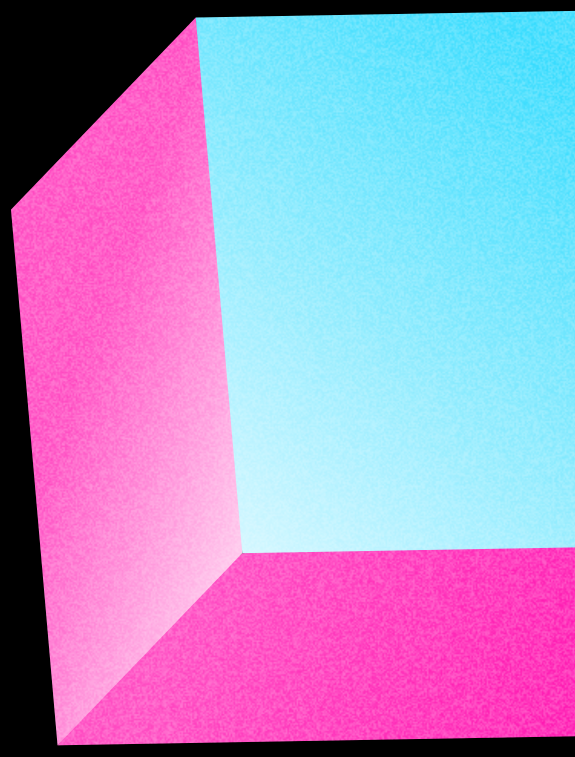
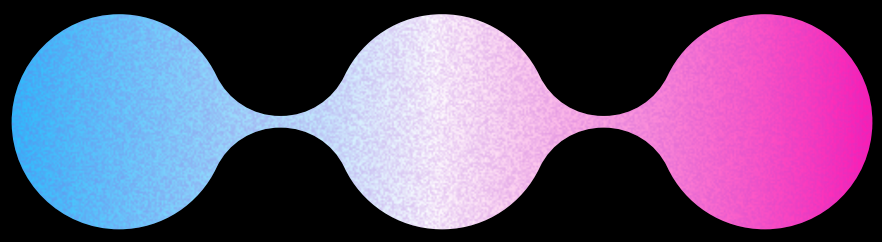


UNDERSTANDING USECONTEXT HOOK

IN REACT JS

Abhijeet Chakane



INTRODUCTION TO THE USECONTEXT HOOK

The useContext hook is a powerful tool in React that allows you to manage global state across your application. It simplifies the process of sharing data between components without prop drilling.

WHY USE USECONTEXT?

- Global State Management: Easily share state across your application.
- Simplified Code: Avoid the complexity of prop drilling.
- Improved Readability: Make your code cleaner and more maintainable.

CREATING AND USING A CONTEXT

First, create a context using React.createContext(), then use the useContext hook to access the context value.

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

// Create a Context
const MyContext = createContext();

function App() {
  const [value, setValue] = useState('Hello, World!');

  return (
    <MyContext.Provider value={value}>
      <ChildComponent />
    </MyContext.Provider>
  );
}

function ChildComponent() {
  const contextValue = useContext(MyContext);
  return <div>{contextValue}</div>;
}

```

UPDATING CONTEXT VALUE

To update the context value, pass both the value and a setter function through the provider.

```jsx
function App() {
  const [value, setValue] = useState('Hello, World!');

  return (
    <MyContext.Provider value={{ value, setValue }}>
      <ChildComponent />
    </MyContext.Provider>
  );
}

function ChildComponent() {
  const { value, setValue } = useContext(MyContext);
  return (
    <div>
      <p>{value}</p>
      <button onClick={() => setValue('New Value')}>Update Value</button>
    </div>
  );
}

```

NESTED CONTEXT CONSUMERS

useContext works seamlessly even in deeply nested components, making it a versatile choice for state management.

```jsx
function GrandChildComponent() {
  const { value } = useContext(MyContext);
  return <p>GrandChild: {value}</p>;
}

function ChildComponent() {
  return (
    <div>
      <GrandChildComponent />
    </div>
  );
}

```

PRACTICAL APPLICATIONS OF USECONTEXT

- Theme Management: Toggle between light and dark modes.
- User Authentication: Share user login status across components.
- Localization: Manage and switch application languages.

KEY POINTS TO REMEMBER

- useContext simplifies state management across your app.
- It removes the need for prop drilling.
- Combine useContext with other hooks for powerful functionality.

ADVANCED USAGE WITH CUSTOM CONTEXT HOOK

Create custom hooks to encapsulate context logic for cleaner code.

```jsx
const useMyContext = () => useContext(MyContext);

function ChildComponent() {
  const { value, setValue } = useMyContext();
  return (
    <div>
      <p>{value}</p>
      <button onClick={() => setValue('Updated Value')}>Change Value</button>
    </div>
  );
}

```

FOLLOW FOR MORE TIPS!

Abhijeet Chakane