🚀 Kasparro - Backend & ETL Systems Assignment

This assignment is designed for **ambitious builders** who want to develop production-grade skills early in their career. Modern tools (ChatGPT, Copilot, open-source libraries) make it possible for motivated engineers to deliver high-quality systems quickly — and we encourage you to use them thoughtfully.

Your work here will simulate owning a **small production backend system end-to-end**.

🎯 Tier Structure

P0 — Foundation (Required)

P1 — Growth Layer (Required)

P2 — Differentiator Layer (Optional, Highly Rewarded)

P0 + P1 form the **expected bar**.
P2 is where you **differentiate yourself** from the crowd.

==========

✅ P0 — FOUNDATION LAYER (REQUIRED)

==========

P0.1 — Data Ingestion (Two Sources)

Build an ETL pipeline that ingests data from:

- **1 API source (using the provided API key)**
- **1 CSV source**

Requirements:

- Store raw data into Postgres (`raw_*` tables)
- Normalize into a unified schema
- Type cleaning and validation (Pydantic or similar)
- Implement **incremental ingestion** (no reprocessing old data)
- Handle authentication securely using the provided key

P0.2 — Backend API Service

Expose a lightweight backend service with two endpoints:

`GET /data`

- Pagination
- Filtering
- Returns metadata: `request_id`, `api_latency_ms`

`GET /health`

- Reports DB connectivity
- Reports ETL last-run status

P0.3 — Dockerized, Runnable System

Your system must run entirely via:

```
None
make up
make down
```

```
make test
```

You must provide:

- Dockerfile
- docker-compose.yml
- Makefile
- README with setup + design explanation

The Docker image **must automatically start the ETL service** and **expose API endpoints immediately**.

P0.4 — Minimal Test Suite

Add basic tests covering:

- ETL transformation logic
- At least one API endpoint
- At least one failure scenario

==========

⭐ P1 — GROWTH LAYER (REQUIRED)

==========

P1.1 — Add a Third Data Source

Choose any:

- Another API
- RSS feed
- A second CSV with quirks

Demonstrate proper **schema unification** across all three sources.

P1.2 — Improved Incremental Ingestion

Implement:

- Checkpoint table
- Resume-on-failure logic
- Idempotent writes

P1.3 — `/stats` Endpoint

Expose ETL summaries:

- Records processed
- Duration
- Last success & failure timestamps
- Run metadata

P1.4 — Comprehensive Test Coverage

Your tests must now cover:

- Incremental ingestion
- Failure scenarios
- Schema mismatches
- API endpoints
- Rate limiting logic (if implemented)

P1.5 — Clean Architecture

Organize code with clear separation of concerns (suggested layout):

```
None
ingestion/
```

```
api/
services/
schemas/
core/
tests/
```

==========

🚀 P2 — DIFFERENTIATOR LAYER (OPTIONAL)

==========

P2 is **not required**, but attempting even one item shows exceptional ambition and engineering maturity.

P2.1 — Schema Drift Detection

Implement automatic detection of schema changes using:

- Fuzzy matching
- Confidence scoring
- Warning logs

P2.2 — Failure Injection + Strong Recovery

Introduce a controlled mid-run ETL failure. Your system must:

- Resume cleanly from last checkpoint
- Avoid duplicates
- Record detailed run metadata

P2.3 — Rate Limiting + Backoff

Implement per-source:

- Rate limits
- Retry + exponential backoff
- Logging

P2.4 — Observability Layer

Add any of:

- `/metrics` (Prometheus format)
- Structured JSON logs
- ETL metadata tracking

P2.5 — DevOps Enhancements

Optional but impressive:

- GitHub Actions CI pipeline
- Automatic image publishing
- Docker health checks

P2.6 — Run Comparison / Anomaly Detection

Add endpoints like:

- `/runs?limit=N`
- `/compare-runs`

Identify anomalies across ETL runs.

==========

🧪 FINAL EVALUATION REQUIREMENTS (MANDATORY)

==========

These requirements apply *regardless* of P0/P1/P2 completion.

1. API Access & Authentication

- All ETL + API calls **must use the provided API key**.
- Evaluators will verify:
    - Correct authentication
    - Secure handling (no hard-coded keys)

---

2. Docker Image Submission

You must provide a working Docker image that:

- Automatically starts the ETL service
- Exposes API endpoints immediately
- Runs locally exactly as per instructions

Evaluators will verify your system **without modifying any code**.

---

3. Cloud Deployment (AWS / GCP / Azure)

Your system must be deployed to a cloud provider.

Deployment must include:

- Public API endpoints
- Cloud-based scheduled ETL runs (cron jobs / Cloud Scheduler / EventBridge)

- Logs + metrics visible in cloud console

During evaluation, you will demonstrate:

- Cron execution
- Logs and metrics (from your laptop via cloud dashboard)

---

4. Automated Test Suite

Must cover:

- ETL transformations
- Incremental ingestion
- Failure recovery
- Schema drift (if implemented)
- API endpoints
- Rate limiting logic (if implemented)

All tests must be accurate and reliable.

---

5. Smoke Test (End-to-End Demo)

You must run a live smoke test demonstrating:

- Successful ETL ingestion
- API functionality
- ETL recovery after restart
- Rate limit correctness (if implemented)

---

6. Verification by Evaluators

Evaluators will verify:

- Docker image
- Cloud deployment URL
- Cron job execution
- Logs + metrics in cloud

- ETL resume behavior
- API correctness
- Rate limit adherence

Submissions missing any of the following will be considered **incomplete**:

- Docker image
- Cloud deployment
- Cron setup
- Full test suite
- Successful smoke test

Final Note

P0 and P1 define the expected bar.
P2 is where ambitious builders stand out.

You are encouraged to use modern tools, explore new ideas, and push yourself.
This assignment is not just a test — it is a **real learning experience** that mirrors how engineers work at Kasparro.

Build with curiosity.
Build with clarity. Build to differentiate yourself.