ABDOULIE KUJABI 22216033
INTERNET AND WEB II Full Stack Assignment

INVENTORY MANAGEMENT SYSTEM

1. PROJECT OVERVIEW

Description: The Inventory Management System is a full-stack web application built with Angular 17+ frontend and Express.js backend. It provides comprehensive inventory tracking, sales management, staff management, and dashboard analytics capabilities. The system is designed to help businesses efficiently manage their product inventory, track sales, monitor stock levels, and generate reports.

Key Features:
• Real-time inventory tracking
• Sales management and reporting
• Staff management
• Role-based access control (RBAC)
• User authentication with JWT
• Dashboard with analytics
• Product management with low stock alerts
• Comprehensive API with Swagger documentation
• Security features (CORS, Rate limiting, Helmet)

Live URL: https://inventory-system-comstruck.vercel.app/
Live backend api: https://inventory-backend-0h6z.onrender.com/

2. TECHNOLOGY STACK

Backend Technologies:
• Runtime: Node.js 20+
• Framework: Express.js with TypeScript
• Database: MongoDB with Mongoose ODM
• Authentication: JWT (JSON Web Tokens)
• Password Hashing: bcrypt
• Validation: Joi schema validation
• Logging: Winston logger
• Security: Helmet, CORS, express-rate-limit

• API Documentation: Swagger/OpenAPI
• Server Port: 3000

Frontend Technologies:
• Framework: Angular 17+
• Language: TypeScript (Strict mode)
• Styling: SCSS
• Architecture: Standalone components
• HTTP Client: Angular HttpClient
• State Management: Angular Services
• Route Guards: Role-based access control
• Authentication: JWT token handling with interceptors
• UI Port: 4200

Database:
• MongoDB (Local or MongoDB Atlas)
• Collections: Users, Products, Sales, Staff

Deployment:
• Frontend: Vercel
• Backend: Railway.app
• Database: MongoDB Atlas

3. PROJECT STRUCTURE

Root Directory:
• /backend - Express.js TypeScript backend application
• /frontend - Angular 17+ frontend application
• /docs - Documentation files
• package.json - Root package configuration
• README.md - Project overview
• vercel.json - Vercel deployment configuration
• railway.json - Railway deployment configuration

Backend Structure:
/backend/src/
• index.ts - Application entry point
• app.ts - Express app configuration
• /config - Configuration files

• /controllers - Route controllers for business logic
• /models - Mongoose schema models
• /routes - API route definitions
• /services - Business logic services
• /middleware - Express middleware (auth, validation, error handling)

Frontend Structure:
/frontend/src/
• /app - Angular application modules
• /core - Core services, guards, interceptors
• /features - Feature modules (products, sales, staff, dashboard)
• /shared - Shared components and utilities
• styles.scss - Global styles
• main.ts - Angular entry point

4. INSTALLATION & SETUP

Prerequisites:
• Node.js 20+ and npm 10+
• MongoDB (local or MongoDB Atlas)
• Git

