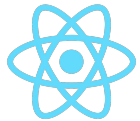
Context in React

Kemet Dugue

kemetdugue

kdugue



Agenda

1. Brief Introduction
2. React Context
    a. Props and Overall Goal of Context
    b. Introduction & Examples
    c. Best Practices and Usage Decisions
3. Resources
4. Q+A

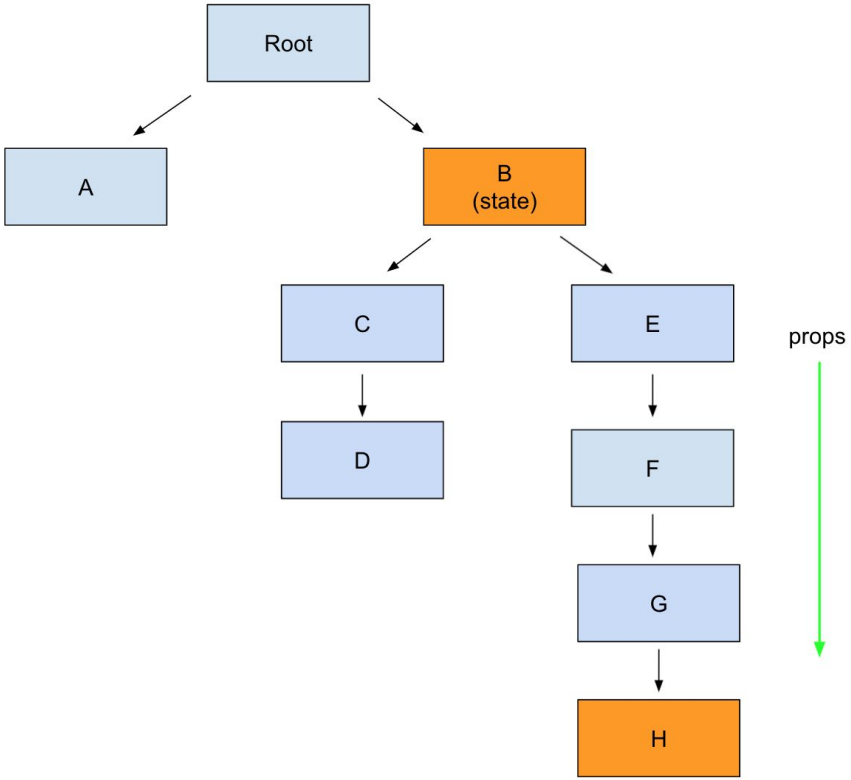
About Me

- Engineer at
- Some Interests:
  - Lo fi / chill
  - yoga

Whatever your heart desires!

Opinions and Q+A

Props and Passing Data



Prop Drilling

- Pass props through components that don't need it

- Downsides:
  - Unintended Mutations
  - Dependency on the intermediary components
  - Re-renders in intermediary components

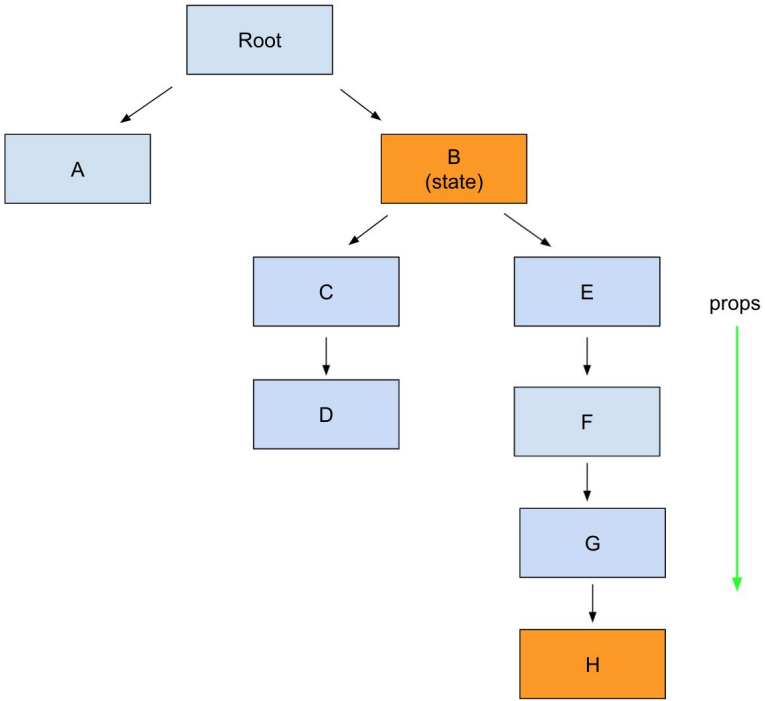
“Context provides a way to pass data through the component tree without having to pass pros down manually at every level”

