Secure File Management System

In the Context of South African Police Service (SAPS)

Security and Accountability

Mr. L Saohatse

Date: September 23, 2025

Individual Project

Version: 1.0

Contents

1 Introduction

1.1 Overview

The SAPS Secure File Management System (SFMS) is a cloud-based solution designed to digitize, manage, and secure files and dockets for the South African Police Service (SAPS). Hosted within the AWS ZA region, it ensures compliance with the Protection of Personal Information Act (POPIA) and the National Policy on Data and Cloud 2024, providing a secure, scalable, and user-friendly platform for law enforcement operations.

1.2 Objectives

- Ensure data sovereignty by restricting data to the AWS Cape Town region.
- Provide secure file upload, storage, and retrieval with encryption.
- Enable digitization and version control of dockets.
- Support multi-language interfaces (English, Afrikaans, Zulu, Xhosa, Sotho).
- Maintain audit trails and compliance with government standards.

2 System Architecture

2.1 Overview

The SFMS employs a client-server architecture optimized for security and compliance in the South African public sector. It features a mobile-responsive web frontend, a robust backend API server, and a cloud-hosted database, all deployed within the AWS ZA region.

2.2 Components

- **Frontend**: Developed using React.js with Next.js for server-side rendering, ensuring mobile-responsiveness and support for multiple South African languages via i18n libraries.
- **Backend**: Built with Spring Boot (Java) for enterprise-level security features, including built-in OAuth2 support and robust authentication mechanisms.
- **Database**: PostgreSQL for relational data integrity, ensuring ACID compliance crucial for audit logs and chain of custody.
- **Cloud Infrastructure**: Hosted on AWS in the Cape Town region to comply with South African data sovereignty laws, offering services like S3 for encrypted file storage and RDS for databases.
- **Security Layer**: Integrates MFA via Auth0, biometric authentication using WebAuthn, AES-256 encryption, and comprehensive logging with ELK Stack.

