[UI/UX THREAD] Focus on: User interface design, user experience, component layout, styling, accessibility, and frontend implementation. Consider usability, responsiveness, and visual consistency.

Implementation Request

RICE FACT Framework

Role: You are a Senior UI/Frontend Developer specializing in user experience, accessibility, and modern frontend patterns.

Instruction: Implement the user interface components for this user story. Create clean, reusable components with proper styling and user interactions.

Context: This is the UI implementation thread. The backend may be implemented separately, so focus on the presentation layer and use mocks or stubs for data if needed.

Example: Follow existing component patterns in the codebase. Reference similar UI elements for consistency in styling and behavior.

Format: Provide complete component code with HTML structure, CSS/styling, and JavaScript interactions. Include accessibility attributes (ARIA labels, keyboard navigation).

Aim: Create an intuitive, responsive, and visually consistent user interface that provides excellent user experience.

Constraints: Follow existing design patterns and component architecture. Ensure accessibility (WCAG 2.1). Do not modify backend logic or data models.

Tone: User-focused and detail-oriented. Prioritize clarity, usability, and visual polish.

User Story

Include GUI Definition in Prompt Context

As a user, I want to include a GUI definition in my prompt context so that Claude understands the visual design I have in mind for implementation

Acceptance Criteria

1. User can select from available GUI definitions when composing a prompt
2. Selected GUI definition is included in the context sent to Claude
3. GUI definition is formatted in a way Claude can interpret for implementation guidance
4. Multiple GUI definitions can be included if needed for complex features

Approved Implementation Plan

The following plan was approved for this sprint. Please follow this guidance:

Based on my thorough exploration of the codebase, I now have a comprehensive understanding of the existing architecture. Let me provide you with a detailed implementation plan.

Designer Plugin Implementation Plan

Architecture Analysis

Current State

The codebase already has a robust plugin architecture in `src/main/plugins/`: - **PluginManager**: Orchestrates lifecycle (activate/deactivate/enable/disable) - **PluginContext**: Provides sandboxed API for plugins with storage, logging, IPC handlers - **PluginRegistry**: Central registry for actions, acceptors, reactors, components - **PluginStateStore**: Persists plugin enabled/disabled state in `~/.puffin/plugin-state.json`

GUI definitions are currently stored in `.puffin/gui-definitions/` via `PuffinState` (`puffin-state.js:28-29, 542-634`). The existing methods include: - `saveGuiDefinition(name, description, elements)` - `listGuiDefinitions()` - `loadGuiDefinition(filename)` - `updateGuiDefinition(filename, updates)` - `deleteGuiDefinition(filename)`

Prompt context is built in `prompt-editor.js` and sent via `window.puffin.claude.submit()`, which includes a `guiDescription` field that formats GUI elements for Claude.

Key Insight: Plugin vs Core Decision

The stories request a **Designer Plugin** directory structure (`.puffin/plugins/designer/designs/`), but the current GUI definitions are stored in core Puffin state (`.puffin/gui-definitions/`).

Two possible approaches:

1. **Migrate to Plugin Architecture**: Create a true "designer" plugin that owns its storage

2. **Rename Directory Only**: Keep the existing `PuffinState` patterns but change the directory path

Recommendation: Option 1 - Full plugin architecture. This aligns with the existing plugin infrastructure and makes the designer feature truly portable and self-contained.

Implementation Order

Order	Story	Rationale
1	Story 1: Directory Structure	Foundation - creates the storage layer
2	Story 2: Namespace Enforcement	Builds on storage with validation
3	Story 3: Prompt Context	Uses the storage from Stories 1 & 2

Story 1: Designer Plugin Directory Structure

Technical Approach

Create a Designer Plugin that follows the existing plugin architecture patterns.

