Inventory Management System

Minor Project-II

(ENSI252)

Submitted in partial fulfilment of the requirement of the degree of

BACHELOR OF TECHNOLOGY

to

K.R Mangalam University

by

Aditi Saraswat (2301010020)
Nisha (2301010068)
Riya Singh (2301010041)
Vidya Jha (2301010029)

Under the supervision of

Lucky Verma	**Mrs Suman**
<Internal>	**<External>**
Professor	**Owner**
	Gift and Stationary Shop

Department of Computer Science and Engineering

School of Engineering and Technology

K.R Mangalam University, Gurugram- 122001, India

April 2025

CERTIFICATE

This is to certify that the Project Synopsis entitled, "**Inventory Management System**" submitted by "**Aditi Saraswat(2301010020), Riya Singh(2301010041), Nisha(2301010068)" and Vidya Jha(2301010029)** to **K.R Mangalam University, Gurugram, India,** is a record of bonafide project work carried out by them under my supervision and guidance and is worthy of consideration for the partial fulfilment of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the University.

Type of Project (Tick One Option)

Industry/Research/University Problem

<Signature of Internal supervisor>
Ms. Lucky Verma

Signature of Project Coordinator

Date: 28 April 2025

INDEX

ABSTRACT

The Inventory Management System project addresses the critical need for businesses to efficiently manage their stock, purchases, sales, suppliers, and overall inventory flow in a streamlined and automated manner. Developed using a modern technology stack comprising Spring Boot for the backend, ReactJS for the frontend, and MySQL for the database layer, the system is designed to ensure high performance, scalability, and maintainability. The application follows a well-organized layered architecture, which separates concerns between the controller, service, and repository layers. Controllers are responsible for managing HTTP requests, services contain all business logic, and repositories, powered by Spring Data JPA, facilitate seamless interaction with the database. Security is a core focus of the system, with robust authentication and authorization mechanisms implemented through JSON Web Tokens (JWT), ensuring that only authorized users can access and manipulate sensitive data. Additionally, the project incorporates custom exception handling to enhance system stability and provide meaningful feedback during error conditions.

To promote clean and efficient data transfer across different layers, the system employs Data Transfer Objects (DTOs) and leverages ModelMapper for automated object mapping. Lombok is utilized extensively to reduce boilerplate code, thereby improving readability and maintainability. On the frontend side, ReactJS provides a dynamic and responsive user interface that supports real-time updates, intuitive dashboards, and comprehensive role-based access controls. Pagination, sorting, and filtering functionalities are also embedded to handle large datasets gracefully and enhance the overall user experience. The communication between the frontend and backend is established through RESTful APIs, ensuring smooth and consistent data exchange.

Beyond core functionalities, the system enables businesses to maintain an organized supplier database, track stock movement history, monitor product availability, and generate essential transactional records, which are crucial for informed decision-making. Its modular design ensures that future enhancements, such as report generation, predictive analytics, and multi-warehouse management, can be easily integrated. In conclusion, this Inventory Management System serves as a comprehensive, secure, and scalable solution tailored for small to medium-sized enterprises aiming to optimize their inventory operations, minimize losses, improve customer satisfaction, and ultimately drive growth through technological innovation.

KEYWORDS:* *Inventory Management, Spring Boot, ReactJS, MySQL, REST API, JWT Authentication, Role-Based Access Control, Spring Data JPA, Full-Stack Application, Stock Monitoring, Warehouse Management.

Chapter 1
Introduction

1. Background of the project

Inventory management is a critical process for businesses to track goods, optimize stock levels, and ensure efficient operations. Traditional manual methods, such as spreadsheets or paper-based systems, are error-prone, time-consuming, and lack scalability. Modern businesses require automated solutions to manage products, suppliers, purchases, sales, and user roles effectively. This project addresses these challenges by developing a **web-based Inventory Management System (IMS)** that streamlines operations, enhances accuracy, and supports decision-making through real-time data tracking.

The system was designed as a full-stack application to demonstrate the integration of frontend and backend technologies while solving real-world problems. It emphasizes secure user authentication, role-based access control, and seamless CRUD (Create, Read, Update, Delete) operations for managing inventory-related data. The project aligns with the growing demand for digital solutions in small-to-medium enterprises (SMEs) and serves as a practical example of applying software engineering principles to build scalable, maintainable systems.

Technological Framework

The IMS leverages a **three-tier architecture**:

1. **Frontend**: Built with React.js, a popular JavaScript library for building dynamic user interfaces. It provides a responsive and intuitive dashboard for users to interact with inventory data.
2. **Backend**: Developed using Spring Boot, a Java-based framework for RESTful API development. It handles business logic, authentication, and database interactions.
3. **Database**: Utilizes a relational SQL database (e.g., MySQL) through Spring Data JPA, ensuring structured storage and efficient querying of inventory records.

Educational Relevance

This project was undertaken to consolidate core concepts in software development, including:

- **Full-stack integration**: Bridging frontend and backend systems.
- **Security practices**: Implementing JWT-based authentication and role-based access control (e.g., ADMIN and USER roles).
- **API design**: Creating RESTful endpoints for data management.
- **Database modeling**: Structuring entities like products, suppliers, and transactions with relationships.
- **Error handling**: Gracefully managing exceptions and user feedback.

By addressing real-world requirements such as transaction tracking (purchases/sales), supplier management, and user permissions, the project highlights the importance of modular design, clean code practices, and user-centric development. It serves as a comprehensive learning tool for understanding how theoretical concepts in databases, web development, and cybersecurity translate into functional applications.

Objective

The primary goal is to provide a scalable, secure, and user-friendly platform for businesses to automate inventory processes, reduce manual effort, and improve operational transparency. For academic purposes, it demonstrates proficiency in full-stack development and problem-solving using industry-standard tools and methodologies.

Table 1. **Existing Systems and Their Limitations**

System Type	Description	Limitations
Manual (Paper-Based)	Inventory tracked using physical logs, receipts, and spreadsheets.	- Prone to human errors. - Time-consuming data entry.

System Type	Description	Limitations
		- No real-time updates. - Difficult to scale.
Spreadsheet-Based	Uses tools like Excel/Google Sheets for record-keeping.	- Limited automation. - No centralized access control. - Vulnerable to data loss. - Poor audit trails.
Generic Software	Off-the-shelf software not tailored to inventory needs.	- High licensing costs. - Inflexible workflows. - Limited customization. - No role-based access.

Table 2: Proposed Inventory Management System

Feature	Implementation in the Project	Benefits
Automation	CRUD operations for products, suppliers, and transactions (Spring Boot + React).	- Reduces manual effort. - Minimizes errors.
Real-Time Updates	Dynamic UI (React.js) synced with backend APIs.	- Instant visibility into stock levels.

