

SWE 432 -Web Application Development

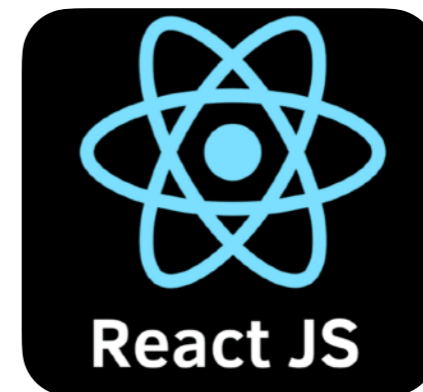
Spring 2023



George Mason
University

Dr. Kevin Moran

Week 10: Introduction to React





Administrivia

- Midterm - Grades will be posted by next Tuesday.
- HW Assignment 2 - Grades will be posted Thursday! (Sorry for Delay)
- HW Assignment 3 - Instructions out now, Tutorial out Friday, Due April 11th, before class



HW Assignment 3

Step 1: Sign up on GitHub Classroom to Clone the Starter Project

Please follow the instructions for setting up this homework assignment in GitHub Classroom and deployment of your project via Heroku. The starter project includes code for both a React Front-End and an Express back-end. You may reuse your code from HW2 for your backend to satisfy requirements for this assignment.

[Click Here to View HW 3 Tutorial](#)

Step 2: Use Persistence to Store Data Gathered through Fetch

In your node backend, you will again gather data from an external API through fetch. In this HW, rather than store data as a local variable in memory, you will instead use Firebase to persist data. You should take data generated by interacting with your 3rd party API, store the data in Firebase, and then retrieve this data later in order to handle requests made to your microservice.



HW Assignment 3

Step 3: Using React as a Template Engine to Display Data

In this step, you will build a simple frontend to display data from your microservice. Your frontend will be organized into several React components.

For example, for a cities web app, you might have an index.html page that displays a list of all cities and their names. Clicking on a single city loads a second view displaying data for that city. Clicking a link for weather on that page loads a third view displaying current weather data for that city.

Requirements:

- Use fetch to retrieve a dataset from a remote web service.
 - Data should be persisted (see Persistence below) so that the same data is only retrieved from the remote web service once during the lifetime of your microservice.
 - You should handle at least one potential error generated by the third-party API.
- Endpoints
 - Include at least 2 GET endpoints.
- Persistence
 - Within your node backend, ensure that all state which is reused across different HTTP requests is persisted into a Firebase datastore.
 - Within your node backend, retrieve data from your Firebase datastore to handle client requests.
- HTML
 - Create at least 3 separate React components corresponding to different views. These may be structured as a single page or as 3 separate pages.
 - Use at least 3 different semantic markup elements, for example:

```
1 <nav><article><aside><section><figcaption><address><cite><strong><abbr>
```

- React
 - On each of 3 views, after the page loads, use fetch to retrieve appropriate data from your backend and then use React to generate HTML for this data.



Class Overview

- This Week - **Part 1:** An Introduction to React Programming
- Next Week - **Part 2:** More React and Hands-On with the React Tutorial (In-Class Activity).

Review: HTML: HyperText Markup Language

- Language for describing *structure* of a document
- Denotes hierarchy of elements
- What might be elements in this document?





HTML Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css" />
<title>Prof Moran's Webpage</title>
</head>
<body>

<br>
<div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
<h2>Welcome, students!</h2>
<p> <a href="https://www.youtube.com/watch?v=d0w4w9WgXW">See how to make this
page</a> </p>
<h2> Some funny links </h2>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
<p> <h3> About Prof Moran </h3>
<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>
<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Use `<h1>`, `<h2>`, ..., `<h5>` for headings

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999





HTML Example

```

<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
<title>Prof Moran's Webpage</title>
</head>
<body>
<h1> Prof Kevin Moran </h1>
<div>
<p> 
<br>
<div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
</div>
</body>
</html>

```

Paragraphs (<p>) consist of related content. By default, each paragraph starts on a new line.

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999





HTML Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
<title>Prof Moran's Webpage</title>
</head>
<body>
<h1> Prof Kevin Moran </h1>
<div>
<p> 
<br>
<div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
<h2>Welcome, students!</h2>
<p> <a href="https://www.youtube.com/watch?v=d0w4w9WgXW">See how to make this
```

Unordered lists () consist of list items () that each start on a new line. Lists can be nested arbitrarily deep.