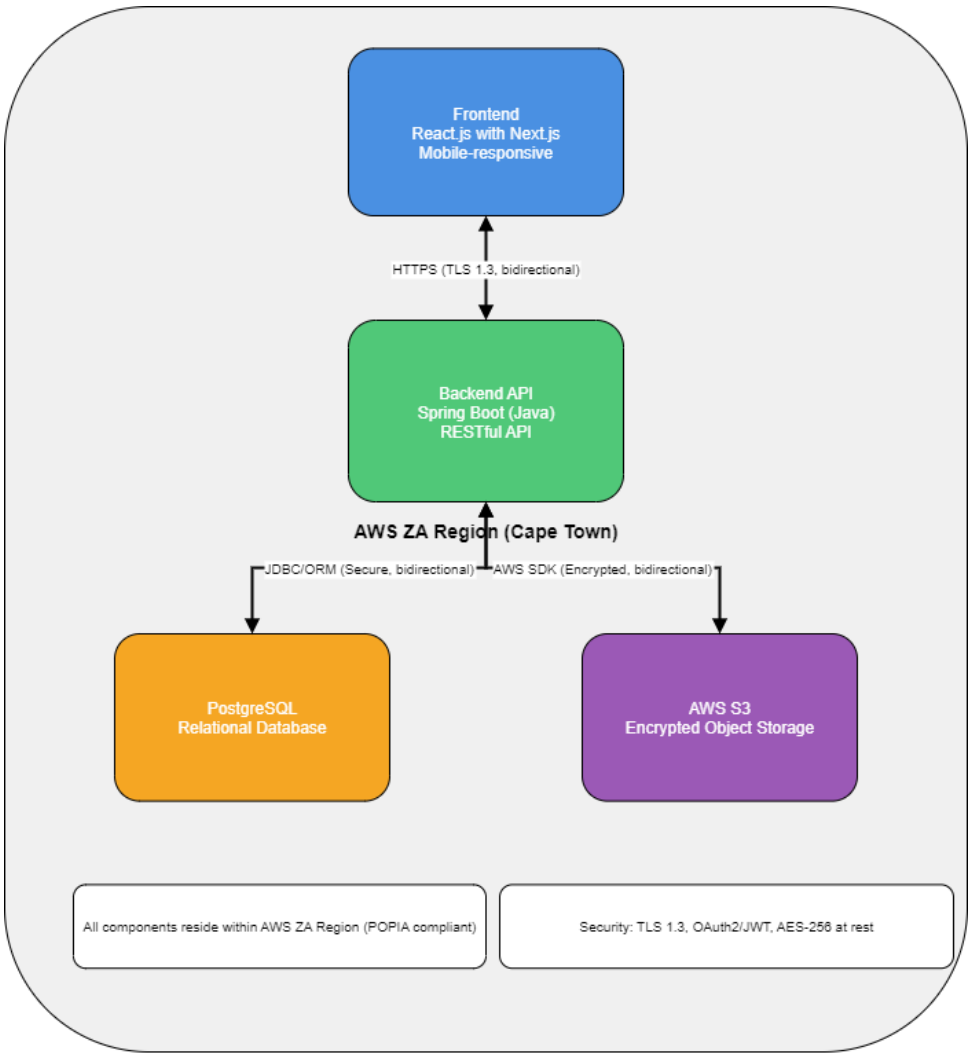

Figure 1: High-level architecture showing frontend connecting to backend API, backend interacting with PostgreSQL and AWS S3, all within AWS ZA region.

[System Architecture Diagram Placeholder: A high-level diagram showing frontend connecting to backend API, backend interacting with PostgreSQL and AWS S3, all within AWS ZA region.]

[Notes] - All components reside within AWS ZA Region for data sovereignty compliance. - Communication secured with TLS 1.3 and OAuth2/JWT. - Diagram uses arrows to indicate data flow (Frontend -> Backend -> PostgreSQL/S3).

3 Detailed Design

3.1 Module 1: Secure File and Docket Management

3.1.1 Description

This module handles the secure upload, storage, digitization, organization, and tracking of files and dockets, ensuring integrity and compliance.

3.1.2 Interfaces

- **Input**: File data (PDF, images, documents), metadata (case ID, individual details), physical location info.
- **Output**: Stored file IDs, version history, search results, integrity verification status.

3.1.3 Responsibilities

- Secure file upload/download using HTTPS and client-side encryption.
- Digitization integration with scanner hardware via WebUSB API.
- File organization and categorization with metadata tagging.
- Version control using Git-like mechanisms in Spring Boot.
- Association of dockets with individuals/cases via database relations.
- Physical docket location tracking, restricted by RBAC to original station.
- Integrity verification using SHA-256 hashes.

3.2 Class Diagrams

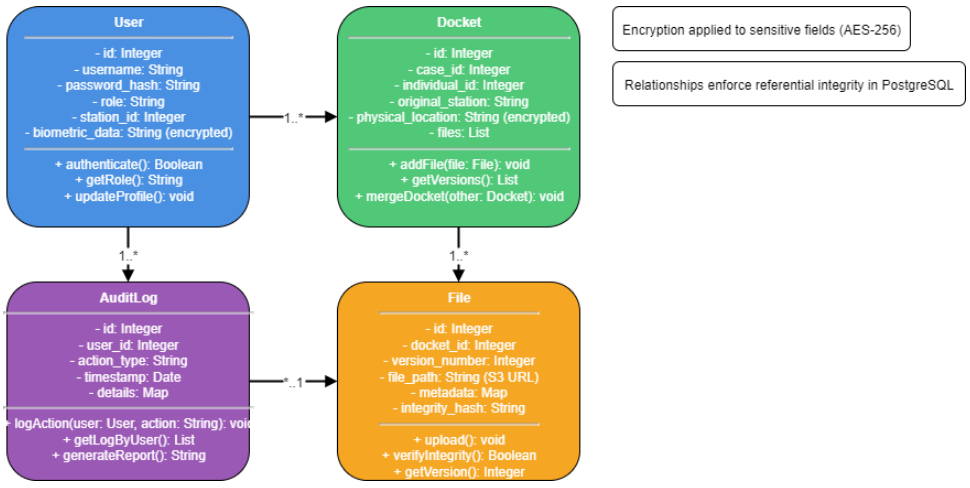

Figure 2: UML Class Diagram for User entity.

[Class Diagrams Placeholder: UML diagrams for key entities like User, Docket, File, AuditLog.]

[Notes] - All attributes marked with '-' are private. - Methods marked with '+' are public. - Encryption applied to sensitive fields (e.g., biometric$_data, physical_location) using AES-256. – Relationships enforce referential integrity in PostgreSQL.

