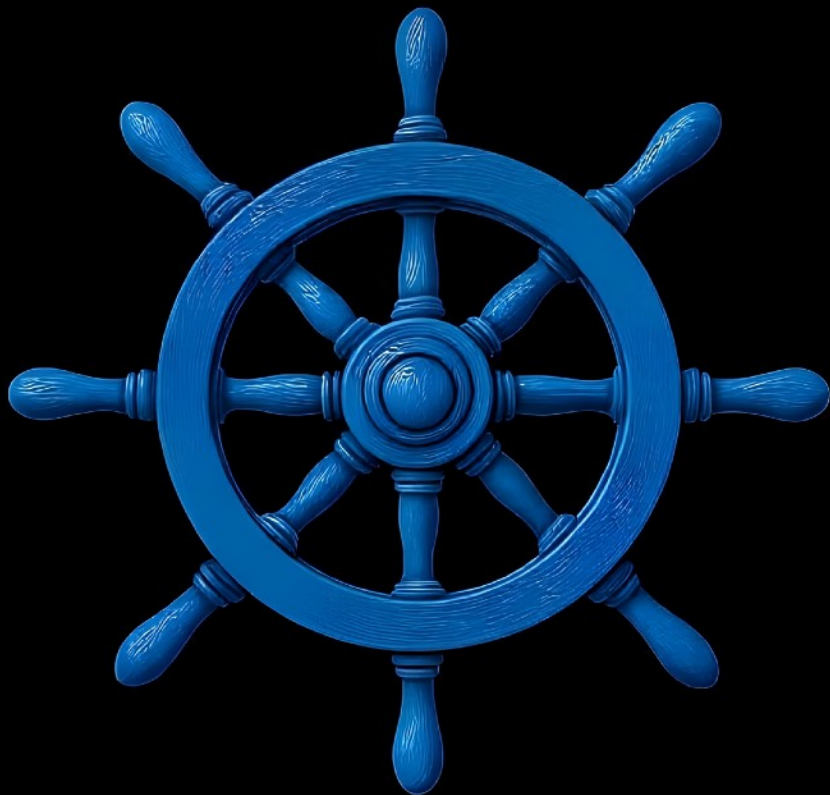


AI AT THE HELM

BUILDING BRAND IN A MINUTE



AI at the Helm

A Case Study on building brandinaminate.com

Hassan Djirdeh

Addy Osmani

© Hassan Djirdeh | Addy Osmani

AI at the Helm – Building brandinaminute.com

www.largeapps.dev/case-studies/ai-at-the-helm

Introduction5

AI APIs7

THE OLD WORLD: CUSTOM MODELS AND HIGH BARRIERS7

TODAY: AI AS A SERVICE.....7

AI Wrappers9

BrandInAMinute.....11

THE IDEA11

THE PRODUCT.....11

THE TECH13

Building BrandInAMinute14

GENERATING A STYLE GUIDE.....15

Parallel Visual Generation.....23

LOGO GENERATION25

HERO IMAGE GENERATION.....27

SOCIAL MEDIA ADS.....29

BRAND ILLUSTRATION.....33

Piecing It All Together.....36

Download Capabilities39

INDIVIDUAL DOWNLOADS39

ZIP ARCHIVE.....39

PDF BRAND GUIDE.....39

Wrap Up42

KEEPING IT SIMPLE42

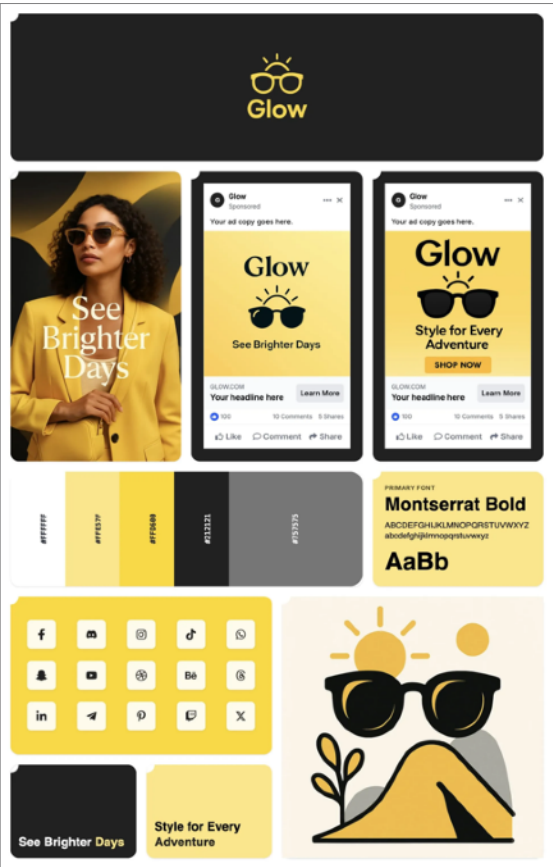
WHAT WE DIDN'T COVER.....42

WHERE TO GO FROM HERE.....43

Introduction

A few years ago, building an AI-powered application meant assembling machine learning (ML) engineers, securing funding for computing resources, and spending months training models. Today, a developer can spin up intelligent features over a weekend. This transformation has fundamentally changed what’s possible for individual builders and small teams.

In this case study, we explore BrandInAMinute, a weekend project we built that generates complete brand identities, including logos, color palettes, and marketing materials in under a minute.



The focus here is on the ideation process, the glue code, prompt engineering, and system design that turn disparate AI capabilities into a more unified product experience. While ChatGPT, Gemini, or Claude can generate individual brand assets, they can't orchestrate a complete, downloadable brand kit with consistent visual identity across multiple formats in one go. It's this orchestration, taking a simple brand description and transforming it into logos, color systems, and marketing materials that work together, that creates value beyond what general-purpose AI tools offer, even though the technical reality is surprisingly straightforward.

Let's begin by understanding how we arrived at this point and how AI APIs (Application Programming Interfaces) evolved into practical building blocks for everyday applications.

AI APIs

The AI development landscape has undergone significant changes over the past decade. What once required dedicated research teams and substantial computational resources is now available through accessible APIs that developers can integrate in minutes. This shift is more than just a technical milestone; it's redefined who can build AI-powered products and how quickly those products can come to life.

