**Software Design Document**

**SH EXPRESS Manufacture Management System**

Version: 1.0

**Team Members:**

Shahil Hossain, Aditta Morshed

**Supervisor:** Nazmul Hasan

Presidency University

**Table of Contents:**

1.0. Introduction

1.1. Purpose

1.2. Scope

1.3. References

1.4. Overview

2.0. Diagram

1.1. Use-case diagram

1.2. BPMN diagram

1.3. ERD diagram

3.0. System Architecture

3.1. Overview

3.2. High-Level Architecture

4.0. Detailed Design

4.1. Authentication Module

4.2. Dashboard Module

4.3. Quality Control Module

5.0. Database Design

5.1. MongoDB Collections

6.0. Data Design

6.1. Data Models

7.0. Interface Design

6.1. User Interfaces

6.2. API Interfaces

8.0. Security Considerations

9.0. Error Handling

10.0. Non-Functional Requirements

11.0. Dependencies

12.0. Deployment

13.0 Appendix: Code Summary

14.0 Future Enhancements

15.0. Conclusion

**1.0 Introduction**

**1.1 Purpose :**

This document describes the architecture and design of the SH Express Manufacturing Management System (MMS), a web-based application designed to manage manufacturing operations, including inventory, quality control, and logistics.

**1.2. Scope :**

The system contains the full description of the design of SH Express MMS that includes:

* User authentication and authorization
* Inventory CRUD operations
* Quality control tracking
* Logistics tracking (shipments, status updates, Excel export)
* Dashboard with key performance indicators

The basic setup includes a web server that operates on a client-server model, with the basic pages created using HTML, CSS, and JavaScript.

**1.3. References:**

*Software Design Description (SDD) sample* (2014) *SlideShare*. Slideshare. Available at: https://www.slideshare.net/slideshow/sdd-software-des-sample/34428403#20 (Accessed: 12 April 2025).

‌

**1.4. Overview :**

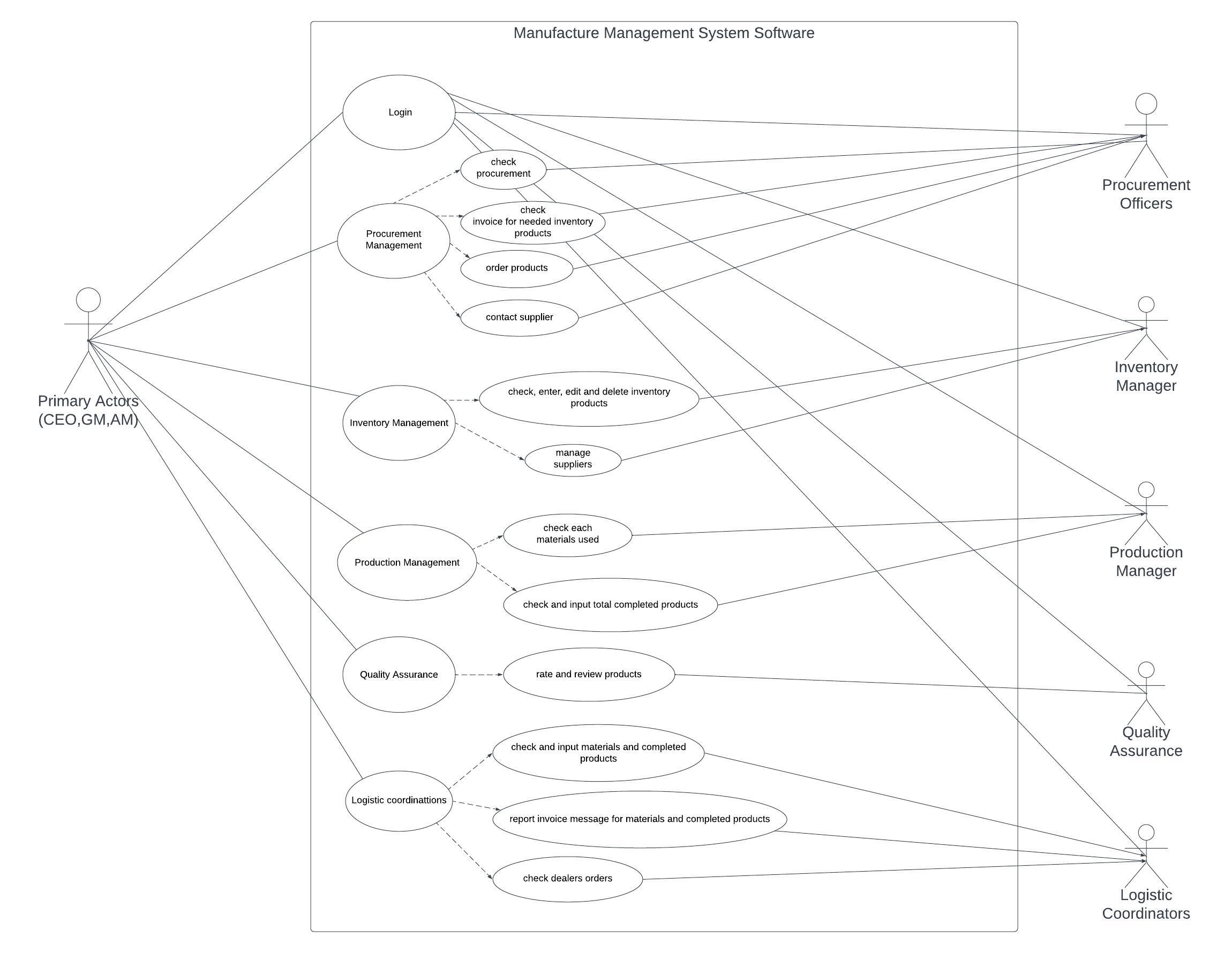
The remaining contents that will be discussed are given below:

