State Management

Last Updated: January 13, 2026 **Related Docs:** ARCHITECTURE.md | ROUTING.md | MASTER-OVERVIEW

Table of Contents

State Architecture Overview

State Distribution Strategy

```
┌─────────────────────────────────────────────────────────────┐
│                     GLOBAL STATE                            │
│               (React Context API)                           │
│                                                              │
│  ┌─────────────────────────────────────────────────┐       │
│  │         ThemeContext                            │       │
│  │  • isDark: boolean                              │       │
│  │  • toggleTheme: function                        │       │
│  │  • theme: 'dark' | 'light'                      │       │
│  │                                                  │       │
│  │  Used by: 20+ components                        │       │
│  │  Persistence: localStorage                      │       │
│  └─────────────────────────────────────────────────┘       │
└─────────────────────────────────────────────────────────────┘
                          │
                          ▼
┌─────────────────────────────────────────────────────────────┐
│                     LOCAL STATE                             │
│                   (useState Hooks)                          │
│                                                              │
│  Component-Scoped State:                                   │
│  • UI interactions (modals, tooltips, expanded cards)      │
│  • Form inputs                                             │
│  • Animation triggers                                      │
│  • Scroll positions                                        │
│  • Mobile detection                                        │
│  • Hover states                                            │
└────────────────────────────────────────────────────────────┘
                          │
                          ▼
┌────────────────────────────────────────────────────────────┐
│                     DERIVED STATE                          │
│                (useMemo, useCallback)                      │
│                                                             │
│  Computed Values:                                          │
│  • Filtered lists                                          │
│  • Calculated positions (physics, timeline)                │
│  • Color interpolations                                    │
│  • Formatted data                                          │
└────────────────────────────────────────────────────────────┘
                          │
                          ▼
┌────────────────────────────────────────────────────────────┐
│                     DATA LOADING                           │
│           (DataContext per report)                         │
│                                                             │
│  Async Data:                                               │
│  • Report JSON data (fetch)                                │
│  • 3D models (GLB files via useGLTF)                       │
│  • Blog posts (import.meta.glob)                           │
│                                                             │
│  Loading States: loading, data, error                      │
└────────────────────────────────────────────────────────────┘
```

Why This Architecture?

Single Global State (Theme): - Only one piece of true global state - Lightweight Context API sufficient - No Redux/Zustand overkill

Local State Dominance: - Most state is component-scoped - Simpler reasoning about data flow - Better performance (no global re-renders)

No Prop Drilling: - Theme accessed via useTheme() hook - Report data via DataContext per report - No passing isDark through 5 components

---

Global State - ThemeContext

Implementation

```
// /src/contexts/ThemeContext.jsx

import React, { createContext, useContext, useState, useEffect } from 'react';

const ThemeContext = createContext();

export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (context === undefined) {
    throw new Error('useTheme must be used within a ThemeProvider');
  }
  return context;
};

export const ThemeProvider = ({ children }) => {
  // Initialize from localStorage to avoid flash
  const [isDark, setIsDark] = useState(() => {
    const savedTheme = localStorage.getItem('portfolio-theme');
    return savedTheme !== null ? savedTheme === 'dark' : true; // Default dark
  });

  const toggleTheme = () => {
    const newIsDark = !isDark;
    setIsDark(newIsDark);
    localStorage.setItem('portfolio-theme', newIsDark ? 'dark' : 'light');
  };

  // Apply theme to body class for CSS
  useEffect(() => {
    if (isDark) {
      document.body.classList.add('dark-theme');
      document.body.classList.remove('light-theme');
    } else {
      document.body.classList.add('light-theme');
      document.body.classList.remove('dark-theme');
    }
  }, [isDark]);

  const value = {
    isDark,
    toggleTheme,
    theme: isDark ? 'dark' : 'light'
  };

  return (
    <ThemeContext.Provider value={value}>
      {children}
    </ThemeContext.Provider>
  );
};
```

Provider Setup

```
// /src/App.jsx

function App() {
  return (
    <ThemeProvider>
      <Router>
        {/* App content */}
      </Router>
    </ThemeProvider>
  );
}
```

