name: "Prompt Platform: Community-Driven Prompt Database with Whop Integration" description: |

**Purpose**

Build a production-ready prompt database platform for a Whop community that enables users to discover, save, and vote on curated prompts. The platform supports both text and image prompts with rich metadata, search capabilities, and future extensibility for prompt testing and agent rooms.

**Core Principles**

1. **Start Simple**: MVP first, avoid over-engineering
2. **User-Centric**: Fast search, easy navigation, mobile-responsive
3. **Scalable Architecture**: Support 100→1000+ prompts, 200→2000 users
4. **Type-Safe**: TypeScript throughout with Zod validation
5. **Performance First**: Efficient queries, proper caching, optimistic updates

**Goal**

Create a fully functional prompt database where users can search, save, and vote on prompts. Admins can bulk import prompts via CSV with automatic metadata extraction. The platform integrates with Whop for authentication and provides a foundation for future features like OpenRouter testing and agent rooms.

**Why**

* **Business Value**: Monetizable prompt library for Whop community
* **User Need**: Centralized, searchable repository of proven prompts
* **Competitive Edge**: Metadata-rich, community-validated prompt database
* **Future Revenue**: Foundation for credit-based prompt testing

**What**

A Next.js 14 application with:

* Whop OAuth authentication
* Supabase backend with RLS
* Text and image prompt browsing
* Search with filters (category, tags, platform)
* User saves and voting system
* Admin CSV import with AI metadata extraction
* View tracking and popularity sorting

**Success Criteria**

* [ ] Users can authenticate via Whop OAuth
* [ ] Browse and search text/image prompts
* [ ] Save prompts to personal collection
* [ ] Vote (upvote/downvote) on prompts
* [ ] Admin can bulk import via CSV
* [ ] Metadata auto-extracted for prompts
* [ ] Mobile-responsive design
* [ ] Search completes in <500ms
* [ ] Page loads in <2s

**All Needed Context**

**Documentation & References**

# Core Documentation

- url: https://supabase.com/docs/guides/auth/social-login/auth-whop

why: Whop OAuth integration with Supabase Auth

- url: https://nextjs.org/docs/app/building-your-application/routing

why: App Router patterns for our 5-tab structure

- url: https://ui.shadcn.com/docs/components/card

why: Card components for prompt display

- url: https://dev.whop.com/reference/oauth

why: Whop OAuth2 flow implementation details

- url: https://supabase.com/docs/guides/database/postgres-policies

why: RLS policies for user data protection

- url: https://tanstack.com/query/latest/docs/react/overview

why: Data fetching and caching patterns

- file: app/globals.css

why: Existing Tailwind configuration with CSS variables

- file: components.json

why: Shadcn/ui configuration already set up

**Current Codebase Structure**

.

├── app/

│ ├── globals.css # Tailwind styles configured

│ └── (empty structure) # Ready for implementation

├── components/ # Empty, ready for components

├── lib/ # Empty, for utilities

├── public/ # Static assets

├── .env.local.example # Environment template

├── components.json # Shadcn configured

├── next.config.mjs # Next.js config

├── tailwind.config.ts # Tailwind configured

└── tsconfig.json # TypeScript configured

**Target Codebase Structure**

.

├── app/

│ ├── layout.tsx # Root layout with providers

│ ├── page.tsx # Redirect to /prompts

│ ├── globals.css # (existing)

│ ├── api/

│ │ ├── auth/

│ │ │ └── callback/route.ts # Whop OAuth callback

│ │ ├── prompts/

│ │ │ ├── route.ts # GET/POST prompts

│ │ │ └── [id]/

│ │ │ └── route.ts # Single prompt ops

│ │ ├── votes/route.ts # Voting endpoint

│ │ └── admin/

│ │ └── import/route.ts # CSV import (admin only)