* Section 2 provides the 3 diagrams that are built for this MMS.
* Section 3 shows the system architecture of the frontend codes, backend codes, and libraries.
* Section 4 provides the detailed design of each module of the MMS.
* Section 5 provides the database designs of MongoDB that are used for the MMS.
* Section 6 provides the data designs with data models.
* Section 7 shows 2 interface designs (API, User) and figures of them.
* Section 8 provides security considerations for this system.
* Section 9 provides the error-handling part of the MMS.
* Section 10 provides non-functional requirements.
* Section 8 provides dependencies between modules and frontend, backend codes, etc.
* Section 12 provides deployment of the system.
* Section 13 shows the code summary of the MMS.
* Section 14 provides the future works that will be done for MMS.
* Section 15 provides the conclusion of this document.

**2.0. Diagrams**

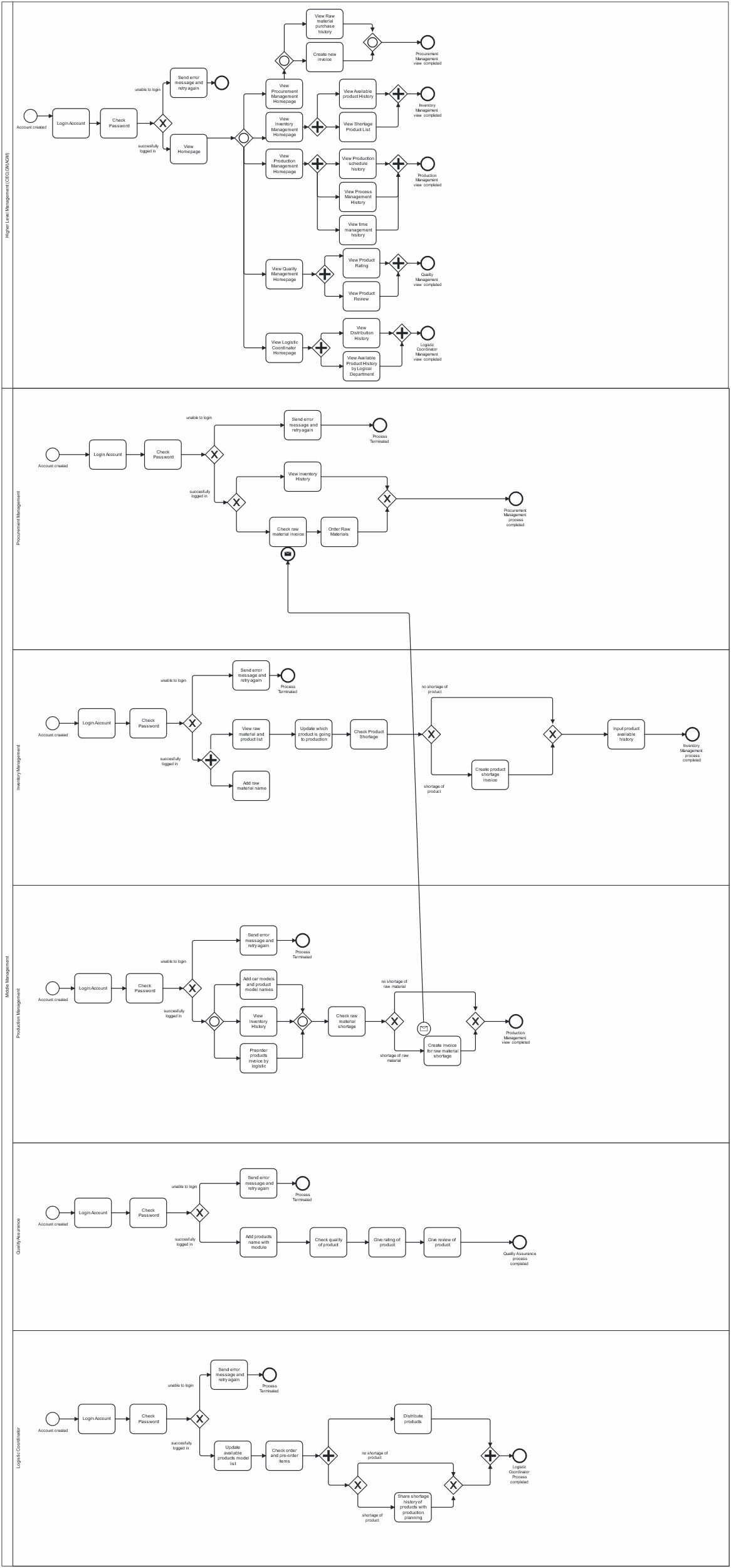
**2.1. Use-Case Diagram:**

To get the idea of MMS, a use-case diagram is built. It is given below:

****

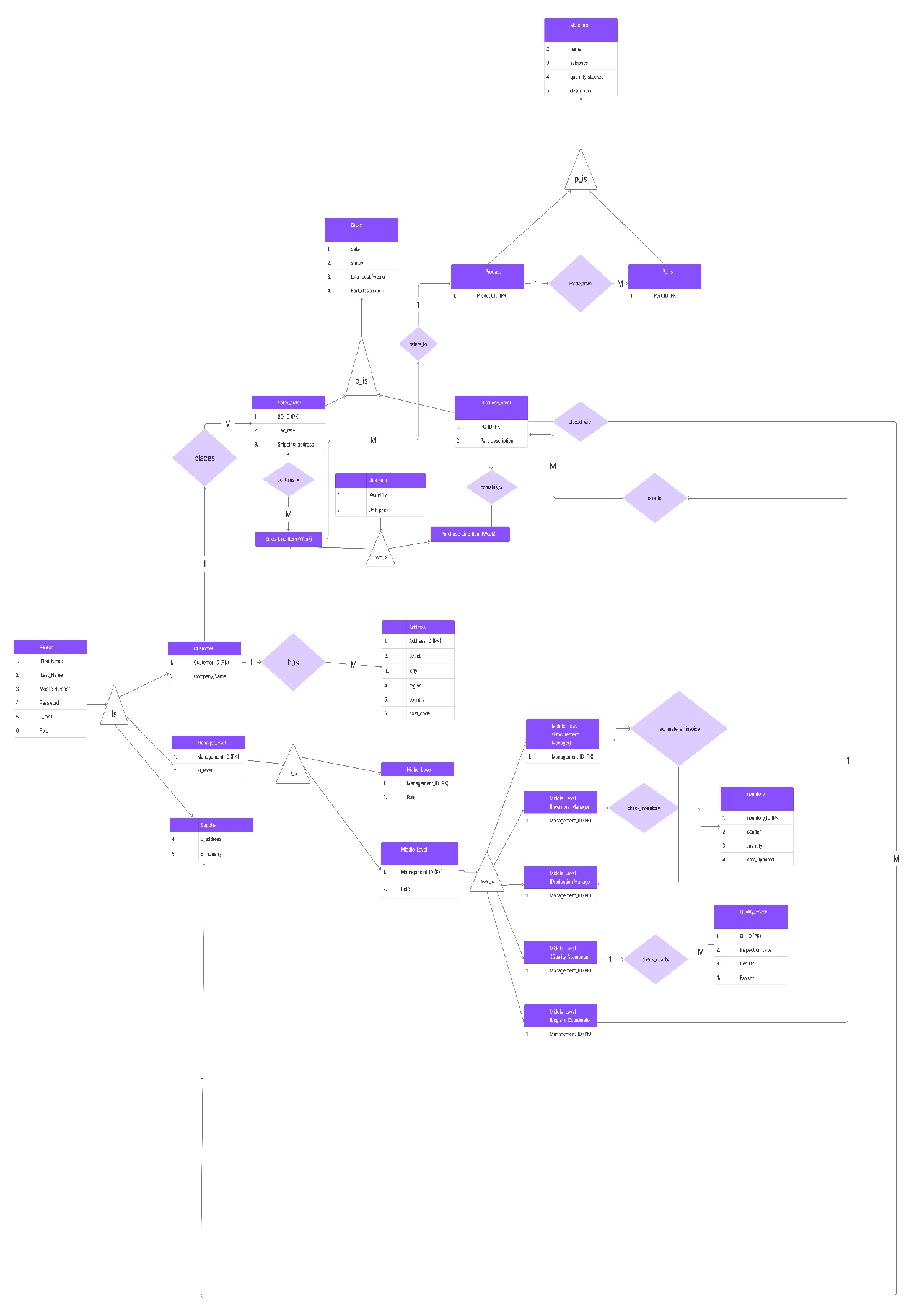
**2.2. BPMN Diagram:**

This diagram easily provides the process of each management level and their work, and their relationship with each other.

****

**2.3. ERD Diagram:**

This diagram shows the most important classes and their attributes needed for the system. It also shows their relationship and generalization of the MMS.

****

**3.0. System Architecture**

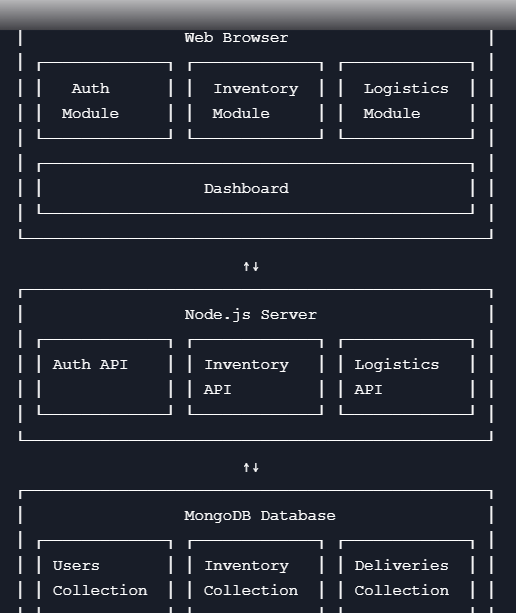
**3.1. Overview :**

The system follows a client-server architecture with-

* **Frontend**: HTML5, CSS3, JavaScript with Bootstrap for responsive design
* **Backend**: Node.js with Express.js
* **Database**: MongoDB
* **Authentication**: JWT tokens
* **Excel Export:** ExcelJS

**3.2. High-Level Architecture:**

This architecture shows a clear view of how the MMS is built and its structure.

****

**4.0. Detailed Design**

**4.1. Authentication Module:**

**i. Frontend Components**:

* **Login Form** (loginForm in index.html)
* **Signup Form** (signupForm in index.html)
* **JWT Storage** (localStorage in script.js)

**ii. Key Functions**:

* POST /api/auth/signup: User registration
* POST /api/auth/login: User authentication
* JWT middleware for route protection
* logout(): Ends user session

**4.2. Dashboard Module:**

**i. Frontend Components:**

**a. Inventory Management**:

* + Add/Edit/Delete items (productModal)
  + Chart visualization (inventoryChart using Chart.js)

**b. Logistics Management**:

* + Dynamic district selection (updateDistricts() in logistics.js)
  + Shipment status tracking (status-passed, status-failed CSS classes)
  + Excel export (downloadExcel() in logistics.js)

**ii. Key Functions**:

* GET /api/inventory: Get all items
* POST /api/inventory: Create new item
* PUT /api/inventory/:id: Update item
* DELETE /api/inventory/:id: Delete item

**4.3. Quality Control Module:**

**i. Components**:

* Inspection form
* Defects tracking table
* Status indicators

**ii. Key Functions**:

* initializeQuality(): Loads quality data
* editInspection(index): Loads inspection for editing
* deleteInspection(index): Removes inspection record
* updateQualityDashboard(): Updates quality metrics

