

KEEP TRACK OF INVENTORY

- **Project title:** Store manage - keepof inventory
- **Team Id:** NM2025TMID29977
- **Team Leader:** Tharini S & 202400015@sigc.edu
- **Team Member:**
 - **R. Shalinipriya &** 202400383@sigc.edu
 - **K. Riya Sri &** 202400186@sigc.edu
 - **S.Sivashakthini &** 202400351@sigc.edu

2. Project Overview:

Purpose: The Inventory Tracking Project is designed to monitor, manage, and update stock levels efficiently. The main goal is to ensure accurate record-keeping of products, minimize shortages, avoid overstocking, and support smooth business operations.

Key objectives include:

Maintaining a real-time record of available items.

Monitoring stock inflow and outflow.

Generating alerts/notifications for low stock level

3. Architecture:

>>Inventory Tracking System Architecture

1. Presentation Layer (Front-End)

Users: Admin, Staff, Customers (optional)

Interface: Web app / Mobile app

Add/Remove/Update inventory items

Low stock alerts & notifications

Search & filter item

2. Application Layer (Business Logic)

Inventory Management Module

Tracks inflow & outflow of items

3. Data Layer (Database)

Database (SQL or NoSQL) stores:

Product details (ID, Name, Category, Expiry, etc.)

Stock quantity & thresholds

Transactions (purchase, sale, return)

User accounts & roles

Backup & Recovery system for reliability

4.Setup Instructions:

1. Plan the System

Decide where you'll use it:

Small shop, warehouse, office supplies, etc.

2. Set Up the Environment

>> Front-End (User Interface)

Install a framework or use basic web:

Option 1 (Basic): HTML, CSS, JavaScript

>>Back-End (Server Logic)

Choose a language/framework:

Python (Flask/Django/FastAPI)

Set up environment:

Install Python/Node/Java

3. Connect Front-End and Back-End

Front-end → sends requests (Add, Update, View stock).

Back-end → processes requests and updates database.

Use REST API or GraphQL for communication.

Example API routes:

POST /add-product → Add new item

GET /products → List items

PUT /update-stock/:id → Update quantity

(Spring Boot)

5. Folder Structure

1. For a Full-Stack Project (Front-End + Back-End + Database)

inventory-system/

- | — frontend/ **# User Interface (React, Angular, or plain HTML/CSS/JS)**
- | | — public/ **# Static assets (images, icons, favicon)**
- | | — src/
- | | | — components/ **# UI components (Navbar, Forms, Dashboard)**
- | | | — pages/ **# Screens (Login, Inventory List, Reports)**
- | | | — services/ **# API calls (fetch stock, add product, etc.)**
- | | | — App.js
- | | | — index.js
- | | — package.json **# Frontend dependencies**
-
- | — backend/ **# Server-side code (Flask, Django, Node.js, etc.)**
- | | — api/ **# REST API or GraphQL endpoints**
- | | | — product_routes.py (or .js) **# Add/edit/delete products**
- | | | — stock_routes.py **# Stock in/out transactions**
- | | | — report_routes.py **# Generate reports**
- | | | — user_routes.py **# Authentication, roles**
- | | — models/ **# Database models (Product, Stock, User, Transaction)**
- | | — services/ **# Business logic (low stock alert, reorder, etc.)**
- | | — config/ **# Database connection, environment configs**
- | | — tests/ **# Unit/integration tests**
- | | — app.py **# Main backend entry point (Flask/Express app)**
- | | — requirements.txt **# (Python) dependencies OR package.json (Node)**
-
- | — database/ **# Database setup**

- | └─ migrations/ # Schema changes (for SQL DBs)
- | └─ seed/ # Initial sample data
- | └─ schema.sql # Tables definition (if SQL)
- |
- | ─ docs/ # Documentation (API docs, setup guide, ER diagrams)
- | ─ .env # Environment variables (DB password, API keys)
- | ─ .gitignore # Ignore node_modules, __pycache__, etc.
- | ─ README.md # Project overview & setup instructions

