets change the code in our App.js file to display our lovely message.
- ▶ First, delete everything in our App.js file so we can start from scratch.

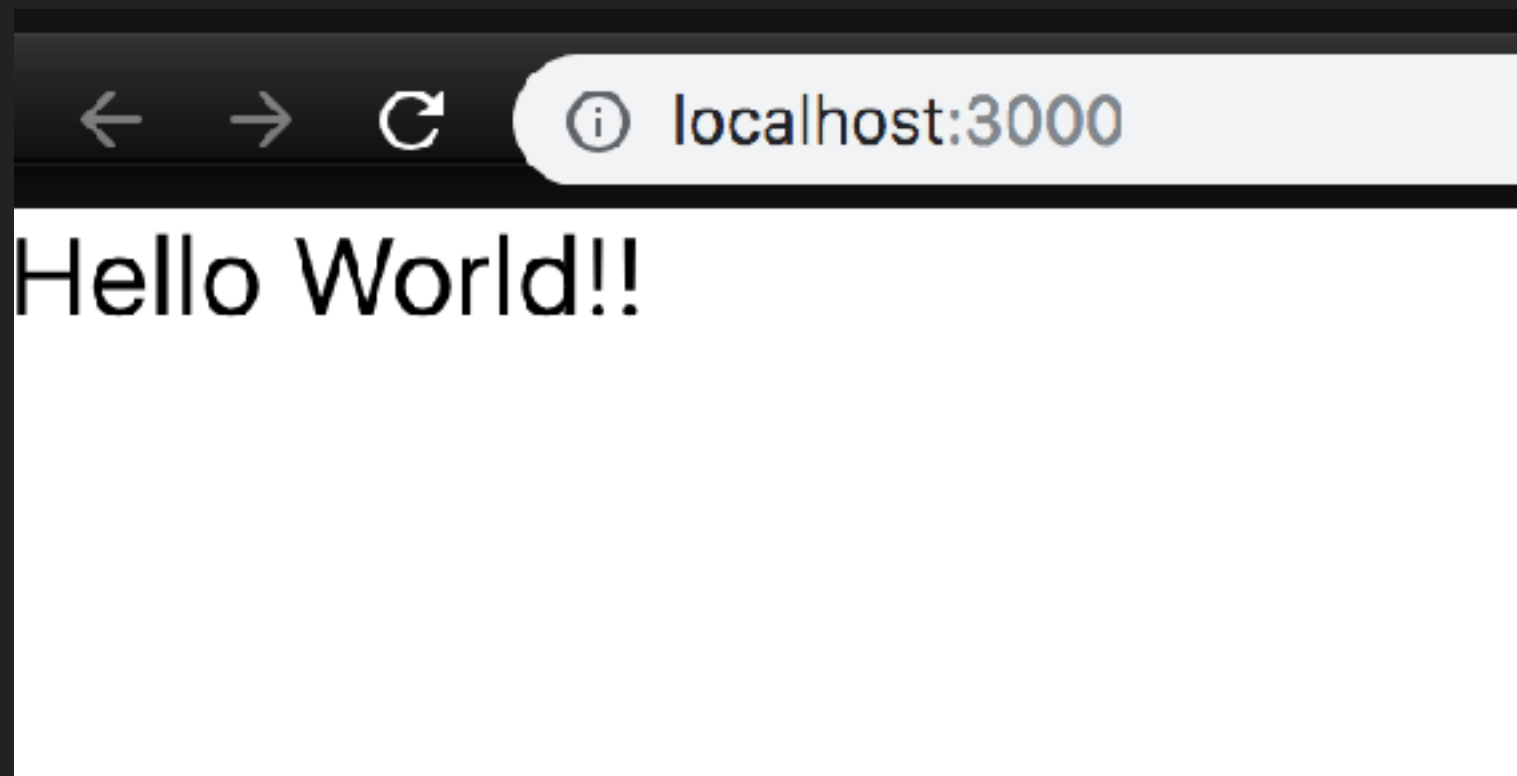
HELLO WORLD

- ▶ Lets start by importing the React library and its extended class Component.
- ▶ Then we create our HelloWorld Component. This will only render out a single div with Hello World between it.

