



UNIVERSITY OF
LEICESTER

**Department of Informatics
University of Leicester
CO7201 Individual Project**

Final Report

Stock Control System

Krunal Dhavle

kbd6@student.le.ac.uk

229038739

**Project Supervisor: Prof Nicole Yap
Second Marker: Prof Muhammad Kazim**

**Word Count: 11880
08 September 2023**

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Krunal Dhavle

Date: 08 September 2023

Abstract

For ecommerce companies to match stock levels with consumer demand and avoid expensive out-of-stocks, effective inventory management is essential. The primary objective of this project is to create an integrated web-based stock control system that gives firms real-time visibility and smart analytics to improve inventory decisions. Real-time stock tracking, automatic reordering based on user-defined thresholds, and the production of actionable insights from inventory data are all features of the system. Warehouse mapping for storage optimisation, order processing and alerts, customer-facing product availability information, and strong security standards are key aspects. The solution allows data-driven and proactive inventory management by combining inventory tracking, reordering, and analytics in a single platform.

Developmental challenges include integrating React and Spring Boot, creating personalised features like chatbots for customer support, managing rapid inventory changes in high volume e-commerce, and providing safe authentication rules. If the system is a success, it will give companies the ability to match stocks to demand, avoid stockouts and lost sales, and improve the ordering experience for customers. This project shows the crucial role of technology in elevating inventory management to the level of a strategic effort while also promoting greater effectiveness, sustainability, and competitiveness. As quoted by Henry Ford - "It is not the employer who pays the wages. Employers only handle the money. It is the customer who pays the wages." [1] Good inventory management ensures you can meet customer demand.

In conclusion, the offered stock control system seeks to automate, evaluate and enhance inventory management. Businesses might be able grow sales, cut down on waste, and enhance sustainability by using it to provide better visibility and data.

Table of Contents

1. Introduction.....	1
1.1 Aim.....	1
1.2 Challenges.....	1
1.3 Risk.....	1
2. Background Research	2
2.1 Economic Order Quantity (EOQ)	2
2.2 Research on Existing Top Inventory Planner	3
2.3 Project Ideas and learning	4
3. Methodology	5
3.1 Project Planning	5
3.2 Requirement Analysis.....	5
3.3 Design Phase.....	5
3.4 Development Approach.....	5
3.5 Front-End Development Approach.....	6
3.6 Back-End Development Approach	6
4. Requirements	6
4.1 High-level Requirements.....	6
4.2 Detailed Requirements.....	8
4.2.1 Essential Features.....	8
4.2.2 Recommended Features.....	11
5 Technical Specification	12
6 Dependency and Library.....	14
7 Design	15
7.1 MVC Architecture Design.....	15
7.2 Use Case Diagram.....	16
7.3 Interface Diagram.....	17
7.4 Sequence Diagram	18
7.4.1 Admin Sequence Diagram	18
7.4.2 Customer Sequence Diagram.....	19
7.5 Database Design.....	20
8 Project Outcome	21
8.1 Admin Interface.....	21
8.1.1 Home.....	21
8.1.2 Inventory Management.....	23

8.1.3	Purchase Management	26
8.1.4	Sales Management	28
8.2	User Interface	29
8.2.1	Home Page	29
8.2.2	Shop Page.....	29
8.2.3	View Product Page.....	29
8.2.4	Shopping Cart Page.....	30
8.2.5	Order Confirmation and Checkout Pages.....	30
8.2.6	Order Tracking and Refund	30
8.3	Authentication Interface.....	32
8.3.1	Login Page	32
8.3.2	Registration Page.....	33
9	Testing.....	34
9.1	Manual Testing	34
9.2	Integration Testing	39
10	Conclusion.....	39
10.1	Future improvement.....	39
10.1.1	Integrating Chat Bot.....	39
10.1.2	Suggesting the product or Autosuggest Product.....	40
10.2	Evaluation.....	40
11.	References	41

1. Introduction

1.1 Aim

The aim of this project is to develop an inventory management web application that provides efficient stock control and order fulfilment for e-commerce businesses. The system will enable admins to closely monitor stock levels and quickly replenish inventory to avoid out-of-stocks, which can result in lost sales [2]. Key features will include automated inventory tracking, reordering and analytics to optimize decision making, and seamless order placement for customers [3]. By giving admins real-time visibility into inventory status and equipping them with data-driven insights, the system will help maximize product availability and order fulfilment rates [4]. The integrated customer-facing e-commerce platform will also improve satisfaction by enabling seamless product browsing, purchasing, payment processing and order tracking. Overall, the project seeks to build a comprehensive stock control solution that empowers businesses to efficiently align inventories with demand, prevent stockouts, and boost sales.

1.2 Challenges

- Integrating the React and Spring Boot frameworks to create a high-performing and scalable system. Designing and building the application from the ground up, ensuring a personalized approach to meet specific requirements.
- Facilitating seamless communication for notifications, tracking, and data sharing among administrators, the database, the website, and clients.
- Implementing 2D warehouse mapping with location tracking and visualization, and calculating the optimal space required for products.
- Optimizing essential inventory management functions such as stock control, order processing, analytics, billing, and live updates.
- Challenged by the integration of a chatbot to provide customer assistance with order tracking and product information, given limitations in data availability for training and potential complexity in implementation.
- Establishing robust authentication and access controls to enhance API and data security through the utilization of JWT [JSON Web Token] [5].
- Handling frequent inventory changes and returns in high volume ecommerce [6]

1.3 Risk

- **Risk1:** Limited React and Spring Boot proficiency.
Action: Prioritize learning Framework through online courses, tutorials, and practical coding exercises to ensure a strong foundation before project commencement.
- **Risk 2:** Incompatibility with different systems due to required environments lacking, project relies on Spring Boot (backend) and React (frontend).

Action: Mitigate by aiming to deploy the project on a cloud platform for accessible and standardized usage.

- **Risk 3:** Complexity of implementing part location in the warehouse and mapping on a 2D map increases with deeper exploration. Starting with a straightforward approach and gradually progressing to more intricate algorithms.

Action: Begin with a simplified strategy and progressively transition to more advanced algorithms while assessing feasibility and impact at each stage.

2. Background Research

2.1 Economic Order Quantity (EOQ)

Economic Order Quantity (EOQ) model is a mathematical technique used to determine the optimal order size that minimizes total inventory costs. It was developed by Ford W. Harris in 1913 and establishes the relationship between the costs of ordering, carrying inventory, and stockouts. Minimizes total inventory costs by balancing ordering and holding costs. [4]

1. Identify the given values:

A = Total SKU Value Annually

K = Carrying Cost

R = Replenishment Cost

P = cost per unit.

2. Use the values in the EOQ formula as follows:

$$\text{EOQ} = \sqrt{(2 \cdot A \cdot R) / (P \cdot K)} \quad [4]$$

Where:

- sqrt represents the square root function
- A, R, P, and K are the given values
- 2 and P are constants in the formula

3. Explanation of each terms in more detail:

- A = Total Value of SKU Per Year

This is the total annual demand value for the stock keeping unit (SKU)

Calculated as: **Annual demand quantity * Price per unit**

For example, if annual demand is 5000 units, and price per unit is £100, then $A = 5000 \cdot £100 = £500,000$

- K = Carrying Cost (The K Factor)

This is the cost of holding inventory, as a percentage of the inventory value
Includes costs like storage, insurance, taxes, deterioration

Typically ranges from 20%-40% annually
 Converted to decimal - so 25% would be **K = 0.25**

- R = Replenishment Cost (The R Factor)

This is the fixed cost of placing one-time purchase order or production setup. Includes costs like transportation, ordering, clerical work. Does not vary based on the order quantity **R = Maximum One Time Replenishment Cost**

- P = Price Per Unit

This is the cost per unit paid to acquire the inventory. Based on the supplier pricing for the specific SKU. P = 150

4. Calculating EOQ for products

A = Total Value of SKU Per Year = 5000 * 100

K = Carrying Cost (The K Factor) = 0.25

R = Replenishment Cost (The R Factor) = 125

P = Price Per Unit = 150

The optimal order quantity that minimises the overall cost is calculated using these numbers as inputs into the EOQ calculation. This may also serve as a reminder to reorder when the minimum units are achieved.

EOQ = $\sqrt{(2 \cdot A \cdot R) / (P^2 \cdot K)}$

EOQ = $\sqrt{(2 \cdot 5000 \cdot 100 \cdot 125) / (150^2 \cdot 0.25)}$

EOQ = 1112

Unit	A	R	P	K	EOQ
5000	500,000	110	100	0.25	1044
3500	437,500	125	170	0.25	933
2000	500,000	150	250	0.25	773
1000	100,000	100	600	0.25	447
300	300,000	100	1000	0.25	245
300	300,000	300	1000	0.25	424

2.2 Research on Existing Top Inventory Planner

Extensive background research was conducted by studying the features and functionality of leading inventory management solutions including ZOHO Inventory [7] and Inventory Planner [8]. These systems provide core capabilities like inventory tracking, purchase order generation, and reporting.

However, there were several limitations identified that this project aims to address:

- **Lack of an integrated customer-facing e-commerce platform** - this project will allow customers to directly browse inventory and place orders online.
- **Minimal advanced analytics** - this project incorporates interactive data visualizations like graphs and dashboards for enhanced insights.
- **No optimization algorithms** - this project implements automated reordering based on calculated EOQ.
- **No warehouse mapping** - this project enables 2D visualization of warehouse storage locations.
- **No integrated payment tracking** - this project will seamlessly integrate Stripe for processing payments and enable tracking of transactions as well as refunds.
- **No automatic stock inactivation** - this project allows setting minimum stock alerts so products are automatically marked inactive when inventory levels are critically low.

While leveraging some of the underlying inventory management concepts from ZOHO [7] and Inventory Planner [8], this project expands the functionality significantly by developing a robust customer-facing portal, adding advanced analytics and optimization, and increasing customization.

2.3 Project Ideas and learning

- I did some preliminary research before I started this project to find learning tools that will really assist me understand the ideas. I followed a number of YouTube lessons that gave me the fundamental skills I needed to carry out the project successfully. In order to gain practical experience and further strengthen my knowledge of the fundamental concepts, I also produced two small demo projects. By taking these proactive measures, I was able to tackle this job with the necessary expertise and understanding to quickly provide a well-designed solution. Utilising online learning resources and producing test projects made sure I had a solid technical foundation before getting started, supporting a well-structured and efficient development approach.
- Resources Used are as Follow
 - Create Responsive Admin Dashboard. [9]
 - Spring Boot Security for web application using JWT and Role based authentication. [10]
 - Sending mail to user using Gmail API [11]
 - Learning React was very important for this project I didn't had any experience in this framework I learned this from React Bootcamp. [12]

3. Methodology

3.1 Project Planning

The project was planned using the Scrum agile framework which focuses on an iterative approach with continuous feedback and collaboration. Scrum delivers the product in increments called sprints which are fixed duration cycles of development typically 1-2 weeks long [13].

User stories were created at the outset of the project to specify the necessary features and scope the work. These stories were separated into sprints after being prioritised based on their business importance.

I managed project utilising the Scrum methodology and planned 12-week increments known as sprints rather than all at once. At first, I didn't create a lengthy list of tasks. Instead, I had weekly meetings with our supervisor to discuss the project's requirements. This helps to adjust and enhance things as I went along in this manner. It was simpler to make changes as went along because I built the project piece by piece.

3.2 Requirement Analysis

I researched other websites that focused on inventory management and read books on the subject to get the data our project required. I evaluated existing systems to see what worked and what didn't in order to determine how we might make them better. Additionally, I gained knowledge of key guidelines for efficient inventory management. I established what the system needed to be able to achieve, like place orders, and how well it had to be able to perform these things, like guarantee the security of inventory data.

I included all of this data in a requirements document that described all the Functional and Non-functional requirements the programme should be able to perform. To make sure the programme will actually help people overcome their inventory management difficulties, this research was conducted before it was developed.

3.3 Design Phase

The initial plan was to design UI/UX mock-ups and wireframes in Figma. However, due to lack of experience, this proved time-intensive. Instead, simple paper sketches were used to layout key screens and workflows to visualize the user interface. When it came to designing the system's structure, my main focus was on reliability and security. To achieve this, I relied on Spring Boot's strong and dependable architecture and made use of MySQL for its data integrity features. This ensured that the system would be both trustworthy and robust.

3.4 Development Approach

I used an Agile Scrum approach to create the project, allowing us to adjust to evolving demands. To keep track of changes to the code over time, we used SVN as version control. The work was divided into 1-week sprints with project deliverable increments. Jira boards and regular weekly meetings lasting 30

minutes with the supervisor helped us track our progress. Tasks, priorities, and any other difficulties were discussed in these meetings, which were very helpful. This method allowed for close communication and made sure that our work was in line with the sprint objectives. It also provided a flexible yet controlled way to develop project features steadily.

