Project Documentation – SecureVault (BalkanID Capstone Task)

1. Setup Instructions

Backend (Go + PostgreSQL)

1. **Clone repository :**
   git clone https://github.com/BalkanID-University/vit-2026-capstone-internship-hiring-task-riya9927

2. **Configure .env**
```
POSTGRES_DB=balkanid
POSTGRES_USER=postgres
POSTGRES_PASSWORD=yourpassword
POSTGRES_PORT=5432

BACKEND_PORT=8080
DATABASE_URL=host=localhost user=postgres password=yourpassword
dbname=balkanid port=5432 sslmode=disable
UPLOAD_PATH=./uploads
STORAGE_QUOTA_BYTES=10485760
RATE_LIMIT_PER_SEC=2
RATE_LIMIT_BURST=4
```

3. **Run migrations**
```
go run migrate.go
```

4. **Start backend**
```
go run main.go
```
   or with Docker Compose:
```
docker-compose up --build
```

Frontend (React + TypeScript + Vite)

1. **Clone repository :**
   git clone https://github.com/BalkanID-University/vit-2026-capstone-internship-hiring-task-riya9927

2. **Install dependencies :**
```
npm install
```

3. **Configure .env**
```
VITE_API_URL=http://localhost:8080
```

4. **Run development server:**
```
npm run dev
```

5. Visit frontend at http://localhost:5173.

2. Database Schema Overview

Tables

- **users** – user accounts (id, username, role).
- **files** – uploaded files (id, filename, hash, uploader_id, size, type, download_count).
- **folders** – optional grouping of files.
- **shared_file_access** – file sharing records.
- **shared_folder_access** – folder sharing records.

Relationships

- users (1) → (M) files
- users (1) → (M) folders
- files (M) ↔ (M) users via shared_file_access
- folders (M) ↔ (M) users via shared_folder_access

Deduplication is implemented via **hashing (SHA-256)**:

- If file hash exists, only a reference is stored, not duplicate content.

3. API Schema

File Endpoints
- POST /upload – upload file(s)
- GET /files – list files
- GET /files/:id – get file metadata.
- GET /files/:id/download – download (increments count)
- DELETE /files/:id – delete file
- POST /files/:id/share – toggle public share
- POST /files/:id/share/user – share with specific user
- DELETE /files/:id/share/user – revoke share
- GET /files/:id/shared_with – list shared users.

Folder Endpoints
- POST /folders – create folder
- GET /folders – list folders
- GET /folders/:id/files – list files in folder
- POST /folders/:id/share/user – share folder

Statistics
- GET /storage/stats – per-user stats (original vs deduped).
- GET /admin/stats – global stats (admin only).

Search & Filtering
- GET/search?q=&mime=&minSize=&maxSize=&startDate=&endDate=&tags=&uploader=

Admin
- GET /admin/files – list all files with uploader.
- POST /admin/share/:fileID – share file with user.

4. Code Documentation

- **Go (backend): functions documented with GoDoc style.**

Example:

```
// UploadHandler handles single and multiple file uploads.
// Validates MIME type and applies deduplication using SHA-256.
 func UploadHandler(c *gin.Context) { ... }
```

- **TypeScript (frontend): components documented with JSDoc.**

Example:

```
/**
Upload component with drag-and-drop support.
Shows progress and validates file types before upload.
**/
export default function Upload() { ... }
```

5. Design & Architecture

Backend

- Written in Go (Gin framework).
- Layered design: models/, handlers/, migrations/.
- PostgreSQL for persistence.
- Deduplication via SHA-256 hashing.
- Rate limiting (token bucket).
- Quotas enforced per user.
- Real-time updates via SSE for download counts.

Frontend

- React + Vite + TypeScript.
- Components: Upload, UserFiles, Search, Statistics, AdminPanel.
- State management via React hooks.
- Minimal CSS (clean, responsive).

Deployment

- **Local: Docker Compose for backend, frontend, PostgreSQL.**

6. Extensibility

- GraphQL API can be added as a layer on top of current REST.
- Auth system can be extended with JWT/OAuth instead of X-User.
- File preview can be added for supported MIME types.
- Advanced analytics for admin (charts, graphs).