- React Docs

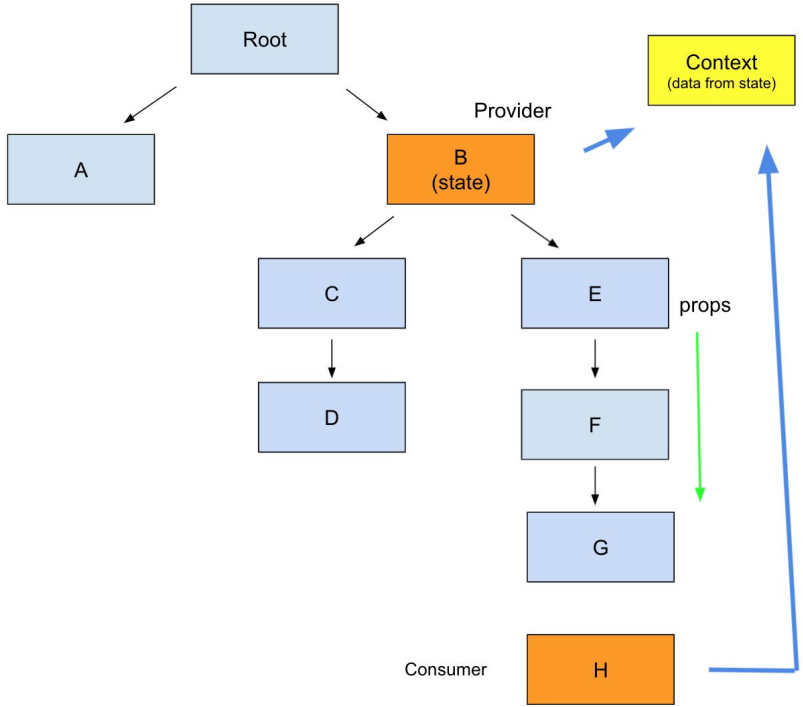
Using Context: 3 Parts

1. Context Object
    a. Creates context that will hold some information
2. Provider
    a. Declare the data in component tree that will be available to components that consume it
3. Consumer
    a. Allows components to subscribed to data from Provider

Without Context



With Context



```jsx
import React, { useContext, createContext } from "react";

export const ThemeContext = createContext("dark");

export default function App() {
  return (
    <ThemeContext.Provider value="light">
      <NavBar />
      <NewsFeed />
    </ThemeContext.Provider>
  );
}

function NavBar() {
  const theme = useContext(ThemeContext);

  return (
    <nav className={theme}>
      <a href="#home">Home</a>
      <a href="#about">About</a>
    </nav>
  );
}

function NewsFeed() {
  return <div> Lots of data</div>;
}
```

```jsx
import React, { useState, useContext, createContext } from "react";

export const ThemeContext = createContext();

export default function App() {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <NavBar />
      <NewsFeed />
    </ThemeContext.Provider>
  );
}

function NavBar() {
  const { theme, setTheme } = useContext(ThemeContext);
  const desiredTheme = theme === "light" ? "dark" : "light";

  return (
    <nav className={theme}>
      <a href="#home">Home</a>
      <a href="#about">About</a>
      <button onClick={() => setTheme(desiredTheme)}>
        Switch Theme to {desiredTheme}
      </button>
    </nav>
  );
}

function NewsFeed() {
  return <div> Lots of data</div>;
}
```

Context Use Cases

- Theming
- Localization
- User data
- Non-frequently changing data

Re-Renders and Performance

- Only components that call useContext will re-render whenever the context's state changes

- Every component that consumes a given context re-renders every time that context's state changes

Best Practices

- For Stateful Context, be sure to have the update state logic not defined in the Context itself
- Use Multiple Contexts, and keep state close to its Dependent Components

Deciding to Use Context

Not a good idea when:

- Trying to replace state or state management tool (Redux)
- Replace every instance of props passing (2-3 levels is okay)

Possibly a good idea when:

- Making a portion of your React component tree more accessible
- Static, or non-frequently changing data

Resources

- React docs on Context: https://reactjs.org/docs/context.html
- Memoizing Context: https://github.com/facebook/react/issues/15156#issuecomment-474590693
- Prop Drilling: https://kentcdodds.com/blog/prop-drilling

Questions????