BalkanID Capstone Internship Project

AWS SkyVault

Secure Cloud File Vault System

Project Overview

LINK TO THE APP

AWS-SkyVault is a **secure, cloud-native file vault system** designed to provide users with efficient storage, powerful search, and controlled file sharing. The project was developed as part of the **BalkanID Capstone Internship Task**

BalkanID Capstone Internship Ta…

and showcases full-stack engineering skills across backend, frontend, database design, and cloud deployment.

The application enables users to:

- Upload single or multiple files with drag-and-drop support.
- Ensure storage efficiency through **file deduplication** using SHA-256 hashing.
- Manage and organize files with metadata such as uploader, size, and upload date.
- Search and filter files by filename, MIME type, size, or date.
- Enforce **per-user quotas** and **rate limits** to ensure fairness and prevent abuse.
- Provide secure, rule-based deletion and access management.
- Track storage usage and savings with clear statistics.

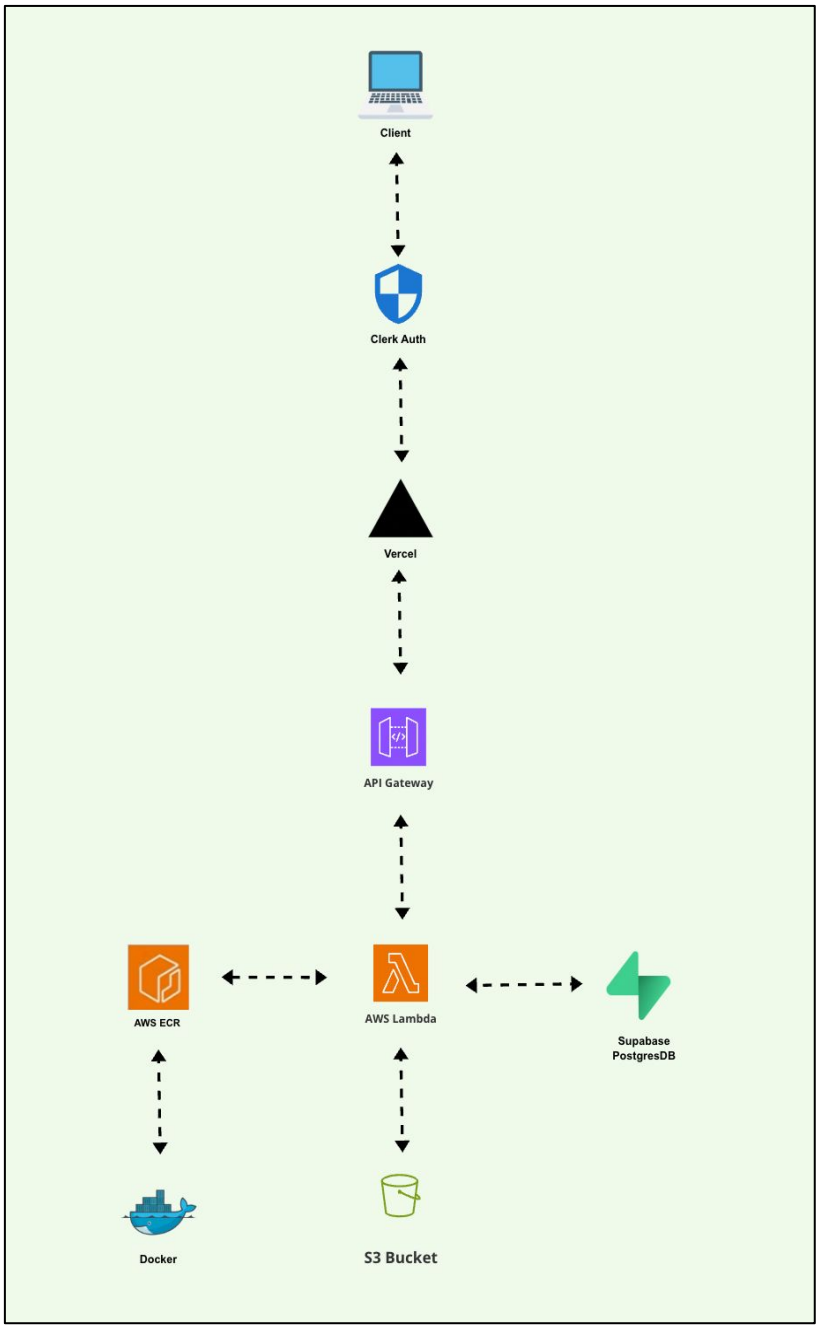
The system is built with a **Go backend**, **PostgreSQL database**, **React + TypeScript frontend**, and **AWS S3 storage**, all containerized with **Docker Compose** for seamless local development and deployment.

