Architecture Options for Image Approval App

This document outlines various architectural options for building an image approval app. The app is designed to manage Midjourney-generated images and track their metadata and approval status using Airtable.

Key Functional Requirements

- • Display images from Midjourney
- • Fetch/store metadata in Airtable (e.g. prompt, creator, timestamp)
- • Allow approval actions (e.g. Approve / Reject / Comment)
- • Record approval status in Airtable
- • Optional: Login/authentication, notifications, multi-user roles

Architecture Options

1. No-Code / Low-Code Stack (Fastest to Deploy)

Great for MVPs, non-developers, or internal tools.

Frontend/UI: Glide, Softr, Stacker, or Retool
Backend: Airtable
Automation: Airtable Automations, Zapier, Make.com, or n8n
Image Hosting: Cloudinary, Airtable attachments, or public Discord CDN

Pros:	Cons:
• Extremely fast setup • UI templates for Airtable • Minimal coding	• Limited customization • Performance bottlenecks at scale • May not handle granular roles/permissions well

2. Jamstack App (Modern Web App w/ Static Frontend)

Great for performance, flexibility, and custom UI.

Frontend: Next.js or Astro with Tailwind CSS
Backend: Serverless functions via Vercel, Netlify, or AWS Lambda
Data Layer: Airtable API
Auth: Clerk, Auth0, or Firebase Auth
Image Hosting: Midjourney URL or Cloudinary

Pros:
• Full control over UI/UX
• Can scale to production
• Serverless = cheap + fast

Cons:
• Requires dev expertise
• Slight learning curve on Airtable API & rate limits

3. Full-Stack App with Express or Flask (Traditional Approach)

Great if you want tight backend control or to migrate away from Airtable later.

Frontend: React / Vue
Backend: Node.js (Express) or Python (Flask/FastAPI)
Database: Airtable via REST API (can later migrate to PostgreSQL)
Image Hosting: Midjourney URL or Cloudinary/S3
Auth: JWT or third-party service

Pros:
• Ultimate flexibility
• Easily portable backend logic
• Custom validation, rate limiting, caching

Cons:
• Slower to build
• Must host and maintain backend

4. ChatGPT Plugin or AI Agent Interface

Great for reviewing images conversationally or if part of an AI workflow.

Frontend: Chat UI (e.g. custom ChatGPT plugin or Streamlit app)
Backend: n8n or custom API that interfaces with Airtable
Image UI: Embed thumbnails with approve/reject buttons inline
Use case: 'Show me all unapproved images from this week' → clickable approval actions

Pros:
• Highly interactive
• Perfect for reviewers using AI already

Cons:
• Niche use case
• Limited UI customization

Integration Considerations

• Midjourney: Use Discord bot → webhook + upload to Airtable or Cloudinary
• Airtable API: REST API / Airtable JS SDK / n8n or Make
• Image Uploads: Use Airtable Attachments / Cloudinary URLs
• Approvals: Checkbox / Select field in Airtable
• Notifications: Slack, Email (via Zapier / n8n)
• Automation: n8n, Zapier, Make

