Backend Task: Custom Job Queue System

Custom Job Queue System

Objective:

Design a job queue system that separates the job producer (API) and consumer (worker), with proper retry logic, prioritization, and minimal dependency on libraries like Celery or Bull.

What You Need to Build:

1. Job Producer API (REST)

- A REST endpoint to submit a job request (e.g., `/submit-job`).
- Job schema:

```
json
CopyEdit
{"job_type": "send_email","priority": "high",  // or "low""payload": {"to": "user@example.com","subject":
"Test","message": "Hello!"}}
```

- Jobs should be saved into a **queue (Redis/DB)** in the order of priority (i.e., high-priority jobs are picked before low-priority ones).

2. Job Worker

- A separate process (can be a CLI or thread) that polls the queue, picks the next job, and executes it (mock email sending using `time.sleep()` and `print()`).
- Support retries:
  - Retry failed jobs up to 3 times with exponential backoff.
  - Mark as permanently failed after 3 unsuccessful attempts.

3. Additional Requirements to Discourage Copy-Paste:

4. Expected Output:

- A working REST API to submit jobs.
- **Add a constraint**: Your solution should not use any off-the-shelf task queue system (e.g., no Celery, RQ, Sidekiq).
- Include timestamps for:

    - Created
    - Picked
    - Completed or Failed
- A CLI/worker to consume and process jobs.
- Sample console output or logs showing:
    - Prioritized job execution
    - Retry attempts
    - Final statuses (success/failure)
- **Bonus**: Build a `/jobs/status/:job_id` endpoint to check job status.

You are free to use any programming language or tech stack of your choice to complete the task. We're more interested in understanding your problem-solving approach, code structure, and how you reason through the requirements, rather than the specific tools or frameworks you use. Just make sure to clearly document how we can run and test your solution.