Military Asset Management System

1. Project Overview

The Military Asset Management System (MAMS) is a full-stack web application designed to manage military assets efficiently. It provides modules for asset tracking, assignments, purchases, transfers, notifications, and user management. The system enforces strict access control and transaction logging to ensure security and accountability.

Assumptions:

- All users are authenticated before accessing protected resources.
- Asset data is updated in real-time across modules.
- The system is deployed in a secure, access-controlled environment.
- Users have unique roles and permissions.

Limitations:

- No integration with external military or government systems.
- No real-time GPS tracking of assets.
- Notification system is limited to in-app alerts (no SMS/email integration).
- Designed for moderate scale; may require optimization for very large deployments.

2. Tech Stack & Architecture

Backend: Node.js + Express.js

- Chosen for its non-blocking I/O, scalability, and rich ecosystem.
- RESTful API design for modularity and ease of integration.

Frontend: React.js

- Enables fast, interactive UIs and component-based architecture.
- Uses Context API for state management (Auth, Notifications).

Database: MongoDB

- Flexible schema for evolving asset data.
- Document-based storage suits hierarchical and relational data.

Architecture Diagram:

[Client (React)] <-> [Express API] <-> [MongoDB]

- Authentication via JWT tokens.
- Middleware for RBAC and logging.
- Modular route structure for maintainability.

3. Data Models / Schema

User:

- _id, name, email, passwordHash, role (admin/officer/clerk), createdAt

Asset:

- _id, name, type, serialNumber, status, location, createdAt

Assignment:

- _id, assetId, userId, assignedTo (unit/user), assignedBy, assignedAt, status

Purchase:

- _id, assetId, purchasedBy, purchaseDate, vendor, cost

Transfer:

- _id, assetId, fromUnit, toUnit, transferredBy, transferredAt, status

Notification:

- _id, userId, message, type, createdAt, read

Balance:

- _id, assetType, unit, quantity

Relationships:

- One user can have many assignments.
- One asset can have many assignments, purchases, and transfers.
- Notifications are linked to users and asset events.

Sample MongoDB Schema (Asset):

```
{
  _id: ObjectId,
  name: "Rifle A1",
  type: "Weapon",
  serialNumber: "SN12345",
  status: "Active",
  location: "Unit HQ",
  createdAt: ISODate
}
```

4. RBAC Explanation

Roles:

- **Admin:** Full access (CRUD on all modules, user management).
- **Officer:** Can view, assign, transfer assets; limited user management.
- **Clerk:** Can view assets, update status, create notifications.

Access Matrix:

Module	Admin	Officer	Clerk
Assets	CRUD	R/U	R/U
Assignments	CRUD	R/U	R
Purchases	CRUD	R	R
Transfers	CRUD	R/U	R
Users	CRUD	R	-
Notifications	CRUD	R/U	R/U

Enforcement:

- Middleware checks JWT and role before route access.

- Example (Express):

```
function authorize(roles = []) {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Forbidden' });
    }
    next();
  };
}
```

- Frontend hides UI elements based on user role.

5. API Logging

- All API requests and responses are logged via Express middleware.
- Logs include: userId, endpoint, method, timestamp, status, payload.

- Example log entry:

```
{
  userId: "abc123",
  endpoint: "/api/assets",
  method: "POST",
  timestamp: "2025-08-21T10:00:00Z",
  status: 201,
  payload: { name: "Rifle A1" }
}
```

- Logs stored in a dedicated MongoDB collection or server log files.
- Used for audit, debugging, and compliance.

6. Setup Instructions

Backend:

1. Navigate to `server/`.
2. Run `npm install` to install dependencies.
3. Create `.env` file with:
   - MONGODB_URI=mongodb://localhost:27017/mams

  - JWT_SECRET=your_jwt_secret

4. Start server: `npm start`

Database:

- Install MongoDB (https://www.mongodb.com/try/download/community)
- Start MongoDB service: `mongod`
- Seed sample data: `node scripts/seed-sample-data.js`

Frontend:

1. Navigate to `client/`.
2. Run `npm install`.
3. Start frontend: `npm start`
4. Access at `http://localhost:3000`

7. API Endpoints (Sample)

Auth:

- `POST /api/auth/login` - Login, returns JWT
- `POST /api/auth/register` - Register new user (admin only)

Assets:

- `GET /api/assets` - List all assets
- `POST /api/assets` - Add asset (admin)
- `PUT /api/assets/:id` - Update asset
- `DELETE /api/assets/:id` - Delete asset

Assignments:

- `GET /api/assignments` - List assignments
- `POST /api/assignments` - Assign asset

Transfers:

- `POST /api/transfers` - Transfer asset
- `GET /api/transfers` - List transfers

Notifications:

- `GET /api/notifications` - Get notifications
- `POST /api/notifications` - Create notification

Sample Request (Assign Asset):

```
POST /api/assignments
Headers: Authorization: Bearer <token>
Body: {
  assetId: "123",
  assignedTo: "Unit B",
  assignedBy: "Officer X"
}
```