**FEATURE:**

Build a comprehensive prompt database platform integrated with Whop authentication for a community of ~2000 users. The platform will store and manage both text prompts and image prompts with rich metadata, allowing users to search, save, vote, and eventually test prompts across different LLMs.

Core functionality:

* **Database for Prompts**: Store text and image prompts with automatic metadata extraction
* **User System**: Whop authentication with saved prompts and voting
* **Search & Discovery**: Filter by category, tags, popularity with simple text search
* **Admin Tools**: Bulk CSV import for initial ~1000 prompts from GitHub repos
* **Future-Ready**: Architecture supporting OpenRouter integration, agent rooms, and user submissions

The UI should have 5 tabs:

1. Text Prompts (implemented)
2. Image Prompts (implemented)
3. Video Prompts (coming soon)
4. Prompt Testing with OpenRouter (coming soon)
5. Agent Rooms (coming soon)

**EXAMPLES:**

In the app/ folder, there's the existing v0 interface showing:

* app/ - Next.js app structure (empty but configured)
* Current UI mockups show prompt cards with metadata tags, voting, and save functionality
* The search interface with category filters and sort options
* Existing Next.js setup with TypeScript and Tailwind CSS

Key patterns to follow:

* Use Next.js 14 App Router structure
* Shadcn/ui components for consistency
* Supabase for database and auth integration
* TypeScript for type safety throughout

**DOCUMENTATION:**

**Supabase Documentation:**

* https://supabase.com/docs/guides/auth/social-login/auth-whop - Whop OAuth integration
* https://supabase.com/docs/reference/javascript/introduction - Supabase JS Client
* https://supabase.com/docs/guides/database/postgres-policies - Row Level Security setup

**Next.js 14 Documentation:**

* https://nextjs.org/docs/app - App Router documentation
* https://nextjs.org/docs/app/building-your-application/data-fetching - Data fetching patterns

**Whop Documentation:**

* https://dev.whop.com/reference/oauth - OAuth2 flow for Whop
* https://dev.whop.com/docs/whop-api - API reference

**Shadcn/ui:**

* https://ui.shadcn.com/docs/installation/next - Next.js installation
* https://ui.shadcn.com/docs/components - Component library

**CSV Processing:**

* Papa Parse for CSV parsing in browser/Node.js
* Example CSV format for bulk import provided

**OTHER CONSIDERATIONS:**

**Critical Requirements:**

* Start simple - MVP first, then iterate
* Users can only save/vote on prompts initially (no creation)
* Admin-only prompt upload via CSV or UI
* Metadata should be auto-analyzed using AI (OpenAI API)
* Each prompt needs view tracking for popularity sorting
* Mobile-responsive design is essential
* Fast search with debouncing
* Infinite scroll or pagination for large lists

**Database Considerations:**

* Prompts will scale from 100 → 1000+ quickly
* User base: ~200 initially → 2000 max
* Need efficient indexes for search and filtering
* Vote aggregation should be performant
* View counts need rate limiting per user/session

**Security & Performance:**

* All user data stays in Whop ecosystem
* Row Level Security (RLS) on all Supabase tables
* API routes should be rate-limited
* Image URLs should be CDN-ready
* Implement proper caching strategies

**CSV Import Format Example:**

title,prompt\_content,category,tags,platform,style

"UI/UX Expert","You are an experienced UI/UX Designer...","design","ui,ux,design,evaluation",,

"Red Team Consultant","You are an ethical hacker...","coding","security,testing,pentesting",,

"Cyberpunk Portrait","Create a cyberpunk style portrait...","art","cyberpunk,portrait,neon","midjourney","cyberpunk"

**Future Features to Keep in Mind (but don't implement):**

* OpenRouter integration for testing prompts
* Credit system for API usage
* User prompt submissions with moderation
* Agent personas from saved prompts
* Real-time agent rooms with orchestration

**Common Gotchas:**

* Supabase requires email confirmation by default (disable for Whop users)
* Next.js App Router has different patterns than Pages Router
* Whop OAuth tokens need refresh handling
* Don't over-engineer - keep it simple initially
* Vote counts should be cached, not computed on every request
* Search indexing needs to handle both title and content