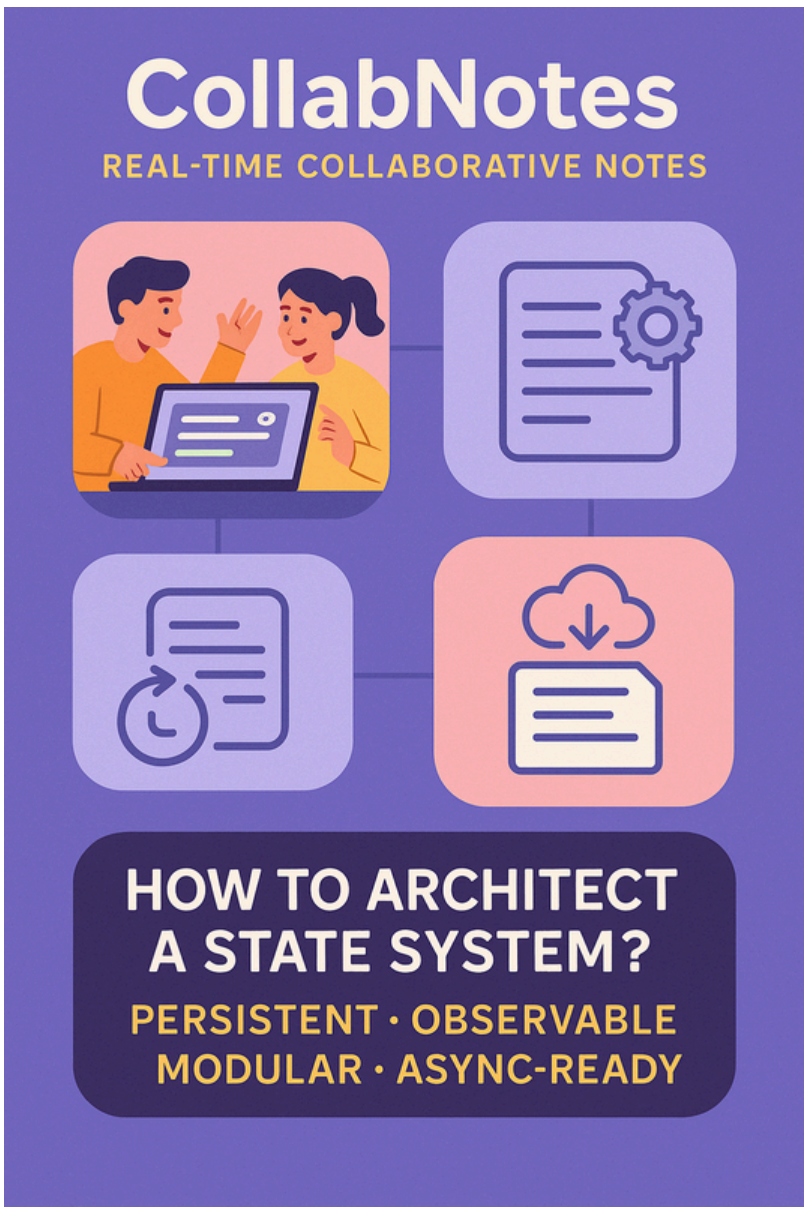
Advanced State Management with Zustand: Middleware, Persistence, and Async Patterns

1. Problem Statement

Case Study: CollabNotes – Real-Time Collaborative Notes

CollabNotes is a real-time note-taking platform for teams:

- Users can create, edit, and delete notes, and see updates from teammates instantly.
- Notes, user preferences, and session info must persist across reloads and browser sessions.
- Every change (edit, delete, sync) should be logged for audit and undo/redo features.
- The app fetches notes from a cloud API and keeps them in sync with local state.
- As the app grows, the team needs minimal boilerplate, type safety, and high performance.



The challenge:
How do you architect a state system that is **persistent, observable, modular, and async-ready**—with minimal code, maximum reliability, and no unnecessary re-renders?

2. Learning Objectives

By the end of this tutorial, you will:

- Use Zustand middleware for devtools, immutability (immer), and state persistence.
- Persist only selected parts of state to localStorage/sessionStorage.
- Migrate and version persisted state for schema changes.

- Log all state changes for audit/history.
- Combine Zustand with React Query for async data fetching and syncing.
- Avoid common pitfalls and follow best practices for scalable state management.

---

3. Concept Introduction with Analogy

Analogy: The CollabNotes Command Center

- **Middleware** are like security cameras and safes: they log every change and keep important data safe even if the power goes out.
- **Persistence** is the vault: your notes and preferences are always there, even after a crash or reload.
- **React Query** is the courier: it fetches the latest notes from the cloud and syncs them with your local store, so everyone sees the same thing in real time.

---

4. Technical Deep Dive

A. Zustand Middleware: devtools, persist, immer, and custom logging

1. Devtools Middleware

- Lets you inspect and time-travel state changes in Redux DevTools.
- Should be the **last** middleware applied.

```
import { create } from 'zustand';
import { devtools } from 'zustand/middleware';

const useNoteStore = create(
 devtools((set) => ({
   notes: [],
   addNote: (note) => set((state) => ({ notes: [...state.notes, note] })),
 }))
);
```

2. Persist Middleware

- Persists state across reloads using localStorage/sessionStorage.
- Use `partialize` to persist only selected fields.

```
import { persist, createJSONStorage } from 'zustand/middleware';

const usePreferencesStore = create(
  persist(
    (set) => ({
      theme: 'light',
      fontSize: 14,
      setTheme: (theme) => set({ theme }),
      setFontSize: (size) => set({ fontSize: size }),
    }),
    {
      name: 'collabnotes-preferences',
```

```
      storage: createJSONStorage(() => localStorage),
      partialize: (state) => ({ theme: state.theme, fontSize: state.fontSize }),
      version: 2,
      migrate: (persisted, version) => {
        if (version < 2) return { ...persisted, fontSize: 14 };
        return persisted;
      },
    }
  )
);
```