The Old World: Custom Models and High Barriers

Consider what building an AI-powered application looked like in 2016. If you wanted to add intelligent features like natural language understanding or image generation, the process was complex. You often needed to hire machine learning (ML) engineers, secure GPU infrastructure, collect and label data, and spend time training and refining models. Even then, success wasn't always certain. Models that performed well in testing often failed in real-world scenarios.

For example, in 2015, Pinterest introduced a visual cropping tool that enables users to search for objects within images. Behind the scenes, this required training custom visual embeddings using deep learning models like VGG16 and later ResNet, powered by carefully curated datasets and investment in infrastructure and talent [[Unifying visual embeddings for visual search at Pinterest | Pinterest Engineering Blog](#)].

Because of these challenges, AI was largely reserved for well-funded startups and tech giants. Individual developers and small teams were rarely able to compete with the resources needed to build and deploy custom models.

Today: AI as a Service

Fast forward to 2025, and the landscape looks entirely different. Companies like [OpenAI](#), [Anthropic](#), [Google](#), and others have commoditized access to state-of-the-art AI models through simple HTTP

APIs. What used to take months of engineering can now be done in a weekend with just a few lines of code and a credit card.

These APIs have turned AI capabilities into modular components that developers can use much like any other service. Add text generation here, image creation there, and speech recognition where needed. The complexity of training, fine-tuning, and hosting models is handled by the providers, who spread those costs across thousands of customers. This model enables developers to focus on building products rather than infrastructure.

A project like Pinterest's early visual search, which once required custom visual embeddings, model experimentation, and GPU-backed infrastructure, could now be quickly prototyped using off-the-shelf APIs such as [OpenAI's Vision Capabilities](#) or [Gemini's Image Understanding](#). Instead of building models from scratch, developers can focus on crafting the user experience and connecting prebuilt tools to deliver meaningful results in hours rather than months.

It's worth noting that not all use cases can or should rely on general-purpose APIs. Custom models remain essential for applications that require strict control over outputs, work with proprietary data, demand highly optimized performance, or operate in specialized domains. The rise of accessible APIs hasn't eliminated the need for custom AI; it has simply broadened the range of who can start building. Even with today's advanced APIs, Pinterest's visual search across 200+ billion images may require custom solutions to meet their scale and performance demands.

AI Wrappers

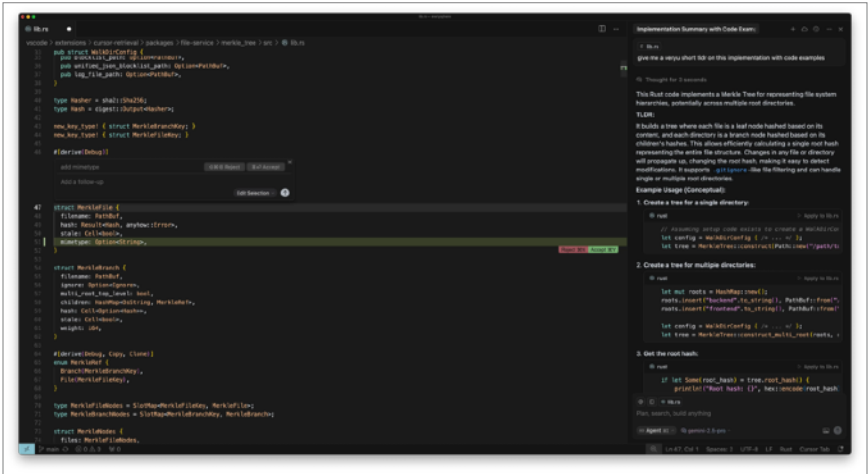
This transformation of AI from “research project” to API endpoint has created an entirely new category of products: **AI wrappers**. These applications don’t train their own models or push the boundaries of machine learning. Instead, they combine existing AI capabilities in thoughtful ways to solve specific problems.

The term “AI wrapper” might sound dismissive, but it describes a powerful pattern. These products take general-purpose AI models and package them into *focused, user-friendly experiences*. They’re the equivalent of building a beautiful storefront for wholesale goods: the value isn’t in manufacturing the product but in making it accessible and useful for specific audiences.

Consider some success stories. Jenni AI, which helps students and academics write better papers, has grown to over 5 million users and generates more than \$600,000 in monthly recurring revenue. The product doesn’t utilize proprietary AI; instead, it wraps existing language models in an interface designed specifically for academic writing workflows.

Cal AI has attracted 3 million users by taking image recognition APIs and turning them into a dead-simple calorie tracking experience. Users snap a photo of their meal, and the app handles the rest. It’s not revolutionary AI, but it’s revolutionary convenience.

Perhaps most striking is Cursor, which has become the fastest-growing SaaS product in history, scaling from \$1 million to \$100 million in annual recurring revenue in just 12 months. Here’s the thing: Cursor doesn’t even have its own AI model. It’s built on top of GPT-4, Claude, and other models that are accessible to anyone. What Cursor does differently is create the best possible coding experience around those models, with features like codebase-wide context, intelligent autocomplete, and natural language editing that feel magical in practice.



These examples reveal an essential truth about today's AI economy. In a world where cutting-edge AI capabilities are just an API call away, the winning strategy often isn't to build better AI; it's to **develop better products with the AI that already exists**. The real work happens in the last mile: understanding user needs, designing intuitive interfaces, and orchestrating multiple capabilities into something that feels simple and powerful.

BrandInAMinute

The Idea

We've been building with AI APIs frequently, launching various hobby products and experiments. Each time we started something new, we'd go through the same brand creation process: generate a logo here, tweak colors there, search for fonts that match, and create social assets separately. Even with powerful AI tools at our disposal, we were still doing the work of making everything feel unified.

We aren't alone in this experience. A quick search for "[logo design and branding kit](#)" on [Fiverr](#) returns thousands of results, with sellers offering packages ranging from \$50 to \$500+. The demand is pretty straightforward: people need complete brand identities, not just individual assets. However, hiring a designer introduces delays and costs that many early-stage projects can't justify.

