AI Prompt Library

for Design Systems

Copy-paste prompts for documentation, audits, and code generation

By Petri Lahdelma - 2026 Edition

DOCUMENTATION

Component Documentation

Generate documentation for a [ComponentName] component. Include: 1) One-line description, 2) When to use / when not to use, 3) Props table with types and defaults, 4) Accessibility considerations, 5) Code examples for each variant.

Changelog Entry

Write a changelog entry for version [X.Y.Z]. Changes: [list changes]. Format: Summary sentence, then bullet points grouped by Added/Changed/Fixed/Deprecated. Keep it scannable.

CODE GENERATION

React Component from Figma

Create a React TypeScript component matching this Figma spec: [paste Figma specs]. Use these design tokens: [token names]. Include: TypeScript types, forwardRef, displayName, and Storybook story.

Unit Tests

Write unit tests for [ComponentName] using React Testing Library. Test: 1) Default rendering, 2) All prop variants, 3) User interactions, 4) Accessibility (role, aria). Aim for 90%+ coverage.

AUDITS

Accessibility Review

Review this component for WCAG 2.1 AA compliance: [paste code]. Check: keyboard navigation, focus management, ARIA attributes, color contrast, screen reader compatibility. List issues with severity and fix suggestions.

Token Usage Audit

Analyze this code for design token usage: [paste code]. Find: 1) Hard-coded colors (hex, rgb), 2) Hard-coded spacing, 3) Non-system typography. For each violation, suggest the correct semantic token.

MIGRATION

Token Migration

Create a codemod to migrate from [old token format] to [new token format]. Handle: color tokens, spacing tokens, typography tokens. Output jscodeshift transform with before/after examples.

Component API Update

```
Write a migration guide for [ComponentName] from v[X] to v[Y]. Breaking changes: [list]. Include: 1) What changed and why, 2) Before/after code examples, 3) Automated codemod if possible.
```

Usage Tips

1. Always review AI output before shipping - it's a draft, not final
2. Provide context: paste relevant code, specs, or docs
3. Be specific about output format (TypeScript, JSDoc, Markdown)
4. Iterate: if the first output isn't right, refine the prompt
5. Save successful prompts to your team's prompt library