

Inventory Management System using Spring Boot

A PROJECT REPORT

Submitted by

Manu Bansal – 22BCS15969

Abhishek Kumar Dwivedi – 22BCS15991

Rajat Katiyar – 22BCS15928

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING



May 2025

BONAFIDE CERTIFICATE

Certified that this project report “**Inventory Management System using Spring Boot**” is the bonafide work of “**Manu Bansal (22BCS15969), Rajat Katiyar (22BCS15928) and Abhishek Kumar Dwivedi (22BCS15991)**” who carried out the project work under my/our supervision.

SIGNATURE

Dr. Sandeep Singh Kang
Administrative Director
Department of Computer Science
and Engineering

SIGNATURE

Er. Shubham Mishra
Supervisor
Department of Computer Science
and Engineering

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGMENT

We express our profound gratitude to **Chandigarh University, Punjab**, for providing the necessary resources and a conducive research environment for the successful completion of this project on "**Inventory Management System.**"

We extend our sincere appreciation to our research supervisor, **Er. Shubham Mishra**, for his invaluable guidance, constructive feedback, and continuous support throughout the course of this project work. His expertise and encouragement have played a pivotal role in shaping the direction of this study.

We also acknowledge the collaborative efforts of our research team:

- **Manu Bansal (22BCS15969)**
- **Rajat Katiyar (22BCS15928)**
- **Abhishek Kumar Dwivedi (22BCS15991)**

Their dedication, intellectual contributions, and collective efforts have been instrumental in the completion of this work. Additionally, we extend our gratitude to the faculty members of the university for their academic insights and support, as well as to our peers and families for their encouragement throughout this research endeavor.

Authors

Manu Bansal, Rajat Katiyar, Abhishek Kumar Dwivedi
Department of Computer Science and Engineering
Chandigarh University, Punjab

TABLE OF CONTENT

CHAPTER 1. INTRODUCTION	05
1.1. Identification of client/need/relevant contemporary issue	05
1.2. Identification of problem	05
1.3. Identification of tasks	06
1.4. Timeline of the project development	06
1.5. Organization of report	06
CHAPTER 2. LITERATURE REVIEW	08
2.1. Timeline of the reported problem	08
2.2. Existing Solutions	08
2.3. Bibliometric Analysis	09
2.4. Review Summary	09
2.5. Problem Definition	10
CHAPTER 3. DESIGN FLOW AND PROCESS	11
3.1. Design Flow	11
3.2. Design Constraints	12
3.3. Feature Selection	12
CHAPTER 4. RESULT ANALYSIS AND VALIDATION	14
4.1. Implementation	14
4.2. Result Analysis	15
CHAPTER 5. CONCLUSION AND FUTURE WORK	18
5.1. Key Findings	18
5.2. Future Scope	19
REFERENCES	20

CHAPTER 1

INTRODUCTION

Inventory management is a critical component in many businesses, helping to keep track of stock, manage product movement, and ensure that supply meets demand efficiently. This project aims to develop a lightweight **Inventory Management System (IMS)** using **Java and Spring Boot**, specifically tailored to perform essential **CRUD operations** on product data using **Java Collections Framework** instead of an external database.

The application leverages **Spring Boot with Thymeleaf for the frontend**, integrates **Spring Web for REST APIs**, and incorporates **Lombok** for concise model classes. The goal is to provide a functional, responsive, and user-friendly inventory management tool suitable for small to medium-sized businesses, especially where quick deployment without database dependency is beneficial.

1.1 Identification of Client / Need / Relevant Contemporary Issue

With the rise of e-commerce, retail chains, and digital-first inventory systems, the need for accessible and efficient inventory management software has grown significantly. Businesses often require a simple solution that works offline or within a local environment without relying on complex databases or external services.

Importance and Applications:

- **Retail and E-commerce:** Managing stock, sales, and returns.
- **Small Businesses:** Need lightweight systems with no DB dependency.
- **Educational Institutions:** Practical demo systems for teaching CRUD, APIs, Spring Boot, and Java Collections.
- **Offline Environments:** Operates without database integration for remote setups.

1.2 Identification of Problem

Many existing inventory management systems are either too complex, expensive, or require heavy backend database support. For academic or quick-prototyping purposes, there's a lack of lightweight Java-based systems that:

- Store everything in memory using data structures like ArrayList, Map, etc.
- Offer a complete frontend experience with Thymeleaf templates.
- Allow API-based access and control.
- Use standard Java + Spring Boot stack with minimal setup.

