Full-Stack Inventory Management System - Capstone Project

Problem Statement

Modern businesses require efficient inventory management systems to streamline stock tracking, prevent shortages, and optimize warehouse operations. Many small-to-medium enterprises struggle with outdated manual methods or fragmented tools, leading to inventory mismanagement and operational inefficiencies. This project aims to develop a **Full-Stack Inventory Management System** to enable real-time product tracking, role-based access control, and seamless user interactions.

User Story-Based Requirement

User Personas

1. **Admin**: Manages users, sets roles, and oversees inventory analytics.
2. **Manager**: Adds, updates, and deletes product records while monitoring stock levels.
3. **Staff**: Views available stock and updates inventory movement.

User Stories

- **As an Admin**, I want to create and manage user roles so that different levels of employees have appropriate access.
- **As a Manager**, I want to add, update, and remove product details so that inventory remains accurate.
- **As a Staff Member**, I want to view inventory details and update stock movements so that data remains up to date.
- **As an Admin**, I want to generate reports on stock movement so that I can analyze inventory trends.
- **As a Manager**, I want to receive notifications when stock reaches a low level so that I can reorder supplies.

Key Features

- **CRUD Operations**: Add, edit, delete, and view products and inventory records.
- **React Frontend + .NET Core API Backend**
- **JWT Authentication**: Secure access with token-based authentication.

- **Role-Based Access Control**: Different permissions for Admin, Manager, and Staff.
- **Redux State Management**: Efficient application state management.
- **Deployment on Azure/AWS**: Cloud-hosted solution for scalability.

Standard Submission Guidelines

Project Deliverables

1. **Code Repository**:
   - Submit GitHub/GitLab repository link with clear documentation.
   - Include meaningful commit messages.
2. **README File**:
   - Project Overview
   - Setup Instructions
   - API Documentation
3. **Presentation**:
   - Problem Statement & Solution
   - Live Demo or Screenshots
4. **Deployment Link**:
   - Azure/AWS hosted application URL
5. **Test Cases**:
   - Functional and API tests with sample requests & responses.

API Testing Requirements

Endpoints & Testing Scenarios

Endpoint	Method	Description	Sample Request
`/api/auth/login`	POST	Authenticates user & returns JWT	`{ "username": "admin", "password": "admin123" }`
`/api/products`	GET	Retrieves all products	N/A
`/api/products/{id}`	GET	Retrieves product by ID	N/A

/api/products	POST	Adds new product	{ "name": "Laptop", "quantity": 10, "price": 1200 }
/api/products/{id}	PUT	Updates product details	{ "quantity": 15 }
/api/products/{id}	DELETE	Removes a product	N/A

Project Directory Structure

Unset

```
Inventory-Management-System/
|── backend/                    # .NET Core API
|   ├── Controllers/             # API controllers
|   ├── Models/                  # Database models
|   ├── Services/                # Business logic services
|   ├── Data/                    # Database context
|   ├── Middleware/              # JWT authentication logic
|   ├── Program.cs               # Entry point
|
|── frontend/                   # React Frontend
|   ├── src/
|   |   ├── components/         # UI Components
```

```
|   |   ├── pages/              # Screens
|   |   ├── store/              # Redux store
|   |   ├── services/           # API Calls
|   |   ├── App.js              # Main app component
|   |   ├── index.js            # Entry point
|
|── tests/                      # API & UI tests
|── README.md                   # Documentation
|── package.json                # Frontend dependencies
|── docker-compose.yml           # Dockerized deployment
```