Build Your Own AI-Powered Asset Manager

A complete guide: from zero to a production-ready DAM with Google Drive, Gemini AI, Supabase, and Next.js

DAM

Version 1.0 | 2026

Internal Use Only

What You're Building

This guide walks you through building a custom Digital Asset Manager (DAM) that runs locally on a Mac, connects to your Google Drive, and uses AI to automatically tag and search 9,000+ photos and videos using natural language.

You don't need to be a developer. This guide assumes you're starting from nothing — no code, no accounts, no infrastructure. By the end, you'll have a working app that lets anyone on your team type *"dark moody product shot"* and instantly find the right asset.

What the finished app does:

- Connects to any Google Drive folder as your asset library
- Uses Google's Gemini AI to auto-tag every photo and video with descriptive labels
- Stores all tags in a cloud database (Supabase) for fast search
- Provides a web UI with natural language search and folder browsing
- Runs locally — open a Terminal, run one command, open a browser
- Handles 9,000+ assets with incremental indexing (only new files are re-processed)

Frontend stack	Backend & services
• Next.js 14 (React framework)	• Google Drive API
• Tailwind CSS (styling)	• Google Gemini AI (vision tagging)
• TypeScript	• Supabase (PostgreSQL database)
• Node.js runtime	• Google OAuth (login)

What's In This Guide

Step 1	**Set up your accounts & API keys**	All the free services you need before writing a line of code
Step 2	**Create the project**	Scaffold a Next.js app with the right folder structure
Step 3	**Connect Google Drive**	OAuth flow and the Drive API — browse folders and list files
Step 4	**Set up the database**	Create your Supabase project and the assets table schema

Step 5	**Build AI tagging**	Use Gemini Vision to analyze each asset and store searchable tags
Step 6	**Build the search engine**	Text search across AI-generated tags with filters
Step 7	**Build the frontend UI**	The search bar, folder sidebar, and asset grid
Step 8	**Add authentication**	Lock the app to your Google Workspace domain
Step 9	**Deploy locally**	Run it on any Mac with a single terminal command
SOP	**How to use the app (daily reference)**	Day-to-day use reference

1 Set Up Your Accounts & API Keys

Before writing any code, create these 4 free accounts

1.1 Google Cloud Console

This is where you get credentials to talk to Google Drive and handle login.

- Go to **console.cloud.google.com** and sign in with your Google Workspace account
- Click **New Project** → name it *My-DAM* → Create
- In the left menu: **APIs & Services** → **Library**
- Search for and enable: **Google Drive API**
- Go to **APIs & Services** → **OAuth consent screen**
- Choose **Internal** (only your org can log in) → fill in App name: *My Asset Manager*
- Go to **APIs & Services** → **Credentials** → **Create Credentials** → **OAuth 2.0 Client ID**
- Application type: **Web application**
- Authorized redirect URI: http://localhost:3000/api/auth/callback/google
- Click Create → **copy your Client ID and Client Secret** (you'll need these)

1.2 Google AI Studio (Gemini API)

This powers the AI tagging — Gemini Vision analyzes each photo/video and writes descriptions.

- Go to **aistudio.google.com**
- Click **Get API key** → **Create API key**
- Copy the key — it looks like: AIzaSy...

■ *Gemini Vision is free for moderate usage. At 9,000 assets, expect to stay within free tier limits during initial indexing.*

1.3 Supabase (Database)

Supabase is a free hosted PostgreSQL database. It stores all your AI-generated asset tags.

- Go to **supabase.com** → Sign up → New Project
- Name: *my-dam*, pick a region close to you, set a database password
- Wait ~2 min for provisioning

- Go to **Project Settings** → **API**
- Copy: **Project URL** and **anon public key**

1.4 Node.js

Node.js is the JavaScript runtime that runs the app on your Mac.

- Go to **nodejs.org** and download the LTS version
- Install it (standard Mac .pkg installer)
- Verify: open Terminal and run node --version — you should see v20.x.x or higher

2 Create the Project

Scaffold a Next.js app with the correct structure

2.1 Scaffold the app

Open Terminal and run these commands one at a time:

```
# Navigate to your Desktop
cd ~/Desktop

# Create the Next.js project
npx create-next-app@latest my-dam
```

When prompted, answer:

- TypeScript: **Yes**
- ESLint: **Yes**
- Tailwind CSS: **Yes**
- App Router: **Yes**
- All other prompts: default (press Enter)