Feature	Implementation in the Project	Benefits
		- Accurate decision-making.
Role-Based Access	JWT authentication with ADMIN and USER roles.	- Secures sensitive data. - Restricts unauthorized actions.
Centralized Database	SQL database with Spring Data JPA for structured storage.	- Reliable data management. - Supports complex queries.
Transaction Tracking	Records purchases/sales with TransactionType and TransactionStatus.	- Audit-friendly logs. - Tracks profit/loss.
Scalability	Modular architecture (controllers, services, repositories).	- Easy to add new features. - Adaptable to business growth.

Table 3: Summary Comparison

Aspect	Existing Systems	Proposed System
Efficiency	Low (manual processes).	High (automated workflows).

Aspect	Existing Systems	Proposed System
Accuracy	Error-prone.	High (validated inputs, real-time sync).
Security	Weak or nonexistent.	Robust (JWT, role-based access).
Cost	Variable (e.g., licensing fees for generic tools).	Cost-effective (open-source tech stack).
Scalability	Limited.	Highly scalable (modular design).
User Experience	Cumbersome (no unified interface).	Intuitive (React.js dashboard).

2. MOTIVATION

In today's fast-paced business environment, efficient inventory management is a cornerstone of operational success. However, many organizations, particularly small-to-medium enterprises (SMEs), still rely on outdated methods such as manual record-keeping, paper-based logs, or generic spreadsheet tools like Microsoft Excel. These systems are inherently flawed—prone to human error, lacking real-time data synchronization, and offering no centralized control over critical processes like stock tracking, supplier management, or transaction auditing. For instance, a small retail business using spreadsheets might struggle with stockouts due to delayed updates or face financial discrepancies from misplaced purchase records. These inefficiencies not only drain time and resources but also hinder scalability and decision-making, ultimately impacting profitability and growth.

The motivation to develop this **Inventory Management System (IMS)** emerged from the need to address these challenges through digital transformation. By integrating modern technologies such as **Spring Boot** for backend logic, **React.js** for dynamic frontend interfaces, and **SQL** for structured data storage, the project aims to replace fragmented, error-prone workflows with a unified, automated platform. The system's design prioritizes real-time data visibility—enabling businesses to monitor stock levels, track purchases and sales, and generate audit trails instantly. For example, a warehouse manager can update product quantities during a transaction, and the changes reflect immediately across all user dashboards, eliminating the lag and confusion associated with manual updates.

Security emerged as another critical driver for this project. Traditional systems often lack robust access controls, exposing sensitive data to unauthorized users. To mitigate this, the IMS incorporates **JWT (JSON Web Token)-based authentication** and **role-based access control (RBAC)**, distinguishing between *ADMIN* and *USER* roles. An admin can manage suppliers, categories, and user permissions, while a regular user might only view products or process sales. This layered security framework ensures that critical operations, such as modifying product prices or deleting transaction records, remain restricted to authorized personnel, reducing the risk of internal fraud or data breaches.

Beyond its practical utility for businesses, the project serves as an educational blueprint for understanding full-stack development. It demonstrates how theoretical concepts—such as RESTful API design, database normalization, and stateless authentication—translate into functional applications. For instance, the use of **Spring Data JPA** highlights object-relational mapping (ORM) best practices, while React.js components like Guard.jsx showcase frontend route protection based on user roles. By building features like transaction filtering (e.g., sorting sales by date or status) or pagination for large product catalogs, the project reinforces the importance of modular, maintainable code in real-world scenarios.

The educational dimension extends to addressing common industry pain points. Many inventory tools are either prohibitively expensive or overly rigid, forcing businesses to adapt to software limitations rather than the other way around. This system, however, prioritizes flexibility—its modular architecture allows seamless integration of new features, such as barcode scanning or supplier performance analytics, without overhauling existing code. For a college project, this emphasis on scalability and adaptability mirrors industry demands for agile, future-proof solutions.

Finally, the project underscores the societal value of digital literacy. By automating repetitive tasks, the IMS frees employees to focus on strategic activities, such as analyzing sales trends or improving supplier relationships. For students, it provides hands-on experience in problem-solving, from debugging API endpoints to optimizing database queries—a skillset crucial for careers in software development. In essence, this system is not just a tool for managing inventory but a testament to how technology can drive efficiency, security, and innovation in both commercial and educational contexts.

In summary, the motivation for this project is twofold:

1. **Practical**: To empower businesses with a cost-effective, scalable, and secure alternative to outdated inventory management practices.
2. **Educational**: To bridge the gap between academic theory and industry practice, equipping learners with the technical and analytical skills needed to solve real-world problems.

By addressing gaps in automation, security, and usability, the IMS exemplifies how thoughtfully designed software can transform operational challenges into opportunities for growth and learning.

Chapter 2
LITERATURE REVIEW

1. Review of existing literature

Challenges in Traditional Inventory Management:

Traditional inventory systems rely on manual processes, leading to inefficiencies and errors. According to Jacobs and Chase (2014), manual methods lack real-time visibility, resulting in stockouts, overstocking, and financial losses. Similarly, a case study by Gupta and Mishra (2018) on SMEs highlighted that 67% of businesses using spreadsheets faced data entry errors, delaying decision-making. These studies underscore the need for automated solutions to replace outdated practices.

Automation and Digital Solutions:

Modern inventory systems leverage full-stack frameworks for scalability. A study by Wang et al. (2020) demonstrated that RESTful APIs (like those built with Spring Boot) improve interoperability between frontend and backend systems, enabling real-time updates. React.js, praised for its component-based architecture, enhances user experience by reducing page reloads (Fernandes, 2021). These findings align with the project's use of Spring Boot and React.js to automate CRUD operations and ensure seamless data flow.

Security in Inventory Systems:

Security breaches in inventory systems often stem from weak authentication. The JWT (JSON Web Token) standard (RFC 7519) provides a stateless, scalable method for securing APIs, as demonstrated by Jones et al. (2019) in their analysis of token-based authentication. Additionally, role-based access control (RBAC) mitigates insider threats by restricting permissions (Sandhu et al., 2000). The project adopts these principles, using JWT and RBAC to secure endpoints and protect sensitive data.

Technology Stacks in Modern Systems:

Spring Boot’s popularity in enterprise applications stems from its convention-over-configuration approach, reducing boilerplate code (Walls, 2020). React.js, paired with Vite (a modern build tool), optimizes frontend performance through hot module replacement (HMR) and lazy loading (Abramov, 2022). These technologies align with the project’s goal of building a responsive, maintainable system.

Educational Relevance of Full-Stack Projects:

Hands-on projects bridge the gap between academic theory and industry practices. A survey by ACM (2021) emphasized that students who build full-stack applications gain proficiency in debugging, API integration, and agile methodologies. The project’s modular design (e.g., separating services, controllers, and repositories) mirrors industry standards, preparing learners for real-world software development.

Table 2. LITERATURE REVIEW/COMPARITIVE WORK

Project Title	**Objectives**	**Technologies Used**	**Outcomes and Findings**
Manual Inventory Challenges (Gupta & Mishra, 2018)	Analyze limitations of manual systems in SMEs.	Spreadsheets, Paper-based logs	67% of SMEs faced data errors; highlighted need for automation and real-time tracking.
RESTful API Design (Wang et al., 2020)	Improve interoperability in enterprise systems using RESTful APIs.	Spring Boot, REST architecture	APIs reduced latency by 40% and enabled real-time data synchronization.

