SUNIL VISHWAKARMA
@linkinsunil

useContext( )

read & subscribe to context in React

Want to share information between components without prop-drilling? use useContext hook.

Lets See into

1. What problem does useContext solve
2. Example of prop drilling
3. Introduce Context to the app
4. Creating context
5. Using the context in component
6. TL;DR

swipe ⟶

Out there lot many React developers struggle with the use of useContext( ) hook in React.

Even I was unaware of what it is and how to use it for a long time but you won't be, I promise.
Just quickly follow me and smash the 🔔 on my profile to get notified first about my future posts like this one.

Lets conquer it today. Save it first. 🫠
Ready? Repost this and leave a like. 🔁 👍

Lets go...

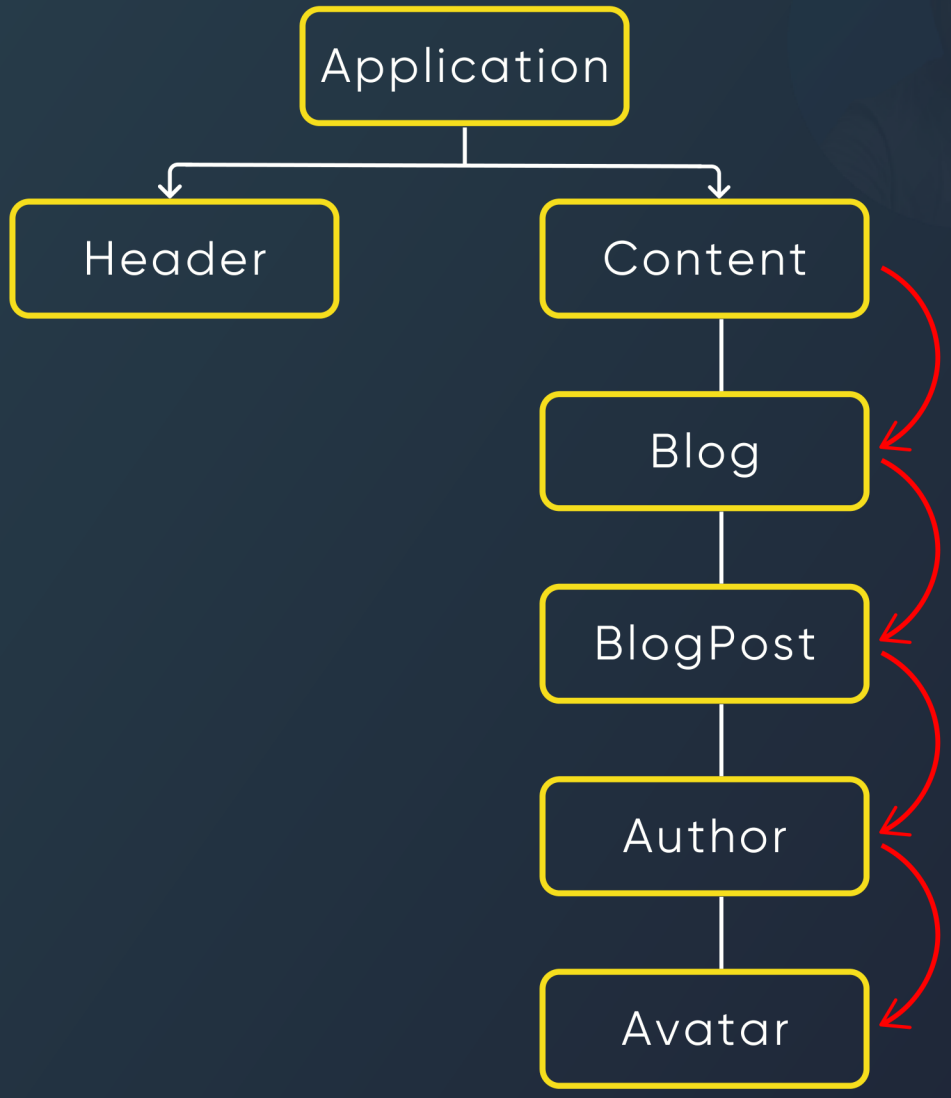
swipe ⟶

SUNIL VISHWAKARMA
@linkinsunil

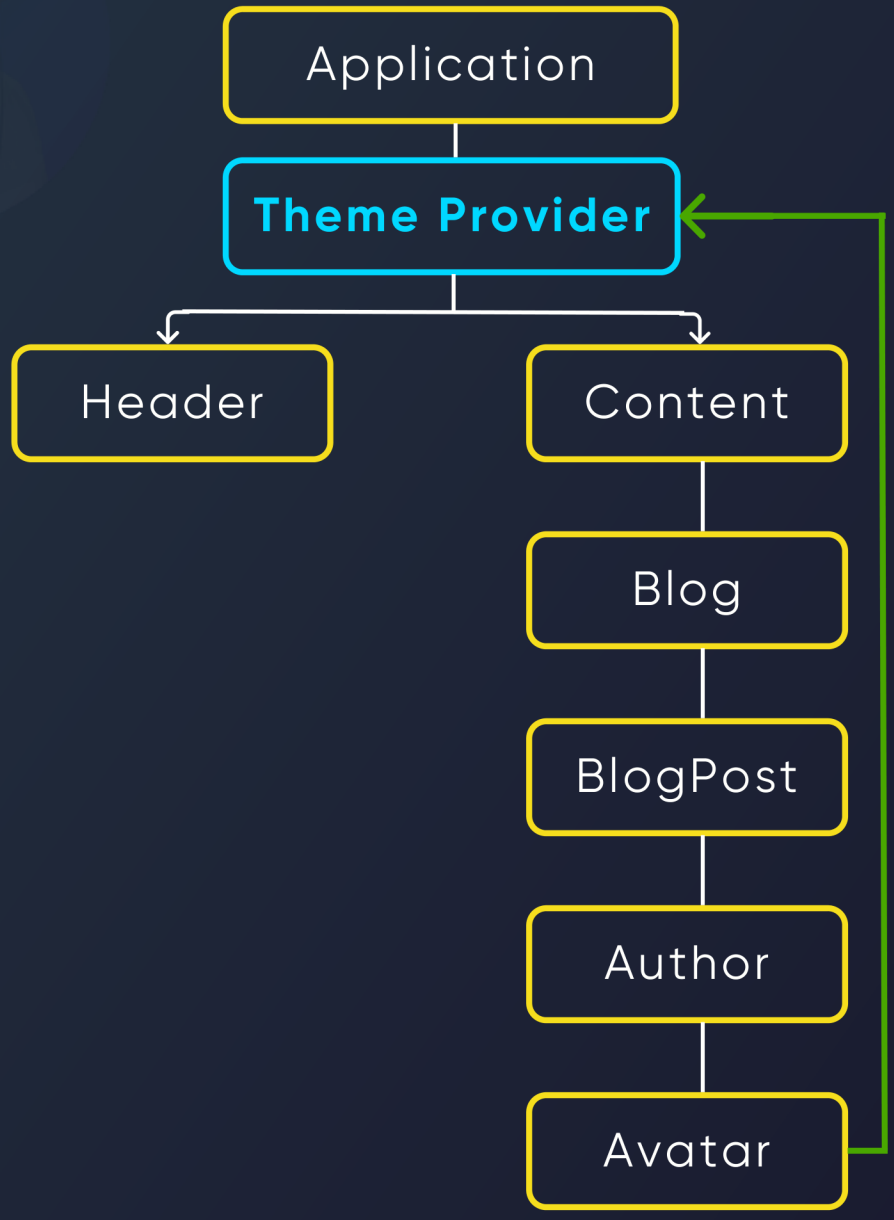
1. What Problem does useContext Solve?

Drilling props deep complicates every component and can even cause performance issues due to components re-rendering too often. This can easily happen when prop drilling down through a component tree. If a prop at the top level changes, it will cause all the components down the tree to re-render. This causes issues especially if any of these components perform expensive calculations.

Prop Drilling



Context API



swipe ⟶

2. Example Of Prop Drilling: App.js

```jsx
import { useState } from "react";

const App = () => {
 const [theme, setTheme] = useState("light");

 const toggleTheme = () => {
   setTheme(theme === "light" ? "dark" : "light");
 };

 return (
   <Content theme={theme} toggleTheme={toggleTheme}>
     <ThemeToggler theme={theme} toggleTheme={toggleTheme} />
   </Content>
 );
};
```

In this example, the theme value and toggleTheme function is passed down from the App component to the Content and ThemeToggler components.

