iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85550

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

Digital Inventory Management System - A Secure Role-Based Web Platform for Real-Time Stock Control, GST Billing and Low-Stock Alerting

Mrs. C. Akshaya[1], Rakesh M[2], Santhosh Kumar M[3], Tamilselvan M[4]
[1]*Assistant Professor,* [2, 3, 4]*B.E. Students, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Dr. M.G.R. Nagar, Hosur – 635130, Tamil Nadu, India*

Abstract: ***Retail shops that sell many kinds of goods under one roof, such as stationery, electronics, groceries, clothing and home appliances, find it difficult to keep an accurate record of stock. Handwritten registers and loosely maintained spreadsheets lead to stock mismatches, unnoticed stock-outs, calculation errors in bills and a lack of timely information for the owner. This paper presents the Digital Inventory Management System for a Multicategory Shop, a secure, role-based web application that unifies stock control, customer management, GST billing and reporting. The system is a full-stack JavaScript application: a React 19 front end built with Vite, a Node.js and Express REST API, and MongoDB storage with an automatic fallback to a local JSON file. An Admin maintains the product master, restocks items individually or in batches, monitors low-stock alerts with recommended reorder quantities, manages customers and membership tiers, reviews the sales ledger, prints GST tax invoices, exports a CSV inventory report and broadcasts notifications, while a Customer browses a category-wise catalogue, checks out and tracks orders and membership benefits. Passwords are hashed with salted scrypt, sessions use signed JWT (HS256) tokens, every route is guarded by role-based middleware, and prices, discount, 5% GST, invoice numbers and stock deductions are computed only on the server, with atomic stock reservation to prevent overselling. A working prototype was exercised with sample data of 20 products, 6 customers and 5 orders, and the results show an accurate, real-time and easy-to-use inventory and billing platform.***

Index Terms: ***Inventory management, multicategory shop, stock control, low-stock alerts, GST billing, role-based access control, JWT authentication, MERN stack.***

I. INTRODUCTION

A. Background

A multicategory shop sells products from very different families, for example pens and notebooks, headphones and keyboards, coffee and cooking oil, jackets and trousers, and kitchen appliances. Each family has its own price range, speed of sale and re-ordering pattern, so keeping track of the whole range on paper is slow and error prone. A product that has run out is discovered only when a customer asks for it, while a slow-moving product quietly ties up the owner's money on the shelf. Inventory management is the discipline of knowing what is in stock, what it is worth and when it must be replenished, and the classical literature treats accurate stock information as the foundation of every ordering policy [1]–[5]. Visibility of inventory is also a precondition for a business that can respond to disruption [12].

In a modern shop inventory is closely tied to billing, because every sale must reduce the stock and every bill must carry the correct tax [11]. When the two are kept in separate tools the records drift apart and the owner loses trust in both. The proposed system brings stock control, customer management, billing and reporting into one web application. The admin works from a dashboard showing total products, available units, stock valuation, today's sales, customers and low-stock alerts at a glance, while customers use a separate storefront to browse by category, fill a cart and place an order. Every order automatically reduces stock, applies the membership discount and 5% GST, and produces a numbered tax invoice that can be printed or saved as PDF. The interface carries the product name WIP – Warehouse Inventory Pro; in this paper the software is called the Digital Inventory Management System, or simply the system.

©IJRASET: All Rights are Reserved

B. Objectives

The aim is to design and implement a secure, real-time and user-friendly inventory system that a small or medium multicategory shop can use every day. Specifically, the system (i) maintains a centralised product master holding SKU, category, price, stock quantity, minimum-stock alert level, description and image; (ii) keeps stock accurate in real time by deducting stock automatically on every order and preventing overselling through atomic reservation; (iii) detects low-stock and out-of-stock products automatically and suggests a reorder quantity with its estimated cost; (iv) computes subtotal, membership discount, 5% GST (CGST 2.5% + SGST 2.5%) and the final amount on the server and issues numbered tax invoices; (v) separates the Admin and Customer roles with signed tokens, hashed passwords and ownership checks; (vi) rewards loyal customers through Silver, Gold and Platinum membership tiers; and (vii) supports decision making through KPIs, category-wise valuation, a seven-day sales trend and a CSV export.

C. Contributions

The main contributions are: a single platform that couples inventory control, GST billing, customer relationship management and reporting; a server-authoritative billing design in which the browser sends only product identifiers and quantities, so prices and totals cannot be tampered with; atomic stock reservation with rollback that prevents two buyers from purchasing the last unit; a storage layer that transparently falls back from MongoDB to a JSON file so that the system runs without a database server; and a working prototype with separate Admin and Customer dashboards.