3.5 Front-End Development Approach

React was chosen as the front-end framework for the website in project. It's like selecting a toolkit with reusable and adjustable components to make our work more productive. In order to ensure that our site looks professional and cohesive, we used Material UI, which offers a collection of pre-designed components. We used ESLint, a tool that acts as a helpful assistant, finding errors and making sure our code adheres to best practises, to maintain our code clear and error-free. Together, they enabled us to create a website that is of excellent quality and visually appealing.

3.6 Back-End Development Approach

Spring Boot framework is used to quickly build the backend and fully customizable as per needs. It functions as a useful tool for creating the server-side component from scratch. Because MySQL is flexible, has a large data capacity, and is free for most needs, we decided to utilise it to store data. It also safeguards our information. I used a framework that makes the process simpler to communicate with the database. We saved time by connecting our product to other platforms for things like payments and analytics rather than implementing everything from scratch. To ensure that our code functions properly even before it is complete, we also tested it by doing manual testing and Junit Testing. We created a solid and dependable system in this manner.

4. Requirements

4.1 High-level Requirements

4.1.1 Admin

The admin's interaction with the software initiates with an authentication step. Once successfully logged in, the admin is directed to the Admin Dashboard, a centralized hub presenting an overview of inventory statistics. This dashboard offers access to various interfaces, enabling the admin to efficiently manage Purchase, Vendor Inventory, and Sales operations.

Admin needs to add vendors details, then admin can purchase the product from selected vendor, system will send customize email to vendor along with invoice attachment. After sending email bills will generated and marked as unpaid. Admin can check if the orders are delivered by vendors. If yes then admin need to mark this order as received and then only admin can add this ordered units into Inventory. If the product exist then only quantity will be added else admin can add new product with details. Admin can make the product Active/Inactive, only active

products are listed on website. When user placed the order, admin can see the order in Sales, and then proceed to deliver the orders. If user requested for refunds then admin can decline or accept to process refund. Admin need to add Category before ordering any products.

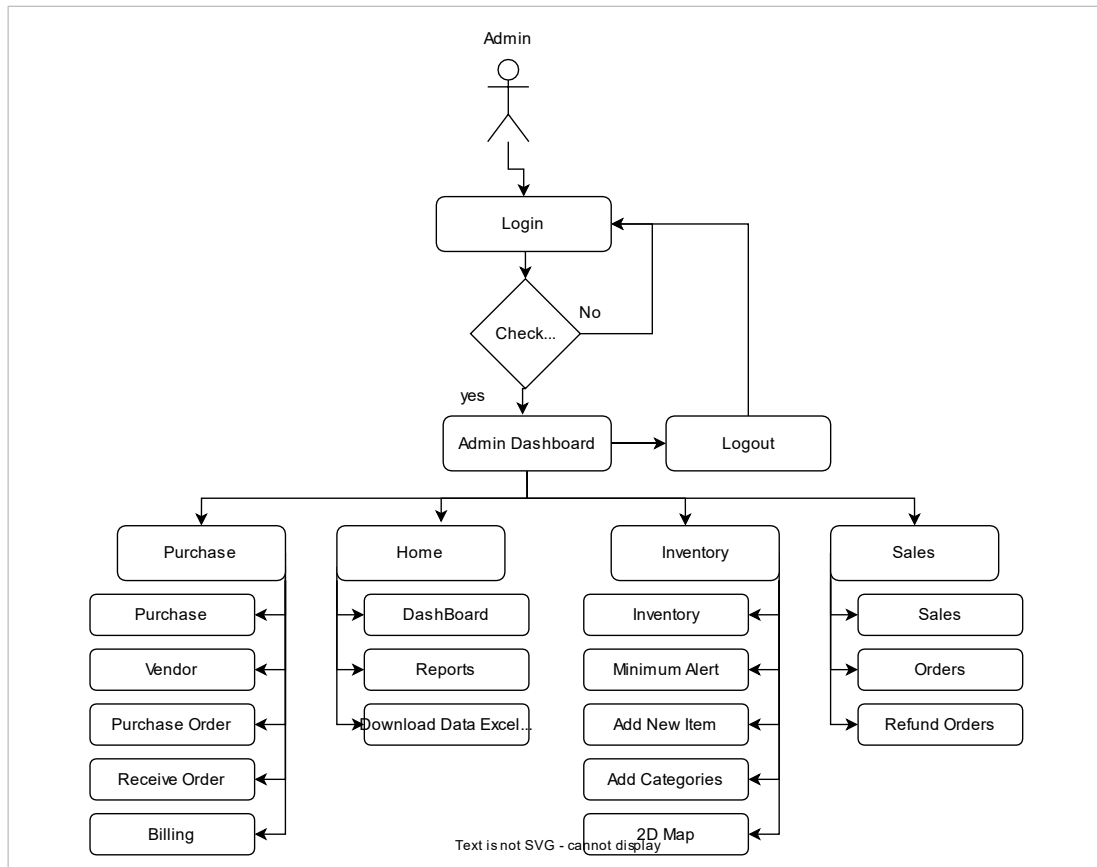


Figure 1 Admin Interface, with Website Pages

4.1.2 Customer

Customer engagement with the stock control management system begins by necessitating registration. Once successfully registered and logged in, customers gain access to the system's functionalities. This includes perusing computer products listed on the website, searching or sorting the products by price or category or new products, adding items to their shopping cart, placing orders, adding delivery address and finalizing transactions using the Stripe payment gateway. Customers are further empowered to monitor the progress of their placed orders. If Customers are not satisfied with bought product, they can request for the refunds for the specific product.

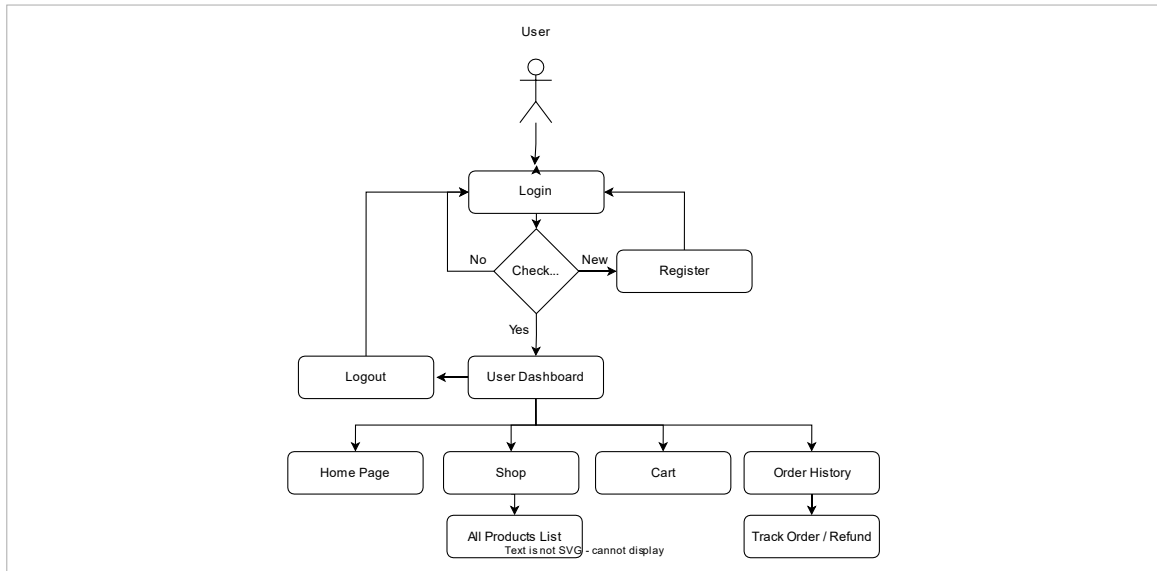


Figure 2 Client Interface with Website Pages

4.2 Detailed Requirements

4.2.1 Essential Features

4.2.1.1 Admin

- Login-Logout Functionality:** For secure admin login authentication and authorisation, the system makes use of JSON Web Tokens (JWT) [5]. Administrators use encrypted passwords to sign in. A distinct JWT token is created by the system and allocated to the admin user session upon successful login. The JWT token serves as the equivalent of an access pass, identifying the user as an authorised admin [5]. The header of each request the admin makes to the server will contain this token. In order to process the admin requests, the server first verifies the token. The server will reject requests that are received without a valid token, preventing access. Only admins are able to access the dashboard and carry out admin tasks thanks to the JWT token. By logging out, the admin session is securely closed and the token is destroyed.
- Admin Dashboard with Data Visualization (Home Page):** The Dynamic Admin Dashboard provides at-a-glance visibility into overall inventory health on the Dashboard Page through data visualizations like revenue charts, order gauges, and top seller, while the Report page enables in-depth analysis with interactive reports using advanced graphs to visualize sales, revenue, and other trends. Finally, the Download Data page allows data export of products, vendors, orders, and more as downloadable in CSV and Excel files format for offline usage and integration into accounting or ERP systems. With an intuitive status overview, deep analytics, and data accessibility, the Admin Dashboard offers comprehensive real-time intelligence to inform data-driven decisions and optimize inventory operations. (Refer Home in Figure 1)
- Vendor Management:** The admin can fully manage vendors from start to finish - adding new vendor information, updating or removing vendor details, placing orders to restock inventory, creating and paying bills when shipments come, reordering from trusted vendors to get good prices and delivery, and keeping

detailed records of every order. This gives the admin complete control over the entire vendor process from bringing on new suppliers to reordering. The system makes vendor management easy for the admin so they can get inventory efficiently. (Refer Purchase in Figure 1)

- **Purchase Management:** The purchase management process enables the admin to fully track and control the ordering workflow - first adding new vendors into the system, then creating purchase orders by selecting a vendor and entering product details like name, category, buy price and quantity which sends an email order to the supplier, once the shipment arrives, the admin marks the items as received in the received orders page which generates a bill for payment; this allows monitoring the entire workflow from order creation to shipment receipt and billing in one seamless system. (Refer Purchase in Figure 1)
- **Inventory Management:** The inventory management functions enable the admin to perform full CRUD (create, read, update, delete) operations on products - adding new items as shipments are received, modifying or removing existing products, creating and managing product categories and attributes, setting minimum quantity alerts for reorders, and viewing warehouse storage locations on an interactive 2D map. This provides a streamlined system for admins to closely monitor and control the entire inventory lifecycle. (Refer Inventory in Figure 1)
- **Sales Management:** The Admin has complete control over all sales orders. The administrator can review and process sales orders, ensuring timely delivery to customers. Additionally, the system allows the administrator to easily track refund requests made by users, providing a comprehensive view of customer interactions and order management. (Refer Sales in Figure 1)
- **Data Integrity Measures:** System implements strict data validation procedures to guarantee data integrity. All forms are put through strict checks for accuracy before any data is saved into the database. The probability of incorrect or incomplete data entries is considerably decreased by this validation process. Additionally, administrators are alerted to any incorrect or missing data, enabling them to take immediate corrective action. Additionally, strong exception handling techniques are put in place to guarantee that the server keeps running smoothly even in the event of mistakes, preventing disruptions to the functionality of the system.

