Method 2: componentDidCatch

Let's understand the purpose of the componentDidCatch method.

WE'LL COVER THE FOLLOWING

- Adding the `componentDidCatch` Method

The `componentDidCatch` method is called after an error in a descendant component is thrown. Apart from the error thrown, it is passed one more argument which represents more information about the error:

```
componentDidCatch(error, info) {
}
```

In this method, you can send the `error` or `info` received to an external logging service. Unlike `getDerivedStateFromError`, the `componentDidCatch` allows for side-effects:

```
componentDidCatch(error, info) {
        logToExternalService(error, info) // this is allowed.
        //where logToExternalService may make an API call.
}
```

Adding the `componentDidCatch` Method

Let's update the `ErrorBoundary` component to use this lifecycle method:

```
import React, { Component } from "react";
class ErrorBoundary extends Component {
  state = { hasError: false };
  static getDerivedStateFromError(error) {
    console.log(`Error log from getDerivedStateFromError: ${error}`);
    return { hasError: true };
  }
  componentDidCatch(error, info) {
    console.log(`Error log from componentDidCatch: ${error}`);
    console.log(info);
```

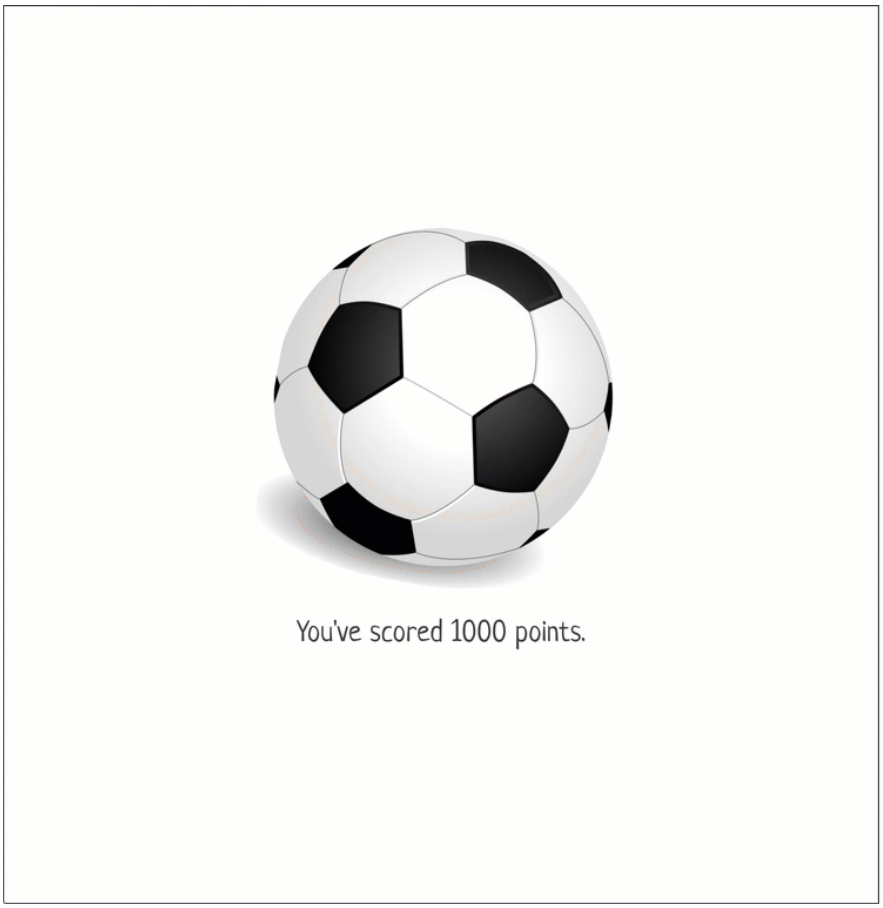
```
  }
  render() {
    return null
  }
}
export default ErrorBoundary;
```

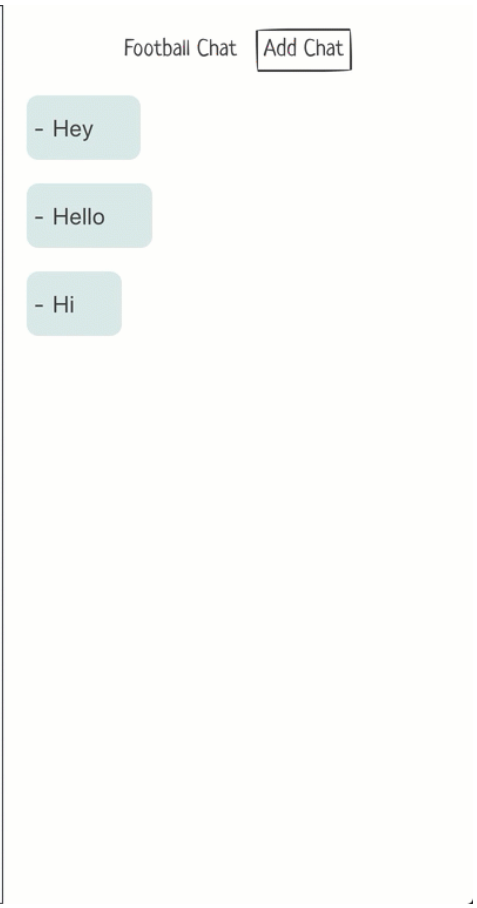
Since `ErrorBoundary` can only catch errors from descendant components, we'll have the component render whatever is passed as `Children` or render a default error UI if something went wrong:

```
...
render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
 }
```

I have simulated a JavaScript error whenever you add a 5th chat message. Have a look at the `ErrorBoundary` at work:





Here's the implementation of the app so far:

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

```
class ErrorBoundary extends Component {
  state = { hasError: false };
  static getDerivedStateFromError(error) {
    console.log(`Error log from getDerivedStateFromError: ${error}`);
    return { hasError: true };
  }
  componentDidCatch(error, info) {
    console.log(`Error log from componentDidCatch: ${error}`);
    console.log(info);
  }
    render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}
export default ErrorBoundary;
```

Let’s move on to the conclusion of this project in the next lesson.