## **Privacy‑first architecture that still scales**

You can absolutely design CleanList to be privacy‑first without sacrificing speed or scale. The key is to make privacy an architectural property (not a feature) and pair it with proven scalability patterns.

### **Core principles**

* **Data minimization:** Only collect what you need for core features. Default to “no PII by default,” and make every new field justify itself with a user value statement.
* **End‑to‑end encryption:** Encrypt in transit (TLS 1.3) and at rest (AES‑256). Use application‑level encryption for any high‑sensitivity fields so the DB only ever stores ciphertext.
* **Zero‑trust access:** No implicit trust inside your VPC. Enforce mutual TLS between services, short‑lived tokens, and least‑privilege IAM roles for every component.
* **Privacy by design:** Treat privacy reviews like schema migrations: required, versioned, and reversible (via data deletion hooks and retention policies).
* **Observability with restraint:** Collect operational telemetry that avoids user identifiers unless strictly necessary; prefer aggregation and on‑device pre‑processing where possible2.

### **Reference architecture**

* **Client layer:**
  + **Sensitive prefs on device:** Store allergy flags/diet prefs locally; send only derived, non‑identifying filters when possible.
  + **Selective sync:** If user opts in to cloud sync, encrypt preferences client‑side with a user‑controlled key (or platform KMS–wrapped key).
* **API and services:**
  + **Thin API gateway:** Rate limiting, JWT verification, schema validation, and request scrubbing (drop accidental PII).
  + **Service boundaries:** Separate “Accounts,” “Ingredients,” “Catalog,” and “Search” into services so data access is narrowly scoped. This supports independent scaling and least‑privilege data paths.
  + **Privacy guardrails service:** Centralize consent checks, purpose binding, and data‑access decisions so every request proves it has a lawful purpose before touching data.
* **Data layer:**
  + **Primary OLTP DB:** Use row‑level encryption for any quasi‑identifiers. Split identity (email, auth) from activity (queries, lists) into separate schemas or databases to reduce linkage risk.
  + **Search index:** Keep ingest pipelines that strip PII and only index product/ingredient text. No user data in the index.
  + **Analytics:** Stream events through a proxy that hashes or drops user identifiers; default to aggregated metrics (k‑anonymity thresholds) before storage.
* **Security & keys:**
  + **KMS everywhere:** Envelope encryption with a cloud KMS; rotate data keys automatically. Keep a distinct key per dataset (identity, preferences, payments).
  + **Secrets management:** Centralized vault; no secrets in env vars or CI logs. Short TTL tokens for service‑to‑service auth.
  + **Network posture:** Private subnets, egress‑only NAT, WAF in front of the gateway, mutual TLS between services.
* **Scalability:**
  + **Horizontal first:** Stateless services behind an autoscaling layer; externalize session state (Redis with ACLs).
  + **Caching:** Read‑through cache for catalog/ingredient lookups; CDN for static/media; application‑level cache with explicit TTLs for privacy‑safe responses.
  + **Queues and jobs:** Use a queue for slow tasks (ingesting labels, enriching ingredients) to keep P99 latencies low during spikes2.

Sources: 2

### **Concrete stack for CleanList**

* **Frontend:** React/Next.js with on‑device encrypted storage for dietary prefs; send only filter tokens to the backend.
* **Gateway:** FastAPI/Express with OPA (policy engine) or a lightweight custom “consent/purpose” middleware.
* **Services:**
  + Accounts (AuthN/Z, email, OAuth)
  + Catalog (SKUs, brands)
  + Ingredients (parsing/mapping)
  + Search (read‑only, PII‑free)
* **Data:** Postgres (row‑level security; separate schemas), OpenSearch/Meilisearch for product text, Redis for cache/queues.
* **Security:** Cloud KMS, Vault for secrets, mTLS between services, WAF + rate limits.
* **Infra:** IaC (Terraform), containers, autoscaling, blue‑green or canary deploys with automatic rollback.

### **Privacy operations you’ll actually use**

* **Data inventory & classification:** Tag every field by sensitivity; map each to a legal basis and retention window.
* **Consent ledger:** Store immutable records of user consent with purpose/version; evaluate on each request.
* **User rights automation:** Build endpoints for export, correction, and deletion; ensure cascading deletes and cryptographic erasure of encrypted blobs.
* **Logging policy:** Separate security logs (minimal, access‑focused) from app metrics (aggregated). Redact by default; sample sparingly.
* **Third‑party governance:** Route webhooks and vendors through a proxy; sign DPAs; restrict scopes and rotate vendor tokens frequently.

### **Deployment patterns that balance privacy and scale**

* **Multi‑tier isolation:** Dev/stage/prod in separate accounts/projects. No production data in lower envs—use synthetic datasets.
* **Per‑service IAM:** Each microservice gets the minimum database role and KMS key rights it needs—nothing more.
* **Edge where it helps, not where it hurts:** Cache public catalog data at the edge; keep any user‑derived responses out of edge caches unless you use per‑user keys and short TTLs.
* **Resilience:** Health probes, bulkheads, circuit breakers; rate limits at the edge and gateway to survive traffic spikes gracefully2.

### **Getting started in 30 days**

1. **Week 1 — Foundations:** Define data classes, consent model, and PII schema split; set up KMS, Vault, and Terraform.
2. **Week 2 — MVP services:** Ship Accounts, Catalog, Ingredients with policy middleware; implement mTLS and per‑service IAM.
3. **Week 3 — Observability:** Add structured logs (redacted), metrics, and privacy‑safe analytics; integrate queues and caches.
4. **Week 4 — Hardening:** Run threat modeling, chaos drills, backup/restore tests, and data‑deletion rehearsals; add blue‑green deploys.

This gives you privacy as a default, clear blast‑radius boundaries, and headroom to scale without re‑platforming.

If you share your current stack (host, language, DB), I’ll map this into a tailored checklist and Terraform/module picks so you can implement it step by step.