UI Studio

Product Requirements Document

Author	Ajay Mehta	**Status**	Draft	**Updated**	February 7, 2026

PROBLEM STATEMENT

Product designers and PMs at the organization cannot quickly turn design ideas into interactive, coded prototypes that use the company's real design system. Today, creating a high-fidelity prototype requires either engineering resources (expensive, slow, competes with roadmap work) or generic prototyping tools like Figma or Framer that produce output disconnected from the actual component library. This means design reviews happen on static mockups, stakeholder demos lack interactivity, and design-to-engineering handoff introduces drift because the prototype was never grounded in real components.

The cost of inaction is compounding: every sprint, designers wait on engineering for throwaway prototypes, engineers context-switch to build demos, and shipped UIs deviate from the design system because the source of truth was never used in exploration.

GOALS

1. **Reduce time-to-prototype by 80%** — A designer should go from idea (or screenshot) to a clickable, multi-page prototype in under 30 minutes, down from the current multi-day cycle involving engineering support.
2. **100% design system fidelity** — Every generated prototype uses the organization’s actual design system components and tokens, eliminating drift between prototype and production.
3. **Zero engineering dependency for prototyping** — PMs and designers can create, iterate, and share coded prototypes without filing a single engineering ticket.
4. **Adoption by 70%+ of the design and PM team within 60 days** — UI Studio should become the default prototyping tool, not a novelty.
5. **Reduce design-to-dev handoff friction** — Since prototypes are already built on real components, the handoff artifact is closer to production code, reducing implementation misinterpretation.

NON-GOALS

1. **Production code generation** — UI Studio produces prototypes, not shippable production code. While output uses real components, it is not expected to meet production standards for error handling, accessibility compliance, performance, or test coverage.

Separate initiative; premature to conflate prototyping with code generation.

2. **Real backend integration** — Prototypes use a generated mock data layer, not live APIs or databases.

Connecting to real backends introduces auth, security, and data integrity concerns out of scope for a prototyping tool.

3. **Replacing Figma or the design tool of record** — UI Studio complements the design workflow; it does not replace wireframing, visual exploration, or design system documentation.

Different stage of the design process.

4. **Multi-user real-time collaboration** — V1 is single-user. Sharing happens by exporting or sending a link to a running prototype.

Collaborative editing is complex; ship single-player first and learn.

5. **Cross-organization or white-label deployment** — V1 is built for internal use with a single design system.

Generalization adds complexity without validated demand.

USER STORIES

Designer Persona

- As a product designer, I want to **paste a screenshot or wireframe into a chat** and get back an interactive prototype built with our design system, so that I can validate layout and interaction ideas in minutes instead of days.
- As a product designer, I want to **give natural-language feedback** and see the prototype update, so that I can iterate without writing code.
- As a product designer, I want to **build multi-page prototypes with working navigation**, so that I can demonstrate complete user flows during design reviews.
- As a product designer, I want the agent to **automatically use our design system’s components, tokens, and patterns**, so that prototypes look and behave like the real product.
- As a product designer, I want to **upload a Figma export or design spec file** to seed the prototype, so that I can bridge my existing workflow into UI Studio.

PM Persona

- As a PM, I want to **describe a feature concept in plain language** and get a clickable prototype, so that I can quickly validate ideas with stakeholders before investing design or engineering time.
- As a PM, I want to **share a running prototype via a URL** with stakeholders, so that they can experience the proposed feature firsthand.
- As a PM, I want the prototype to **include realistic mock data**, so that demos feel credible and stakeholders can evaluate the design in context.

Edge Cases

- As a user, I want to **see a clear error message** when the agent cannot interpret my input, so that I know how to rephrase or provide better context.
- As a user, I want to **undo the last agent change or roll back** to a previous version, so that I can recover from iterations that went in the wrong direction.
- As a user, I want to **start a new prototype from a previous one**, so that I can fork and explore variations without losing my original.

REQUIREMENTS

Must-Have (P0)

#	Requirement	Acceptance Criteria
P0-1	**Chat interface with multimodal input** — Users can type messages and attach images or files to the chat.	Given a user attaches a PNG wireframe and types “build this,” when the agent processes the input, then a prototype is generated reflecting the layout in the image.
P0-2	**Coding agent generates UI using the org’s design system** — Connected via MCP to the design system documentation. Generated code uses real component names, props, and design tokens.	Given the design system MCP is configured, when the agent generates a prototype, then 100% of UI elements map to documented design system components or tokens.
P0-3	**Iterative refinement via chat** — Users provide follow-up feedback in natural language. The agent modifies the existing prototype without starting from scratch.	Given a prototype is rendered, when the user says “make the header sticky,” then the agent updates the prototype to add a sticky header without losing other elements.
P0-4	**Multi-page prototype support** — Prototypes can contain multiple pages/views with working navigation.	Given a user requests “add a settings page,” when the agent generates it, then the new page is accessible via navigation and back-navigation works.
P0-5	**Mock data layer** — The agent generates realistic mock data appropriate to the prototype’s domain.	Given a prototype includes a data table, when rendered, then the table is populated with realistic, varied mock data.
P0-6	**Live preview** — The generated prototype renders in a preview panel alongside the chat, updating in near-real-time.	Given the agent finishes a code change, when the preview refreshes, then the updated prototype is visible within 5 seconds.
P0-7	**Local Electron app packaging** — UI Studio runs as a local desktop application via Electron.	Given a user installs the app, when they launch it, then the chat and preview are available locally without internet (except for LLM API calls).
P0-8	**Built on Anthropic Agent SDK (TypeScript)** — The coding agent uses the Anthropic Agent SDK.	Agent orchestration, tool use, and conversation management use the SDK’s primitives.
P0-9	**Configurable MCP connections** — Supports plugging in different MCP servers via configuration.	Given an admin edits the MCP config file, when the app restarts, then the agent has access to the newly configured MCP server.