Challenges Addressed:

- No dependency on databases.
- Real-time CRUD operations with simple REST + MVC integration.
- Full-stack experience using Java ecosystem only.

- Use of in-memory structures for simplicity and speed.

1.3 Identification of Tasks

To accomplish the creation of this inventory system, the project was divided into several phases, from planning to implementation and testing. Each task ensures a modular and scalable approach.

Key Tasks in the Project:

- **Requirement Analysis:** Understand key operations like add, delete, update, and fetch items.
- **Model Creation:** Use Lombok to define Product, Category, etc., with Java classes.
- **API Development:** Implement controllers for product management via Spring Web.
- **Frontend UI:** Design HTML templates using Thymeleaf, CSS, and JavaScript.
- **Data Handling:** Use List, Map, and other Java Collections to store and manage data.
- **Integration:** Connect frontend forms with backend controllers.
- **Testing:** Ensure all APIs and forms work correctly and handle edge cases.
- **Documentation:** Summarize methods, endpoints, and use cases.

1.4 Timeline of Project Development

A week-wise breakdown was planned to align with academic timelines, ensuring steady progress across all stages of the project:

Week	Milestone
Week 1–2	Requirement analysis and data structure planning
Week 3–4	Backend setup with Spring Boot, Lombok, and Controller skeleton
Week 5–6	Thymeleaf integration and frontend UI development
Week 7–8	Implementation of all CRUD operations
Week 9–10	Integration testing and bug fixing
Week 11–12	Report writing and final demo preparation

1.5 Organization of the Report

This report is structured in the following chapters to provide a comprehensive walkthrough of the system's objectives, implementation, and outcomes.

Report Structure:

- **Chapter 1: Introduction** – Project overview, objectives, and planning.
- **Chapter 2: Literature Review** – Summary of related works and system comparison.

- **Chapter 3: Design Flow and Process** – Software architecture, technologies used, and system design.
- **Chapter 4: Implementation and Result Analysis** – Backend logic, controller APIs, and frontend forms with output screenshots.
- **Chapter 5: Conclusion and Future Work** – Final thoughts and possible enhancements.

This structured approach ensures clarity in understanding the system's purpose, technical development, and practical utility.

In conclusion, this Inventory Management System project lays the groundwork for building a modern, responsive, and modular web application using the Java Spring Boot ecosystem. By integrating essential technologies such as **Spring Web for controller and API handling**, **Thymeleaf for dynamic user interfaces**, **Lombok for reducing boilerplate code**, and **Java's Collection Framework** for data storage, the system ensures ease of development, maintainability, and real-time performance without relying on external databases. This approach not only simplifies deployment but also enhances understanding of core backend and frontend integration for budding developers and academic environments. The structured task list and timeline offer a clear development path, while the real-world relevance of inventory management in businesses—from retail to logistics—highlights its practical utility. As we progress through the report, we delve deeper into existing solutions, system architecture, implementation logic, and evaluation to showcase how such a system can be both educationally enriching and functionally effective.

CHAPTER 2

LITERATURE REVIEW

Inventory management systems have evolved significantly over time, transitioning from manual record-keeping to complex, database-driven enterprise solutions. However, for lightweight applications or educational purposes, there is still a gap in solutions that avoid database dependency and instead use in-memory data handling via programming constructs like Java Collections. This chapter explores the evolution of inventory systems, existing solutions, their architectural patterns, and identifies the relevance of building a system using Spring Boot with collections as a backend. It also compares popular approaches and identifies the scope of our project within this landscape.

2.1 Timeline of the Reported Problem

The need to manage inventory efficiently has existed for centuries. With the digital transformation of businesses, the focus shifted from manual ledgers to software-based systems. Over time, various technologies have been used to build inventory systems, each offering different capabilities and complexity.

Key Milestones in Inventory Management Evolution:

- **Pre-1990s – Manual Records:** Inventory was maintained using handwritten logs, spreadsheets, and basic desktop tools like MS Excel.
- **1990s – Early Digital Systems:** Standalone inventory software emerged, built using C/C++ or Visual Basic, storing data in local databases.
- **2000s – Web-based Systems:** PHP, ASP.NET, and JSP frameworks enabled browser-accessible inventory systems backed by SQL databases.
- **2010s – Cloud and ERP Integrations:** Advanced inventory modules became part of large ERP systems (like SAP, Oracle NetSuite).
- **Present – Lightweight RESTful Architectures:** Developers now use frameworks like **Spring Boot** to build scalable, modular inventory systems. For prototypes or educational use, **in-memory storage via Java Collections** is gaining popularity for its simplicity and effectiveness.

