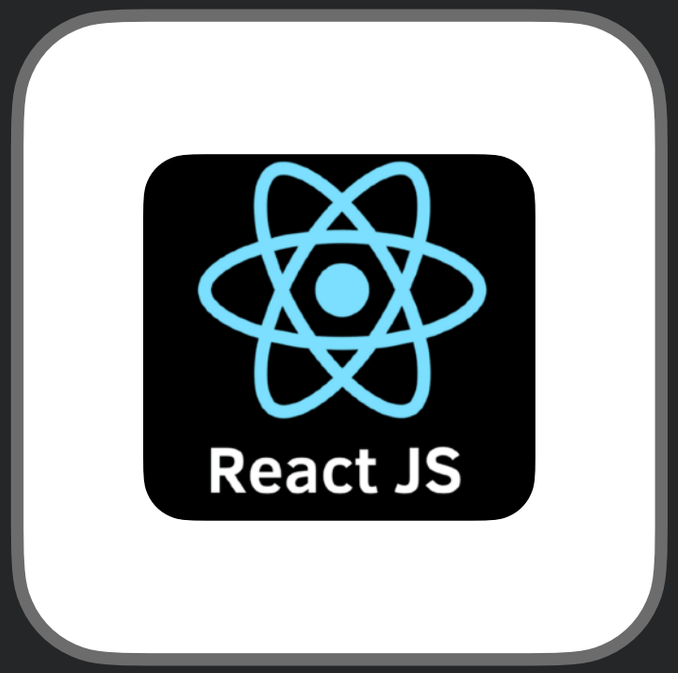
SWE 432 -Web Application Development

Fall 2022

George Mason University

Dr. Kevin Moran

Week 9: Introduction to React, Part 2



Administrivia

- *Midterm* - Grades will (actually) be posted today (Sorry for delay).

- *HW Assignment 2 -* Grades posted - please check!

- *HW Assignment 3* - Out now, Due October 27th (in one *week!*), before class

Class Overview

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

Quick Review of Key React Concepts

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')
);
```

https://reactjs.org/docs/handling-events.html

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.

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 manipulated by descendants (e.g., a control), change events should invoke a handler on common ancestor
  - Handler function should be passed to descendents

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

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

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

- 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

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>
    );
  }
```

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>
    );
  }
```

https://codesandbox.io/s/adoring-ishizaka-bf3ym?fontsize=14

```
  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!

```jsx
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;
```

```jsx
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://bit.ly/3EYcTpB
- 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