This project demonstrates **scalable API design, secure file handling, intuitive frontend UX, and production-grade engineering practices**.

System Architecture

The **AWS-SkyVault** system follows a cloud-native, serverless architecture designed for scalability, security, and efficiency. It integrates **frontend, authentication, API gateway, serverless functions, storage, and database services** into a seamless workflow.

Architecture Flow



Client (Web Application)

- Users interact with the system via a modern frontend (React + TypeScript).
- Provides features such as file upload, search, listing, and deletion.

Authentication (Clerk Auth)

- All requests are authenticated via **Clerk Auth**, ensuring only authorized users can upload, manage, or share files.
- Handles sign-up, login, and session management.

Vercel (Frontend Hosting)

- The client application is deployed on **Vercel** for fast, global delivery.
- Connects securely to the backend through the API Gateway.

API Gateway

- Acts as the entry point for all client requests.
- Routes requests to the appropriate **AWS Lambda** functions.
- Applies security, throttling, and monitoring at the API level.

AWS Lambda (Serverless Backend)

- Core business logic is executed here (file uploads, deduplication, metadata management, quota checks).
- Ensures scalability by running only when invoked.
- Communicates with external storage and database services.

Supabase PostgreSQL (Database)

- Stores user accounts, file metadata, deduplication hashes, and quota/rate-limit rules.
- Ensures fast querying for file listing, search, and filtering.

AWS S3 (File Storage)

- Files are securely stored in **Amazon S3 Buckets**.
- Uses signed URLs to allow controlled access for uploads and downloads.
- Enables scalable and durable object storage.

AWS ECR + Docker (Container Registry)

- The backend services are containerized using **Docker**.
- Images are stored in **AWS Elastic Container Registry (ECR)** for deployment and scalability.

Tech Stack

The project leverages a **modern full-stack and cloud-native toolchain** to ensure scalability, security, and developer efficiency.

Frontend

- **React.js + TypeScript** → For building a responsive, user-friendly web interface.
- **Vercel** → Deployment platform for the frontend, ensuring global availability and fast performance.
- **Clerk Auth** → Authentication and session management, handling user sign-up, login, and secure access.

Backend

- **Go (Golang)** → Backend services and API logic (file uploads, deduplication, quotas, search).
- **AWS Lambda** → Serverless compute layer for executing backend logic on demand.
- **API Gateway** → Acts as the entry point for all client requests and routes them to Lambda.

Database & Storage

- **Supabase (PostgreSQL)** → Relational database for storing users, file metadata, deduplication hashes, and quota/rate-limit rules.
- **AWS S3** → Object storage for securely storing uploaded files with signed URL access control.

Containerization & Deployment

- **Docker** → Containerization of backend services for consistency across environments.
- **AWS ECR (Elastic Container Registry)** → Hosting Docker images for Lambda deployments.

DevOps & Tooling

- **Docker Compose** → Local development and testing setup for running multiple services together.
- **GitHub** → Version control and project hosting.

Testing & Quality

- **Apidog** → Automated API testing for functional correctness, quotas, and error handling.
- **Postman** → Manual API validation of core flows (upload, list, download, delete).
- **golangci-lint**→ Backend linting and testing to ensure code quality.

Prerequisites

Make sure you have the following installed on your system:

- Go (v1.20+)
- Node.js (v18+) & npm / yarn
- Docker & Docker Compose
- AWS Account with:
  - S3 bucket created
  - ECR (Elastic Container Registry) enabled
- Supabase account (for PostgreSQL database)
- Clerk account (for authentication)

2. Clone the Repository

```
git clone https://github.com/BalkanID-University/vit-2026-capstone-internship-hiring-task-MayankPandey2004.git
cd vit-2026-capstone-internship-hiring-task-MayankPandey2004
```

