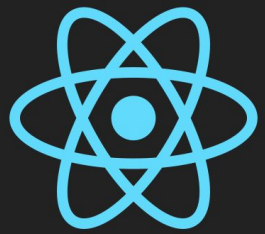
Lecture 3.0

Handing Events



Topics

- Handling Events
  - DOM Events vs React Events
  - e.PreventDefault
  - Event Binding
  - Passing Arguments in Event Handlers

Events

Handling Events

- React is at it best in highly interactive user interfaces that deals with events

- Handling events with React elements is very similar to handling events on DOM elements. There are some syntactic differences:
  - React events are named using camelCase, rather than lowercase.
  - With JSX you pass a function as the event handler, rather than a string.

HTML

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

React

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

DOM Events

- DOM Events are the events generated by the browser. All browsers provide an event based programming model.
- React provides a normalized event abstraction called SyntheticEvents.
- DOM Events are the events that receive a react SyntheticEvents object (e)

You cannot return false to prevent default behavior in React!

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me
</a>
```

Prevent Default

- Another important difference, is that you cannot return false to prevent default behavior in React. You must call preventDefault() explicitly.

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

Component Events

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // This syntax ensures `this` is bound within handleClick
    return (
      <button onClick={(e) => this.handleClick(e)}>
        Click me
      </button>
    );
  }
}
```

- A common pattern for class components is for an event handler to be a method on the class

```jsx
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(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

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

- You have to be careful about the meaning of this in JSX callbacks.
- In JavaScript, class methods are not bound by default.
- If you forget to bind this.handleClick and pass it to onClick, this will be undefined when the function is actually called.

Event Binding in Class Components

If you don't want to use bind, you can wrap the event callback in an arrow function!

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // This syntax ensures `this` is bound within handleClick
    return (
      <button onClick={() => this.handleClick()}>
        Click me
      </button>
    );
  }
}
```

Passing Arguments to Event Handlers

- These two calls are equivalent, one uses the arrow function and the other Function.prototype.bind

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

- e argument represents the React "synthetic" event in both cases
- With an arrow function, we need to pass it explicitly
- With bind, any further arguments will be automatically forwarded

Video - Events

https://youtu.be/OcM__8q6p4c