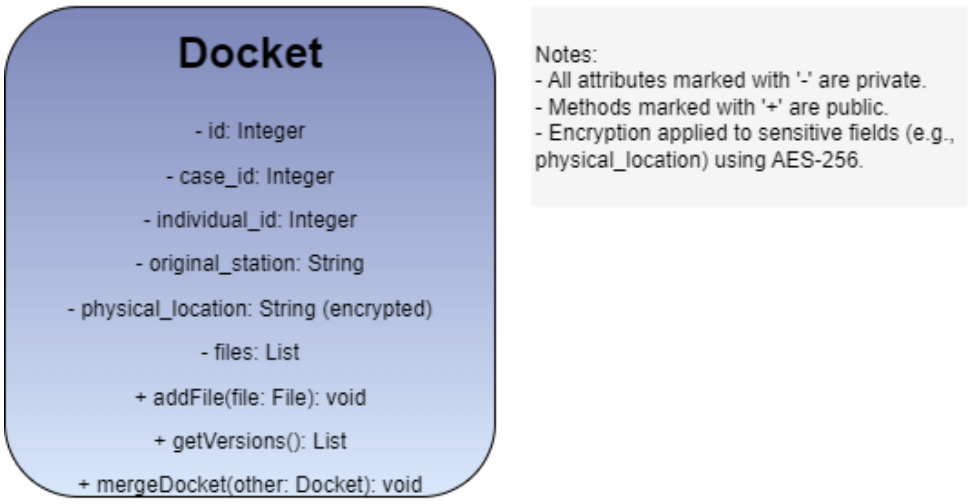

Figure 3: UML Class Diagram for Docket entity.

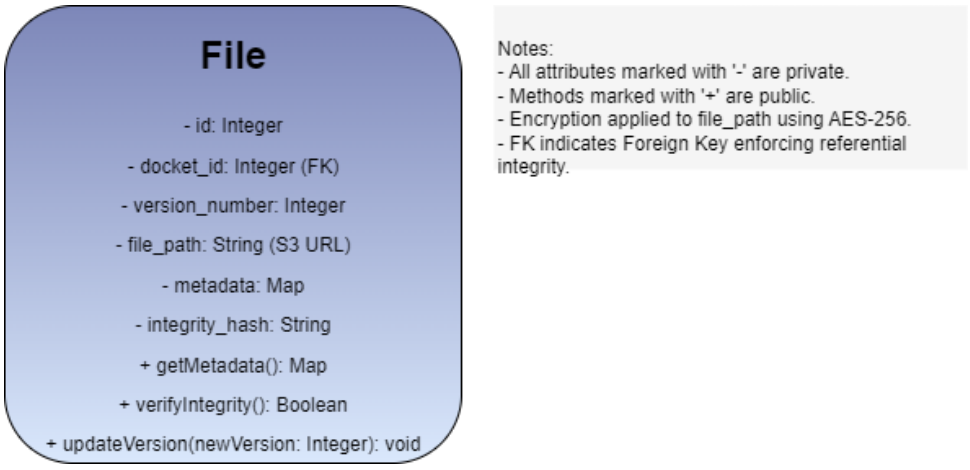

Figure 4: UML Class Diagram for File entity.

3.3 Data Model

3.3.1 Data Model

The data model uses PostgreSQL schemas to ensure relational integrity and compliance.

- **Users Table**: id (PK), username, password$_hash$, $role$, $station_id$, $biometric_data$(*encrypted*).
- **Dockets Table**: id (PK), case$_id$, $individual_id$, $original_station$, $physical_location$(*encrypted*).
- **Files Table**: id (PK), docket$_id$$(FK)$, $version_number$, $file_path(S3URL)$, $metadata$, $integrity_hash$.
- **Audit**$_Logs Table$: $id(PK)$, $user_id(FK)$, $action_type$, $timestamp$, $details(JSONB)$.
- **Sharing**$_Requests Table$: $id(PK)$, $from_station$, $to_station$, $docket_id(FK)$, $status$, $approval_time$.

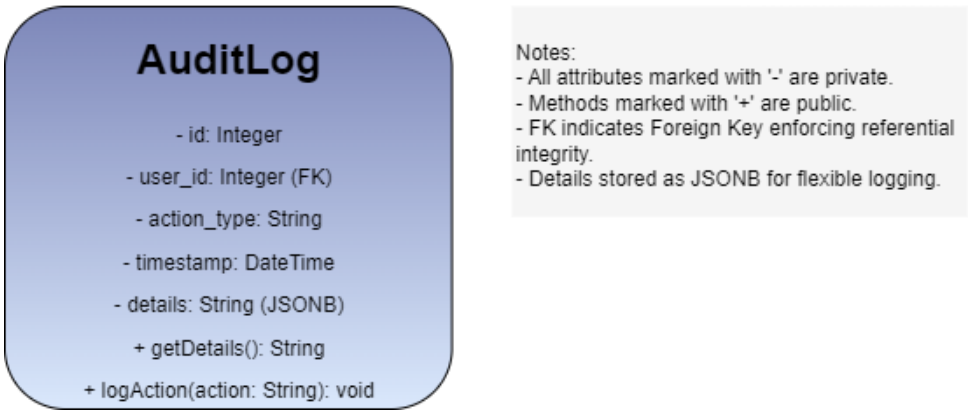

Figure 5: UML Class Diagram for AuditLog entity.

