React Context

Topics : React JS
Written on January 03, 2024

React Context is a feature that allows you to share state or other values like themes or authentication status between components without having to explicitly pass props through every level of the component tree. It's particularly useful when you have deeply nested components that need access to shared data.

Here's a basic overview of how React Context works:

1. Creating a Context:

You create a context using the `createContext` function. This function returns an object with two components: `Provider` and `Consumer`. However, with the introduction of the `useContext` hook in React 16.8, using the `Consumer` component is less common.

```
// MyContext.js
import { createContext } from 'react';

const MyContext = createContext();

export default MyContext;
```

2. Providing the Context Value:

The `Provider` component is used to wrap the part of the component tree where you want to make the context available. It takes a `value` prop, which is the data you want to share.

```
// App.js
import React from 'react';
import MyContext from './MyContext';

const App = () => {
const sharedValue = 'Hello from Context!';

return (
<MyContext.Provider value={sharedValue}>
{/* Your components go here */}
</MyContext.Provider>
);
};

export default App;
```

3. Consuming the Context Value:

Components that want to access the shared data can use the `useContext` hook. This hook takes the context object created earlier and returns the current context value.

```
// SomeComponent.js
import React, { useContext } from 'react';
import MyContext from './MyContext';

const SomeComponent = () => {
const contextValue = useContext(MyContext);

return (
<div>
<p>{contextValue}</p>
</div>
);
};

export default SomeComponent;
```

Alternatively, you can still use the `Consumer` component for class components or when you need to consume multiple contexts.

```
// SomeComponent.js
import React from 'react';
import MyContext from './MyContext';

const SomeComponent = () => (
<MyContext.Consumer>
{(contextValue) => (
<div>
<p>{contextValue}</p>
</div>
)}
</MyContext.Consumer>
);

export default SomeComponent;
```

By using React Context, you can avoid "prop drilling" (passing props down multiple levels) and make your code more maintainable, especially in scenarios where multiple components need access to the same data or state.