BalkanID Full Stack Engineering Intern — Capstone Hiring Task

Overview

You need to build a secure file vault system that supports efficient storage, powerful search, and controlled file sharing. Your task is to design and implement a **production-grade file vault application** with both backend and frontend components.

This project will evaluate your ability to:

- Design and implement scalable APIs (preferred using **GraphQL**).
- Build backend services in **Go (Golang)**.
- Model and query relational data in **PostgreSQL**.
- Develop a modern, user-friendly **frontend application**.
- Demonstrate good software design, documentation, and engineering practices.

Tech Stack

- **Backend:** Go (Golang)
- **API Layer:** REST (GraphQL preferred)
- **Database:** PostgreSQL
- **Frontend:** React.js (or similar modern framework) with Typescript
- **Containerization:** Docker Compose

Core Features

1. File Deduplication

- Detect duplicate uploads using **content hashing (e.g., SHA-256)**.

- Do not store duplicate content—store references instead.
- Track and display per-user **storage savings**.

2. File Uploads

- Support **single and multiple file uploads**.
- Support **drag-and-drop upload** in the frontend.
- Validate file content against declared MIME type to prevent mismatched uploads (e.g., renamed image as .docx).

3. File Management & Sharing

- **List and view files:**
  Users can view a list of all files they own, and see detailed metadata about each file, such as uploader, size, upload date, and deduplication information.
- **Organize Files into Folder *(optional/bonus)***
  Users have the option to organize their files into folders for easier management and navigation.
- **File and Folder Sharing Options:**
  • Share files or folders publicly, making them accessible to anyone with the link.
  • Choose to keep files/folders private, visible only to the owner.
  • (Optional/bonus) Share files or folders with specific users, giving them direct access without making the content public.
- **Public File Statistics:**
  Publicly shared files show download counters, so owners can track how many times their files have been downloaded.
- **Delete files** with strict rules:
  - Only the uploader can delete their files.
  - Files uploaded/owned by others cannot be deleted.
  - Deduplicated files must respect reference counts before actual deletion.

4. Search & Filtering

- Search by filename.
- Filter by: MIME type, size range, date range, tags, uploader’s name.
- Multiple filters can be combined.
- Queries should be optimized for performance at scale.

5. Rate Limiting & Quotas

- Enforce per-user **API rate limits** (default: 2 calls per second, configurable).
- Enforce per-user **storage quotas** (default: 10 MB, configurable).
- Return proper error codes/messages when limits are exceeded.

6. Storage Statistics

- Display:
  - **Total storage used** (deduplicated).
  - **Original storage usage** (without deduplication).
  - **Storage savings in bytes and percentage**.

7. Admin Panel

- Admins can upload files and share them with other users.
- Admin panel should list all files, showing **uploader details**.
- Admins can view **download counts and usage stats**.

Deliverables

- **Backend**: REST or GraphQL API in Go with full feature coverage.
- **Database**: PostgreSQL schema + migrations.
- **Frontend**: Responsive UI with:
  - File uploads (multi-upload + drag-and-drop).
  - Advanced search and filtering.
  - File details, download, and delete functionality.

    - Admin panel with graphs, sharing and analytics (Bonus)
- **Deployment**: Docker setup for local development and testing.
- **Documentation**:
    - Setup instructions (backend + frontend).
    - Database schema overview.
    - API schema with proper documentation (Postman or OpenAPI Spec, GraphQL SDL etc.).
    - Well documented code (think JSDoc, GoDoc)
    - Short design/architecture writeup.
    - Well-written README.

Bonus Points (Optional but Encouraged)

Candidates will receive additional credit for:

- Write a UAT checklist for your app, then automate a few test cases (UI/API/backend) using tools of your choice (e.g. Playwright, Cypress, Postman, Jest). Wire these into your CI pipeline (e.g. GitHub Actions), and add short docs on which QA/UAT methods you tried and why (Storybook, exploratory testing, etc. welcome).
- Implementing **real-time updates** (e.g., live download counts).
- Adding **file previews** for supported file types (e.g., images, PDFs).
- Implementing **role-based access control (RBAC)** for admins and users.
- Improving frontend UX with **progress bars** for uploads.
- Adding **audit logs** for file activity (upload, download, delete).
- **Cloud Deployment**: Deploying the app to any cloud provider and sharing the deployed project link.
- Any additional thoughtful features that improve usability, security, or performance.

Evaluation Criteria

We will evaluate submissions on:

- **Correctness**: Does your implementation meet the functional requirements?
- **Code Quality**: Is the code clean, maintainable, and modular?
- **Design Choices**: Are your architecture and abstractions sensible?
- **UI/UX**: Is the frontend polished, intuitive, and user-friendly?
- **Documentation**: Is your project easy to set up, understand, and extend?
- **Creativity**: Did you go beyond the requirements with useful ideas or features?

Submission

The task submission will be done via Github Classrooms at the link below:

https://classroom.github.com/a/2xw7QaEj

Deadline: All submissions must be completed by **22nd September 2025, 11:59 PM IST**. This is a strict cutoff date, and no submissions will be accepted after the deadline.

Important: All commits will be carefully reviewed for clarity and consistency. Please ensure you maintain a clean and well-organized version control history by committing and pushing your changes regularly to GitHub. Additionally, any form of plagiarism is strictly prohibited and will result in immediate disqualification from all subsequent rounds.