2.2 Existing Solutions

Many full-fledged inventory systems exist, often using robust database solutions and cloud services. However, lightweight systems that use Java's in-built data structures for in-memory storage are rare, despite their usefulness for learning, prototyping, or running small-scale operations.

Popular Approaches & Their Characteristics:

- **Database-Driven Inventory Systems**
 - Uses MySQL, PostgreSQL, MongoDB, etc.
 - Supports large datasets with persistent storage

- Requires setup, configuration, and connectivity
- **Spring Boot with JPA & Hibernate**
 - Commonly used in enterprise-grade systems
 - Simplifies database interaction through ORM
 - Adds complexity for simple use cases
- **In-Memory Java Collection-Based Systems** (*our approach*)
 - Uses ArrayList, HashMap, etc., to store and retrieve products
 - Ideal for prototypes, demos, and learning projects
 - Fast, no setup, but non-persistent
- **NoSQL or Cloud-Hosted APIs** (Firebase, AWS DynamoDB)
 - Great for scaling
 - Requires internet and account setup
 - Not suitable for offline, beginner-level learning

2.3 Bibliometric Analysis

To understand current trends, academic research, and industry practices, a bibliometric analysis of inventory system-related literature reveals the increasing focus on microservices, REST APIs, and simplicity.

Key Insights from Literature and Code Repositories:

- **Growing Adoption of Spring Boot** for backend development in Java due to its simplicity and ecosystem support.
- **Thymeleaf and MVC** are widely adopted for Java web projects where frontend and backend are tightly integrated.
- **Educational Projects** prefer **Java Collections** for data handling, avoiding the overhead of database setup.
- **GitHub Repositories** show increasing numbers of inventory-based apps built with Java + Spring Boot for CRUD demos and internal tools.
- **YouTube & Blogs** are rich in tutorials promoting full-stack apps with in-memory storage for faster prototyping and explanation.

2.4 Review Summary

The existing literature and open-source implementations show a strong shift towards using lightweight frameworks for faster web application development. While enterprise systems still rely on databases and cloud platforms, smaller use cases—such as academic projects or internal tools—can benefit from in-memory data handling.

Summary Points:

- **Spring Boot** is the preferred backend framework for Java-based web applications.
- **In-memory CRUD applications** serve well in classroom, demo, and rapid prototyping environments.
- **Thymeleaf + Spring MVC** offers seamless integration between frontend and backend.
- **Java Collections** (like List, Map) provide efficient storage and manipulation of runtime data.
- There's a lack of structured, documented examples using this specific combination—filling this gap is a key objective of this project.

2.5 Problem Definition

This project aims to develop a **Java-based Inventory Management System** using **Spring Boot** without any traditional database. The main challenge lies in designing a functional and user-friendly system where all data is stored using **Java Collections** (like ArrayList, HashMap, etc.) while still supporting **complete CRUD functionality**, **REST APIs**, and **form-based interactions** through **Thymeleaf and Spring Controllers**.

This addresses the problem faced by learners and developers who need a clean, database-free inventory app that is fast, modular, and demonstrates core concepts of Spring Boot, APIs, MVC architecture, and frontend integration—without the overhead of DB setup or cloud services.

In conclusion, the literature and analysis clearly highlight a growing demand for lightweight, modular, and database-free web applications—especially in the context of learning, prototyping, and small-scale business utilities. While most existing inventory systems are tied to complex database structures and enterprise frameworks, there is a noticeable gap in simple, Spring Boot-based applications that utilize in-memory data storage through Java Collections. This project aims to bridge that gap by delivering a functional inventory management system that is easy to understand, quick to deploy, and effective in demonstrating the core principles of full-stack Java web development. By combining RESTful API design, Spring MVC, and a dynamic Thymeleaf-driven frontend, the system ensures a modern user experience while remaining lightweight and accessible. This chapter forms the foundation for selecting the appropriate design, tools, and architecture that will be discussed in the following section.

CHAPTER 3

DESIGN FLOW AND PROCESS

Designing an inventory management system requires a clear understanding of user requirements, appropriate selection of technologies, and a structured workflow that guides the development process. In this chapter, we outline the design flow of the system, detail the development environment, highlight key constraints, and explain the reasoning behind selected features. The goal is to ensure smooth integration between backend logic and frontend UI, enabling efficient CRUD operations using Java's in-memory collection framework—all developed and tested using **IntelliJ IDEA**.