II. LITERATURE SURVEY

The literature relevant to this work falls into three groups. The first is classical inventory theory. Harris [1] introduced the economic order quantity, which balances ordering cost against holding cost, and Wilson [2] popularised order-point and order-quantity rules for routine stock control; both are mathematical models that do not themselves provide software for recording stock movements. Silver et al. [3] treat safety stock, service levels and the role of information, but give no implementation for small shops. Waters [4] explains stock records, stock-taking, ABC classification and the cost of holding stock, which motivated the valuation and top-asset reports of our Admin dashboard, and Chopra and Meindl [5] show that inventory visibility is essential for customer service, without focusing on low-cost retail tools.

The second group concerns the software architecture. Fielding [6] defines the REST style of stateless requests and uniform resource interfaces that the back end of this system follows. Tilkov and Vinoski [7] describe the event-driven, non-blocking model of Node.js, well suited to the I/O-bound read and write pattern of inventory applications, and Chodorow and Dirolf [8] explain MongoDB's document model and atomic single-document updates, which inspired the atomic stock-reservation approach. The third group is security: RFC 7519 [9] defines the JSON Web Token used for stateless sessions, and RFC 7914 [10] specifies the memory-hard scrypt function used for password storage. Table I summarises the surveyed work and the gap each leaves.

Table I. Summary of surveyed work and identified gap

Ref.	Focus	Limitation addressed by the proposed system
[1]	Economic order quantity	Mathematical model only; no software to record stock
[2]	Order-point stock control	Designed for paper records; no automatic evaluation
[3]	Inventory policies, safety stock	Theoretical; no implementation for small shops
[4]	Stock records, ABC classification	Practical guidance only; no integrated billing
[5]	Supply-chain visibility	Low-cost retail tools are not its focus
[6]	REST architectural style	Style definition; applied here to inventory resources
[7]	Node.js event-driven model	General runtime study; not an inventory solution
[8]	MongoDB document model	Database guide; atomic reservation adapted here
[9], [10]	JWT and scrypt standards	Security primitives; combined here with role checks

No single study reviewed combines live stock control, server-side GST billing, membership management, role-based security and reporting in one lightweight, easily deployed application for a multicategory shop. The present work targets that gap.

III. EXISTING SYSTEM

In many small and medium shops stock is still recorded in handwritten registers or basic spreadsheets. Purchases, over-the-counter sales and returns are noted separately and the closing stock is calculated only during periodic stock-taking. Bills are prepared by hand or in a word processor and tax is calculated on a calculator. Because the records are updated by different people at different times, the register rarely matches the shelf, the owner learns that an item is finished only when a customer asks for it, and the reorder decision depends on memory. The principal disadvantages are as follows.

1) Inaccurate stock records. Manual entry and delayed updates cause differences between recorded and physical stock. 2) No early warning. Low-stock and out-of-stock items are noticed late, resulting in lost sales. 3) Billing errors. Manual discount and GST calculations are slow and often wrong, and invoices are not numbered or stored consistently. 4) Poor security and accountability. Anyone who can open the register can change it, and there is no record of who did what. 5) Scattered customer data. Membership and purchase history are not available at the counter. 6) Time-consuming reporting. Valuation, category-wise analysis and sales summaries must be prepared by hand.

IV. PROPOSED SYSTEM

A. Overview

The proposed system is a web-based inventory and billing platform with two clearly separated roles. The Admin signs in to an operations dashboard to manage products, adjust stock, view alerts, manage customers and membership tiers, read the sales ledger, print invoices and broadcast notifications. The Customer signs in to a storefront to search the catalogue, add items to a cart, pay by the preferred mode and view order history and invoices. All business rules run on the server: on receiving an order the server reads current prices from the database, applies the active membership discount, adds 5% GST, reserves the stock atomically, generates the invoice number and records the sale. The browser sends only product identifiers and quantities. The data layer uses MongoDB through Mongoose and, if MongoDB cannot be reached, performs the same operations on a local JSON file.

B. Architecture and Functional Modules

As shown in Fig. 1, a user starts at the landing page and opens the login page, where the Admin or Customer role is selected; a new customer may register. After successful authentication the server issues a signed JWT and the user is redirected to the Admin or Customer dashboard. An authentication context keeps the user's information across the application, and a central API service attaches the token to every protected request and signs the user out when the session expires. Six cooperating modules sit behind the API: authentication and authorisation, product management, order and billing, stock control, customer and membership management, and reports and notifications. Each is an Express router with its own validation, and all share one data-access layer (the store) that hides whether data lives in MongoDB or in the JSON file.

