Senior Integration Engineer – Code Test

1) Integration Gateway Implementation

Build a small Integration Gateway (API service) that exposes a stable public API and orchestrates two upstream systems. You can use the backend language of your choice (C#, go, Node, etc.), however specify the reason for choosing the specific language

Business scenario

You are integrating the following services via Gateway API service you are developing:

- **ERP Service** (where authoritative product data resides)
- **Warehouse Service** (where stock/availability of the product can be fetched)

  Note: Treat ERP and Warehouse as **remote HTTP services** (you can stub them).

The gateway service must expose a **stable v1 API** for clients while supporting evolution to **v2** (additive, backward compatible).

Public API (HTTP; language/framework of your choice)

- GET /products — Return a merged list (ERP + stock).
- POST /products — Create/update a product via ERP; ensure idempotency.
- GET /products/{id} — Return merged product.
- PUT /products/{id} — Update via ERP; ensure idempotency.
- DELETE /products/{id} — Delete via ERP (soft delete acceptable).

Notes

- Implement **request timeout**, **retries with exponential backoff + jitter**, and a **circuit breaker** when calling upstreams.
- Writes (POST/PUT) must be **idempotent** using an Idempotency-Key header (dedupe by key + operation + body hash).
- Maintain a simple **in-memory cache** for GET /products with TTL to reduce upstream fan-out (thread-safe).
- **Backward compatibility:** ship v2 by **adding** optional response fields and/or a new endpoint (e.g., /v2/products) **without breaking v1**.

Minimum requirements

- Idiomatic project structure for your language.
- Config via env (no secrets in code).

- Unit tests for at least one read, and one write path (include an idempotency test).

2) Design & Architecture Questions (answers/DESIGN.md)

Brief answers (3–8 sentences each):

1. Your approach to **backward-compatible contract evolution** for APIs and/or events.
2. **Retries, timeouts, and circuit breakers:** your defaults and how you set budgets.
3. **Idempotency** strategies for writes and replay handling.
4. **Observability**: logs/metrics/traces you'd emit to debug slow/failed integrations.
5. Security controls you'd apply (authN/Z at the edge, input validation, **rate limiting**, config/secrets, SSRF/misconfig hardening).
6. Your preferred **framework/tooling** and why for this use case.

5) Security Expectations (Bonus if implemented)

- Choose **JWT** validation or **HMAC signature** for inbound calls; reject missing/invalid tokens/signatures.
- **Input validation** and safe JSON parsing.
- No secrets in code (use env/secret store).

Submission Requirements

- Source code with proper structure (any language). You can share the GitHub repository link with required access
- **Tests** (unit/integration) + instructions to run them.
- **README.md** (how to run gateway, stubs, perf test).
- **answers/DESIGN.md** and **answers/CODEREVIEW.md**.
- **OpenAPI** spec for v1 and v2 to reflect the backward compatible changes you have added in V2.