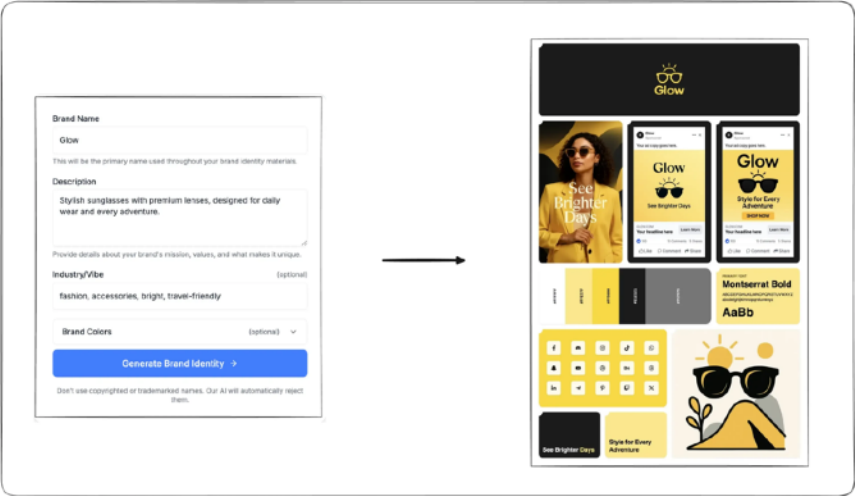
With the emergence of powerful AI image generation APIs like [OpenAI's 4o Image Generation](#) and [Google's Gemini 2.5 image generation capabilities](#), the image quality barrier had finally been crossed. Where earlier models produced blurry, distorted images that barely followed prompts, these new systems could generate crisp, professional visuals that actually matched what you asked for. We saw an opportunity to orchestrate these capabilities into a single, streamlined experience.

The Product

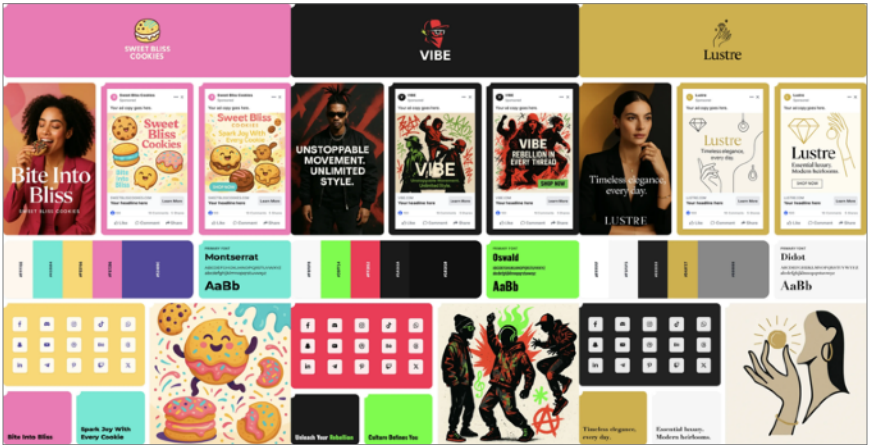
When users land on [BrandInAMinute](#), they're greeted with a simple form that asks for four things: their brand name, a description of what they do, and, optionally, their industry/vibe and recommended colors.

From this minimal input, the system generates a comprehensive brand kit in under a minute. Each kit includes a professionally designed logo, a five-color palette (comprising primary, secondary, highlight, shadow, and background colors), typography recommendations, a portrait hero image, two social media ad templates, a conceptual brand illustration, and two

compelling taglines. Everything is designed to work together as a cohesive system.



You can see the entire process in action by watching the demo video in this [Google Drive link](#) or take a look at some of the demo brand kits we show on the [app's homepage](#):



At the end of the brand kit generation process, users can download individual assets, export everything as a ZIP file, or get a professionally formatted PDF brand guide.

The Tech

Before diving into the AI orchestration and prompt engineering that powers BrandInAMinute, let's quickly outline the technical foundation. We built this as a modern web application using a straightforward stack.

The frontend is built with React and TypeScript, providing type safety and a component-based architecture that makes the UI easy to maintain and extend. For routing and server-side rendering, we chose React Router v7, which gives us a full-stack framework with built-in data loading patterns and seamless client-server transitions.

Styling is handled entirely with Tailwind CSS, which allowed us to rapidly prototype and iterate on the design without writing custom CSS. The utility-first approach paired well with our component library, based on shadcn/ui, providing us with accessible, customizable UI components out of the box.

For payments, we integrated Stripe using their React components and server-side Node SDK. This handles the entire payment flow, from capturing card details to processing transactions securely.

The asset generation pipeline relies heavily on the OpenAI npm package for both text generation (creating style guides) and image generation (logos, ads, and illustrations).

For delivering the brand kit deliverables to users, we implemented download functionality using jsPDF for generating professional PDF brand guides and JSZip for bundling all assets into a downloadable archive.

You can replace any of the above tools with your preference of choice. This is just what we used for BrandInAMinute. The patterns and approaches we'll discuss next are framework-agnostic and can be adapted to whatever stack you're comfortable with.

For the remainder of this guide, we'll focus on the API calls we use, how we stitch them together, and the key aspects that combine these individual capabilities into a cohesive product. **We won't be showing the nitty-gritty of all code implementation**, but rather the key patterns and decisions that make the system work.

Building BrandInAMinute

The generation pipeline of BrandInAMinute orchestrates a two-phase process:

Phase 1: Style Guide Generation

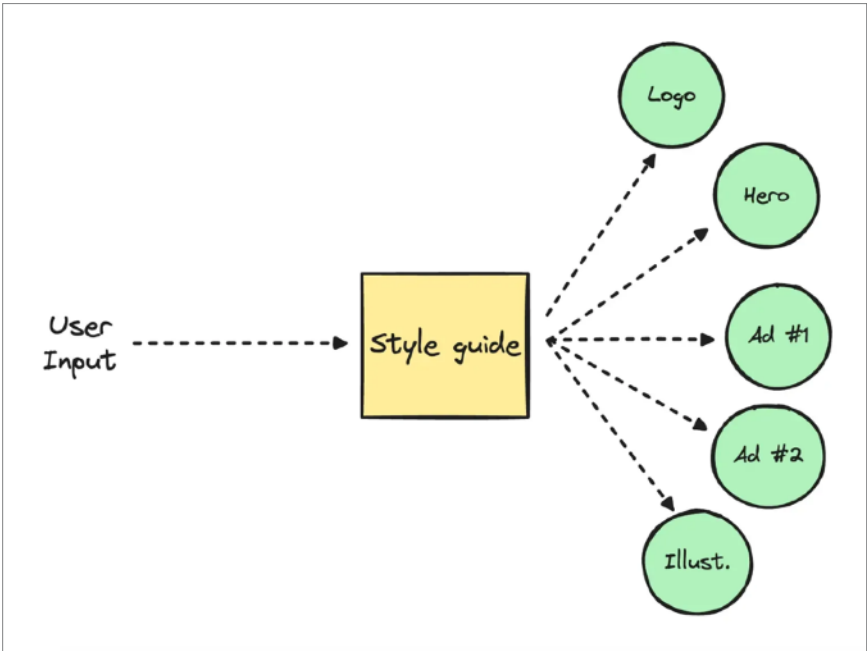
We make a single API call to OpenAI's [GPT-4.1](#) to generate a comprehensive style guide from the user's inputs. This becomes our source of truth for all visual assets.

Phase 2: Parallel Visual Generation

Using the style guide, we fire off five simultaneous image generation requests to OpenAI's image API ([GPT Image 1](#)):

- Logo
- Hero image with tagline
- Social media ad #1
- Social media ad #2 (promotional variant)
- Brand illustration

In total, we make six API calls: one for the style guide and five for visual assets. Let's dive into how each phase works.



Generating a Style Guide

BrandInAMinute starts with a simple form. Users provide a few key pieces of information, and from these minimal inputs, we orchestrate a brand identity. Here's a very simplified version of our React form component:

```

import { useState } from "react";

export function BrandForm() {
  const [brandName, setBrandName] = useState("");
  const [description, setDescription] = useState("");
  const [vibe, setVibe] = useState("");

  const handleSubmit = async (e: React.FormEvent) =>
  {
    e.preventDefault();

    const formData = new FormData();
    formData.append("brandName", brandName);
    formData.append("description", description);
    formData.append("vibe", vibe);

    // Submit to API...
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="brandName"
        placeholder="Enter your brand name"
        required
      />
      <textarea
        name="description"
        placeholder="Describe your brand in a few
sentences"
        required
      />
      <input
        name="vibe"
        placeholder="E.g., tech, luxury, playful,
eco-friendly"
      />
      <button type="submit">Generate Brand Identity</
button>
    </form>
  );
}

```


We need enough information from the user to generate meaningful results, but not so much that users feel overwhelmed. Through experimentation, we found that a brand name and description are essential, while an industry or vibe enhances the results without being mandatory.

When a user submits the form, we might receive user-submitted data like the following:

```
{
  brandName: "Glow",
  description:
    `Stylish sunglasses with premium lenses,
    designed for daily wear and every adventure`,
  vibe: "fashion, accessories, bright, travel-friendly"
}
```

We take these simple inputs and transform them into a comprehensive style guide that will inform every visual asset we generate.

Structured Outputs

In the early days of AI APIs, getting consistent JSON responses was a challenge. You'd ask for JSON and sometimes get it, sometimes get markdown with JSON inside, and sometimes get an apology about not being able to format JSON properly. With OpenAI, that changed with the introduction of [OpenAI's Structured Outputs](#), which brought reliability and control to JSON generation.

With Structured Outputs (in TypeScript), we define precisely what we want using a [Zod](#) schema. Zod is a TypeScript-first schema validation library that lets us define the shape of our data with type safety. In our case, we use it to tell OpenAI precisely what structure we expect for our style guide:

```
import { z } from "zod";

const StyleGuideSchema = z.object({
  style_name: z.string(),
  inspiration: z.array(z.string()),
  color_palette: z.object({
    primary: z.string(),
    secondary: z.string(),
    highlight: z.string(),
    shadow: z.string(),
    background: z.string(),
  }),
  typography: z.object({
    primary_font: z.string(),
    secondary_font: z.string(),
  }),
  themes: z.array(z.string()),
  taglines: z.array(z.string()),
  rendering: z.object({
    technique: z.string(),
    special_effects: z.array(z.string()),
  }),
  asset_background_style: z.object({
    type: z.string(),
    details: z.string(),
  }),
  subjects: z.object({
    style: z.string(),
    details: z.string(),
  }),
  usage_notes: z.string(),
});
```

This schema defines a style guide containing the following:

- **Visual foundation:** Color palette and typography that will be directly used in the brand kit.
- **Creative direction:** Themes, inspiration, and rendering techniques that guide our image generation.
- **Copy elements:** Taglines that appear in our marketing materials.

- **Technical specifications:** Background styles and subject details that inform our prompts.

Before making the API call, we need to craft the prompt. After numerous iterations, we landed on this approach that consistently produces comprehensive, usable style guides:

```
function generateStyleGuidePrompt(
  brandName: string,
  description: string,
  vibe?: string
): string {
  return `
```

```
You are a senior Brand Identity Designer tasked
with generating a JSON object for a visual style
guide.
```

```
<design_brief>
Brand Name: "${brandName}"
Description: ${description}
${vibe ? `Industry/Vibe: ${vibe}` : ""}
Desired Logo Impression: Modern, professional,
memorable, clean.
</design_brief>
```

```
Generate a comprehensive style guide that
includes:
```

- A cohesive color palette (5 colors: primary, secondary, highlight, shadow, background)
- Typography recommendations
- Visual themes and inspiration keywords
- Rendering style for assets (vector, flat design, etc.)
- Two compelling taglines that capture the brand essence
- Specific guidance for visual subjects and backgrounds

```
Ensure all elements work together harmoniously
and reflect the brand's identity.
```

```
` .trim();
}
```

While we don't technically need to specify "return as JSON" thanks to Structured Outputs, we found it helps reinforce the task. This prompt evolved through trial and error, and there's always room for improvement, but it consistently delivers the comprehensive style guides we need.