## ****5.0. Database Design****

### ****5.1 MongoDB Collections:****

#### **1.**Users**Collection**

javascript

{

\_id: ObjectId,

email: String (unique),

password: String (hashed),

}

#### **2.**Inventory**Collection**

javascript

{

\_id: ObjectId,

name: String,

quantity: Number,

type: String ("kg" or "pieces"),

userId: ObjectId (ref: "User"),

lastUpdated: Date,

}

#### **3.**Deliveries**Collection**

javascript

{

\_id: ObjectId,

orderId: String,

division: String,

district: String,

address: String,

status: String ("Scheduled", "In Transit", "Delivered"),

date: String,

}

**6.0. Data Design**

**6.1. Data Models:**

**// User Model**

**{**

**email: String, // Unique user identifier**

**password: String // Hashed password**

**}**

**// Inventory Model**

**{**

**name: String, // Product name**

**quantity: Number, // Current stock**

**type: String, // Measurement type**

**userId: ObjectId // Owner reference**

**}**

**// Delivery Model**

**{**

**orderId: String, // Unique order identifier**

**division: String, // Administrative division**

**district: String, // Local district**

**address: String, // Full delivery address**

**status: String, // Delivery status**

**date: String // Delivery date**

**}**

**7.0. Interface Design**

**7.1. User Interfaces:**

1. **Authentication Screens**
   * Login form
   * Signup form
2. **Main Dashboard**

* Welcome banner
* Shipment status counters
* Navigation menu

1. **Inventory Management**

* Product table with CRUD operations
* Add/edit modal
* Chart visualization

1. **Quality Control**
   * Inspection form
   * Defects table
2. **Logistics**

* Delivery form with district selector
* Shipments table
* Excel export button

**7.2. API Interfaces (Future Implementation):**

**javascript**

**// Authentication**

**POST /api/auth/signup - { email, password }**

**POST /api/auth/login - { email, password }**

**// Inventory**

**GET /api/inventory - Returns all items**

**POST /api/inventory - { name, quantity, type }**

**PUT /api/inventory/:id - { name, quantity, type }**

**DELETE /api/inventory/:id**

**// Logistics**

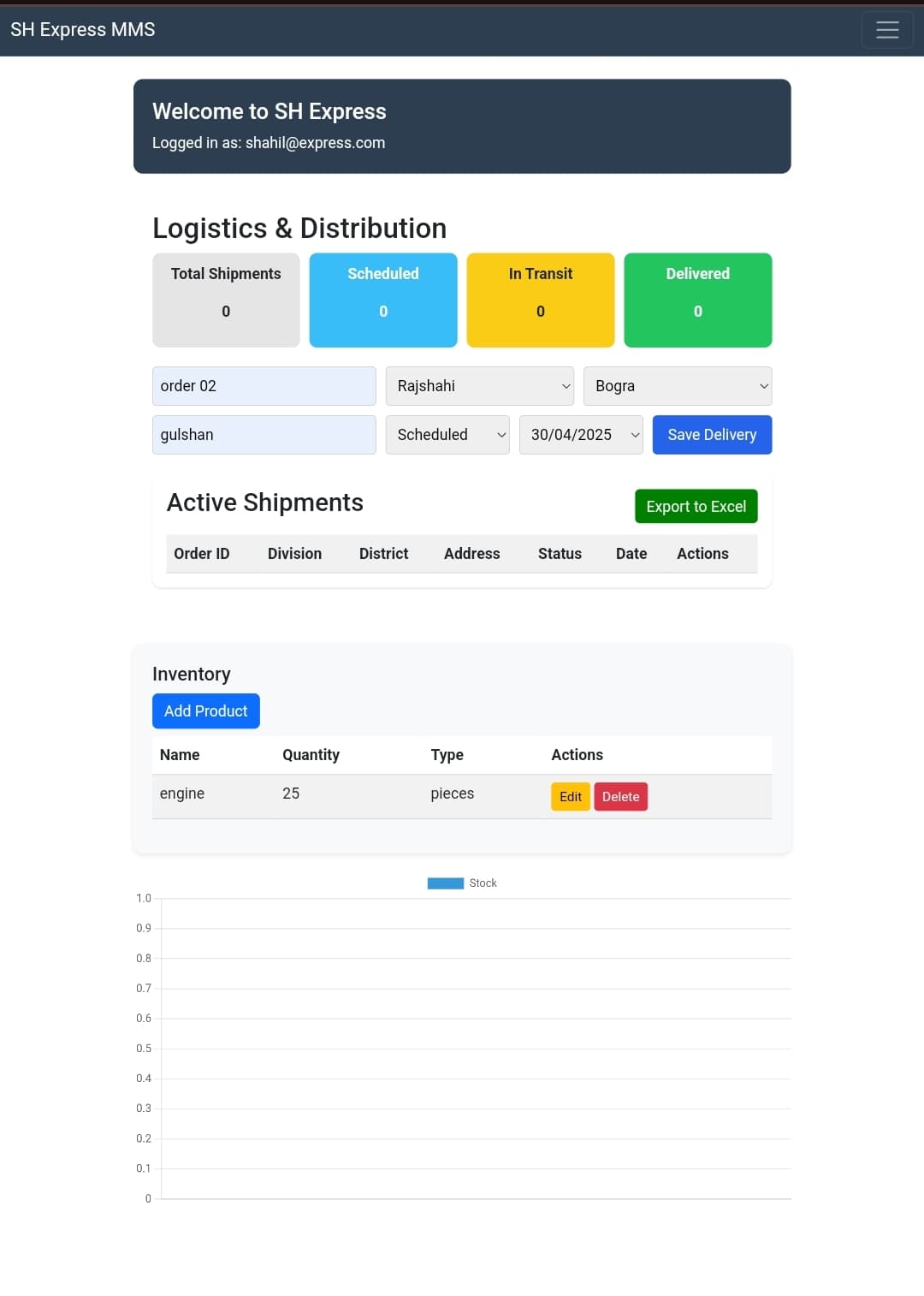
**GET /api/logistics - Returns all deliveries**