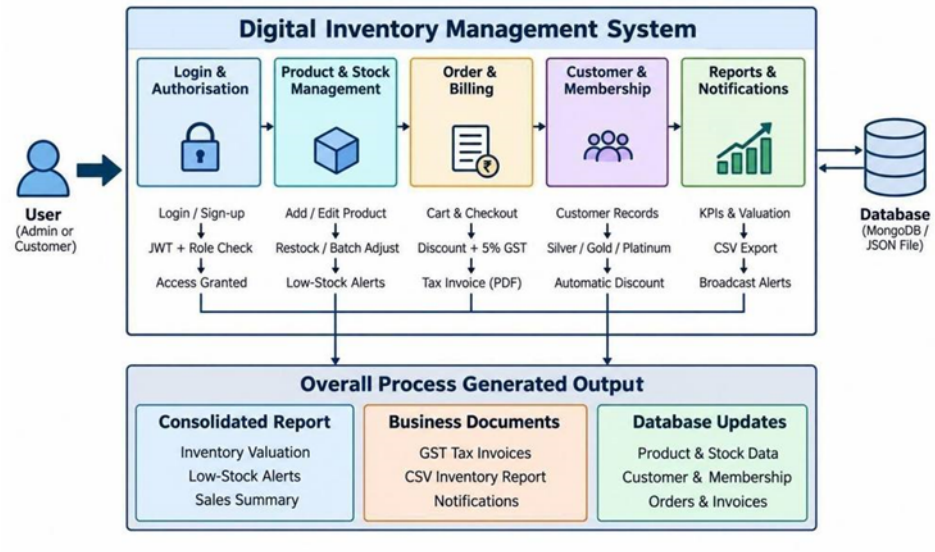


Fig. 1: Architectural design of the proposed system

The flow of data between the user entities, processes and data stores is shown in Fig. 2. The Authenticate and Authorize process verifies the token and role or creates the account (store D2); Product Management updates the product records (D1); Order and Billing reads prices from D1 and the membership discount from D2, reserves stock, applies GST, stores the sale in D3 and returns the tax invoice; Stock Control applies restock and batch-adjust requests and sets the stock status; Customer and Membership applies tier and status changes; and the Notification Engine receives the stock status and raises alerts.

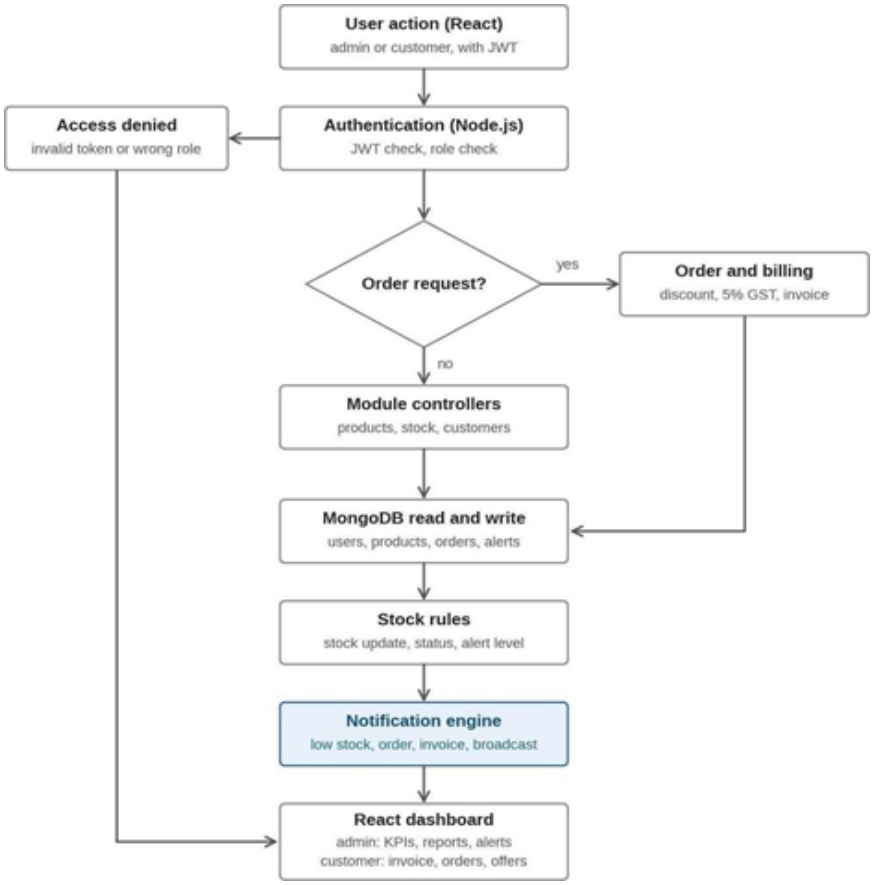


Fig. 2: Data flow diagram of the proposed system

C. Advantages

The design provides real-time stock accuracy, because every sale deducts stock instantly and atomic reservation prevents overselling; automatic alerts, with products at or below their minimum level flagged as Warning, High or Critical together with a recommended reorder quantity; tamper-proof billing, since prices, discount, GST, totals and invoice numbers are computed only on the server; strong security through salted scrypt hashes, signed JWT tokens, role checks on every route, login throttling and per-customer data isolation; insightful reports including KPIs, valuation, category analytics, top-value assets, a seven-day sales chart and CSV export; customer loyalty through automatic membership discounts; and easy deployment with an open-source stack and a JSON-file fallback.