Install Dependencies (Root):
npm run install:all

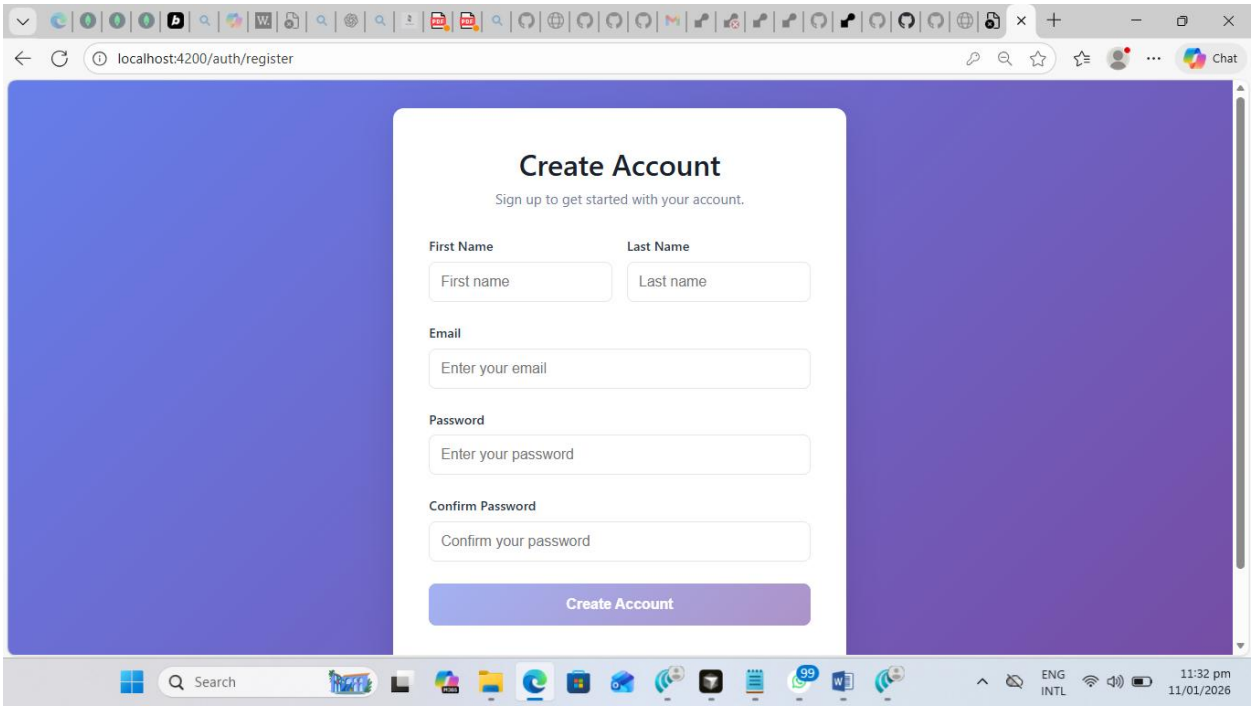
Backend Setup:
1. Navigate to backend directory: cd backend
2. Create .env file from .env.example: cp .env.example .env
3. Configure environment variables (PORT, MONGODB_URI, JWT_SECRET, etc.)
4. Start development server: npm run dev
5. Server runs on http://localhost:3000
6. Access API docs: http://localhost:3000/api-docs

Frontend Setup:
1. Navigate to frontend directory: cd frontend
2. Install dependencies: npm install
3. Start development server: npm start
4. Application runs on http://localhost:4200

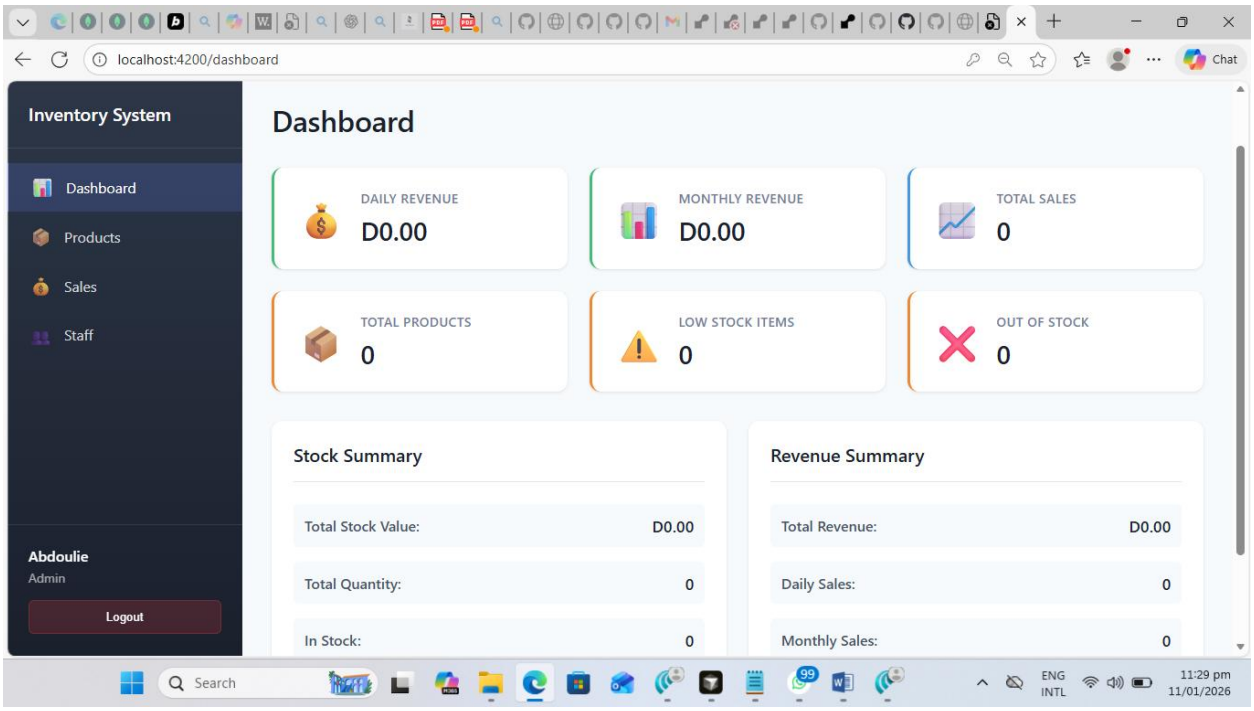
5. USER ROLES & PERMISSIONS

Role Hierarchy:
• ADMIN - Full access to all features and system management
• MANAGER - Dashboard, Sales, Staff management access
• WAREHOUSE - Product management and inventory control
• USER - Basic read-only access (default role)