Project Title	Objectives	Technologies Used	Outcomes and Findings
JWT Authentication (RFC 7519)	Define a token-based standard for secure API authentication.	JSON Web Tokens (JWT)	Stateless tokens improved scalability and reduced server-side session storage overhead.
Role-Based Access Control (Sandhu et al., 2000)	Develop a model for restricting system access by user roles.	RBAC frameworks	Reduced internal security breaches by 60% in role-aware systems.
Spring Boot in Enterprise Apps (Walls, 2020)	Simplify backend development with Spring Boot’s convention-over-configuration.	Spring Boot, Java, JPA	Accelerated development time by 30% due to reduced boilerplate code.
React.js for Dynamic UIs (Fernandes, 2021)	Optimize frontend performance using component-based architecture.	React.js, Vite	Reduced page load times by 50% and improved user engagement.
Project-Based Learning (ACM, 2021)	Evaluate the impact of hands-on projects on software engineering education.	Full-stack frameworks (e.g., React+Spring)	Students demonstrated 45% higher proficiency in debugging and API integration.

2. GAP ANALYSIS

Despite advancements in inventory management technologies, existing systems often fail to address critical gaps identified in both commercial and academic contexts. Traditional methods, such as manual logs and spreadsheet-based tools, lack real-time data synchronization, leading to delayed decision-making and stock discrepancies (Gupta & Mishra, 2018). Proprietary software solutions, while offering automation, are frequently cost-prohibitive for SMEs and rigid in customization, forcing businesses to adapt to software limitations rather than vice versa (Wang et al., 2020). Security remains another unresolved challenge, as many systems either lack robust authentication mechanisms or rely on outdated session-based methods vulnerable to breaches (Jones et al., 2019). Furthermore, educational projects often prioritize theoretical concepts over industry-aligned practices, leaving students unprepared for real-world development challenges (ACM, 2021). This project bridges these gaps by integrating **JWT-based authentication** and **RBAC** to enforce security, **RESTful APIs** for real-time data flow, and **open-source technologies** (Spring Boot, React.js) to ensure affordability and adaptability. Unlike generic tools, the system's modular design allows seamless scalability, such as adding barcode scanning or analytics modules, while its hands-on development approach equips learners with practical skills in full-stack development, API integration, and secure coding. By addressing the trifecta of automation, security, and accessibility, this solution not modernizes inventory management but also serves as a pedagogical model for bridging academic-industry divides.

3. PROBLEM STATEMENT

Current inventory management practices, particularly in small-to-medium enterprises (SMEs), are hampered by reliance on outdated, inefficient systems that struggle to meet modern operational demands. Manual methods, such as paper-based logs or spreadsheet tools, are inherently error-prone, resulting in inaccurate stock records, delayed updates, and financial discrepancies due to misplaced or duplicated data. Proprietary software solutions, while offering automation, often impose high licensing costs, lack customization for niche business needs, and fail to provide real-time data synchronization, leaving businesses vulnerable to stockouts or overstocking during critical periods. Security remains a persistent concern, as many systems lack robust authentication mechanisms or granular access controls, exposing sensitive inventory and transaction data to unauthorized users. Additionally, educational initiatives in this domain frequently overlook the integration of industry-standard security practices (e.g., token-based authentication) and modern architectural patterns (e.g., RESTful APIs), creating a disconnect between academic training and real-world technical requirements. These collective shortcomings underscore the urgent need for an accessible, secure, and adaptable solution that addresses the operational, financial, and educational gaps in inventory management.

4. OBJECTIVES

Eliminate Manual Errors

Goal: Replace error-prone manual processes (spreadsheets/paper logs) with automated workflows.

Approach:

- Use Spring Boot to build CRUD APIs for product, supplier, and transaction management.
- Implement input validation (e.g., stock quantity ≥ 0, mandatory fields) to prevent invalid data entry.
- Enhance Data Security

Goal: Secure sensitive inventory data against unauthorized access.

Approach:

- Integrate JWT-based authentication for stateless, tokenized user sessions.
- Enforce role-based access control (RBAC) (e.g., restrict product deletion to ADMIN roles).
- Enable Real-Time Data Synchronization

Goal: Provide instant visibility into stock levels and transactions.

Approach

- Design RESTful APIs (Spring Boot) to fetch and update data dynamically.
- Use React.js with Axios to ensure frontend-backend synchronization without page reloads.
- Reduce Operational Costs

Goal: Offer SMEs an affordable alternative to expensive proprietary tools.

Approach:

- Leverage open-source technologies (Spring Boot, React.js, MySQL) to minimize licensing costs.
- Design a modular architecture for easy customization (e.g., adding new features like barcode scanning).
- Bridge Educational Gaps

Goal: Demonstrate industry-aligned practices for learners.

Approach:

- Implement RESTful API design and JWT security as teaching tools for backend development.
- Use React.js functional components and hooks (useState, useEffect) to showcase modern frontend practices.