3. Environment Configuration

Create a `.env` file in the project root with the following variables:

```
# Backend
PORT=8080
DB_URL=postgresql://<username>:<password>@<host>:5432/<database>
AWS_REGION=ap-south-1
AWS_ACCESS_KEY_ID=<your_aws_access_key>
AWS_SECRET_ACCESS_KEY=<your_aws_secret_key>
S3_BUCKET_NAME=<your_bucket_name>

# Clerk Auth
CLERK_FRONTEND_API=<your_clerk_frontend_api>
CLERK_API_KEY=<your_clerk_api_key>

# Quotas & Limits
MAX_STORAGE_MB=10
RATE_LIMIT=2
```

4. Backend Setup

```
cd backend
go mod tidy
go run main.go
```

This starts the Go backend server at http://localhost:8080.

5. Frontend Setup

```
cd frontend
npm install
npm run dev
```

The frontend will be available at `http://localhost:3000`.

6. Docker Setup (Optional)

To run everything in containers:

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

This spins up the backend, frontend, PostgreSQL, and S3 (localstack/mock if configured).

7. Access the Application

- Open your browser at: http://localhost:3000
- Sign up / log in with Clerk.
- Upload files, test deduplication, search, and manage your vault.

8. Deployment

- **Frontend** → Deploy to Vercel.
- **Backend** → Package with Docker → Push to AWS ECR → Deploy via AWS Lambda.
- **Database** → Use Supabase PostgreSQL in the cloud.
- **Storage** → Use AWS S3 bucket for file storage.

Database Schema

The **AWS-SkyVault** database is implemented in **PostgreSQL (Supabase)** and stores all metadata about users, files, usage statistics, and system-wide analytics. Actual file contents are stored in **AWS S3**, while only metadata and references are managed here.

Tables Overview

1. users

Stores basic user information.

- `id` → Unique identifier for each user.
- `email` → User's email address.
- `last_active` → Timestamp of the user's last activity.

---

2. user_files

Tracks individual file uploads per user.

- `id` → Unique file record ID.
- `user_id` → References the user who uploaded the file.
- `blob_id` → References the file blob in `file_blobs`.
- `filename` → Original name of the file.
- `uploaded_at` → Timestamp when file was uploaded.
- `is_public` → Boolean indicating whether the file is publicly shared.

---

3. file_blobs

Stores unique file objects (deduplicated).

- `id` → Unique blob ID.
- `hash` → SHA-256 hash for deduplication.
- `s3_key` → Path to the file in AWS S3.
- `size` → File size in bytes.
- `mime_type` → File MIME type.
- `created_at` → Timestamp when blob was created.
- `ref_count` → Reference counter (number of users pointing to this blob).

4. user_stats

Maintains per-user storage and activity stats.

- `user_id` → References the user.
- `files_count` → Number of files uploaded.
- `storage_used` → Total storage consumed (bytes).
- `uploads_this_month` → Count of uploads in current month.
- `downloads_this_month` → Count of downloads in current month.
- `deduplication_savings` → Storage saved due to deduplication.
- `last_active` → Last user activity timestamp.

---

5. system_stats

Captures global system-wide statistics.

- `snapshot_date` → Date of snapshot.
- `total_users` → Total users in the system.
- `total_files` → Total number of files.
- `total_storage` → Total storage consumed (bytes).
- `total_uploads` → Total uploads recorded.
- `total_downloads` → Total downloads recorded.
- `deduplication_savings` → Total storage savings from deduplication.