4.2.1.2 Customer

- **Registration and Login Pages:** The platform includes dedicated pages for user registration and login in order to facilitate seamless access. The registration page allows new users to securely sign up for accounts by providing necessary information. The login page enables existing users to securely access their accounts through authentication powered by JSON Web Tokens, ensuring only authorized users can gain entry to personalized content and

features after logging in. Together, these pages provide simple yet secure access to the platform. (Refer Figure 2)

- **Homepage Access:** After a successful login, customers are redirected to the homepage landing page where they can discover new arrivals and featured products. The homepage allows customers to navigate to other sections like the Shop page, which displays the full product catalogue that can be sorted by price, category, etc. Customers can view more details about each product and then add items to their cart or proceed to checkout. The homepage provides an easy way for custom. (Refer Home page and shop Figure 2)
- **Cart Management:** Customers can easily add, remove, or change items in their cart. They can checkout and pay directly from the cart using Stripe.
- **Order Management:** Once a customer places an order and payment is processed, their purchase appears in the Order list. This allows customers to track the delivery status of their purchases. After an order is delivered, customers also have the option to request a refund through the order management system. With robust order tracking and refund capabilities, customers can monitor their orders from purchase to delivery and handle any post-purchase needs.
- **Stripe Payment Gateway:** The platform leverages Stripe as the payment gateway, enabling customers to pay by card in a secure manner compliant with regulations. Stripe ensures security as it does not allow user credentials to be stored server-side per new rules. Instead, Stripe provides its own implemented gateway page to securely save credentials on their platform and returns a **CLIENT_INTENT** [14] that tracks payments on the Stripe dashboard. This allows for seamless and secure card payments without storing sensitive user data on the server. (Refer Figure 3)

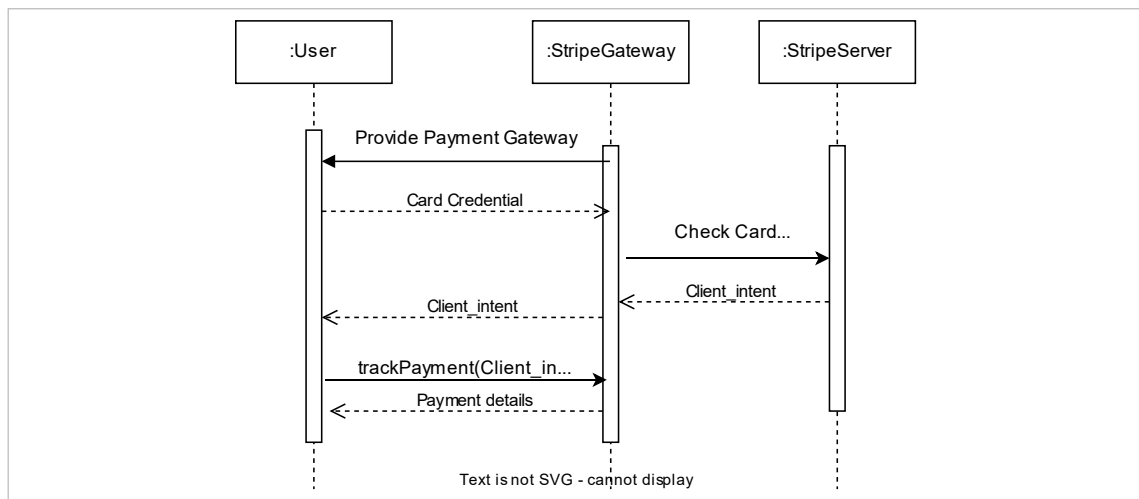


Figure 3 Stripe Sequence Diagram

4.2.2 Recommended Features

4.2.2.1 Admin

- **Auto Alert for Minimum Stock Levels:** An automated alert system that keeps track of inventory levels and tells administrators when any item's stock goes below the specified minimum number can assist avoid stockouts by turning the item inactive so that customers can't place orders. The admin can reorder and replenish the item due to this alert before it runs out totally. Setting minimum stock level requirements for each product and incorporating an auto alert feature into the inventory system enable admin to receive proactive warnings when stock is running low so they may take action to replenish it. This assists in making sure there is enough inventory on hand to fulfil client demand and avoids sales and operations being disrupted owing to stock shortages [15].
- **Auto Stock Reorder:** The auto stock reorder allows the admin to set up automatic reordering of inventory. When enabled, this feature lets the admin easily reorder the same products in the same quantities from previous orders with the vendor without having to search for order details, saving admin time. The system does not automatically send the reorder without the admin's approval. Instead, when stock falls below the minimum threshold, the admin receives an alert and can review and then manually click a button to quickly place the reorder and send the confirmation email to the vendor. This gives the admin control to decide if the reorder should be placed before sending it. The admin can also proactively click the reorder button at any time to replenish stock. In both cases, the admin must manually send the email to complete the reorder. This empowers the admin to automate reordering while still overseeing the process and enables fast reordering without having to look up previous order details, saving valuable admin time. Additionally, the admin has the option to set the auto reorder feature to inactive for any product, disabling the automated reordering functions for items where it is not needed [15].
- **2D Warehouse Map:** The 2D warehouse map provides a visual representation of the inventory layout that enhances stock management and tracking. This dynamic map adjusts product placement based on quantity, calculating the volume / Size / (Length * Breadth * Height) of items to determine space requirements and the number of shelves needed. The products are organized on the 2D map in a grid layout that maximizes warehouse storage space. The system calculates the optimal length and width dimensions for the rows and columns on the grid to fit the total volume of each product's inventory. The map interface leverages MUI's masonry layout to efficiently organize products into a 10-column format within the warehouse outline. As quantities fluctuate, the map automatically updates, reallocating space to reflect changes. This allows precise tracking of item locations and streamlines assembling orders, picking, and stocking processes by providing an interactive visualization of where inventory is stored within the warehouse. The 2D map boosts organization and efficiency in warehouse operations. (Refer Figure 4)

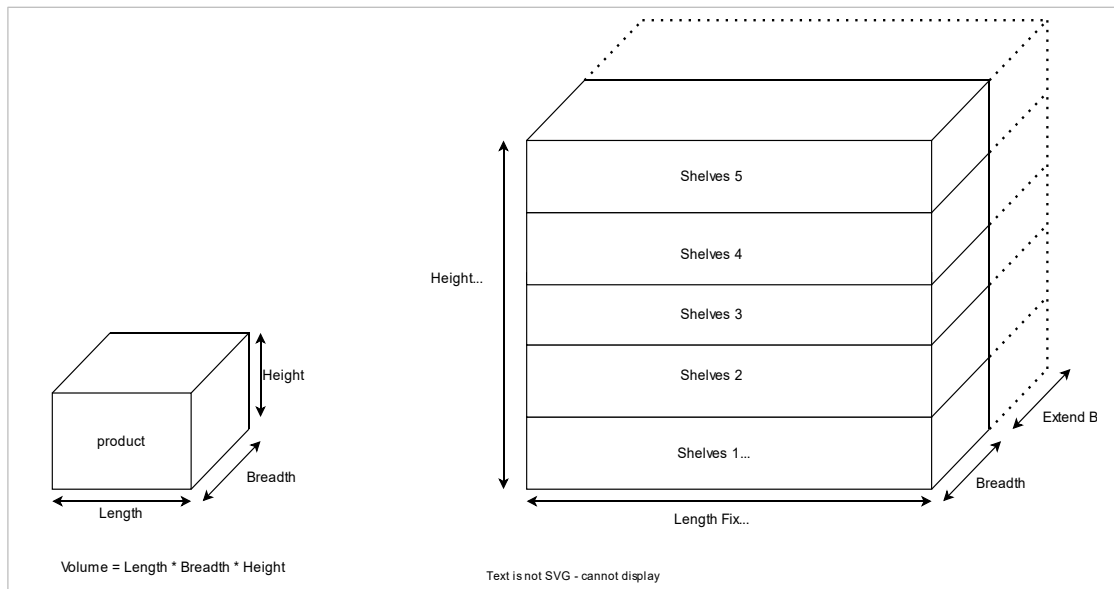


Figure 4 2d map Visualization diagram

4.2.2.2 Customer

- Return and Refund Functionality:** Customers can initiate returns and ask for refunds using the return and refund capability if they're unhappy with their purchase. Customers can log into their accounts after an order is delivered and make a return request that is visible to the admin. After reviewing each return request, the admin can determine whether to allow or deny the reimbursement in accordance with the return policy. When a return is accepted, the administrator can process a refund using the customer's original payment method. Customers may easily initiate returns and monitor their progress using this method, and the admin has control and insight over return requests so they can decide on refunds in a well-informed manner. The system offers clients the ability to request a refund if they are dissatisfied with an order.

5 Technical Specification

Type	Name	Summary
Programming Language	Html5	This project uses HTML5, the latest standard markup language for structuring web content, to define the frontend user interface. [16]
	Css3	This project employs CSS3, the latest stylesheet language, to visually style and layout the user interface. [17]
	JavaScript	JavaScript, a scripting language for web interactivity, is utilized for front-end logic like data validation, UI effects, dynamic content, and API integration.

	Java	Java, an object-oriented language, handles the back-end application logic, databases, data processing, and content delivery. [18]
Framework	React	React is an open-source JavaScript library created by Facebook for building fast and interactive user interfaces for web applications. [19] The key feature of React is its component-based architecture that allows developers to build UI components in isolation as reusable, composable pieces.
	Spring Boot	For creating scalable back-end applications and microservices, Spring Boot is an open-source Java-based platform built on top of the well-known Spring framework. By offering defaults and auto-configuration, Spring Boot streamlines and quickens development so that programmes can be created with the least amount of code possible. [20]
	Material UI	The project uses Material UI, a React framework with Material Design components, to build the responsive front-end UI. [21]
Operating System	Windows 7 or newer	To access the web application, any modern web browser. As long as you have a relatively modern OS and web browser, the app should perform well.
	MacOS 10.11 or newer	
Database	MySQL 8	MySQL is the database management system used by the application backend to store and manage all data.
	MySQL Workbench	MySQL Workbench, the official IDE for MySQL, is recommended for database architecture design, querying, and management. It provides a visual interface for the MySQL database used in the application. [22]
IDE	Vs code	VS Code, a lightweight open-source editor, is recommended for front-end development. It supports JavaScript, HTML, CSS, and has extensions for React development
	IntelliJ IDEA	IntelliJ IDEA is the recommended IDE for efficient Java and Spring Boot back-end development due to its specialized tools and features.

Version Control	Git and SVN	A hybrid Git/GitHub and SVN version control strategy provides robust code and artifact management.
Other Technologies	Postman	Postman, used for API testing and documentation, improves development and collaboration around the project's REST APIs.

6 Dependency and Library

Dependency Name	Usages
Openpdf (1.3.30)	The OpenPDF library enables dynamic PDF generation in the Spring Boot backend.
Itextpdf (5.5.13.3)	The iText PDF library allows for PDF generation and manipulation in the Spring Boot backend.
jakarta.mail-api (2.1.2) and jakarta.mail (1.0.0)	The jakarta.mail API enables sending and receiving emails directly from the Spring Boot backend code for use cases like order confirmations, alerts, and notifications.
poi-ooxml (5.2.3)	The Apache POI library provides Excel (XLSX) read/write capabilities in the Spring Boot backend for use cases like data import/export.
Opencsv (5.7.1)	The OpenCSV library enables CSV parsing in Spring Boot, supporting use cases like data import and export.
stripe-java (21.12.0)	The Stripe Java SDK enables payment integration in Spring Boot via the Stripe API.
Apexcharts react-apexcharts	ApexCharts provides interactive charting capabilities in the frontend with various visualization options.
axios	Axios enables simplified HTTP client-server communication in the frontend, providing easy asynchronous request/response handling for tasks like API data retrieval, user authentication, and real-time updates.
dayjs	dayjs is a compact JavaScript package used in web applications to manipulate dates and timings.
formik	The Formik library streamlines form management in React with tools for validation, state handling, and submission.
jodit-react	jodit-react is a React text editor for creating personalized email content.

react-circular-progress-bar	React-circular-progress-bar is a JavaScript package for creating circular progress bars in React apps to visually show the status of operations or tasks.
react-image-magnify	react-image-magnify allows image zoom in React apps, enhancing user experience for detailed photos.
react-router-dom	react-router-dom is vital for handling client-side routing in single-page React apps, allowing smooth, dynamic navigation without page reloads.
react-toastify	react-toastify is vital for displaying notifications in React apps, providing essential user feedback.
recharts	recharts is essential for creating dynamic charts and graphs in React apps, offering various chart types like bar, line, and pie charts to effectively visualize data.
yup	Yup is essential for React apps, validating form data to ensure accuracy and data integrity, improving the user experience.

7 Design

7.1 MVC Architecture Design

In order to divide responsibilities, Spring Boot web projects often use the Model-View-Controller pattern. Models often contain data and business logic, Controllers manage requests from API calls and answers them, and Views produce the user interface. The Spring Boot back-end in this project does not use Views because this project uses React for the front-end UI [23].

Instead, the Spring Boot application has Controllers to expose APIs, Services for business logic, Repositories to communicate with the database, and Model entities to represent data structures. All display and user interface tasks are handled by the React front-end, which communicates with the back-end only through API requests and responses, devoid of Views. With this modified version of MVC, the front end and back end are separated, with React creating the Views and Spring Boot concentrating on the Model and Controller components. The back-end and front-end components can still be developed and tested independently because to the separation of concerns throughout the stack.

To break down concerns and enable more maintainable code, I adopted the Model-View-Controller model in the project. Prior to MVC, our code was tightly connected logic or tightly coupled, UI, and data access were either merged in an untidy manner or were all contained in a single file, which made it challenging to isolate and reuse code. As the software expanded, even minor adjustments introduced issues, making it challenging to comprehend how different parts worked together.

We produced more modular and loosely linked components by dividing the data layer, business logic, and user interface into Models, Controllers, and Views. Now

that I don't have to worry about front-end code, I can concentrate on the backend Models. We created reusable web-supporting services. Features can be added without disrupting current code because changes to one layer don't immediately affect changes to others. Our programme became scalable thanks to MVC, which also made finding and resolving problems easier. Rearchitecting was necessary, yet there are maintainability advantages [24].

