Project Documentation

Store Manager - Keeps Track of Inventory

1.Introduction:

Project Title: Store Manager-Keeps Track of Inventory

Team ID: NM2025TMID37132

Team Leader: Nibha Kumari S - nibhakumari5282@gmail.com

Team Members

1.Siyana Fathima S- sheiksai569@gmail.com

2.Nithyashree P -nithyashree1512007@gmail.com

3.Pavithra E - gomathipavilove@gmail.com

2.Project Overview

- **Purpose:**

The primary purpose of the Store Manager Project is to provide a digital solution for managing the day-to-day operations of a store.

- It ensures accurate inventory tracking, so store owners know exactly how much stock is available at any time.
- It simplifies billing and invoicing, reducing errors caused by manual calculations.
- It generates sales and stock reports, helping the owner make better business decisions.
- It improves efficiency, saves time, and reduces costs by automating repetitive tasks.
- By implementing this system, store managers can focus on business growth rather than spending unnecessary time on paperwork or manual record-keeping.

Features of Store Manager System

- **Project Posting and Bidding**

The Project Posting and Bidding feature allows store managers or suppliers to post new inventory requirements or supply projects. This feature makes the process of ordering and supplying goods transparent and competitive. Suppliers

can place bids for fulfilling inventory requirements, allowing the store manager to choose the best offer based on price, quality, and delivery time. This ensures efficient procurement and helps in cost reduction.

Benefits:

Streamlined procurement process

Encourages competitive pricing

Easy tracking of orders and bids

- **Secure Chat System**

The Secure Chat System enables direct communication between store managers, suppliers, and staff within the platform. All messages are encrypted to ensure confidentiality, preventing unauthorized access. Managers can quickly clarify order details, discuss inventory issues, or coordinate deliveries without relying on external communication channels.

Benefits:

Instant communication

Data security and privacy

Efficient problem resolution

- **Feedback and Review System**

The Feedback and Review System allows store managers to rate suppliers and staff based on performance, delivery speed, and product quality. Suppliers can also provide feedback to the store regarding order management or process improvement. This feature builds trust, encourages accountability, and ensures continuous improvement in inventory management operations.

Benefits:

Improves supplier and staff performance

Builds transparency and trust

Helps in decision-making for future procurement

- **Admin Control Panel**

The Admin Control Panel is the central dashboard for the store manager to oversee all activities in the system. It provides complete control over user accounts, inventory records, orders, bids, and feedback. The admin can generate reports, monitor stock levels, and make strategic decisions for the store efficiently. This panel is designed to make management easier and reduce manual work.

Benefits:

Centralized control of store operations

Real-time monitoring of inventory and orders

Easy generation of reports and analytics

3. Architecture of Store Manager System

The Store Manager System is designed using a modern web application architecture that ensures smooth performance, scalability, and maintainability. The system is divided into three main components: Frontend, Backend, and Database.

- **Frontend**

The frontend is the part of the application that interacts directly with users. It is built using React.js, a powerful JavaScript library for building dynamic and responsive user interfaces. To enhance the visual appeal and usability, Bootstrap and Material UI are used. These frameworks provide pre-designed components like buttons, forms, and navigation menus, making the application visually consistent and easy to navigate.

Key Features of the Frontend:

- Responsive design for desktops, tablets, and mobile devices
- Dynamic rendering of inventory, bids, and feedback data
- Interactive dashboards for easy monitoring
- Secure forms for login, registration, and project posting

- **Backend**

The backend handles the server-side logic and processes requests from the frontend. It is developed using Node.js, which allows efficient handling of

multiple requests simultaneously, and Express.js, a lightweight framework for building APIs. The backend is responsible for:

- Managing user authentication and authorization
- Handling inventory management logic
- Processing project posting, bidding, and feedback submissions
- Managing chat messages and notifications
- Serving APIs for the frontend to fetch and send data

Advantages of this Backend Architecture:

- Fast and scalable server performance
- Easy integration with frontend and database
- Simplified routing and API management
- Supports real-time communication for chat system

➢ **Database**

- The system uses MongoDB, a NoSQL database, to store all essential data. This includes:
- User information (store managers, suppliers, staff)
- Project details and bids
- Inventory records

Chat messages and feedback/reviews

MongoDB is chosen for its flexibility, allowing storage of data in JSON-like documents, which can easily adapt to changes in application requirements. Its scalability ensures that as the store or number of users grows, the system continues to perform efficiently.