2. For a Small Beginner Project (Flask + HTML + SQLite)

inventory-system/

- | ─ app.py # Main backend file
- | ─ static/ # CSS, JS, Images
- | ─ templates/ # HTML templates (Jinja2)
- | └─ index.html
- | └─ login.html
- | └─ inventory.html
- | └─ report.html
- | ─ database/
- | └─ inventory.db # SQLite DB file
- | ─ models.py # Database models
- | ─ routes.py # Routes for products, stock, reports
- | ─ requirements.txt # Dependencies

6. Running the Application:

Steps to Run the Inventory Application

1. Set up the environment

Make sure you have:

Python 3.x installed

pip (Python package manager)

A code editor (VS Code recommended)

2. Create and Activate Virtual Environment

Create project folder

mkdir inventory-system

cd inventory-system

Create virtual environment

python -m venv venv

Activate it

On Windows

venv\Scripts\activate

On Mac/Linux

3. Install Dependencies

Create a file requirements.txt with:

4. Run Database Setup

If you're using SQLite:

The database file (inventory.db) will be created automatically the first time you run the app.

Make sure the folder database/ exists.

7. API Documentation:

 **API Documentation – Keep Track of Inventory System**

 **Authentication**

POST /login

Description: User login with username & password.

Request (JSON):

{


```
"username": "admin",  
"password": "12345"  
}
```

Response:

```
{  
  "token": "abc123xyz",  
  "role": "admin"  
}
```

Products API

1. Get all products

GET /api/products

Response:

```
[  
  { "id": 1, "name": "Laptop", "quantity": 10, "price": 60000 },  
  { "id": 2, "name": "Mouse", "quantity": 50, "price": 500 }
```

]

2. Get product by ID

GET /api/products/{id}

Example: /api/products/1

Response:

{ "id": 1, "name": "Laptop", "quantity": 10, "price": 60000 }

3. Add a new product

POST /api/products

Request (JSON):

```
{  
  "name": "Keyboard",  
  "quantity": 20,  
  "price": 1500  
}
```

Response:

```
{ "message": "Product added successfully", "id": 3 }
```

4. Update product

PUT /api/products/{id}

Request (JSON):

```
{  
  "name": "Gaming Keyboard",  
  "quantity": 25,  
  "price": 2000  
}
```

Response:

```
{ "message": "Product updated successfully" }
```

5. Delete product

DELETE /api/products/{id}

Response:

```
{ "message": "Product deleted successfully" }
```

Stock Management API

6. Add stock (Stock-In)

POST /api/stock-in

Request (JSON):

```
{  
  "product_id": 1,  
  "quantity": 5  
}
```

Response:

```
{ "message": "Stock updated", "new_quantity": 15 }
```

8. Authentication:

Authentication in Inventory Tracking System

1. Types of Authentication

Username & Password (Basic login) → simplest form

Token-based Authentication (JWT, OAuth2) → modern & secure

Role-based Access Control (RBAC) → different permissions (Admin vs Staff)

2. How It Works (Flow)

1. User enters credentials (username + password).

2. Backend verifies credentials against the database.

3. If valid → backend generates a JWT token (JSON Web Token).

4. The token is returned to the client (frontend/mobile app).

5. For every next request (e.g., /api/products), the token must be sent in the Authorization header.

6. The backend validates the token → grants or denies access.

3. API Endpoints for Authentication

◆ Register User (only Admin can do this)

POST /api/register

```
{  
  "username": "staff1",  
  "password": "mypassword",  
  "role": "staff"  
}
```

Response:

```
{ "message": "User registered successfully" }
```

◆ Login

POST /api/login

```
{  
  "username": "admin",  
  "password": "12345"  
}
```

Response:

```
{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "role": "admin"  
}
```

◆ Protected Request (Example: Get Products)

GET /api/products

Headers:

Authorization: Bearer <your_token_here>

4. Role-Based Access Example

Admin

Can add/update/delete products

Can view all reports

Can register new users

Staff

Can view products

Can update stock (in/out)

Cannot delete products

Manager

Can view reports

Cannot modify stock directly