D. Problem–Solution Fit and Ideation

Ideation began by listing the daily difficulties of a shop owner and converting each into a software feature (Fig. 3). Stock mismatches are resolved by automatic deduction, atomic reservation and restocking tools; late identification of stock-outs by minimum-stock alerts, a Low Stock filter and an urgent reorder panel; billing errors by server-side calculation of discount and GST with numbered invoices; loyalty by membership tiers stored in the customer record; unauthorised changes by role-based middleware and admin-only routes; and slow reporting by a dashboard of KPIs, valuation and category analytics. Table II contrasts the existing and proposed approaches.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

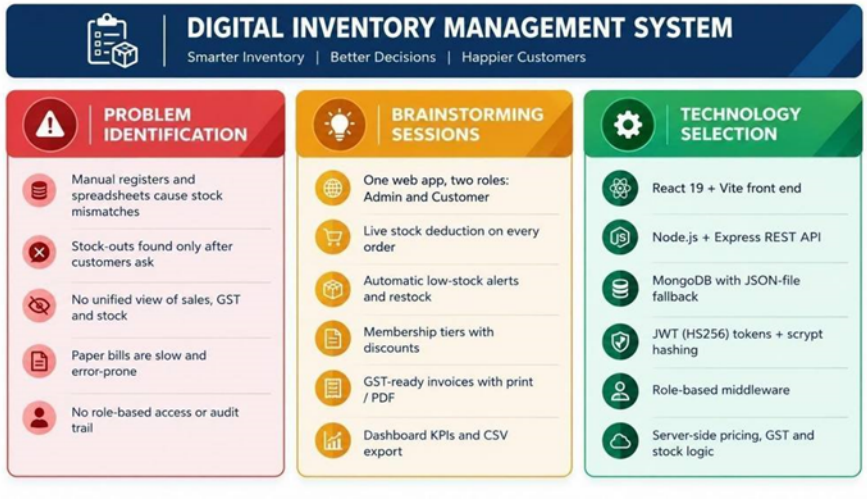


Fig. 3: Ideation and brainstorming outcome

Table II. Existing approach versus proposed system

Aspect	Existing approach	Proposed system
Stock record	Handwritten register or spreadsheet	Central product master updated on every sale
Shortage warning	Noticed when a customer asks	Automatic low-stock alert with reorder quantity
Billing	Manual discount and GST	Server-side calculation, numbered tax invoice
Security	No access control or audit trail	JWT, scrypt hashing, role-based middleware
Customer data	Notebooks, inconsistent	CRM with Silver, Gold and Platinum tiers
Reporting	Hours of manual summation	Dashboard KPIs, valuation, CSV export

V. IMPLEMENTATION

The system is implemented as a full-stack JavaScript application. The server folder holds the Express API, the Mongoose models, the middleware, the routes and the data-access layer; the client folder holds the React application with its pages and components; and a root package offers single commands to install all packages and to start the API (port 5001) and the website (port 5173) together. Table III lists the technology stack.

Table III. Technology stack

Layer	Technology	Role
Front end	React 19, React Router 7, Vite 6	Landing, login, Admin and Customer dashboards, protected routes
Back end	Node.js, Express 4	REST API, business rules, role checks, error handling
Database	MongoDB, Mongoose 8	Product, Customer, Sale and Notification schemas
Fallback store	Local JSON file (runtime-db.json)	Same operations when MongoDB is unavailable
Security	JWT (HS256), scrypt	Signed sessions and salted password hashes
Tools	VS Code, Git, npm, ESLint, MongoDB Compass	Development, version control, code quality, data inspection

A. User Registration and Authentication

A customer registers with a name, a unique and valid e-mail address, a password of at least eight characters and, optionally, a phone number and address. The password is converted to a salted scrypt hash [10] using a random 16-byte salt, and the string “salt:hash” is stored. At login the same function recomputes the hash and compares it in constant time to avoid timing attacks.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The server then returns an HS256-signed JWT [9] carrying the user id, role and e-mail, which expires after a configurable time (8 hours by default) and is attached to every later request as a Bearer token. A protected route in the React router prevents a customer from opening the Admin dashboard and vice versa, and repeated failed logins are throttled.

B. Product Management

The Admin can add, edit and delete products and upload a product image. Each product has a name, a category chosen from five allowed categories (Stationery, Electronics, Groceries, Clothing and Home Appliances), a price, stock quantity, minimum-stock alert level, description and a unique SKU; if the SKU is blank the server generates one of the form WIP-XXX-1234. The status (In Stock, Low Stock or Out of Stock) is derived automatically from stock and alert level. Listing and searching are open to any logged-in user, whereas creating, editing, deleting and changing stock require the Admin role, and a cleaning function validates fields, category and numeric ranges before saving.