- **Messages Table: id (PK), sender**$_id(FK), receiver_id(FK), content(encrypted), timestamp.$

- **Emails Table: id (PK), docket**$_id(FK), sender, receiver, content(encrypted), timestamp.$

3.3.2 Data Flow

- User authentication flows from frontend to backend, validating against Users table.
- File uploads are encrypted client-side, stored in AWS S3, metadata in Files table.
- Actions trigger inserts into Audit$_Logsforimmutablerecords.$
- Sharing requests update Sharing$_Requestsandtriggernotifications.$
- All data remains in SA AWS region for sovereignty.

[ER Diagram Placeholder: Entity-Relationship diagram showing relationships between tables.]

[Notes] - PK indicates Primary Key. - FK indicates Foreign Key enforcing referential integrity. - Encrypted fields (e.g., biometric$_data, physical_location, content)useAES-256. - Allrelationshipsareimplementedin PostgreSQLwithforeignkeyconstraints.$

4 User Interface Design

4.1 Overview

The UI is a single-page application (SPA) built with React.js, supporting multiple South African languages (English, Afrikaans, Zulu, Xhosa, Sotho) and accessible on desktops and mobiles. Design follows WCAG standards for usability in diverse SAPS environments.

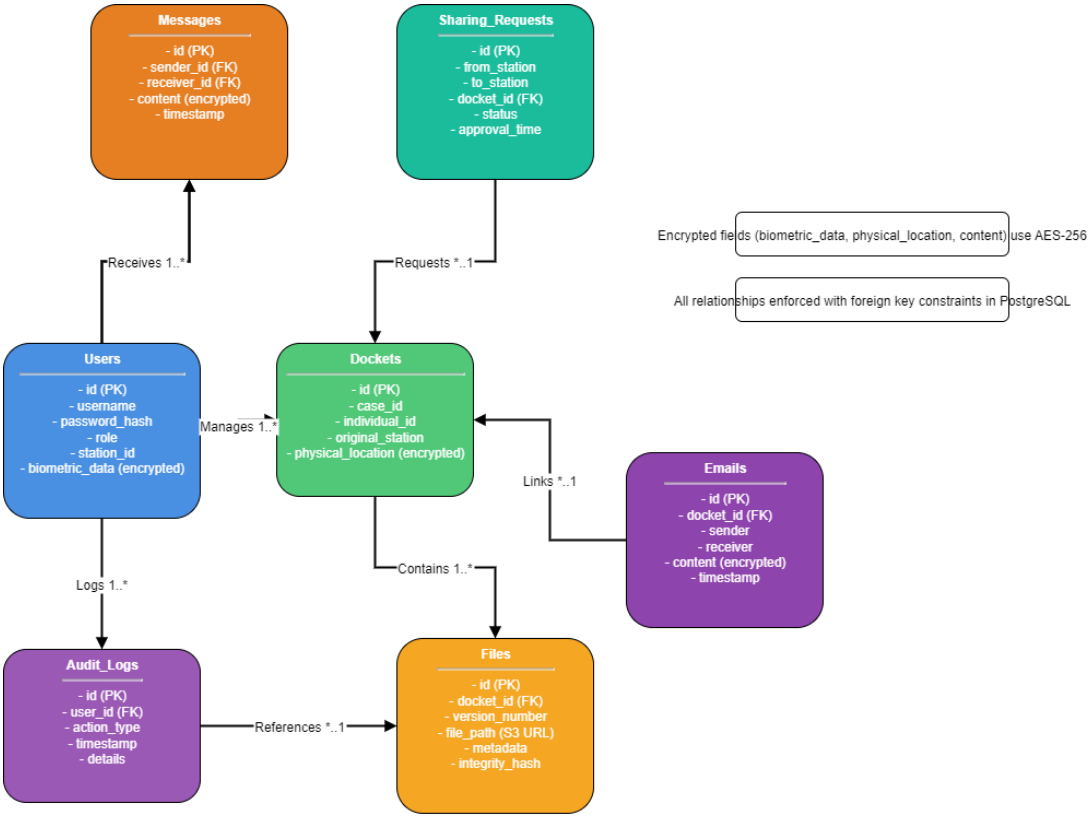

Figure 6: Entity-Relationship diagram showing relationships between tables.