7.2 Use Case Diagram

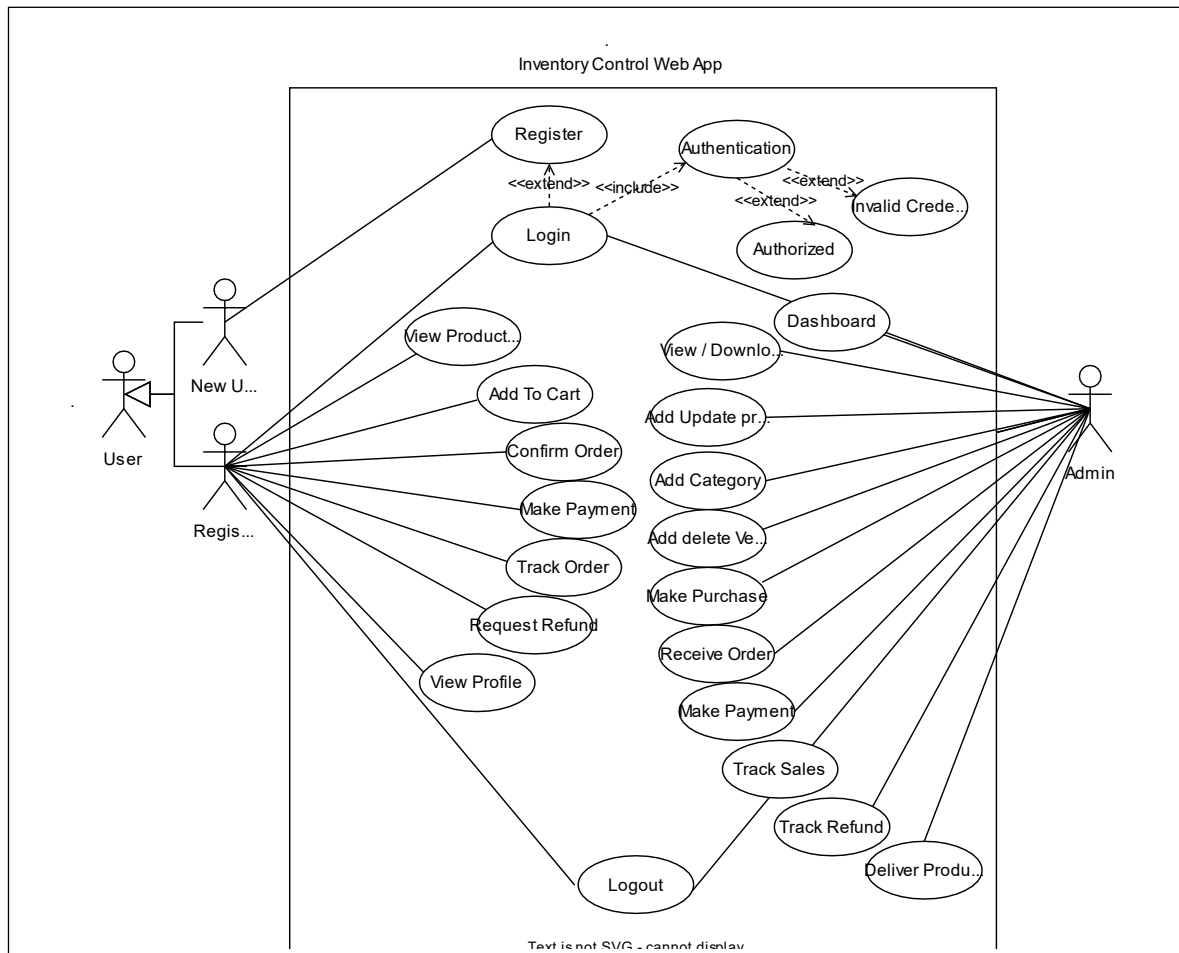


Figure 5 Use Case Diagram of Inventory Control

A use case diagram is a visual representation of how users interact with a system or application.

User Types:

- **New User:** This represents a person who is new to your application and needs to register before accessing its features.
- **Registered User:** This user type has already gone through the registration process and can directly log in to the application.
- **Admin:** This user type is responsible for managing and overseeing the application's operations.

User Actions:

- Users can register, log in, view products, add items to their cart, confirm orders, make payments, track order status, request refunds, view profiles, and log out in our application.
- For the admin, the actions in our application include logging in to access the dashboard, viewing and downloading data, adding or updating products and categories, making purchases, adding or deleting vendors, receiving orders, making payments, tracking sales and refunds, delivering products, and logging out

7.3 Interface Diagram

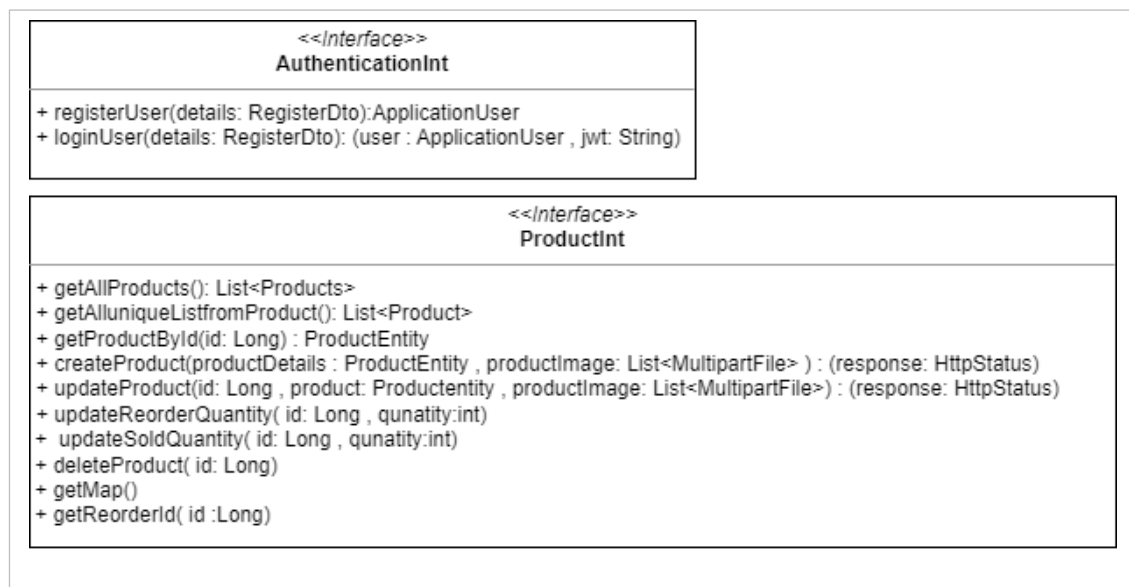


Figure 6 Interface Diagram

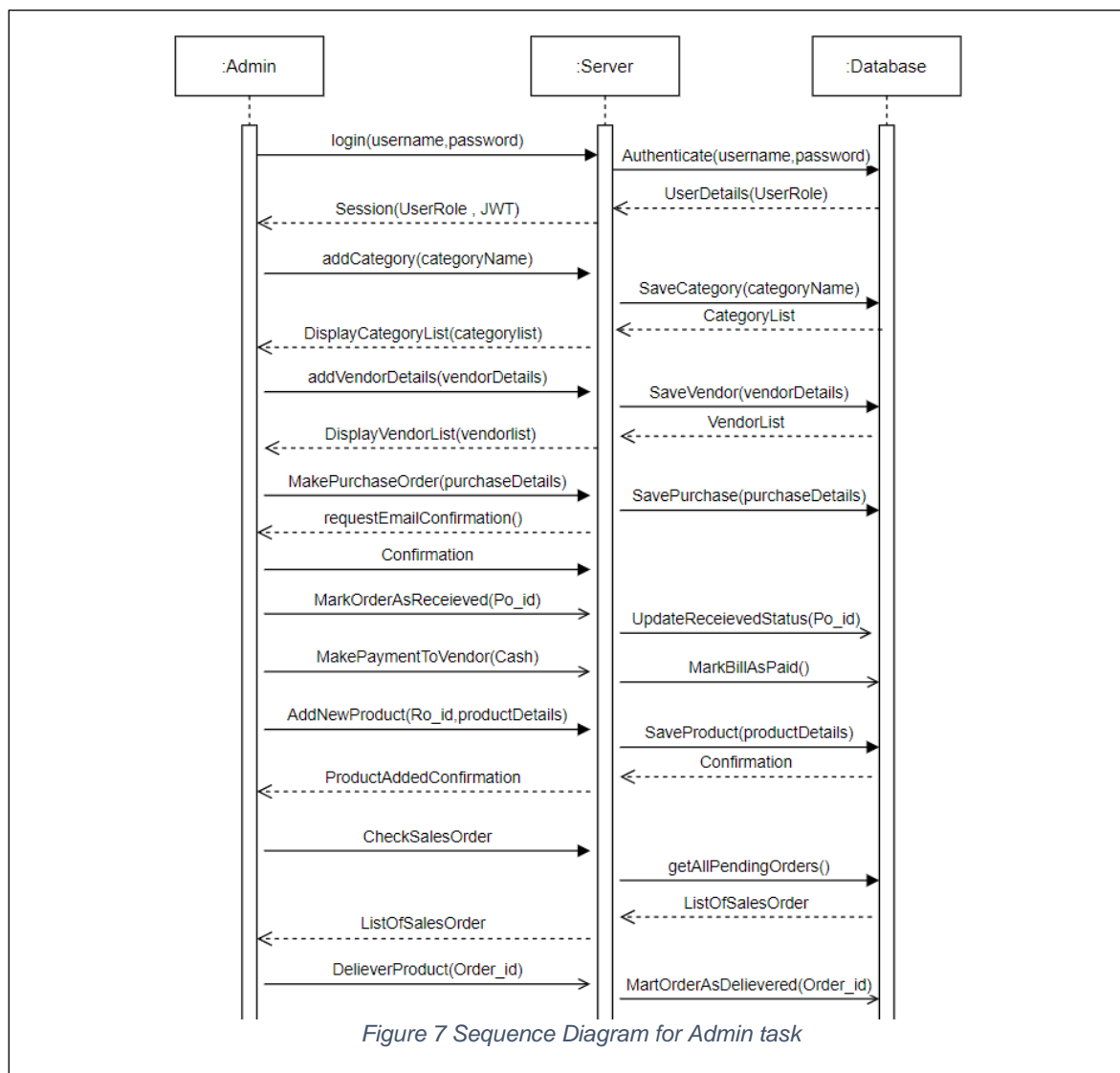
The Product Interface diagram captures the core API operations related to product management. It includes methods to retrieve products, get details by ID, create, update, and delete products. Additional operations like updating reorder quantity and sold quantity enable keeping track of inventory flow. This interface encapsulates the product data backend capabilities required by the system.

The Authentication Interface models the API endpoints needed for user identity management. It has register and login methods to handle new user creation and authentication. Return types indicate the user details and JWT token returned on successful login. This interface provides the fundamental user management APIs for account creation, authentication, and session management.

These two interfaces serve as an aspect of the backend API functionality that the front-end needs. They allow reasoning about the capabilities of the individual parts required to perform user account management and the basic product catalogue functionality. For the whole backend API scope to be captured, further interfaces would model the remaining sections, such as orders, the shopping cart, etc. The essential backend interfaces are logically summarised in these diagrams.

7.4 Sequence Diagram

7.4.1 Admin Sequence Diagram



The interaction between the admin, server, and database parts of an inventory control web application is depicted in the diagram.

The administrator must first log in by entering a username and password. To validate the credentials, the Server calls the Database's Authenticate function. The UserRole that the Server uses to construct a Session with a JWT token is returned by the Database.

The Admin then calls the Server's addCategory function while giving the categoryName to add a new category. To save the new category, the server executes SaveCategory on the database. The DisplayCategoryList function displays a CategoryList to the Admin from the Database.

The steps for adding a new vendor, creating a buy order, adding products, etc. are similar. Every time the Admin takes a step, the Server receives a call, and the Server then communicates with the Database to complete the task.

