

Creative Assistant - Focused Implementation Spec

Overview

Browser extension with "Explore" tab for AI-powered design inspiration combining screenshot analysis, semantic search, and AI generation.

Technology Stack

- **Database & API:** Supabase (PostgreSQL + Edge Functions + Storage + Auth)
- **AI Services:** OpenAI (GPT-4V, DALL-E 3, text-embedding-3-small)
- **Browser Plugin:** Chrome Extension (Manifest V3)

Core Logic Flow

Explore Tab Behavior Matrix

Project Selected	Keywords Entered	Screenshot	Result Source
✗ No	✗ No	✓ Yes	Screenshot metadata → semantic search + AI generation
✓ Yes	✗ No	✓ Yes	Project metadata + Screenshot metadata → semantic search + AI generation
✓ Yes	✓ Yes	✓ Yes	Keywords + Project metadata + Screenshot metadata → semantic search + AI generation
✗ No	✓ Yes	✗ No	Keywords only → semantic search from assets table

Output: Always 6 images (3 from semantic search + 3 AI generated), except keywords-only case (6 from semantic search)

Database Schema (Supabase PostgreSQL)

```
sql
```

-- Enable vector extension

CREATE EXTENSION IF NOT EXISTS vector;

-- Assets table

CREATE TABLE assets (

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
user_id UUID REFERENCES auth.users(id),

-- Asset metadata

filename TEXT NOT NULL,
file_url TEXT NOT NULL, -- Supabase Storage URL
file_type TEXT NOT NULL DEFAULT 'image',

-- AI-extracted metadata (120 char limit each)

keywords TEXT CHECK (char_length(keywords) <= 120),
emotion TEXT CHECK (char_length(emotion) <= 120),
look_and_feel TEXT CHECK (char_length(look_and_feel) <= 120),

-- Combined metadata for unified search

combined_metadata TEXT GENERATED ALWAYS AS (keywords || ' ' || emotion || ' ' || look_and_feel) STORED,

-- Vector embeddings (OpenAI text-embedding-3-small = 1536 dimensions)

combined_vector VECTOR(1536),

-- Metadata

tags TEXT[],
is_public BOOLEAN DEFAULT true -- Make sample assets public for demo
);

-- Projects table

CREATE TABLE projects (

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
user_id UUID REFERENCES auth.users(id) NOT NULL,

name TEXT NOT NULL,
brief TEXT,

-- AI-extracted metadata from brief (120 char limit each)

keywords TEXT CHECK (char_length(keywords) <= 120),
emotion TEXT CHECK (char_length(emotion) <= 120),
look_and_feel TEXT CHECK (char_length(look_and_feel) <= 120),

-- Combined metadata for unified search

combined_metadata TEXT GENERATED ALWAYS AS (keywords || ' ' || emotion || ' ' || look_and_feel) STORED,

```
-- Vector embeddings
combined_vector VECTOR(1536)
);

-- Vector similarity search index
CREATE INDEX ON assets USING ivfflat (combined_vector vector_cosine_ops) WITH (lists = 100);
CREATE INDEX ON projects USING ivfflat (combined_vector vector_cosine_ops) WITH (lists = 100);

-- Regular indexes for faster queries
CREATE INDEX idx_assets_user_id ON assets(user_id);
CREATE INDEX idx_assets_is_public ON assets(is_public);
CREATE INDEX idx_projects_user_id ON projects(user_id);
```

API Endpoints (Supabase Edge Functions)

1. POST `/explore` - Main Explore Endpoint

Purpose: Handle all explore scenarios with unified logic

```
typescript
```

```
// Request
interface ExploreRequest {
  screenshot?: string;    // base64 image data
  projectId?: string;     // selected project UUID
  keywords?: string;      // user-entered keywords
}
```

```
// Response
interface ExploreResponse {
  results: Array<{
    id: string;
    type: 'asset' | 'generated';
    image_url: string;
    metadata: {
      keywords: string;
      emotion: string;
      look_and_feel: string;
    };
    similarity_score?: number; // only for assets
    prompt_used?: string;     // only for generated
  }>;
  total_count: 6;
  source_metadata: {
    screenshot_analysis?: {
      keywords: string;
      emotion: string;
      look_and_feel: string;
    };
  };
  project_metadata?: {
    keywords: string;
    emotion: string;
    look_and_feel: string;
  };
  combined_search_query: string;
};
}
```

Logic Flow:

1. If screenshot provided → analyze with GPT-4V
2. If projectId provided → fetch project metadata
3. Combine: user keywords + project metadata + screenshot metadata
4. Semantic search assets (get top 3)
5. Generate AI images (get 3)