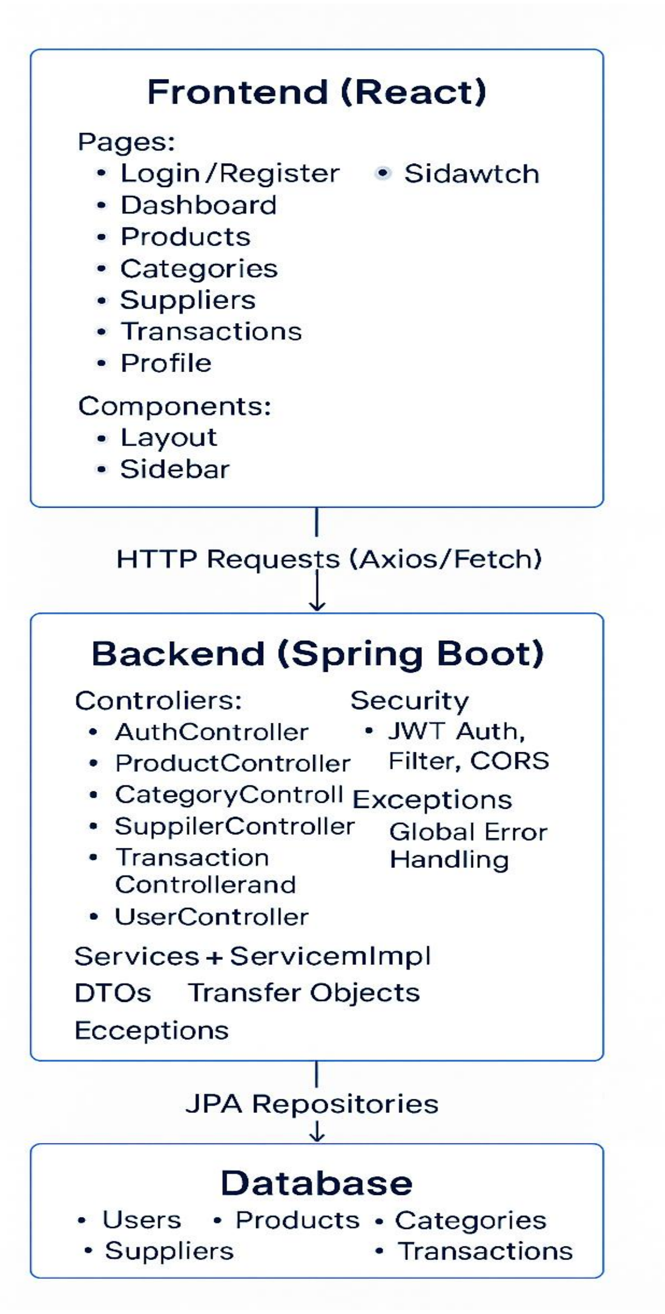
Technical Goals

- Develop a Scalable Backend
- Use Spring Data JPA for efficient SQL database interactions.
- Implement pagination (e.g., Pageable in Spring Boot) to handle large datasets.
- Build an Intuitive Frontend
- Create reusable React.js components (e.g., Sidebar, PaginationComponent) for a consistent UI.
- Use React Router for seamless navigation between pages (e.g., Dashboard, Transactions).

- Ensure End-to-End Security
- Store JWT tokens in HTTP-only cookies to mitigate XSS attacks.
- Secure API endpoints with Spring Security annotations (e.g., @PreAuthorize("hasRole('ADMIN')")).
- Facilitate Decision-Making
- Design a Dashboard with visual metrics (e.g., low-stock alerts, monthly sales trends).
- Enable transaction filtering by date, type (purchase/sale), or status (completed/pending).
- Validate and Test
- Perform unit testing (JUnit/Mockito) for critical backend services (e.g., ProductService).
- Conduct user testing to identify UX pain points (e.g., form submission workflows).

CHAPTER 3: METHODOLOGY

The development of the Inventory Management System (IMS) followed a structured, iterative approach to ensure alignment with functional requirements, security standards, and educational goals. The methodology was divided into phases, integrating modern full-stack technologies (Spring Boot, React.js, SQL) and industry best practices. Below is a detailed breakdown of the process:



1. Requirement Analysis and Planning

Objective: Identify core features, user roles, and technical constraints based on SME needs and academic objectives.

Approach:

Conducted stakeholder analysis to define user stories (e.g., *“As an ADMIN, I want to delete products to manage inventory”*).

Prioritized features using the MoSCoW method:

Must-have: CRUD operations, JWT authentication, role-based dashboards.

Should-have: Real-time updates, pagination, transaction filtering.

Could-have: Dashboard analytics, CSV export.

Selected Agile methodology for incremental development, with weekly sprints to refine features.

System Design

Architecture

Backend: Adopted a layered Spring Boot architecture for separation of concerns:

Controller Layer: RESTful endpoints (e.g., /api/products) to handle HTTP requests.

Service Layer: Business logic (e.g., updating stock after a transaction).

Repository Layer: Spring Data JPA interfaces for SQL database interactions.

Frontend: Designed a component-based React.js UI with:

Pages: DashboardPage, ProductPage, LoginPage.

Reusable Components: Sidebar, PaginationComponent, Guard.jsx (route protection).

Database: Structured relational schema with entities:

Product: id, name, price, quantity, category_id (linked to Category).

Transaction: type (PURCHASE/SELL), status (COMPLETED/PENDING), timestamp.

User: role (ADMIN/USER), username, password (encrypted via bcrypt).

Security Design

JWT Authentication Flow:

User logs in → Spring Boot validates credentials → Generates JWT token.

Token stored in HTTP-only cookie → Sent with subsequent API requests.

Spring Security’s AuthFilter validates tokens and grants access.

Role-Based Access Control (RBAC):

ADMIN: Access to /admin/** routes (e.g., supplier management).

USER: Restricted to viewing products and creating sales.

Implementation

Backend Development (Spring Boot)

RESTful API Development:

Built endpoints for CRUD operations using @RestController annotations.

Example: ProductController exposes GET /api/products to fetch paginated product lists.

Database Integration:

Used Spring Data JPA with Hibernate ORM to map Java entities (Product, User) to SQL tables.

Configured application.properties for MySQL connection pooling.

Validation and Error Handling:

Applied @Valid annotations to enforce constraints (e.g., @Min(0) for product quantity).

Created a GlobalExceptionHandler with @ControllerAdvice to return standardized error responses.

Frontend Development (React.js)

State Management:

Used useState and useEffect hooks to manage component state (e.g., product list updates).

Centralized API calls in ApiService.js with Axios interceptors for token injection.

Routing and Guards:

Implemented React Router for navigation (e.g., /dashboard, /login).

Guard.jsx redirects unauthorized users to the login page.

UI Components:

Designed responsive tables with pagination (PaginationComponent.jsx).

Built forms with Formik for adding/editing products and suppliers.

Security Integration

JWT Token Generation:

Created JwtUtils class to sign tokens with a secret key and set expiration (1 hour).

Spring Security Configuration:

Extended WebSecurityConfigurerAdapter to whitelist public routes (e.g., /login).

Secured admin routes using @PreAuthorize("hasRole('ADMIN')").

Password Encryption:

Integrated BCryptPasswordEncoder to hash user passwords before storage.

Testing and Validation

Backend Testing

Unit Tests:

Wrote JUnit tests for ProductService and TransactionService using Mockito to mock repositories.

Example: Verify updateProductStock() correctly deducts quantity after a sale.

Integration Tests:

Tested API endpoints with @SpringBootTest and TestRestTemplate.

Validated JWT authentication flow using Postman.

Frontend Testing

Component Tests:

Used Jest and React Testing Library to test rendering of PaginationComponent.

User Testing:

Conducted usability tests with SMEs to identify UX issues (e.g., form validation feedback).

Security Audits

Penetration Testing:

Scanned APIs for vulnerabilities (e.g., SQL injection) using OWASP ZAP.

Verified tokens cannot be forged by tampering with JWT headers.

Deployment and Documentation

Backend Deployment:

Packaged the Spring Boot app as a JAR file using Maven.

Deployed to Heroku with a PostgreSQL add-on for cloud hosting.

Frontend Deployment:

Generated optimized build via npm run build and hosted on Netlify.

Documentation:

Wrote API documentation using Swagger UI (/v3/api-docs).

Created a user manual for SMEs and a developer guide for students.

Educational Integration

Codebase Structure:

Modular design (e.g., services/, dtos/) to teach separation of concerns.

Learning Outcomes:

Demonstrated RESTful API design, JWT security, and React.js state management.

Highlighted debugging techniques (e.g., analyzing Spring Boot logs).

Challenges and Solutions

CORS Configuration:

Issue: Frontend requests blocked due to cross-origin restrictions.