7.4.2 Customer Sequence Diagram

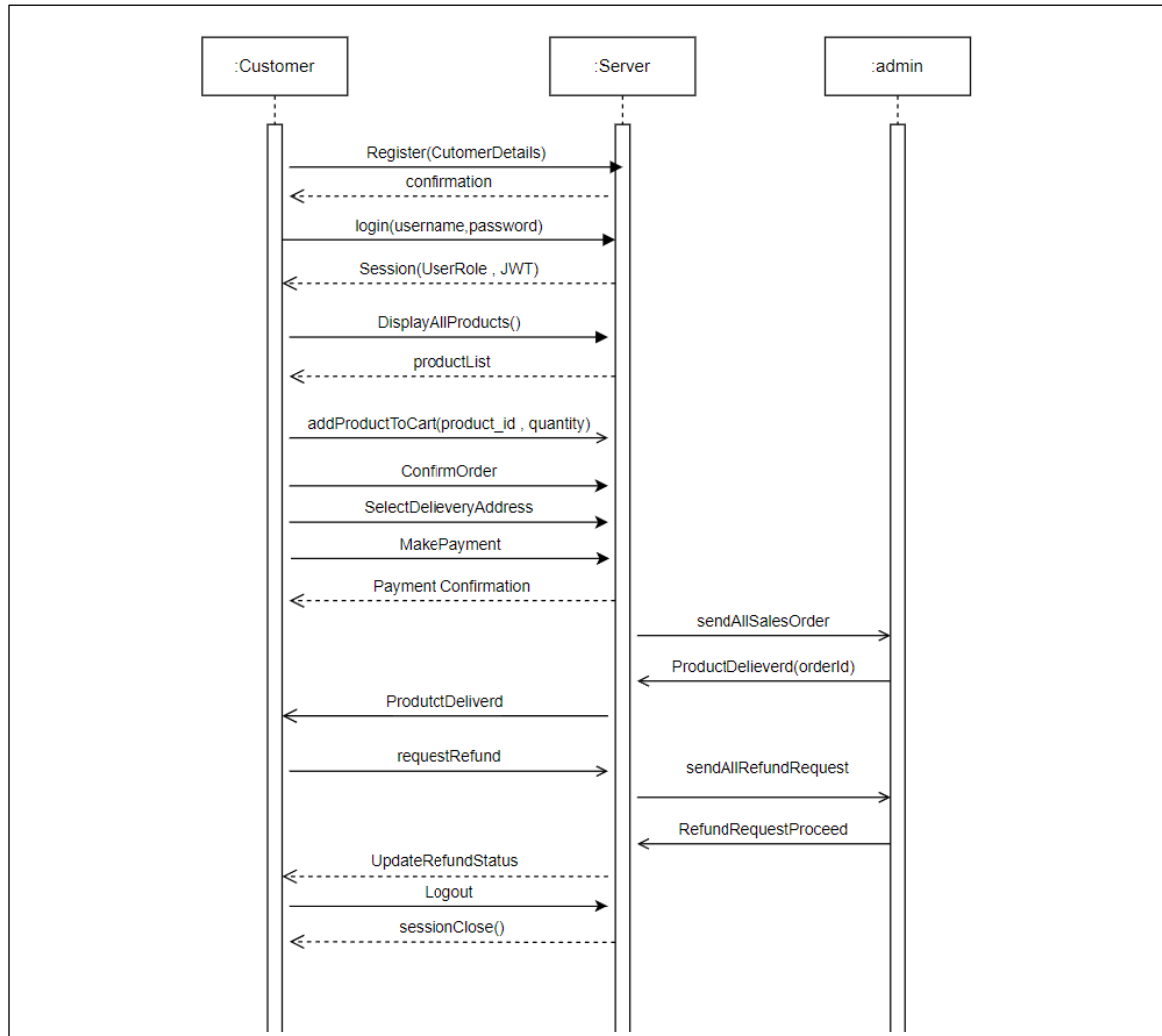


Figure 8 Sequence Diagram for Customer task

The interaction between the Customer, Server, and Admin parts of an e-commerce platform is shown in the diagram.

The consumer logs in with their credentials to begin. Following the server's verification, a session is created with a JWT token for the UserRole. The customer can see every product that is offered on the server. Products can be added by the Customer to their Cart.

When everything is prepared, the customer confirms the order, starting the delivery procedure. The fulfilment of orders is handled by the Admin. The customer has the option to ask for a refund after the product is delivered if necessary. The refund request is then either approved or rejected by the Admin.

7.5 Database Design

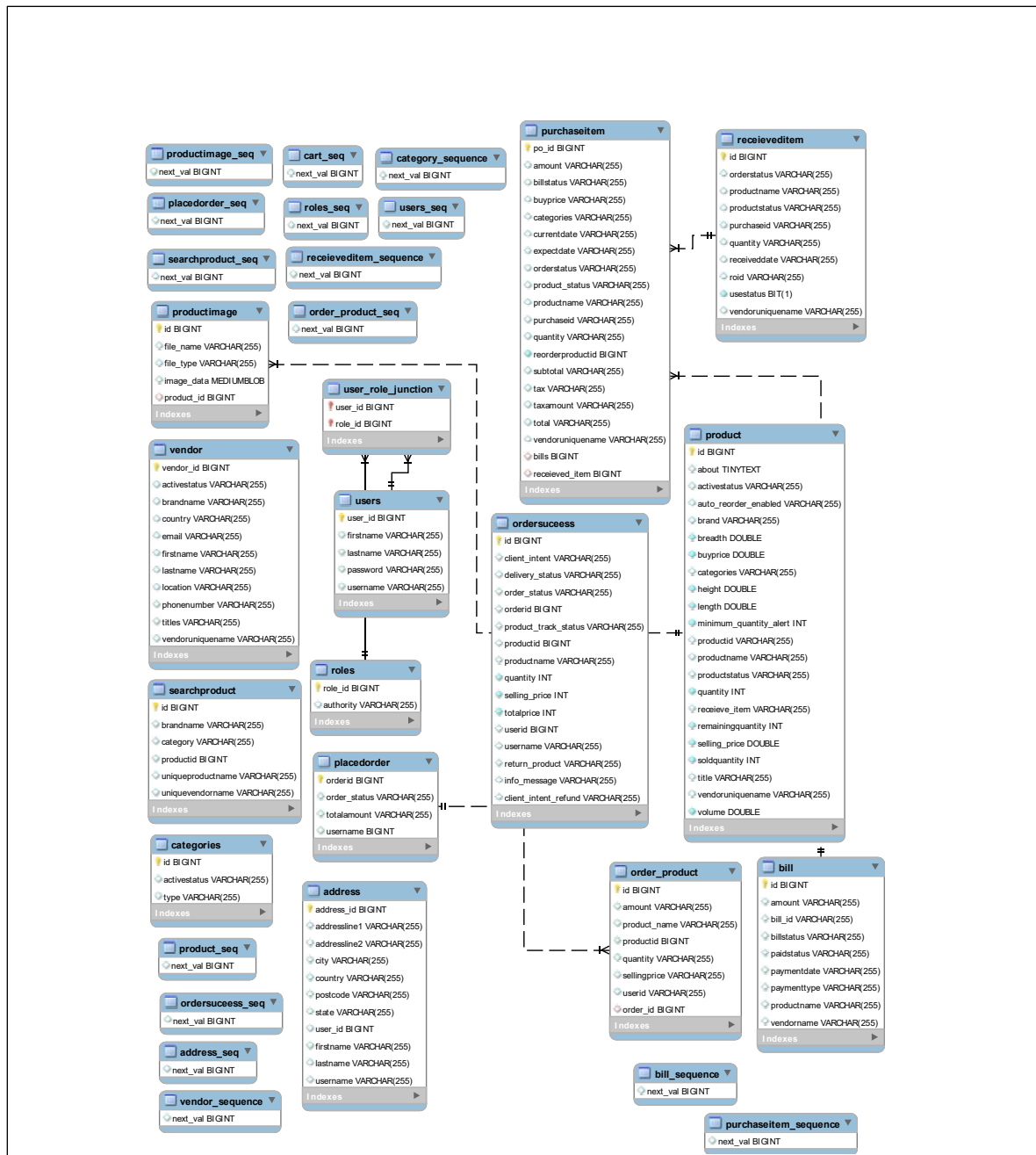


Figure 9 Database ER Diagram

The diagram shows the table structure and relationships for an inventory control management database.

The core tables are:

Users - Stores user account information such as name, username, password etc. It has a one-to-many relationship with Ordersuccess table.

Products - Stores all information about products such as name, description, price, quantity etc. It has a many-to-one relationship with Category table and a one-to-many relationship with Order_product table.

Categories - Stores information about product categories. It has a one-to-many relationship with Products table.

Vendors - Stores vendor information. It has a one-to-many relationship with Products table.

Ordersuccess - Stores order information. It has foreign keys to reference Users, Products, and Vendor tables.

Order_product - Junction table to capture individual products in an order. References Ordersuccess and Products tables.

Addresses - Stores user's addresses. References Users table.

Placedorders - Stores summarized order details. References Users and Ordersuccess.

Purchaseitem - Stores purchase order details. References Vendors and Products.

Receieveditem - Tracks received items against purchase orders. References Purchaseitem.

Bills - Stores billing details for orders. References Ordersuccess table. The diagram also shows other supplementary tables like Search product, Roles, Cart to support additional functionality.

8 Project Outcome

8.1 Admin Interface

8.1.1 Home

8.1.1.1 Dashboard

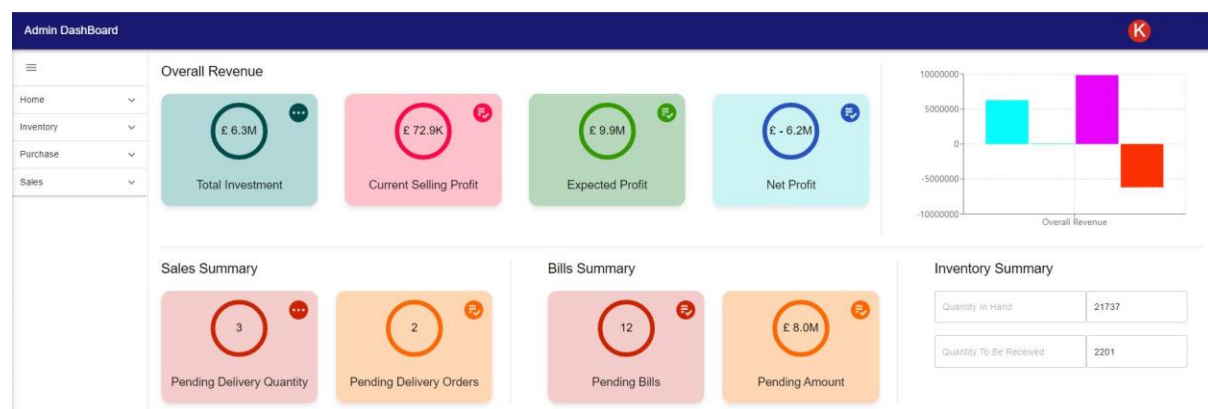


Figure 10 Admin Dashboard 1

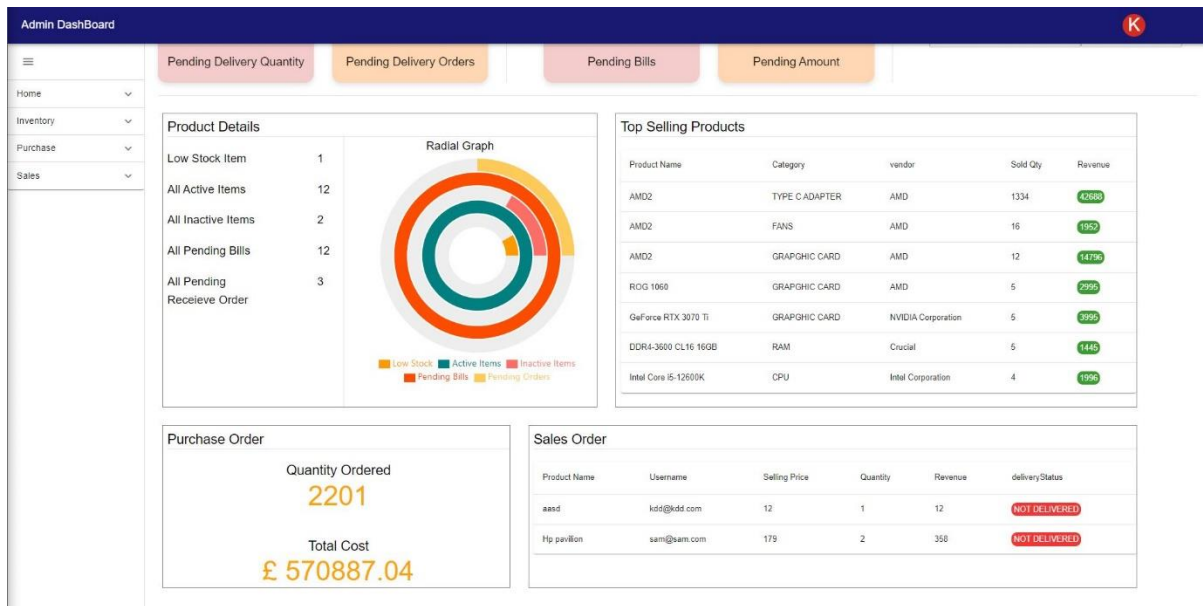


Figure 11 Admin Dashboard 2

The dashboard provides quick insights into the state of the business's finances using visual cards that display revenue, outstanding invoices, orders, and top sellers' goods. In order to support data-driven decisions, it delivers sales order and purchase order summaries. The bill summary keeps track of payables by showing the entire amount due. Additionally, it provides inventory overview, including active, inactive, and low stock products.

