

Military Asset Management System (MAMS)

Project Documentation

Submitted by: Siddharth Bhadaureya

January 13, 2026

Contents

1	Project Overview	2
1.1	Description	2
1.2	Assumptions	2
1.3	Limitations	2
2	Tech Stack & Architecture	2
2.1	Backend: Python (FastAPI)	2
2.2	Frontend: React (Vite + TypeScript)	2
2.3	Database: SQLite (via SQLAlchemy)	2
3	Data Models / Schema	3
4	RBAC Explanation (Role-Based Access Control)	3
5	API Logging & Audit Trails	3
6	Setup Instructions	4
6.1	Prerequisites	4
6.2	Step 1: Backend Setup	4
6.3	Step 2: Frontend Setup	4
6.4	Step 3: Access	4
7	Key API Endpoints	4
8	Login Credentials	4

1 Project Overview

1.1 Description

The Military Asset Management System (MAMS) is a full-stack web application designed to track and manage the inventory of defense assets across multiple military bases. It provides real-time visibility into stock levels, facilitates secure asset transfers between bases, and enforces role-based access control to ensure operational security.

1.2 Assumptions

- The system operates in a trusted offline environment or intranet.
- Asset transfers are instantaneous for simulation purposes.
- All users are pre-authorized and provided credentials securely.

1.3 Limitations

- **Database:** Uses SQLite for portability, which is not suitable for high-concurrency production environments.
- **Authentication:** Token refresh mechanisms are simplified; sessions expire after 30 minutes requiring re-login.

2 Tech Stack & Architecture

2.1 Backend: Python (FastAPI)

Why Chosen: FastAPI provides high performance, automatic API documentation (Swagger UI), and easy implementation of asynchronous database operations. It allows for rapid development of RESTful endpoints.

2.2 Frontend: React (Vite + TypeScript)

Why Chosen: React offers a component-based architecture perfect for dynamic dashboards. Vite ensures lightning-fast build times, and TypeScript adds type safety to prevent runtime errors. Tailwind CSS was used for rapid, professional styling.

2.3 Database: SQLite (via SQLAlchemy)

Why Chosen: SQLite is serverless and zero-configuration, making it ideal for a portable, self-contained submission. SQLAlchemy serves as the ORM to manage relationships cleanly.

3 Data Models / Schema

The system relies on five core entities managed via Relational Database Management System (RDBMS) principles.

1. **Users**: Stores credentials, roles, and assigned base.
2. **BaseLocation**: Represents physical military bases (e.g., HQ Northern Command).
3. **AssetType**: Catalog of items (e.g., INSAS Rifle, Arjun MBT).
4. **Inventory**: The join table linking Bases and Assets with a quantity.
5. **Transactions**: An immutable ledger recording every movement of assets.

4 RBAC Explanation (Role-Based Access Control)

Security is handled via JWT (JSON Web Tokens) containing the user's role and base ID.

Role	Access Level	Capabilities
Commander	Base-Specific Read/Write	Can view dashboard for their specific base. Can initiate transfers <i>from</i> their base only.
Logistics	Transfer Only	Restricted view. Primarily used for moving stock between locations without strategic dashboard access.
Admin	Global Access	Can view system-wide stats and move assets between any bases freely.

Enforcement Method: Every protected API endpoint utilizes a dependency injector `get_current_user` which decodes the JWT header. If the token is invalid or the role lacks permission for the specific action, a `403 Forbidden` error is raised immediately.

5 API Logging & Audit Trails

While standard HTTP logging is handled by the server (Uvicorn), business-logic logging is handled via the **Transactions** table.

Every time an asset is moved, a record is created containing:

- **timestamp**: Exact time of transfer.
- **user_id**: Who performed the action.
- **source_base / dest_base**: The flow of goods.
- **quantity**: Amount moved.

This ensures a permanent audit trail that cannot be deleted by standard users.

6 Setup Instructions

6.1 Prerequisites

- Python 3.10+
- Node.js 18+

6.2 Step 1: Backend Setup

```
1 cd backend
2 pip install fastapi uvicorn sqlalchemy pydantic python-jose passlib
   python-multipart
3 python seed.py
4 uvicorn main:app --reload --port 8001
```

6.3 Step 2: Frontend Setup

```
1 cd frontend
2 npm install
3 npm run dev
```

6.4 Step 3: Access

Open browser to <http://localhost:5173>

7 Key API Endpoints

- **POST /token:** Authenticates user and returns JWT access token.
- **GET /stats:** Returns inventory levels for the user's assigned base.
- **POST /transfer:** Executes an ACID-compliant transaction to move assets.
- **GET /assets:** Fetches the catalog of available weapon types.

8 Login Credentials

The database is pre-seeded with the following accounts for testing:

Role	Username	Password	Assigned Base
Commander	cmdr_north	pass123	HQ Northern Command (J&K)
Logistics	log_west	pass123	INS Hamla (Mumbai)
Admin	admin	admin123	Global