2.2 Install dependencies

```
cd ~/Desktop/my-dam

# Google Drive + Auth
npm install googleapis next-auth @auth/supabase-adapter

# Database client
npm install @supabase/supabase-js

# Google Gemini AI
npm install @google/generative-ai
```

2.3 Create your .env.local file

This file holds all your secret keys. It lives in your project folder and is never committed to git or shared. Create it:

```
# In Terminal, from your project folder:
touch ~/Desktop/my-dam/.env.local
```

```
# Open it in TextEdit or any editor and paste:
GOOGLE_CLIENT_ID=your_client_id_here
GOOGLE_CLIENT_SECRET=your_client_secret_here
GEMINI_API_KEY=your_gemini_key_here
NEXT_PUBLIC_SUPABASE_URL=your_supabase_url_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key_here
NEXTAUTH_SECRET=any_random_string_32_chars
NEXTAUTH_URL=http://localhost:3000
BRAND_FOLDER_ID=your_google_drive_folder_id
```

■ *To find your Brand Folder ID: open the folder in Google Drive, look at the URL — the long string after /folders/ is the ID.*

3 Connect Google Drive

OAuth login and Drive API — list folders and fetch files

3.1 Set up NextAuth for Google login

Create the file app/api/auth/[...nextauth]/route.ts:

```
import NextAuth from 'next-auth'
import GoogleProvider from 'next-auth/providers/google'

const handler = NextAuth({
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
      authorization: {
        params: {
          scope: 'openid email profile https://www.googleapis.com/auth/drive.readonly'
,
        },
      },
    }),
  ],
  callbacks: {
    async jwt({ token, account }) {
      if (account) token.accessToken = account.access_token
      return token
    },
    async session({ session, token }) {
      session.accessToken = token.accessToken as string
      return session
    },
  },
})
export { handler as GET, handler as POST }
```

3.2 Create a Drive API helper

Create lib/drive.ts:

```
import { google } from 'googleapis'

export function getDriveClient(accessToken: string) {
  const auth = new google.auth.OAuth2()
  auth.setCredentials({ access_token: accessToken })
  return google.drive({ version: 'v3', auth })
}

// List all subfolders inside a parent folder
export async function listFolders(accessToken: string, parentId: string) {
  const drive = getDriveClient(accessToken)
  const res = await drive.files.list({
    q: `'${parentId}' in parents and mimeType='application/vnd.google-apps.folder'`,
    fields: 'files(id, name)',
    pageSize: 200,
  })
  return res.data.files || []
}

// List image/video files in a folder
export async function listAssets(accessToken: string, folderId: string) {
  const drive = getDriveClient(accessToken)
  const res = await drive.files.list({
    q: `'${folderId}' in parents and (mimeType contains 'image/' or mimeType contains 'video/')`,
    fields: 'files(id, name, mimeType, thumbnailLink, webViewLink)',
    pageSize: 1000,
  })
  return res.data.files || []
}
```

4 Set Up the Database

Create the Supabase table that stores all asset tags

4.1 Create the assets table

In your Supabase dashboard, go to **SQL Editor** and run this query to create the table that stores every asset and its AI-generated tags:

```
create table assets (
  id          uuid primary key default gen_random_uuid(),
  file_id     text unique not null,    -- Google Drive file ID
  name        text not null,           -- filename
  mime_type   text,
  folder_id   text,
  folder_name text,
  thumbnail   text,                    -- Google Drive thumbnail URL
  web_link    text,
  tags        text[],                  -- AI-generated tag array
  description text,                    -- AI-generated description
  indexed_at  timestamp default now()
);

-- Index for fast text search across all tag arrays
create index assets_tags_gin on assets using gin(tags);

-- Index for folder filtering
create index assets_folder_id on assets(folder_id);
```

4.2 Create the Supabase client

Create lib/supabase.ts:

```
import { createClient } from '@supabase/supabase-js'

export const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
)
```

5 Build AI Tagging

Use Gemini Vision to analyze assets and store searchable tags

5.1 Create the AI tagging function

Create lib/tagger.ts. This function takes a file from Drive, sends it to Gemini Vision, and gets back descriptive tags:

```
import { GoogleGenerativeAI } from '@google/generative-ai'

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY!)

export async function tagAsset(imageUrl: string, mimeType: string) {
  const model = genAI.getGenerativeModel({ model: 'gemini-1.5-flash' })

  const prompt = `Analyze this image and return a JSON object with:
    - tags: array of 8-15 descriptive keywords (subjects, mood, colors,
      setting, style, product category)
    - description: one sentence describing the image
    Return only valid JSON, no markdown.`

  // Fetch the image bytes from Drive
  const imageRes = await fetch(imageUrl)
  const imageData = await imageRes.arrayBuffer()
  const base64 = Buffer.from(imageData).toString('base64')

  const result = await model.generateContent([
    { inlineData: { data: base64, mimeType } },
    prompt,
  ])

  const text = result.response.text()
  return JSON.parse(text) as { tags: string[], description: string }
}
```