Nice-to-Have (P1)

#	Requirement	Acceptance Criteria
P1-1	**Configurable modes with custom prompts** — Different modes (wireframe, high-fidelity, mobile) each with a configurable system prompt.	Given a user selects “mobile mode,” when they generate a prototype, then the output defaults to a mobile viewport.
P1-2	**Version history and rollback** — Users can see a timeline of agent changes and revert to any previous state.	Given 5 iterations have been made, when the user clicks “revert to version 3,” then the prototype rolls back to that state.

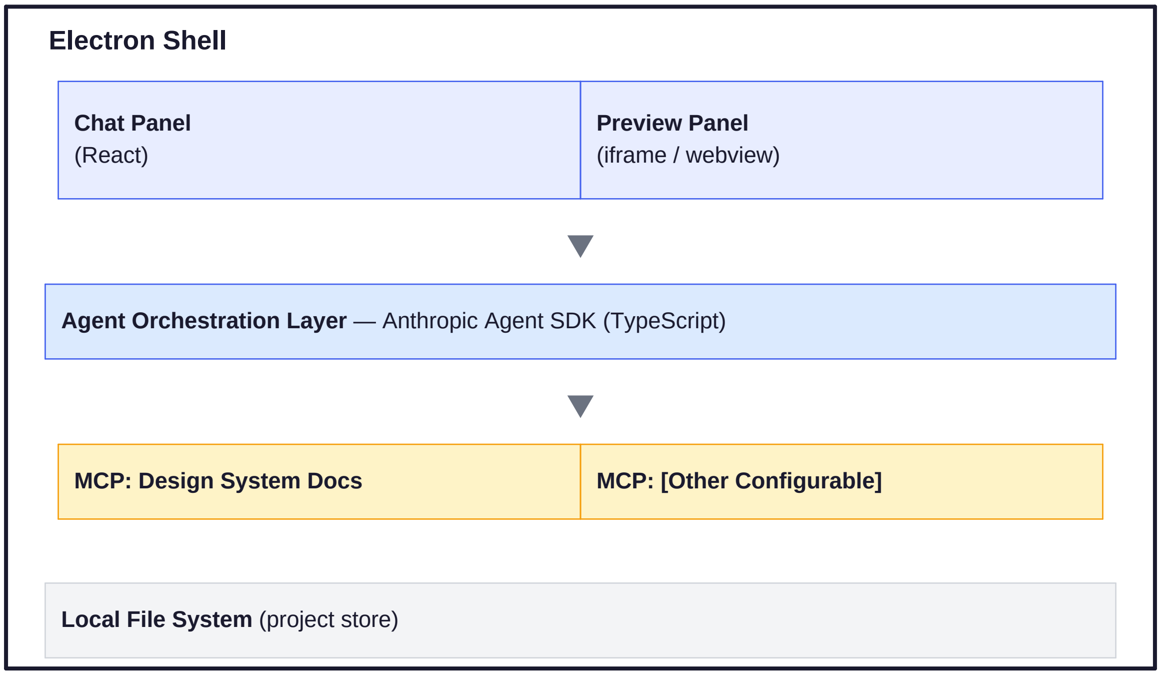
#	Requirement	Acceptance Criteria
P1-3	**Export prototype as static bundle** — Export as a standalone HTML/JS bundle for sharing.	Given a user clicks “Export,” then a self-contained folder is created that runs in any browser.
P1-4	**Shareable preview URL (local network)** — Generate a local URL for others on the same network.	Given the user clicks “Share,” then a URL is generated and accessible from another machine on the same LAN.
P1-5	**Prototype forking** — Duplicate an existing prototype as a starting point for a new exploration.	Given a user clicks “Fork,” then a new project is created with the same code and chat history.

Future Considerations (P2)

#	Requirement	Notes
P2-1	**Cloud-hosted preview sharing** — Share prototypes via public URL.	Requires hosting infrastructure; defer until adoption validates demand.
P2-2	**Figma plugin integration** — Import directly from a Figma file or frame.	Depends on Figma API access. Design the MCP interface to be extensible for this.
P2-3	**Multi-user collaborative editing** — Multiple users iterate simultaneously.	Significant complexity. Validate single-player adoption first.
P2-4	**Component-level code export** — Export individual components for production use.	Requires alignment with engineering on code standards.
P2-5	**Usage analytics and prototype library** — Track and search past prototypes.	Useful for measuring ROI; not needed for v1 adoption.