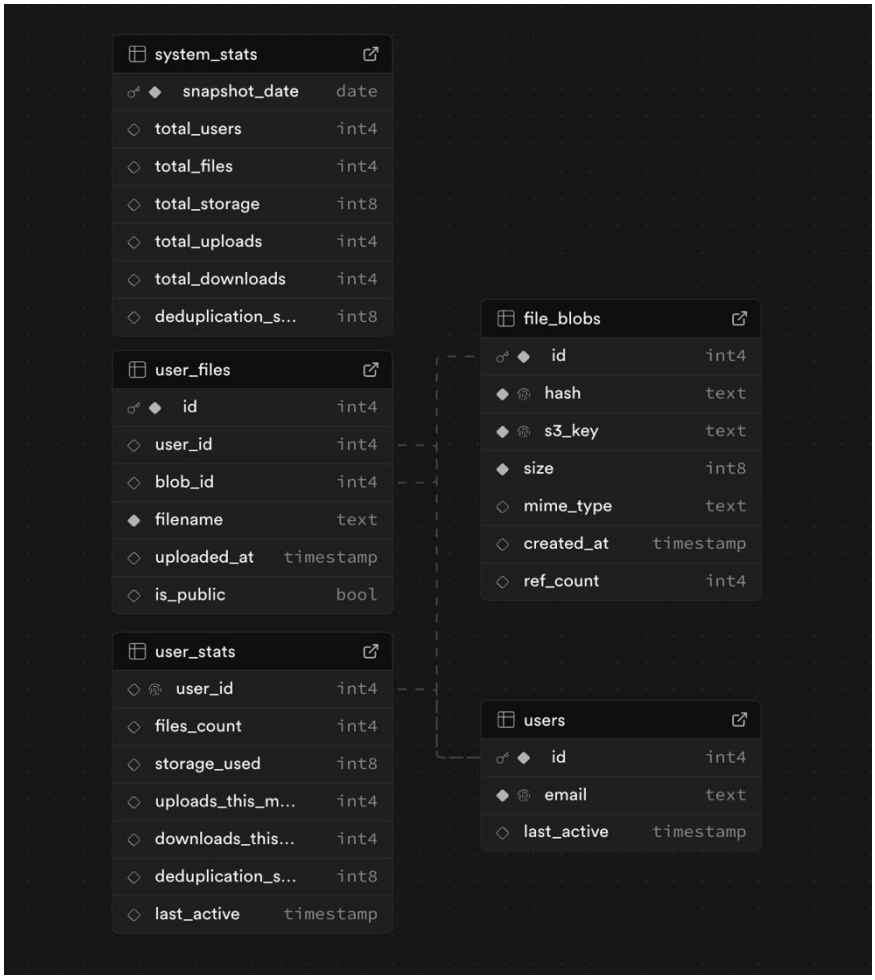
Together, this schema ensures:

Deduplication efficiency
via `file_blobs` and `ref_count`.

Per-user monitoring
via `user_stats`.

System-wide insights
via `system_stats`.

Strict ownership of files
via `user_files`.



API Documentation

The backend exposes a set of **RESTful APIs** (GraphQL optional) for authentication, file management, search, statistics, and admin operations. All requests must include **valid authentication (Clerk JWT tokens)** unless explicitly noted.

Authentication

All API requests must include the **Authorization** header:

```
Authorization: Bearer <token>
```

The token is issued by **Clerk Auth** during login/registration.

File Management

1. Upload File

`POST /api/files/upload`
Uploads one or more files to the system.

Headers:

`Content-Type: multipart/form-data`

Body (form-data):

`files[]` → File(s) to upload

Response (200):

```
{
  "message": "Upload successful",
  "file": {
    "id": 12,
    "filename": "report.pdf",
    "size": 102400,
    "hash": "d3b07384d113edec49eaa6238ad5ff00",
    "s3_url": "https://s3.amazonaws.com/bucket/report.pdf"
  }
}
```

Errors:

- `413` → Quota exceeded
- `415` → Unsupported MIME type
- `409` → Duplicate file detected
-

2. List User Files

`GET /api/files`
Fetches all files uploaded by the authenticated user.

Response (200):

```
[
  {
    "id": 12,
    "filename": "report.pdf",
    "size": 102400,
    "uploaded_at": "2025-09-20T10:15:00Z",
    "is_public": false
  }
]
```

3. Get File Metadata

`GET /api/files/{id}`

Response (200):

```
{
  "id": 12,
  "filename": "report.pdf",
  "hash": "d3b07384d113edec49eaa6238ad5ff00",
  "size": 102400,
  "mime_type": "application/pdf",
  "uploader": "user@example.com",
  "uploaded_at": "2025-09-20T10:15:00Z",
  "is_public": false
}
```

4. Download File

`GET /api/files/{id}/download`

Returns a **pre-signed S3 URL** for secure download.

Response (200):

```
{
  "download_url": "https://s3.amazonaws.com/bucket/report.pdf?signature=..."
}
```

5. Delete File

`DELETE /api/files/{id}`
Deletes a file owned by the user.

