Context in React

**What is React Context?**

React Context is a method to pass props from parent to child component(s), by storing the props in a store(similar in Redux) and using these props from the store by child component(s) without actually passing them manually at each level of the component tree.

**Why React Context?** **We have Redux!!**

Using Redux to interact with states from parent to child components is not only quite difficult to understand but also gives you a more complex code. Through the usage of Context, the understanding of concept and code is far easier than that of Redux.

**When to use React Context?**

Anytime you want! There is no iron-clad rule like when to use Context in your application. Whenever you want a store to keep your states or variables in and use them elsewhere in your program, use Context. Generally, when we have two or more levels(height) in our component tree, it is viable to use a store instead of passing props and then lifting the state as this will create confusion and unnecessary lengthy code.

**Example:**

If we have three components in our app, A->B->C where A is the parent of B and B is the parent of C. To change a state from C and pass it to A, keep the state of A in a store, then extract the state from store and use it in C. This completely eliminates the necessity of the state to pass through B. So the flow is like A->C.

**API**

React.createContext

**const MyContext = React.createContext(defaultValue);**

Creates a Context object. When React renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above it in the tree.

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This default value can be helpful for testing components in isolation without wrapping them. Note: passing undefined as a Provider value does not cause consuming components to use defaultValue.

**Context.Provider**

**<MyContext.Provider value={/\* some value \*/}>**

Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.

The Provider component accepts a value prop to be passed to consuming components that are descendants of this Provider. One Provider can be connected to many consumers. Providers can be nested to override values deeper within the tree.

All consumers that are descendants of a Provider will re-render whenever the Provider’s value prop changes. The propagation from Provider to its descendant consumers (including .contextType and useContext) is not subject to the shouldComponentUpdate method, so the consumer is updated even when an ancestor component skips an update.

Changes are determined by comparing the new and old values using the same algorithm as Object.is.

**Class.contextType**

class MyClass extends React.Component {

componentDidMount() {

let value = this.context;

/\* perform a side-effect at mount using the value of MyContext \*/

}

componentDidUpdate() {

let value = this.context;

/\* ... \*/

}

componentWillUnmount() {

let value = this.context;

/\* ... \*/

}

render() {

let value = this.context;

/\* render something based on the value of MyContext \*/

}

}

MyClass.contextType = MyContext;

**Context.Consumer**

<MyContext.Consumer>

{value => /\* render something based on the context value \*/}

</MyContext.Consumer>

A React component that subscribes to context changes. Using this component lets you subscribe to a context within a function component.

Requires a function as a child. The function receives the current context value and returns a React node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree. If there is no Provider for this context above, the value argument will be equal to the defaultValue that was passed to createContext().

**Context.displayName**

Context object accepts a displayName string property. React DevTools uses this string to determine what to display for the context.

For example, the following component will appear as MyDisplayName in the DevTools:

const MyContext = React.createContext(/\* some value \*/);

MyContext.displayName = 'MyDisplayName';

<MyContext.Provider> // "MyDisplayName.Provider" in DevTools

<MyContext.Consumer> // "MyDisplayName.Consumer" in DevTools

**Examples**

**Dynamic Context**

A more complex example with dynamic values for the theme:

**theme-context.js**

export const themes = {

light: {

foreground: '#000000',

background: '#eeeeee',

},

dark: {

foreground: '#ffffff',

background: '#222222',

},

};

export const ThemeContext = React.createContext(

themes.dark // default value

);

**themed-button.js**

import {ThemeContext} from './theme-context';

class ThemedButton extends React.Component {

render() {

let props = this.props;

let theme = this.context;

return (

<button

{...props}

style={{backgroundColor: theme.background}}

/>

);

}

}

ThemedButton.contextType = ThemeContext;

export default ThemedButton;

**app.js**

import {ThemeContext, themes} from './theme-context';

import ThemedButton from './themed-button';

// An intermediate component that uses the ThemedButton

function Toolbar(props) {

return (

<ThemedButton onClick={props.changeTheme}>

Change Theme

</ThemedButton>

);

}

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

theme: themes.light,

};

this.toggleTheme = () => {

this.setState(state => ({

theme:

state.theme === themes.dark

? themes.light

: themes.dark,

}));

};

}

render() {

// The ThemedButton button inside the ThemeProvider

// uses the theme from state while the one outside uses

// the default dark theme

return (

<Page>

<ThemeContext.Provider value={this.state.theme}>

<Toolbar changeTheme={this.toggleTheme} />

</ThemeContext.Provider>

<Section>

<ThemedButton />

</Section

</Page>

);

}

}

ReactDOM.render(<App />, document.root);

**Updating Context from a Nested Component**

It is often necessary to update the context from a component that is nested somewhere deeply in the component tree. In this case you can pass a function down through the context to allow consumers to update the context:

**theme-context.js**

// Make sure the shape of the default value passed to

// createContext matches the shape that the consumers expect!

export const ThemeContext = React.createContext({

theme: themes.dark,

toggleTheme: () => {},

});

theme-toggler-button.js

import {ThemeContext} from './theme-context';

function ThemeTogglerButton() {

// The Theme Toggler Button receives not only the theme

// but also a toggleTheme function from the context

return (

<ThemeContext.Consumer>

{({theme, toggleTheme}) => (

<button

onClick={toggleTheme}

style={{backgroundColor: theme.background}}>

Toggle Theme

</button>

)}

</ThemeContext.Consumer>

);

}

export default ThemeTogglerButton;

**app.js**

import {ThemeContext, themes} from './theme-context';

import ThemeTogglerButton from './theme-toggler-button';

class App extends React.Component {

constructor(props) {

super(props);

this.toggleTheme = () => {

this.setState(state => ({

theme:

state.theme === themes.dark

? themes.light

: themes.dark,

}));

};

// State also contains the updater function so it will

// be passed down into the context provider

this.state = {

theme: themes.light,

toggleTheme: this.toggleTheme,

};

}

render() {

// The entire state is passed to the provider

return (

<ThemeContext.Provider value={this.state}>

<Content />

</ThemeContext.Provider>

);

}

}

function Content() {

return (

<div>

<ThemeTogglerButton />

</div>

);

}

ReactDOM.render(<App />, document.root);

**Styling React Using Sass**

What is Sass

Sass is a CSS pre-processor.

Sass files are executed on the server and sends CSS to the browser.

**Can I use Sass?**

If you use the create-react-app in your project, you can easily install and use Sass in your React projects.

Install Sass by running this command in your terminal:

>npm i sass

**Create a Sass file**

Create a Sass file the same way as you create CSS files, but Sass files have the file extension .scss

In Sass files you can use variables and other Sass functions:

**my-sass.scss:**

**Create a variable to define the color of the text:**

$myColor: red;

h1 {

color: $myColor;

}

**Import the Sass file the same way as you imported a CSS file:**

**index.js:**

import React from 'react';

import ReactDOM from 'react-dom';

import './my-sass.scss';

const Header = () => {

return (

<>

<h1>Hello Style!</h1>

<p>Add a little style!.</p>

</>

);

}

ReactDOM.render(<Header />, document.getElementById('root'));