Wrapping Order: 1. ThemeProvider (outermost) 2. Router 3. Routes

Why This Order? - Theme affects entire app (including Router) - Router needs theme context available - Routes inherit theme from context

Consumer Pattern

Using the Hook:

```
import { useTheme } from '../../contexts/ThemeContext';

function BackgroundScene() {
  const { isDark } = useTheme();

  // Use theme for conditional rendering
  const shaderMode = isDark ? 2 : 1;

  return (
    <Canvas>
      <TopographicCytoplasmPlane mode={shaderMode} />
      {isDark && <FloatingParticles count={3000} />}
    </Canvas>
  );
}
```

Common Usage Patterns:

```
// Pattern 1: Conditional Rendering
const { isDark } = useTheme();
{isDark && <DarkModeOnlyComponent />}

// Pattern 2: Prop Passing
<ShaderBackground mode={isDark ? 'dark' : 'light'} />

// Pattern 3: CSS Class
<div className={isDark ? 'card-dark' : 'card-light'}>

// Pattern 4: Computed Values
const bgColor = isDark ? '#0b0b0b' : '#f8fafc';

// Pattern 5: Toggle Button
const { toggleTheme } = useTheme();
<button onClick={toggleTheme}>Switch Theme</button>
```

Theme Persistence

localStorage Strategy:

```
// Save on toggle
localStorage.setItem('portfolio-theme', newIsDark ? 'dark' : 'light');

// Load on mount (in useState initializer)
const savedTheme = localStorage.getItem('portfolio-theme');
return savedTheme !== null ? savedTheme === 'dark' : true;
```

Benefits: - Persists across browser sessions - Prevents flash of wrong theme (FOIT) - Respects user preference

Fallback: - If no saved theme → Default to dark mode - Could check `prefers-color-scheme` media query (future enhancement)

Body Class Side Effect

```
useEffect(() => {
  if (isDark) {
    document.body.classList.add('dark-theme');
    document.body.classList.remove('light-theme');
  } else {
    document.body.classList.add('light-theme');
    document.body.classList.remove('dark-theme');
  }
}, [isDark]);
```

CSS Integration:

```
/* /src/styles/index.css */

:root {
  --bg-color: #0b0b0b;  /* Dark default */
  --text-color: white;
}

body.light-theme {
  --bg-color: #f8fafc;  /* Light override */
  --text-color: #1f2937;
}
```

All components automatically update via CSS variables!

Local State Patterns

Pattern 1: UI Interaction State

Modal Open/Close:

```
// /src/components/about/AcademicJourney.jsx

const [modalOpen, setModalOpen] = useState(false);
const [selectedCard, setSelectedCard] = useState(null);

// Open modal
const openModal = (cardIndex) => {
  setSelectedCard(cardIndex);
  setModalOpen(true);

  // Lock scroll
  document.body.classList.add('prevent-scroll');
};

// Close modal
const closeModal = () => {
  setModalOpen(false);
  setSelectedCard(null);

  // Unlock scroll
  document.body.classList.remove('prevent-scroll');
};

return (
  <>
    <TimelineCard onClick={() => openModal(0)} />

    {modalOpen && (
      <Modal onClose={closeModal}>
        <ProteinViewer protein={proteins[selectedCard]} />
      </Modal>
    )}
  </>
);
```

Pattern 2: Form Input State

Contact Form:

```
// /src/components/Contact.jsx

const [formData, setFormData] = useState({
  name: '',
  email: '',
  message: ''
});

const handleChange = (e) => {
  setFormData({
    ...formData,
    [e.target.name]: e.target.value
  });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  // Submit to Formspree or backend
  await fetch('/api/contact', {
    method: 'POST',
    body: JSON.stringify(formData)
  });
};

return (
  <form onSubmit={handleSubmit}>
    <input
      name="name"
      value={formData.name}
      onChange={handleChange}
    />
    {/* ... */}
  </form>
);
```