4.2 Screens

- **Login Screen**: Fields for username, password, MFA, biometric prompt.
- **Dashboard**: Widgets for recent files, notifications, quick searches, analytics charts.
- **File Management Screen**: Upload forms, digitization scanner integration, file list with filters.
- **Docket Details Screen**: Version history timeline, merge button, physical location map (if applicable).
- **Sharing Requests Screen**: Request form, pending approvals list, status tracker.
- **Messaging Screen**: Real-time chat, email archive viewer.
- **Audit Reports Screen**: Report generator, log viewer with filters.

[Notes] - Responsive design for mobile and desktop. - Language options include English, Afrikaans, Zulu, Xhosa, Sotho. - Biometric login optional via WebAuthn. - Chart placeholders use bar or line formats. - Tables and lists are paginated or scrollable on mobile.

5 Conclusion

The SFMS design ensures a secure, compliant, and efficient solution for SAPS file management, leveraging modern cloud technologies and robust security measures. Future

iterations will focus on user testing and scalability enhancements.

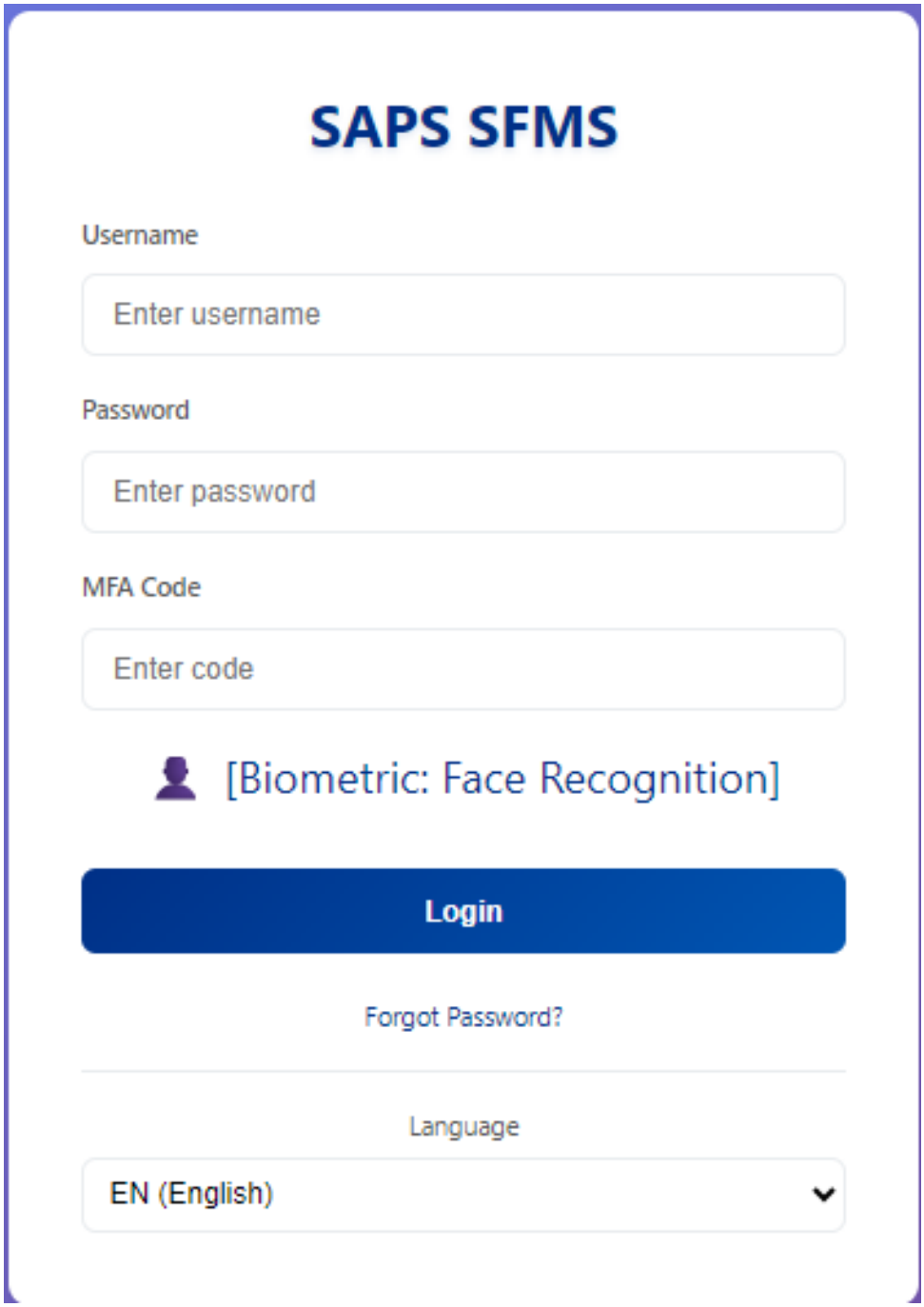

Figure 7: Wireframe sketch of the Login Screen.

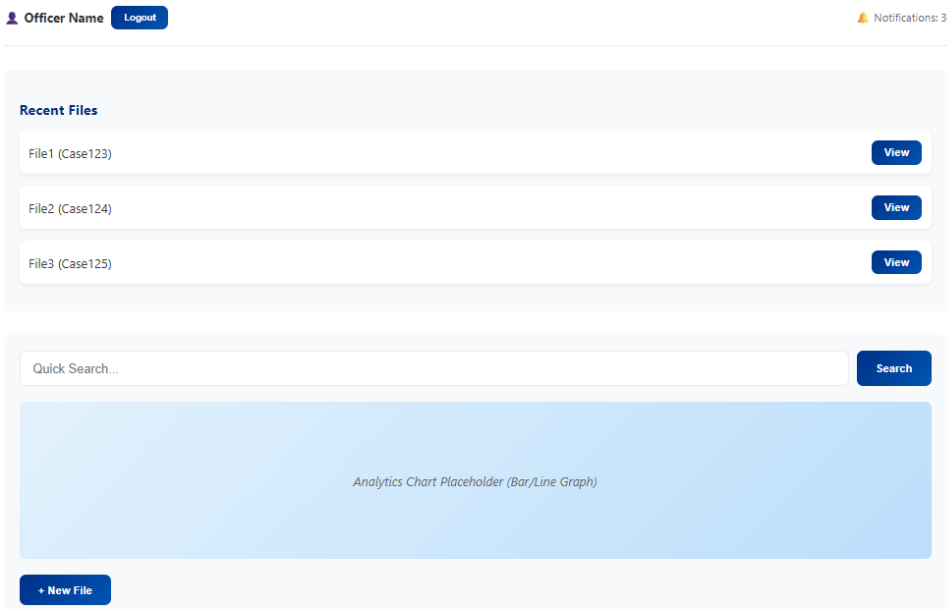

Figure 8: Wireframe sketch of the Dashboard.

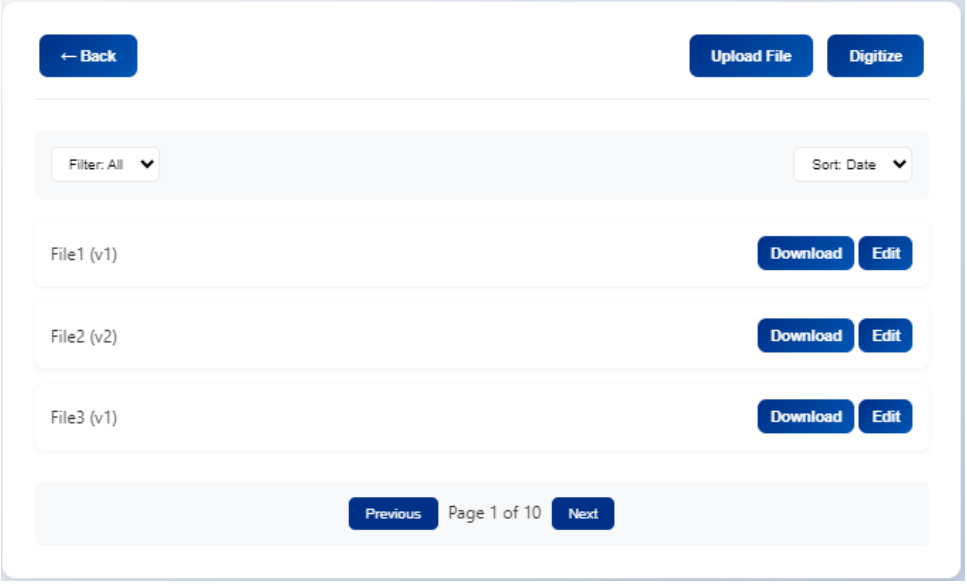

Figure 9: Wireframe sketch of the File Management Screen.

