# State Management

# State Management Architecture

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

# Document History

| Version | Date | Author | Changes |
|---|---|---|---|
| 0.1 | {{DATE}} | {{AUTHOR}} | Initial draft |

---

# 1. State Architecture Overview

**{{PROJECT_NAME}}** uses a layered state management approach:

| Layer | Library | Scope |
|---|---|---|
| Server state | `{{TanStack Query / SWR / Apollo}}` | API data, caching, synchronization |
| Client / UI state | `{{Zustand / Redux Toolkit / Pinia}}` | Application-wide UI state |
| URL state | Native router APIs | Filters, pagination, search |
| Form state | `{{React Hook Form / Formik / VeeValidate}}` | Form data, validation |
| Persistent state | `{{localStorage / cookies}}` | User preferences, tokens |

**Guiding principle:** Server state is NOT stored in client state. API data lives in the query cache — client state holds only UI concerns (sidebar open, selected theme, modal visibility).

---

# 2. Data Flow Diagram

```
flowchart TD
    User["User Interaction"] --> Component["Component"]

    Component -->|"API call"| QueryCache["Query Cache\n(TanStack Query)"]
    Component -->|"UI action"| ClientStore["Client Store\n(Zustand)"]
    Component -->|"Navigate"| URLState["URL State\n(Router)"]
    Component -->|"Form input"| FormState["Form State\n(RHF)"]

    QueryCache -->|"fetch / mutation"| API["Backend API"]
    QueryCache -->|"cached data"| Component
    ClientStore -->|"state slice"| Component
    URLState -->|"params"| Component
    FormState -->|"values / errors"| Component

    ClientStore -->|"user prefs"| LocalStorage["localStorage\n(Persistent)"]
    LocalStorage -->|"hydrate on load"| ClientStore
```

---

# 3. State Categories

## 3.1 Server State (API Data)

**Library:** `{{TanStack Query v5}}`

**Configuration:**

```
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5,        // 5 minutes
      gcTime: 1000 * 60 * 30,          // 30 minutes garbage collection
      retry: 2,
      refetchOnWindowFocus: true,
      refetchOnReconnect: true,
    },
    mutations: {
      retry: 0,
    },
  },
```

```
});
```

**Query key convention:**

```
// Hierarchical keys for precise invalidation
['users']                          // all user queries
['users', { page: 1, search: '' }] // paginated user list
['users', userId]                  // single user
['users', userId, 'posts']         // user's posts
```

**Stale time per resource type:**

| Resource | Stale Time | GC Time | Rationale |
|---|---|---|---|
| User profile | 5 min | 30 min | Changes infrequently |
| Dashboard stats | 30 sec | 5 min | Near-realtime |
| Config / enums | 1 hour | 24 hours | Rarely changes |
| Notifications | 0 (always fresh) | 2 min | Time-sensitive |

---

# 3.2 Client State (UI State)

**Library:** `{{Zustand}}`

**Store slices:**

| Slice | File | Responsibility |
|---|---|---|
| `uiStore` | `src/stores/ui.store.ts` | Sidebar open, active modal, theme |
| `authStore` | `src/stores/auth.store.ts` | Current user, roles, session token |
| `notificationStore` | `src/stores/notification.store.ts` | Toast queue, unread count |
| `{{featureStore}}` | `src/stores/{{feature}}.store.ts` | {{Feature-specific UI state}} |

**Slice template:**

```
// src/stores/ui.store.ts
import { create } from 'zustand';

interface UIState {
  sidebarOpen: boolean;
  activeModal: string | null;
  theme: 'light' | 'dark' | 'system';
```

```
}

interface UIActions {
  setSidebarOpen: (open: boolean) => void;
  openModal: (id: string) => void;
  closeModal: () => void;
  setTheme: (theme: UIState['theme']) => void;
}

export const useUIStore = create<UIState & UIActions>((set) => ({
  sidebarOpen: true,
  activeModal: null,
  theme: 'system',

  setSidebarOpen: (open) => set({ sidebarOpen: open }),
  openModal: (id) => set({ activeModal: id }),
  closeModal: () => set({ activeModal: null }),
  setTheme: (theme) => set({ theme }),
}));
```

---

# 3.3 URL State

URL state is the source of truth for:

- Search / filter queries
- Pagination (page, pageSize)
- Sort column and direction
- Active tab / view mode
- Modal ID (when deep-linkable)

**Convention:**

```
/users?page=2&pageSize=25&search=john&sort=name&dir=asc&status=active
```

**Library:** Native `URLSearchParams` + router `useSearchParams` hook

**Serialization helper:** `src/lib/url-state.ts`

```
// Type-safe URL param parsing with fallbacks
export function parseListParams(params: URLSearchParams): ListParams {
  return {
```

```
    page: Number(params.get('page') ?? 1),
    pageSize: Number(params.get('pageSize') ?? 25),
    search: params.get('search') ?? '',
    sort: params.get('sort') ?? 'createdAt',
    dir: (params.get('dir') as 'asc' | 'desc') ?? 'desc',
  };
}
```

---

# 3.4 Form State

**Library:** `{{React Hook Form v7}}` **Validation:** `{{Zod}}`

**Pattern:**

```
const schema = z.object({
  email: z.string().email('Invalid email'),
  name: z.string().min(2, 'Name must be at least 2 characters'),
});

const form = useForm<z.infer<typeof schema>>({
  resolver: zodResolver(schema),
  defaultValues: { email: '', name: '' },
});
```

