Projects Portfolio - Architecture Guidelines

Vision

Build a scalable portfolio of scientific reports and interactive projects that integrate seamlessly with the main site while maintaining clean boundaries and consistent user experience.

Why This Architecture?

Problems We're Solving:

- **CSS Collisions**: Site-level styles (Three.js, global variables, theme system) conflicting with project-specific styles
- **Scalability**: Need to build multiple reports/projects without repeating setup work
- **Performance**: 3D background + multiple data-heavy reports require careful code splitting
- **Consistency**: Projects should feel cohesive while allowing creative freedom

Our Solutions:

- **Reports Platform**: Shared foundation (`reports-core/`) for consistent theming, components, and utilities
- **Namespace Isolation**: Each project lives in its own CSS/JS scope
- **Theme Integration**: Projects harmonize with site theme without global conflicts
- **Lazy Loading**: Projects load only when visited

Directory Structure

```
src/components/projects/
├── PROJECTS_ARCHITECTURE.md     # This file
├── reports-core/                # Shared foundation for all reports
│   ├── tokens/                  # CSS variables, theme bridge
│   ├── styles/                  # Base reset, shared styles
│  ├── components/          # Reusable components (ChartFrame, Tooltip, etc.)
│   └── utils/                 # Chart helpers, formatters, data transforms
├── HearingLoss/                 # Individual report projects
│   ├── REPORT_PLAN.md           # Project-specific plan
│   ├── index.jsx                # Entry point
│   └── ... (project structure)
```

```
├── GeneticsReport/              # Future report
├── BiotechAnalysis/             # Future project
└── InteractiveDemo/             # Non-report interactive projects
```

Styling Strategy

The "Namespace + Bridge" Pattern:

1. **Site Level**: Global theme variables (`--bg-color`, `--text-color`)
2. **Platform Level**: Shared report tokens (`--rp-bg`, `--rp-text`, `--rp-accent`)
3. **Project Level**: Specific tokens (`--hl-primary`, `--genetics-highlight`)

CSS Isolation Rules:

- **DO**: Use wrapper classes (`.hl-root`, `.genetics-root`)
- **DO**: Prefix project variables (`--hl-*`, `--genetics-*`)
- **DO**: Apply resets only within project scope
- **DON'T**: Modify global styles from within projects
- **DON'T**: Use bare element selectors (`h2 {}` → `.hl-root h2 {}`)

Data Management Patterns

For Reports (JSON-heavy):

- Static JSON files in `/public/data/project-name/`
- Simple DataContext for loading + sharing
- Pre-aggregate heavy computations in Python/R
- Memoize derived values in React

For Interactive Projects:

- Local state with useState/useReducer
- Custom hooks for complex logic
- External APIs only if needed

Performance Guidelines

Code Splitting:

```javascript
// In App.jsx route definitions:
const HearingLossReport = lazy(() => import('./projects/HearingLoss'));
const GeneticsReport = lazy(() => import('./projects/GeneticsReport'));
```

Asset Optimization:

- Tree-shake D3 imports: `import { scaleLinear } from 'd3-scale'`
- Optimize JSON file sizes (pre-aggregate in Python)
- Use CSS custom properties for dynamic theming (not JS)

Theme Integration

The Theme Bridge Pattern:

```css
/* In reports-core/tokens/bridge.css */
.rp-root {
  /* Map site theme to platform tokens */
  --rp-bg: var(--bg-color);
  --rp-text: var(--text-color);
  --rp-accent: var(--project-link-color);
}

/* Projects inherit and extend */
.hl-root {
  --hl-primary: var(--rp-accent);
  --hl-secondary: #your-custom-color;
}
```

Responsive Strategy

Mobile-Later Approach:

1. **Phase 1**: Build desktop-first with flexible layouts
2. **Phase 2**: Adapt key breakpoints for mobile
3. **Guidelines**: Use `grid`/`flex`, avoid hover-only interactions

Project Types & Patterns

Data Reports (like HearingLoss):

- Heavy on charts, statistics, methodology
- Pre-processed JSON data
- Section-based structure
- Interactive cards/tooltips

Interactive Demos (future):

- Real-time interactions
- WebGL/Canvas elements
- User input handling
- Performance-critical

Case Studies (future):

- Mixed content (text, images, interactive elements)
- Lighter data requirements
- Narrative structure

Development Workflow

Starting a New Project:

1. Copy template structure from `reports-core/template/`
2. Create project-specific namespace (`.your-project-root`)
3. Set up route in `App.jsx` with lazy loading
4. Define project tokens extending platform tokens
5. Build incrementally, test isolation early

Shared Component Updates:

- Update `reports-core/` components
- Test across all existing projects
- Version shared components if breaking changes needed

Success Metrics

- **Isolation**: No CSS/JS conflicts between projects and site
- **Performance**: Each project adds <100KB gzipped to bundle
- **Consistency**: Projects feel cohesive with site theme
- **Developer Experience**: New projects start from proven template
- **Scalability**: Can add new projects without refactoring existing ones

---

This architecture evolves as we build. Update this document when patterns change.