5.2 Create the indexing API route

Create app/api/index-library/route.ts. This is what runs when a user clicks 'Index Library':

```
import { getServerSession } from 'next-auth'
import { listAssets } from '@/lib/drive'
```

```
import { tagAsset } from '@/lib/tagger'
import { supabase } from '@/lib/supabase'

export async function POST() {
  const session = await getServerSession()
  if (!session?.accessToken) return Response.json({ error: 'Unauthorized' }, { status:
 401 })

  // Get all files from Drive
  const files = await listAssets(session.accessToken, process.env.BRAND_FOLDER_ID!)

  // Get already-indexed file IDs (don't re-process)
  const { data: existing } = await supabase.from('assets').select('file_id')
  const indexed = new Set(existing?.map(r => r.file_id) || [])

  const newFiles = files.filter(f => !indexed.has(f.id!))
  let processed = 0

  // Process in batches of 5
  for (let i = 0; i < newFiles.length; i += 5) {
    const batch = newFiles.slice(i, i + 5)
    await Promise.all(batch.map(async (file) => {
      const aiResult = await tagAsset(file.thumbnailLink!, file.mimeType!)
      await supabase.from('assets').upsert({
        file_id: file.id,
        name: file.name,
        mime_type: file.mimeType,
        thumbnail: file.thumbnailLink,
        web_link: file.webViewLink,
        tags: aiResult.tags,
        description: aiResult.description,
      })
      processed++
    }))
  }

  return Response.json({ indexed: processed, total: files.length })
}
```

6 Build the Search Engine

Full-text search across AI tags with optional folder filter

Create app/api/search/route.ts:

```
import { supabase } from '@/lib/supabase'

export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const query = searchParams.get('q') || ''
  const folderId = searchParams.get('folder')

  let dbQuery = supabase
    .from('assets')
    .select('*')
    .limit(100)

  // Folder filter
  if (folderId) dbQuery = dbQuery.eq('folder_id', folderId)

  // Text search: match query against tags array and description
  if (query) {
    const terms = query.toLowerCase().split(' ').filter(Boolean)
    dbQuery = dbQuery.or(
      terms.map(t => `tags.cs.{${t}},description.ilike.%${t}%`).join(',')
    )
  }

  const { data, error } = await dbQuery
  if (error) return Response.json({ error }, { status: 500 })
  return Response.json({ assets: data })
}
```

7 Build the Frontend UI

Search bar, folder sidebar, and asset grid

7.1 Main page layout

Replace app/page.tsx with:

```
'use client'
import { useState, useEffect } from 'react'
import { useSession } from 'next-auth/react'

export default function Home() {
  const { data: session } = useSession()
  const [query, setQuery] = useState('')
  const [assets, setAssets] = useState<any[]>([])
  const [folders, setFolders] = useState<any[]>([])
  const [activeFolder, setActiveFolder] = useState<string | null>(null)

  // Search whenever query or folder changes
  useEffect(() => {
    const params = new URLSearchParams()
    if (query) params.set('q', query)
    if (activeFolder) params.set('folder', activeFolder)
    fetch(`/api/search?${params}`)
      .then(r => r.json())
      .then(d => setAssets(d.assets || []))
  }, [query, activeFolder])

  if (!session) return <SignInPrompt />

  return (
    <div className='flex h-screen bg-gray-50'>
      <Sidebar folders={folders} active={activeFolder} onSelect={setActiveFolder} />
      <main className='flex-1 flex flex-col'>
        <SearchBar value={query} onChange={setQuery} />
        <AssetGrid assets={assets} />
      </main>
    </div>
  )
```

```
}
```

7.2 Asset grid component

Each asset card shows the thumbnail, filename, and AI description:

```
function AssetGrid({ assets }: { assets: any[] }) {
  return (
    <div className='grid grid-cols-4 gap-3 p-4 overflow-y-auto'>
      {assets.map(asset => (
        <a key={asset.file_id} href={asset.web_link} target='_blank'
          className='bg-white rounded-lg shadow-sm hover:shadow-md transition-shadow'>
          <img src={asset.thumbnail} alt={asset.name}
               className='w-full h-40 object-cover rounded-t-lg' />
          <div className='p-2'>
            <p className='text-xs font-medium truncate'>{asset.name}</p>
            <p className='text-xs text-gray-400 truncate'>{asset.description}</p>
          </div>
        </a>
      ))}
    </div>
  )
}
```

