ETHOS

ETHOS

A workplace that's yours.

GOAL: Create a Miro integration that uses AI to transform a blank canvas into a hyper-personalized adaptive operating system.

PROBLEM: Modern productivity tools force users into pre-set structures (Kanban, List, Grid) that are usually either underwhelming or overwhelming. Users spend more time setting up the tool rather than doing the work and they end up stuck with generic, corporate aesthetics that kill both creativity and productivity.

SOLUTION: Ethos is a Miro Web SDK integration that acts as a generative layer between the user and the canvas.

- Ethos Sees: It analyzes moodboards and/or prompts to extract your visual identity (Colors, Organization)

- Ethos Thinks: It digests raw data (URLs, PDFs, Brain Dumps) using Google Gemini and integrates it into your workspace.

- Ethos Morphs: It automatically generates Miro widgets, layouts, and structures that match how the user thinks (e.g, an "Architect" wants a grind, while an "Artist" wants a mindmap).

TECHNICAL STACK
Frontend: Next.js + React
Styling: Tailwind CSS + shadcn/ui + Lucide React
AI Engine: Google Gemini 1.5 Flash via Google AI Studio
Integration: Miro Web SDK 2.0
Backend: Supabase (PostgreSQL)
Hosting: Vercel

UI AESTHETICS
BG: #FDFCFB (Bone white)
TEXT: #1A1A1A (Soft black)
BORDERS: #E5E5E1 (Thin)
TYPOGRAPHY: Instrument Serif, Inter sans-serif

UI FIXING

CONTEXT: Building the "Ethos" Sidebar UI I have `app.tsx` set up. I need to build a Sidebar UI using Tailwind CSS and `lucide-react` for icons. The aesthetic is **'Sophisticated Minimalist'**:

- **Background:** `#FDFCFB` (Bone White)
- **Text:** `#1A1A1A` (Soft Black)
- **Borders:** `#E5E5E1` (Thin, crisp 1px)
- **Fonts:** `Instrument Serif` (Headers) and `Inter` (Body).

REQUIREMENTS:

1. Setup & Imports

- Import `lucide-react` icons: `Eye` (for Vibe), `Brain` (for Brain), `Archive` (for Memory), and `Loader2` (for loading).
- **CRITICAL:** Add this style tag at the top of the component to load the fonts:
  CSS

```
None

@import
url('https://fonts.googleapis.com/css2?family=Instrument+Serif:it
al@0;1&family=Inter:wght@300;400;600&display=swap');
```

-
- Apply `font-family: 'Instrument Serif', serif;` to all Headers.
- Apply `font-family: 'Inter', sans-serif;` to the body/inputs.

2. State Management Create state variables for:

- `activeTab` (string: 'vibe', 'brain', 'memory')
- `inputText` (string)
- `selectedLayout` (string, default 'grid')
- `isProcessing` (boolean)
- `processingStatus` (string, e.g., "Reading your vibe...", "Constructing grid...")

3. The Layout (Mobile-Responsive)

- **Container:** `h-screen w-full bg-[#FDFCFB] flex flex-col`.

- **Mobile Handling:** Use a Tailwind media query so that on small screens (`<600px`), the sidebar becomes a **bottom sheet** fixed to the bottom of the screen (`h-[50vh] w-full absolute bottom-0 rounded-t-xl shadow-xl`).

4. The Tabbed Interface Create a segmented control at the top with 3 tabs.

- **Tab 1: 'Vibe' (Icon: Eye)**
  - Content: A dashed-border file upload zone (`border-dashed border-2 border-[#E5E5E1]`) that says "Drop Moodboard".
  - Action: A primary button "Sync Ethos".
- **Tab 2: 'Brain' (Icon: Brain)**
  - Content: A minimalist textarea (no shadow, border `#E5E5E1`) for Brain Dumps/URLs.
  - Controls: A dropdown for 'Layout DNA' (Options: Architect [Grid], Gardener [MindMap], Pilot [Timeline]).
  - Action: A primary button "Crystalize".
- **Tab 3: 'Memory' (Icon: Archive)**
  - Content: A list of saved states (placeholder for now).

5. Styling Specs (Tailwind)

- **Buttons:** `bg-[#1A1A1A] text-white rounded-md px-4 py-3 hover:opacity-90 transition-all font-medium flex items-center justify-center gap-2`.
- **Inputs:** `bg-white border border-[#E5E5E1] rounded-md p-3 text-sm focus:ring-1 focus:ring-[#1A1A1A] outline-none`.
- **Loading State:** When `isProcessing` is true, disable inputs and show the `processingStatus` text pulsing with a `Loader2` spinner.