TECHNICAL ARCHITECTURE

The diagram below illustrates UI Studio’s high-level component architecture:


Electron Shell
Chat Panel (React)
Preview Panel (iframe / webview)
Agent Orchestration Layer — Anthropic Agent SDK (TypeScript)
MCP: Design System Docs
MCP: [Other Configurable]
Local File System (project store)

Key Technical Decisions

- **Anthropic Agent SDK (TypeScript)** for agent orchestration — handles conversation state, tool execution, and streaming.
- **MCP (Model Context Protocol)** for pluggable context sources — the design system docs MCP is the primary integration, but the architecture supports adding MCPs for component APIs, brand guidelines, etc.
- **Configurable modes** are implemented as named prompt templates stored in a config file. Each mode specifies a system prompt, default MCP set, and rendering defaults (viewport, theme).
- **Electron** for local packaging — the preview panel is an iframe/webview rendering the generated code. The app writes generated files to a local project directory.
- **Mock data generation** — the agent generates a mockData.ts file per prototype that exports typed mock data, imported by the generated pages.

SUCCESS METRICS

Leading Indicators (1–2 weeks post-launch)

Metric	Target	Stretch	Measurement
Adoption rate — % of designers + PMs who create at least one prototype	50% within 2 weeks	70% within 2 weeks	App analytics
Time to first prototype — Median time from first message to usable preview	< 10 minutes	< 5 minutes	Instrumented in-app
Iteration depth — Avg refinement messages per session	≥ 5	≥ 10	Chat message count
Design system hit rate — % of UI elements mapping to design system	≥ 90%	≥ 97%	Agent audit + spot checks

Lagging Indicators (30–60 days)

Metric	Target	Stretch	Measurement
Weekly active users	60% of team	80%	App analytics
Engineering prototype requests eliminated	50% reduction	75% reduction	Survey + ticket tracking
Stakeholder feedback quality	≥ 4/5 score	≥ 4.5/5	Post-demo survey
Design-to-dev handoff time	30% reduction	50% reduction	Sprint tracking data

---

OPEN QUESTIONS

#	Question	Owner	Blocking?
1	What format is the design system documentation currently in? Does an MCP server already exist, or does one need to be built?	**Engineering**	**Yes**
2	Which framework should the generated prototypes use (React, Vue, plain HTML/CSS)? Should it match the org's production stack?	**Eng + Design**	**Yes**
3	What is the LLM API key management strategy for local installs? Per-user keys, org-level key, or proxy?	**Eng + Security**	**Yes**
4	Are there licensing or compliance concerns with running an AI coding agent on design system IP locally?	**Legal**	No
5	Should the mock data layer support user-defined schemas, or is agent-generated mock data sufficient for v1?	**Design + PM**	No
6	What is the desired behavior when the design system MCP is unavailable?	**Eng + Design**	No

TIMELINE CONSIDERATIONS

Hard constraints: None identified — this is an internally motivated initiative without external deadlines.

Suggested Phasing

Phase	Scope	Duration
Phase 1: Core loop	Chat interface, single-page prototype generation, design system MCP integration, live preview, Electron shell	4–6 weeks
Phase 2: Multi-page + polish	Multi-page support with navigation, mock data layer, configurable modes, version history	3–4 weeks
Phase 3: Sharing + adoption	Export, local network sharing, onboarding flow, internal launch	2–3 weeks

Dependencies

- Design system MCP server must be built or adapted (Phase 1 blocker)
- Anthropic Agent SDK access and API key provisioning (Phase 1 blocker)
- Internal design team availability for beta testing (Phase 2)

APPENDIX: MODE CONFIGURATION EXAMPLE

Modes are defined in a YAML configuration file. Each mode specifies a system prompt, MCP dependencies, and rendering defaults:

```yaml
modes:
  high-fidelity:
    name: "High-Fidelity Prototype"
    system_prompt: |
      You are a senior frontend engineer building a
      high-fidelity interactive prototype. Use the
      organization's design system components exclusively.
    mcps:
      - design-system-docs
      - component-api
    defaults:
      viewport: "1440x900"
      theme: "light"


  wireframe:
    name: "Quick Wireframe"
    system_prompt: |
      You are building a low-fidelity wireframe prototype.
      Use simple grayscale layouts with placeholder components.
    mcps:
      - design-system-docs
    defaults:
      viewport: "1440x900"
      theme: "wireframe"


  mobile:
    name: "Mobile Prototype"
    system_prompt: |
      You are building a mobile-first prototype. Use the
      organization's mobile design system components.
    mcps:
      - design-system-docs
      - mobile-patterns
    defaults:
      viewport: "390x844"
      theme: "light"
```