React Context API

Introduction

The Context API in React is a built-in way to share values (like global data) between components without having to pass props manually at every level of the component tree.

When to use Context

Use the Context API when:

- You have global or app-wide data (like theme, auth status, user settings).
- You want to avoid prop drilling — passing data through many layers of components unnecessarily.

General Syntax

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

export const MyContext = createContext(); // optional default value
```

Then wrap your app with the built-in Provider

Use MyContext.Provider / MyContext where you render your app (e.g., in index.js or App.js):

React 18 Context Syntax

Before (React 18 and below)

```jsx
<MyContext.Provider value={value}>
  <App />
</MyContext.Provider>
```

React 19 Context Syntax

Now in React 19

```jsx
<MyContext value={value}>
  <App />
</MyContext>
```

Example

Create the Context

```jsx
jsx

// ConfigContext.js
import { createContext } from 'react';

export const ConfigContext = createContext();
```

Use Context as a Component in React 19

```jsx
const config = {
  apiUrl: 'https://api.example.com',
  theme: 'dark',
  appName: 'My Awesome App',
};

ReactDOM.createRoot(document.getElementById('root')).render(
  <ConfigContext value={config}>
    <App />
  </ConfigContext>
);
```

Consume the Context

```jsx
// App.jsx or any child component
import React, { useContext } from 'react';
import { ConfigContext } from './ConfigContext';

const App = () => {
  const config = useContext(ConfigContext);

  return (
    <div>
      <h1>{config.appName}</h1>
      <p>Theme: {config.theme}</p>
      <p>API URL: {config.apiUrl}</p>
    </div>
  );
};

export default App;
```

Example: Fetching Data (like user info) and Passing via Context

Create the Context

jsx

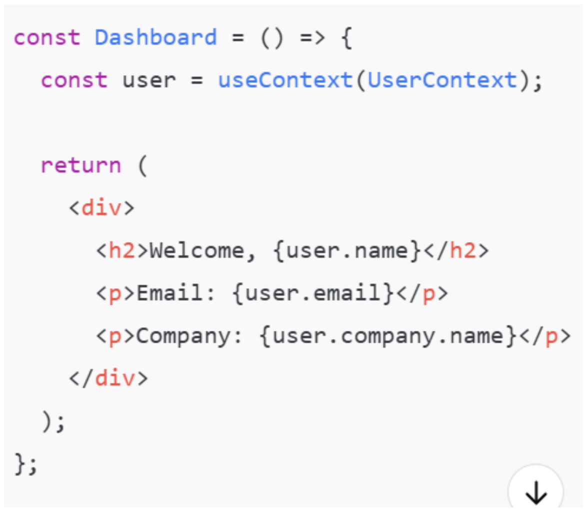
```jsx
// UserContext.js
import { createContext } from 'react';

export const UserContext = createContext();
```

Fetch API in App and Provide Context

Copy

```jsx
const App = () => {
  const [user, setUser] = useState(null);

  useEffect(() => {
    // Simulate API call (you can replace this with a real API)
    const fetchUser = async () => {
      const res = await fetch('https://jsonplaceholder.typicode.com/users/1');
      const data = await res.json();
      setUser(data);
    };

    fetchUser();
  }, []);

  if (!user) return <p>Loading user...</p>;

  return (
    <UserContext value={user}>
      <Dashboard />
    </UserContext>
  );
};
```

Consume Context in Dashboard



```
const Dashboard = () => {
  const user = useContext(UserContext);

  return (
    <div>
      <h2>Welcome, {user.name}</h2>
      <p>Email: {user.email}</p>
      <p>Company: {user.company.name}</p>
    </div>
  );
};
```

Avoid using Context for

- High-frequency updates (it may cause unnecessary re-renders).
- Local component state — keep that local.

Thus, Context is perfect for global, stable, or config-style data, and with React 19’s syntax improvements, it's smoother than ever.