Database Features:

- Fast data retrieval for real-time operations
- Secure data storage with user access control
- Easy backup and recovery
- Supports complex queries and aggregations for reports

System Flow Diagram (Architecture Overview)

The architecture can be visualized as a three-tier system:

- Frontend (React.js + Bootstrap/Material UI) → interacts with → Backend (Node.js + Express.js) → communicates with → Database (MongoDB)

- Backend also handles APIs for chat, project posting, bidding, and feedback management
- This architecture ensures that the Store Manager System is modular, efficient, and scalable, making it suitable for modern inventory and store management requirements.

4. Setup Instructions for Store Manager System

The Store Manager System is a web-based application that requires a specific environment to run correctly. Follow the instructions below to set up the system on your local machine.

❖ **Prerequisites**

❖ Before setting up the project, ensure the following tools and technologies are installed:

➢ **Node.js**

Node.js is a JavaScript runtime that allows you to run server-side code.

https://nodejs.org/

➢ **MongoDB**

MongoDB is a NoSQL database used to store application data.

https://www.mongodb.com/

➢ **React.js**

React.js is used for building the frontend user interface.

Installed automatically via create-react-app or can be added using npm:

➢ **Express.js & Mongoose**

Express.js is used for backend routing and API creation.

Mongoose is used to interact with MongoDB in a structured way.

Install them using npm in your project folder:

➢ **Visual Studio Code (VS Code)**

VS Code is a lightweight and powerful code editor.
Download from https://code.visualstudio.com/
Recommended extensions: ESLint, Prettier, MongoDB for VS Code

5.Folder Structure

- A well-organized folder structure is essential for maintaining a scalable and maintainable Store Manager System. Below is an example of a folder structure for a typical full-stack application using Node.js (for backend), Express.js (for API), and Vanilla JS or React (for frontend).
- This structure can also be adapted based on whether you're using a more complex setup like Vue.js or Angular for the frontend, but the core concepts will remain similar.

Folder Structure Overview:

```
/store-manager-system
│
├── /client              # Frontend (React / Vanilla JS / etc.)
│   ├── /public          # Static assets (images, favicon, etc.)
│   ├── /src             # Source code for frontend
│   │   ├── /components  # Reusable UI components (buttons, modals, etc.)
│   │   ├── /pages       # Page components (Login, Dashboard, etc.)
│   │   ├── /services    # API calls and services
│   │   ├── /utils       # Utility functions (date formatting, etc.)
│   │   ├── /App.js      # Main app component
│   │   └── /index.js    # Main entry point
│   ├── package.json     # Frontend dependencies and scripts
│   └── .env             # Frontend environment variables (e.g., API base URL)
│
├── /server              # Backend (Node.js + Express)
```

```
│   ├── /controllers       # API route handlers (login, dashboard, etc.)
│   ├── /middlewares       # Middlewares (authentication, error handling)
│   ├── /models            # Database models (User, Product, Order, etc.)
│   ├── /routes            # API routes (userRoutes.js, productRoutes.js, etc.)
│   ├── /utils             # Utility functions for backend
│   ├── /config            # Configuration files (db config, JWT secret, etc.)
│   ├── /services          # Business logic (order processing, inventory
management)
│   ├── /validators        # Request validation logic (email, password format
checks)
│   ├── /app.js            # Main Express.js app file
│   ├── /server.js         # Server entry point
│   └── .env               # Backend environment variables (e.g., DB URI, JWT
secret)
│
├── /database              # Database-related files (migration scripts, seed data)
│   ├── /migrations        # Database migration files (create tables, alter
schema)
│   ├── /seeds             # Sample data (for initial seeding)
│   └── /config.js         # Database configuration
│
├── /tests                 # Automated tests (unit, integration, e2e)
│   ├── /frontend          # Frontend-specific tests
│   ├── /backend           # Backend-specific tests
│   └── jest.config.js     # Jest configuration
│
├── /docs                  # Documentation (API docs, user manuals, etc.)
├── .gitignore             # Git ignore file (node_modules, env files, etc.)
```

```
├── package.json          # Root dependencies and scripts
└── README.md             # Project overview
```

Detailed Breakdown:

1. /client (Frontend)

This folder contains everything related to the frontend of the application. If you're using a React app, the structure might look like this:

/public: Contains static assets like images, icons, and the index.html file.

