ChatGPT

AI Food Video Generator – Technology Stack

The mobile app is built in **React Native** (Android-first) and bundled with all features (user and admin) in one app. Backend logic and APIs run on **Vercel** (serverless Node/Edge functions) and a **Supabase** backend (PostgreSQL + Auth + Storage). This stack leverages free tiers and open-source tools to minimize cost (e.g. Supabase free plan, Vercel Hobby plan, Stripe pay-as-you-go) while covering all required features. Below is a structured breakdown by function:

Client (React Native Android app)

- **Framework:** React Native (can use **Expo** or bare RN). UI built from the provided design system (custom components or UI libraries like React Native Paper, RNEUI, TailwindCSS for RN, etc.).
- **Navigation & State:** React Navigation for screen flow; state via Context/Redux/MobX/Zustand as needed.
- **Image Upload/Validation:** Use libraries like `react-native-image-picker` or `react-native-camera` to let users take/upload photos. Perform basic validation on-device (check resolution, format). For content moderation (ensure food images), an optional step is calling a vision API (e.g. Google Cloud Vision SafeSearch) or a simple on-device model; for cost-saving, initial version may rely on user trust or community reporting. After capture, upload images directly to cloud storage (see **Storage**).
- **Preset Selection:** Predefined "presets" (e.g. video styles, filters, voiceover selections) can be hardcoded or loaded from backend. The UI will allow selecting a recipe or style preset. These presets (configurations for video generation) are stored in the database and sent to the backend to guide the AI/video engine.
- **Authentication UI:** Use @supabase/supabase-js on React Native to integrate **Supabase Auth** (email/password, social logins). Supabase's React Native quickstart shows how to set up Auth with AsyncStorage persistence [1]. On sign-in, store the session JWT locally (via AsyncStorage) and include it in API calls.
- **Payment UI:** For subscriptions, integrate Stripe's React Native SDK (`@stripe/stripe-react-native`). Users can enter payment info or subscribe to plans. The app calls backend endpoints to create Stripe Checkout Sessions or manage subscriptions.

Authentication & User Management

- **Supabase Auth:** Handles sign-up/sign-in, password reset, and session management. Supabase supports email, OAuth (Google/Apple), etc. User records (auth.users) store basic info; a separate `profiles` table can hold extended data (e.g. role, preferences).
- **Roles:** Implement a `role` column (e.g. "user" vs "admin") in the user profile table. Use Supabase Row-Level Security (RLS) policies or backend checks to restrict admin-only operations. The React Native app UI hides admin screens unless the signed-in user has an "admin" role.
- **Session Management:** Supabase auto-refreshes tokens; use the Auth change listeners to maintain logged-in state.

Backend/API (Vercel serverless / Supabase Edge)

- **API Hosting:** Use Vercel (Hobby tier) to host Node.js serverless functions (Next.js API routes or standalone Lambdas). Vercel Hobby plan is free and includes up to 1,000,000 function invocations and 4 CPU-hours per month [2], which is sufficient for modest usage. Alternatively, Supabase Edge Functions (Deno) could be used for some endpoints (also free tier-friendly).
- **Database Access:** Use `@supabase/supabase-js` or REST to query the Supabase Postgres database. The backend functions handle business logic (auth checks, reading/writing user data, subscription status, etc.).
- **Payment Webhooks:** Implement Vercel endpoints to receive Stripe webhook events (e.g. `checkout.session.completed`, subscription updates). These update user subscription status in the DB. Vercel supports long-running webhooks (60s on Hobby).
- **Presets & Admin APIs:** Expose APIs for admins to create/update video presets, view all users/videos, etc. These check the user's role token before allowing actions.

Database & Storage (Supabase)

- **Database (PostgreSQL):** Supabase provides a hosted Postgres DB. Use tables for **users**, **profiles** (extending auth), **videos** (metadata: user, status, preset, storage path, timestamps), **presets**, and **subscriptions**. RLS policies enforce data access (e.g. users see only their own videos; admins can see all). Supabase's free plan includes 500MB DB and 50k MAU [3], and scales if needed.
- **Storage (Files):** Use Supabase Storage buckets for user-uploaded images and generated videos. The free tier includes 1GB storage [3]. Store raw uploads (e.g. `avatars/` or `uploads/food.jpg`) and final videos (e.g. `videos/user123_456.mp4`). If project scales beyond 1GB, consider an S3-compatible service (e.g. DigitalOcean Spaces or Backblaze B2) via Supabase's S3 integration.
- **File Serving:** Final video files can be delivered via signed URLs from Supabase Storage. Use Supabase's client or the Storage API in backend to generate a time-limited download link for users.