These customised summaries and overviews promote effective money management and well-informed choices.

8.1.1.2 Reports



Figure 12 Admin Report

These 9 fully customizable visualisations provide quick insights into inventory data like brand and vendor distribution, category percentages, temporal trends in stock

levels and sales, and profitability. Users may efficiently analyse and comprehend crucial inventory and financial data thanks to the variety of graph kinds that portray the same information from many visual angles.

8.1.1.3 Download Data in Excel and CSV



Figure 13 Admin Download Data

Administrators can extract vital information from this page, including items, categories, orders, billing, purchases, vendors, and users. The necessary data can be chosen and exported to CSV or Excel for offline analysis and record-keeping. Data retrieval is made easier and data management is strengthened as a result.

8.1.2 Inventory Management

8.1.2.1 Inventory

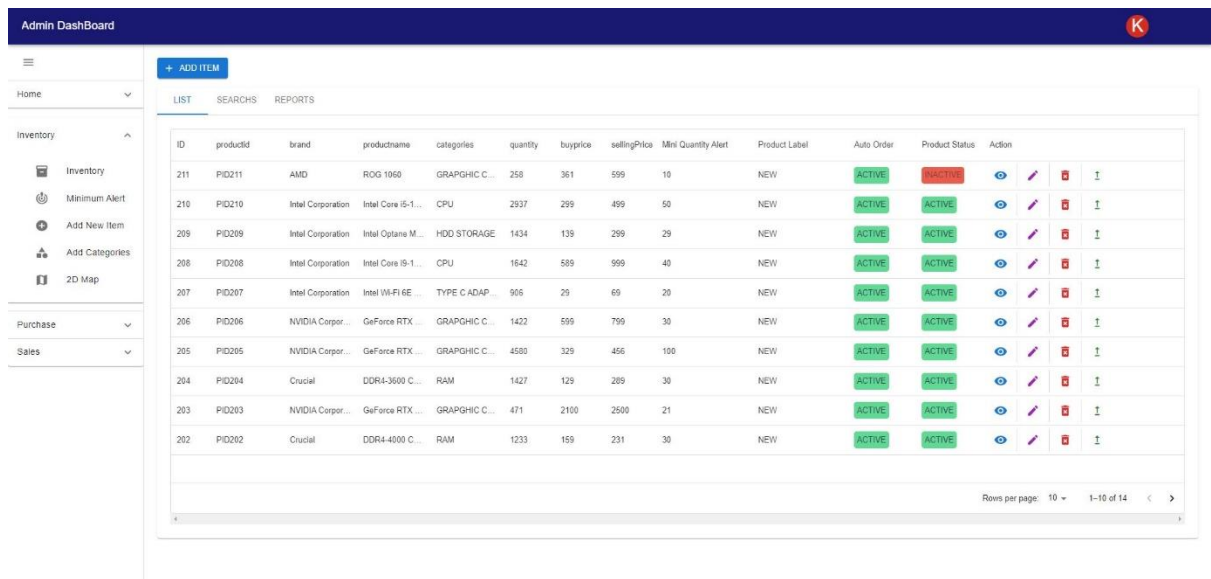


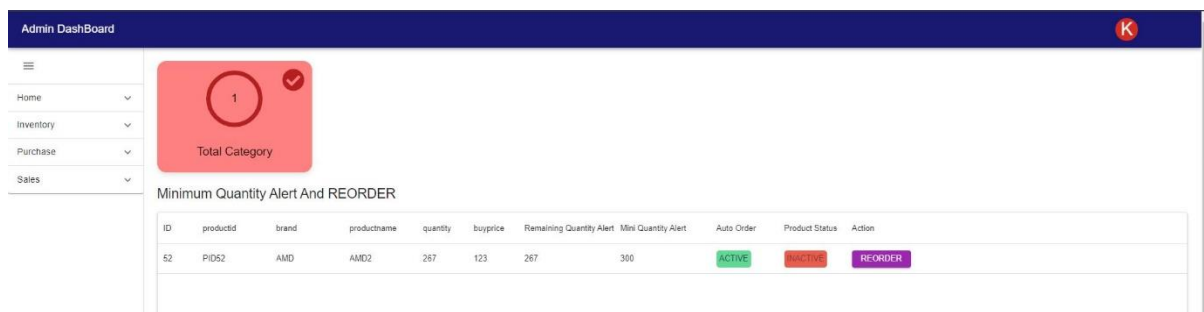
Figure 14 Inventory

Provides administrators with a user-friendly interface in the inventory section so they can effectively manage their inventory and conduct CRUD (Create, Read, Update, Delete) actions on products. With two views a thorough list format and an

attractive card format it gives flexibility. The built-in search feature makes it simple for administrators to look for particular products.

In addition to performing typical CRUD operations, administrators can use this interface to start product reorders, make products active or inactive, and change the state of auto reorders. They may swiftly restock stock levels thanks to the "Reorder" button, ensuring efficient inventory management.

8.1.2.2 Minimum Quantity Alert



The screenshot shows the Admin Dashboard with a sidebar menu on the left containing 'Home', 'Inventory', 'Purchase', and 'Sales'. A red notification card at the top right displays 'Total Category' with a count of 1. Below this, a table titled 'Minimum Quantity Alert And REORDER' is visible. The table has columns for ID, productid, brand, productname, quantity, buyprice, Remaining Quantity Alert, Mini Quantity Alert, Auto Order, Product Status, and Action. One product is listed with ID 52, productid PID52, brand AMD, productname AMD2, quantity 267, buyprice 123, Remaining Quantity Alert 267, Mini Quantity Alert 300, Auto Order set to ACTIVE, and Product Status set to INACTIVE. The Action column contains buttons for 'ACTIVE', 'INACTIVE', and 'REORDER'.

ID	productid	brand	productname	quantity	buyprice	Remaining Quantity Alert	Mini Quantity Alert	Auto Order	Product Status	Action
52	PID52	AMD	AMD2	267	123	267	300	ACTIVE	INACTIVE	REORDER

Figure 15 Minimum Quantity Alert

The process of locating and managing products with low stock levels is simplified by this web page. For effective inventory control, it automatically flags them as inactive to avoid overselling and offers managers an easy way to inspect and reorder goods as necessary.

8.1.2.3 Add New Item

Vendor-supplied products must be entered before adding them to the inventory, according to the Add Product page. Administrators are required to enter thorough product information and photos, and built-in validation ensures that the data is accurate. This verification guarantees accurate records and makes it possible to manage inventory effectively.

8.1.2.4 Add Categories

Administrators can add, activate, Inactivate, and delete product categories using the "Add Category" webpage. Administrators can easily monitor and manage categories because of the status cards at the top, which give a brief summary of the category information. Maintaining a systematic and well-organized product catalogue requires this feature.

8.1.2.5 2D map

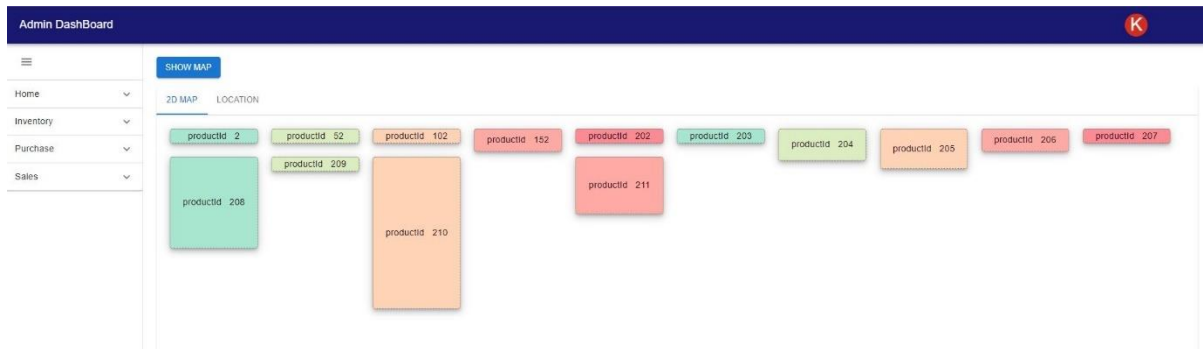


Figure 16 Inventory 2D map

A dynamic and well-organized visual representation of your inventory is provided by the 2D Map webpage, where products are grouped in a grid arrangement with a set number of columns and dynamically modifying rows. In particular when working with a big number of products, this style efficiently allocates space for each item while also assisting users in managing and navigating their inventory.

```
@Component
public class Warehouse {
    private final int numRows = 10; // Number of rows in the warehouse
    private final int numColumnsPerRow = 10; // Number of columns per row in the warehouse
    private boolean[][] occupiedLocations = new boolean[numRows][numColumnsPerRow];
    private final Map<Integer, Location> productLocations = new HashMap<>();

    public Location[] addProducts(Product[] products) {
        Map<Integer, Location> productLocations = new HashMap<>();
        occupiedLocations = new boolean[numRows][numColumnsPerRow];
        Location[] locations = new Location[products.length];
        for (int i = 0; i < products.length; i++) {
            Product product = products[i];
            double volume = product.getVolume();
            double totalVolume = product.getTotalVolume();
            // Calculate the number of shelves and breadth required
            int noOfShelves = (int) Math.ceil(totalVolume / 450000); // 450000 is the default shelf volume
            int breadthRequired = (int) Math.ceil(noOfShelves / 5.0);
            breadthRequired = breadthRequired * 15; // 15 minimum breadth required
            int columnIndex = findEmptyColumn();
            if (columnIndex != -1) {
                Location location = allocateProduct(columnIndex, product, noOfShelves, breadthRequired);
                locations[i] = location;
            }
        }
        return locations;
    }

    private int findEmptyColumn() {
        for (int row = 1; row <= numRows; row++) {
            for (int column = 1; column <= numColumnsPerRow; column++) {
                if (!occupiedLocations[row - 1][column - 1]) {
                    return (row - 1) * numColumnsPerRow + (column - 1);
                }
            }
        }
        return -1;
    }

    private Location allocateProduct(int columnIndex, Product product, int noOfShelves, int breadthRequired) {
        int row = columnIndex / numColumnsPerRow + 1;
        int column = columnIndex % numColumnsPerRow + 1;
        int identifier = (row * 1000) + column;
        Location location = new Location(row, column, noOfShelves, product.getTotalVolume(), breadthRequired,
            product.getProductid(), product.getQuantity(), true);
        occupiedLocations[row - 1][column - 1] = true;
        productLocations.put(identifier, location);
        return location;
    }
}
```

```

public Location[] getAllLocations() {
    Location[] allLocations = new Location[numRows * numColumnsPerRow];
    int index = 0;
    for (int row = 1; row <= numRows; row++) {
        for (int column = 1; column <= numColumnsPerRow; column++) {
            long productId = -1L; // (not occupied)
            int quantity = 0; // (not occupied)
            int columnIndex = (row - 1) * numColumnsPerRow + (column - 1);
            int identifier = (row * 1000) + column;
            Location location = productLocations.get(identifier);
            if (location != null) {
                productId = location.getProductId();
                quantity = location.getQuantity();
                allLocations[index] = new Location(row, column, location.getNoOfshelves(), location.getTotalVolume(),
                    location.getBreadthRequired(), productId, quantity, true);
            } else {
                allLocations[index] = new Location(row, column, 0, 0.0, 15, -1, 0, false);
            }
            index++;
        }
    }
}

```

Code Snippet 1 2D map

This code implements a warehouse storage management system. It models a warehouse with rows and columns, tracks which locations are occupied, and allows adding new products to empty locations. When adding products, it calculates the number of shelves and breadth needed based on product volume. It allocates products to the first available column with enough space, records the location details, and returns the locations to the caller. The code also allows retrieving all warehouse locations and their occupancy status.

8.1.3 Purchase Management

8.1.3.1 Purchase

Tracking and controlling purchase orders is handled through a central purchase webpage. For managing and tracking purchase orders, it works as a complete tool. Administrators can easily track the progress of each order from placement to receipt and bill payment thanks to the list of purchase orders that is provided together with their information and dynamic updates to order status. The buy management process is streamlined by this functionality, and the procurement process is made transparent and accountable.