/src: Source code for the application.

/components: Reusable components like buttons, modals, navigation bars, etc.

/pages: Page-level components like the Login page, Dashboard page, etc.

/services: Contains API calls and external services (e.g., authentication, data fetching).

/utils: Helper functions and utilities, such as date formatting, validation functions, etc.

App.js: The main app component.

index.js: The entry point where React is rendered.

2. /server (Backend)

This folder is where your Express.js server and API logic will reside:

/controllers: Contains route handlers for your APIs. For example, authController.js for handling login or registration, productController.js for managing product listings.

/middlewares: Contains middlewares such as authentication checks, error handling, request validation, etc.

/models: Your database schema models (e.g., User.js, Product.js, Order.js). If you're using MongoDB, you might use Mongoose models here.

/routes: Contains route files like userRoutes.js, productRoutes.js, etc., which map HTTP methods (GET, POST, PUT, DELETE) to controller functions.

/utils: Utility functions that are shared across the app, such as logging, token generation, etc.

/config: Configuration files for environment variables (e.g., database URI, JWT secret key).

/services: Contains business logic or processing services. For example, orderService.js for handling order processing.

/validators: Contains functions that validate the incoming data in API requests (e.g., email format, password strength).

app.js: The main Express application setup, where routes and middlewares are initialized.server.js: The entry point of your server (listens to a specified port.

3./Database

This folder manages database-related tasks such as migrations, seed data, and configuration:

/migrations: Contains migration scripts for setting up or altering the database schema.

/seeds: Provides sample data or seed data for populating the database on startup.

config.js: Configuration for database connections, like MongoDB URI or MySQL credentials

4**. /tests (Testing)**

Automated testing is a crucial part of maintaining any system. This folder would contain:

/frontend: Tests for frontend components (React testing with Jest, Cypress for E2E testing, etc.).

/backend: Tests for backend routes, services, and controllers.

jest.config.js: Configuration file for running tests (Jest, Mocha, etc.).

5. /docs (Documentation)

This folder would contain any documentation related to the system. Examples include:

API documentation (e.g., OpenAPI specs or simple Markdown files).

User guides or setup instructions.

6. Configuration & Root Files

.gitignore: Ensures that sensitive or unnecessary files (e.g., node_modules, .env) are not committed to version control.

README.md: A file that explains the project, setup instructions, and other details for contributors or developers.

package.json: Manages dependencies and scripts for both the frontend and backen

Example Workflow:

- Login Flow:

The client sends a POST request to the backend's /login API with the username and password.

The backend validates the credentials and returns a JWT token if successful.

The client stores the JWT token (in localStorage or sessionStorage) and uses it for future authenticated requests to secure APIs.

2. Role-based Access:

The backend checks the user's role (e.g., store manager, admin) via the JWT token.

Certain routes or pages may be restricted based on the user's role.

3. Product Management:

The store manager can create, update, or delete products via the backend's /product routes.

The client sends the appropriate requests to the API, and the backend handles business logic.

Running the Application:

The Store Manager System follows a clean and modular folder structure that separates the frontend (client) and backend (server) into different directories. This structure makes the project easy to manage, scalable, and maintainable.

Project Directory Overview

SB-Works/

```
├-- client/        # React frontend
│   ├── components/  # Reusable UI components (buttons, forms, tables)
│   └── pages/       # Application pages (dashboard, login, inventory, etc.)
│
├-- server/        # Node.js backend
│   ├── routes/      # Defines API endpoints (inventory, users, projects, chat)
│   ├── models/      # MongoDB schemas and data models
│   └── controllers/ # Business logic and request handling
```

Folder Explanation

1.**client/ (Fronte nd)**

Contains all the files related to the React.js frontend.

Handles the user interface and communicates with the backend through APIs.

components/

Includes reusable UI components like Navbar, Sidebar, Forms, Buttons, Tables, and Cards.

Promotes reusability and clean code practices.

pages/

Contains full-page components such as Login, Register, Dashboard, Inventory Management, Bidding Page, Feedback Page.

Each page represents a specific route in the frontend.