**Rules:**

- Form state NEVER leaks into global store
- Schema validation lives in `src/schemas/` — reused for API validation
- Complex multi-step forms use form context + Stepper component

---

# 3.5 Persistent State

| Data | Storage | Library | Encryption |
|---|---|---|---|
| Theme preference | `localStorage` | Zustand persist middleware | No |
| Sidebar collapsed | `localStorage` | Zustand persist middleware | No |
| Language preference | `localStorage` | Native | No |
| Auth token | `httpOnly cookie` | Server-set | Yes (TLS) |
| Refresh token | `httpOnly cookie` | Server-set | Yes (TLS) |

**RULE:** Auth tokens NEVER in `localStorage`. HttpOnly cookies only.

**Zustand persistence example:**

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

export const usePrefsStore = create(
  persist<PrefsState>(
    (set) => ({ theme: 'system', /* ... */ }),
    { name: 'user-preferences' }
  )
);
```

---

# 4. Caching Strategy

## 4.1 Optimistic Updates

```
// Optimistic update pattern with rollback
const mutation = useMutation({
  mutationFn: updateUser,
  onMutate: async (newData) => {
    // Cancel outgoing refetches
    await queryClient.cancelQueries({ queryKey: ['users', userId] });
    // Snapshot current state for rollback
    const snapshot = queryClient.getQueryData(['users', userId]);
    // Optimistically update cache
    queryClient.setQueryData(['users', userId], (old) => ({ ...old, ...newData }));
    return { snapshot };
  },
  onError: (err, vars, context) => {
    // Rollback on error
    queryClient.setQueryData(['users', userId], context?.snapshot);
  },
  onSettled: () => {
    // Always refetch to sync with server
    queryClient.invalidateQueries({ queryKey: ['users', userId] });
  },
```

```
});
```

## 4.2 Cache Invalidation Rules

| Mutation | Invalidates |
|---|---|
| Create user | `['users']` (list) |
| Update user | `['users', userId]` |
| Delete user | `['users']` (list) |
| Update user role | `['users', userId]`, `['permissions']` |

---

# 5. Real-Time State

**Protocol:** `{{WebSocket | Server-Sent Events | None}}` **Library:** `{{Socket.io | native WebSocket | @microsoft/signalr}}`

**Pattern — WebSocket to Query Cache:**

```
// On incoming WS event, update query cache directly
socket.on('user.updated', (user: User) => {
  queryClient.setQueryData(['users', user.id], user);
  queryClient.invalidateQueries({ queryKey: ['users'], exact: false });
});
```

**Connection management:**

- Reconnect with exponential backoff: 1s, 2s, 4s, 8s, 16s, max 30s
- Show "reconnecting" banner in UI after 5s disconnect
- Batch updates: max 50ms batching window to prevent UI thrashing

**TODO:** Document specific WebSocket event schema — reference `event-schema-documentation.md`.

---

# 6. Hydration Strategy (SSR ? Client)

**Approach:** `{{Dehydrate/Hydrate (TanStack Query) | getServerSideProps | prefetchQuery}}`

```
// Server: prefetch and dehydrate
export async function getServerSideProps() {
```

```
  const queryClient = new QueryClient();
  await queryClient.prefetchQuery({
    queryKey: ['users'],
    queryFn: fetchUsers,
  });
  return {
    props: { dehydratedState: dehydrate(queryClient) },
  };
}

// Client: hydrate — no refetch until staleTime expires
export default function Page({ dehydratedState }) {
  return (
    <HydrationBoundary state={dehydratedState}>
      <UserList />
    </HydrationBoundary>
  );
}
```

**Rule:** Prefetch all critical page data on server. No loading spinners on initial navigation.

---

# 7. State Debugging Tools

| Tool | Usage | Enabled In |
|---|---|---|
| TanStack Query Devtools | Inspect cache, queries, mutations | Dev only |
| Zustand devtools middleware | Redux DevTools integration | Dev only |
| React DevTools | Component state tree | Dev only |
| Redux DevTools Extension | If using Redux | Dev only |

**Setup:**

```
// Devtools enabled only in development
const devtools = process.env.NODE_ENV === 'development'
  ? (await import('zustand/middleware')).devtools
  : (f: any) => f;
```

---

# 8. Performance Considerations

## 8.1 Selector Pattern (Zustand)

```
// BAD — subscribes to entire store, re-renders on any change
const { sidebarOpen, theme } = useUIStore();

// GOOD — subscribe to only what the component needs
const sidebarOpen = useUIStore((s) => s.sidebarOpen);
const theme = useUIStore((s) => s.theme);
```

## 8.2 Memoization Rules

| Scenario | Tool | When to Use |
|---|---|---|
| Expensive derived data | `useMemo` | Only if profiling shows issue |
| Stable callback refs | `useCallback` | Only if passed to memoized child |
| Stable component output | `React.memo` | Only if parent re-renders frequently |

**Rule:** Do NOT pre-emptively memoize. Profile first, optimize second.

## 8.3 Query Deduplication

TanStack Query automatically deduplicates identical queries rendered simultaneously. No additional work needed.

**TODO:** Run React Profiler on critical paths and document findings.

---

# Approval

| Role | Name | Date | Signature |
|---|---|---|---|
| Author | | | |
| Frontend Lead | | | |
| Tech Lead | | | |

---

Revision #15
Created 2026-02-23 15:27:29 UTC by John
Updated 2026-05-25 07:32:56 UTC by John