3.1 Design Flow

The project is developed using a layered architecture to separate concerns and maintain modularity. It follows the **MVC (Model-View-Controller)** pattern to enable smooth communication between user interface, business logic, and data storage layers.

Design Flow Overview:

1. Requirement Analysis

- Define core functionalities: Add, View, Update, and Delete Products.
- Understand user roles and basic interactions with inventory data.

2. Project Setup in IntelliJ IDEA

- Create a Spring Boot project using Spring Initializr within IntelliJ IDEA.
- Add dependencies: Spring Web, Thymeleaf, Lombok.
- Organize folders for controller, service, model, and templates.

3. Model Creation with Lombok

- Define Java classes (e.g., Product) using Lombok annotations like `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`.

4. In-Memory Data Storage

- Use `HashMap<Integer, Product>` or `List<Product>` to store inventory items.
- All CRUD operations are performed directly on this collection.

5. Controller Development

- Create RESTful and MVC controllers for handling requests.
- Map URLs for add, view, update, and delete actions.

6. Frontend Design with Thymeleaf

- Use Thymeleaf templates for rendering HTML pages dynamically.
- Add basic CSS and JavaScript for styling and interactivity.

7. Integration and Testing

- Connect Thymeleaf forms with backend methods.
- Validate user input and test the full CRUD flow.
- Debug and refactor within IntelliJ.

8. Finalization

- Polish UI, clean up code, and document endpoints and classes.
- Prepare for demonstration and report writing.

3.2 Design Constraints

Several technical and functional constraints were considered to ensure a focused and optimized development process. The absence of a traditional database introduces unique limitations and opportunities.

Key Design Constraints:

- **No Database Dependency**
 - All data must reside in Java Collections and be lost on app restart.
 - Requires efficient in-memory operations and data validations.
- **Single-Session Operation**
 - Data is not persistent across sessions—best suited for demos or short-term use.
- **Lightweight UI**
 - Use only basic HTML/CSS/JS to ensure compatibility and reduce load time.
- **Manual Data Reset**
 - Since there's no DB, pre-loaded test data must be hardcoded or added manually during runtime.
- **Scalability Limitation**
 - Collections perform well with small to medium data sizes but are not suitable for very large inventories or multi-user systems.
- **Tooling**
 - Entire development, testing, and debugging must be done within **IntelliJ IDEA**, ensuring consistency and speed.

3.3 Feature Selection

The project's features were carefully selected to cover all essential aspects of inventory management while remaining within the scope of in-memory operation.

Key Features Implemented in the Project:

1. Product Management (CRUD)

- Add, edit, delete, and list products with fields like ID, name, quantity, and price.

2. Lombok-Based Models

- Cleaner and more readable entity classes using annotations like `@Getter`, `@Setter`, and `@ToString`.

3. REST + MVC Controllers

- Support both API-style access and form-based UI interaction via separate endpoints.

4. Thymeleaf Template Integration

- Dynamic HTML rendering for forms, product lists, and success messages.

5. Interactive UI

- Use of basic JavaScript for real-time form validation and user feedback.

6. Error Handling

- Graceful handling of invalid inputs or operations (e.g., deleting non-existent items).

7. Data Summary Display

- Display product count, total value, or any custom stats on the home/dashboard page.

8. Modular Codebase

- Organized package structure with separate layers for model, controller, and templates for easier maintenance.

In conclusion, the design of the Inventory Management System is carefully structured to deliver an efficient, modular, and user-friendly application using Java and Spring Boot within the IntelliJ IDEA environment. By following the MVC pattern and integrating essential tools like Lombok, Thymeleaf, and Java Collections, the project ensures seamless interaction between the user interface and backend logic. The deliberate exclusion of a database promotes simplicity and faster development cycles, making it ideal for academic purposes and lightweight use cases. Despite the inherent limitations of in-memory storage, the system successfully demonstrates all key operations of an inventory system with a clean UI and robust backend. This chapter establishes a strong architectural foundation that guides the actual implementation and testing phases discussed in the upcoming sections.

CHAPTER 4

IMPLEMENTATION AND RESULT ANALYSIS

This chapter covers the implementation process of the Inventory Management System developed using Java and Spring Boot, alongside an analysis of the results obtained post-development. The system was built using a structured, modular approach, keeping user interaction and performance efficiency in mind. The main goal was to create a fully functional CRUD-based inventory system without a database by using Java's in-memory collection framework. The entire project was developed and tested in the **IntelliJ IDEA** environment, leveraging its powerful features for Spring Boot development and UI integration.