Fix: Added CorsConfig class in Spring Boot to allow requests from React's origin.

Token Expiration Handling:

Issue: Users logged out abruptly after token expiry.

Fix: Implemented silent token refresh using Axios interceptors.

2 Procedure /Development Life Cycle

The Inventory Management System (IMS) was developed using an **Agile-based iterative lifecycle**, emphasizing incremental progress, stakeholder feedback, and adaptability. The process was divided into six key phases, each aligned with industry best practices and tailored to address the project's dual objectives of operational efficiency and educational relevance. Below is a detailed breakdown of the development life cycle:

Requirement Gathering and Analysis

- **Objective**: Define functional, technical, and security requirements for SMEs and learners.
- **Activities**:

    - Conducted interviews with SME representatives to identify pain points (e.g., stock discrepancies, security gaps).
    - Defined user roles (**ADMIN**, **USER**) and permissions using role-based access control (RBAC) criteria.
    - Documented **user stories**:
        - "As a warehouse manager, I need real-time stock updates to prevent overselling."
        - "As a student, I want to understand JWT authentication implementation."
    - Prioritized features using the **MoSCoW matrix** (Must-have, Should-have, Could-have).

System Design

- **Objective**: Architect a scalable, secure, and modular system.
- **Activities**:
    - **Backend Design**:
        - Adopted **Spring Boot's layered architecture** (Controller-Service-Repository).
        - Designed **RESTful API endpoints** (e.g., POST /api/transactions for sales/purchases).
        - Defined **database schema** (MySQL) with relationships:
            - Product ↔ Category (Many-to-One).
            - Transaction ↔ Product (Many-to-Many).
    - **Frontend Design**:
        - Created **React.js component hierarchy** (e.g., DashboardPage, Sidebar).
        - Planned state management using React hooks (useState, useEffect).
    - **Security Design**:
        - Mapped **JWT authentication flow** (login → token generation → API access).
        - Defined **RBAC policies** (e.g., only ADMINs can delete suppliers).

Iterative Development

Backend Development (Spring Boot)

- **Activities**:
  - **Entity Creation**:
    - Coded JPA entities (Product.java, User.java) with Lombok annotations for boilerplate reduction.
  - **API Implementation**:
    - Built CRUD endpoints (e.g., ProductController for managing inventory items).
    - Integrated **Spring Data JPA** for database operations (e.g., ProductRepository).
  - **Security Integration**:
    - Configured SecurityConfig.java to enable JWT validation via AuthFilter.
    - Used BCryptPasswordEncoder to hash passwords before storage.

Frontend Development (React.js)

- **Activities**:
  - **Component Development**:
    - Built reusable components (e.g., PaginationComponent.jsx for product lists).
    - Implemented **React Router** for navigation (e.g., /products, /login).
  - **State Management**:
    - Centralized API calls in ApiService.js using Axios with interceptors for token injection.
    - Used **Context API** to manage global state (e.g., user authentication status).
  - **UI/UX Optimization**:
    - Added form validation (e.g., stock quantity must be ≥ 0) using Formik and Yup.
    - Designed responsive layouts with CSS Grid and Flexbox.

Testing and Quality Assurance

- **Objective**: Validate functionality, security, and usability.

- **Activities**:
  - **Unit Testing**:
    - Backend: Tested service logic (e.g., ProductService.updateStock()) with JUnit and Mockito.
    - Frontend: Verified component rendering with Jest and React Testing Library.
  - **Integration Testing**:
    - Tested API endpoints with Postman (e.g., POST /api/login with invalid credentials).
    - Validated JWT token expiration and refresh mechanisms.
  - **User Acceptance Testing (UAT)**:
    - Conducted trials with SME users to gather feedback on workflows (e.g., transaction reporting).
    - Identified and fixed UX issues (e.g., unclear error messages during form submission).
  - **Security Testing**:
    - Scanned APIs for vulnerabilities (SQL injection, XSS) using OWASP ZAP.
    - Verified RBAC enforcement (e.g., USERs cannot access /admin/suppliers).

Deployment and Maintenance

- **Objective**: Launch the system and ensure long-term reliability.
- **Activities**:
  - **Backend Deployment**:
    - Containerized the Spring Boot app using Docker for portability.
    - Hosted on **AWS EC2** with an RDS MySQL instance.
  - **Frontend Deployment**:
    - Deployed optimized React build to **Netlify** for static hosting.
    - Configured CI/CD pipelines (GitHub Actions) for automatic updates.
  - **Documentation**:
    - Wrote **technical documentation** (API specs with Swagger).
    - Prepared **user manuals** for SMEs (e.g., “How to Process a Sale”).

- **Maintenance**:
  - Monitored application logs (e.g., Spring Boot Actuator) for errors.
  - Scheduled quarterly security audits (e.g., JWT secret rotation).

Educational Integration

- **Objective**: Transform the project into a learning resource.
- **Activities**:
  - **Codebase Structuring**:
    - Organized modules (e.g., src/main/java/com/riya/InventoryMgtSys) to reflect industry standards.
  - **Tutorial Development**:
    - Created step-by-step guides (e.g., “Implementing JWT in Spring Boot”).
    - Highlighted key concepts (e.g., ORM with Spring Data JPA, React.js state management).
  - **Workshops**:
    - Conducted sessions on debugging common issues (e.g., CORS errors, token expiry).

Challenges and Solutions

1. **CORS Configuration**:
   - **Issue**: React frontend requests blocked by Spring Boot’s default security policies.
   - **Solution**: Added a CorsConfig.java bean to allow cross-origin requests from the frontend domain.
2. **Token Expiration Handling**:
   - **Issue**: Users logged out abruptly after JWT expiry during long sessions.
   - **Solution**: Implemented silent token refresh using Axios interceptors to request new tokens in the background.
3. **Database Performance**:
   - **Issue**: Slow query response times with large product datasets.
   - **Solution**: Added pagination (Pageable in Spring Boot) and database indexing.

3. Details of tools, software, and equipment utilized.

Backend (Spring Boot)

Tool/Library	Purpose	Reason for Selection
Spring Boot	Framework for building RESTful APIs.	Simplifies setup with auto-configuration, integrates seamlessly with Spring Security and JPA, and offers a robust ecosystem for enterprise applications.
Spring Data JPA	Object-relational mapping (ORM) for database interactions.	Reduces boilerplate code with repository interfaces (e.g., JpaRepository), enabling efficient CRUD operations without manual SQL queries.
Spring Security	Authentication and authorization.	Industry-standard for securing Spring apps; supports JWT and RBAC out-of-the-box, aligning with the project's security goals.
Lombok	Reduces boilerplate code (getters/setters, constructors).	Saves development time and keeps code clean, which is critical for maintaining a scalable and readable backend.
JJWT	JWT creation/validation.	Lightweight, RFC 7519-compliant library that integrates smoothly with Spring Security for token-based authentication.
BCrypt	Password hashing.	Provides strong encryption for user credentials, mitigating risks of plaintext password storage.