C. Order and Billing

A customer places an order from the cart or with the Buy Now button, enters delivery details and chooses UPI, card, net banking or cash on delivery. The placeOrder function of the data-access layer merges duplicate lines, validates each quantity (1 to 1000), loads the price of each product, and computes the amounts in Eq. 1 and Eq. 2, where p_i and q_i are the unit price and quantity of line i and d is the membership discount percentage, which is non-zero only when the membership is Active and not expired.

$$S = \Sigma\, p_i\, q_i\,, \quad D = S \times d / 100 \tag{1}$$

$$G = 0.05 \times (S - D)\,, \quad F = (S - D) + G \tag{2}$$

Here S is the subtotal, D the discount, G the GST (split equally into CGST 2.5% and SGST 2.5%) and F the final payable amount. As a hypothetical illustration, a subtotal of ₹1,000 with a 10% discount gives D = ₹100, a taxable amount of ₹900, G = ₹45 and F = ₹945. Stock is then reserved line by line; if any line fails, earlier reservations are undone. On success a unique invoice number is generated, the sale is saved, the customer's total purchases are updated and notifications are created for the customer and the Admin, including a low-stock alert when a product crosses its alert level. The cart preview in the browser is for display only, because the request carries just product ids, quantities and payment method and the server recalculates every value.

D. Stock Control and Alerts

The Stock Management tab lists all products with current stock, alert level and status. The PATCH stock endpoint sets an exact quantity, quick-restock buttons add 10, 25 or 50 units, and the batch-adjust endpoint accepts either a new value or a delta for each product after a physical count and returns the previous and updated stock for every item with any errors. The status is recalculated after every change, and a Low Stock filter shows only products at or below their alert level, so replenishment can be timed to prevent both stock-outs and over-stocking.

E. Customer and Membership Management

The Customer CRM tab lists every customer with contact details, status, membership tier, discount percentage and total purchases. The Admin can change the tier (Silver, Gold or Platinum) and status (Active, Expired or Suspended) and view a customer's orders. A self-or-admin middleware lets a customer read or edit only his or her own name, phone and address and view membership privileges and the expiry date.

F. Reports, Notifications and Data Layer

The Overview tab presents total products, available stock, total valuation, today's sales, total customers and low-stock alerts with a seven-day sales chart. Further endpoints provide inventory statistics with a health score, category analytics, valuation with the top five assets and a CSV export. The health score is the share of products above their alert level, as in Eq. 3, where N_{ok} is the number of products above the alert level and N is the total number of products.

$$H = (N_{ok} / N) \times 100\ \% \tag{3}$$

Notifications are stored with a type and target so that each user sees only what is meant for him or her, and the Admin can broadcast an information, alert or promotion notice to all customers or to members only. All modules reach the data through one store module, which uses MongoDB when available and otherwise a JSON file with the same structures; request bodies up to 5 MB are accepted because product images are uploaded as base64 text.

VI. SYSTEM REQUIREMENTS

The application is lightweight and entirely open source. The server or development computer, which runs the Node.js API, the database and the Vite development server, needs a dual-core 2.0 GHz processor (Intel Core i3 or AMD Ryzen 3), 4 GB RAM, 10 GB of free storage and a LAN or broadband connection; a quad-core processor, 8 GB RAM, 20 GB of SSD space and a 10 Mbps connection are recommended, with a UPS to avoid data loss during power cuts. Client devices need only a modern web browser on a smartphone, tablet, laptop or desktop with a 1 GHz dual-core processor, 2 GB RAM and a 2 Mbps connection. The software requirements are Node.js, Express 4, React 19, React Router 7, Vite 6, MongoDB Community Edition with Mongoose 8, and a .env file defining PORT, MONGODB_URI and JWT_SECRET. The system runs on Windows, Linux or macOS, either on a mini-PC in the shop or on a cloud virtual machine, for which 1 vCPU and 2 GB RAM is a reasonable starting point for a small number of users.

VII. TESTING, RESULTS AND DISCUSSION

The prototype was executed with the sample data supplied with the project, comprising 20 products, 6 customers and 5 orders. Table IV summarises the behaviour observed for each major function, and Figs. 4–11 show representative screens of the Admin and Customer dashboards. Fig. 4 shows the Admin overview with its KPIs; Fig. 5 the product master and asset registry; Fig. 6 the stock replenishment centre with quick-restock buttons; and Fig. 7 the Low Stock Only filter. Fig. 8 shows the orders, tax invoices and sales ledger, and Fig. 9 an official tax invoice. On the customer side, Fig. 10 shows the category-wise product catalogue and Fig. 11 the shopping cart with the membership discount applied.