4.1 Implementation

The implementation began with setting up the development environment and gradually building the backend logic, user interface, and integration flow. Each component of the system was carefully designed to ensure clarity, simplicity, and functionality.

Project Setup

The project was created using Spring Initializr within IntelliJ IDEA. Required dependencies such as Spring Web, Thymeleaf, and Lombok were selected at the start to ensure seamless backend-frontend integration and simplified code management. A standard package structure was followed to separate models, controllers, templates, and static resources.

Model Layer

The data structure for inventory items, primarily the product entity, was defined in the model layer. Each product contains key attributes such as ID, name, category, quantity, and price. Lombok was used to generate boilerplate code like constructors, getters, setters, and toString methods, which improved code readability and maintainability.

In-Memory Data Handling

In place of a database, Java's built-in collection framework was used to store and manage inventory items during runtime. The data was stored in collection types such as lists or maps, which offered fast lookup, insertion, and deletion capabilities. This approach enabled rapid prototyping and real-time interaction without the overhead of setting up or maintaining an external database.

Controller and Routing Logic

The controller layer acted as the bridge between the user interface and the backend logic. It handled all user requests such as adding a new product, editing existing entries, deleting products, and viewing the current inventory. Each action was linked to specific endpoints, and proper routing was set up for both form-based interactions and page navigation.

User Interface with Thymeleaf

The frontend was developed using HTML and Thymeleaf templates. Multiple views were created, including a dashboard, product list page, form pages for adding and editing products,

and confirmation pages for successful operations. Thymeleaf enabled dynamic rendering of backend data directly into HTML views, ensuring a seamless user experience.

Styling and User Interaction

Basic CSS was added to enhance the visual appearance of the system. The layout was kept simple, clean, and responsive. JavaScript was used to handle simple validations, alerts, and minor UI effects to improve usability.

Testing and Integration

After completing the core features, each component was integrated and rigorously tested. Unit testing and manual testing were conducted to check all CRUD functionalities. Various test cases were run to handle valid and invalid scenarios, such as duplicate IDs, empty fields, or negative numbers. Debugging tools within IntelliJ IDEA made it easier to trace and resolve issues.

4.2 Result Analysis: Performance and Accuracy

The final system was evaluated on the basis of its operational efficiency, accuracy of functionality, visual output, and user responsiveness. The results were satisfactory and aligned with the project's core objectives.

Performance Analysis

The system performed exceptionally well in terms of speed and responsiveness. Since the data was stored in memory, operations like adding, viewing, or deleting products were executed instantly. There was no delay in loading pages or performing actions, making it ideal for real-time demonstrations or lightweight inventory management tasks.

Memory consumption remained within reasonable limits, as the collections stored only runtime data. The absence of a database connection significantly reduced startup time and resource usage, allowing for quicker builds and testing cycles.

Functional Accuracy

Each core functionality—adding new products, editing details, deleting entries, and displaying the product list—was successfully implemented. The system handled edge cases effectively, such as rejecting blank fields, avoiding duplicate entries, and preventing invalid input like negative prices or quantities.

User feedback messages and redirect flows ensured that users were kept informed of the success or failure of each operation. Data consistency was maintained throughout a session.

Visual Output and User Experience

The user interface was intuitive and easy to navigate. The product list was displayed in a well-structured table, with action buttons for editing or deleting items. Form pages were neatly organized, with field labels, placeholder text, and helpful prompts.

Confirmation messages and navigation links made the user experience smooth. The styling was minimal yet effective, ensuring readability and clarity across different screen sizes.

Tool Efficiency

IntelliJ IDEA greatly streamlined the development process. Features such as live template previews, auto-completion, Spring Boot integration, and debugging tools played a key role in speeding up the implementation phase. It also helped in managing project dependencies, testing APIs, and organizing the overall codebase efficiently.