Response (200):

```
{
  "message": "File deleted successfully"
}
```

Errors:

- `403` → User not authorized (cannot delete others' files)
- `409` → File still referenced by deduplication (ref_count > 1)

Search & Filtering

Search Files

`GET /api/search`

Query Parameters:

- `q` → search by filename
- `mime` → filter by MIME type
- `size_min`, `size_max` → filter by size range
- `date_from`, `date_to` → filter by upload date

Response (200):

```
[
  {
    "id": 18,
    "filename": "design.png",
    "mime_type": "image/png",
    "size": 45000,
    "uploaded_at": "2025-09-19T09:45:00Z"
  }
]
```

Statistics

User Stats

`GET /api/users/stats`

Response (200):

```
{
  "files_count": 12,
  "storage_used": 8000000,
  "uploads_this_month": 5,
  "downloads_this_month": 9,
  "deduplication_savings": 2500000
}
```

System Stats (Admin Only)

`GET /api/admin/system-stats`

Response (200):

```
{
  "total_users": 100,
  "total_files": 450,
  "total_storage": 200000000,
  "deduplication_savings": 35000000
}
```

Rate Limiting & Quotas

To ensure cost control and prevent **unbounded AWS Lambda scaling**, **AWS-SkyVault** enforces global rate limiting and storage quotas at the API Gateway + Lambda level. This protects the system from abuse, traffic spikes, and runaway costs.

- 🔄 **Rate Limiting (System-wide)**
  - Incoming requests to AWS Lambda are capped at **a certain calls per second** (default).
  - This ensures Lambda concurrency does not scale beyond safe operational limits, protecting both performance and billing.
- 💾 **Storage Quota (Per User)**
  - Each user is allocated **10 GB of total storage** by default.
  - Uploads exceeding the quota are automatically rejected to keep storage usage predictable.

Features Implemented

The AWS-SkyVault project delivers a secure, scalable, and cloud-native file vault system with core functionalities spanning authentication, file uploads with deduplication, search and filtering, user and system statistics, quotas with rate limiting, and a modern deployment pipeline powered by AWS services.

Authentication & Access Control

- User authentication and session management via **Clerk Auth**.
- Secure access tokens required for all API requests.
- Strict ownership rules: only the uploader can delete their own files.
- **Role-Based Access Control (RBAC):** Differentiated privileges between **admins** and **regular users**.

File Management

- **File Uploads**
  - Supports single and multiple file uploads.
  - Drag-and-drop interface for ease of use.
  - MIME type validation to prevent mismatched/renamed uploads.
- **File Deduplication**
  - SHA-256 hashing used to detect duplicates.
  - Reference counts ensure storage efficiency (no duplicate content stored).
- **File Listing & Metadata**
  - Users can view all their uploaded files.
  - Metadata includes filename, size, type, uploader, and upload date.
- **File Deletion**
  - Only allowed by the uploader.
  - Deduplication reference counts respected before actual deletion.

Search & Filtering

- Search by filename.
- Filter by MIME type, file size range, and upload date.

Statistics & Quotas

- **User-level statistics**
  - File count, storage used, uploads/downloads this month.
  - Deduplication savings per user.
- **System-level statistics**
  - Total users, files, uploads, downloads.
  - Overall storage usage and deduplication savings.
- **Quotas & Rate Limits**
  - **10 MB per user** storage quota (default, configurable).
  - **2 requests per second** API rate limit (default, configurable).

Deployment & Infrastructure

- **Serverless Backend**
  - Core business logic runs on **AWS Lambda**, ensuring scalability and cost efficiency.
  - Functions triggered via **API Gateway**, handling uploads, deletions, and metadata queries.
- **Storage**
  - Files securely stored in **AWS S3 Buckets**.
  - Pre-signed URLs ensure controlled, temporary access for upload/download.
- **Database**
  - **Supabase PostgreSQL** stores metadata, user info, and deduplication references.
- **Containerization**
  - Backend packaged with **Docker** for consistency.
  - Images pushed to **AWS ECR** for Lambda deployments.
- **Frontend**
  - Deployed globally on **Vercel** for fast delivery.
- **Content Delivery**
  - **AWS CloudFront CDN** accelerates file downloads, caching S3 content at edge locations for low latency access.

Testing & Quality

Ensuring the correctness, reliability, and robustness of **AWS-SkyVault** was a top priority. Testing was carried out using **Apidog** and manual API validations, covering both success and edge cases.