Pattern 3: Animation Triggers

Scroll Lock in Hero:

```
// /src/components/hero/Hero.jsx

const [scrollLocked, setScrollLocked] = useState(false);
const [inspectMode, setInspectMode] = useState(false);

useEffect(() => {
  if (scrollLocked) {
    document.body.style.overflow = 'hidden';
  } else {
    document.body.style.overflow = 'auto';
  }

  return () => {
    document.body.style.overflow = 'auto';
  };
}, [scrollLocked]);

const handleInspectClick = () => {
  setScrollLocked(true);
  setInspectMode(true);

  // Trigger DNA animation
  // ...
};
```

Pattern 4: Mobile Detection

Responsive State:

```
// Common pattern across components

const [isMobile, setIsMobile] = useState(false);

useEffect(() => {
  const checkMobile = () => {
    setIsMobile(window.innerWidth <= 768);
  };

  checkMobile();
  window.addEventListener('resize', checkMobile);

  return () => window.removeEventListener('resize', checkMobile);
}, []);

// Use in rendering
return (
  <Canvas camera={{
    position: [0, 0, isMobile ? 10 : 15],
    fov: isMobile ? 60 : 50
  }}>
    {/* ... */}
  </Canvas>
);
```

Pattern 5: Hover States

Protein Structure Highlighting:

```
// /src/components/about/AcademicJourney.jsx

const [hoveredStructure, setHoveredStructure] = useState(null);

return (
  <div>
    <p>
      The <span
```

```
        onMouseEnter={() => setHoveredStructure('Struct_Helix')}
        onMouseLeave={() => setHoveredStructure(null)}
      >
        alpha-helix
      </span> wraps around the DNA.
    </p>

    <ProteinViewer highlightStructure={hoveredStructure} />
  </div>
);
```

Data Loading Patterns

Pattern 1: DataContext for Reports

HearingLoss DataContext:

```
// /src/components/projects/HearingLoss/data/DataContext.jsx

import React, { createContext, useContext, useState, useEffect } from 'react';

const DataContext = createContext();

export const useData = () => {
  const context = useContext(DataContext);
  if (!context) {
    throw new Error('useData must be used within DataProvider');
  }
  return context;
};

export const DataProvider = ({ children }) => {
  const [data, setData] = useState({});
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const loadData = async () => {
      try {
        const [demographics, riskFactors, forestPlot] = await Promise.all([
          fetch('/data/hearing-loss/demographics.json').then(r => r.json()),
          fetch('/data/hearing-loss/risk_factors.json').then(r => r.json()),
          fetch('/data/hearing-loss/forest_plot.json').then(r => r.json()),
        ]);

        setData({ demographics, riskFactors, forestPlot });
        setLoading(false);
      } catch (err) {
        setError(err.message);
        setLoading(false);
      }
    };

    loadData();
  }, []);

  return (
    <DataContext.Provider value={{ data, loading, error }}>
      {children}
    </DataContext.Provider>
  );
};
```

Report Component:

```
// /src/components/projects/HearingLoss/index.jsx

import { DataProvider, useData } from './data/DataContext';

function HearingLossContent() {
  const { data, loading, error } = useData();

  if (loading) return <div>Loading data...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <>
      <DemographicsSection data={data.demographics} />
      <RiskFactorsSection data={data.riskFactors} />
      <ForestPlotSection data={data.forestPlot} />
    </>
  );
}

export default function HearingLossReport() {
  return (
    <DataProvider>
```

```
      <HearingLossContent />
    </DataProvider>
  );
}
```

Pattern 2: 3D Model Loading

Protein GLB Loading:

```
// /src/components/about/ProteinViewer.jsx

import { useGLTF } from '@react-three/drei';
import { Suspense } from 'react';

function ProteinModel({ modelPath }) {
  const { scene } = useGLTF(modelPath);
  return <primitive object={scene} />;
}

export default function ProteinViewer({ modelPath }) {
  return (
    <Canvas>
      <Suspense fallback={null}>
        <ProteinModel modelPath={modelPath} />
      </Suspense>
    </Canvas>
  );
}

// Preload models for instant rendering
useGLTF.preload('/assets/HistoneH1_V2.glb');
useGLTF.preload('/assets/GFP_v2.glb');
```

Pattern 3: Blog Post Discovery

Auto-Discovery with Vite:

```
// /src/posts/load.js

const postModules = import.meta.glob('./**/*.{jsx,mdx}', { eager: true });

export const posts = Object.entries(postModules).map(([path, module]) => ({
  path,
  meta: module.meta,
  content: module.default
}));
```