2. server/ (Backend)

guests. Contains the Node.js and Express.js backend.

Handles authentication, API requests, database interaction, and business logic.

routes/

Defines RESTful API endpoints for handling different req

Example: /api/users, /api/inventory, /api/projects, /api/chat.

models/

Contains Mongoose schemas for MongoDB collections.

Example models: User.js, Inventory.js, Project.js, Message.js.

controllers/

Defines business logic and functionality for each route.

Example: userController.js handles login, registration, and authentication logic.

Example: inventoryController.js manages inventory add, update, delete, and track operations.

Benefits of this Folder Structure:

Separation of Concerns: Frontend and backend are clearly separated.

Scalability: New features can be added without disrupting existing code.

Maintainability: Easy to debug, update, and manage code.

Reusability: Components and modules can be reused across the project.

Running the Application

After setting up the project and installing all dependencies, you can run both the frontend and backend to launch the Store Manager System. Follow the steps below:

1. Running the Frontend (React.js)

Navigate to the client folder and start the React development server:

cd client

npm start

The frontend application will run on http://localhost:3000 by default.

It provides the user interface where the store manager, suppliers, and staff interact with the system.

2. Running the Backend (Node.js + Express.js)

Open a new terminal, navigate to the server folder, and start the backend server:

cd server

npm start

The backend server will run on http://localhost:5000 (default port).

It handles API requests, processes data, manages authentication, and communicates with the MongoDB database.

3. Accessing the Application

Once both the frontend and backend servers are running:

Open a web browser.

Visit http://localhost:3000 to access the Store Manager System.

Ensure MongoDB is running in the background for proper database connectivity.The Store Manager System provides a set of RESTful APIs that allow communication between the frontend and the backend. These APIs handle user authentication, project posting, bidding, chat messaging, and inventory management.

Below is the structured documentation of the key API endpoints:

1. User APIs

Register User

Endpoint:

POST /api/user/register

Description: Registers a new user (store manager, supplier, or staff).

Request Body Example:

```
{
 "name": "Roslin",
 "email": "roslin@example.com",
 "password": "securePassword"
}
```

Response Example:

```
{
 "message": "User registered successfully",
 "userId": "64a92f"
}
```

Login User

Endpoint:

POST /api/user/login

Description: Logs in a user with email and password.

Request Body Example:

```
{
 "email": "roslin@example.com",
 "password": "securePassword"
}
```

Response Example:

```
{
 "message": "Login successful",
 "token": "jwtToken123"
}
```

2. Projects APIs

Create Project

Endpoint:

POST /api/projects/create

Description: Allows a store manager to create a new project or inventory requirement.

Request Body Example:

{
"title": "Need 50 units of rice bags",
"description": "Supplies required within 7 days",
"budget": 2000
}

Response Example:

{
"message": "Project created successfully",
"projectId": "p123"
}

Get Project by ID

Endpoint:

GET /api/projects/:id

Description: Retrieves project details using the project ID.

Response Example:

{
"projectId": "p123",
"title": "Need 50 units of rice bags",
"status": "Open",
"bids": []

}

3. Applications APIs

Apply for Project

Endpoint:

POST /api/apply

Description: Allows a supplier to apply or place a bid for a project.

Request Body Example:


```
{
 "projectId": "p123",
 "supplierId": "s456",
 "bidAmount": 1800,
 "deliveryTime": "5 days"
}
```


Response Example:


```
{
 "message": "Application submitted successfully",
 "applicationId": "a789"
}
```


4. Chat APIs

Send Chat Message

Endpoint:

POST /api/chat/send

Description: Sends a secure chat message between store manager and supplier.

Request Body Example:

{

"senderId

The Store Manager System implements a secure authentication mechanism to ensure that only authorized users (store managers, suppliers, or staff) can access protected resources. This helps maintain data security, user privacy, and prevents unauthorized access.

1. JWT-Based Authentication

The system uses JSON Web Tokens (JWT) for user authentication.

When a user logs in with valid credentials, the backend generates a JWT token and sends it back to the client.

This token is stored in the client-side (usually in localStorage or cookies) and is included in the header of every request to verify identity.

Example Login Flow:

1. User submits login credentials.

2. Backend verifies credentials and generates a JWT.

3. JWT is sent to the client and stored.

4.For every protected request (like posting a project, bidding, or viewing inventory), the JWT token must be included in the Authorization Header

2.Middleware for Route Protection

The backend uses Express.js middleware to protect private routes.