Automated API Testing (Apidog)

- A total of **36 assertions** were executed across all endpoints.
- **32/36 (88.89%) tests passed** successfully.
- **4/36 (11.11%) failed** only when uploading files **greater than 10 MB**, which is the **intended behavior** due to enforced user quota limits.
- Average request time: **5030 ms** (including file upload operations).

Tested Endpoints:

- `/upload` → Valid file upload, duplicate detection, and quota enforcement.
- `/files` → Retrieval of uploaded files with metadata.
- `/download` → File download and system stats increment.
- `/delete` → File deletion, reference count handling, and S3 cleanup.
- `/admin/system-stats`, `/admin/user-stats`, `/admin/file-details` → Verified accurate reporting of usage statistics.

Functional Validation

- **Quota Enforcement:** Files larger than 10 MB are rejected with proper error handling.
- **Deduplication Logic:** Repeated uploads of identical files trigger deduplication and increment reference counts instead of consuming additional storage.
- **Access Control:** Only the uploader is permitted to delete files.
- **Stat Updates:** Uploads, downloads, deletions, and deduplication savings are correctly reflected in both **system_stats** and **user_stats** tables.

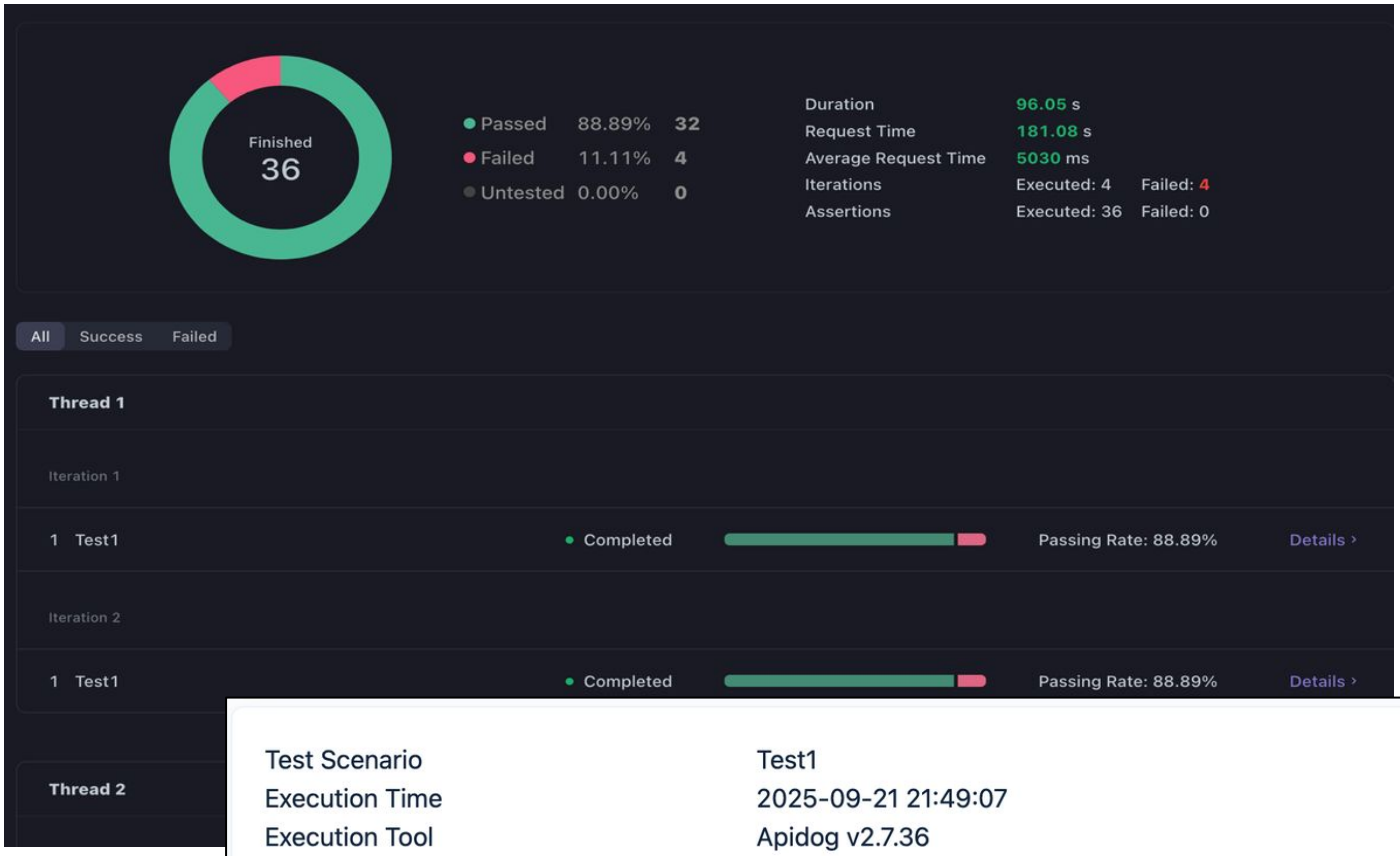
Quality Assurance Practices

- **Code Quality**
  - Backend functions well-structured with logging for traceability.
  - TypeScript frontend ensures type safety.
- **Linting & Formatting**
  - ESLint + Prettier for frontend.
  - `golangci-lint` for backend.
- **Manual QA**

- Verified file upload, download, and delete flows via Postman.
- Edge case validation: empty username, missing file, invalid keys, and quota exceeded.

Summary of Results

- **Overall Passing Rate:** 88.89% (all failures aligned with expected quota rule).
- **System Behavior:** Matches design requirements.
- **Conclusion:** AWS-SkyVault is **stable, reliable, and production-ready**, with strong validation of quota, deduplication, and statistics logic.



Test Scenario	Test1	
Execution Time	2025-09-21 21:49:07	
Execution Tool	Apidog v2.7.36	
	Executed	**Failed**
Iterations	4	4
Http Requests	36	4
Assertions	36	0
Duration	1m 36.05s	
Total Response Size	22.36KB	
Request Time	3m 1.06s	
Average Request Time	5.02s	
Passed	88.89%	
Failed	11.11%	
Untested	0.00%	