Usage:

```
// /src/components/blog/BlogSection.jsx

import { posts } from '../../posts/load';

export default function BlogSection() {
  return (
    <div className="blog-grid">
      {posts.map(post => (
        <BlogItem key={post.meta.slug} post={post} />
      ))}
    </div>
  );
}
```

Custom Hooks

useScrollGradient

Purpose: Interpolate colors based on scroll position between sections

Implementation:

```
// /src/hooks/useScrollGradient.js

import { useEffect, useState } from "react";

export default function useScrollGradient(sections, colors) {
  const [color, setColor] = useState(colors[0] || [0, 0, 0]);

  useEffect(() => {
    // Get DOM elements from section IDs
    const elements = sections
      .map((sec) => typeof sec === "string" ? document.getElementById(sec) : sec)
      .filter(Boolean);

    if (!elements.length) return;
    function update() {
      const scrollY = window.scrollY;
      const offsets = elements.map((el) => el.offsetTop);

      // Find current section index
      let index = 0;
      while (index < offsets.length - 1 && scrollY >= offsets[index + 1]) {
        index += 1;
      }

      // Calculate interpolation factor
      const start = offsets[index];
      const end = index < offsets.length - 1 ? offsets[index + 1] : document.body.scrollHeight;
      const t = Math.min(Math.max((scrollY - start) / (end - start), 0), 1);

      // Interpolate between current and next color
      const c1 = colors[index];
      const c2 = colors[Math.min(index + 1, colors.length - 1)];

      setColor([
        c1[0] + (c2[0] - c1[0]) * t,
        c1[1] + (c2[1] - c1[1]) * t,
        c1[2] + (c2[2] - c1[2]) * t,
      ]);
    }

    update();
    window.addEventListener("scroll", update);
    window.addEventListener("resize", update);

    return () => {
      window.removeEventListener("scroll", update);
      window.removeEventListener("resize", update);
    };
  }, [sections, colors]);

  return color;
}
```

Usage:

```
// /src/components/BackgroundScene.jsx

import useScrollGradient from '../hooks/useScrollGradient';
```

```
export default function BackgroundScene() {
  const sections = ["hero", "about", "work", "blog", "contact"];
  const colors = [
    [0.03, 0.03, 0.03], // hero - Charcoal
    [0.03, 0.03, 0.03], // about - Charcoal
    [0.03, 0.03, 0.03], // work - Charcoal
    [0.03, 0.03, 0.03], // blog - Charcoal
    [0.03, 0.03, 0.03], // contact - Charcoal
  ];

  const scrollColor = useScrollGradient(sections, colors);

  // Pass color to shader
  useEffect(() => {
    if (fogRef.current) {
      fogRef.current.setBaseColor(scrollColor);
    }
  }, [scrollColor]);
}
```

useTheme (Already Covered)

Purpose: Access theme context without prop drilling

Usage:

```
const { isDark, toggleTheme, theme } = useTheme();
```

State Flow Examples

Example 1: Theme Toggle Flow

```
User clicks ThemeToggle button
         ↓
toggleTheme() called in ThemeContext
         ↓
1. setIsDark(!isDark) - Update React state
2. localStorage.setItem('portfolio-theme', newTheme) - Persist
         ↓
useEffect triggers on isDark change
         ↓
document.body.classList.toggle('light-theme')
```

```
         ↓
All components consuming useTheme() re-render
         ↓
├─ BackgroundScene switches shader mode
├─ D3 charts update color scales
├─ Timeline dots change gradients
├─ CSS variables switch via body class
└─ Buttons update hover states
```

Example 2: Modal Open Flow

```
User clicks timeline card
         ↓
openModal(cardIndex) called
         ↓
1. setSelectedCard(cardIndex) - Store which card
2. setModalOpen(true) - Show modal
3. document.body.classList.add('prevent-scroll') - Lock scroll
         ↓
Component re-renders
         ↓
{modalOpen && <Modal />} - Conditional render
         ↓
Modal uses ReactDOM.createPortal
         ↓
Rendered outside component hierarchy (top-level)
         ↓
ProteinViewer loads GLB model
         ↓
useGLTF fetches model (if not cached)
         ↓
Suspense shows loading fallback
         ↓
Model renders in Three.js Canvas
```