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

HELLO WORLD

Now we should switch to our browser and open our page that is displaying our application.

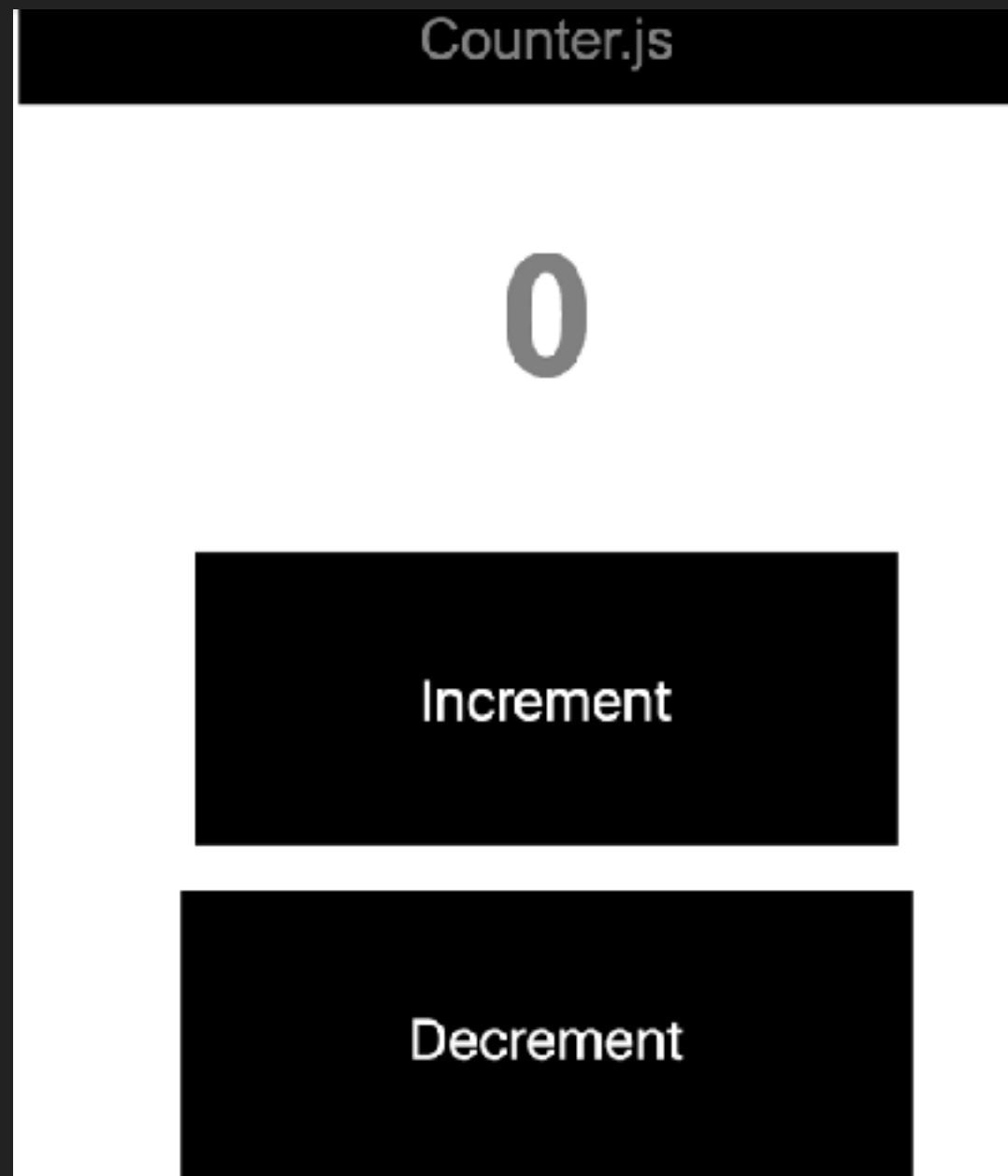


COUNTER APPLICATION

- ▶ First off... lets take a moment and say goodbye to our HelloWorld code by deleting every line of code in app.js
You want to become comfortable with writing components from scratch. So what better time to start than now.