Now, let's make the actual API call. OpenAI's newer API uses a message format with different roles. The "developer" role sets high-level instructions about how the model should behave, while the "user" role contains the specific request:

```
import { zodTextFormat } from "openai/helpers/zod";

const response = await openai.responses.parse({
  model: "gpt-4.1",
  input: [
    {
      role: "developer",
      content:
        "You are a brand style guide expert. Only " +
        "respond with valid JSON matching the " +
        "provided schema."
    },
    {
      role: "user",
      content: generateStyleGuidePrompt(
        brandName,
        description,
        vibe
      )
    }
  ],
  text: {
    format: zodTextFormat(
      StyleGuideSchema,
      "style_guide"
    )
  }
});

const styleGuide = response.output_parsed;
```

Let's see what this produces for our "Glow" sunglasses brand:

```

{
  "style_name": "Glow Modern Adventure",
  "inspiration": [
    "Minimalist fashion brands (e.g., Warby Parker,",
    "Ray-Ban)",
    "Clean typographic design",
    "Outdoor lifestyle imagery",
    "Urban adventure aesthetics"
  ],
  "color_palette": {
    "primary": "#FFD86F",
    "secondary": "#23272F",
    "highlight": "#FF6F61",
    "shadow": "#A2A7B8",
    "background": "#FFFFFF"
  },
  "rendering": {
    "technique":
      "Flat vector graphics with subtle gradients",
    "special_effects": [
      "Soft drop shadows",
      "Glossy light reflections to evoke lenses",
      "Slight transparency overlays for depth"
    ]
  },
  "asset_background_style": {
    "type": "Clean solid",
    "details":
      "Primary background is white or a subtle " +
      "gradient, keeping focus on the product"
  },
  "subjects": {
    "style": "Stylized realism",
    "details":
      "Sunglasses illustrated or photographed on " +
      "diverse, fashion-forward individuals"
  },
  "themes": [
    "Everyday adventure",
    "Effortless style",
  ],
  "usage_notes":
    "Maintain generous whitespace around logo " +
    "and key elements.",
  "typography": {
    "primary_font": "Montserrat Bold",
    "secondary_font": "Roboto Regular"
  },
  "taglines": [
    "See More. Live Brighter.",
    "Adventure in Style."
  ]
}

```

Notice how different elements serve different purposes in our brand kit:

Direct usage - These appear exactly as specified:

- **Color palette:** All five colors are used throughout the brand kit UI.
- **Typography:** Font recommendations are displayed in the typography card.
- **Taglines:** Featured in hero images and social media ads.

Image generation guidance - These inform our prompts for visual assets:

- **Rendering technique:** “Flat vector graphics” guides the logo style
- **Special effects:** “Glossy light reflections” might appear in product shots
- **Themes:** “Everyday adventure” influences the overall visual narrative
- **Inspiration:** References to Warby Parker help establish the aesthetic benchmark

This style guide becomes the DNA for everything that follows. The logo, ads, and illustration all reference these colors, themes, and visual directions to maintain consistency across the entire brand kit.

Parallel Visual Generation

With our style guide in hand, we now enter the second phase: generating all the visual assets. We need to create five different images, each serving a specific purpose in the brand kit.

Before diving into the code, it's worth noting that OpenAI's GPT Image 1 model offers three quality tiers: low, medium, and high. For BrandInAMinute, we use medium quality throughout, which provides us with the best balance between generation speed, cost, and visual quality.

After generating the style guide, we navigate the user to the /**results** route, where we orchestrate multiple image generations in parallel:

```
// app/routes/results.tsx
export async function loader({
  request
}: LoaderFunctionArgs) {
  const url = new URL(request.url);
  const styleGuideB64 = url.searchParams.get(
    "styleGuide"
  );
  const brandName = url.searchParams.get("brandName");
  const description = url.searchParams.get(
    "description"
  );
  const vibe =
    url.searchParams.get("vibe") || "";

  // Decode the style guide from base64
  const styleGuideJson = atob(styleGuideB64);
  const styleGuide = JSON.parse(styleGuideJson);

  // Generate all assets in parallel
  const [
    imageUrl,
    heroImageUrl,
    adImageUrl,
    secondAdImageUrl,
    brandIllustrationUrl
  ] = await Promise.all([
    generateLogo(
      brandName,
```

```

        description,
        vibe,
        styleGuide
    ),
    generateHeroImage(
        brandName,
        styleGuide.taglines[0],
        vibe,
        styleGuide
    ),
    generateAdImage(
        brandName,
        styleGuide.taglines[0],
        vibe,
        styleGuide,
        "social_media"
    ),
    generateAdImage(
        brandName,
        styleGuide.taglines[1],
        vibe,
        styleGuide,
        "promotional"
    ),
    generateBrandIllustration(
        brandName,
        description,
        vibe,
        styleGuide
    )
  ]);

  return {
    imageUrl,
    heroImageUrl,
    adImageUrl,
    secondAdImageUrl,
    brandIllustrationUrl,
    styleGuide,
    brandName
  };
}

```

The key insight here is using `Promise.all()` for parallel execution. Since each OpenAI image generation takes approximately 30 seconds, running them sequentially would take 150 seconds (2.5 minutes). By running them in parallel, we reduce this to just 30 seconds total.

Let's examine each image generation function and see what it produces.

Logo Generation

The logo is arguably the most important visual asset. It needs to be versatile, memorable, and work across different contexts. Notice how we embed various elements from the style guide directly into the prompt:

```
async function generateLogo(  
  brandName: string,  
  description: string,  
  vibe: string,  
  styleGuide: StyleGuide  
) {  
  const logoPrompt = `  
Generate a professional combination mark logo  
for the brand "${brandName}".  
  
**Structure:** The logo MUST consist of:  
1. A unique, simple, abstract or symbolic  
**icon/symbol**.  
2. The brand name "**${brandName}**" clearly  
typeset beside or below the symbol.  
  
**Visual Style:**  
- Adhere strictly to a  
**${styleGuide.rendering.technique} vector art  
style**.  
- The overall aesthetic should be  
**${styleGuide.style_name}**, reflecting themes of  
${styleGuide.themes.join(", ")}.  
- Inspired by:  
${styleGuide.inspiration.join(", ")}.  
- Subject style: ${styleGuide.subjects.style}  
with ${styleGuide.subjects.details}.  
- **Crucially Important:** Keep the design  
**minimalist, clean, modern, and instantly  
recognizable**.  
  
**Color Palette and Contrast:**  
- Use **ONLY** these colors:  
  - Primary: ${styleGuide.color_palette.primary}  
  - Secondary:  
${styleGuide.color_palette.secondary}  
- **ESSENTIAL:** Set the logo background to the  
primary color (${styleGuide.color_palette.primary})
```

and ensure great contrast.