Fig. 4: Admin dashboard – overview and KPIs

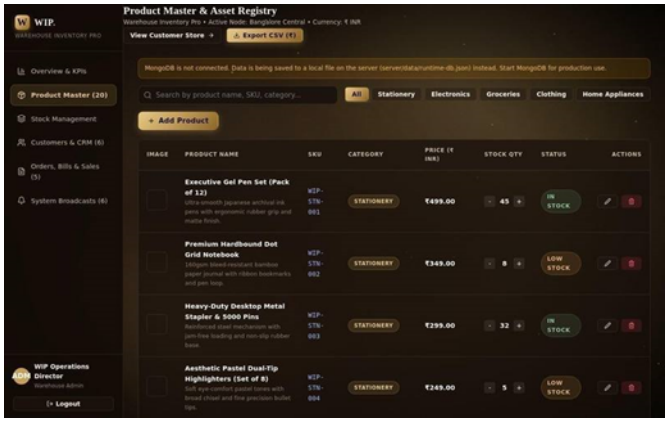


Fig. 5: Product master and asset registry

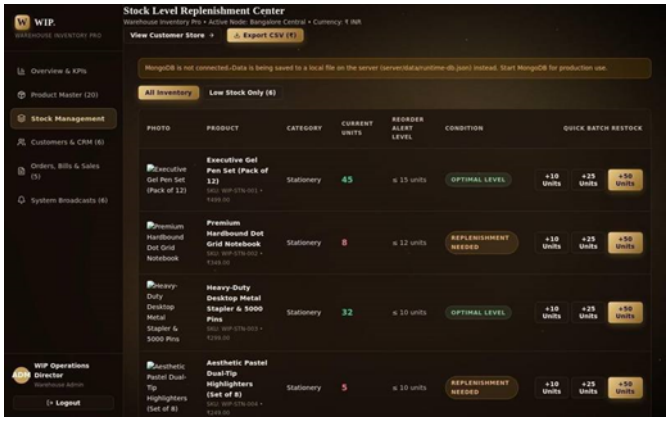


Fig. 6: Stock level replenishment center

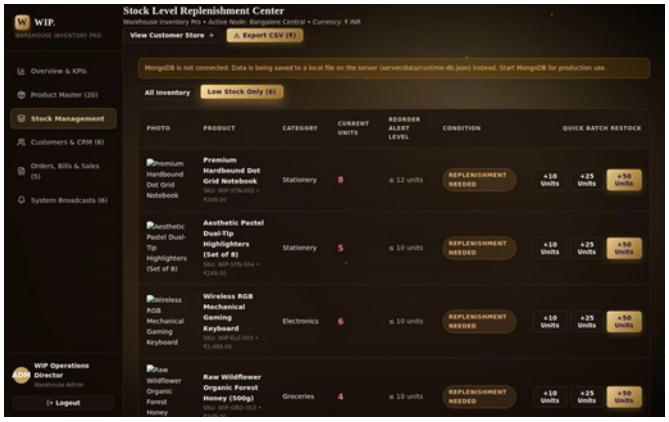


Fig. 7: Low Stock Only filter

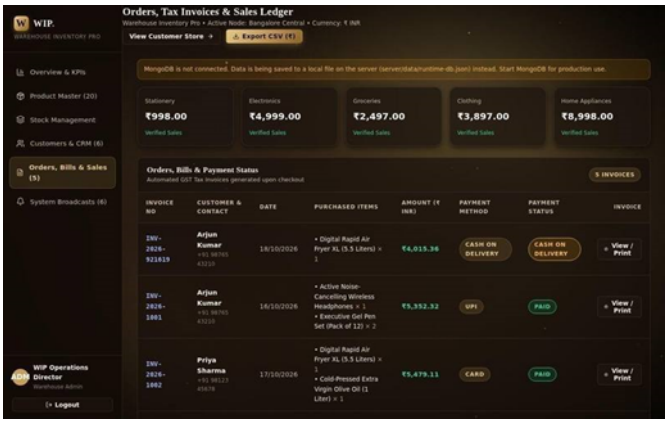


Fig. 8: Orders, tax invoices and sales ledger

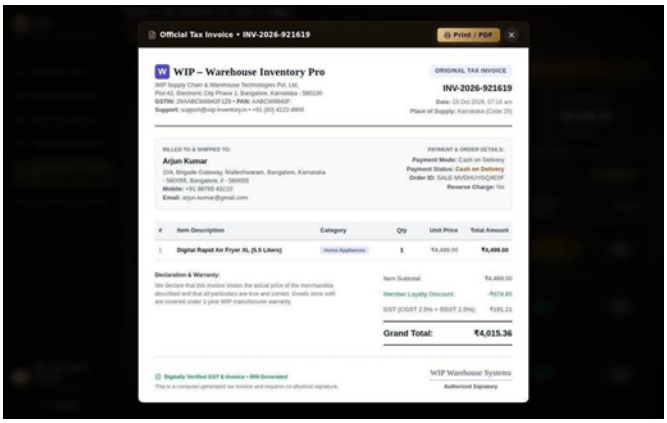


Fig. 9: Official tax invoice

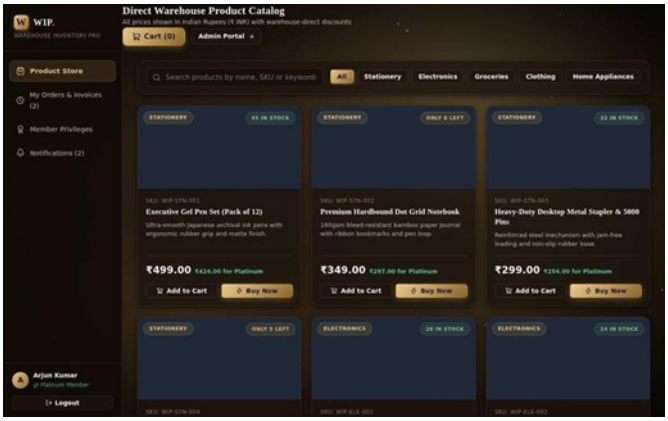


Fig. 10: Customer product catalogue

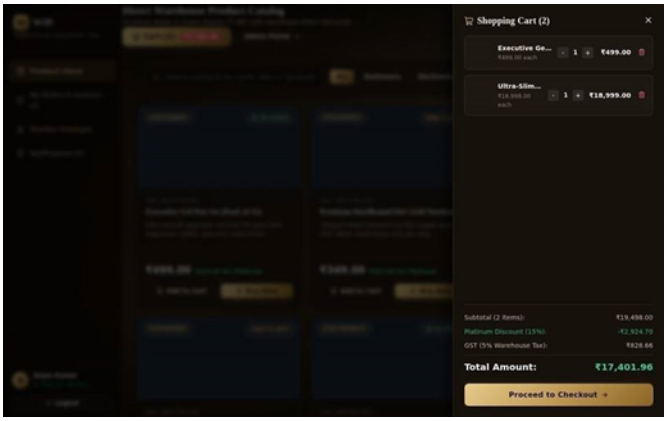


Fig. 11: Shopping cart with membership discount

Table IV. Functions verified with the sample data

Function	Observation
Authentication and registration	Customers registered and logged in; Admin and Customer roles opened only their own dashboards
Product management	Products added, edited and deleted; status derived from stock and alert level
Order processing	Discount, 5% GST and totals calculated on the server; stock deducted; numbered invoice generated
Low-stock alerts	When stock reached or fell below the alert level the status changed to Low Stock and the product appeared in the alerts section
Batch stock adjustment	Several quantities updated together; audit notifications generated
CSV export	Inventory report downloaded in the expected file format
Storage fallback	Server started without MongoDB saved data to runtime-db.json and showed a notification to the Admin

A. Discussion

The results show that the major functional modules behaved according to the intended rules: stock changed immediately after each order, invoices carried correct totals, alerts appeared when a product crossed its alert level, and role-based access, customer data isolation, input validation and error handling worked as designed. Compared with the manual practice described in Section III, the system joins stock control and billing in one record, so the register can no longer drift from the shelf, and the owner is warned of a shortage before a customer asks for the item.

We note that the evaluation used a small demonstration data set rather than a long-running shop deployment; in some screenshots product thumbnails appear empty because the sample images are loaded from external internet sources. Claims about performance at larger scale remain to be validated in field trials.

VIII. CONCLUSION AND FUTURE SCOPE

The Digital Inventory Management System for a Multicategory Shop brings stock control, billing, customer management and reporting into one secure web application. The Admin obtains an accurate, real-time view of stock and its value, early warnings about shortages and tools to restock quickly, while customers obtain a convenient storefront with transparent prices, discounts and invoices. Placing all critical calculations on the server, protecting every route with role checks, storing passwords as salted hashes and reserving stock atomically with rollback make the system reliable, and the JSON-file fallback makes it easy to run without a database server. The project shows that a modern open-source stack can provide an affordable inventory solution for small and medium shops.

Future work includes barcode and QR scanning for faster stock entry and billing; a supplier and purchase-order module that raises orders from low-stock alerts; integration of real UPI, card and net-banking payment gateways; demand forecasting from sales history to suggest reorder quantities; batch, expiry and returns tracking; multi-branch support with stock transfers; e-mail and SMS alerts outside the application; and a mobile application with offline mode.