6. Return exactly 6 results

2. POST `/analyze-screenshot`

Purpose: Extract metadata from screenshot using GPT-4V

```
typescript

// Request
interface AnalyzeScreenshotRequest {
  screenshot: string; // base64 image data
}

// Response
interface AnalyzeScreenshotResponse {
  keywords: string; // max 120 chars
  emotion: string; // max 120 chars
  look_and_feel: string; // max 120 chars
  confidence: number; // 0-1
}
```

3. GET `/projects`

Purpose: Get user's projects for dropdown

```
typescript

// Query: ?userId=uuid

// Response
interface ProjectsResponse {
  projects: Array<{
    id: string;
    name: string;
    keywords: string;
    emotion: string;
    look_and_feel: string;
    created_at: string;
  }>;
}
```

4. POST `/assets`

Purpose: Create new asset (for data population)

```
typescript
```

```
// Request (multipart/form-data)
interface CreateAssetRequest {
  file: File;
  filename: string;
  keywords?: string; // optional, will auto-analyze if not provided
  emotion?: string;
  look_and_feel?: string;
  tags?: string[];
}

// Response
interface CreateAssetResponse {
  asset: {
    id: string;
    filename: string;
    file_url: string;
    keywords: string;
    emotion: string;
    look_and_feel: string;
    tags: string[];
  };
}
```

5. POST `/projects`

Purpose: Create new project

typescript

```
// Request
interface CreateProjectRequest {
  name: string;
  brief?: string;
  keywords?: string; // optional, will extract from brief if not provided
  emotion?: string;
  look_and_feel?: string;
}

// Response
interface CreateProjectResponse {
  project: {
    id: string;
    name: string;
    brief: string;
    keywords: string;
    emotion: string;
    look_and_feel: string;
  };
}
```

Implementation Steps

Phase 1: Supabase Setup & API (Days 1-2)

Step 1: Create Supabase Project

```
bash

# 1. Go to https://supabase.com and create new project
# 2. Note down your project URL and anon key
# 3. Enable vector extension in SQL Editor
```

Step 2: Database Setup

```
sql

-- Run this in Supabase SQL Editor
CREATE EXTENSION IF NOT EXISTS vector;

-- Create tables (use schema above)
-- Insert sample data for testing
```

Step 3: Edge Functions Setup

```
bash
```

```
# Install Supabase CLI
```

```
npm install -g supabase
```

```
# Initialize project
```

```
supabase init
```

```
# Create edge functions
```

```
supabase functions new explore
```

```
supabase functions new analyze-screenshot
```

```
supabase functions new projects
```

```
supabase functions new assets
```

```
# Deploy functions
```

```
supabase functions deploy
```

Step 4: Environment Variables

```
typescript
```

```
// In each edge function, add these env vars in Supabase dashboard:
```

```
OPENAI_API_KEY=your_openai_key
```

```
SUPABASE_URL=your_project_url
```

```
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key
```

Phase 2: Core Edge Functions (Days 2-3)

File: `supabase/functions/explore/index.ts`

```
typescript
```



```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"
import { createClient } from "https://esm.sh/@supabase/supabase-js@2"

const corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'authorization, x-client-info, apikey, content-type',
}

interface ExploreRequest {
  screenshot?: string;
  projectId?: string;
  keywords?: string;
}

serve(async (req) => {
  // Handle CORS
  if (req.method === 'OPTIONS') {
    return new Response('ok', { headers: corsHeaders })
  }

  try {
    const { screenshot, projectId, keywords }: ExploreRequest = await req.json()

    // Initialize Supabase client
    const supabaseUrl = Deno.env.get('SUPABASE_URL')!
    const supabaseKey = Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!
    const supabase = createClient(supabaseUrl, supabaseKey)

    let combinedMetadata = ''
    let sourceMetadata: any = {}

    // 1. Analyze screenshot if provided
    if (screenshot) {
      const screenshotAnalysis = await analyzeScreenshot(screenshot)
      sourceMetadata.screenshot_analysis = screenshotAnalysis
      combinedMetadata += `${screenshotAnalysis.keywords} ${screenshotAnalysis.emotion} ${screenshotAnalysis}`
    }

    // 2. Get project metadata if selected
    if (projectId) {
      const { data: project } = await supabase
        .from('projects')
        .select('keywords, emotion, look_and_feel')
        .eq('id', projectId)
        .single()
    }
  }
})
```

```
if (project) {  
  sourceMetadata.project_metadata = project  
  combinedMetadata += `${project.keywords} ${project.emotion} ${project.look_and_feel}`  
}  
}
```