- The logo symbol and text must have excellent contrast against the chosen background color.

****Typography:****

- The text style should visually match the characteristics of

****`${styleGuide.typography.primary_font}`****.

- The text **MUST** be clearly visible and have high contrast.

****Technical Requirements:****

- Show the logo against the selected colored background.
- Centered composition.
- No photographic elements, no complex illustrations, no realistic textures, not 3D.

****Reference Description:**** The company is:

`${description}. ${vibe ? `Vibe: ${vibe}.` : ""}`
``;`

```
const result = await openai.images.generate({
  model: "gpt-image-1",
  prompt: logoPrompt,
  n: 1,
  size: "1536x1024",
  quality: "medium"
});
```

```
const base64Image = result.data?.[0].b64_json;
return `data:image/png;base64,${base64Image}`;
}
```

For our sunglasses brand, the above will generate a logo that features a minimalist sun or light-related icon (perhaps rays, a lens shape, or abstract circles) rendered in the brand's golden yellow (**#FFD86F**), paired with "GLOW" in clean, bold typography.



Keep in mind that it's AI, so the exact logo design will vary each time. Sometimes, it might be a sunburst, other times overlapping circles suggesting light or even abstract rays, but it will always maintain the specified colors and modern, minimalist aesthetic.

Hero Image Generation

The hero image serves as the primary visual for websites and presentations. It needs to be impactful while incorporating the brand's tagline:

```
async function generateHeroImage(  
  brandName: string,  
  tagline: string,  
  vibe: string,  
  styleGuide: StyleGuide  
) : Promise<string> {  
  const heroPrompt = `  
Create a branded hero image for the company  
"${brandName}".  
The image should reflect the brand's tone:  
"${vibe}" and themes:  
${styleGuide.themes.join(", ")}.  
  
Style the image in a modern, editorial, and
```

polished aesthetic.

Include a subject that visually aligns with the brand's personality.

Use background elements like abstract shapes, gradients, or soft patterns that incorporate:

- Primary color:

```
${styleGuide.color_palette.primary}
```

- Secondary color:

```
${styleGuide.color_palette.secondary}
```

****TEXT PLACEMENT REQUIREMENTS:****

- Overlay this tagline prominently: `"${tagline}"` in bold, modern typography.

- CRITICAL: Position the text with generous margins from all edges (at least 15% from any edge).

- Ensure the text is centered or positioned in a visually balanced way.

- Place text in an area with good contrast for maximum readability.

- Do NOT place text at the very top, bottom, or edges of the image.

Ensure a strong visual hierarchy with clean studio lighting and high clarity.

Do not include logos or any text besides the specified tagline.

```
`;
```

```
const result = await openai.images.generate({
  model: "gpt-image-1",
  prompt: heroPrompt,
  n: 1,
  size: "1024x1536",
  quality: "medium"
});
```

```
const base64Image = result.data?.[0]?.b64_json;
return `data:image/png;base64,${base64Image}`;
```

```
}
```

The above will generate a portrait hero image that showcases a lifestyle scene related to sunglasses and adventure. The AI might create an image with a model wearing sunglasses against a bright background, or perhaps an abstract composition suggesting sunshine and travel. The tagline “See

Brighter Days” should appear prominently in the image with safe margins.



Social Media Ads

We generate two different ad variants to provide variety for marketing campaigns:

```
async function generateAdImage(  
  brandName: string,  
  tagline: string,  
  vibe: string,  
  styleGuide: StyleGuide,  
  adType: string = "social_media"  
) : Promise<string> {
```

```

let adPrompt = "";

if (adType === "social_media") {
  adPrompt = `
Create a compelling social media advertisement
image for the brand "${brandName}".

**Content Requirements:**
- Feature the brand name "${brandName}"
prominently.
- Include this tagline: "${tagline}"
- Create a visually engaging composition for
social media feeds.
- Design for a Facebook/Instagram-style square
format.

**Text Positioning Requirements:**
- Position ALL text elements with generous
margins (at least 15% from any edge).
- Ensure no text is placed too close to borders.
- Place text over areas with appropriate contrast.

**Visual Style:**
- Follow a ${styleGuide.style_name} aesthetic.
- Use themes: ${styleGuide.themes.join(", ")}.
- The image should reflect the brand's tone:
"${vibe}"

**Color Palette:**
- Primary: ${styleGuide.color_palette.primary}
- Secondary: ${styleGuide.color_palette.secondary}
- Highlight: ${styleGuide.color_palette.highlight}

**Background:**
- Type:
${styleGuide.asset_background_style?.type || "clean"}
- Details:
${styleGuide.asset_background_style?.details ||
  "minimal"}
  `;
} else if (adType === "promotional") {
  adPrompt = `
Create a promotional advertisement image for
the brand "${brandName}".

**Content Requirements:**
- Feature the brand name "${brandName}"
prominently.
- Include this tagline: "${tagline}"

```

```

- Create a visually striking composition that
conveys value.
- Design for a web display ad format.

**Visual Style:**
- Follow a ${styleGuide.style_name} aesthetic with
emphasis on eye-catching elements.
- Use themes: ${styleGuide.themes.join(", ")}.
- Create a sense of urgency or opportunity.

**Color Palette:**
- Primary: ${styleGuide.color_palette.primary}
- Secondary: ${styleGuide.color_palette.secondary}
- Highlight: ${styleGuide.color_palette.highlight}
(use for CTA or important elements)

**Technical Requirements:**
- Apply bold, attention-grabbing typography.
- Include visual elements that suggest value.
- Maintain brand consistency while driving action.
  `;
}

const result = await openai.images.generate({
  model: "gpt-image-1",
  prompt: adPrompt,
  n: 1,
  size: "1024x1024",
  quality: "medium"
});

const base64Image = result.data?.[0].b64_json;
return `data:image/png;base64,${base64Image}`;
}

```

The above will generate a social media ad that’s optimized for Instagram or Facebook feeds. The AI will create a square composition featuring sunglasses prominently, with “GLOW” branding and the tagline “See Brighter Days” integrated into the design.