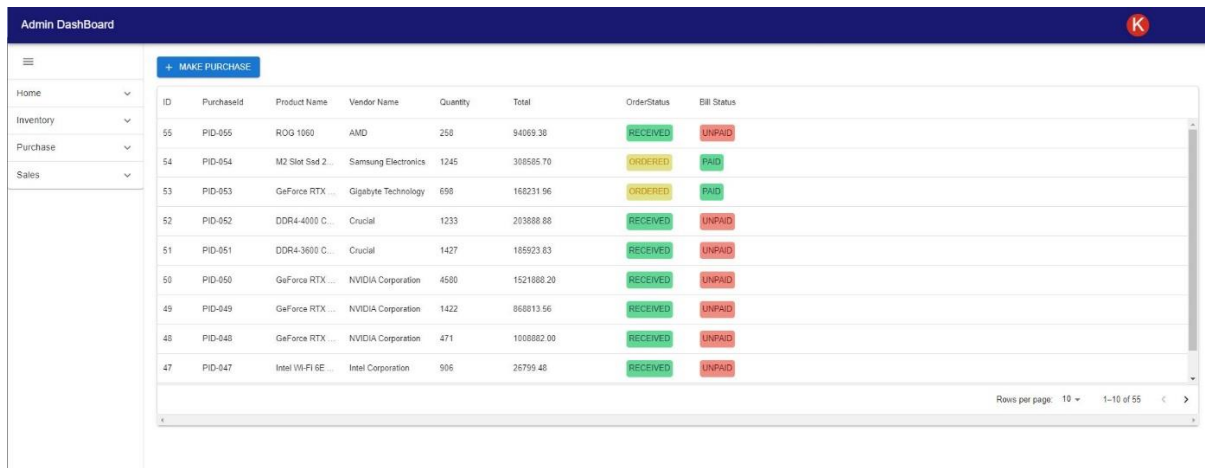
8.1.3.2 Vendor Management

ID	Full Name	Brand name	DisplayName	Email	Contact Number	Location	Country	Status	Action
28	MR. Henry Carter	ASRock	ASRock	krunaldh@le1999@gmail.com		Nairobi (Kenya)	Kenya	ACTIVE	
27	MRS. Emily Green	Fractal D.	Fractal Design	krunaldh@le1999@gmail.com		Lagos (Nigeria)	Nigeria	ACTIVE	
26	MR. Samuel Baker	NZXT	NZXT	krunaldh@le1999@gmail.com		Cairo (Egypt)	Egypt	ACTIVE	
25	MRS. Abigail Turner	Cooler Ma...	Cooler Master	krunaldh@le1999@gmail.com		Riyadh (Saudi ...	Saudi Arabia	ACTIVE	
24	MR. Benjamin Lewis	Razer Inc.	Razer Inc.	krunaldh@le1999@gmail.com		Seoul (South ...	South Korea	ACTIVE	
23	MRS. Amelia Adams	Logitech	Logitech	krunaldh@le1999@gmail.com		Madrid (Spain)	Spain	ACTIVE	
22	MR. Oliver Hall	Toshiba C...	Toshiba Corpo...	krunaldh@le1999@gmail.com		Rome (Italy)	Italy	ACTIVE	

Figure 17 vendor Mangement

Information management for vendors is facilitated via the Vendor section. With the ability to activate, modify, or delete them, the Vendor List page offers a list of all current vendors. A simple form for adding new vendor information is available on the Add Vendor page, and validation tests are included to guarantee the validity of the input. By making it simple to retain and modify vendor information as necessary, this functionality simplifies vendor management.

8.1.3.3 Purchase Order



ID	Purchaseid	Product Name	Vendor Name	Quantity	Total	Order/Status	Bill Status
55	PID-055	ROG 1060	AMD	250	94069.30	RECEIVED	UNPAID
54	PID-054	M2 Slot Ssd 2	Samsung Electronics	1245	308585.70	ORDERED	PAID
53	PID-053	GeForce RTX	Gigabyte Technology	698	168231.96	ORDERED	PAID
52	PID-052	DDR4-4000 C	Crucial	1233	203088.88	RECEIVED	UNPAID
51	PID-051	DDR4-3600 C	Crucial	1427	185923.83	RECEIVED	UNPAID
50	PID-050	GeForce RTX	NVIDIA Corporation	4580	1521888.20	RECEIVED	UNPAID
49	PID-049	GeForce RTX	NVIDIA Corporation	1422	868813.56	RECEIVED	UNPAID
48	PID-048	GeForce RTX	NVIDIA Corporation	471	1008882.00	RECEIVED	UNPAID
47	PID-047	Intel Wi-Fi 6E	Intel Corporation	906	26799.48	RECEIVED	UNPAID

Figure 18 Purchase Management list

Using the Purchase Item Form, administrators can place orders for goods from suppliers quickly. They decide on a vendor, enter information about the product, such as its name, category, quantity, and price, get quantity recommendations, and validate the information. The system then connects users to an email page where administrators can submit orders to vendors along with invoice attachments and autogenerated content. This streamlines management and communication of procurement.

8.1.3.4 Receive Order

The ability to mark Ordered Products as Received Product has two pages, one of which lists ordered and received products with status cards displaying pending and received order statistics. The form to enter product IDs and mark orders as received is on the second page. By precisely updating inventory when products are received based on order information, this simplifies order tracking.

8.1.3.5 Billing

The Billing page includes lists of paid and unpaid bills, along with status cards that provide payment information. Administrators can pay for outstanding bills using integrated forms that make it easy to choose payment options and record transaction information. The system updates bill statuses appropriately after a successful payment. This functionality tracks unpaid, paid, and settled debts to manage billing in an efficient manner.

8.1.4 Sales Management

8.1.4.1 Sales

Admin have an effective approach to handle and keep track of sales orders and payments on the Sales page. It makes it simpler to guarantee that all purchases are completed accurately and quickly by providing a thorough view of sales order data, payment statuses, and payment history.

8.1.4.2 Orders

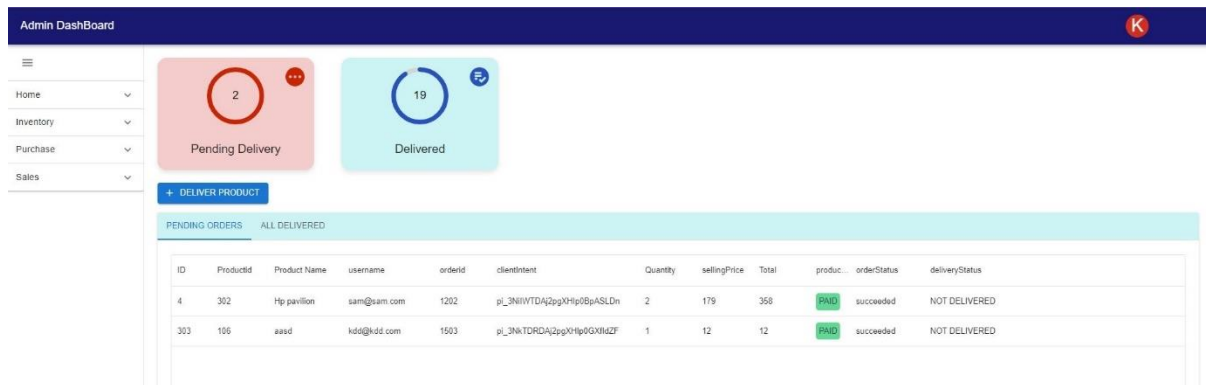


Figure 19 Sales Order

The Order page gives administrators the resources they need to efficiently handle and track user orders. When products are in stock, it ensures that they are delivered quickly to clients and marks them as delivered in the system for record-keeping. This streamlines the delivery process.

8.1.4.3 Refund Orders

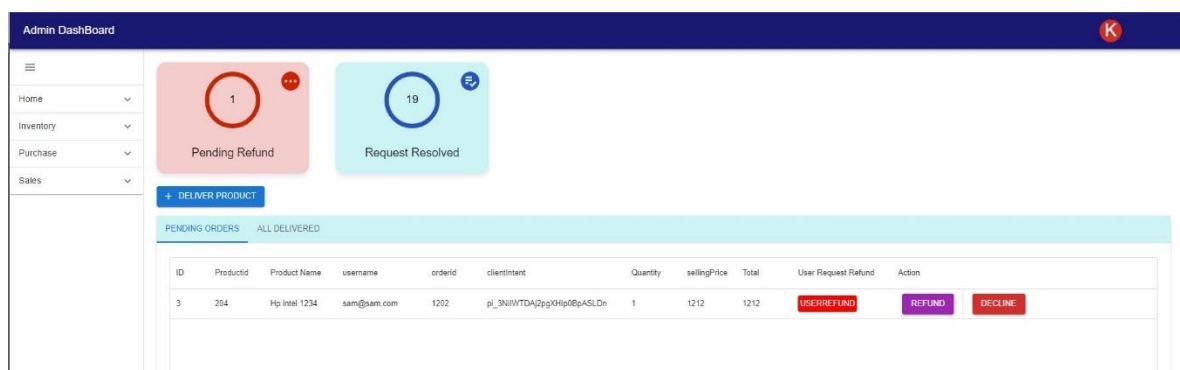


Figure 20 sales Refund Request

Admin may review the details of user refund requests on the Refund Tracking page and choose whether to approve or reject them using the action buttons. Refund requests that have been approved are processed using integrated payment gateways like Stripe, which manage the financial transaction, update statuses, and keep an accurate record of all activities taken. This makes managing refunds easier.

8.2 User Interface

8.2.1 Home Page

The homepage features a Hero section with images and a call to action to draw visitors in, displays new and featured products, and offers users site navigation. "Call to action" is referred to as CTA. It is a component on a webpage—typically a button or link—that encourages users to perform a desired action, such as signing up for something or buying for something. It acts as the primary entry point, giving an overview and inciting consumers to explore options by exhibiting goods and branding alongside navigation.

8.2.2 Shop Page

Through a variety of filters and features, users may conveniently browse and buy products on the Shop page.

Filters are available on the left sidebar to narrow down searches by product type and category. Users can pick certain categories to view matched products and filter results to new, featured, or all items. Sorted product listings that fit the filters are shown in the main area. Customers can sort products by price and click on specific items to view details. In order to produce a user-friendly shopping experience that makes it simple to identify and assess pertinent products, this combines filtering, sorting, and exploration of product listings.

Overall, the flexible filtering and sorting options make searching easier, and the well-organized product lists facilitate exploration. The shopping and purchasing processes are improved by this Shop page interface.

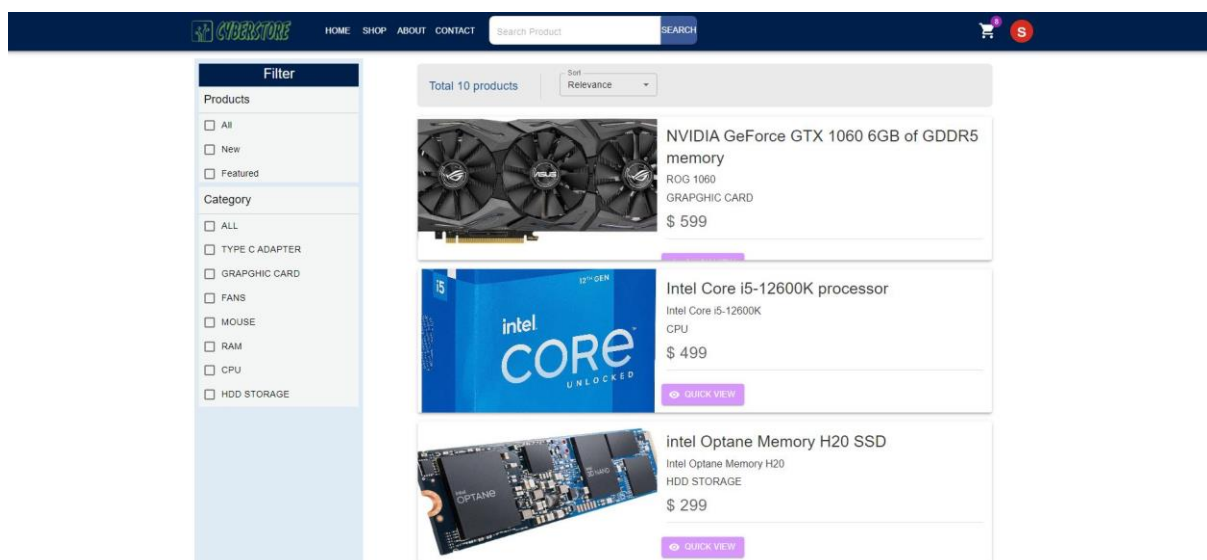


Figure 21 User Shop Page

8.2.3 View Product Page

The View Product page offers thorough information on a specific product, including images, costs, supply information, and a product summary. Users have

the option to enlarge the pictures for improved visibility. Using integrated buttons, users can change their order quantities, add products to their cart, or Proceed to buy directly. Customers' purchasing processes are made simpler by this integrated presentation of product information and purchasing options.

8.2.4 Shopping Cart Page

Users can manage the things they have chosen and make final purchases from the Shopping Cart page. It enables opportunities for users to change quantities, delete things, or carry on shopping while offering transparency in terms of product details and costs. This improves the whole shopping experience.