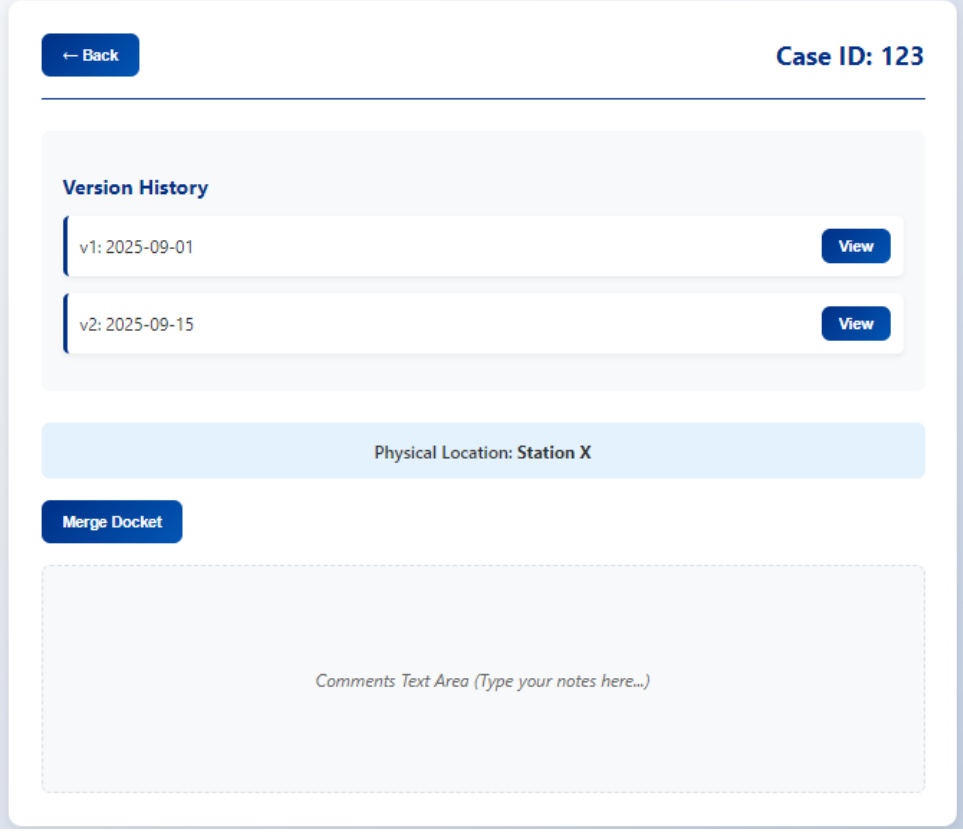

Figure 10: Wireframe sketch of the Docket Details Screen.

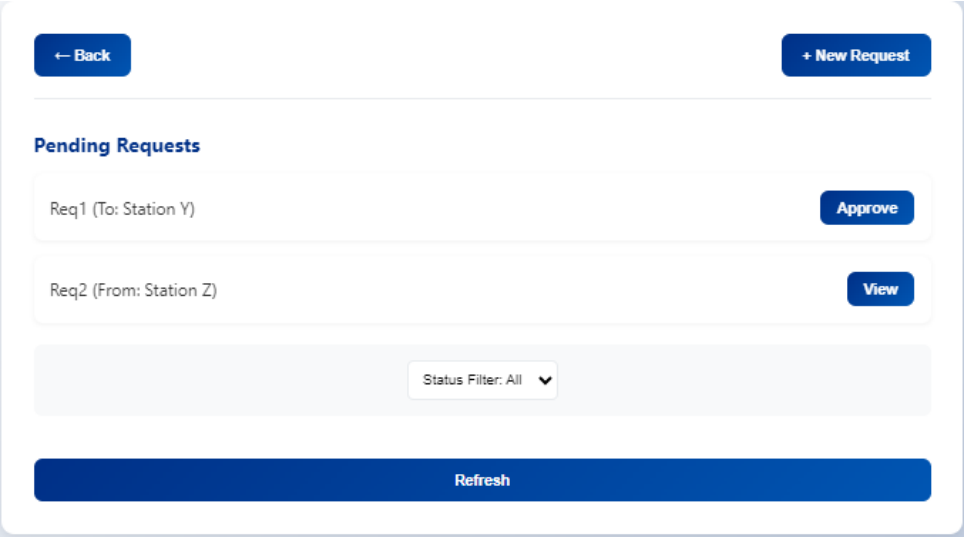

Figure 11: Wireframe sketch of the Sharing Requests Screen.

