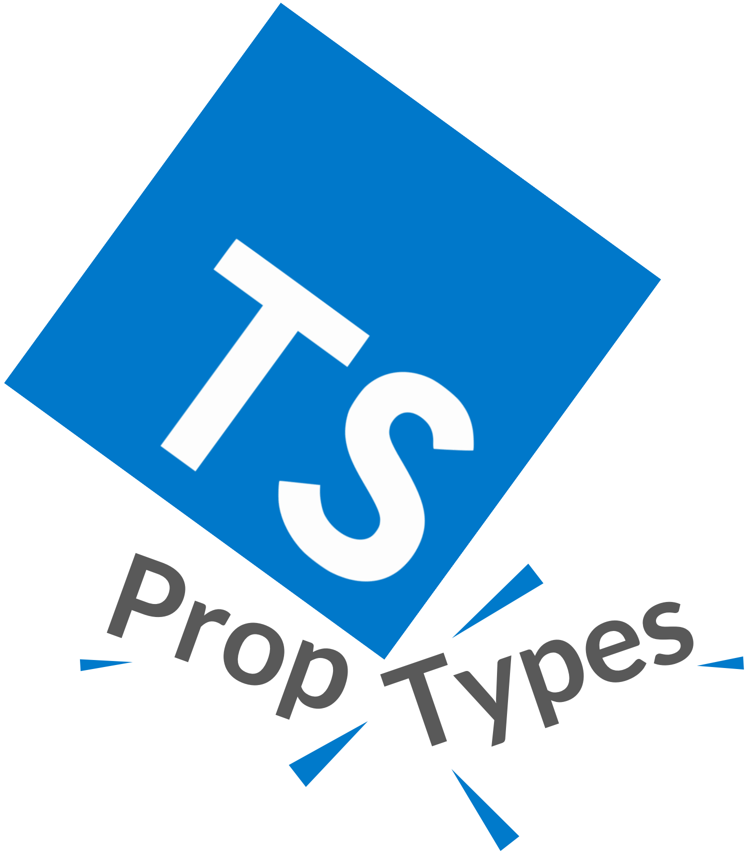
Evolution Engineering

Advanced React

Bootcamp [TypeScript]

PropTypes



https://reactjs.org/docs/typechecking-with-proptypes.html

Context

ℹ️ Context is designed to share data that can be considered "global" for a tree of React components.

✅ When to use:

- Prop drilling avoiding.
- If you have static data that undergoes lower-frequency updates such as preferred language, time changes, location changes, and user authentication, passing down props with React Context may be the best option.

🚫 When not to use:

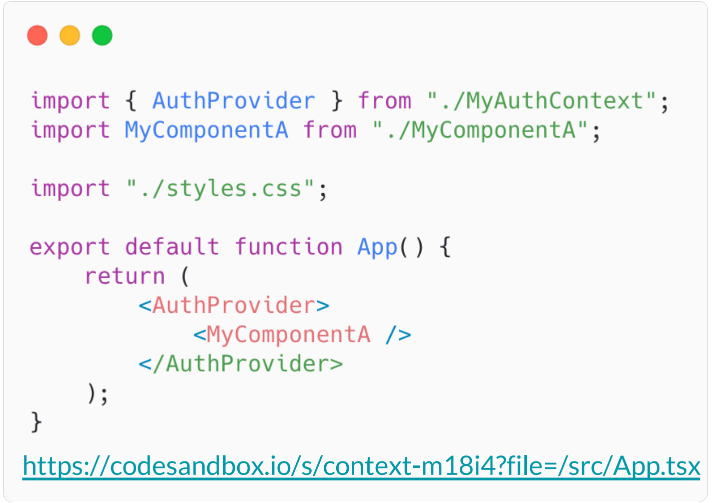
- If your state is frequently updated, React Context may not be as effective or efficient as for example a tool like React Redux.

https://reactjs.org/docs/context.html

Context

➡️ [React.createContext()]

1. Context.Provider
2. Context.Consumer or Class.contextType (static contextType)



```
import { AuthProvider } from "./MyAuthContext";
import MyComponentA from "./MyComponentA";

import "./styles.css";

export default function App() {
    return (
        <AuthProvider>
            <MyComponentA />
        </AuthProvider>
    );
}
```

https://codesandbox.io/s/context-m18i4?file=/src/App.tsx

https://reactjs.org/docs/context.html

Refs

ℹ️ Refs provide a way to access DOM nodes or React elements created in the render method.