Glow



See Brighter Days

For the promotional variant, the above will generate an ad that's more sales-focused. The AI might create a layout showcasing multiple sunglass styles, special offer messaging, or a dynamic composition with the "Style for Every Adventure" tagline. It will likely include elements suggesting value or urgency, perhaps a "Shop Now" call-to-action or visual elements like arrows or badges.



Brand Illustration

The brand illustration provides a conceptual, artistic representation of the brand's essence:

```
async function generateBrandIllustration(  
  brandName: string,  
  description: string,  
  vibe: string,  
  styleGuide: StyleGuide  
) : Promise<string> {  
  const illustrationPrompt = `
```

```
Create a conceptual illustration that represents  
the essence of the brand "${brandName}".
```

****Content Requirements:****

- This is NOT a logo, but a creative illustration

representing the brand's values.

- Do not include the brand name text in the illustration.
- Create an artistic, conceptual visual that communicates the brand's core essence.
- The illustration should be symbolic and metaphorical rather than literal.

****Brand Details:****

- Brand name: "\${brandName}"
- Brand description: "\${description}"
- Brand vibe/tone: "\${vibe}"

****Visual Style:****

- Follow a \${styleGuide.style_name} aesthetic.
- Use themes related to: \${styleGuide.themes.join(", ")}
- Subject style: \${styleGuide.subjects.style} with \${styleGuide.subjects.details}.
- The illustration should feel imaginative, creative, and distinctive.

****Color Palette:****

- Primary: \${styleGuide.color_palette.primary}
- Secondary: \${styleGuide.color_palette.secondary}
- Highlight: \${styleGuide.color_palette.highlight}
- Shadow: \${styleGuide.color_palette.shadow}
- Background: \${styleGuide.color_palette.background}
- Use these colors thoughtfully to convey mood and meaning.

****Technical Requirements:****

- Create a high-quality illustration with professional brand asset aesthetic.
- It should be memorable and unique, distinct from generic stock imagery.
- The final illustration should feel custom-made for this specific brand.

`;

```
const result = await openai.images.generate({
  model: "gpt-image-1",
  prompt: illustrationPrompt,
  n: 1,
  size: "1024x1024",
  quality: "medium"
});
```

```
const base64Image = result.data?.[0].b64_json;
```

```
return `data:image/png;base64,${base64Image}`;  
}
```

The above will generate a brand illustration that abstractly represents the concept of enhanced vision and bright adventures. The AI might create overlapping lens shapes with light effects, abstract sun rays filtering through geometric forms, or a conceptual landscape viewed through a prism of colors.



Piecing It All Together

With all five images generated, we now have the raw materials for a complete brand kit. But dropping these assets onto a page isn't enough. The magic happens in how we surface and present them as a cohesive, professional package that feels greater than the sum of its parts.

We'll have a page component responsible in surfacing the entire brand kit interface:

```
export function BrandResults() {
  const {
    imageUrl,
    heroImageUrl,
    adImageUrl,
    secondAdImageUrl,
    brandIllustrationUrl,
    styleGuide,
    brandName,
  } = useLoaderData();

  const { color_palette, typography, taglines } =
    styleGuide;

  return (
    <div className="brand-kit-container">
      { /* Brand Assets Grid */ }
      <div className="grid grid-cols-12 gap-5">
        { /* Logo Card - Full Width */ }
        <BrandCard
          backgroundColor={color_palette.primary}
          className="col-span-full"
        >
          <img
            src={imageUrl}
            alt={` ${brandName} logo` }
          />
        </BrandCard>

        { /* Color Palette Section */ }
        <ColorPalette
          styleGuide={styleGuide}
          onColorSwatchClick={
            handleColorSwatchDownload
          }
        />
      </div>
    </div>
  );
}
```

```

/>

{/* Typography Card */}
<TypographyCard
  primaryFont={typography.primary_font}
  backgroundColor={color_palette.highlight}
/>

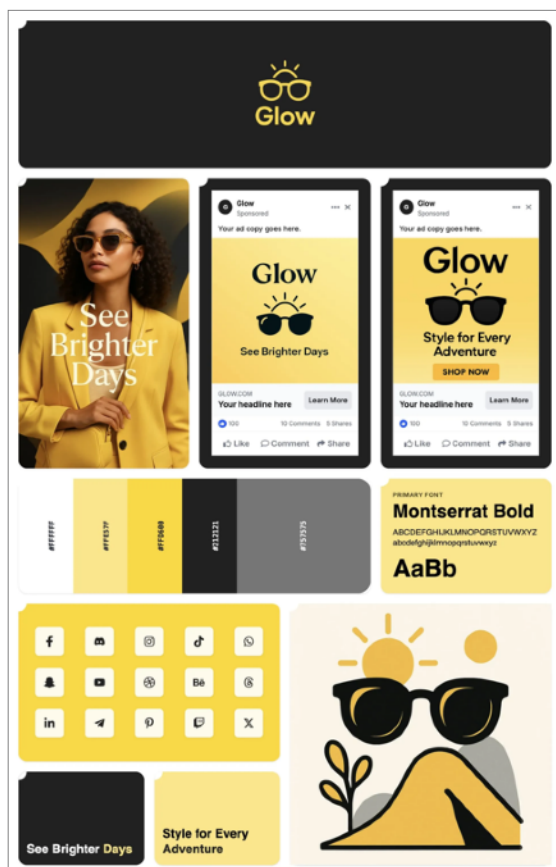
{/* Taglines Cards */}
<TaglineCard
  tagline={taglines[0]}
  palette={color_palette}
/>
<TaglineCard
  tagline={taglines[1]}
  palette={color_palette}
/>

{/* Social Icons Grid */}
<SocialMediaIcons palette={color_palette} />
</div>
</div>
);
}

```

The presentation layer adds value through dynamic color integration (every element uses the generated palette), typography, interactive swatches (colors can be downloaded individually), pre-styled social icons, and visual tagline cards.