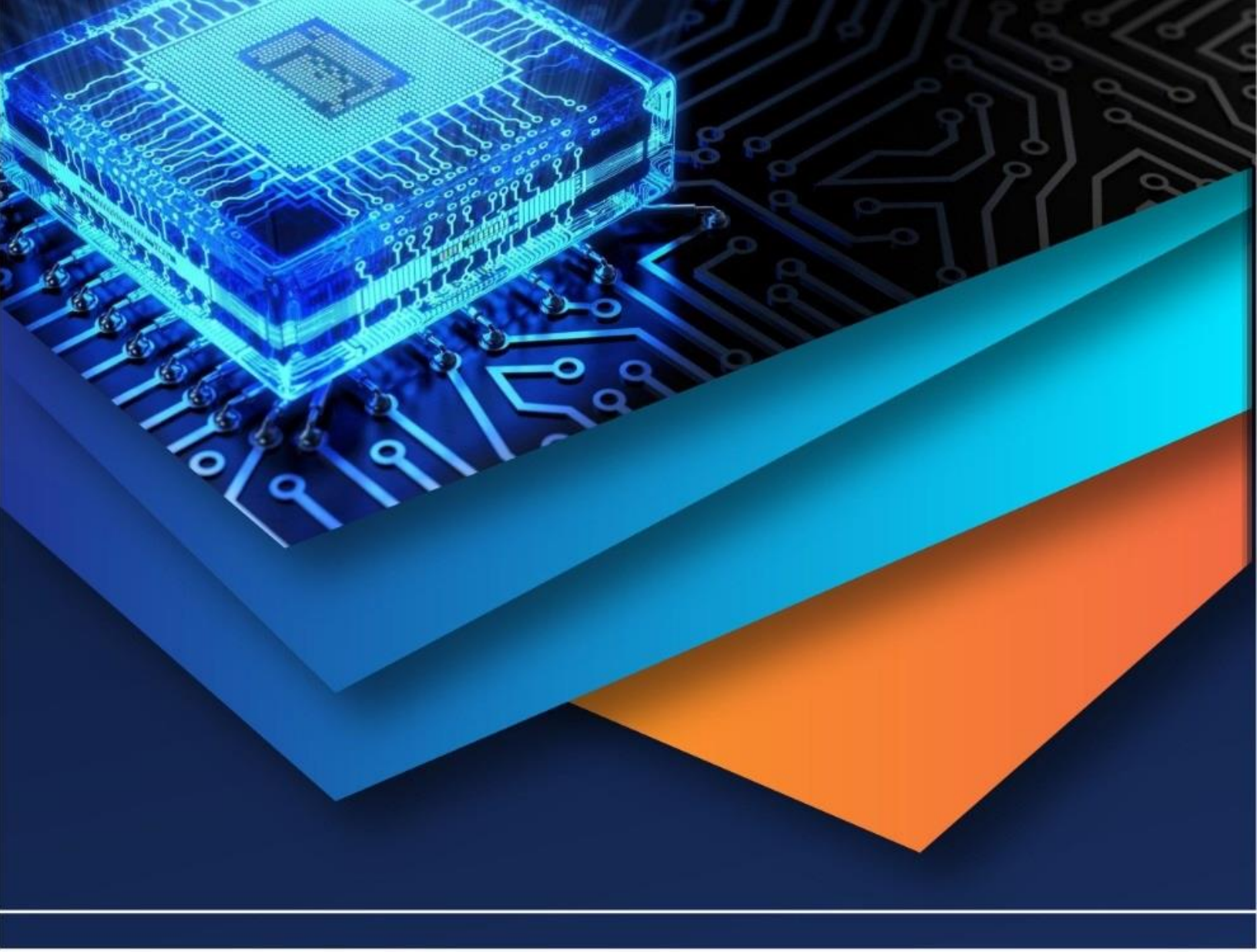
IX. ACKNOWLEDGMENT

The authors thank Dr. R. Radhakrishnan, Principal, and Dr. G. Fathima, Professor and Head, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, for the opportunity and encouragement, and the project guide Mrs. C. Akshaya, Assistant Professor, the project coordinator and the faculty of the department for their support.

REFERENCES

[1] F. W. Harris, “How Many Parts to Make at Once,” Factory, The Magazine of Management, vol. 10, no. 2, pp. 135–136, 1913.
[2] R. H. Wilson, “A Scientific Routine for Stock Control,” Harvard Business Review, vol. 13, no. 1, pp. 116–128, 1934.
[3] E. A. Silver, D. F. Pyke, and D. J. Thomas, Inventory and Production Management in Supply Chains, 4th ed. Boca Raton, FL, USA: CRC Press, 2016.
[4] D. Waters, Inventory Control and Management, 2nd ed. Chichester, U.K.: John Wiley & Sons, 2003.
[5] S. Chopra and P. Meindl, Supply Chain Management: Strategy, Planning, and Operation, 7th ed. Pearson, 2019.
[6] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.
[7] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” IEEE Internet Computing, vol. 14, no. 6, pp. 80–83, 2010.
[8] K. Chodorow and M. Dirolf, MongoDB: The Definitive Guide. Sebastopol, CA, USA: O’Reilly Media, 2010.
[9] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015.
[10] C. Percival and S. Josefsson, “The scrypt Password-Based Key Derivation Function,” RFC 7914, Internet Engineering Task Force, Aug. 2016.
[11] Government of India, The Central Goods and Services Tax Act, 2017.
[12] M. Christopher and H. Peck, “Building the Resilient Supply Chain,” The International Journal of Logistics Management, vol. 15, no. 2, pp. 1–14, 2004.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)