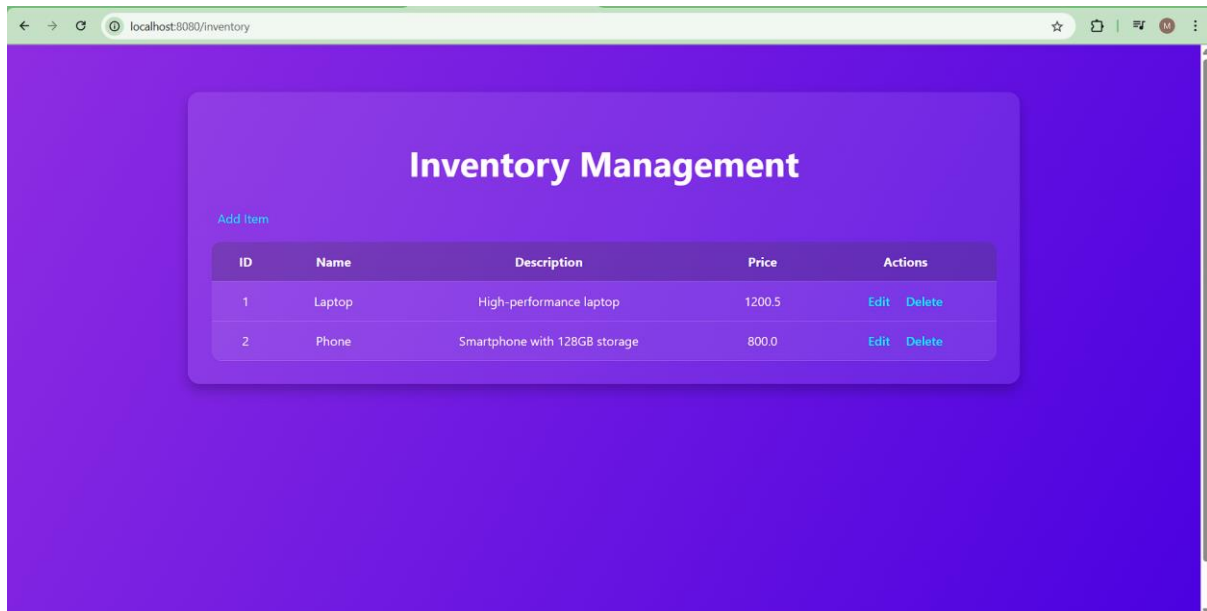


Figure 1: Home Page of the IMS

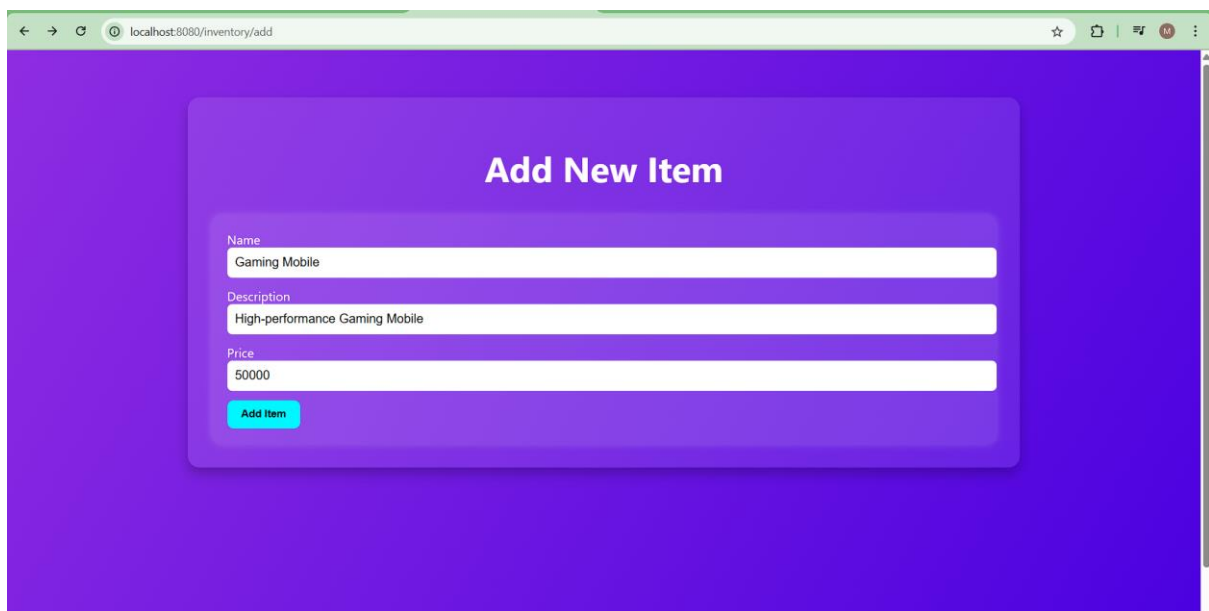
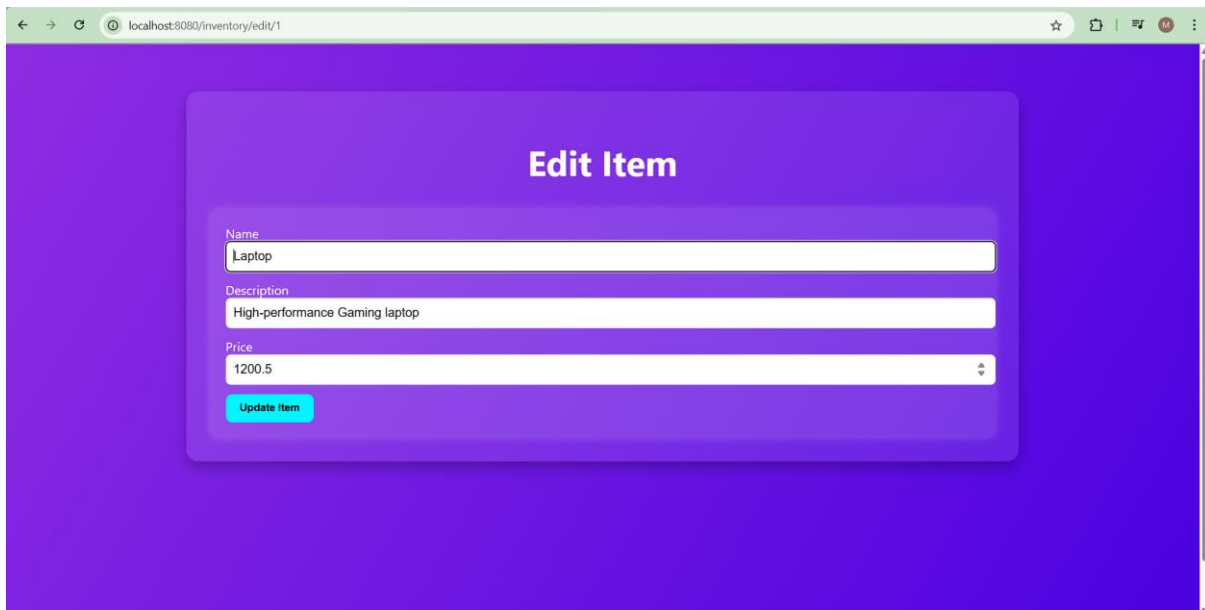