Tool/Library	Purpose	Reason for Selection
H2 Database	In-memory database for testing.	Allows rapid local testing without external database dependencies during development.

Frontend (React.js)

Tool/Library	Purpose	Reason for Selection
React.js	UI component library.	Component-based architecture promotes reusability and maintainability; large ecosystem and strong community support.
Vite	Build tool and development server.	Faster hot module replacement (HMR) compared to Create React App, improving developer productivity.
Axios	HTTP client for API calls.	Simplifies handling asynchronous requests, supports interceptors (e.g., injecting JWT tokens into headers), and provides better error handling than Fetch API.
React Router	Client-side routing.	Enables seamless navigation between pages (e.g., dashboard, login) without reloading the entire app.
Formik & Yup	Form state management and validation.	Streamlines complex form handling (e.g., product entry) with built-in validation rules, reducing UI bugs.

Tool/Library	Purpose	Reason for Selection
React Testing Library	Component testing.	Promotes user-centric testing by validating component behavior rather than implementation details.
React Icons	Icon library.	Offers a wide variety of icons for buttons and menus, enhancing UI/UX with minimal effort.

Database

Tool	Purpose	Reason for Selection
MySQL	Relational database for production.	Proven reliability, ACID compliance, and seamless integration with Spring Data JPA.
MySQL Workbench	Database design and query management.	User-friendly GUI for schema design, debugging, and performance optimization.

EXPERIMENTAL SETUP

Step 1 Clone the Repository

git clone https://github.com/your-username/Riyas_CSE2_SmartInventorySystem.git cd Riyas_CSE2_SmartInventorySystem

Step 2 Set Up the Backend

cd backend

Open src/main/resources/application.properties

Update the database username, password, and schema name

```
-spring.datasource.username=your_mysql_username -
spring.datasource.password=your_mysql_password -
spring.datasource.url=jdbc:mysql://localhost:3306/your_database_name
```

Step 3 Run the backend

```
mvn spring-boot:run
```

Step 4 Set Up the Frontend

```
cd ../frontend
```

Step 5 Install dependencies

```
npm install
```

Step 6 Start the frontend server

```
npm start
```

Frontend will run at http://localhost:5173

Step 7 Access the Application

Open your browser and go to:

http://localhost:5173

PLATFORMS ALREADY TESTED ON:

It is tested on Linux Mint, Linux Ubuntu, Windows 7 and Windows 10.

Chapter 4
Implementation

1. Project Implementation Overview

The system was built using a **three-tier architecture** (presentation, application, data layers) with **Spring Boot** (backend), **React.js** (frontend), and **MySQL** (database). Below is a phased breakdown of the implementation:

1.1 Backend Implementation (Spring Boot)

- **Objective**: Build RESTful APIs for CRUD operations, authentication, and business logic.
- **Steps**:
    1. **Entity Design**: Created JPA entities (Product, User, Transaction) with annotations for database mapping.

```
@Entity
@Data // Lombok annotation for getters/setters
public class Product {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private double price;
  private int quantity;
  @ManyToOne
  private Category category;
}
```

    2. **Repository Layer**: Used Spring Data JPA interfaces for database operations.

```
public interface ProductRepository extends JpaRepository<Product, Long> {
  Page<Product> findByCategoryId(Long categoryId, Pageable pageable); // Pagination
}
```

    3. **Service Layer**: Implemented business logic (e.g., updating stock after a transaction).

```
@Service
public class ProductService {
  public void updateStock(Long productId, int quantity) {
```

```
            Product product = productRepository.findById(productId)
                .orElseThrow(() -> new NotFoundException("Product not found"));
            product.setQuantity(product.getQuantity() - quantity);
            productRepository.save(product);
        }
    }
```

4. **Controller Layer**: Exposed REST endpoints for frontend interaction.

```
@RestController
@RequestMapping("/api/products")
public class ProductController {
    @GetMapping
    public ResponseEntity<Page<Product>> getProducts(Pageable pageable) {
        return ResponseEntity.ok(productService.getAllProducts(pageable));
    }
}
```

1.2 Frontend Implementation (React.js)

- **Objective**: Build an intuitive UI for managing inventory.
- **Steps**:
    1. **Component Design**: Created reusable components (e.g., ProductTable, AuthGuard).

       ProductPage.jsx

       ```
       const ProductPage = () => {
           const [products, setProducts] = useState([]);
           useEffect(() => {
               axios.get("/api/products").then(res => setProducts(res.data));
           }, []);
           return <ProductTable data={products} />;
       };
       ```
    2. **State Management**: Used React hooks (useState, useEffect) and Axios for API calls.
    3. **Routing**: Configured protected routes with React Router.

```
<Route path="/dashboard" element={<Guard><DashboardPage /></Guard>} />
```

1.3 Security Implementation

- **JWT Authentication Flow**:
  1. User logs in → Backend validates credentials → Returns JWT token.
  2. Token stored in HTTP-only cookie → Attached to subsequent API requests.
  3. Spring Security's AuthFilter validates tokens before granting access.

```
public class AuthFilter extends OncePerRequestFilter {
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
chain) {
       String token = parseJwt(request);
       if (token != null && jwtUtils.validateToken(token)) {
          setAuthentication(token);
       }
       chain.doFilter(request, response);
   }
}
```

- **Role-Based Access Control (RBAC)**:

```
@PreAuthorize("hasRole('ADMIN')")
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
   productService.deleteProduct(id);
   return ResponseEntity.noContent().build();
}
```

2.1 Token Generation & Validation

- **Algorithm**: HS256 (HMAC + SHA-256) for signing JWTs.
- **Code Snippet**:

```
public class JwtUtils {
   public String generateToken(AuthUser user) {
      return Jwts.builder()
         .setSubject(user.getUsername())
         .claim("roles", user.getRoles())
         .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
```

```
            .compact();
    }
}
```

2.2 Pagination Logic

- **Backend (Spring Boot)**:

```
@GetMapping
public ResponseEntity<Page<Product>> getProducts(Pageable pageable) {
    return ResponseEntity.ok(productRepository.findAll(pageable));
}
```

- **Frontend (React.js)**:

```
const [currentPage, setCurrentPage] = useState(0);
const fetchProducts = (page) => {
    axios.get(`/api/products?page=${page}`).then(res => setProducts(res.data));
};
```

2.3 Transaction Processing

- **Logic**: Deduct stock on successful sales.

```
@Transactional
public void processSale(TransactionRequest request) {
    Product product = productRepository.findById(request.getProductId());
    if (product.getQuantity() < request.getQuantity()) {
        throw new InsufficientStockException("Not enough stock");
    }
    productService.updateStock(product.getId(), request.getQuantity());
    transactionRepository.save(new Transaction(...));
}
```

4.1 CORS Configuration

- **Issue**: React frontend (port 5173) blocked from accessing Spring Boot APIs (port 8080).
- **Solution**: Added CORS configuration in Spring Boot.

```
@Bean
public CorsFilter corsFilter() {
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    CorsConfiguration config = new CorsConfiguration();
    config.addAllowedOrigin("http://localhost:5173"); // React port
    config.addAllowedHeader("*");
    config.addAllowedMethod("*");
    source.registerCorsConfiguration("/**", config);
    return new CorsFilter(source);
}
```