Role-Based Features:
ADMIN:
- User management
- System configuration
- Full analytics access
- Report generation

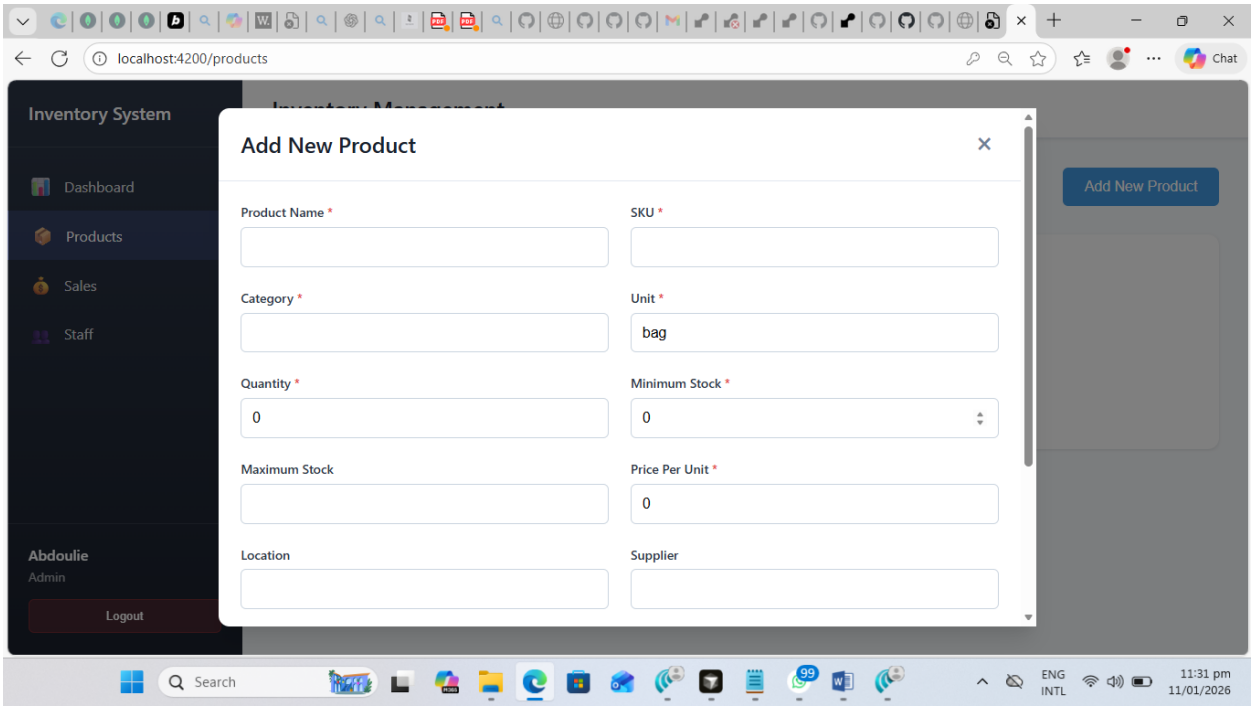


MANAGER:
- View dashboard
- Manage sales
- Manage staff
- Generate reports



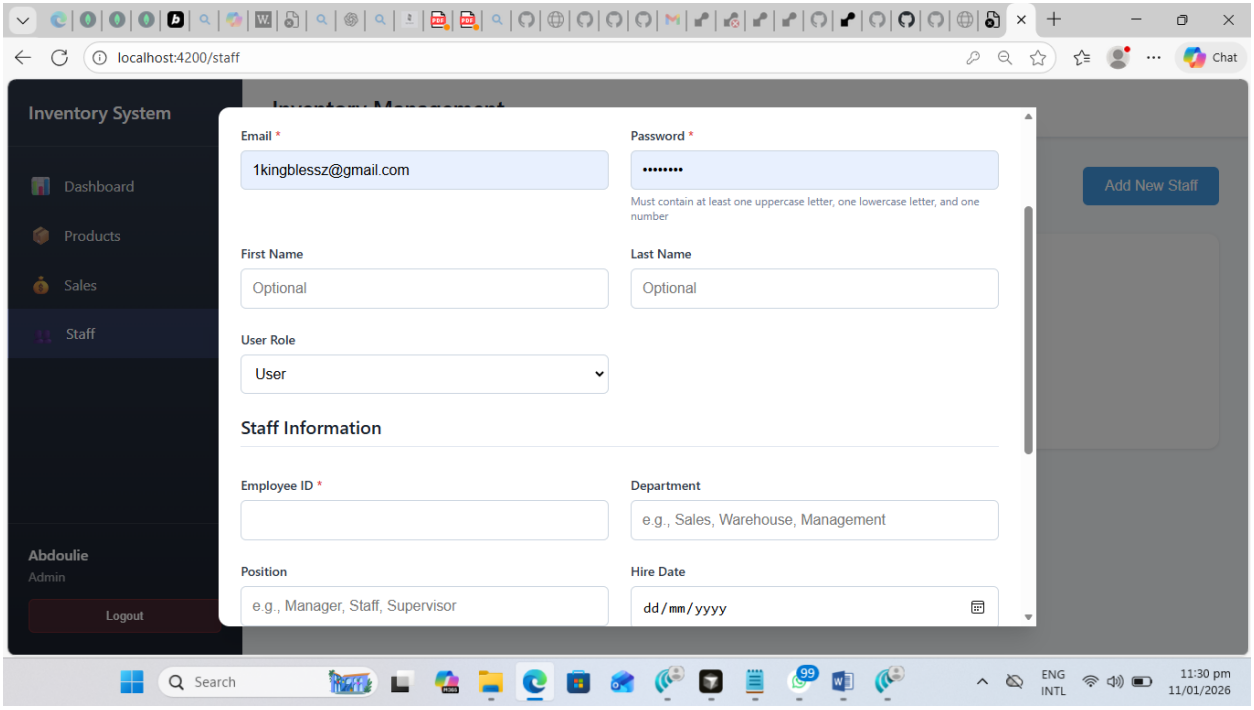
WAREHOUSE:

- Manage products
- Update inventory
- Track stock levels



USER:

- View inventory
- View basic reports



6. API ENDPOINTS

Authentication Endpoints:
POST /api/auth/register - Register new user
POST /api/auth/login - User login
POST /api/auth/refresh - Refresh access token
POST /api/auth/logout - Logout user

Product Endpoints:
GET /api/products - Get all products (paginated)
GET /api/products/:id - Get product by ID
POST /api/products - Create new product
PUT /api/products/:id - Update product
DELETE /api/products/:id - Delete product
GET /api/products/low-stock - Get low stock products

Sales Endpoints:
GET /api/sales - Get all sales (paginated)
POST /api/sales - Create new sale
GET /api/sales/:id - Get sale by ID
GET /api/sales/daily-report - Get daily sales report
GET /api/sales/monthly-report - Get monthly sales report

Staff Endpoints:
GET /api/staff - Get all staff (paginated)
POST /api/staff - Create staff member
PUT /api/staff/:id - Update staff member
DELETE /api/staff/:id - Delete staff member

Dashboard Endpoints:
GET /api/dashboard/stats - Get dashboard statistics

