Slim

React js tips

How to Set Up useContext?

A Beginner's Guide

Slim toumi

Slim

React js tips

What is useContext ?

- useContext is a hook that makes it easy to access shared data without **“prop-drilling”** (manually passing props through many layers).
- It's ideal for global data like user settings, themes, or other information you want to access across components.

Example Uses for useContext :

- **Authentication:** Share user login info with components like Navbar or Dashboard.
- **Themes:** Enable light/dark mode across components.
- **Language Preferences:** Easily switch languages in the app.

Quick Tips :

- **Automatic Updates:** If the context value changes, components using it will re-render with the new data.
- **Avoid Overuse:** Use useContext wisely—too many contexts can make your app complex.

Slim toumi

How to Use useContext ?

Step 1: Create the Context

In this step, we create a context, which acts like a "container" for shared data. Place this in its own file to keep things organized.

1. .Create a new file called MyContext.jsx.
2. .Define and export the context by using createContext.

MyContext.jsx

```jsx
import React, { createContext } from 'react';

// Creates a new context
const MyContext = createContext();

export default MyContext;
```

Note: This file only sets up the context—it doesn't contain any data yet. The data will be provided in the next step.

Step 2: Provide the Context to Components

Next, we wrap our main component with the context provider. This makes the shared data accessible to any component inside the provider.

1. .Open App.jsx (or your main file where you have the root component).
2. .Import MyContext from the file you created.
3. .Wrap your component tree inside <MyContext.Provider>, and pass in the data you want to share using the value prop.

App.jsx

```jsx
import React from 'react';
import MyContext from './MyContext';
import ChildComponent from './ChildComponent';

function App() {
  const sharedData = "Hello from Context!";
  return (
    <MyContext.Provider value={sharedData}>
      <ChildComponent />
    </MyContext.Provider>
  );
}
export default App;
```

Note: sharedData can be any data type (string, array, object, etc.). Here, it's a simple string for demonstration.

Step 2: Access the Context in a Child Component

Now that we've provided the data, we can access it in any child component inside <MyContext.Provider>.

1. .Create a new file called ChildComponent.jsx
2. .Import useContext and MyContext.
3. .Use useContext to access the data provided by MyContext.

ChildComponent.jsx

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

function ChildComponent() {
  const data = useContext(MyContext);

  return <p>{data}</p>;
}

export default ChildComponent;
```

Note: useContext(MyContext) allows ChildComponent to directly access the value from <MyContext.Provider>. This makes it easy to share data between components without using props.

Recap

1 .Define the context in its own file (MyContext.jsx).

2 .Wrap your components inside <MyContext.Provider> in App.jsx, passing the shared data with value.

3 .Use useContext in any child component to access the shared data.

This setup allows you to easily share data across multiple components without "prop-drilling" (passing data down through multiple component layers). It’s a powerful way to manage state and data in React!

Slim

React js tips

Hopefully You Found It Usefull!

Be sure to save this post so you can come back to it later

like

Comment

Share

Slim toumi