INVENTORY MANAGEMENT SYSTEM

MEMBERS:

1.ASWATHY SHYJI

2.BHAVYA P

3.SAHADUDHEEN B

4.SAJJAD P

Abstract

Inventory management is one of the most important tasks in any organization that deals with products, whether it is a retail store, wholesale distributor, or a manufacturing unit. Manual record-keeping using notebooks or spreadsheets often leads to inaccuracies, duplication, delays, and poor decision-making.

The Inventory Management System (IMS) is a web-based application built using Java (Spring Boot) and MySQL that aims to overcome these challenges. The system automates inventory tracking, sales recording, stock updates, and report generation. Users can access the system through a simple, browser-based interface, ensuring real-time updates and accuracy.

The project emphasizes simplicity, efficiency, and scalability. It is designed for small to medium-scale businesses that need a lightweight yet powerful inventory solution. With features like user authentication, CRUD operations, sales tracking, and reports, IMS demonstrates the importance of full-stack development using Java + MySQL + Thymeleaf.

Work Completed So Far

We have followed a step-by-step software development approach. Each stage contributed to a clearer, more structured project outcome.

1. Abstract Creation

The abstract was the foundation of our project planning. It summarized:

1.Problem Statement: Businesses need a more accurate, efficient way to manage stock.

2.Proposed Solution: A Java-based Inventory Management System with database integration.

3.Key Features: Product management, stock tracking, sales management, report generation.

4.Technologies: Java 21, Spring Boot, MySQL, Thymeleaf, Maven.

This step gave us a clear vision of what we were building.

2. Detailed Design Document

The Detailed Design Document (DDD) explained the blueprint of the system. It included:

System Modules:

a. Product Management (add, edit, delete, view)

b. Stock Management (monitor stock levels, reorder points)

c. Sales Management (record customer purchases, update stock automatically)

d. Reporting (generate stock and sales reports)

System Architecture:

a. Presentation Layer → Thymeleaf templates (HTML pages)

b. Business Layer → Service classes (Java)

c. Data Access Layer → JPA Repositories

d. Database → MySQL

->Workflow Example: When a user records a sale, the system updates both the sales table and the stock table simultaneously.

The DDD gave us clarity on how each feature interacts with the others.

3. Complete Class Diagram

We designed a UML Class Diagram that showed all the important entities:

>User: Attributes → id, username, password, role

>Product: Attributes → productId, name, category, price

>Stock: Attributes → stockId, productId, quantity, reorderLevel

>Sales: Attributes → salesId, productId, quantitySold, date

>Report: Attributes → reportId, type, generatedDate

Relationships:

>One Product can have many Sales records.

>One Product corresponds to one Stock record.

>Reports can be generated from both Product and Sales.

This diagram was crucial in helping us design the database schema.

4. Functional Flow Diagram

We created a Functional Flow Diagram to show the user journey:

1. User Login → verifies credentials from users table.

2. Dashboard → displays product summary, stock alerts, and sales updates.

3. Add Product → inserts data into products and stock.

4. Update Stock → modifies stock table.

5. Record Sale → adds entry into sales and decreases quantity in stock.

6. Generate Report → pulls data from multiple tables and displays summary.

This ensured that every user action was mapped to a backend process.

5. Database/File Storage Design

We designed the database schema using MySQL Workbench.

>users: userId, username, password, role

>products: productId, name, category, price

>stock: stockId, productId, quantity, reorderLevel

>sales: salesId, productId, quantitySold, saleDate

>reports: reportId, type, generatedDate

Features:

>Primary keys for unique identification.

>Foreign keys for relationships (e.g., productId in stock and sales).

>Normalization to avoid redundancy.

>Constraints like NOT NULL and AUTO_INCREMENT for reliability.

The database was tested with sample data in MySQL Workbench.

6. User Interaction Diagram

The User Interaction Diagram explained how different roles use the system:

Admin:

>Add users

>Manage product categories

>Access all reports

Employee/User:

>Add/edit/delete products

>Record sales

>View stock reports

System:

>Auto-update stock on sale

>Generate notifications if stock is low

>Produce daily/weekly/monthly sales reports

This helped in ensuring role-based access control.

