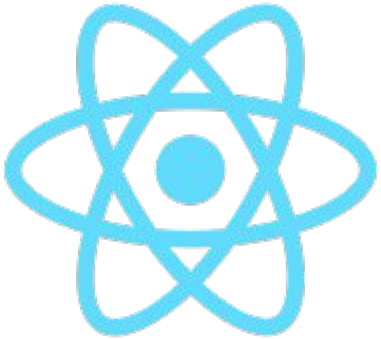
React Events



No lifecycle here

Events

Events

- In the real world, you often need to update the state of your components based on user interactions.

- This is where events come in.

- By adding event handling methods to your components, you can respond to user actions and update the state accordingly.

Events - Example

```
class EventExample extends Component {
  save() {
    alert("clicked save!");
  }
  render() {
    return (
      <div>
        <button onClick={this.save}>Save</button>
      </div>
    );
  }
}
```

Function Binding

- Be careful when your event functions are using “this”
- React in ES6 does not automatically bind your functions for event handlers. Any event handlers need to be bound with one of two ways
  1. Using the constructor function
  2. Using an arrow function

Forms

- The different kinds of form components are 'input', 'option', and 'textarea'.

- React form components are special in that their property values can be changed by the user as well as the code.

- We call these special props "Interactive Props".

Forms - Interactive Props

- ‘value’: This prop is supported by ‘<input>’ and ‘<textarea>’ components. It allows you to set the initial value of the input or textarea, and it also represents the current value entered by the user.

- ‘checked’: This prop is supported by ‘<input>’ components of type checkbox or radio. It allows you to control whether the checkbox or radio button is checked or not.

- ‘selected’: This prop is supported by ‘<option>’ components. It allows you to specify which option should be selected by default in a ‘<select>’ dropdown.

Controlled Components

- Form components with the "value" property defined require an "onChange" handler to track user modifications.

- Defining the "value" property allows you to set the initial or current value of the form element.

- By providing an "onChange" handler, you can be notified whenever the user modifies the form element, enabling synchronization between the component's state and the user's input.

Controlled Component - Example

```
class FormTest extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "David",
    };

    this.handleNameChange = this.handleNameChange.bind(this);
  }
  render() {
    return (
      <div>
        <input
          type="text"
          value={this.state.name}
          onChange={this.handleNameChange}
        />
      </div>
    );
  }
  handleNameChange(event) {
    this.setState({
      name: event.target.value,
    });
  }
}
```

Uncontrolled Component

- For form components that do not have a "value" property defined, you can use the "defaultValue" property to initialize the component with a specific value.

- The "defaultValue" property allows you to set an initial value for the form component without explicitly tracking its changes.

- Unlike the "value" property, changes made by the user will not be reflected in the component's state automatically when using "defaultValue". If you want to capture user modifications, you will still need to use the "onChange" handler.

- By using "defaultValue", you can provide an initial value for the form component without the need for two-way data binding.

Uncontrolled Component - Example

```
<select value="CA" onChange={this.handleSelectChange}>
  <option value="CA">California</option>
  <option value="FL">Florida</option>
  <option value="NY">New York</option>
</select>;
```