Great instincts. You 100% want the AI call **server-side**, not from the mobile app. Here’s a tight, production-ready pattern that keeps costs and abuse under control while staying simple to build now and scale later.

**Recommended architecture (Phase 1+)**

**Mobile (RN/Expo)** → **API Gateway (Cloudflare/NGINX)** → **App Server (Node/TS)** → **Model API**  
**DB**: Postgres (Supabase) for users, prefs, plan cache, usage; **Redis** for rate-limits/queues.  
**Billing/Entitlements**: RevenueCat webhooks → your server → user entitlement table.

**Why server-side only**

* **Key protection**: model API keys never ship in the app.
* **Abuse control**: you enforce auth, entitlements, rate limits, quotas, and budget “circuit breakers.”
* **Cost control**: caching, batching, and fallbacks live on your server.
* **Observability**: central logs for prompts, outputs (redacted), and cost per request.

**Minimal flow**

1. App collects preferences → **POST** /plans/generate with mealsWanted + prefs.
2. Server authenticates (JWT) + checks **RevenueCat** entitlement.
3. Server enforces **rate/quotas/budget guards**.
4. Server calls the **model API**, composes the plan, normalizes ingredients, and **maps to product URLs/SKUs**.
5. Server stores and returns {planId, meals[], items[]}.
6. App displays plan → user can export or handoff to Woolies basket.

**Abuse & cost-control guardrails (copy/paste checklist)**

**Authentication & gating**

* Require **OIDC JWT** from your auth provider on every request.
* Verify **RevenueCat entitlement** server-side (cache + webhook updates).

**Per-user controls**

* **Token bucket** (Redis): e.g., 10 plan generations/day, 2 concurrent.
* **Quota** per subscription tier (month caps).
* **Idempotency**: accept Idempotency-Key to avoid duplicate model calls on retries.

**Budget safety**

* **Org budget caps**: refuse when monthly model spend > threshold.
* **Circuit breaker**: auto-fallback to cached/cheaper model if latency/cost spikes.

**Abuse & bot signals**

* **Device attestation**: Apple App Attest / DeviceCheck; Android Play Integrity.
* **Request signing**: app sends (payload, timestamp) with an **ephemeral HMAC** you issued from your server (valid 60s) to prevent replay from scraped apps.
* **IP reputation** (via gateway); optional **Turnstile**/reCAPTCHA for suspicious spikes.

**Caching**

* Hash (prefs, mealsWanted, catalogVersion) → if seen in last N days, **return cached plan** (huge savings).
* Also cache **ingredient→SKU mapping** results.

**Prompt/output hygiene**

* Strip PII from prompts; cap token sizes; validate output schema; clamp counts/units to sane bounds.

**Simple API contract**

**POST** /plans/generate

{

"mealsWanted": {"breakfasts":4, "lunches":5, "dinners":6},

"prefs": {

"diet": ["vegetarian"],

"exclusions": ["peanuts"],

"timePerMeal": "≤30m",

"budget": "medium",

"householdSize": 3,

"units": "metric",

"locale": "en-ZA"

}

}

**200 OK**

{

"planId":"plan\_123",

"meals":[{"slot":"Mon D","recipeId":"rec\_42","title":"Chickpea Curry", "image":"..."}],

"items":[{"title":"WW Basmati Rice 1kg","qty":1,"unit":"bag","url":"...","sku":"...","category":"Grains"}],

"costEstZAR": 742.50

}

**Server skeleton (Node/TS pseudo)**

// Express-style

app.post("/plans/generate", authJWT, requireEntitlement("pro"), idempotency, rateLimit, async (req,res) => {

const input = validate(req.body); // zod/valibot

const cacheKey = hash(input, CATALOG\_VERSION);

const cached = await redis.get(cacheKey);

if (cached) return res.json(JSON.parse(cached));

guardMonthlyBudget(); // throws if exceeded

throttleConcurrent(req.user.id); // e.g., max 2 in-flight

const prompt = buildPrompt(input); // deterministic, compact

const modelResp = await callModel(prompt, { maxTokens: 1200, timeoutMs: 20\_000 });

const plan = parseAndValidate(modelResp); // schema -> meals[], items[]

const mapped = await mapToRetailerSKUs(plan.items); // uses your canonical mapping table

const result = { planId: uid(), ...plan, items: mapped };

await db.savePlan(req.user.id, result);

await redis.set(cacheKey, JSON.stringify(result), "EX", 60\*60\*24\*7); // 7 days

res.json(result);

});

**Mobile → server security notes**

* Mobile **never** talks to the model vendor; only to **your** API.
* Use **short-lived access tokens** (e.g., 60 min) and refresh flow.
* Add an **app-bound nonce** (ephemeral HMAC) to sensitive endpoints to make replay from non-official clients harder.
* Persist an **anonymous device ID** to correlate abuse without storing PII.

**Where to run it**

* **Serverless** (Vercel/Cloud Functions/Fly): perfect to start—fast to ship, easy scaling. Keep **Redis** for rate-limits/queues and **Postgres** for plans.
* Put **Cloudflare** in front for WAF + DoS protection + caching of static assets.

**Ops you’ll appreciate later**

* **Cost meter**: record estimated tokens/cost per request; nightly report to Slack/email.
* **A/B model routing**: toggle between cheaper/faster models via feature flag.
* **Replay harness**: save anonymized prompts/outputs to re-run against new models for quality without spending live traffic.

**TL;DR**

* Do **all AI selection on your server**.
* Gate with **auth + entitlements**, enforce **rate/quotas**, and add **budget circuit breakers**.
* Save money with **caching** and SKU-mapping reuse.
* Keep mobile thin; treat the model like a behind-the-scenes service.

If you want, I can drop a ready-to-paste **/plans/generate** endpoint (Express + Zod + Redis guard + Supabase JWT verify + RevenueCat webhook consumer) into the canvas to hand straight to your Codex agent.

Top of Form

Thinking

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.