8 Add Authentication

Lock the app to your Google Workspace domain

Add a domain check to your NextAuth config so only your organization's accounts can log in. Update app/api/auth/[...nextauth]/route.ts:

```
callbacks: {
  async signIn({ account, profile }) {
    // Only allow your org's Google accounts
    return profile?.email?.endsWith('@yourcompany.com') ?? false
  },
  async jwt({ token, account }) {
    if (account) token.accessToken = account.access_token
    return token
  },
  async session({ session, token }) {
    session.accessToken = token.accessToken as string
    return session
  },
},
```

8.2 Protect all pages with middleware

Create middleware.ts in the root of the project:

```
export { default } from 'next-auth/middleware'

export const config = {
  matcher: ['/((?!api/auth|_next|favicon).*)'],
}
```

■ *This redirects any unauthenticated user to the sign-in page automatically.*

9 Deploy Locally

Run the finished app on any Mac with one command

9.1 Run the development server

```
cd ~/Desktop/my-dam && npm run dev
```

Wait for ✓ Ready in Xs, then open **http://localhost:3000** in Chrome or Safari.

9.2 First-time setup checklist

- ✓ **Sign in** Click Sign in with Google → authorize with your Google Workspace account
- ✓ **Verify Drive access** The left sidebar should populate with your Brand folder structure
- ✓ **Run initial indexing** Click Index Library — first run will take a while (9,000+ assets)
- ✓ **Test search** Type a concept like 'campfire night' — results should appear
- ✓ **Verify thumbnails** If images show blank, hard-refresh with Cmd+Shift+R

Sharing the app with your team

- Each team member needs Node.js installed on their Mac
- Share the my-dam folder (with .env.local) via a secure method — NOT via email or Slack
- Each user runs the same command: cd ~/Desktop/my-dam && npm run dev
- The Supabase database is shared — everyone searches the same indexed asset library
- Only one person needs to run Index Library when new assets are added

Daily Use Reference (SOP)

The following section is the Standard Operating Procedure for day-to-day use of your Asset Manager once it's built and running.

Opening the App

- Press **Command + Space**, type **Terminal**, press Enter
- Paste: cd ~/Desktop/my-dam && npm run dev then press Return
- Wait for: ✓ Ready in Xs
- Open Chrome or Safari → go to **http://localhost:3000**
- Sign in with your your Google Workspace account

■ *If you see 'Port 3000 is in use, trying 3001' — go to http://localhost:3001 instead.*

Searching

Type any concept, mood, product, or description. The AI understands natural language — no exact filenames needed.

`camp chair outdoor`	`dark moody product shot`
`lifestyle people smiling`	`winter snow adventure`
`flask campfire`	`white background product`

Adding New Assets

- Upload new photos/videos to your Google Drive Brand folder as usual
- Open the app → click **Index Library** in the top right
- Wait for indexing to complete (about 20-25 seconds per 5 new assets)
- New assets are now searchable

Closing the App

- Click the Terminal window → press **Ctrl + C**
- Press **Command + Q** to fully quit Terminal

- Close the browser tab

■ *Do NOT just close the Terminal window — this leaves background processes running. Always Ctrl+C first.*

Troubleshooting

localhost refused to connect	Terminal is not running. Open Terminal and run the start command.
npm error: cannot find package.json	You're in the wrong folder. Always use the full command: cd ~/Desktop/my-dam && npm run dev
Port 3000 is in use	Another instance is running. Try http://localhost:3001, or Ctrl+C and restart.
Search returns no results	Assets may not be indexed. Click Index Library and wait.
No preview available	Google Drive thumbnail issue. Hard refresh: Command + Shift + R
App is slow to load	Normal on first load — fetching Drive data. Wait 10-15 seconds.

Important Accounts & Credentials

These four accounts power the entire app. Keep them accessible — losing access to any of them will break the app.

Service	Where	What it does
Google Workspace	`@yourcompany.com`	Drive access and login authentication
Google Cloud Console	`console.cloud.google.com`	OAuth 2.0 credentials (Client ID + Secret)
Google AI Studio	`aistudio.google.com`	Gemini API key — powers AI asset tagging
Supabase	`supabase.com`	Cloud database storing all tags and metadata

■ *Never share your .env.local file or paste credentials into any chat or email. These keys give access to your database and AI billing account. Store a backup copy in your password manager.*