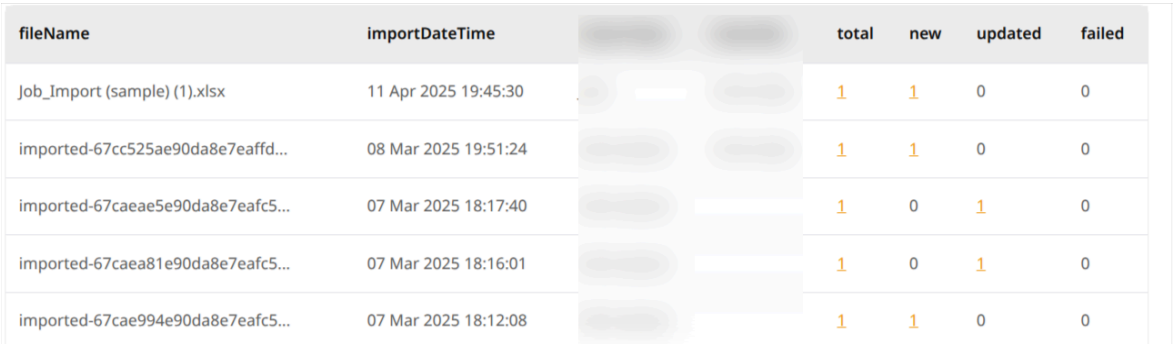
# 🚀 Task: Scalable Job Importer with Queue Processing & History Tracking

## Objective

Design and build a **scalable job import system** that pulls data from an external API(You have multiple APIs that give you a list of jobs), queues jobs using **Redis**, imports them into MongoDB using worker processes, and provides a screen to view **Import History Tracking**

Sample Screen to view import history:-
The fileName will be your URL

Total:- The number of total Jobs inserted for feed
New:- New Record created
Updated: Record updated
Failed: Record failed for any reason(DB Error, validation error, etc..)



| fileName | importDateTime | total | new | updated | failed |
|---|---|---|---|---|---|
| Job_Import (sample) (1).xlsx | 11 Apr 2025 19:45:30 | 1 | 1 | 0 | 0 |
| imported-67cc525ae90da8e7eaffd... | 08 Mar 2025 19:51:24 | 1 | 1 | 0 | 0 |
| imported-67caeae5e90da8e7eafc5... | 07 Mar 2025 18:17:40 | 1 | 0 | 1 | 0 |
| imported-67caea81e90da8e7eafc5... | 07 Mar 2025 18:16:01 | 1 | 0 | 1 | 0 |
| imported-67cae994e90da8e7eafc5... | 07 Mar 2025 18:12:08 | 1 | 1 | 0 | 0 |

This assignment assesses you:
- System design thinking
- Problem-solving via DSA-style logic
- Code structure and modularity
- Documentation and communication skills
- Ability to work end-to-end (optional Docker/Vercel deployment is a bonus)

## 📌 Key Functional Requirements

### 1. Job Source API Integration

- Build a service to fetch jobs from a real API(This you can store into one MongoDB collection and run cron every 1 hour to fetch the job and insert/update into MongoDB).

- This API will provide an XML response, and you need to convert it into JSON.
- API Listing
    - https://jobicy.com/?feed=job_feed
    - https://jobicy.com/?feed=job_feed&job_categories=smm&job_types=full-time
    - https://jobicy.com/?feed=job_feed&job_categories=seller&job_types=full-time&search_region=france
    - https://jobicy.com/?feed=job_feed&job_categories=design-multimedia
    - https://jobicy.com/?feed=job_feed&job_categories=data-science
    - https://jobicy.com/?feed=job_feed&job_categories=copywriting
    - https://jobicy.com/?feed=job_feed&job_categories=business
    - https://jobicy.com/?feed=job_feed&job_categories=management
    - https://www.higheredjobs.com/rss/articleFeed.cfm

#### 2. Queue-Based Background Processing (using Redis)

- Use **Redis + Bull or BullMQ** to manage a background job queue.
- Process jobs in a worker system with **configurable concurrency**.
- Handle and log failures (e.g., invalid data, DB errors).

3. **Import History Tracking**

**For each import run, log:**

- `timestamp`
- `totalFetched`
- `totalImported`
- `newJobs`
- `updatedJobs`
- `failedJobs` **with reasons**

Store these in a separate MongoDB collection named `import_logs`.

## ✅ Required Technologies:

- Frontend: Next.js (Admin UI)
- Backend: Node.js (Express / Nest)
- Database: MongoDB (Mongoose )
- Queue: Bull or BullMQ
- Queue Store: Redis (local or Redis Cloud)

## 2. Architecture Requirements

- Choose your own tech stack – demonstrate your senior-level decision-making.
- Ensure:
  - Clear code separation and modular design.
  - Use of services, helpers, middlewares (if applicable).
  - Scalable design thinking — can this evolve to microservices or plug in later?

## Expected Submission

Your GitHub repository should be structured like this:

```
Java
/client  // Frontend app (e.g., Next.js)
/server // Backend app (e.g., Node.js Express)
/README.md // Setup, usage, and how to run tests
/docs/architecture.md    // System design explanation,
decisions made
```

Use markdown in documentation. Visuals (draw.io, Excalidraw) are highly encouraged.

## Evaluation Criteria

| Criteria | Details | Weight |
|---|---|---|
| Matching Logic | Clean, modular code with good naming and abstraction | 20% |
| Queue Processing & Retry | Concurrency, Redis queue usage, job status tracking | 20% |
| MongoDB Design & Upsert Logic | Avoid duplication, efficiently handle updates | 15% |
| Import History | Clear tracking, filtering, pagination | 15% |
| Docs & Architecture | Architecture.md and README.md clarity and depth | 15% |

### Bonus Points

- Real-time updates using **Socket.IO** or **Server-Sent Events**
- Retry logic and exponential backoff
- Add environment-configurable **batch size** and **max concurrency**

- Deploy to **Render/Vercel**, use **MongoDB Atlas** + **Redis Cloud**

## Time Expectation

You may take up to 1-2 days to complete this assignment. We're looking for thoughtful, complete solutions — not just working code.

## Submission Guidelines

- Submit a public GitHub repository with the required structure.
- Ensure the project is runnable with clear setup instructions.
- Include explanations of key logic and architecture decisions.
- Mention any assumptions in your README.md.

## Notes from Team Lead

- Design a scalable solution that can efficiently insert and update over 1 million job records coming from a feed. Although the feed we currently use is limited, the solution must be capable of handling data on a large scale
- The code should follow industry best practices and coding standards about
- Ensure that the README file includes proper setup guidelines. We do not need an AI-generated document—just accurate and complete setup instructions.