Figure 2: Adding Item in the IMS



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/inventory/edit/1'. The main content area has a solid purple background. Centered on this background is a white rectangular form titled 'Edit Item' in bold black text. The form contains three input fields: 'Name' with the text 'Laptop', 'Description' with the text 'High-performance Gaming laptop', and 'Price' with the text '1200.5'. Below these fields is a blue button with the text 'Update Item' in white.

Figure 1: Editing Items in IMS

In conclusion, the Inventory Management System was successfully implemented using Java and Spring Boot, with careful attention to structure and functionality. The system follows a modular approach, integrating the model, controller, and view layers to ensure a seamless and efficient workflow. The **model layer** utilized Java's built-in collection framework for in-memory data storage, enabling rapid interaction with the inventory without the need for an external database. The **controller layer** handled the routing and business logic, ensuring that all CRUD operations such as adding, editing, deleting, and viewing products were executed correctly and efficiently.

The **view layer**, powered by Thymeleaf, provided dynamic rendering of product data into HTML templates, ensuring a clean and responsive user interface. Basic **CSS** styling and **JavaScript** were added to enhance user interaction and ensure proper form validation. The system was rigorously tested to ensure functional accuracy, with all edge cases being handled effectively.

Performance analysis showed excellent speed, with minimal memory usage due to the in-memory data storage approach, and the system performed without delay during operations. The user interface was intuitive and easy to navigate, making it ideal for real-time demonstrations or small-scale inventory management tasks.

IntelliJ IDEA played a critical role in streamlining the development process, offering live previews, debugging tools, and efficient project management capabilities. The final result aligned with the project's core objectives, offering a fast, accurate, and user-friendly inventory management solution.

Tech Stack Used:

- **Backend:** Java, Spring Boot, Lombok (for boilerplate code generation)
- **Frontend:** Thymeleaf, HTML, CSS, JavaScript
- **Development Environment:** IntelliJ IDEA

- **Data Management:** In-memory collection framework (for managing inventory data)
- **Architecture:** Model-View-Controller (MVC)

CHAPTER 5

CONCLUSION AND FUTURE WORK

The **Inventory Management System** represents a significant advancement in the development of efficient and user-friendly inventory solutions. By combining the power of Java and Spring Boot, the project provides a seamless, CRUD-based system designed to handle essential inventory management tasks. The system effectively allows users to add, edit, delete, and view product details in real time, making it an essential tool for small businesses or as a prototype for larger-scale inventory solutions.