OUR GOAL

- ▶ This is what our counter application will look like if complete correctly



- ▶ 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 component 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 UX?

- ▶ We know that this application will display a count based on the number of times a increment and decrement button is clicked.
- ▶ Our counter has to start at 0. 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.
- ▶ Lets not worry this round about how to style and set up this application. I'll provide the CSS and guide you along the way on how to add in styling.

- ▶ Okay, phew... that was a lot of reading. Now lets start coding.
- ▶ Lets start by importing the proper libraries.
- ▶ 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 first stateful class component.

```
5  class Counter extends Component {  
6    render() {  
7      return(  
8        <div></div>  
9      )  
10   }  
11 }
```

Note that all of the JSX describing our view is being returned by a method on the class called `render`. This is a special method that all class components must have.

Tip:

When naming our class component make sure to use names that are clear and declarative. Like classes in Javascript, make sure to capitalize the first letter of the class name.

- ▶ Now we want to make sure that our Counter component can be used throughout our application.
- ▶ We accomplish this by using the export default command. Our export default statement will come after the last code block of the Counter component.

```
11  export default Counter;
```

COUNTER APPLICATION

- ▶ At this point your App.js file should look exactly like this.

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

- ▶ Remember, we're building a stateful component. Think for a moment: what should change about the view of our Counter app as user interact with it, and how could we represent that data?
- ▶ The data that changes in our UI is the number representing the count. We can represent this as a JavaScript number. 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 counter class and define state on the component. Note that state is always defined as a Javascript object that have key value pairs.

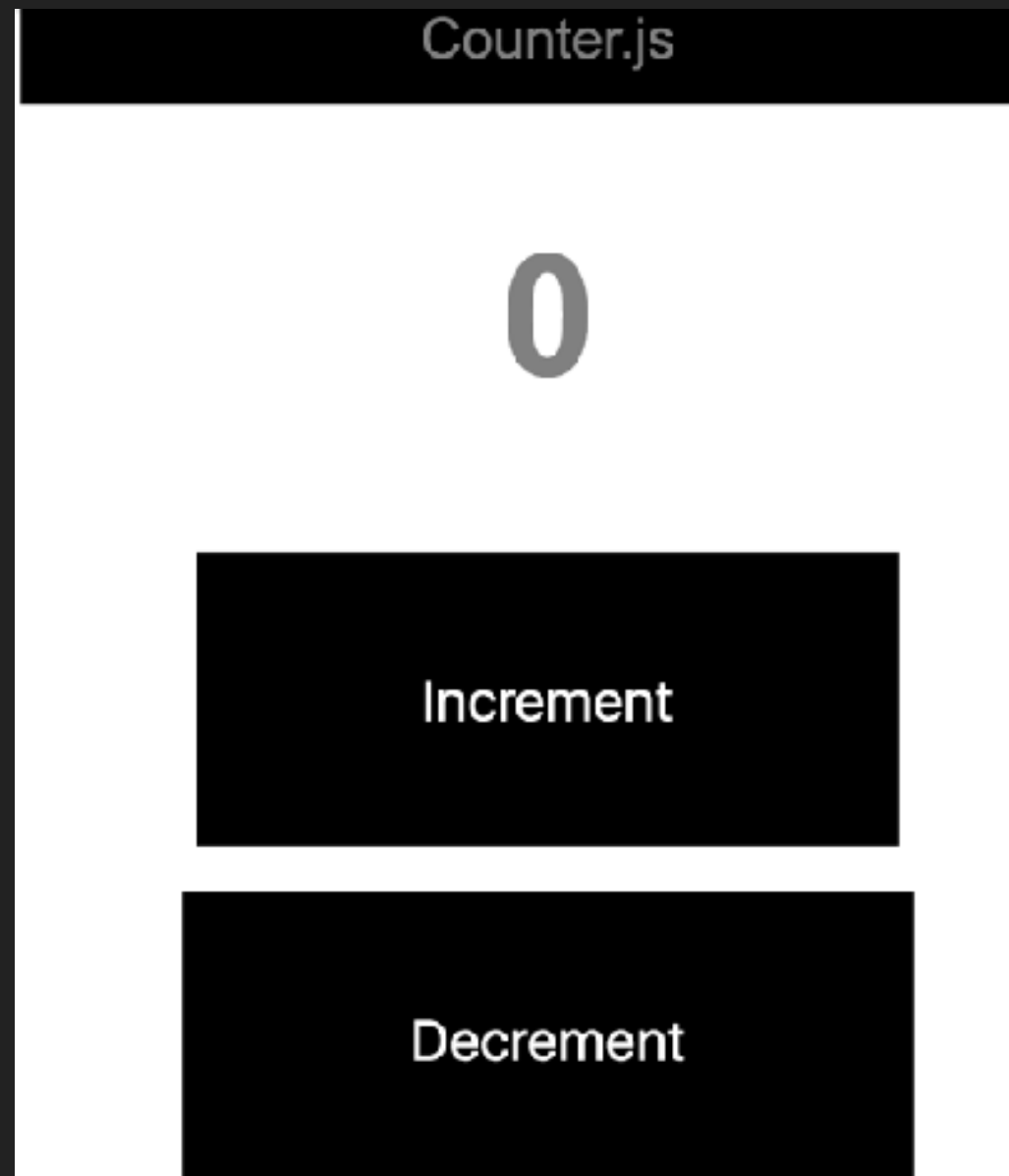
- ▶ Make sure to add your constructor directly under where we create the Counter class and before the render method.

```
5  class Counter extends Component {  
6    constructor() {  
7      super();  
8      this.state = {  
9        count: 0,  
10     };  
11  }
```

