📂 Backend Assignment: Scalable Image Import System from Google Drive

🧩 Objective

Build a scalable backend system to import images from a public Google Drive folder URL and store them in cloud object storage (e.g., AWS S3). The backend should also persist image metadata in a SQL database and expose APIs for importing and retrieving the images. Additionally, create a basic frontend to interact with these APIs. This assignment is designed to assess your ability to architect a scalable, maintainable, and modular backend system using modern best practices.

✅ Functional Requirements

Backend APIs

1. `POST /import/google-drive`

- Accepts a **public Google Drive folder URL** in the request body.
- The backend should:
  - Fetch all **images** from that folder.
  - Upload them to a cloud storage service like **AWS S3** (or equivalent).
  - Persist the following metadata for each image in a SQL database:
    - `name`: Image filename
    - `google_drive_id`: File ID from Google Drive
    - `size`: File size in bytes
    - `mime_type`: MIME type of the file
    - `storage_path`: Path/URL of the file in cloud storage

2. `GET /images`

- Returns a list of all imported images with their metadata.

Frontend Requirements

Create a **basic frontend** that includes:

- A form to input Google Drive folder URL and trigger import
- A page to display the list of imported images
- Basic styling and user interface

⚙️ Technical Requirements

- The system **must be built using a multi-service architecture**. A monolithic implementation will not be accepted.
- It must be designed to handle **large-scale imports (e.g., 10,000+ images)** efficiently. Even if your execution is limited by free-tier quotas, your design must support this scale.
- The system should be **cloud-ready** and designed for **scalability and fault-tolerance**.
- Each service must be **Dockerized**.
- **Both backend and frontend must be deployed** and publicly accessible.
- Deployment can be done using any modern approach:
    - **Container orchestration** (e.g., Kubernetes, ECS, Nomad)
    - **Serverless** (e.g., AWS Lambda, Google Cloud Functions)
    - **Other scalable setups** (e.g., Docker Compose + background workers)
- Use any SQL database (e.g., PostgreSQL, MySQL) and any compatible cloud storage service (e.g., AWS S3, MinIO).

💡 Design Guidelines

- Do **not** build this as a single backend application.
- Think about concurrency, retries, failure recovery, and throughput from day one.
- Your services should be **loosely coupled and independently scalable**.
- Keep configuration portable using environment variables or config files.
- Code must be **modular, clean, and production-ready**.

📎 Bonus (Optional)

- ✅ Add support for importing images from a **Dropbox public folder URL**.
- ✅ Add filters to the `/images` API based on the source (e.g., Google Drive, Dropbox).
- ✅ Provide a **Postman collection** or **CURL scripts** to demonstrate API usage.

📁 Deliverables

Submit **ONLY** the following:

1. **GitHub Repository URL**: Public repository containing all source code
2. **Working Site URL**: Deployed application that is publicly accessible

❌ NO OTHER SUBMISSION FORMATS ACCEPTED

- No ZIP files or attachments
- Only the two URLs above

Repository Must Include:

1. Source code of all backend services and frontend
2. `README.md` containing:
    - **Working site URL** (prominently displayed)
    - Setup instructions (for local and cloud)
    - API documentation with request and response formats
    - Explanation of your architecture and service breakdown
    - Notes on scalability and how large-scale imports are handled
3. Dockerfiles for all services
4. (Optional) Postman collection or API test instructions

🕑 Assignment submission day -

You have to submit the assignment within 7 days

🙌 Final Note

You are encouraged to use **AI tools** during the development process to improve speed and code quality. We're looking for well-structured, scalable backend systems—not just working code. Be thoughtful with your design, think beyond the happy path, and showcase your architectural thinking. Good luck! 🚀