│ ├── (auth)/

│ │ ├── login/page.tsx # Login page

│ │ └── logout/page.tsx # Logout handler

│ └── (dashboard)/

│ ├── layout.tsx # Dashboard with tab nav

│ ├── prompts/page.tsx # Text prompts tab

│ ├── images/page.tsx # Image prompts tab

│ ├── videos/page.tsx # Coming soon

│ ├── test/page.tsx # Coming soon

│ ├── rooms/page.tsx # Coming soon

│ └── saved/page.tsx # User's saved prompts

├── components/

│ ├── ui/ # Shadcn components

│ ├── layout/

│ │ ├── navbar.tsx # Top navigation

│ │ ├── tab-navigation.tsx # 5-tab navigation

│ │ └── user-menu.tsx # User dropdown

│ ├── prompts/

│ │ ├── prompt-card.tsx # Prompt display card

│ │ ├── prompt-grid.tsx # Grid layout

│ │ ├── prompt-search.tsx # Search bar

│ │ ├── prompt-filters.tsx # Category/tag filters

│ │ └── prompt-detail-modal.tsx # Prompt details

│ ├── common/

│ │ ├── save-button.tsx # Save to collection

│ │ ├── vote-buttons.tsx # Upvote/downvote

│ │ └── copy-button.tsx # Copy prompt

│ └── admin/

│ └── csv-import.tsx # Admin import UI

├── lib/

│ ├── supabase/

│ │ ├── client.ts # Browser client

│ │ ├── server.ts # Server client

│ │ ├── middleware.ts # Auth middleware

│ │ └── types.ts # Database types

│ ├── utils/

│ │ ├── metadata-analyzer.ts # AI metadata extraction

│ │ ├── csv-parser.ts # CSV import logic

│ │ └── cn.ts # Class name helper

│ └── hooks/

│ ├── use-prompts.ts # Prompt data hooks

│ ├── use-auth.ts # Auth state hook

│ └── use-votes.ts # Voting logic

├── types/

│ ├── database.ts # Supabase generated types

│ ├── prompt.ts # Prompt models

│ └── user.ts # User models

├── scripts/

│ ├── setup-database.sql # Database schema

│ └── import-prompts.ts # Bulk import script

└── middleware.ts # Auth protection

**Known Gotchas & Critical Details**

// CRITICAL: Whop OAuth requires specific redirect URI format

// CRITICAL: Supabase RLS must be enabled on all tables

// CRITICAL: Use server components for initial data fetch

// CRITICAL: Client components only for interactivity

// CRITICAL: Vote counts should be materialized, not computed

// CRITICAL: Search should use Postgres full-text search

// CRITICAL: Metadata extraction needs OpenAI API key

// CRITICAL: CSV import should batch inserts (max 100 at a time)

// CRITICAL: Image prompt URLs should be validated

// CRITICAL: Use optimistic updates for voting UX

**Implementation Blueprint**

**Database Schema**

-- Enable necessary extensions

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Users table (synced with Whop)

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

whop\_user\_id VARCHAR(255) UNIQUE NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

username VARCHAR(100) UNIQUE,

display\_name VARCHAR(255),

avatar\_url TEXT,

role VARCHAR(50) DEFAULT 'user',

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Text prompts table

CREATE TABLE text\_prompts (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

title VARCHAR(255) NOT NULL,

description TEXT,

prompt\_content TEXT NOT NULL,

category VARCHAR(100),

tags TEXT[],

metadata JSONB DEFAULT '{}',

author\_id UUID REFERENCES users(id),

is\_published BOOLEAN DEFAULT true,

view\_count INTEGER DEFAULT 0,

vote\_score INTEGER DEFAULT 0,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Image prompts table

CREATE TABLE image\_prompts (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

title VARCHAR(255) NOT NULL,

prompt\_content TEXT NOT NULL,

platform VARCHAR(100),

style VARCHAR(100),

model\_version VARCHAR(50),

example\_image\_url TEXT,

metadata JSONB DEFAULT '{}',

author\_id UUID REFERENCES users(id),

is\_published BOOLEAN DEFAULT true,

view\_count INTEGER DEFAULT 0,

vote\_score INTEGER DEFAULT 0,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Saved prompts

CREATE TABLE saved\_prompts (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES users(id) ON DELETE CASCADE,

prompt\_id UUID NOT NULL,

prompt\_type VARCHAR(20) NOT NULL,

saved\_at TIMESTAMP DEFAULT NOW(),

UNIQUE(user\_id, prompt\_id, prompt\_type)

);

-- Votes table

CREATE TABLE prompt\_votes (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES users(id) ON DELETE CASCADE,

prompt\_id UUID NOT NULL,

prompt\_type VARCHAR(20) NOT NULL,

vote\_type INTEGER CHECK (vote\_type IN (-1, 1)),

created\_at TIMESTAMP DEFAULT NOW(),

UNIQUE(user\_id, prompt\_id, prompt\_type)

);