```
<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>
```

```
<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

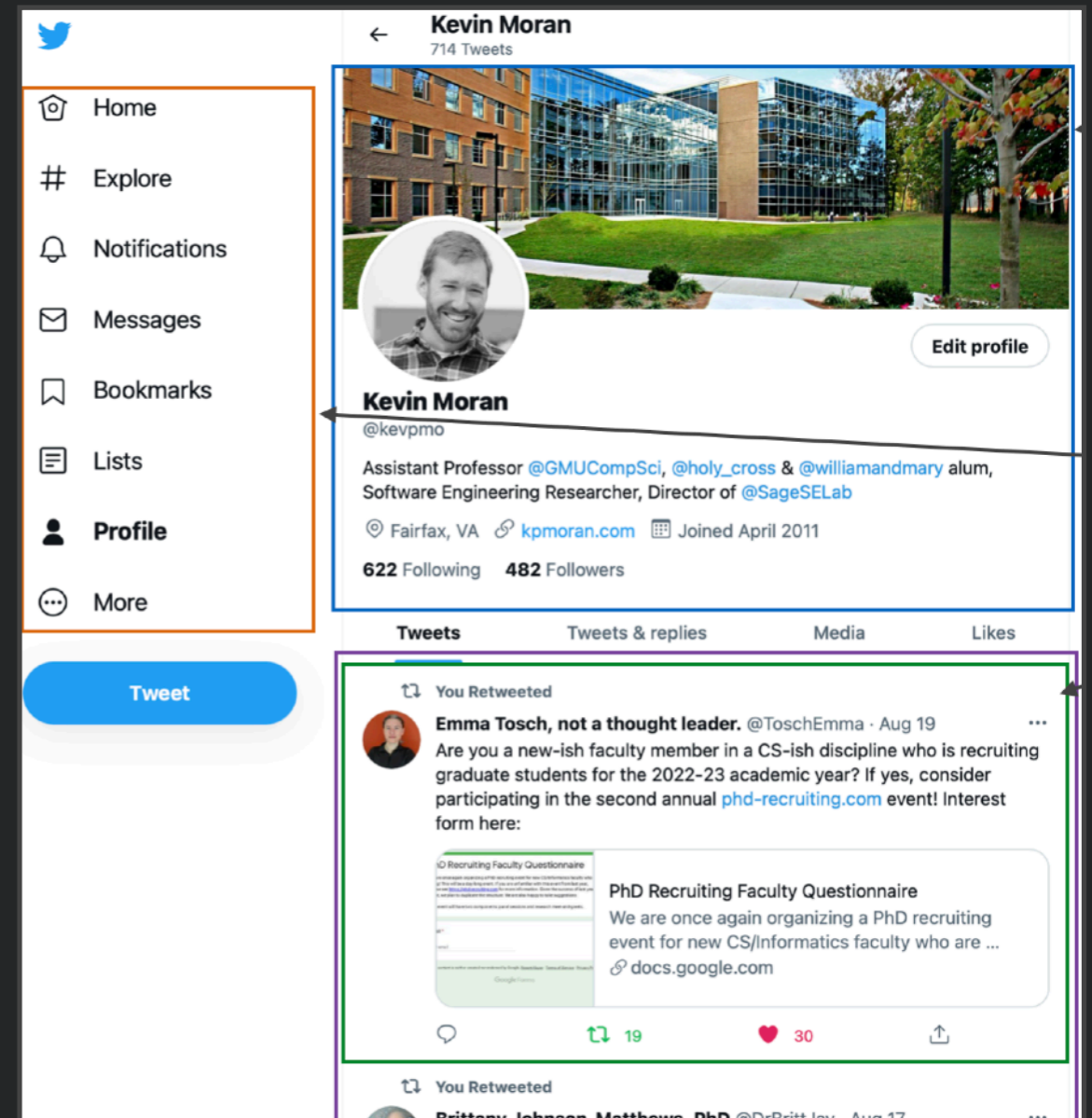
Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999



Review: Components

- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
 - Templates that correspond to a specific widget
 - Encapsulates related logic & presentation using language construct (e.g., class)





Anatomy of a React Component

```
class Toggle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {isToggleOn: true};  
  
    // This binding is necessary to make `this` work in the callback  
    this.handleClick = this.handleClick.bind(this);  
  }  
}
```

```
handleClick() {  
  this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));  
}
```

```
render() {  
  return (  
    <button onClick={this.handleClick}>  
      {this.state.isToggleOn ? 'ON' : 'OFF'}  
    </button>  
  );  
}
```

```
ReactDOM.render(  
  <Toggle />, document.getElementById('root')  
);
```




Reacting to change

- What happens when state of component changes?
 - e.g., user adds a new item to list
- Idea
 1. Your code updates `this.state` of component when event(s) occur (e.g., user enters data, get data from network) using `this.setState(newState)`
 2. Calls to `this.setState` *automatically* cause *render* to be invoked by framework
 3. Reconciliation: Framework *diffs* output of *render* with *previous* call to *render*, updating only part of DOM that *changed*



What is state?

