Bindings

Binding comes in very handy while creating React apps. You'll see how by as you go read this lesson.

It is important to learn about bindings in JavaScript classes when using React ES6 class components. In the previous chapter, you have bound your class method `onDismiss()` in the constructor.

```
class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list,
    };

    this.onDismiss = this.onDismiss.bind(this);
  }

  ...
}
```

The binding step is necessary because class methods don't automatically bind `this` to the class instance. Let's demonstrate it with the help of the following ES6 class component:

```
class ExplainBindingsComponent extends Component {
  onClickMe() {
    console.log(this);
  }

  render() {
    return (
      <button
        onClick={this.onClickMe}
        type="button"
      >
        Click Me
      </button>
    );
  }
}
```

The component renders just fine, but when you click the button, you see `undefined` in your developer console log. This is one of the main sources of bugs developers encounter in React. If you want to access `this.state` in your class method, it cannot be retrieved because `this` is `undefined`. To make `this` accessible in your class methods, you have to bind the class methods to `this`.

In the following class component the class method is properly bound in the class constructor:

```
class ExplainBindingsComponent extends Component {
  constructor() {
    super();

    this.onClickMe = this.onClickMe.bind(this);
  }

  onClickMe() {
    console.log(this);
  }

  render() {
    return (
      <button
        onClick={this.onClickMe}
        type="button"
      >
        Click Me
      </button>
    );
  }
}
```

Class method binding can happen somewhere else too. For instance, it can happen in the `render()` class method:

```
class ExplainBindingsComponent extends Component {
  onClickMe() {
    console.log(this);
  }

  render() {
    return (
      <button
        onClick={this.onClickMe.bind(this)}
        type="button"
      >
        Click Me
      </button>
    );
  }
}
```

Avoid this practice, however, because it binds the class method every time the `render()` method runs, meaning every time the component updates, which will hurt your application’s performance eventually. Binding the class method in the constructor need only be done once, when the component is instantiated.

Some developers will define the business logic of their class methods in the constructor:

```
class ExplainBindingsComponent extends Component {
  constructor() {
    super();

    this.onClickMe = () => {
      console.log(this);
    }
  }

  render() {
    return (
      <button
        onClick={this.onClickMe}
        type="button"
      >
        Click Me
      </button>
    );
  }
}
```

Avoid this approach as well, as it will clutter your constructor over time. The constructor is only there to instantiate your class with all its properties, so the business logic of class methods should be defined outside the constructor.

```
class ExplainBindingsComponent extends Component {
  constructor() {
    super();

    this.doSomething = this.doSomething.bind(this);
    this.doSomethingElse = this.doSomethingElse.bind(this);
  }

  doSomething() {
    // do something
  }

  doSomethingElse() {
    // do something else
  }

  ...
}
```

Class methods can be auto-bound using JavaScript ES6 arrow functions:

```
class ExplainBindingsComponent extends Component {
  onClickMe = () => {
    console.log(this);
  }

  render() {
    return (
      <button
        onClick={this.onClickMe}
        type="button"
      >
        Click Me
      </button>
    );
  }
}
```

Use this method if the repetitive binding in the constructor annoys you. The official React documentation sticks to the class method bindings in the constructor, so this course will stick with those as well.

Exercises:

- Try different approaches of bindings and console log the `this` object

Further Reading:

- Learn more about an alternative React component syntax