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.

COUNTER APPLICATION

- ▶ 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

Here we will want our count to be displayed

Here we will need two buttons
one for increment and one for
decrement.

COUNTER APPLICATION

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

```
render() {  
  return (  
    <div className="container">  
      <div className="navbar">Counter.js</div>  
      <div className="counter">  
        <h1>What do we put here?</h1>  
        <button type="button">Increment</button>  
        <button type="button">Decrement</button>  
      </div>  
    </div>  
  );  
}
```

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

Note: Look at what is between the h1 tags.

How can we represent our counters current state?

COUNTER APPLICATION

- ▶ Remember that we want our counters state to be represented in our JSX. How do we do this?

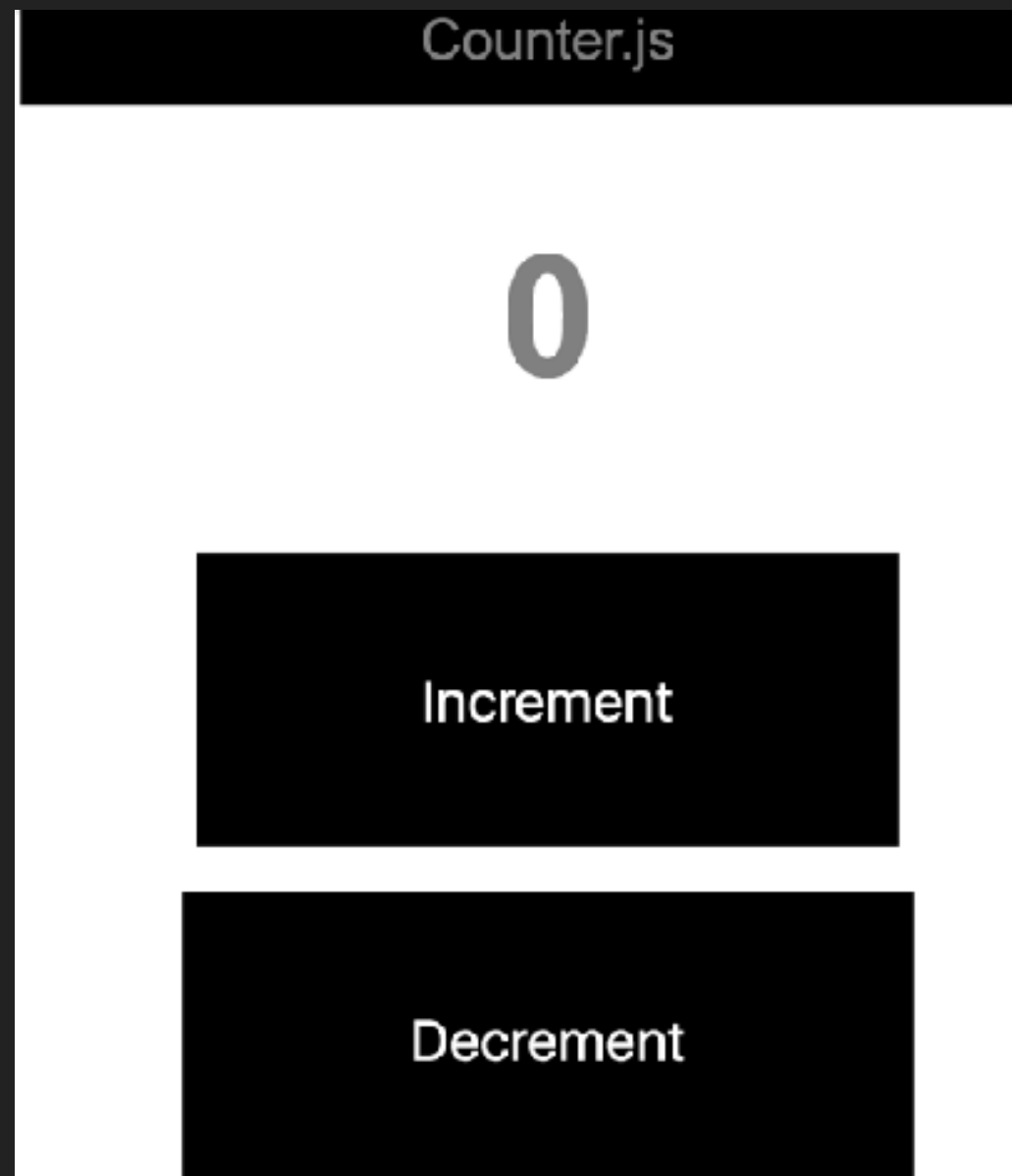
```
render() {  
  return (  
    <div className="container">  
      <div className="navbar">Counter.js</div>  
      <div className="counter">  
        <h1>{this.state.count}</h1>  
        <button type="button">Increment</button>  
        <button type="button">Decrement</button>  
      </div>  
    </div>  
  );  
}
```

When adding javascript into our JSX. We start by using open and closing brackets {}. This is a very powerful feature of React that allows us to add Javascript in our JSX.

Note: Now look how we add the state in-between our h1 tags. The count should be displayed and represent our initial value of 0.


COUNTER APPLICATION

- ▶ Now that we have some JSX built we need to add in the CSS.
- ▶ Copy in the CSS from index.css in the workshop-1 folder or from <https://github.com/Voodoobrew/workshop-1/tree/master/workshop-1/CSS>