- All internal component data that, when changed, should trigger UI update
 - Stored as single JSON object `this.state`
- What isn't state?
 - Anything that could be computed from state (redundant)
 - Other components - should build them in render
 - Data duplicated from properties.



Properties vs. State

- Properties should be *immutable*.
 - Created through attributes when component is instantiated.
 - Should *never* update within component
 - Parent may create a new instance of component with new properties

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

- State *changes* to reflect the current state of the component.
 - Can (and should) change based on the current internal data of your component.



Working with State

- Constructor should initialize state of object

```
constructor(props) {  
  super(props);  
  this.state = {date: new Date()};  
}
```

- Use this.setState to update state

```
this.setState({  
  date: new Date()  
});
```

- Doing this (asynchronously) will eventually result in render being invoked
 - Multiple state updates may be batched together and result in a single render call (handled by the framework)



Partial State Updates

- State is an object, may contain whatever properties you add
- Can use `setState` to update only parts of state you'd like to update

```
fetchPosts().then(response => {
  this.setState({
    posts: response.posts
  });
});

fetchComments().then(response => {
  this.setState({
    comments: response.comments
  });
});
```



Handling Events

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```




Handling Events

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within
  handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

Event Dispatching

- Each event target can have (0...n) listeners registered for any given event type, called in arbitrary order
- What happens with nested elements?

body

form

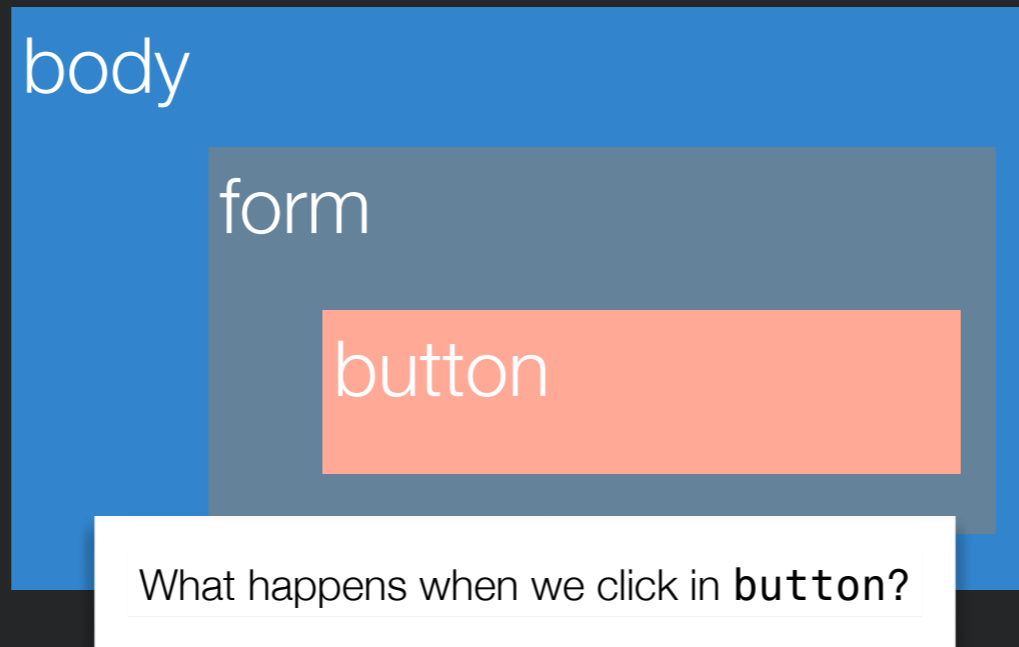
button

```
Listener1: body onClick  
Listener2: form onClick  
Listener3: button onClick
```

What happens when we click in `button`?



Event Bubbling



Called



- Listener1: body onClick
- Listener2: form onClick
- Listener3: button onClick

This is the default behavior



Event Capturing

body

form

button

What happens when we click in **button**?