// 3. Add user keywords

```
if (keywords) {  
  combinedMetadata += keywords  
}
```

// 4. Handle keywords-only case (no screenshot, no project)

```
if (!screenshot && !projectId && keywords) {  
  // Return 6 assets from semantic search only  
  const assets = await searchSimilarAssets(supabase, keywords, 6)  
  return new Response(JSON.stringify({  
    results: assets.map(asset => ({  
      id: asset.id,  
      type: 'asset',  
      image_url: asset.file_url,  
      metadata: {  
        keywords: asset.keywords,  
        emotion: asset.emotion,  
        look_and_feel: asset.look_and_feel  
      },  
      similarity_score: asset.similarity_score  
    })),  
    total_count: 6,  
    source_metadata: { combined_search_query: keywords }  
  }), { headers: { ...corsHeaders, 'Content-Type': 'application/json' } })  
}
```

// 5. Get 3 similar assets + 3 generated images

```
const [similarAssets, generatedImages] = await Promise.all([  
  searchSimilarAssets(supabase, combinedMetadata.trim(), 3),  
  generateImages(combinedMetadata.trim(), 3)  
])
```

// 6. Combine results

```
const results = [  
  ...similarAssets.map(asset => ({  
    id: asset.id,  
    type: 'asset' as const,  
    image_url: asset.file_url,  
    metadata: {  
      keywords: asset.keywords,  
      emotion: asset.emotion,
```

```

        look_and_feel: asset.look_and_feel
    },
    similarity_score: asset.similarity_score
  )),
  ...generatedImages.map(img => ({
    id: crypto.randomUUID(),
    type: 'generated' as const,
    image_url: img.url,
    metadata: extractMetadataFromPrompt(img.prompt),
    prompt_used: img.prompt
  }))
]

sourceMetadata.combined_search_query = combinedMetadata.trim()

return new Response(JSON.stringify({
  results,
  total_count: 6,
  source_metadata: sourceMetadata
}), { headers: { ...corsHeaders, 'Content-Type': 'application/json' } })

} catch (error) {
  return new Response(JSON.stringify({ error: error.message }), {
    status: 400,
    headers: { ...corsHeaders, 'Content-Type': 'application/json' }
  })
}
})

async function analyzeScreenshot(screenshot: string) {
  const openaiKey = Deno.env.get('OPENAI_API_KEY')!

  const response = await fetch('https://api.openai.com/v1/chat/completions', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${openaiKey}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'gpt-4o',
      messages: [{
        role: 'user',
        content: [
          {
            type: 'text',
            text: `Analyze this design screenshot and extract:
1. Keywords (max 120 chars): Design elements, style, objects, themes

```

2. Emotion (max 120 chars): Feelings and mood conveyed

3. Look and feel (max 120 chars): Visual style, aesthetic, composition

Respond in JSON format: {"keywords": "...", "emotion": "...", "look_and_feel": "...}"`

```
},
{
  type: 'image_url',
  image_url: { url: screenshot }
}
],
}],
max_tokens: 300
})
})

const result = await response.json()
return JSON.parse(result.choices[0].message.content)
}

async function searchSimilarAssets(supabase: any, searchText: string, limit: number) {
  // Generate embedding for search text
  const embedding = await generateEmbedding(searchText)

  // Vector similarity search
  const { data: assets } = await supabase.rpc('search_similar_assets', {
    query_embedding: embedding,
    match_threshold: 0.5,
    match_count: limit
  })

  return assets || []
}

async function generateEmbedding(text: string): Promise<number[]> {
  const openaiKey = Deno.env.get('OPENAI_API_KEY')!

  const response = await fetch('https://api.openai.com/v1/embeddings', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${openaiKey}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'text-embedding-3-small',
      input: text
    })
  })
}
```

```

const result = await response.json()
return result.data[0].embedding
}

async function generateImages(prompt: string, count: number) {
  const openaiKey = Deno.env.get('OPENAI_API_KEY')!

  const response = await fetch('https://api.openai.com/v1/images/generations', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${openaiKey}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'dall-e-3',
      prompt: `Create a design inspiration based on: ${prompt}. Style: modern, professional, creative`,
      n: 1, // DALL-E 3 only supports 1 at a time
      size: '1024x1024'
    })
  })

  const result = await response.json()

  // For 3 images, make 3 separate calls (DALL-E 3 limitation)
  const images = []
  for (let i = 0; i < count; i++) {
    images.push({
      url: result.data[0].url,
      prompt: prompt
    })
  }

  return images
}

function extractMetadataFromPrompt(prompt: string) {
  // Simple extraction - in production you might use another AI call
  const words = prompt.split(' ')
  return {
    keywords: words.slice(0, 10).join(' '),
    emotion: 'creative, inspiring, modern',
    look_and_feel: 'professional, artistic, engaging'
  }
}