Example 3: Scroll Gradient Flow

```
User scrolls page
         ↓
'scroll' event fires
         ↓
useScrollGradient update() function called
         ↓
1. Calculate current scroll position
```

```
2. Find which section user is in
3. Calculate interpolation factor (0-1)
4. Interpolate between section colors
         ↓
setColor([r, g, b]) - Update hook state
         ↓
BackgroundScene re-renders with new color
         ↓
useEffect in BackgroundScene triggers
         ↓
fogRef.current.setBaseColor(scrollColor)
         ↓
Shader uniform updated in Three.js
         ↓
GPU re-renders shader with new color
         ↓
Smooth color transition visible to user
```

Best Practices

1. Colocate State

Bad: State at top level when only used in one component

```jsx
// App.jsx
const [modalOpen, setModalOpen] = useState(false);
<AcademicJourney modalOpen={modalOpen} setModalOpen={setModalOpen} />
```

Good: State in component that uses it

```jsx
// AcademicJourney.jsx
const [modalOpen, setModalOpen] = useState(false);
```

2. Lift State Only When Needed

When to lift: - Multiple siblings need to share state - Parent needs to coordinate children

When NOT to lift: - State only used in one component - No sibling communication needed

3. Use Derived State

Bad: Storing computed values in state

```
const [data, setData] = useState([]);
const [filteredData, setFilteredData] = useState([]);

useEffect(() => {
  setFilteredData(data.filter(item => item.active));
}, [data]);
```

Good: Compute on render

```
const [data, setData] = useState([]);
const filteredData = useMemo(
  () => data.filter(item => item.active),
  [data]
);
```

4. Avoid State Duplication

Bad: Copying props to state

```
function ProteinViewer({ initialProtein }) {
  const [protein, setProtein] = useState(initialProtein);
  // If initialProtein changes, state doesn't update!
}
```

Good: Use props directly or key

```
function ProteinViewer({ protein }) {
  // Just use the prop
}

// Or force remount with key
<ProteinViewer key={protein.id} protein={protein} />
```

5. Initialize State from Functions

Bad: Reading localStorage on every render

```
const [theme, setTheme] = useState(localStorage.getItem('theme') || 'dark');
```

Good: Lazy initialization

```
const [theme, setTheme] = useState(() => {
  return localStorage.getItem('theme') || 'dark';
});
```

6. Clean Up Side Effects

Bad: Memory leaks from event listeners

```
useEffect(() => {
  window.addEventListener('scroll', handleScroll);
  // Missing cleanup!
}, []);
```

Good: Return cleanup function

```
useEffect(() => {
  window.addEventListener('scroll', handleScroll);
  return () => window.removeEventListener('scroll', handleScroll);
}, []);
```

Performance Considerations

Avoid Unnecessary Re-renders

Problem: Theme context causes all consumers to re-render

Solution: Split context if needed

```
const ThemeStateContext = createContext();
const ThemeUpdateContext = createContext();

// Components that only need toggleTheme don't re-render on isDark change
```

Memoize Expensive Calculations

```
const processedData = useMemo(() => {
  return data.map(item => expensiveTransform(item));
}, [data]);
```

Throttle/Debounce Event Handlers

```
import { throttle } from 'lodash';

useEffect(() => {
  const handleScroll = throttle(() => {
    // Handle scroll
  }, 100); // Update max once per 100ms

  window.addEventListener('scroll', handleScroll);
  return () => {
    window.removeEventListener('scroll', handleScroll);
    handleScroll.cancel(); // Cancel pending throttled calls
  };
}, []);
```

Related Documentation

- ARCHITECTURE.md - Overall application architecture
- ROUTING.md - Routing system
- THREE-JS-COMPONENTS.md *(coming soon)* - 3D component state
- D3-VISUALIZATIONS.md *(coming soon)* - Chart data flow

This state management approach balances simplicity, performance, and maintainability for a highly interactive portfolio.