The Font Hack: Since you can't easily upload fonts to Miro's canvas, use a specific CSS font-family in your Sidebar to sell the vibe.

Add to CSS: @import url('https://fonts.googleapis.com/css2?family=Instrument+Serif:ital@0;1&family=Inter:wght@300;400;600&display=swap');

Apply: font-family: 'Instrument Serif', serif; for all your headers.

The "Magic" Loading State: While Gemini is thinking, don't just show a spinner. Show a text loop:

"Reading your vibe..."

"Mixing palette..."

"Refining aesthetics..." (Judges love "delightful" UI details).

Mobile Triage: In your CSS, use media queries.

CSS
```
@media (max-width: 600px) {
  /* Make the sidebar full width on mobile */
  .sidebar-container { width: 100%; height: 50vh; bottom: 0; position: absolute; }
}
```

Generate the full React component code now and let me know where exactly to place it.

AESTHETICS ENGINE

CONTEXT: Building the "Ethos Sees" (Aesthetics Engine) Logic

I need to implement the handleSyncStyle function in app.tsx. This function acts as the bridge between the user's uploaded moodboard and the Miro board's visual style.

DEPENDENCIES:

- @google/generative-ai (Gemini SDK) is installed.
- miro.board (Miro Web SDK) is available.
- State variables: isProcessing (bool), processingStatus (string).

REQUIREMENTS:

1. The Trigger

- The function is triggered when a user uploads a file to the input in the 'Vibe' tab.
- **Step 1:** Set isProcessing(true).
- **Step 2:** Update processingStatus to "Reading your vibe...".

2. Image Processing & Gemini Vision

- Create a helper to convert the uploaded File object to a Base64 string (remove the prefix if needed for the Gemini API, or handle it as a standard InlineData part).
- **API Call:** Send the Base64 image to **Gemini 1.5 Flash**.
    - **System Prompt:**
      "Analyze this image. You are a Design System Expert. Extract the visual identity.
      Return a **STRICT JSON** object (no markdown formatting, no ```json blocks) with:
        1. palette: object with 5 hex codes: background, primary, secondary, accent, text.
        2. fontVibe: string (either 'serif', 'sans', or 'handwritten')."

3. JSON Sanitization (CRITICAL FIX)

- Gemini might wrap the response in markdown. Before parsing, run a cleanup:
  JavaScript

```
None

const cleanJson = responseText.replace(/```json/g,
'').replace(/```/g, '').trim();
```

-

- Parse cleanJson into an object.

4. The "Ethos Injection" (Miro SDK)

- Update processingStatus to "Mixing palette...".
- Call const selection = await miro.board.getSelection().
- If selection is empty, show an alert: "Select items on the board first to infuse them with Ethos."
- **Loop through selected items:**
    - **If sticky_note:** Set style.fillColor to palette.primary.
    - **If shape:** Set style.fillColor to palette.secondary, style.color to palette.text.
    - **If text:** Set style.color to palette.text.
    - **Font Logic:** Map fontVibe to Miro's available fonts.
        - If serif $\rightarrow$ fontFamily: 'tiempos' (or closest Miro equivalent).
        - If sans $\rightarrow$ fontFamily: 'inter' (or closest Miro equivalent).
- **Background (Optional):** If possible via SDK, set the board background color to palette.background.

5. Cleanup

- Set isProcessing(false).
- Update processingStatus to "Ethos Injected.".

Generate the complete handleSyncStyle function and the helper fileToBase64 function now and tell me exactly where to put it.

NOTEBOOKLM BRAIN

CONTEXT: Building the "Ethos Thinks" (Layout Engine) Logic I need to implement the `handleGenerateLayout` function in `app.tsx`. This function takes raw text/URLs, interprets them via AI, and spatially arranges them on the Miro board.

DEPENDENCIES:

- `@google/generative-ai` (Gemini SDK) is installed.
- `miro.board` (Miro Web SDK) is available.
- State variables: `isProcessing` (bool), `processingStatus` (string).

REQUIREMENTS:

1. The Trigger

- The function accepts `inputText` (string) and `selectedLayout` (string: 'Grid', 'MindMap', 'Timeline').
- **Step 1:** Set `isProcessing(true)`.
- **Step 2:** Update `processingStatus` to "Crystalizing thoughts...".