✅ When to use:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

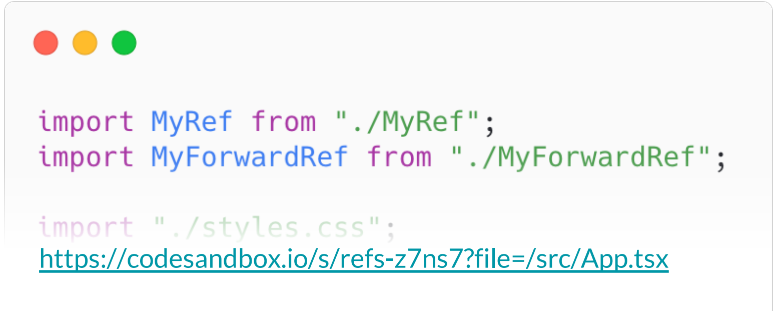
🚫 When not to use:

- Avoid using refs for anything that can be done declaratively.

https://reactjs.org/docs/refs-and-the-dom.html

Refs

- ➡️ **React.createRef()**
  - The **ref prop** is used to return a reference to the element.
  - When a **ref** is passed to an element in render, a reference to the node becomes accessible at the current attribute of the ref.
- ➡️ **React.forwardRef()**
  - Ref forwarding technique is used for exposing DOM Refs to Parent Components.
  - This is generally not recommended because it breaks component encapsulation.



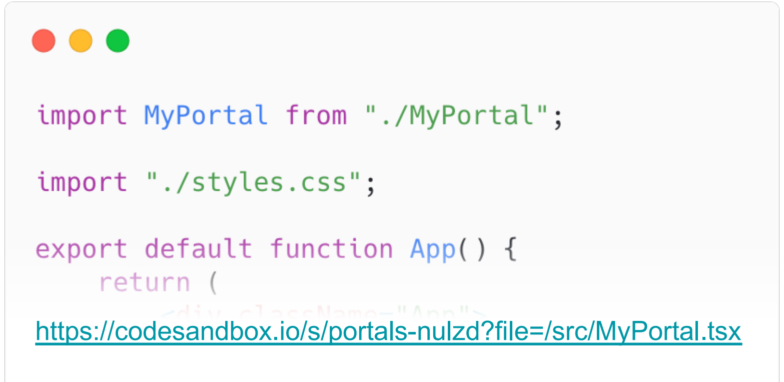
```
import MyRef from "./MyRef";
import MyForwardRef from "./MyForwardRef";

import "./styles.css";
```

https://reactjs.org/docs/refs-and-the-dom.html https://codesandbox.io/s/refs-z7ns7?file=/src/App.tsx

Portals

ℹ️ Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.
It is useful for implementing popups, toasts, tooltips, etc.

- Event Bubbling will work according to React tree ancestors, regardless of the Portal node location in the DOM.
- Context and lifecycle work the same way since the Portal still exists in the React tree.



```
import MyPortal from "./MyPortal";

import "./styles.css";

export default function App() {
    return (
      <div className="App">
```

https://reactjs.org/docs/refs-and-the-dom.html

https://codesandbox.io/s/portals-nulzd?file=/src/MyPortal.tsx

Hooks

ℹ️ Hooks let you use state and other React features without writing a class. They let you “hook into” React state and lifecycle features from function components. **Hooks are made for function components.**

Motivation

- **It’s hard to reuse stateful logic between components.**
  Hooks allow you to reuse stateful logic without changing your component hierarchy.
- **Complex components become hard to understand.**
  Hooks let you split one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data)
- **Classes confuse both people and machines.**
  Hooks let you use more of React's features without classes.