Complexity: Medium

Key Technical Decisions

1. **Storage Location**: `.puffin/plugins/designer/designs/` (per user story requirement)
2. **File Format**: JSON with schema: `{ name, elements, createdAt, lastModifiedAt }`

3. **Plugin Registration**: Register via `PluginManager` with IPC handlers for CRUD operations
4. **Initialization**: Plugin creates `designs/` directory on activation if missing

Files to Create

File	Purpose
`src/main/plugins/designer/index.js`	Plugin main module with activate/deactivate
`src/main/plugins/designer/designer-storage.js`	Storage service for design files
`src/main/plugins/designer/manifest.json`	Plugin manifest

Files to Modify

File	Changes
`src/main/ipc-handlers.js`	Register designer plugin IPC handlers
`src/renderer/components/gui-designer/gui-designer.js`	Update save/load to use plugin IPC

Implementation Details

```
DesignerStorage class:
  - constructor(pluginDataPath)  // .puffin/plugins/designer
  - ensureDesignsDirectory()     // creates designs/ if missing
  - saveDesign(name, elements)   // returns { filename, design }
  - loadDesign(filename)         // returns design object
  - listDesigns()                // returns array of design metadata
  - updateDesign(filename, updates)
  - deleteDesign(filename)
```

Story 2: GUI Definition Namespace Enforcement

Technical Approach

Add uniqueness validation at the storage layer with clear error messaging.

Complexity: Low

Key Technical Decisions

1. **Validation Point**: In `DesignerStorage.saveDesign()` before writing
2. **Conflict Check**: Scan existing filenames, compare sanitized names
3. **Error Type**: Throw descriptive error with suggestion (e.g., "Design 'Login Form' already exists")
4. **Rename Support**: `renameDesign(oldFilename, newName)` with same uniqueness check

Files to Modify

File	Changes
`src/main/plugins/designer/designer-storage.js`	Add `checkNameUniqueness()`, update `saveDesign()`
`src/renderer/components/gui-designer/gui-designer.js`	Handle duplicate name errors in UI

Implementation Details

```
// In designer-storage.js
async checkNameUniqueness(name, excludeFilename = null) {
  const sanitized = this.sanitizeFilename(name)
  const existing = await this.listDesigns()
  const conflict = existing.find(d =>
    d.filename !== excludeFilename &&
    this.sanitizeFilename(d.name) === sanitized
  )
  if (conflict) {
    throw new Error(`A design named "${conflict.name}" already exists`)
  }
}

async renameDesign(filename, newName) {
  await this.checkNameUniqueness(newName, filename)
  // Rename logic...
}
```

Story 3: Include GUI Definition in Prompt Context

Technical Approach

Extend the existing prompt submission flow to include selected GUI definitions.

Complexity: Low-Medium

Key Technical Decisions

1. **Selection UI**: Already exists - `includeGuiBtn` dropdown in `prompt-editor.js`
2. **Data Flow**: Load design from plugin storage → format for Claude → include in submit payload
3. **Formatting**: Use existing `buildGuiDescription()` method
4. **Multiple Definitions**: Allow array of selected definitions (accordion in dropdown)

Files to Modify

File	Changes
`src/renderer/components/prompt-editor/prompt-editor.js`	Fetch from designer plugin, support multiple selection
`src/main/claude-service.js`	Already supports `guiDescription` - no changes needed

Implementation Details

The existing code in `prompt-editor.js:869-983` already handles: - Dropdown toggle - Loading saved definitions via `window.puffin.state.listGuiDefinitions()` - Selecting and including in prompt

Changes needed: 1. Update IPC calls to use designer plugin channel (`plugin:designer:listDesigns`) 2. Add multi-select checkboxes (like the docs dropdown pattern at line 1017-1176) 3. Combine multiple definitions in `buildGuiDescription()`

Risk Assessment

Risk	Impact	Mitigation
Migration of existing definitions	Medium	Add migration script in plugin activation to copy from `.puffin/gui-definitions/`
Plugin activation order	Low	Designer plugin has no dependencies on other plugins
Backward compatibility	Medium	Keep old IPC handlers temporarily, deprecate after 1 version

---

Shared Components

All three stories share: - `DesignerStorage` class (created in Story 1, extended in Story 2) - Designer plugin manifest and lifecycle - IPC channel namespace: `plugin:designer:*`

Summary

Story	Complexity	Key Files	Dependencies
1: Directory Structure	Medium	3 new, 2 modified	None
2: Namespace Enforcement	Low	2 modified	Story 1
3: Prompt Context	Low-Medium	1 modified	Stories 1 & 2

The implementation follows existing patterns in the codebase (plugin architecture, IPC channels, SAM state management) and requires minimal new abstractions. Total estimated files: **3 new, 4 modified**.

Would you like me to proceed with creating a detailed implementation plan file, or would you prefer to discuss any aspect of this architecture first?

Implementation Notes

This is part of an approved sprint plan. Please implement this story following the established patterns in the codebase and the guidance above.

Please proceed with the implementation.