This transforms five separate AI outputs into a unified brand system that feels intentionally crafted, not just generated.



Download Capabilities

A brand kit isn't truly useful until users can take it with them. We implemented three download options, each serving different needs:

Individual Downloads

Each asset has a hover-activated download button. Users can grab just the logo or a specific ad without downloading everything. This uses a simple fetch-and-download pattern, converting the base64 images to downloadable blobs.

ZIP Archive

Using [JSZip](#), we bundle everything into organized folders. Beyond the raw images, we generate PNG color swatches for each palette color and include a JSON file with typography and tagline information. The ZIP structure looks like:

```
BrandName-Brand-Assets.zip
├── images/
│   ├── BrandName-Logo.png
│   ├── BrandName-Hero.png
│   └── ...
├── colors/
│   ├── BrandName-Primary-Color.png
│   ├── BrandName-Secondary-Color.png
│   └── ...
├── typography_info.json
└── README.txt
```

PDF Brand Guide

The most comprehensive option generates a multi-page PDF using [jsPDF](#). Here's a simplified pseudo-code structure (the actual implementation involves more careful coordinate calculations, text measurement, and page flow logic):

```

async function generateBrandKitPDF(
  styleGuide,
  brandName,
  images,
) {
  const pdf = new jsPDF({
    orientation: "portrait",
    format: "a4",
  });

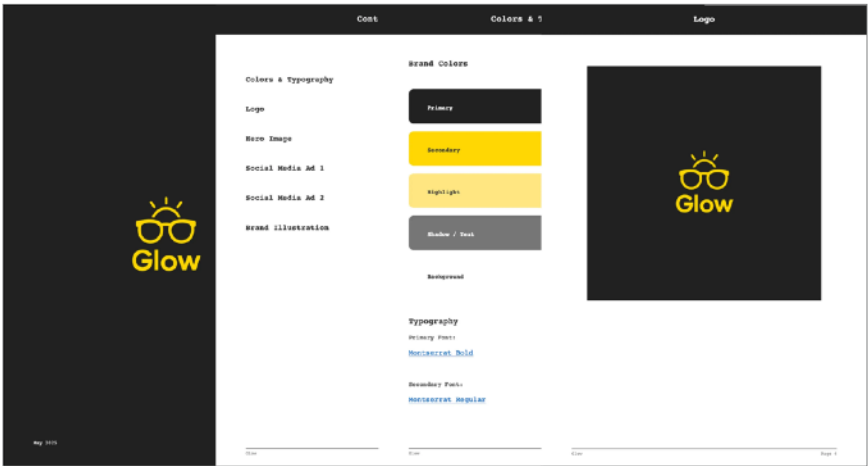
  // Cover page with logo on primary color background
  pdf.setFillColor(styleGuide.color_palette.primary);
  pdf.rect(0, 0, width, height, "F");
  pdf.drawImage(images.logo, centerX, centerY);

  // Table of contents
  pdf.addPage();
  tocItems.forEach((item) => {
    pdf.text(item.title, leftMargin, y);
    pdf.text(item.pageNumber, rightMargin, y);
  });

  // Colors & Typography page
  pdf.addPage();
  // Render color swatches with hex values
  colors.forEach((color) => {
    pdf.setFillColor(color.hex);
    pdf.rect(x, y, swatchWidth, swatchHeight, "F");
    pdf.text(
      color.name + " " + color.hex,
      x + padding,
      y + center,
    );
  });
}

```

The resulting PDF is a professional brand guide with proper page headers, footers, and navigation.



These download options allow users to leave with professional assets they can immediately use in their projects, whether they need a quick logo for social media or a complete brand guide for their design team.

Wrap Up

BrandInAMinute was a simple weekend project that transforms a simple brand description into a complete visual identity. By orchestrating six AI API calls (one for the style guide and five for visual assets), we create logos, color palettes, typography recommendations, hero images, social ads, and brand illustrations that all work together as a cohesive system. The entire process takes under a minute and delivers assets users can download individually, as a ZIP archive, or as a polished PDF brand guide.








Keeping It Simple

One deliberate choice we made was avoiding user accounts and authentication entirely. No sign-ups, no passwords, no email verification flows. Users land on the site, fill out a form, make a payment, and get their brand kit.

We also kept the tech stack straightforward: React, TypeScript, and Tailwind for the front-end, OpenAI's APIs for the heavy lifting, and Stripe for payments. No complex state management, no unnecessary abstractions. Just the code needed to wire everything together.

What We Didn't Cover

There are a few implementation details we glossed over in this case study. The Stripe integration, for instance, adds a payment step between form submission and brand generation. We used [React Stripe.js](#) to embed payment forms directly in a modal, keeping users on our site throughout the entire flow. We even managed to get a dozen or so sales right out of the box without having to market the app much yet. Just sharing it with a few friends and posting it on our socials was enough to validate that people are willing to pay for this kind of instant brand generation.

<input type="checkbox"/>	Amount		Payment method	Description	Customer	Date
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			
<input type="checkbox"/>	\$2.99	USD	Succeeded ✓			

We also didn't dive into error handling, retry logic for failed API calls, or the caching layer we added to avoid regenerating assets when users refresh the page.

Where to Go From Here

If time allows, BrandInAMinute could easily grow to include more brand assets like favicons for websites, app icons for iOS and Android, email signature templates, or business card designs. Each addition is just another API call with a well-crafted prompt.

The style guide generation could also become interactive, letting users tweak colors, adjust themes, or regenerate specific elements they're not happy with. Instead of a one-shot generation, imagine a collaborative flow where users guide the AI toward their vision through iterative refinement.

We could also let users generate variations. Not happy with the first hero image? Generate five more. Want to see the logo with different color combinations?

There's potential for industry-specific templates, too. A restaurant might need menu designs and storefront mockups. A SaaS startup might want product screenshots with their branding applied. An e-commerce brand might need packaging concepts. Each vertical could have its own specialized generation pipeline.

The core insight remains the same: in a world where powerful AI capabilities are just an API call away, the opportunity isn't in building better models. It's in thoughtfully combining existing capabilities to solve real problems. BrandInAMinute does one thing well, but it's just the beginning of what's possible when you put AI at the helm.