QueueCTL - Backend Developer Internship Assignment

Tech Stack:

Your Choice - (Python / Go / Node.js / Java)

Submission:

GitHub Repository (Public) + README

🎯 Objective

Build a **CLI-based background job queue system** called `queuectl`.

This system should manage background jobs with worker processes, handle retries using exponential backoff, and maintain a **Dead Letter Queue (DLQ)** for permanently failed jobs.

🧩 Problem Overview

You need to implement a minimal, production-grade job queue system that supports:

- Enqueuing and managing background jobs
- Running multiple worker processes
- Retrying failed jobs automatically with exponential backoff
- Moving jobs to a **Dead Letter Queue** after exhausting retries
- Persistent job storage across restarts
- All operations accessible through a **CLI interface**

📦 Job Specification

Each job must contain at least the following fields:

```
{
  "id": "unique-job-id",
  "command": "echo 'Hello World'",
  "state": "pending",
  "attempts": 0,
  "max_retries": 3,
  "created_at": "2025-11-04T10:30:00Z",
  "updated_at": "2025-11-04T10:30:00Z"
}
```

🔄 Job Lifecycle

State	Description
`pending`	Waiting to be picked up by a worker
`processing`	Currently being executed

`completed`	Successfully executed
`failed`	Failed, but retryable
`dead`	Permanently failed (moved to DLQ)

💻 CLI Commands

Your tool must support the following commands:

Category	Command Example	Description
Enqueue	`queuectl enqueue '{"id":"job1","command":"sleep 2"}'`	**Add a new job to the queue**
Workers	`queuectl worker start --count 3`	**Start one or more workers**
	`queuectl worker stop`	Stop running workers gracefully
Status	`queuectl status`	**Show summary of all job states & active workers**
List Jobs	`queuectl list --state pending`	**List jobs by state**
DLQ	`queuectl dlq list` / `queuectl dlq retry job1`	**View or retry DLQ jobs**
Config	`queuectl config set max-retries 3`	**Manage configuration (retry, backoff, etc.)**

⚙️ System Requirements

Job Execution

- Each worker must execute the specified command (e.g. `sleep 2`, `echo hello`, etc.)
- Exit codes should determine success or failure.
- Commands that fail or are not found should trigger retries.

Retry & Backoff

- Failed jobs retry automatically.
- Implement exponential backoff:
  - `delay = base ^ attempts` seconds
- Move to DLQ after `max_retries`.

Persistence

- Job data must persist across restarts.
- Use file storage (JSON) or SQLite/embedded DB or anything which you think is best for this usecase.

Worker Management

- Multiple workers can process jobs in parallel.
- Prevent duplicate processing (locking required).
- Implement graceful shutdown (finish current job before exit).

Configuration

- Allow configurable retry count and backoff base via CLI.

🧪 Expected Test Scenarios

Candidates are expected to ensure the following:

Basic job completes successfully.
Failed job retries with backoff and moves to DLQ.
Multiple workers process jobs without overlap.
Invalid commands fail gracefully.
Job data survives restart.

📋 Must-Have Deliverables

Your submission **must include**:

✅ Working CLI application (`queuectl`)
✅ Persistent job storage
✅ Multiple worker support
✅ Retry mechanism with exponential backoff
✅ Dead Letter Queue
✅ Configuration management
✅ Clean CLI interface (commands & help texts)
✅ Comprehensive `README.md`
✅ Code structured with clear separation of concerns
✅ At least minimal testing or script to validate core flows

📘 README Expectations

Your `README.md` should cover:

Setup Instructions — How to run locally
Usage Examples — CLI commands with example outputs
Architecture Overview — Job lifecycle, data persistence, worker logic
Assumptions & Trade-offs — Decisions made, any simplifications
Testing Instructions — How to verify functionality

📊 Evaluation Criteria

Criteria	Weight	Description
Functionality	**40%**	**Core features (enqueue, worker, retry, DLQ)**
Code Quality	**20%**	**Structure, readability, maintainability**
Robustness	**20%**	**Handles edge cases and concurrency safely**
Documentation	**10%**	**Clear setup and usage instructions**
Testing	**10%**	**Demonstrates correctness and reliability**

🌟 Bonus Features (Optional)

Extra credit will be given for:

Job timeout handling

Job priority queues
Scheduled/delayed jobs (`run_at`)
Job output logging
Metrics or execution stats
Minimal web dashboard for monitoring

⚠️ Disqualification / Common Mistakes

Missing retry or DLQ functionality
Race conditions or duplicate job execution
Non-persistent data (jobs lost on restart)
Hardcoded configuration values
Unclear or missing README

🗒️ Submission

Push your complete solution to a **public GitHub repository**.
Include a **README.md** as per above.
Record a working cli demo (upload to drive and provide link in README.md)
Optional: include a short **architecture or design.md** file.
Share the repository link for review.

✅ Checklist Before Submission

- [ ] All required commands functional
- [ ] Jobs persist after restart
- [ ] Retry and backoff implemented correctly
- [ ] DLQ operational
- [ ] CLI user-friendly and documented
- [ ] Code is modular and maintainable
- [ ] Includes test or script verifying main flows
