# State Management in 2025

## The Evolution Beyond Redux

React Paris Meetup #012

2025

## Outline

## The State of State Management

### Is Redux Dead?

- Redux downloads: still 8M+/week on npm
- But... alternatives are growing fast
- React 19 changes the game
- Server state vs client state distinction

### What We'll Explore

1. Redux Toolkit evolution
2. Modern alternatives (Zustand, Jotai, Valtio)
3. Server state with TanStack Query
4. React 19 Actions and form state

## Redux Toolkit: Redux Done Right

### The Old Way (Pain)

```
// actions.js
const ADD_TODO = 'ADD_TODO';
export const addTodo = (text) => ({
  type: ADD_TODO,
  payload: { text, id: Date.now() }
});

// reducer.js
function todosReducer(state = [], action) {
  switch (action.type) {
    case ADD_TODO:
```

## Redux Toolkit: Modern Approach

### createSlice to the Rescue

```
import { createSlice } from '@reduxjs/toolkit';

const todosSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    addTodo: (state, action) => {
      // Immer allows "mutations"!
      state.push({
        id: Date.now(),
        text: action.payload,
```

## RTK Query: Built-in Data Fetching

### API Slice Definition

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/c

export const api = createApi({
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  tagTypes: ['Posts'],
  endpoints: (builder) => ({
    getPosts: builder.query({
      query: () => 'posts',
      providesTags: ['Posts']
    }),
    addPost: builder.mutation({
```

## Zustand: Minimal and Flexible

### Simple Store Creation

```
import { create } from 'zustand';

const useStore = create((set, get) => ({
  count: 0,
  todos: [],

  increment: () => set((state) => ({
    count: state.count + 1
  })),

  addTodo: (text) => set((state) => ({
```

## Jotai: Atomic State

### Bottom-Up State Management

```
import { atom, useAtom, useAtomValue } from 'jotai';

// Primitive atoms
const countAtom = atom(0);
const textAtom = atom('');

// Derived atom (read-only)
const doubleCountAtom = atom((get) =>
  get(countAtom) * 2
);
```

## Valtio: Proxy-Based Reactivity

### Mutable State That Just Works

```
import { proxy, useSnapshot } from 'valtio';

// State can be mutated directly
const state = proxy({
  count: 0,
  todos: [],
  user: null
});

// Actions are just functions
const actions = {
```

# TanStack Query: The Server State Standard

## Queries Made Simple

```
import { useQuery, useMutation, useQueryClient } from '@tanst

function Posts() {
  const queryClient = useQueryClient();

  const { data, isLoading, error } = useQuery({
    queryKey: ['posts'],
    queryFn: () => fetch('/api/posts').then(r => r.json()),
    staleTime: 5 * 60 * 1000  // 5 minutes
  });
```

## Server State vs Client State

### The Distinction

| Server State | Client State |
|---|---|
| Lives on the backend | Lives in browser |
| Shared across users | User-specific |
| Needs sync/caching | Ephemeral |
| Posts, users, products | UI state, forms |
| Use Query/SWR | Use Zustand/Context |

### The Insight

Most "global state" is actually server state!

- User data? Server state

## useActionState: Form State Simplified

### The New Pattern

```
"use client";
import { useActionState } from 'react';
import { submitForm } from './actions';

function ContactForm() {
  const [state, formAction, isPending] = useActionState(
    submitForm,
    { message: '', errors: {} }
  );

  return (
```

## useOptimistic: Instant Feedback

### Optimistic UI Updates

```
import { useOptimistic, useTransition } from 'react';

function TodoList({ todos, addTodoAction }) {
  const [isPending, startTransition] = useTransition();
  const [optimisticTodos, addOptimistic] = useOptimistic(
    todos,
    (state, newTodo) => [
      ...state,
      { ...newTodo, sending: true }
    ]
  );
```

## nuqs: Type-Safe URL State

### Search Params as State

```
import { useQueryState, parseAsInteger, parseAsString } from

function ProductFilters() {
  // Synced with URL: ?page=1&sort=price&q=shoes
  const [page, setPage] = useQueryState(
    'page',
    parseAsInteger.withDefault(1)
  );
  const [sort, setSort] = useQueryState(
    'sort',
    parseAsString.withDefault('relevance')
```

## Benefits of URL State

### Why URL State Wins

- Shareable links with exact state
- Browser back/forward works
- Bookmarkable searches/filters
- SEO benefits
- No hydration mismatch
- Survives page refresh

# When to Use What?

## The Decision Tree

```
Do you need state?
  |
  +-- Server data? --> TanStack Query / SWR
  |
  +-- Form state? --> React 19 Actions / react-hook-form
  |
  +-- URL-worthy? --> nuqs / URL params
  |
  +-- Component-local? --> useState / useReducer
  |
  +-- Shared across routes?
```

## Summary

### Key Takeaways

1. Most "state" is actually server state
2. React 19 handles form state natively
3. URL state is underutilized
4. Pick the simplest tool that works
5. Redux isn't dead, but isn't always needed

### Resources

- TanStack Query
- Zustand
- Jotai

## Thank you!
## React Paris Meetup #012