Context API vs Zustand

(A Performance & Developer Experience Compare)

This project demonstrates the differences between **React Context API** and **Zustand** for state management in React/Next.js applications.

It is implemented as a **monorepo using Turborepo**, with shared components, hooks, data, and types between the two apps for a fair comparison.

**Approach**

1. **Initial Research & Setup**

* Explored monorepo approaches → chose **Turborepo** for shared code and easy setup
* Decided to build **two apps with identical features and UI** (Catalog, Cart, Preferences, Live Ticker).
* The only difference: **state management implementation** (Context vs Zustand).

1. **Implementation Steps**
   1. Built **Context API app** first using useContext, useReducer, and useMemo.
   2. Built **Zustand app** with equivalent features using store slices and selectors.
   3. Ensured both apps behaved identically so the comparison was fair.
2. **Optimization Exploration (Context API)**
   1. Initially, Context caused **full re-renders** when global state changed.
   2. Explored techniques to optimize Context API:
      1. **Specialized Contexts**: Separate ThemeContext, CurrencyContext, CartContext, etc.
      2. **useMemo**: To prevent unnecessary value recreations.
      3. **useReducer**: For structured updates in complex state (e.g., Cart).
   3. Found that **performance could be improved**, but the code became more **complex and harder to maintain**.
3. **Comparison Focus**
   1. **Context API**: Simple to start, but can get messy with multiple providers and optimizations
   2. **Zustand**: Cleaner, selective re-renders by default, less boilerplate, and better suited for frequent updates (e.g., live ticker).

**Comparison points**

1. **Context API**

**Advantage=>**

* Built into React, No external dependancy
* Simple and Straightforward small apps
* Easy to get started

**Disadvantages=>**

* By default, All consumers re-render on state change
* Needs extra optimizatio( useMemo, multiple contexts, reducers )
* Becomes complex and messy with many providers

1. **Zustand**

**Advantage=>**

* Minimal boilerplate
* Built in devtools supporting for debugging
* Scales much better for Large apps

**Disadvantages=>**

* External dependency, No inbuild(but smaller)
* Slight learning curve for teams new to it

**Conclusion**

We can use the ContextAPI and Zustand for all projects, but based on the advantages the most suitable is the ContextAPI for a small apps(around 4 , 5 screens) and Zustand will be better for middle/large applications also which is using in real world scenarios