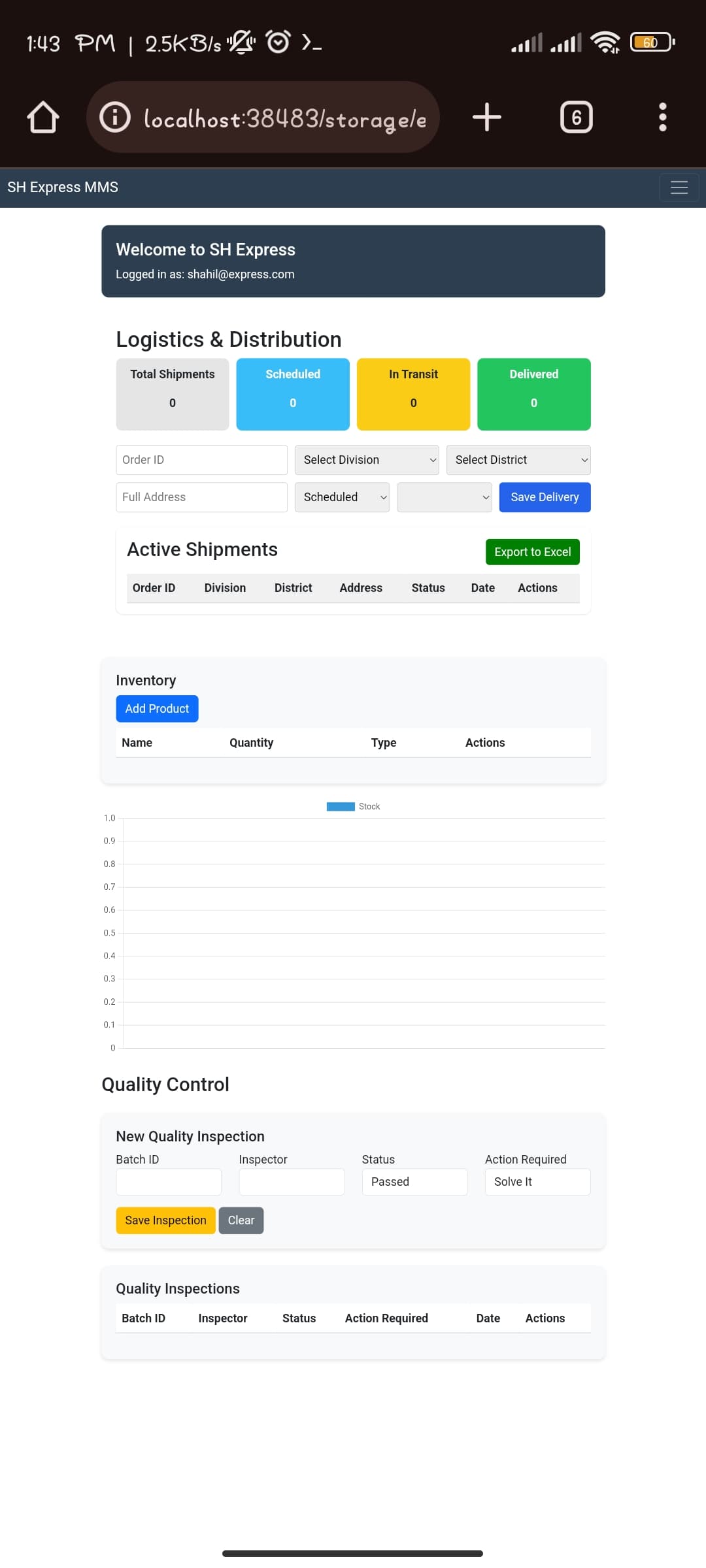
**POST /api/logistics - { orderId, division, district, address, status, date }**