This is an example of prop drilling. The functions are being passed through multiple components in order to be accessed by child components.

swipe ⟶

3. Introduce Context To The App

importing the useContext and createContext Hook from React:

```
import { createContext, useContext } from 'react';
```

Creating context:

```
const ThemeContext = createContext(null);
```

Call useContext at the top level of your component to read and subscribe to context.

```
const theme = useContext(ThemeContext);
```

swipe ⟶

4. Creating Context

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

const ThemeContext = createContext("light");

const ThemeProvider = ({ children }) => {
 const [theme, setTheme] = useState("light");

 const toggleTheme = () => {
   setTheme(theme === "light" ? "dark" : "light");
 };

 return (
   <ThemeContext.Provider value={{ theme, toggleTheme }}>
     {children}
   </ThemeContext.Provider>
 );
};

const useTheme = () => useContext(ThemeContext);

export { ThemeProvider, useTheme };
```

First we create our ThemeContext and set it a default value of "light". We create the ThemeProvider component to wrap our component in it. We only have to define this once and then we can access it in any component with the help of the useTheme custom hook.

swipe ⟶

Now our App.js looks like this:

```
const App = () => {

  return (
    <ThemeProvider>
      <Content>
        <ThemeToggler />
      </Content>
    </ThemeProvider>
  );
};
```

You can see there are no props passed. Instead our content is wrapped with our ThemeProvider. This provides all children components within our ThemeProvider wrapper access to our context values theme and toggleTheme.

swipe ⟶

5. Using The Context in Component

```
const ThemeToggler = () => {
  const { theme, toggleTheme } = useTheme();

