⁂ Distributed Task Queue & Job Processor (Simplified)

Overview

Build a small-scale **distributed job queue and worker system**.\
It should let authenticated users submit tasks, process them with worker nodes, and track progress in real time through a simple dashboard.

The system should be reliable, durable, and observable --- but only to a **prototype level** (about 10 hours of effort).

⁂ Core Requirements

1. Job API

- REST or gRPC API for:
  - Submitting a job (JSON payload)
  - Checking job status (pending, running, done, failed)
- Support optional **idempotency key** to avoid duplicate jobs.

2. Persistence

- Use a small database or file-backed store (e.g., SQLite, Redis, Postgres).
- Jobs should survive restarts.

3. Workers

- Poll jobs from the queue.
- Process them with **lease + ack + retry** logic:
  - Lease: mark job as running.
  - Ack: mark done.
  - Retry: re-queue failed jobs (up to N retries).
- Failed jobs after max retries go to **Dead Letter Queue (DLQ)**.

4. Rate Limits / Quotas

- Enforce a per-tenant (user) limit:
  - e.g., max 5 concurrent jobs.
  - simple rate limiting (e.g., 10 new jobs/minute).

5. Dashboard (Basic UI)

- Responsive web app (React or simple HTML + JS).
- Show:
  - Pending / Running / Completed / Failed jobs
  - DLQ items
- Use WebSockets or polling for live updates.

6. Observability

- Log every major event (submit, start, finish, fail).
- Expose simple metrics (e.g., total jobs, failed jobs, retries).
- Include trace IDs or job IDs in logs.

ﷺ Optional / Conceptual Add-Ons (Stretch)

- Auto-scale workers (document how it would work, not required to implement).
- Include brief notes on design trade-offs in a short README.

✅ Evaluation Focus

Area	What we'll look for
Correctness	Retry / lease / ack logic works properly
Reliability	Jobs persist and recover on restart
API & UX	Clear endpoints and functional UI
Observability	Logs + simple metrics
Code Quality	Structure, comments, simple tests