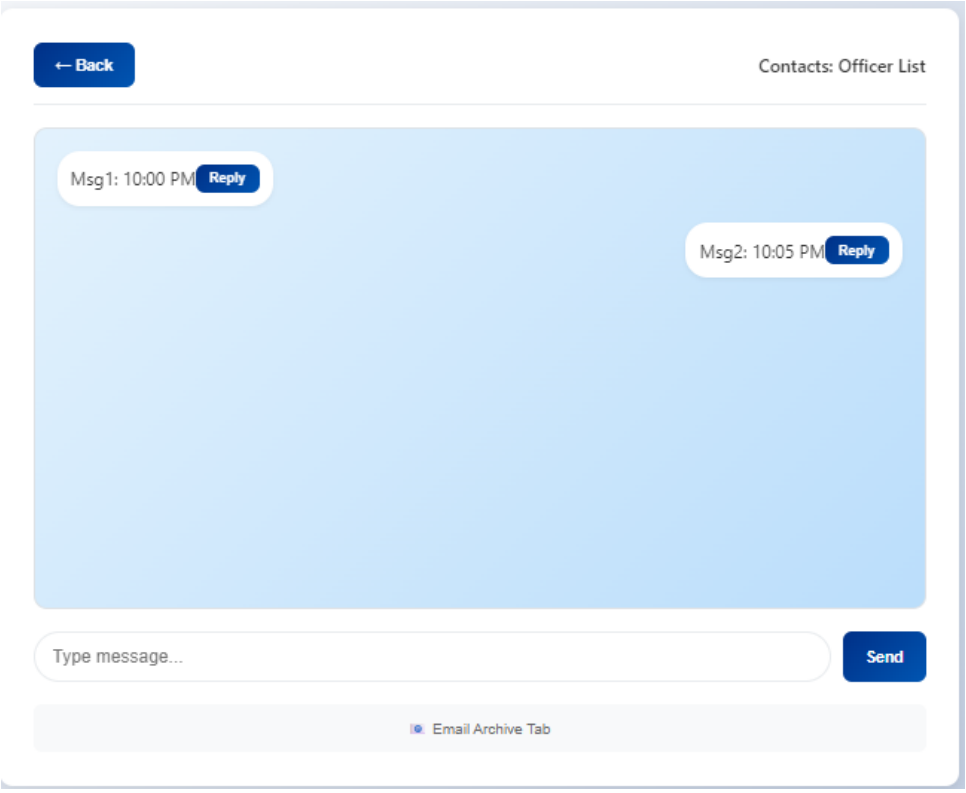

Figure 12: Wireframe sketch of the Messaging Screen.

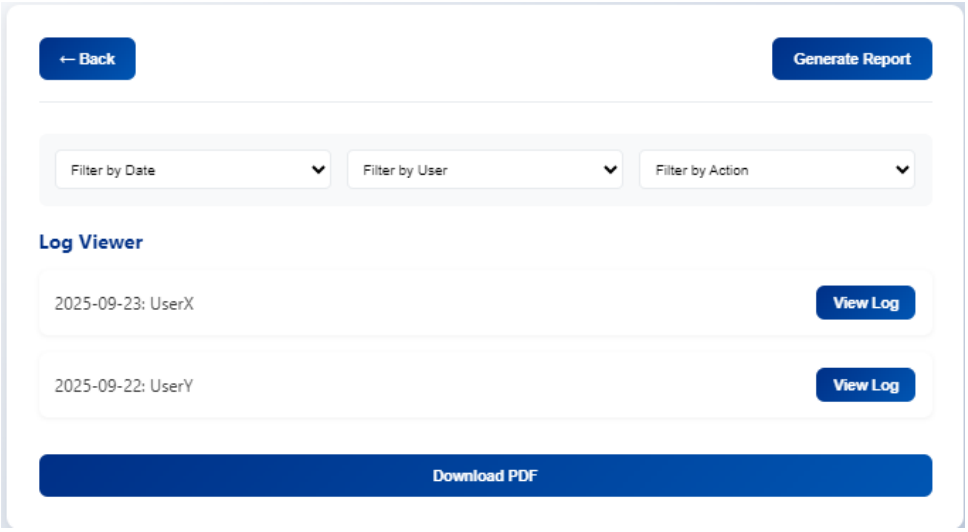

Figure 13: Wireframe sketch of the Audit Reports Screen.