Middleware checks for a valid JWT token before allowing access to routes like:

/api/projects/create

/api/inventory/add

/api/chat/send

If the token is valid → the request continues.

If the token is missing or invalid → the request is rejected with an Unauthorized (401) error.

3.Benefits of JWT Authentication

Stateless: No need to store session data on the server.

Secure: Tokens are digitally signed and cannot be tampered with.

Scalable: Works well in distributed systems and microservices.

Flexible: Can include user roles (e.g., Admin, Store Manager, Supplier) inside the token for role-based access control (RBAC).

User Interface (UI)

The User Interface (UI) of the Store Manager System is designed to be user-friendly, responsive, and intuitive, ensuring smooth interaction for store managers, suppliers, and administrators. The interface is built with React.js, styled using Bootstrap and Material UI, and provides a seamless navigation experience.

1. Landing Page

The Landing Page acts as the entry point of the application.

It provides options for users to log in or register as a store manager, supplier, or staff.

Key highlights and features of the system are displayed, such as Inventory Tracking, Project Posting, Bidding, Chat System, and Feedback Mechanism.

The design is simple yet professional, ensuring new users can easily understand the purpose of the application.

2. Freelancer Dashboard

Once logged in, suppliers (freelancers) are directed to their dashboard.

The dashboard provides:

List of available projects posted by the store manager.

Options to place bids/apply for projects.

Access to chat system for direct communication with managers.

Section to view feedback and ratings from past projects.

This dashboard helps suppliers manage their bids and monitor ongoing projects effectively.

3.Admin Panel

The Admin Panel is a centralized dashboard for administrators (or store managers with higher privileges).

Features include:

User Management: Add, update, or remove users.

Inventory Control: Monitor stock levels, reorder items, and generate reports.

Project Management: Track active projects, bids, and performance analytics.

System Control: Manage feedback, security settings, and access rights.

The panel provides a holistic view of the system, enabling decision-making and efficient store operations.

4. Project Details Page

The Project Details Page provides complete information about a posted project.

Includes:

Project Title and Description (e.g., "Need 100 units of notebooks").

Budget and Deadline.

List of Applications/Bids submitted by suppliers.

Chat option to communicate directly with interested suppliers.

This page ensures transparency by displaying all relevant details in one place, making it easier for the store manager to select the best bid

Testing

Testing is a critical stage in the development life cycle of the Store Manager system. The goal of testing is to ensure that the application works as expected, meets the defined requirements, and delivers a smooth user experience.

Manual Testing during Milestones

Login/Registration Testing: Checked validation for new users, duplicate accounts, and incorrect credentials.

Project Posting & Bidding Testing: Verified creation, updating, and deletion of projects. Tested supplier bids and selection process.

Chat System Testing: Ensured secure message delivery, tested edge cases such as empty messages and long messages.

Feedback System Testing: Checked rating system, verified updates after project completion.

Admin Panel Testing: Validated access controls, ensured only admin accounts can manage system-wide data.

Tools Used

Postman: Used to test backend API endpoints such as /api/user/login, /api/projects/create, and /api/chat/send. Postman verified request/response format and status codes.

Chrome DevTools: Used for debugging React components, checking responsiveness, and analyzing network requests.