System Endpoints:
GET /health - Health check
GET /api-docs - Swagger API documentation

7. SECURITY FEATURES

Authentication & Authorization:
• JWT (JSON Web Token) based authentication
• Password hashing with bcrypt
• Token refresh mechanism (15 minutes access, 7 days refresh)
• Role-Based Access Control (RBAC)
• Route guards for frontend protection

Security Middleware:
• Helmet - Security headers protection
• CORS - Cross-Origin Resource Sharing control
• Express Rate Limiting - Prevents brute force attacks
• Input Validation - Joi schema validation
• Error Handling - Centralized error middleware
• Token Blacklisting - Logout functionality

Security Best Practices:
• Never expose sensitive data in URLs
• Always use HTTPS in production
• Keep dependencies updated regularly
• Validate all user inputs server-side
• Use environment variables for secrets
• Implement proper error messages
• Use secure headers (X-Frame-Options, X-Content-Type-Options)
• Implement rate limiting on all endpoints
• Monitor and log security events

8. ENVIRONMENT VARIABLES

Backend (.env file):
PORT=3000
NODE_ENV=development
MONGODB_URI=mongodb://localhost:27017/inventory_db
JWT_SECRET=your-super-secret-jwt-key-minimum-32-characters-long
JWT_ACCESS_EXPIRES_IN=15m
JWT_REFRESH_EXPIRES_IN=7d
CORS_ORIGIN=http://localhost:4200
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
LOG_LEVEL=info