✅ Hooks are easier to test.

https://reactjs.org/docs/hooks-intro.html

Rules of Hooks

ℹ️ Hooks are JavaScript functions, but they impose two additional rules:

1. **Only call Hooks at the top level.**
   Don’t call Hooks inside loops, conditions, or nested functions.
   By following this rule, you ensure that Hooks are called in the same order each time a component renders.
2. **Only call Hooks from React function components.**
   Don’t call Hooks from regular JavaScript functions.
   By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

✅ These rules might seem limiting or confusing at first, but **they are essential to making Hooks work well.**
React relies on the order in which Hooks are called.

https://reactjs.org/docs/hooks-rules.html

Basic Hooks

➡️ React.useState()

- Think of useState Hook as combination of **this.state** and **this.setState**.

➡️ React.useEffect()

- Think of useEffect Hook as **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** combined.

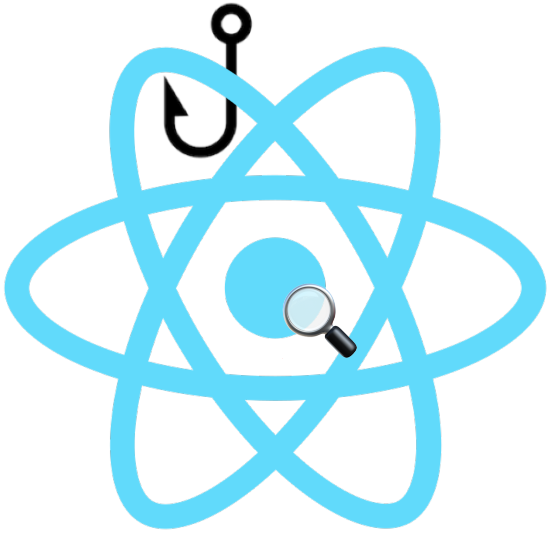
➡️ React.useContext()

- Think of useContext Hook as **Context.Consumer**.

https://reactjs.org/docs/hooks-overview.html
https://reactjs.org/docs/hooks-reference.html#basic-hooks

Basic Hooks

Let’s see...



https://codesandbox.io/s/basic-hooks-2hbbt?file=/src/App.tsx

Additional Hooks

- ➡️ React.useCallback()
    - Think of useCallback Hook as a class method.

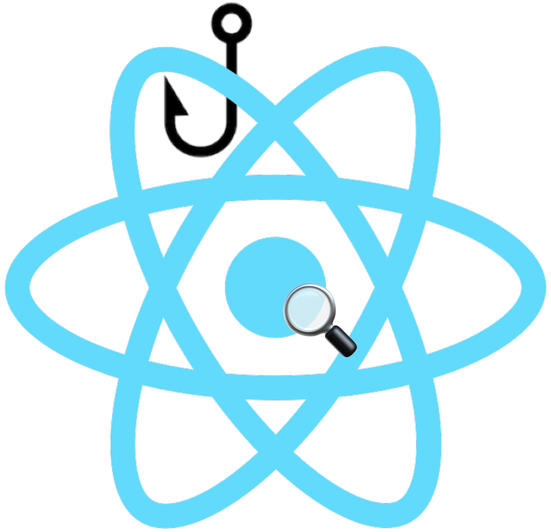
- ➡️ React.useMemo()
    - Think of useMemo Hook as **React.memo**.

- ➡️ React.useRef()
    - Essentially, useRef Hook is like a “box” that can hold a mutable value in its “.current” property.

https://reactjs.org/docs/hooks-reference.html#additional-hooks

Additional Hooks

Let's see…



https://codesandbox.io/s/additional-hooks-h2h6p?file=/src/App.tsx

Additional Materials

1. Application State Management with React: https://kentcdodds.com/blog/application-state-management-with-react
2. How to use React Context effectively: https://kentcdodds.com/blog/how-to-use-react-context-effectively

Thank you!

Advanced React ✅