Payment & Subscription (Stripe)

- **Stripe Integration:** Use Stripe Billing for subscriptions. On the client, call backend to initiate Stripe Checkout or subscription sessions. On the backend, use `stripe-node` to create subscriptions, payment intents, etc.
- **Billing Model:** For example, a recurring monthly plan unlocks unlimited or X videos. Test mode has no fees; production usage is pay-as-you-go (Stripe takes standard transaction fees). No upfront cost: Stripe only charges per transaction.
- **Data Sync:** Listen for Stripe webhooks on Vercel to update the user's subscription status (active/expired) in Supabase. Only allow video generation if the user has an active subscription.

AI Video Generation (Google Gemini Veo 3.x)

- **GenAI API (Gemini/Veo 3.1):** Use Google's Gemini Video model (Veo 3.1) to generate the core food video. Veo 3.1 produces **8-second, up to 4K video clips with native audio** [4]. It supports text prompts, portrait mode, frame-specific generation, and image-based guidance (up to 3 reference images) [4] [5]. For example, send a prompt like "Close-up of a chef plating spaghetti, cinematic lighting". The API returns a URL to the generated video.
- **Implementation:** Call the Gemini API from a backend function (e.g. Vercel or Google Cloud Function). Use the official Node JS library (`@google/genai` shown in Google's docs [6]) or

HTTP REST. Supply the user's uploaded food image as a reference (to ensure the video matches the actual dish) and use preset templates for text prompts. For cost optimization, only invoke Veo when needed, and reuse Free-tier credits if any. (Note: access requires Google AI Pro/Ultra plan.)

- **Audio and Style:** Gemini natively generates background audio (e.g. cooking sounds, narration) as part of the video. For custom music/voiceovers, the app can allow adding additional audio tracks via Remotion.
- **Result Handling:** The Gemini call is asynchronous. Backend logic should poll or use operation IDs to retrieve the video file URL once ready (as shown in the Google example [6]). Store the video file (or its URL) in Supabase Storage for further processing.

Video Assembly & Processing (Remotion)

- **Remotion Framework:** Remotion is a React-based video rendering library (Node.js) for programmatic video composition [7]. We use Remotion to **stitch together** the AI-generated clip with overlays, intros/outros, text, and optional static images or animations. For example: add a title card with the recipe name, overlay subtitles or bullet points, and append an outro logo.
- **Rendering Environment:** Remotion runs on Node, ideally in a serverless compute. Options include **AWS Lambda** (with the `@remotion/lambda` package) or Google Cloud Run. AWS Lambda fits well (free tier: 1M free invocations & 400k GB-seconds/month [8]). Remotion's docs note Lambda as the fastest way to render in cloud [9]. Each video generation task triggers a Remotion render job (e.g. via Lambda or a Docker container).
- **Rendering Flow:**
- **Prepare Assets:** Collect the AI video clip URL, plus any preset images/audio (stored in Storage).
- **Invoke Remotion:** Call a Lambda function (or container) that loads a React video composition (custom code) using these assets as props. Remotion's Node APIs render and encode an MP4. (See Remotion SSR docs.)
- **Output:** The rendered MP4 is saved to Supabase Storage (e.g. `videos/`) via the storage SDK or by uploading to a signed URL.
- **Retry Logic:** Wrap rendering jobs with a queue (e.g. AWS SQS + Lambda, or a Redis-based worker with exponential backoff). On failure, retry the Remotion job. Keep a "status" field in the DB (`pending` → `processing` → `completed`/`failed`).

Task Queue & Background Processing

- **Queue System:** To handle asynchronous video jobs, use a lightweight queue. Options: Redis queue (e.g. **BullMQ** on a small VM or Supabase's Redis add-on), or cloud queues like AWS SQS (free tier: 1M requests) or Google Cloud Tasks. Each new generation request enqueues a job: Gemini call → Remotion render → notify user.
- **Worker:** A backend process (serverless function or small container) pulls jobs. For example, a Vercel Cron (or webhook) could check for `pending` jobs and spawn Lambda renders. Alternatively, use a Firebase/Google PubSub pattern (push to subscription). Use library features to retry on error.

Notifications (Optional Enhancements)

- **In-App Status:** Show the user a "Rendering…" status screen. Poll backend or use WebSockets (Supabase Realtime) to update progress (e.g. "AI generation done, stitching video…").
- **Push Notifications:** For mobile-friendly UX, implement push (e.g. Firebase Cloud Messaging) to notify when video is ready. The backend (after upload) can send a notification token via FCM. This is optional and can use Firebase free tier.