7. Prototype Creation

We built a prototype using HTML + Thymeleaf templates.

Pages included:

>Login Page – simple username/password validation

>Dashboard – summary of stock and sales

>Product Page – form to add or update products

>Sales Page – form to record transactions

>Reports Page – table displaying sales and stock summary

This prototype was shared with the team to gather feedback before coding the backend.

8. Source Code Development

We initialized a Spring Boot project with Maven and created the following structure:

```
InventoryWeb/                    # Root Project Folder
|
 ├─ src/                         # Java source code
|   └─ com/ims/                  # Your package
|       ├─ model/                # POJOs (User, Product,
Supplier, Sale)
|       |    ├─ User.java
|       |    ├─ Product.java
|       |    ├─ Supplier.java
|       |   └─ Sale.java
|       |
|       ├─ dao/                  # Data Access Objects
|       |    ├─ UserDAO.java
|       |    ├─ ProductDAO.java
|       |    ├─ SupplierDAO.java
|       |   └─ SalesDAO.java
|       |
|       ├─ servlet/                # Servlets (controllers)
|       |     ├─ LoginServlet.java
|       |     ├─ ProductServlet.java
|       |     ├─ SupplierServlet.java
|       |    └─ SalesServlet.java
|       |
|       └─ util/                 # Utility classes
|            └─ DBConnection.java
|
├─ WebContent/                    # Web resources (JSP, CSS,
JS)
|    ├─ index.jsp                # Login page
|    ├─ dashboard.jsp              # Dashboard page
|    ├─ products.jsp              # Product management
|    ├─ suppliers.jsp             # Supplier management
|    ├─ sales.jsp                # Sales management
|    ├─ css/                   # Stylesheets
|    |    └─ style.css
|    ├─ js/                    # JavaScript files
|    |    └─ script.js
|    |
|    └─ WEB-INF/                  # Configurations (not publicly
accessible)
```

```
|        ├– web.xml               # Deployment descriptor
|       └─ lib/                   # External libraries
|           └─ mysql-connector-j-9.4.0.jar
|
└─ lib/                           # (optional, if your IDE needs it)
   └─ mysql-connector-j-9.4.0.jar
```

Technologies used:

>Spring Web → to handle HTTP requests

>Spring Data JPA → to simplify database queries

>Thymeleaf → to render dynamic HTML pages

>MySQL Driver → to connect Java application with MySQL database

So far, we have completed the model classes and repositories. We are currently coding the services and controllers.

9. Current Progress (Ongoing Work)

We are now in the integration stage:

>Implementing controllers for CRUD operations.

>Connecting the application with MySQL.

>Testing user authentication and login.

>Building Thymeleaf pages for product and sales management.

>Debugging Maven errors and ensuring dependencies are resolved.

The system is partially functional. Product models and repositories are complete, and we are working on controllers and UI.

10. Remaining Work

The following tasks are in progress or pending:

1. Service Layer Completion – finalize business logic for all modules.

2. Controller Development – handle all requests from UI.

3. Integration with Thymeleaf – display real-time data in web pages.

4. Validation and Error Handling – ensure only valid inputs are processed.

5. Testing

>Unit Testing (for services)

>Integration Testing (for controllers and database)

>User Testing (simulate actual usage)

6. Deployment – host the project on localhost for demo and later push to GitHub.

7. Final Documentation – prepare user manual and final PDF report.

11. Conclusion

The Inventory Management System project has evolved from concept to implementation. We have created all required documentation:

>Abstract

>Detailed Design Document

>Class Diagrams

>Functional Flow Diagram

>Database Schema

>User Interaction Diagram

>Prototype

>Source Code

We are now working on the final development stage. Once completed, the system will:

>Provide real-time inventory tracking

>Simplify sales and stock management

>Generate business reports for decision-making

>Be accessible as a web-based solution

This project demonstrates our ability to use Java, Spring Boot, MySQL, and Thymeleaf in a real-world web application. It has been a learning experience in software design, coding, testing, and documentation.

12. References

>Oracle Java 21 Documentation

>Spring Boot Guides (spring.io)

>MySQL Documentation

>UML & Software Engineering Best Practices