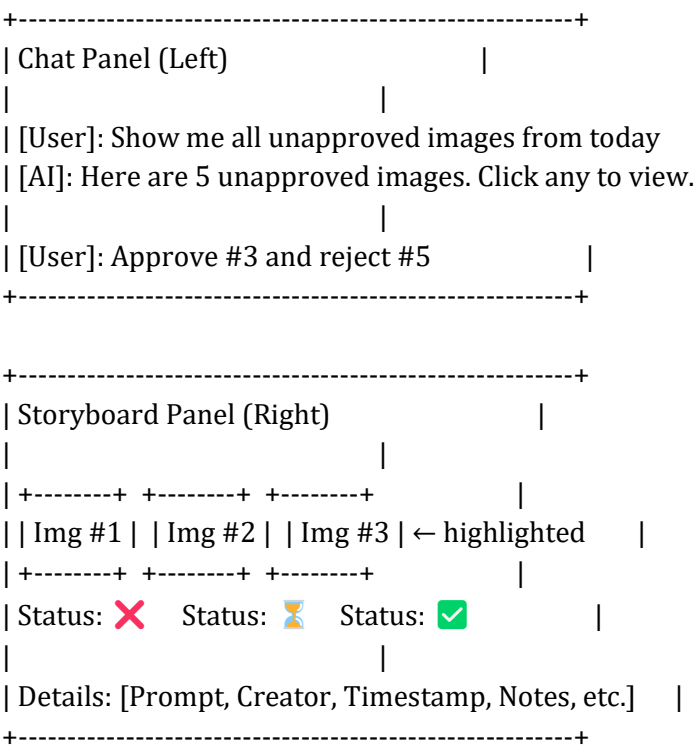
Recommendation Matrix

Priority: MVP / Prototype → Best Option: No-code (Softr, Stacker) → Why: Fastest to launch
Priority: Custom UI/UX → Best Option: Jamstack (Next.js + Airtable) → Why: Balance between speed and control
Priority: Full control → Best Option: React + Express + Airtable → Why: Best for scaling or migration
Priority: AI Integration → Best Option: GPT Plugin + n8n Workflow → Why: Great for conversational review

Combining Chat Interface with Storyboard UI

💡 Concept Overview

Imagine a layout like this:



🧱 Architecture Design Options

Option 1: React Frontend with Chat + Storyboard Layout

Tech Stack:
- Frontend: React + TailwindCSS or Chakra UI
- Chat UI: Custom component with GPT-style logic
- Storyboard UI: Masonry or grid layout showing image cards
- Backend: Node.js / Express (or use n8n to simplify)
- Database: Airtable API

Layout Strategy:
Grid templateColumns="1fr 3fr":
Box (Chat Interface)
Box (Storyboard Grid)

Option 2: Web App with WebSocket or Polling for Real-Time Sync

Real-Time Updates: Use WebSockets (e.g., with Socket.io) or polling to auto-refresh the storyboard when approvals are submitted through the chat.
Chat Input Triggers: When user types “Approve #2,” trigger an action that updates Airtable and re-renders the storyboard panel.

Option 3: Retool or Bubble.io (Low-Code Hybrid UI)

Embed Chat Component: Use OpenAI's Chat API or an iframe-style chat inside a side panel.
Embed Airtable View or Storyboard Grid: Use a visual builder to create the image tiles with interactive elements.
Button Actions: "Approve" and "Reject" buttons per image call JS scripts that update Airtable and post back to the chat feed.

🎯 Key Features to Implement

Feature	Chat Side	Storyboard Side
Text Command Parsing	✅	
Inline Button Interactions	✅	✅
Image Metadata Display		✅
Action Feedback	✅	✅
Search / Filter / Pagination	✅	✅

📦 Suggested Libraries and Tools

- Chat Interface: react-chat-ui, react-simple-chatbot, or custom
- State Management: React Context API or Zustand
- Airtable Integration: airtable JS SDK
- Image Grid UI: react-masonry-css, framer-motion
- Markdown Rendering: react-markdown
- Natural Language Parsing: OpenAI Function Calling or regex parse
- Real-Time Backend: Socket.io, Supabase, or Firebase

🚀 Example Interaction

User Types in Chat:
"Reject the image where the woman is holding the red umbrella."

AI Responds:
"Got it. Rejected image #3. Status updated."

Storyboard Panel:
- Image #3's tile shows ✅ Rejected
- Light animation confirms update

🧩 How to Glue It All Together

Frontend Logic Flow:
1. User types into chat box.
2. Parse text → recognize intent (approve/reject/image ID).
3. Trigger Airtable update.
4. Trigger UI update on storyboard side.
5. AI gives chat feedback confirming change.