2. The Intelligence (Gemini API)

- Call `model.generateContent` with a robust system prompt.
    - **System Prompt:**
      "You are an expert Information Architect for Miro. User Input: ${inputText} Desired Layout: ${selectedLayout} Task: Summarize the input into 6-9 distinct, high-value atomic thoughts. Return a **STRICT JSON array** (no markdown, no code blocks) of objects. **Coordinate Logic (Calculate these precisely):**
        - **If 'Grid':** 3 columns. Start at x:0, y:0. Increment x by 250 for columns, y by 250 for rows.
        - **If 'Timeline':** Start at x:0, y:0. All items have y:0. Increment x by 300 for each step.
        - **If 'MindMap':** Item 0 is the 'Core Theme' at x:0, y:0. For items 1-N, arrange them in a perfect circle radius 400 around the center. (Use cos/sin math: x = 400 * cos(angle), y = 400 * sin(angle)).
- **JSON Structure:** [{ "content": "Summary point...", "x": 0, "y": 0, "shape": "sticky_note" }]"

3. JSON Sanitization (CRITICAL)

- The AI might return markdown (```json). Sanitize it before parsing:
  JavaScript

```
None

const cleanJson = responseText.replace(/```json/g,
'').replace(/```/g, '').trim();
const layoutItems = JSON.parse(cleanJson);
```

-

4. The Rendering Loop

- Update `processingStatus` to "Constructing ${selectedLayout}...".
- Loop through `layoutItems`:
    - Create the item using `await miro.board.createStickyNote({ content: item.content, x: item.x, y: item.y })`.
    - **Styling Detail:** If possible, set the sticky note color to 'white' or 'gray' to match the "Ethos" minimalist aesthetic.

5. Viewport Management

- After the loop, gather all new items.
- Call `await miro.board.viewport.zoomTo(newItems)` so the user instantly sees the result.

6. Cleanup

- Set `isProcessing(false)`.
- Update `processingStatus` to "Done.".

Generate the complete `handleGenerateLayout` function now and tell me exactly where to put it.

SAVE/LOAD LOGIC

CONTEXT: Building the "Ethos Memory" (Persistence Engine) Logic I need to implement the `saveBoardState` and `loadBoardState` functions in `app.tsx` to allow users to snapshot their board and restore it later.

DEPENDENCIES:

- `@supabase/supabase-js` is installed and initialized as `supabase` client.
- `miro.board` (Miro Web SDK) is available.
- State variables: `savedBoards` (array), `isProcessing` (bool), `processingStatus` (string).

REQUIREMENTS:

1. The Save Logic (`saveBoardState`)

- Triggered by a "Save State" button in the 'Memory' tab.
- **Step 1:** Prompt the user for a name: `const name = prompt("Name this Ethos state:");` (Simple JS prompt is fine for hackathon).
- **Step 2:** Update `processingStatus` to "Archiving workspace...".
- **Step 3:** Get all board items: `const items = await miro.board.get()`.
- **Step 4:** Filter items. We only want to save 'sticky_note', 'shape', and 'text' to keep the payload light.
- **Step 5:** Save to Supabase:
  JavaScript

```
None

const { error } = await supabase
  .from('boards')
  .insert({ name: name, data: items, created_at: new Date() });
```

-
- **Step 6:** Handle errors. If success, refresh the list of boards.

2. The Load Logic (`loadBoardState`)

- Triggered when a user clicks a saved item in the 'Memory' list.
- **Step 1:** Confirm action: `if (!confirm("This will clear the current board. Proceed?")) return;`
- **Step 2:** Update `processingStatus` to "Restoring Ethos...".

- **Step 3: Wipe the Canvas.**
  JavaScript

```
None

const currentItems = await miro.board.get();
await miro.board.remove(currentItems);
```

-
- **Step 4: Reconstruct.** Loop through the saved `data` JSON array.
  - Use a `switch` statement on `item.type`.
  - **Case 'sticky_note':** `await miro.board.createStickyNote({ content: item.content, x: item.x, y: item.y, style: item.style })`
  - **Case 'shape':** `await miro.board.createShape({ content: item.content, x: item.x, y: item.y, style: item.style, width: item.width, height: item.height })`
  - **Case 'text':** `await miro.board.createText({ content: item.content, x: item.x, y: item.y, style: item.style })`
  - *Default:* Ignore other types to prevent errors.

3. The Fetch Logic (`fetchSavedBoards`)

- Create a `useEffect` hook that loads saved boards from Supabase on mount.
- Query: `const { data } = await supabase.from('boards').select('*').order('created_at', { ascending: false })`.
- Store result in `savedBoards` state.

4. UI Rendering (Memory Tab)

- In the 'Memory' tab content area, map through `savedBoards`.
- Render each as a card:
  - **Title:** `board.name` (font-serif).
  - **Date:** `new Date(board.created_at).toLocaleDateString()` (text-xs text-gray-400).
  - **Action:** On click, call `loadBoardState(board)`.

Generate the complete functions and the useEffect hook now and tell me exactly where to put it.

GOOD DETAILS