Class-Based Components

An Alternative Way Of Building Components

- What & Why?
- Working with Class-based Components
- Error Boundaries

Class Components vs Functional Components

Functional Components

```
function Product(props) {
  return <h2>A Product!</h2>
}
```

Components are regular JavaScript functions which return renderable results (typically JSX)

The default & most modern approach!

Class-based Components

```
class Product extends Component {
  render() {
    return <h2>A Product!</h2>
  }
}
```

Components can also be defined as JS classes where a render() method defines the to-be-rendered output

Was required in the past

when using React prior to version 16.8

Traditionally, you had to use class-based components to manage "State"

ACADEMIND

React 16.8 introduced "**React Hooks**" for **functional** components

ACADEMIND

Class-based components can't use React Hooks

Class Components Lifecycle

Side Effects in **Functional** Components: **useEffect()**

Class-based components can't use Hooks!

`componentDidMount()`	Called once a component **mounted** → evaluated & rendered by React	`useEffect(…, [])`
`componentDidUpdate()`	Called once a component **updated** → re-evaluated & re-rendered by React	`useEffect(…, [someValue])`
`componentWillUnmount()`	Called right before component is **unmounted** → right before removed from DOM	`useEffect(() => { return () => { … } }, [])`

You Don't Have To Use Functional Components!

You can use class-based components if you prefer them (though it's really not necessarily recommended...)

Which Component Type Should You Use?

Strong Recommendation: You should prefer **functional components!**

Use class-based if …

- …you **prefer** them
- …you're working on an existing project or in a team where they're getting used
- …you're building an Error Boundary