Backend Intern Task

Project Assignment:

File Sharing & Management System (Backend)

TIME DURATION: 48 hours (submissions post this will not be accepted)

Description:

You are tasked with building a file-sharing platform that allows users to upload, manage, and share files. The system should be able to handle multiple users, store metadata in PostgreSQL, manage file uploads to S3 (or locally if cloud integration is not available), implement caching for file metadata. The project must be built in Go and should demonstrate proficiency in handling concurrency and performance optimizations.

Requirements:

1. User Authentication & Authorization:

- **Task**: Implement user authentication (JWT or OAuth2), if you can create your own jwt and store and maintain it in db go ahead we are not limiting you.
- **Requirements**:
  - Users should register/login with email and password.
  - Provide endpoints for registering (POST /register) and logging in (POST /login).
  - Generate JWT tokens upon successful login.
  - Ensure that each user can only manage their own files.

2. File Upload & Management:

- **Task**: Allow users to upload files (e.g., documents, images) to S3 or local storage and manage metadata.
- **Requirements**:
  - Implement an API endpoint (POST /upload) that allows users to upload files.
  - Save file metadata in PostgreSQL (e.g., file name, upload date, size, S3 URL).
  - Return a public URL to access the file.
  - Implement concurrency for processing large uploads using goroutines.

3. File Retrieval & Sharing:

- **Task**: Implement file retrieval API and allow users to share file URLs.
- **Requirements**:
    - Users should be able to retrieve metadata for their uploaded files (GET /files).
    - Provide an option to share the file via a public link (GET /share/:file_id).
    - Metadata should be cached using Redis (or in-memory caching).
    - Optional: Implement URL expiration for shared files.

4. File Search:

- **Task**: Implement search functionality to retrieve files based on metadata.
- **Requirements**:
    - Users can search their files by name, upload date, or file type.
    - The search should be optimized to handle large datasets efficiently.

5. Caching Layer for File Metadata:

- **Task**: Implement a caching mechanism for file metadata to reduce database load.
- **Requirements**:
    - Cache file metadata on retrieval using Redis.
    - Invalidate cache when metadata is updated (e.g., file renamed).
    - Ensure cache is refreshed automatically after expiry (e.g., 5 minutes).

6. Database Interaction:

- **Task**: Design a schema for storing file metadata, S3 locations, and user data in PostgreSQL.
- **Requirements**:
    - Create tables for storing users and files.
    - Ensure efficient queries for retrieving user-specific files.
    - Handle database transactions for critical operations (e.g., file upload).

7. Background Job for File Deletion:

- **Task**: Implement a background worker that periodically deletes expired files from S3 and their metadata from the database.
- **Requirements**:
    - Use Go routines or a background worker to periodically check and delete files.
    - Remove the corresponding metadata from the PostgreSQL database.

8. Testing:

- **Task**: Write tests for your APIs.(can be one or two , just to test if you can do it)

Bonus Tasks (Optional):

- **WebSocket for Real-Time File Upload Notifications**: Implement a WebSocket-based system that notifies users when their file uploads have completed.
- **File Encryption**: Add encryption for files before storing them in S3 and decrypt them when accessed.
- **Hosting** : Host the application on cloud using AWS preferably
- **Implement Rate Limiting:** Add rate limiting to limit only 100 requests per user per minute

---

Requirements:

- **Go** for the backend API and concurrent tasks.
- **Any SQL Database** for the database to store user and file metadata.
- **Redis(in memory if you can handle the volume)** for caching and rate limiting.
- **AWS S3** (or local storage) for file handling.
- **Goroutines and channels** for concurrency in large file uploads and background jobs.
- Dockerize the assignment

Assessment Areas:

1. **API Design**: Quality and structure of API endpoints, request/response handling, and security.
2. **Database Interaction**: Efficient use of PostgreSQL, schema design, and query performance.
3. **Concurrency**: Proper usage of goroutines, channels, and background workers.
4. **Caching and Optimization**: Efficient caching strategies and rate-limiting implementations.
5. **Cloud Integration**: Handling file uploads and storage using S3 (or local).
6. **Error Handling and Logging**: Handling edge cases, rate limits, and structured logging.

DO IT AS MUCH AS POSSIBLE

INSTRUCTIONS:

Submit your final work through GitHub. Add recruitments@trademarkia.com as a contributor on the repository. **Ensure the repository name is `<REGISTER_NUMBER>_Backend`. All the repositories are to be private. We do not encourage plagiarism or over usage of Ai tools .**

Timeline to Submit the task on 30th Mar 2025(Sunday) 8:00 PM IST