Called



Listener1: body onClick

Listener2: form onClick

Listener3: button onClick

Enable event capturing when you register your listener:
`element.addListener('click', myListener, true);`

Event Dispatching

- An individual listener can stop bubbling/capturing by calling
- `event.stopPropagation();`
 - Assuming that `event` is the name of your handler's parameter

body

form

button

```
Listener1: body onClick  
Listener2: form onClick  
Listener3: button onClick
```




Nesting Components

- UI is often composed of nested components
 - Like containers in HTML, corresponds to hierarchy of HTML elements
 - But...now each element is a React component that is generated
- Parent *owns* instance of child
 - Occurs whenever component instantiates other component in render function
 - Parent configures child by passing in properties through attributes



The Data Flows Down

- State that is common to multiple components should be owned by a common ancestor
 - State can be passed into descendants as properties
- When this state can be manipulated by descendants (e.g., a control), change events should invoke a handler on common ancestor
 - Handler function should be passed to descendants

<https://reactjs.org/docs/state-and-lifecycle.html#the-data-flows-down>



The Data Flows Down

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;

    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />
      </div>
    );
  }
}
```



Nesting components

```
render() {  
  return (  
    <div>  
      <PagePic pagename={this.props.pagename} />  
      <PageLink pagename={this.props.pagename} />  
    </div>  
  );  
}
```

Establishes ownership by creating
in render function.

Sets pagename property of child to
value of pagename property of
parent



Single Page App

- In a single page app, there is only one single HTML page loaded by browser.
- When new views are opened by user or data arrives from server, client side JavaScript code generates new views.



What's wrong with this code?

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
    this.interval = setInterval(() => this.tick(), 1000);
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  render() {
    return (
      <div> Seconds: {this.state.seconds} </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```



Component Lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```



Component Lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}
```

```
ReactDOM.render(<Timer />, mountNode);
```

ReactDOM.render(...)
[component created]
constructor(...)
render()
componentDidMount()

tick()
render()

...

[component rendered
again by parent]
componentWillUnmount()
[component created]

...



Controlled Components

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```



Controlled Components

- Single source of truth
- Whenever a control changes its value
 - React is notified
 - State is updated
- Whenever state is updated
 - If necessary, render function executes and generates control with new value

SWE 432 - Web Application Development



George Mason
University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
Oyindamola Oluyemo

Class will start in:
10:00



Reconciliation

```
<Card>  
  <p>Paragraph 1</p>  
  <p>Paragraph 2</p>  
</Card>
```

```
<Card>  
  <p>Paragraph 2</p>  
</Card>
```

- Process by which React updates the DOM with each new render pass
- Occurs based on order of components
 - Second child of Card is destroyed.
 - First child of Card has text mutated.

<https://reactjs.org/docs/reconciliation.html>



Reconciliation with Keys

- Problem: what if children are dynamically generated and have their own state that must be persisted across render passes?
 - Don't want children to be randomly transformed into other child with different state
- Solution: give children identity using keys
 - Children with keys will always keep identity, as updates will reorder them or destroy them if gone

Keys



```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```



Todo in React

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { items: [], text: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  render() {
    return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input
            onChange={this.handleChange}
            value={this.state.text}
          />
          <button>
            Add #{this.state.items.length + 1}
          </button>
        </form>
      </div>
    );
  }
}
```

```
  handleChange(e) {
    this.setState({ text: e.target.value });
  }

  handleSubmit(e) {
    e.preventDefault();
    if (!this.state.text.length) {
      return;
    }
    const newItem = {
      text: this.state.text,
      id: Date.now()
    };
    this.setState(prevState => ({
      items: prevState.items.concat(newItem),
      text: ''
    }));
  }
}
```

```
class TodoList extends Component {
  render() {
    return (
      <ul>
        {this.props.items.map(item => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
```




Functional Components

- What happens if you have a simple component that only has properties and no state.
- Can use the function syntax to create a component.
- Component that is only a render method.

```
function Square(props) {  
  return (  
    <button className="square" onClick={props.onClick}>  
      {props.value}  
    </button>  
  );  
}
```



Functional Components + Hooks

But what if we want state + clean functional components??

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

class Counter extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
          onClick={() =>
            this.setState({ count: this.state.count + 1 })
          }
        >
          Click me
        </button>
      </div>
    );
  }
}

export default Counter;
```



Functional Components + Hooks

Now we can have both with functional components + hooks!

```
import React from 'react';


// how to use the state hook in a React function component
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

In Class Activity

- Working through the React Tutorial in pairs:
 - Visit <https://reactjs.org/tutorial/tutorial.html>
- 
- You will submit your finished tutorials here (for quiz grade):
 - <https://go.gmu.edu/react-tutorial-sub>
 - If we have time, I will ask a team to present their tutorial result at the end of class.



Acknowledgements

- Slides Adapted from Dr. Thomas LaToza's SWE-432 course