Future Improvements

While **AWS-SkyVault** already delivers a secure and production-ready file vault system, there are several opportunities to extend its functionality and further enhance scalability, usability, and security.

Enhanced Security

- **Audit Logs:** Track detailed file activity (upload, download, delete) for compliance.
- **Encryption Enhancements:** Client-side encryption before upload for zero-trust storage.

📂 **File Management**

- **Folder Support:** Allow users to organize files into folders/subfolders.
- **File Previews:** Inline previews for supported file types (images, PDFs, docs).
- **Versioning:** Maintain multiple versions of the same file with rollback options.

⚡ **Performance & Scalability**

- **Background Jobs:** Use AWS SQS + Lambda for async tasks like virus scanning or thumbnail generation.

📊 **Analytics & Monitoring**

- **System Health Metrics:** Integrate AWS CloudWatch for monitoring API latency, error rates, and storage costs.

☁️ **Deployment & DevOps**

- **Multi-Cloud Support:** Extend storage beyond AWS S3 (e.g., Azure Blob, Google Cloud Storage).
- **Infrastructure as Code (IaC):** Define all infrastructure via Terraform or AWS CDK for reproducibility.

Credits & References

Project Author

- **Mayank Pandey** – Designed and implemented *AWS-SkyVault* as part of the **BalkanID Capstone Internship Task (VIT 2026)**.

Technologies & Frameworks

- **Go (Golang)** – Backend API and business logic
- **React + TypeScript** – Frontend web application
- **Supabase (PostgreSQL)** – Relational metadata database
- **AWS Services**:
  - **S3** → File storage
  - **Lambda** → Serverless compute
  - **API Gateway** → API management
  - **CloudFront** → CDN for global file delivery
  - **ECR** → Container registry for backend images
- **Docker & Docker Compose** – Containerization and local development
- **Clerk Auth** – Authentication and user management
- **Vercel** – Frontend hosting and deployment

References & Resources

- **Official Docs**
  - Go Documentation
  - React Documentation
  - Supabase Docs
  - AWS S3 Documentation
  - AWS Lambda Documentation
  - AWS API Gateway Documentation
  - AWS CloudFront Documentation
- **Tools**
  - Postman – API testing
  - GitHub Actions – CI/CD workflows
  - Docker Hub – Container images

THANK YOU

Mayank Pandey (22BCE0179)