- ▶ Then copy it into the index.css file inside your counter project.



This is how your application should look.

- ▶ Now we need to increment the counter by clicking on the increment button.
- ▶ To do this we need to first make an increment method. Methods that will be added to our class component we will put below our constructors code block and before the render method.

```
13   increment = () => {  
14      //your code goes here  
15  }  
16  };
```

- ▶ Remember how we change our state? We accomplish this by calling the `setState()` method when we click the increment button.

```
11  [-]  · · increment = () => {  
12  [-]  · · · · this.setState ({  
13      · · · · · · count: this.state.count + 1  
14      · · · · · })  
15      · · }
```

COUNTER APPLICATION

- ▶ Now that we have our increment method. How do we make sure that every time the increment button is clicked that this method will be evoked?
- ▶ We do this by adding an onClick listener to the increment button. When this button is clicked the increment method will be invoked, and the counter will display the new count.

```
<button type="button" onClick={this.increment}>Increment</button>
```

COUNTER APPLICATION

- ▶ Now, with your newfound knowledge of React, and how to create methods. Implement the decrement method on your own. Also, make sure that our counter cannot go below 0 and above 20.
- ▶ Create a button that when clicked will clear the counter back to 0.
- ▶ Make a button that will toggle between increments of 1 or 2 at a time, and will display "Double Increments" or "Single Increments" in the button name.

- ▶ 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>

<https://reactjs.org/tutorial/tutorial.html>