**GET /api/logistics/export/excel - Returns Excel file**

**7.3. Figures Of The Interfaces:**

The interface will use the same design scheme and colors for MMS. The exact pages are placed here:

****

****

**8.0. Security Considerations**

1. **Authentication**:
   * JWT with expiresIn: '1d' (24-hour expiry).
   * Password hashing (bcryptjs).
2. **Authorization**:
   * Protected routes (auth middleware checks JWT).
3. **Input Validation**:
   * Required fields in Delivery and Inventory models.
   * Frontend form validation (required attribute in HTML).

**9.0. Error Handling**

| **Scenario** | **Response** |
| --- | --- |
| Invalid JWT | 401 Unauthorized |
| Missing required fields | 400 Bad Request |
| Database error | 500 Internal Server Error |
| Invalid login credentials | 400 Bad Request |

**10.0. Non-Functional Requirements**

**i. Performance:**

* Client-side rendering for responsive UI
* MongoDB indexing for fast queries
* Excel streaming for large exports

**ii. Security:**

* JWT authentication
* Password hashing with bcrypt
* Protected API routes
* CORS configuration

**iii. Usability:**

* Responsive Bootstrap layout
* Form validation
* Clear status indicators
* Excel export for data portability

**11.0. Dependencies**

**i. Frontend Libraries:**

* + Bootstrap 5.3 (CSS/JS)
  + Chart.js
  + Axios (for future API calls)

### ii. Backend:

* Express.js
* Mongoose
* Bcrypt.js
* Jsonwebtoken
* ExcelJS

### iii. Database:

* MongoDB

**12.0. Deployment**

**12.1 Environment Variables:**

env

MONGO\_URI=mongodb+srv://<user>:<password>@cluster.mongodb.net/SHExpress

JWT\_SECRET=your\_jwt\_secret\_key

PORT=5000

**12.2 Steps to Run:**

1. Install dependencies:

bash

npm install express mongoose bcryptjs jsonwebtoken cors dotenv exceljs

2. Start the server:

bash

node server.js

3. Access frontend via index.html.

## 13.0. Appendix: File Structure

client/

├── index.html # Main application page

├── style.css # Custom styles

├── script.js # Main application logic

└── logistics.js # Logistics-specific functions

server/

├── server.js # Main server entry point

├── models/ # Database models

│ ├── User.js

│ ├── InventoryItem.js

│ └── Delivery.js

├── routes/ # API routes

│ ├── auth.js

│ ├── inventory.js

│ └── logistics.js

└── middleware/ # Authentication middleware

└── auth.js

## ****14.0. Future Enhancements****

1. **Pagination** for large datasets.
2. **Email notifications** for shipment updates.
3. **Role-based access control** (Admin, Manager, User).
4. **Real-time updates** (WebSockets for live tracking).

### ****15.0. Conclusion****

This SDD provides a **comprehensive design** of the SH Express Management System, covering:

**1. Backend API** (Node.js/Express)  
**2. Frontend UI** (HTML/CSS/JS)  
**3. Database Schema** (MongoDB)  
**4. Security & Error Handling**