  return (
    <button
      className={`${theme === "light" ? "darkMode" : "lightMode"}`}
      onClick={toggleTheme}
    >
      Toggle {theme === "light" ? "Dark" : "Light"} Mode
    </button>
  );
};
```

In our ThemeToggler, we access our theme and toggleTheme function with the useTheme custom hook. This info is being passed to a button. The button will toggle our theme back and forth between light and dark mode.

As long as the component is within ThemeProvider, we can use useTheme to access our current theme value. This saves us from passing the theme prop down from App through the whole component tree to where we need it. In this case that's just App → ThemeToggler.

swipe ⟶

6. TL;DR

Let's break down the core principles of the Context API. If you take anything away from my post, let it be this!

1. Context is a way to pass data through the component tree without having to pass props down manually from parent to child component.

2. Context is" for data that's considered "global" in the application. For example, the current authenticated user, theme, or preferred language.

3. Context should only be used for data that's relevant to multiple components with different parent components.

4. Context is updated using a Provider component. The Provider is wrapped around a component and allows its descendants to access the context data, i.e. it's the context "giver".

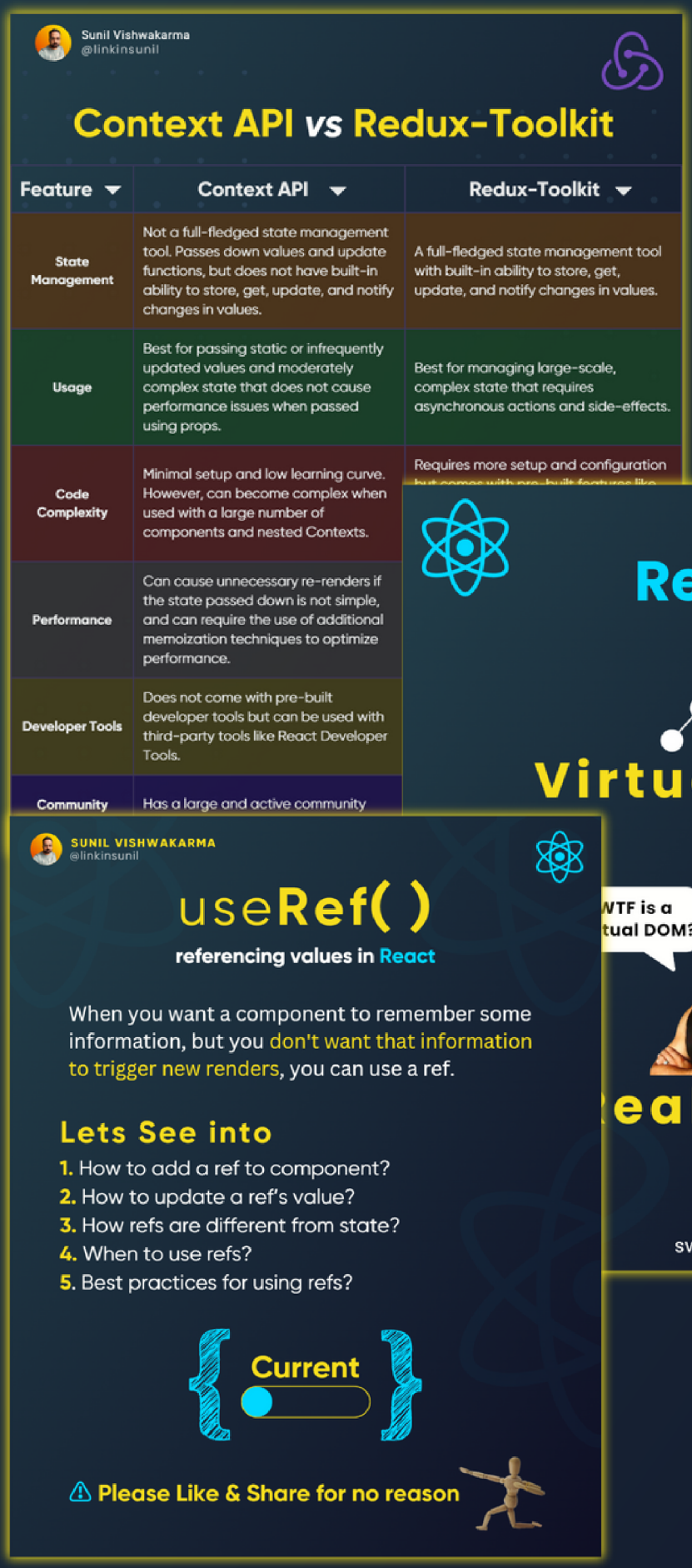
5. A Consumer component "consumes" or accesses this data. Context can be used with the useContext hook in functional components to get the current context value. In class components, the Context.Consumer component is used.

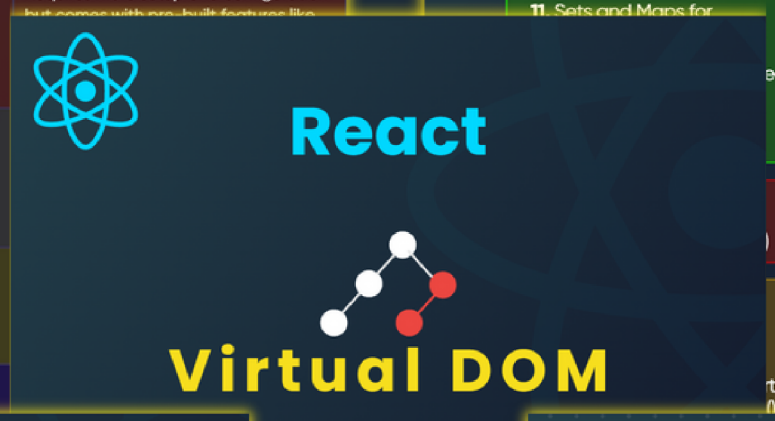
swipe ⟶

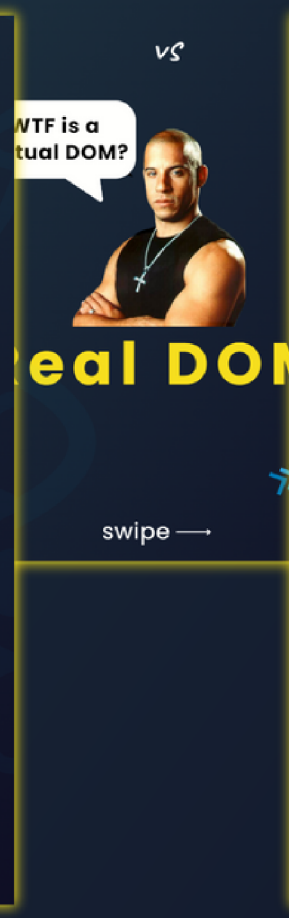
SUNIL VISHWAKARMA
@linkinsunil

Thats a Wrap!

If you liked it, **visit my profile and checkout other posts**







P.S.

Repost this if you think your followers will like it

Enjoyed this?

1. Follow me
2. Click the 🔔 notification
3. Never miss a post