Frontend Environment:
NG_APP_API_URL=http://localhost:3000
NG_APP_TIMEOUT=30000

9. DEPLOYMENT GUIDE

Production Build:
Backend: npm run build:backend
Frontend: npm run build:frontend

Vercel Deployment (Frontend):
1. Connect GitHub repository to Vercel
2. Set environment variables in Vercel dashboard
3. Deploy automatically on push to main
4. URL: https://inventory-system-comstruck.vercel.app

Render.app Deployment (Backend):
1. Create Railway project
2. Connect GitHub repository
3. Set environment variables
4. Deploy from main branch
5. Configure MongoDB Atlas connection

Database Setup:
1. Create MongoDB Atlas account
2. Create cluster
3. Create database user
4. Get connection string
5. Update MONGODB_URI in backend .env

10. TROUBLESHOOTING

Common Issues & Solutions:

CORS Errors:
Problem: "Access to XMLHttpRequest blocked by CORS policy"
Solution: Check CORS_ORIGIN in backend .env matches frontend URL

Connection Refused:
Problem: "Cannot connect to http://localhost:3000"
Solution: Ensure backend is running with 'npm run dev' in backend directory

401 Unauthorized:
Problem: "401 Unauthorized" on API requests
Solution: Check JWT token in localStorage, ensure valid token is being sent

MongoDB Connection Error:
Problem: "MongooseError: Cannot connect to MongoDB"
Solution: Verify MONGODB_URI in .env, check MongoDB service is running

Port Already in Use:
Problem: "EADDRINUSE: address already in use :::3000"
Solution: Change PORT in .env or kill process using the port

Module Not Found:
Problem: "Cannot find module 'express'"
Solution: Run 'npm install' in the respective directory

11. DATABASE SCHEMA

Users Collection:
_id (ObjectId)
firstName (String)
lastName (String)
email (String, unique)
password (String, hashed)
role (ADMIN | MANAGER | WAREHOUSE | USER)
isActive (Boolean)
createdAt (Date)
updatedAt (Date)

Products Collection:
_id (ObjectId)
name (String)
description (String)
price (Number)
quantity (Number)
sku (String, unique)
category (String)
minStockLevel (Number)
createdAt (Date)
updatedAt (Date)

Sales Collection:
_id (ObjectId)
productId (ObjectId, ref: Products)
quantity (Number)
unitPrice (Number)
totalPrice (Number)
selledBy (ObjectId, ref: Users)
saleDate (Date)
createdAt (Date)

Staff Collection:
_id (ObjectId)
firstName (String)
lastName (String)
email (String)
position (String)
department (String)
hireDate (Date)
salary (Number)
isActive (Boolean)
createdAt (Date)
updatedAt (Date)

Git Workflow:
1. Create feature branch: git checkout -b feature/feature-name
2. Make changes and commit: git commit -m "description"
3. Push to remote: git push origin feature/feature-name
4. Create pull request on GitHub
5. Request code review

6. Merge after approval

Testing:
• Test all API endpoints
• Test authentication flows
• Test role-based access
• Test error scenarios
• Test database operations

13. USEFUL COMMANDS

Root Level Commands:
npm run install:all - Install dependencies
npm run dev:backend - Start backend server
npm run dev:frontend - Start frontend server
npm run build:all - Build both projects

Backend:
cd backend && npm run dev - Development with hot reload
cd backend && npm run build - Compile TypeScript
cd backend && npm start - Production server

Frontend:
cd frontend && npm start - Development server
cd frontend && npm run build - Production build

14. SUPPORT & RESOURCES

Documentation Links:
- API Swagger Docs: /api-docs
- GitHub: https://github.com/Netizen-gm/Inventory-system-Comstruck
- Live App: https://inventory-system-comstruck.vercel.app

For Issues:
- Create GitHub issues for bugs
- Include error messages and reproduction steps
- Check existing issues first
- I cannot deploy the live deployment that’s my biggest problem

15. PROJECT SUMMARY

The Inventory Management System (IMS) is a production-ready full-stack application designed for efficient inventory tracking and management. It handles everything from stock monitoring to order fulfillment, built with scalable architecture for real-world deployment. Key strengths include robust security measures and role-based access control (RBAC), ensuring data protection and tailored user permissions.

Key Highlights:
- Full-stack TypeScript implementation
- Secure JWT authentication
- Real-time inventory tracking
- Role-based access control
- Comprehensive API documentation
- Production deployment ready
- MongoDB integration
- Responsive UI with Angular 17+