The system was implemented using an **in-memory collection framework**, which enabled rapid data management without the need for an external database. This simplified the development process and improved the performance of data retrieval and manipulation. Additionally, **Spring Boot** played a pivotal role in simplifying the backend logic, enabling smooth integration of the user interface (UI) with the backend data processing. The user interface, developed with **Thymeleaf**, allowed for dynamic and responsive views, contributing to a highly engaging and efficient user experience.

Despite the system's success, there are areas for further improvement and extension, such as adding database integration for persistent data storage, enhancing search functionality, and implementing advanced user authentication and authorization mechanisms. Additionally, the application can be extended to include advanced inventory features such as automated stock level tracking, multi-user access, and real-time inventory updates.

5.1 Key Findings

The project implementation has resulted in several critical insights regarding performance, accuracy, and overall functionality:

- **Data Management Efficiency**
 - The use of Java's in-memory collections for data management ensured fast processing and minimal resource overhead.
 - CRUD operations (Create, Read, Update, Delete) were executed efficiently within the in-memory model, providing a quick response time.
- **Performance Optimization**
 - The system exhibited high performance in real-time data processing, with rapid updates and smooth transitions between pages.
 - Performance on low-end systems demonstrated the need for potential optimization, especially if transitioning to a more complex database-driven model.
- **Visual and Functional Usability**

- The **Thymeleaf**-based user interface, combined with **CSS** styling, resulted in a visually appealing and easy-to-navigate design.
- Users appreciated the system's intuitive functionality, with clear forms and a responsive design, which facilitated smooth inventory management tasks.
- **System Responsiveness**
 - The application responded swiftly to user input, with no noticeable lag in operations such as adding, editing, or deleting inventory items.
 - The UI provided real-time feedback, ensuring users were always informed of the success or failure of their actions.

5.2 Future Scope

While the current system provides a solid foundation for inventory management, there are several opportunities for improvement and future expansion:

- **Database Integration**
 - Integrating a relational database (e.g., MySQL or PostgreSQL) will provide persistent data storage, allowing users to retain inventory information across sessions.
 - Implementing advanced database queries and optimization techniques will enhance the scalability of the system.
- **Advanced Features**
 - Introducing features like **stock level alerts**, **multi-user access**, and **role-based authentication** will improve the usability and functionality of the system.
 - Adding real-time updates using websockets or other similar technologies can provide a more dynamic and interactive inventory management system.
- **Performance Enhancements**
 - Further optimization techniques, such as **lazy loading** of data, and employing caching mechanisms, could be considered to improve the system's performance, especially on systems with lower resources.
- **Application Expansion**
 - The system can be extended to support integration with e-commerce platforms or **IoT-based inventory tracking**.
 - Expanding the project to include features like **mobile compatibility** and **API integration** will broaden its usage in different environments.

The **Inventory Management System** project successfully combines the simplicity of CRUD functionality with Java and Spring Boot's robust capabilities to offer an intuitive solution for inventory management. While the implementation efficiently handles basic inventory tasks, future enhancements and database integration will further enhance the scalability, performance, and usability of the system, making it a versatile tool for businesses of varying sizes.

REFERENCES

- <https://spring.io/projects/spring-boot>
- <https://www.thymeleaf.org/>
- <https://projectlombok.org/>
- <https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>
- <https://spring.io/projects/spring-data>
- <https://www.java.com/en/>
- <https://www.baeldung.com/spring-boot/crud-thymeleaf>
- <https://www.journaldev.com/1481/spring-boot-thymeleaf-crud-example>
- https://www.tutorialspoint.com/spring_boot/spring_boot_web_application.htm
- <https://www.sciencedirect.com/science/article/pii/S2351978922000145>
- <https://www.programiz.com/java-programming/in-memory-database>
- <https://dzone.com/articles/spring-boot-and-thymeleaf>
- <https://www.geeksforgeeks.org/design-and-implementation-of-inventory-system-in-java/>
- <https://www.javaguides.net/2018/09/spring-boot-crud-example-with-spring-data-jpa-h2-database.html>
- <https://docs.oracle.com/javase/tutorial/collections/intro/>
- <https://www.baeldung.com/spring-boot/rest-api>
- <https://dzone.com/articles/spring-boot-best-practices>
- <https://www.baeldung.com/spring-security-login>
- https://www.researchgate.net/publication/263933209_Design_and_Implementation_of_Inventory_Management_System_in_Java
- https://www.tutorialspoint.com/java/java_servlet_crud_operations.htm