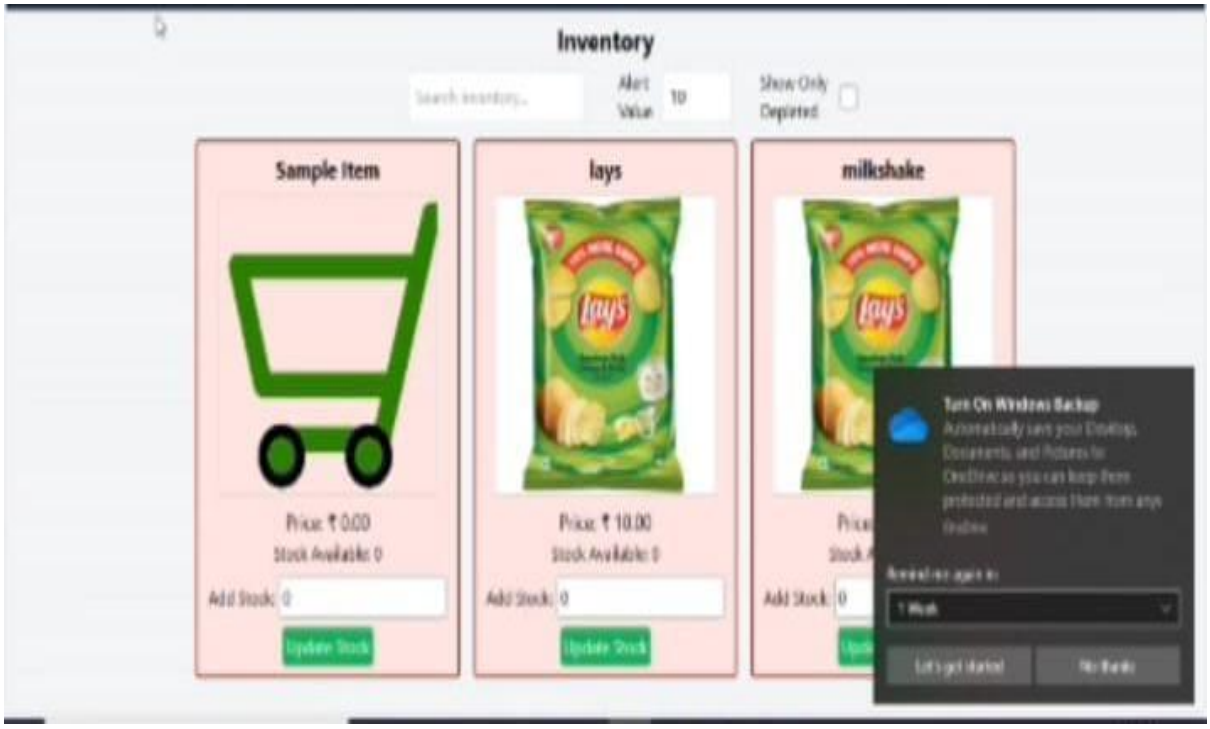
SOURCE CODE

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

OUTPUT



DEMO LINK

https://drive.google.com/drive/folders/1VW8xiIapt7oaf50sqdHFITy8NvZPaZGt

Types of Testing Conducted

Unit Testing: Validated individual functions in backend (e.g., login API, create project API).

Integration Testing: Ensured data consistency between frontend and backend.

System Testing: Checked entire workflow (from login → posting project → bidding → awarding → chat).

User Acceptance Testing (UAT): Gathered feedback from test users to refine usability.

Screenshots or Demo

Screenshots help demonstrate the look and functionality of the Store Manager system. The following screenshots (to be inserted in the final PDF) illustrate core modules:

1. Landing Page – User login and registration.

2. Dashboard – Overview of projects, bids, and inventory.

3. Admin Panel – Controls for managing users, projects, and inventory.

4. Project Details Page – Displays requirements, bids, and supplier information.

5. Chat Window – Secure real-time communication between manager and supplier.

6. Feedback System – Review and rating submission after project completion.

(Each screenshot will be followed by a short explanation in the PDF for clarity.)

Known Issues

During development and testing, a few issues were identified that require future fixes:

Delayed Chat Notifications: Real-time updates sometimes lag when multiple users are online simultaneously.

Large File Uploads: System does not currently support uploading files larger than 10MB.

Mobile Responsiveness: Some dashboard elements do not scale properly on small mobile screens.

Analytics Panel: Admin analytics features (charts, reports) are under development and not fully functional.

Future Enhancements

To improve performance, scalability, and usability, the following enhancements are planned:

1. **Mobile Application:** A dedicated Android/iOS app for store managers and suppliers to manage projects on the go.

2. **AI-based Inventory Prediction:** Machine Learning algorithms to predict inventory shortages and suggest optimal order quantities.

3. **Advanced Analytics Dashboard:** Graphical reports and visualizations for better decision-making by admins and managers.

4. **Cloud Hosting & Scalability:** Deploying the system on cloud platforms (AWS/Azure) for high availability and faster access.

5. **Blockchain-based Transactions:** Securing financial transactions and supplier contracts using blockchain technology.

Multi-language Support: Expanding usability by supporting multiple languages for global accessibility.