3. Immer Middleware

- Enables immutable updates with a mutable API (like Redux Toolkit).

```
import { immer } from 'zustand/middleware/immer';

const useNoteStore = create(
  immer((set) => ({
    notes: [],
    updateNote: (id, text) =>
      set((state) => {
        const note = state.notes.find((n) => n.id === id);
        if (note) note.text = text;
      }),
  }))
);
```

4. Custom Logging Middleware

- Log every change for audit/history.

```
const logMiddleware = (config) => (set, get, api) =>
config((args) => {
 console.log('Before:', get());
 set(args);
 console.log('After:', get());
}, get, api);

const useNoteStore = create(
logMiddleware((set) => ({
 // ...state and actions
}))
);
```

B. State Versioning & Migration

- Use `version` and `migrate` in `persist` to safely upgrade persisted state.

```
persist(
  (set) => ({
    notes: [],
    lastSynced: null,
  }),
  {
    name: 'notes-storage',
    version: 2,
    migrate: (persisted, version) => {
      if (version < 2) return { ...persisted, lastSynced: null };
      return persisted;
    },
  }
)
```

C. Combining Zustand with React Query

- **React Query** fetches and caches async data (notes from the cloud).
- **Zustand** manages local state and syncs with React Query.

```
import { useQuery } from '@tanstack/react-query';
import { create } from 'zustand';

const useNoteStore = create((set) => ({
  notes: [],
  setNotes: (notes) => set({ notes }),
}));

function NotesList() {
  const setNotes = useNoteStore((s) => s.setNotes);
  const notes = useNoteStore((s) => s.notes);

  const { data, isLoading } = useQuery(['notes'], fetchNotesFromAPI, {
    onSuccess: setNotes,
  });

  if (isLoading) return <div>Loading...</div>;
  return (
    <ul>
      {notes.map((n) => (
        <li key={n.id}>{n.text}</li>
      ))}
    </ul>
  );
}
```

5. Step-by-Step Data Modeling & Code Walkthrough

A. Persisted Preferences Store with Migration

```
import { create } from 'zustand';
import { persist, createJSONStorage } from 'zustand/middleware';

const usePreferencesStore = create(
  persist(
    (set) => ({
      theme: 'light',
      fontSize: 14,
      setTheme: (theme) => set({ theme }),
      setFontSize: (size) => set({ fontSize: size }),
    }),
    {
      name: 'collabnotes-preferences',
      storage: createJSONStorage(() => localStorage),
      partialize: (state) => ({ theme: state.theme, fontSize: state.fontSize }),
      version: 2,
      migrate: (persisted, version) => {
        if (version < 2) return { ...persisted, fontSize: 14 };
        return persisted;
      },
    }
  )
);
```

B. Notes Store with Devtools, Immer, and Logging

```
import { create } from 'zustand';
import { devtools, immer } from 'zustand/middleware';

const logMiddleware = (config) => (set, get, api) =>
```

```
  config((args) => {
    console.log('Before:', get());
    set(args);
    console.log('After:', get());
  }, get, api);

const useNoteStore = create(
  devtools(
    immer(
      logMiddleware((set) => ({
        notes: [],
        addNote: (note) =>
          set((state) => {
            state.notes.push(note);
          }),
        updateNote: (id, text) =>
          set((state) => {
            const note = state.notes.find((n) => n.id === id);
            if (note) note.text = text;
          }),
        deleteNote: (id) =>
          set((state) => {
            state.notes = state.notes.filter((n) => n.id !== id);
          }),
      }))
    )
  )
);
```

C. Syncing Notes with React Query

```
import { useQuery } from '@tanstack/react-query';
import useNoteStore from './store/noteStore';

function NotesList() {
  const setNotes = useNoteStore((s) => s.setNotes);
  const notes = useNoteStore((s) => s.notes);

  const { data, isLoading } = useQuery(['notes'], fetchNotesFromAPI, {
    onSuccess: setNotes,
  });

  if (isLoading) return <div>Loading...</div>;
  return (
    <ul>
      {notes.map((n) => (
        <li key={n.id}>{n.text}</li>
      ))}
    </ul>
  );
}
```

6. Interactive Challenge / Mini-Project

Your Turn!

1. Create a persisted Zustand store for user session:
    - Fields: `userId: string`, `token: string`, `expiresAt: number`
    - Only persist `userId` and `token`, not `expiresAt`
    - Add a migration to handle a new field, `role: 'admin' | 'user'` (default ‘user’), in version 2.
2. Use devtools and immer middleware for a note history log:

- Actions: `addHistoryEntry`, `clearHistory`
- Log each entry as `{ noteId: string, action: string, timestamp: number }`

3. Combine Zustand and React Query:
   - Fetch a list of collaborators from an API.
   - Store collaborators in Zustand.
   - Display collaborators in a component, updating automatically when data is fetched.

7. Common Pitfalls & Best Practices

Pitfall	Best Practice
Persisting too much state	Use `partialize` to persist only what's needed
Not versioning persisted state	Use `version` and `migrate` for schema changes
Middleware order mistakes	Apply `devtools` last
Not using selectors in Zustand	Use selectors to prevent unnecessary renders
Mixing async fetch with store	Use React Query for fetching, Zustand for UI

8. Optional: Programmer's Workflow Checklist

- Use middleware (`persist`, `devtools`, `immer`) as needed.
- Use `partialize` to control what gets persisted.
- Add `version` and `migrate` for evolving state.
- Use selectors for efficient reactivity.
- Combine Zustand with React Query for async data.
- Test stores and migrations independently.