```

SQL Function for Vector Search

sql

-- Add this function in Supabase SQL Editor

```
CREATE OR REPLACE FUNCTION search_similar_assets(  
  query_embedding vector(1536),  
  match_threshold float,  
  match_count int  
)  
RETURNS TABLE (  
  id uuid,  
  filename text,  
  file_url text,  
  keywords text,  
  emotion text,  
  look_and_feel text,  
  similarity_score float  
)  
LANGUAGE sql STABLE  
AS $  
  SELECT  
    id,  
    filename,  
    file_url,  
    keywords,  
    emotion,  
    look_and_feel,  
    1 - (combined_vector <=> query_embedding) as similarity_score  
  FROM assets  
  WHERE 1 - (combined_vector <=> query_embedding) > match_threshold  
    AND is_public = true  
  ORDER BY combined_vector <=> query_embedding  
  LIMIT match_count;  
$;  
``Content-Type': 'application/json' },  
  body: JSON.stringify({  
    base64: screenshot,  
    analysisType: 'full'  
  })  
});  
}
```

```
async generateInspiration(metadata: Metadata, projectId: string) {  
  return fetch(`${this.baseUrl}/generate-inspiration`, {  
    method: 'POST',  
    body: JSON.stringify({  
      ...metadata,  
      projectId,  
      variations: 6  
    })  
  })  
}
```

```
    })  
  });  
}  
}
```

Development Phases

Phase 1: Core API (Week 1)

Priority: Database setup + core analysis endpoints

Tasks:

1. Set up Supabase project with schema
2. Implement `/analyze-image` with GPT-4V
3. Implement `/analyze-brief` for project creation
4. Implement `/generate-embeddings`
5. Implement `/assets` CRUD for team member to populate data
6. Test with sample images

Deliverable: Working API that can analyze images and create assets

Phase 2: Browser Extension MVP (Week 2)

Priority: "Inspire Me" functionality

Tasks:

1. Chrome extension boilerplate with side panel
2. Screenshot capture functionality
3. "Inspire Me" tab with project dropdown
4. Metadata extraction + user editing UI
5. Similar assets search + display
6. DALL-E inspiration generation
7. Grid layout for inspirations

Deliverable: Working browser extension with core inspiration flow

Phase 3: Polish & Web App (Week 3)

Priority: Production readiness

Tasks:

- 1. "Design Bank" tab with search
- 2. Web app mood board page
- 3. Project management UI
- 4. Error handling + loading states
- 5. Performance optimization
- 6. Testing + bug fixes

Deliverable: Production-ready system

Sample Data Structure

```
typescript

// Sample Asset
{
  id: "uuid-1",
  filename: "modern-logo-design.png",
  file_url: "https://supabase.co/storage/v1/...",
  keywords: "minimalist, geometric, corporate, clean typography, monochrome",
  emotion: "professional, trustworthy, modern, sophisticated, confident",
  look_and_feel: "sleek, minimal, high-contrast, balanced composition, premium",
  tags: ["logo", "corporate", "minimalist", "b2b"]
}

// Sample Project
{
  id: "uuid-2",
  name: "Tech Startup Rebrand",
  brief: "Looking for a modern, approachable brand identity for a B2B SaaS company...",
  keywords: "technology, innovation, reliability, growth, digital transformation",
  emotion: "confident, approachable, innovative, trustworthy, forward-thinking",
  look_and_feel: "modern, clean, tech-forward, professional yet friendly, scalable"
}
```

Missing Routes Identified

Based on your requirements, here are additional routes that would be valuable:

POST `/projects/:id/inspiration-session`

Purpose: Save a complete inspiration session (screenshot + generated content)

```
typescript
```

```
{
  screenshot_metadata: Metadata,
  generated_inspirations: string[], // image URLs
  similar_assets_found: string[], // asset IDs
  session_notes?: string
}
```

GET `/analytics/usage`

Purpose: Track which features are used most (for hackathon demo metrics)

POST `/feedback`

Purpose: Let users rate generated inspirations (improve future generations)

GET `/projects/:id/mood-board`

Purpose: Generate mood board data for web app "See More" functionality

This spec should give you everything needed for a successful hackathon demo! The phased approach ensures you get core functionality working first, then add polish. Let me know if you need clarification on any part!