-- Create indexes

CREATE INDEX idx\_text\_prompts\_category ON text\_prompts(category);

CREATE INDEX idx\_text\_prompts\_tags ON text\_prompts USING GIN(tags);

CREATE INDEX idx\_text\_prompts\_search ON text\_prompts USING GIN(to\_tsvector('english', title || ' ' || prompt\_content));

CREATE INDEX idx\_image\_prompts\_platform ON image\_prompts(platform);

CREATE INDEX idx\_saved\_prompts\_user ON saved\_prompts(user\_id);

CREATE INDEX idx\_votes\_prompt ON prompt\_votes(prompt\_id, prompt\_type);

-- Row Level Security

ALTER TABLE users ENABLE ROW LEVEL SECURITY;

ALTER TABLE text\_prompts ENABLE ROW LEVEL SECURITY;

ALTER TABLE image\_prompts ENABLE ROW LEVEL SECURITY;

ALTER TABLE saved\_prompts ENABLE ROW LEVEL SECURITY;

ALTER TABLE prompt\_votes ENABLE ROW LEVEL SECURITY;

-- RLS Policies

CREATE POLICY "Users can read all users" ON users FOR SELECT USING (true);

CREATE POLICY "Users can update own profile" ON users FOR UPDATE USING (auth.uid() = id);

CREATE POLICY "Anyone can read published prompts" ON text\_prompts FOR SELECT USING (is\_published = true);

CREATE POLICY "Admins can insert prompts" ON text\_prompts FOR INSERT WITH CHECK (EXISTS (SELECT 1 FROM users WHERE id = auth.uid() AND role = 'admin'));

CREATE POLICY "Users can save prompts" ON saved\_prompts FOR ALL USING (auth.uid() = user\_id);

CREATE POLICY "Users can vote" ON prompt\_votes FOR ALL USING (auth.uid() = user\_id);

**Implementation Tasks**

Task 1: Setup Supabase and Authentication

CREATE lib/supabase/client.ts:

- PATTERN: Singleton pattern for browser client

- Include auth helpers

- Type-safe with generated types

CREATE lib/supabase/server.ts:

- PATTERN: Server client with cookies

- Use cookies() from next/headers

- Handle auth refresh

CREATE app/api/auth/callback/route.ts:

- PATTERN: Whop OAuth callback handler

- Exchange code for session

- Create/update user record

- Redirect to dashboard

Task 2: Create Component Library

CREATE components/ui/:

- Run: npx shadcn-ui@latest init

- Add: card, button, input, select, dialog, tabs

- Add: dropdown-menu, badge, skeleton, toast

CREATE components/layout/navbar.tsx:

- Logo and brand

- User menu with avatar

- Sign in/out buttons

- Mobile responsive

CREATE components/layout/tab-navigation.tsx:

- 5 tabs with icons

- Active state styling

- Coming soon badges

- Use next/link for routing

Task 3: Implement Prompt Display Components

CREATE components/prompts/prompt-card.tsx:

- PATTERN: Card with hover effects

- Display title, description, tags

- Vote buttons with optimistic updates

- Save button with loading state

- Copy to clipboard functionality

CREATE components/prompts/prompt-grid.tsx:

- PATTERN: Responsive grid layout

- Skeleton loading states

- Infinite scroll or pagination

- Empty state message

Task 4: Build Search and Filtering

CREATE components/prompts/prompt-search.tsx:

- PATTERN: Debounced search input

- Search icon and clear button

- Loading indicator

- Keyboard shortcuts (/)

CREATE components/prompts/prompt-filters.tsx:

- Category dropdown

- Tag multi-select

- Sort options (popular, recent, top)

- Mobile-friendly filters

Task 5: Create Data Fetching Hooks

CREATE lib/hooks/use-prompts.ts:

- PATTERN: React Query for caching

- Pagination support

- Search and filter params

- Optimistic updates for votes

CREATE lib/hooks/use-auth.ts:

- Current user state

- Whop user data

- Role checking (isAdmin)

- Session management

Task 6: Implement API Routes

CREATE app/api/prompts/route.ts:

- GET: List with filters

- POST: Create (admin only)

- Include pagination

- Full-text search

CREATE app/api/votes/route.ts:

- POST: Cast vote

- Handle vote changes

- Update vote\_score

- Return new score

Task 7: Build Admin CSV Import

CREATE components/admin/csv-import.tsx:

- File upload with drag-drop

- CSV preview table

- Progress indicator

- Error handling

CREATE lib/utils/csv-parser.ts:

- PATTERN: Papa Parse for parsing

- Validate required fields

- Batch insert logic

- Error reporting

CREATE lib/utils/metadata-analyzer.ts:

- OpenAI integration

- Extract categories and tags

- Analyze prompt purpose

- Cache results

Task 8: Create Main Pages

CREATE app/(dashboard)/prompts/page.tsx:

- Server component for initial data

- Client search/filter components

- Prompt grid with loading

- SEO metadata

CREATE app/(dashboard)/images/page.tsx:

- Similar to prompts page

- Platform/style filters

- Image preview in cards

Task 9: Add User Features

CREATE app/(dashboard)/saved/page.tsx:

- User's saved prompts

- Remove from saved

- Export functionality

CREATE middleware.ts:

- Protect dashboard routes

- Redirect to login

- Handle auth refresh

Task 10: Testing and Optimization

- Add loading.tsx files

- Implement error boundaries

- Add meta tags for SEO

- Test mobile responsiveness

- Optimize images

- Add analytics

**API Route Examples**

// app/api/prompts/route.ts

export async function GET(request: Request) {

const { searchParams } = new URL(request.url);

const page = parseInt(searchParams.get('page') || '1');

const limit = parseInt(searchParams.get('limit') || '20');

const search = searchParams.get('search') || '';

const category = searchParams.get('category') || '';

// Build query with filters

let query = supabase

.from('text\_prompts')

.select('\*', { count: 'exact' })

.eq('is\_published', true);

if (search) {

query = query.textSearch('title', search);

}

if (category) {

query = query.eq('category', category);

}

// Pagination

const start = (page - 1) \* limit;

query = query.range(start, start + limit - 1);

const { data, error, count } = await query;

return NextResponse.json({

prompts: data,

total: count,

page,

totalPages: Math.ceil(count / limit)

});

}

**Validation & Testing**

**Level 1: Build & Type Checking**

# Install dependencies

npm install

# Type checking

npm run build

# Expected: No TypeScript errors

**Level 2: Component Testing**

# Test components render

npm run dev

# Manual tests:

# - Login flow works with Whop

# - Prompts display correctly

# - Search returns results

# - Voting updates UI

# - Saves persist

**Level 3: Database Testing**

-- Test RLS policies

-- As regular user: should only see published

-- As admin: should be able to insert

-- Test vote uniqueness constraint

-- Test search performance with 1000+ records

**Level 4: Performance Testing**

# Lighthouse scores

# - Performance: >90

# - Accessibility: >95

# - Best Practices: >95

# - SEO: >90

# Load testing

# - Search response <500ms

# - Page load <2s

# - Smooth scrolling at 60fps

**Final Checklist**

* [ ] Whop OAuth working
* [ ] All 5 tabs present (2 active, 3 coming soon)
* [ ] Search with debouncing
* [ ] Category and tag filtering
* [ ] Voting system functional
* [ ] Save functionality working
* [ ] Admin CSV import tested
* [ ] Mobile responsive design
* [ ] Loading states everywhere
* [ ] Error boundaries in place
* [ ] RLS policies enforced
* [ ] Environment variables documented

**Anti-Patterns to Avoid**

* ❌ Don't fetch all prompts at once - use pagination
* ❌ Don't compute vote counts on read - materialize them
* ❌ Don't allow unauthenticated access to user data
* ❌ Don't skip loading states - users need feedback
* ❌ Don't hardcode categories - make them dynamic
* ❌ Don't over-engineer - ship MVP first

**Performance Optimizations**

* Use React.memo for prompt cards
* Implement virtual scrolling for large lists
* Cache searches with React Query
* Use optimistic UI updates
* Lazy load images
* Debounce search input (300ms)
* Use Suspense boundaries

**Security Considerations**

* Enable RLS on all tables
* Validate all inputs with Zod
* Sanitize CSV imports
* Rate limit API routes
* Use CSRF protection
* Validate image URLs
* Escape user content

**Confidence Score: 9/10**

High confidence because:

* Clear requirements and examples
* Well-documented APIs (Supabase, Whop, Next.js)
* Established patterns with shadcn/ui
* Comprehensive database schema
* Detailed implementation steps

Minor uncertainty on exact Whop OAuth implementation, but documentation is available.