Balkan Storage – Software Documentation

BalkanID Capstone Internship Hiring Task – VIT 2026

Prepared By:

Name: Mahidhar Reddy G

Email: mahidhar.reddy2003@gmail.com

College: VIT Vellore

Contents Of The Document

1. Introduction

Balkan Storage is a production-ready **file storage and management system** developed as part of the **VIT 2026 Capstone Internship Hiring Task**. It demonstrates full-stack engineering and deployment skills with:

- **Backend:** Go (Gin) + PostgreSQL
- **Frontend:** Next.js (React, TypeScript, TailwindCSS)
- **Containerization:** Docker Compose
- **Extra Tooling:** API Testing (Postman, Newman), UI Testing (Playwright), Storybook

The system provides **secure file storage, deduplication, folder management, audit logging, and a modern UI using glassmorphism design.**

2. Setup Instructions

Backend (Go + PostgreSQL)

1. Clone the repository:

```
git clone <your-repo-url>
cd vit-2026-capstone-internship-hiring-task-mahidharreddyg
```

2. Create `.env` in `backend/` with:

```
DB_HOST=db
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=postgres
DB_NAME=balkan_storage
PORT=8080
JWT_SECRET=supersecret
STORAGE_PATH=/data/storage
```

3. Run backend locally:

```
cd backend
go run main.go
```

API: http://localhost:8080

Frontend (Next.js + React + TypeScript)

1. Create `.env.local` in `frontend/`:

```
NEXT_PUBLIC_API_URL=http://localhost:8080
```

2. Install dependencies:

```
cd frontend
npm install
```

3. Run frontend locally:

```
npm run dev
```

App: http://localhost:3000

Docker (Full System – One Command)

```
docker compose up --build
```

- Frontend → http://localhost:3000
- Backend → http://localhost:8080
- PostgreSQL → `localhost:5432`

3. Database Schema Overview

Users

- Stores account details
- Passwords hashed using bcrypt

Column	Type	Description
id	SERIAL PK	Unique user ID
username	TEXT	Unique username
email	TEXT	User email
password_hash	TEXT	Bcrypt hashed password
created_at	TIMESTAMP	Account creation time

Files

- Supports deduplication (SHA-256 hash)

Column	Type	Description
id	SERIAL PK	File ID
owner_id	INT (FK)	Reference to users
name	TEXT	File name
mime_type	TEXT	Detected file type
size	BIGINT	File size in bytes
hash	TEXT	SHA-256 hash
created_at	TIMESTAMP	Upload timestamp

Folders

Column	Type	Description
id	SERIAL PK	Folder ID
owner_id	INT (FK)	Reference to users
name	TEXT	Folder name
parent_id	INT (FK)	Nested folder support

Audit Logs

Column	Type	Description
id	SERIAL PK	Log ID
user_id	INT (FK)	User performing action
action	TEXT	login, upload, delete
object_type	TEXT	Target entity (file/folder)
object_id	INT	Entity ID
created_at	TIMESTAMP	Action timestamp

4. API Documentation

Base URL: `http://localhost:8080`

Authentication

- `POST /signup`
- `POST /login`
- `GET /verify-token`

File Management

- `GET /files`
- `POST /upload`
- `POST /multi-upload`
- `PATCH /files/:id/move`
- `PATCH /files/:id/trash`
- `DELETE /trash/:id`

Folder Management

- `POST /folders`

- `GET /folders`
- `PATCH /folders/:id`
- `PATCH /folders/:id/trash`

Admin

- `GET /admin/files`
- `GET /admin/stats`

5. Architecture Design

- **Frontend:** Next.js (React + TS + TailwindCSS, glassmorphism UI)
- **Backend:** Go (Gin), REST API with JWT
- **Database:** PostgreSQL with indexing & relational queries
- **Containerization:** Docker Compose to bring up DB, backend, frontend

Features: drag-and-drop uploads, deduplication, audit logging, rate limiting

6. Design Choices

- Security: JWT auth, bcrypt password hashing, CORS enabled
- Scalability: File deduplication + bulk API
- UX: Glassmorphism, drag-and-drop upload, responsive UI
- Observability: Audit logs for all user/file actions
- Extensibility: RBAC and GraphQL can be added

7. User Acceptance Testing (UAT) Checklist

- [x] User signup, login, invalid credentials rejected
- [x] File upload (single & multi), deduplication works
- [x] Folder CRUD and file movements supported
- [x] Trash + restore flow tested

- ☑ Admin can view files and stats
- ☑ Rate limiting works against abuse
- ☑ Search (name, type, date, tags) works

8. Deployment

Local (Docker)

```
docker compose up --build
```

Cloud

- Push Docker images to registry
- Use Kubernetes/Docker Swarm
- Add Nginx + HTTPS proxy

9. Testing & QA Automation

- **Postman + Newman:** API regression tests
- **Playwright:** UI E2E (signup, login, dashboard)
- **CI/CD:** GitHub Actions workflow with backend vetting, Newman, Playwright
- **Database tests:** Can be extended with migration seeds

10. Storybook and Visual Testing

- Storybook spins up component previews on port 6006
- Scripts:

```
npm run storybook
npm run build-storybook
```

- For visual diffs: Percy / Chromatic

11. UAT Automation Plan

- Automate flows: Signup → Login → Upload → File List → Sharing
- API integration: Newman + seeded DB in CI
- UI integration: Playwright headless E2E in GitHub Actions
- Visual regression nightly via Storybook

12. Conclusion

This project highlights **end-to-end full-stack development** using modern tooling:

- Secure Go backend
- PostgreSQL with deduplication
- Cutting-edge Next.js UI with glassmorphism styling
- Dockerized for easy deployment
- Automated testing pipelines with Postman, Playwright, Storybook

Designed to showcase **production-level engineering for recruitment evaluation**.