React Context CheatSheet

by Ndeye Fatou Diop

What is Context?

- Context provides a way to share values between components without passing props explicitly.
- It's useful for avoiding prop drilling when data needs to be accessible by many components.

```
const MyContext = createContext(defaultValue);
```

Using the Context

- Access context values using **useContext**

```
import { useContext } from "react";

function UserInfo() {
  const user = useContext(UserContext);
  return <p>{user.name}</p>;
}
```

Creating a Context

- Create the context using createContext() from React
- Define a Provider to wrap components that need access to the context.

```
export const UserContext = React.createContext();

export function UserProvider({ children }) {
  // TODO: Get this value dynamically
  const user = { name: "Fatou", age: 30 };
  return (
    <UserContext.Provider value={user}>
      {children}
    </UserContext.Provider>
  );
}
```

Splitting Providers for Performance

- Split unrelated contexts to avoid unnecessary re-renders.

```
// ❌ Having a single context can create performance issues
const AppContext = React.createContext();


// ✅ Better
const ThemeContext = React.createContext();
const UserContext = React.createContext();
}
```

Using a Custom Hook for Safe Access

- Encapsulate context logic in a custom hook for cleaner and safer usage.

```
function useUserContext() {
  const context = useContext(UserContext);
  if (!context) {
    throw new Error("useUserContext must be used within a UserProvider");
  }
  return context;
}
```

Performance Considerations

Context causes re-renders: Any component consuming context will re-render even if the context value it uses doesn't change.

```
const AppContext = React.createContext();

function AppProvider({ children }) {
  const [theme, setTheme] = React.useState("light");
  const [user, setUser] = React.useState({ name: "Fatou" });

  return (
    <AppContext.Provider value={{ theme, user }}>
      {children}
    </AppContext.Provider>
  );
}

// ❌ Bad: This will re-render if the user changes
function ThemeDisplay() {
  const { theme } = React.useContext(AppContext);
  return <p>Theme: {theme}</p>;
}
```

Best Practices with Context

- **Minimise context size:** Keep only necessary values in context to avoid excessive re-renders.
- **Don't use the context to cache server state:** Use a library like TanStack Query or tools like GraphQL instead.
- **Memoize values passed to context:** Use useMemo to prevent unnecessary re-renders.

```
const user = useMemo(() => ({ name, email }), [name, email]);

<UserContext.Provider value={value}
>{children}</UserContext.Provider>
```

Wrapping Components in a Provider

- **Important**: Context only works if your component tree is wrapped in the corresponding Provider.
- If the provider is missing, useContext will return the defaultValue from React.createContext(defaultValue).

```
function App() {
  return (
    <UserProvider>
      <UserInfo />
    </
UserProvider>
  );
}
```