- **Email:** If desired, use Supabase's SMTP/SMTP Relay or a service like SendGrid (free tier) to email users when videos are ready.

Admin Features (in-app)

- **Unified App:** Admin views are just special screens within the same React Native app. Conditional rendering shows admin dashboard items if the logged-in user's role is admin. No separate web dashboard is built.
- **Functionality:** Admins can upload new presets (video templates, background music tracks, recipe data), view user activity, and manage subscriptions. These use the same APIs, gated by role checks.
- **Implementation:** On the backend, protect admin endpoints with a middleware checking `user.role === 'admin'`. Use Supabase RLS or JWT claims for extra security.

Deployment & Infrastructure

- **Vercel:** Deploy all frontend web assets (if any) and API routes to Vercel (Hobby plan). Connect the GitHub repo for automatic CI/CD. Vercel provides HTTPS, global CDN, and function hosting (1M invokes/mo free [2]). Serverless functions run Node/TypeScript. For heavier tasks, Vercel can trigger external services (see next point).
- **Supabase Project:** Host the Postgres DB, Auth, and Storage on a Supabase project (start on Free). Configure storage buckets and database schema via the Dashboard. Use environment variables in Vercel to connect (SUPABASE_URL, KEY, etc.). Supabase automatically handles scaling and backups on paid plans.
- **Cloud Compute:** For the AI and video tasks: use **Google Cloud Functions/Vertex AI** for Gemini calls (since it's a Google API), and **AWS Lambda** for Remotion as noted. This hybrid approach avoids vendor lock-in on one cloud and uses each service's strengths. Both have generous free tiers [8].
- **Monitoring & Logging:** Use Vercel's built-in logs for API errors. For background jobs (Lambda/Cloud Functions), monitor via their consoles.
- **CDN / Edge Functions:** Leverage Vercel Edge Functions (JS/V8) for latency-sensitive APIs (e.g. fetching user data), and Node Functions for heavier logic. Edge functions scale automatically.

Cost Optimization

- **Open Source & Free Tiers:** Supabase (free plan: 50k MAU, 1GB storage, 500MB DB) [3] and Vercel Hobby (1M invokes, 4 CPU-hrs) [2] provide zero-cost baselines. Remotion and React Native are OSS (MIT license) with no runtime cost. Stripe charges per transaction only (no subscription overhead).
- **On-Demand Compute:** Heavy tasks (AI inference, video rendering) run on on-demand servers (AWS/GCP) with free quotas [8], ensuring we only pay when used.
- **Avoid Lock-In:** Using Supabase (Postgres) and open-stack tools (React, Node, Stripe) keeps options flexible. If usage outgrows free quotas, these services allow pay-as-you-go scaling.

Data Flow Summary

1. **User** logs in (Supabase Auth) and uploads a food photo (to Supabase Storage).
2. **Client** sends a video generation request with user inputs (photo, preset choices).
3. **Backend** enqueues a job:
4. Calls **Gemini Veo 3.1** API to generate a short video clip [4].
5. Once received, triggers **Remotion** to stitch the clip with templates (using Lambda).

6. **Storage:** The final video MP4 is saved to Supabase Storage.
7. **Notification:** App polls status or receives a push; when done, user can download or share the video from the app.

Each tech choice above directly supports a feature from the PRD (auth, upload, preset-driven generation, subscriptions via Stripe, admin roles, video generation via AI, and final download delivery) while emphasizing cost-efficiency and modern developer tooling.

Sources: Relevant documentation on Supabase pricing [3], Vercel Hobby plan [2], Google Gemini API (Veo 3.1) [4], and Remotion's Node cloud rendering [7], as well as AWS Lambda free tier [8]. These guide the choice of free-tier-friendly and compatible services.

---

[1] Use Supabase Auth with React Native | Supabase Docs
https://supabase.com/docs/guides/auth/quickstarts/react-native

[2] Vercel Hobby Plan
https://vercel.com/docs/plans/hobby

[3] Pricing & Fees | Supabase
https://supabase.com/pricing

[4] [5] [6] Generate videos with Veo 3.1 in Gemini API | Google AI for Developers
https://ai.google.dev/gemini-api/docs/video

[7] [9] Server-Side Rendering | Remotion | Make videos programmatically
https://www.remotion.dev/docs/ssr

[8] Serverless Computing Service - Free AWS Lambda - AWS
https://aws.amazon.com/pm/lambda/