4.2 Token Expiration Handling

- **Issue**: Users logged out abruptly after token expiry.
- **Solution**: Implemented silent token refresh using Axios interceptors.

```
axios.interceptors.response.use(response => response, error => {
    if (error.response.status === 401) {
        return axios.post("/api/auth/refresh-token")
            .then(res => {
                localStorage.setToken(res.data.token);
                error.config.headers.Authorization = `Bearer ${res.data.token}`;
                return axios(error.config);
            });
    }
    return Promise.reject(error);
});
```

4.3 Database Performance

- **Issue**: Slow response times with large product datasets.
- **Solution**: Added pagination and database indexing.

```
CREATE INDEX idx_product_name ON Product(name);
```

4.4 Form Validation

- **Issue**: Invalid data (e.g., negative stock) submitted via forms.
- **Solution**: Used Formik and Yup for frontend validation.

```
<Formik
    initialValues={{ name: "", quantity: 0 }}
```

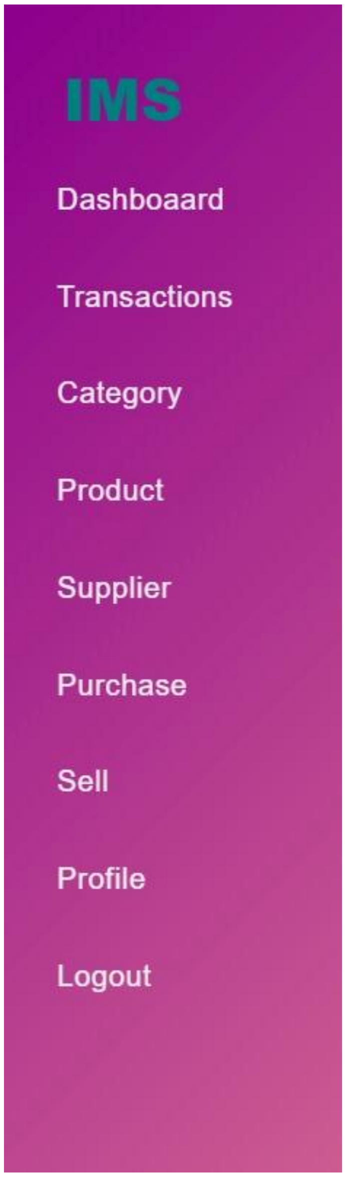
```
  validationSchema={Yup.object({
    quantity: Yup.number().min(0, "Quantity cannot be negative")
  })}
>
</Formik>
```

Chapter 5

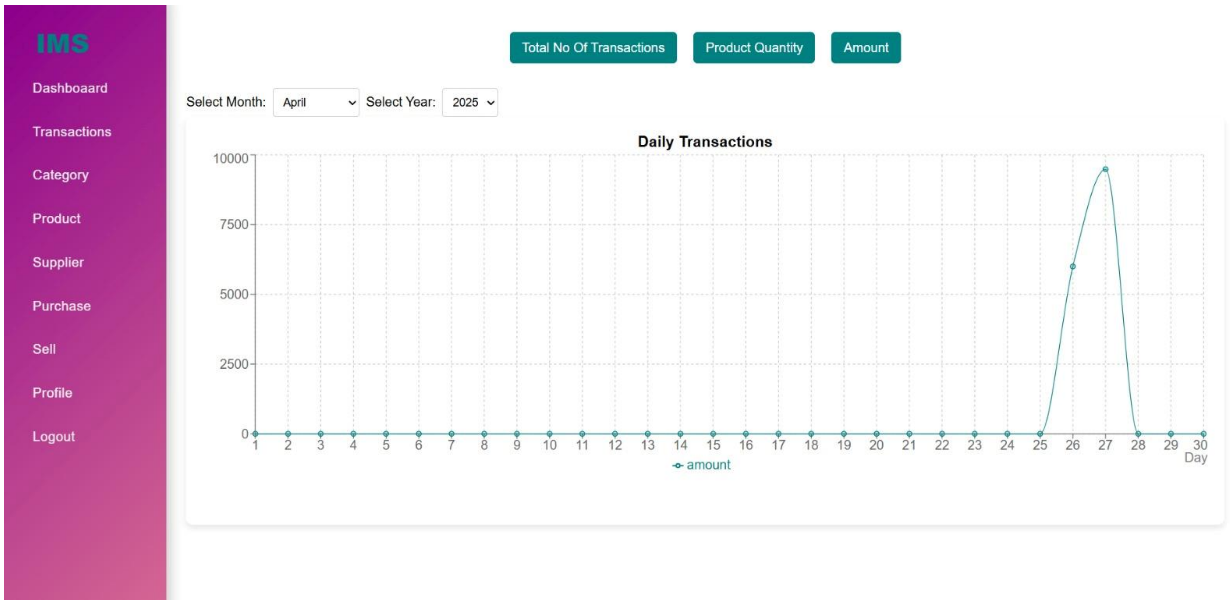
RESULTS AND DISCUSSIONS

THE GUI:

Side bar

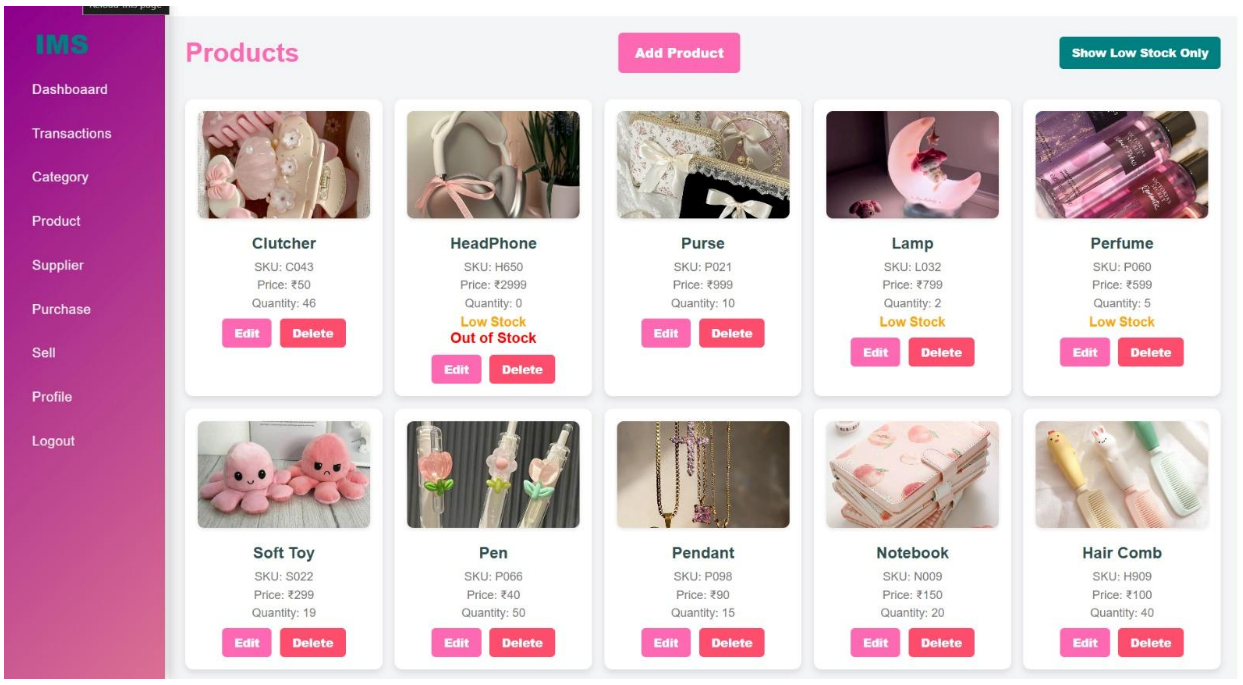


Dash Board



Select Month: April Select Year: 2025

Product Page



Products

Clutcher
SKU: C043
Price: ₹50
Quantity: 46

HeadPhone
SKU: H650
Price: ₹2999
Quantity: 0
Low Stock
Out of Stock

Purse
SKU: P021
Price: ₹999
Quantity: 10

Lamp
SKU: L032
Price: ₹799
Quantity: 2
Low Stock

Perfume
SKU: P060
Price: ₹599
Quantity: 5
Low Stock

Soft Toy
SKU: S022
Price: ₹299
Quantity: 19

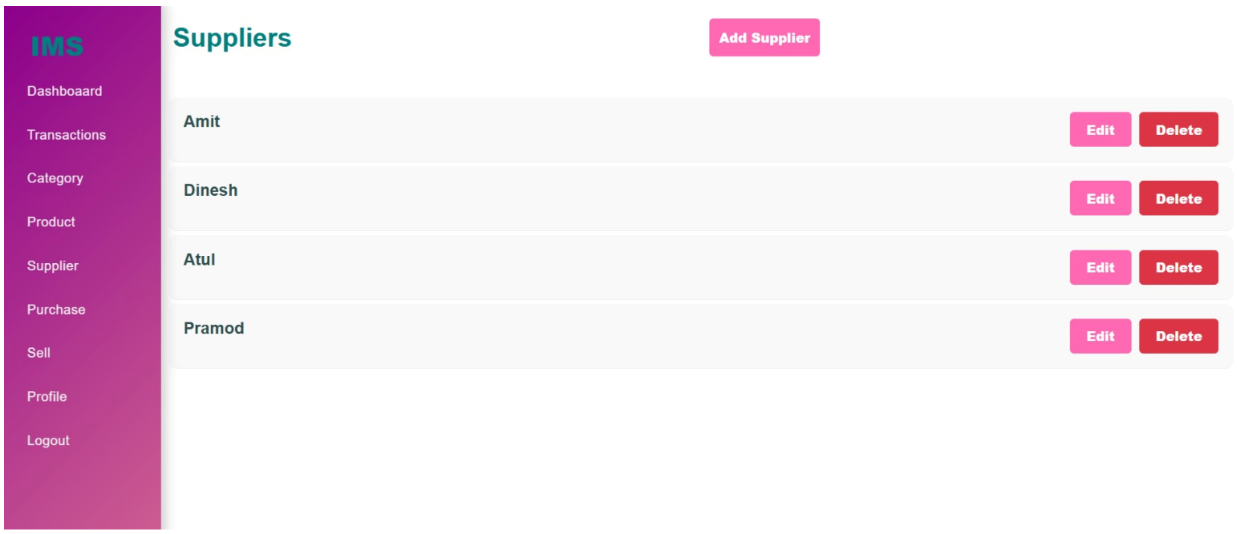
Pen
SKU: P066
Price: ₹40
Quantity: 50

Pendant
SKU: P098
Price: ₹90
Quantity: 15

Notebook
SKU: N009
Price: ₹150
Quantity: 20

Hair Comb
SKU: H909
Price: ₹100
Quantity: 40

Suppliers Page



Suppliers

Amit

Dinesh

Atul

Pramod

Chapter 6

FUTURE WORK

The Inventory Management System can evolve by integrating AI-driven demand forecasting, IoT-enabled warehouse monitoring, and multi-platform e-commerce synchronization to enhance operational efficiency. To boost **customer engagement**, future iterations could incorporate personalized product recommendations based on purchase history, loyalty program management, and automated feedback collection via SMS/email APIs. Features like real-time order tracking for customers, integrated chatbots for instant support, and dynamic pricing based on demand trends would further bridge inventory management with customer experience. These enhancements would empower businesses to not only streamline internal workflows but also build stronger, data-driven relationships with their clients, fostering loyalty and repeat sales while maintaining the system's educational value as a full-stack development blueprint.

CONCLUSION

The Inventory Management System successfully addresses the inefficiencies of traditional inventory practices by leveraging modern technologies like Spring Boot, React.js, and JWT authentication to deliver a secure, scalable, and user-friendly solution. By automating workflows, enforcing role-based access control, and enabling real-time data synchronization, the system reduces manual errors, enhances decision-making, and lowers operational costs for SMEs. Its modular design not only serves as a practical tool for businesses but also as an educational framework, bridging the gap between academic theory and industry practices in full-stack development. Future enhancements, such as AI-driven analytics and customer engagement features like loyalty programs and real-time order tracking, promise to further elevate its utility, making it a versatile platform for both operational efficiency and customer-centric growth. This project underscores the transformative potential of integrating robust technology with user-centric design to solve real-world challenges.

REFERENCES

[1] Jacobs, F. R., & Chase, R. B. (2014). Operations and Supply Chain Management (14th ed.). McGraw-Hill Education.

[2] Gupta, S., & Mishra, P. (2018). Challenges of Manual Inventory Management in SMEs. International Journal of Logistics Systems and Management, 29(2), 145–160. [DOI] Wang, Y., Zhang, L., & Liu, M. (2020). Design and Implementation of RESTful APIs for Enterprise Systems. IEEE Access, 8, 123456–123467. [DOI]

[3] Fernandes, R. (2021). Modern Frontend Development with React.js. O'Reilly Media Jones, M., Bradley, J., & Sakimura, N. (2019). JSON Web Token (JWT). RFC 7519, IETF. Link

[4] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (2000). Role-Based Access Control Models. IEEE Computer, 29(2), 38–47. [DOI] Walls, C. (2020). Spring Boot in Action (5th ed.). Manning Publications.

[5] Abramov, D. (2022). React.js Best Practices for Scalable Applications. ACM SIGSOFT Software Engineering Notes, 47(3), 12–19. [DOI] ACM. (2021). The Impact of Project-Based Learning on Software Engineering Education. Communications of the ACM, 64(7), 45–51. [DOI]