RFC: Adopt Specify Design Tokens for Policy/ Test UI & Documentation

Status: Draft
Author: System Agent
Date: 2025-09-18
Related Issues: TBD (will be created)

Summary

This RFC proposes adopting Specify (https://specifyapp.com/) design tokens to establish a unified design system for the UtilityFog-Fractal-TreeOpen project's policy interfaces, test UI components, and documentation. This will provide consistent theming, improved maintainability, and better developer experience across all visual components.

Motivation

Currently, the project lacks a centralized design system, leading to:
- Inconsistent styling across policy interfaces and test UIs
- Hardcoded design values scattered throughout the codebase
- Difficulty maintaining visual consistency as the project grows
- No systematic approach to theming and branding

Design tokens will solve these issues by providing a single source of truth for all design decisions.

Detailed Design

4-Step Implementation Flow

Step 1: Specify Integration & Token Definition

- Set up Specify workspace and API integration
- Define minimal PoC token set covering core design elements
- Establish token naming conventions and organization structure
- Create initial token categories: colors, spacing, typography, and semantic tokens

Step 2: Planning & Architecture

- Design token pipeline architecture (pull → transform → publish)
- Plan integration points with existing systems
- Define tooling requirements and development workflow
- Create comprehensive implementation roadmap

Step 3: Implementation & Integration

- Build automated token pipeline and transformation tools
- Integrate tokens into policy UI components and test interfaces
- Update documentation system to use design tokens
- Implement theme switching capabilities

Step 4: Testing & Validation

- Comprehensive testing of token pipeline and integrations
- Visual regression testing for UI consistency
- Performance impact assessment
- Documentation and developer experience validation

Minimal PoC Token Set

The initial proof-of-concept will include:

Color Tokens

```
{
  "color": {
    "primary": {
      "50": "#f0f9ff",
      "500": "#3b82f6",
      "900": "#1e3a8a"
    },
    "semantic": {
      "success": "#10b981",
      "warning": "#f59e0b",
      "error": "#ef4444",
      "info": "#3b82f6"
    },
    "neutral": {
      "50": "#f9fafb",
      "500": "#6b7280",
      "900": "#111827"
    }
  }
}
```

Spacing Tokens

```
{
  "spacing": {
    "xs": "0.25rem",
    "sm": "0.5rem",
    "md": "1rem",
    "lg": "1.5rem",
    "xl": "2rem",
    "2xl": "3rem"
  }
}
```

Typography Tokens

```
{
  "font": {
    "family": {
      "sans": ["Inter", "system-ui", "sans-serif"],
      "mono": ["JetBrains Mono", "monospace"]
    },
    "size": {
      "xs": "0.75rem",
      "sm": "0.875rem",
      "base": "1rem",
      "lg": "1.125rem",
      "xl": "1.25rem",
      "2xl": "1.5rem"
    },
    "weight": {
      "normal": "400",
      "medium": "500",
      "semibold": "600",
      "bold": "700"
    }
  }
}
```

Technical Implementation

Token Pipeline Architecture

```
Specify API → Token Fetcher → Transformer → Publisher → Consumers
     ↓              ↓              ↓            ↓           ↓
  Raw JSON    →  Normalized  →  Platform  →  CSS/JS  →  UI Components
                    JSON        Specific     Variables    Documentation
                                Formats                   Test Interfaces
```

Directory Structure

```
tools/specify/
├── config/
│   ├── specify.config.js      # Specify API configuration
│   └── transform.config.js    # Token transformation rules
├── scripts/
│   ├── fetch-tokens.js        # Pull tokens from Specify API
│   ├── transform-tokens.js    # Transform tokens for different platforms
│   ├── publish-tokens.js      # Publish tokens to consumers
│   └── validate-tokens.js     # Token validation and testing
├── templates/
│   ├── css-variables.hbs      # CSS custom properties template
│   ├── js-tokens.hbs          # JavaScript tokens template
│   └── scss-variables.hbs     # SCSS variables template
└── output/
    ├── tokens.css             # Generated CSS custom properties
    ├── tokens.js              # Generated JavaScript tokens
    └── tokens.scss            # Generated SCSS variables
```

Integration Points

1. **Policy UI Components**
   - Apply tokens to existing policy interface elements
   - Ensure consistent theming across all policy-related UIs
   - Maintain accessibility standards with semantic color tokens

2. **Test Interfaces**
   - Style test result displays and interactive elements
   - Provide consistent visual feedback for test outcomes
   - Integrate with existing test infrastructure

3. **Documentation System**
   - Apply design tokens to documentation styling
   - Create living style guide showcasing token usage
   - Ensure documentation reflects current design system

Safeguards & Constraints

No Policy Behavior Changes

- This RFC focuses purely on visual/UI improvements
- No changes to policy logic, evaluation, or behavior
- Existing functionality remains completely unchanged

Maintain Existing Safeguards

- All current branch protection settings remain in place
- CI requirements (green tests + 1 review) continue to apply
- No changes to security or safety mechanisms

Infrastructure Focus

- Initial phase concentrates on tooling and pipeline setup
- Actual UI changes will be implemented in subsequent phases
- Comprehensive testing before any visual modifications

Implementation Plan

Phase 1: RFC & Infrastructure (Current)

- [ ] Create RFC document (this document)
- [ ] Set up basic tooling infrastructure
- [ ] Create related implementation issues
- [ ] Establish development workflow

Phase 2: Token Pipeline Development

- [ ] Implement Specify API integration
- [ ] Build token transformation pipeline
- [ ] Create automated publishing workflow
- [ ] Develop validation and testing tools

Phase 3: Theme Playbook Development

- [ ] Create comprehensive theme guidelines
- [ ] Develop component styling patterns
- [ ] Establish accessibility standards
- [ ] Build developer documentation

Phase 4: Policy Self-Test CLI

- [ ] Integrate design tokens with local OPA evaluation
- [ ] Create CLI tools for policy testing with consistent UI
- [ ] Ensure seamless developer experience

Success Metrics

- **Consistency:** All UI components use design tokens instead of hardcoded values
- **Maintainability:** Design changes can be made by updating tokens, not individual components
- **Developer Experience:** Clear documentation and tooling for working with design tokens
- **Performance:** No negative impact on application performance
- **Accessibility:** Improved accessibility through semantic token usage

Alternatives Considered

1. **Manual CSS Variables:** Less systematic, harder to maintain at scale
2. **Component-Level Styling:** Leads to inconsistency and duplication
3. **Other Design Token Tools:** Specify chosen for its comprehensive API and transformation capabilities

Open Questions

1. Should we implement theme switching (light/dark mode) in the initial phase?
2. How should we handle token versioning and backwards compatibility?
3. What level of customization should be available to end users?

References

- Specify Documentation (https://docs.specifyapp.com/)
- Design Tokens Community Group (https://design-tokens.github.io/community-group/)
- W3C Design Tokens Format Module (https://tr.designtokens.org/format/)

Next Steps:
1. Review and approve this RFC
2. Create implementation issues for each phase
3. Set up Specify workspace and initial token definitions
4. Begin tooling infrastructure development