8.2.5 Order Confirmation and Checkout Pages

The website's checkout procedure consists of a succession of pages that lead users through order confirmation, delivery, payment, and successful completion of their orders.

Before moving on to the checkout page, customers can examine their order details, confirm their intentions, and enter their shipping addresses or choose the last address they used. The integrated Stripe gateway, which enables secure transactions, is used on this page to gather payment information. Customers are routed to an order success page after making a successful payment. Customers may efficiently check out with the help of this user-friendly checkout process, which also offers transaction security.

8.2.6 Order Tracking and Refund

Customers can monitor order statuses, such as delivered or not delivered, submit refund requests, and receive updates on approvals and processing on the order tracking and refund page. This page also provides transparency by sending automatic alerts at each stage of the process.

```
// this code is taken from the official stripe Website. It is predefined code, I have modified the code upto some extent to
//match my requirement. It is predefined code to work with stripe .
@Service
public class StripeService {
    private final OrderSuccessRepo orderSuccessRepo ;
    private String stripeApiKey =
"sk_test_51NeDLBDaj2pgXHlpCjGT3R6kF0ZfO02G2xjcZZuWvVEf3i4qkT0060bxzWjYeXD7794dT0AYMwz5HXPS2p
QuGh100GZYK297e";
    public StripeService(OrderSuccessRepo orderSuccessRepo) {
        this.orderSuccessRepo = orderSuccessRepo;
    }

    @PostConstruct
    public void init(){
        Stripe.apiKey = stripeApiKey;
    }

    private int calculateOrderAmount(Object[] items) {
        int totalAmount = 0;
        for (Object item : items) {
            if (item instanceof CreatePaymentItem) {
                CreatePaymentItem paymentItem = (CreatePaymentItem) item;
                String itemIdString = paymentItem.getId();
                try {
                    int itemId = Integer.parseInt(itemIdString);
                    totalAmount += itemId;
                } catch (NumberFormatException e) {
                    e.printStackTrace();
                }
            }
        }
        return totalAmount*100;
    }
}
// code continue -----
```

```

// code continue -----

public CreatePaymentResponse createPaymentIntent(CreatePayment createPayment) throws StripeException {
    int amount = calculateOrderAmount(createPayment.getItems());
    PaymentIntentCreateParams params = PaymentIntentCreateParams.builder()
        .setAmount(new Long(amount))
        .setCurrency("gbp")
        .setAutomaticPaymentMethods(
            PaymentIntentCreateParams.AutomaticPaymentMethods.builder()
                .setEnabled(true)
                .build()
        )
        .build();
    PaymentIntent paymentIntent = PaymentIntent.create(params);
    CreatePaymentResponse paymentResponse = new
    CreatePaymentResponse(paymentIntent.getClientSecret());
    return paymentResponse ;
}

public String refundProduct(Long id) throws StripeException {
    OrderSuceess orderSuceess = orderSuccessRepo.findById(id).orElse(null);
    if(orderSuceess != null){
        RefundCreateParams params =
            RefundCreateParams.builder()
                .setPaymentIntent(orderSuceess.getClientIntent())
                .setAmount((long) orderSuceess.getTotalprice()*100)
                .build();
        Refund refund = Refund.create(params);
        CreatePaymentResponse paymentResponse = new CreatePaymentResponse(refund.getId());
        return paymentResponse.getClientSecret();
    }
    return null;
}
}
// https://stripe.com/docs/checkout/quickstart

```

Code Snippet 2 Stripe Integration

Utilising the Stripe API, this code implements payment processing. It has a `StripeService` that takes care of setting up payment intents, collecting money, and cancelling orders. The order amount is determined by the `createPaymentIntent` function, which also produces a Stripe payment intent. The `refundProduct` method retrieves the payment intent ID, generates a refund on Stripe, and retrieves the order. The code is customised to interface with the order processing flow, such as retrieving order data from the database prior to refund, although it is adapted from the Stripe documentation [14]. Overall, it demonstrates how to include Stripe payment processing into an application for order management.

8.3 Authentication Interface

8.3.1 Login Page

```
@Configuration
public class SecurityConfiguration {
    private final RSAKeyProperties keys;
    public SecurityConfiguration(RSAKeyProperties keys) {
        this.keys = keys;
    }

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(UserDetailsService detailsService){
        DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(detailsService);
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(daoAuthenticationProvider);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> {
                auth.requestMatchers("/auth/**").permitAll();
                auth.requestMatchers("/admin/**").hasRole("ADMIN");
                auth.requestMatchers("/user/**").hasAnyRole("ADMIN", "USER");
                auth.requestMatchers(HttpMethod.GET, "/user/**").permitAll();
                auth.anyRequest().authenticated();
            });

        http
            .oauth2ResourceServer()
            .jwt(jwt -> jwtAuthenticationConverter(jwtAuthenticationConverter()));

        http
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        return http.build();
    }

    @Bean
    public JwtDecoder jwtDecoder(){
        return NimbusJwtDecoder.withPublicKey(keys.getPublicKey()).build();
    }

    @Bean
    public JwtEncoder jwtEncoder(){
        JWK jwk = new RSAKey.Builder(keys.getPublicKey()).privateKey(keys.getPrivateCrtKey()).build();
        JWKSource<SecurityContext> jwks = new ImmutableJWKSet<>(new JWKSet(jwk));
        return new NimbusJwtEncoder(jwks);
    }

    @Bean
    public JwtAuthenticationConverter jwtAuthenticationConverter(){
        JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter = new JwtGrantedAuthoritiesConverter();
        jwtGrantedAuthoritiesConverter.setAuthoritiesClaimName("roles");
        jwtGrantedAuthoritiesConverter.setAuthorityPrefix("ROLE_");

        JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(jwtGrantedAuthoritiesConverter);
        return jwtAuthenticationConverter;
    }

    @Bean
    public FilterRegistrationBean<CorsFilter> corsFilter(){
        CorsConfiguration corsConfiguration = new CorsConfiguration();
        corsConfiguration.setAllowedOrigins(Collections.singletonList("http://localhost:3000"));
        corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        corsConfiguration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        corsConfiguration.setAllowCredentials(true);
        UriBasedCorsConfigurationSource source = new UriBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", corsConfiguration);
        CorsFilter corsFilter = new CorsFilter(source);
        FilterRegistrationBean<CorsFilter> registrationBean = new FilterRegistrationBean<>(corsFilter);
        registrationBean.setOrder(Ordered.HIGHEST_PRECEDENCE);
        return registrationBean;
    }
}
```

Code Snippet 3 Spring Boot Security [10]

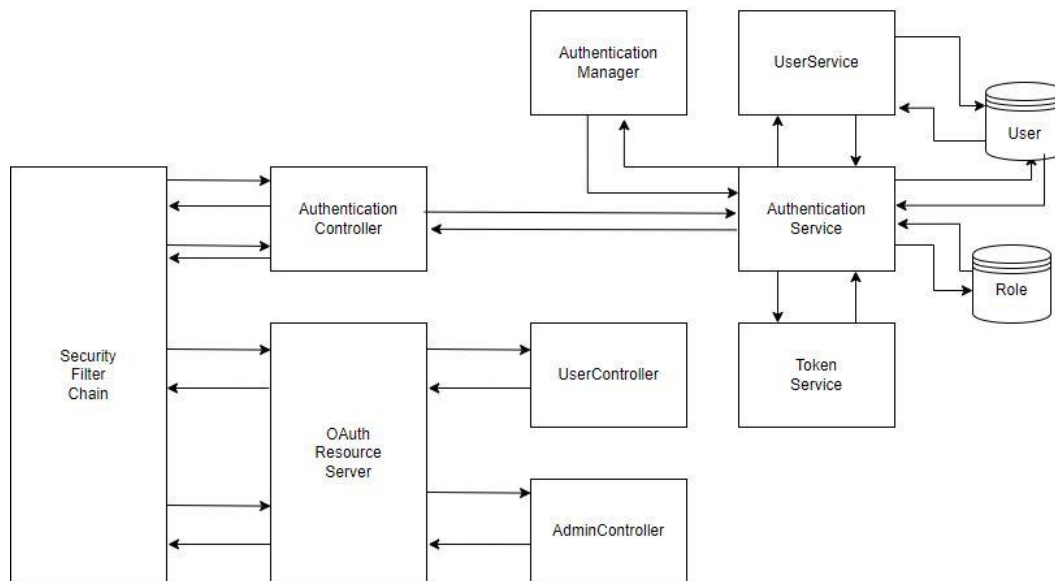


Figure 22 Login Diagram to understand complex flow of using JWT and user based authentication [10]

I leveraged this Spring Security configuration example while implementing security in my own web application. Though I did not author it [10], studying and understanding the code gave me crucial knowledge on utilizing Spring Security's complex authentication, authorization, and CORS functionality. I was then able to apply these concepts in a customized way to securely meet my project's specific requirements.

Spring Security is set up for a web application using the Java code above. `DaoAuthenticationProvider` and `BCrypt` password encoding are used to configure authentication. Based on roles, it specifies authorisation criteria for various URL patterns. The programme validates JWTs by serving as an `OAuth2 resource server`. Management of sessions is stateless. The code sets up cross-origin `Http` request support with `CORS` filtering, `JWT` encoding and decoding, and a `JwtAuthenticationConverter` to handle `JWT` authentication. This configures a secure Spring web app with role-based access control, `JWT` authentication, and `CORS` support.

8.3.2 Registration Page

New users can create accounts on the registration page by providing information like name, email, and a secure password. Before submission, input validation verifies data types, and server-side validation confirms the accuracy, confidentiality, and uniqueness of information prior to account creation. Before being stored in databases, passwords are hashed for encryption. Overall, the multi-step hashing and validation of user-provided data improve security, and helpful feedback guarantees a simple signup process.

9 Testing

9.1 Manual Testing

Web application manual testing is repeatedly engaging with the system to test the user interface, functionality, forms, navigation, and workflows of the programme. The application's behaviour is checked by clicking through the various screens and web pages, filling out forms, and navigating between pages. Simulating actual user situations is the main goal, along with verifying that the requirements have been met. Manual testing offers quick validation of web applications, particularly for user-visible functionality and workflows [25].

Description	Expected Outcome	Actual Outcome	Pass / Fail
Test 01 - Login Function Check Login using valid credential	Register user with valid credential should be logged in and redirected to home page and JWT token should be generated and access to all recourses allocated to Customer	As Expected	Pass
Test 02 - Login Function Check Login using invalid credential. Invalid Password or Invalid Username	Register User with Invalid credential should be not allowed to logged in and show the Error message "Username or Password does not match" and not allow user to access recourses.	As Expected	Pass
Test 03 - Login Function Check Login using Empty credential. No username and no password	System should not submit the form and ask user to fill the details and check whether the enter details are correct according to the specified validation schema	As Expected	Pass
Test 04 - Login Function Check Login using credential that does not follow the validation schema such as email format and password should be 8 character long which include at least one Lower, Upper case and number and symbols	System should not submit the form and ask user to fill the details correctly and again check whether the enter details are correct according to the specified validation schema	As Expected	Pass
Test 05 - Login Function Check Login using valid credential but user has not	System should not allow the new user to login and show Error message	As Expected	Pass

register before login. User is new try to login without register	"Username or Password does not match"		
Test 06 – Register Function Check registration using valid credential and user has enter all details.	System should take the credential and register the user, show the Success message. And redirect user to the login page	As Expected	Pass
Test 07 – Register Function Check registration using invalid credential or empty spaces. User has not entered the complete details	System should show the required fields to the user and show error message. Without full details system should show Success message to user.	As Expected	Pass
Test 08 –Login Function Check Login function, when admin enter the Admin Credential than, Admin should be navigated to the Admin Dashboard and when User login with User credential they should be navigated to user dashboard.	System should navigate to admin dashboard if admin credential is match else all normal users should be navigated to user dashboard. No normal user should have access to admin dashboard.	As Expected	Pass
Test 09 –Admin / Vendor Check Add vendor, Admin should add the new vendor details. With correct information and validation.	System should check if the enter credential are correct according to validation schema then add the vendor to database and show the success message.	As Expected	Pass
Test 10 –Admin / Vendor Check Add vendor, Admin should add the new vendor details. Admin should not enter the existing vendor email address. As it should be unique.	System should not save the vendor which as same email as existing vendor and show the Error message to admin.	As Expected	Pass
Test 11 –Admin / Purchase Check purchase order function. Admin should enter all the required details and then save and sent the email to the Vendor email id. And vendor should get the email with custom invoice and attachment.	System should not save purchase order if admin has not entered all the details. System should show the required fields to be fill. If details are correct then system should redirect the user to email page and send mail with custom details of purchase product along with Invoice attach to it and finally	As Expected	Pass

	navigate the user Purchase List.		
Test 12 –Admin / Purchase Check purchase order function. Admin entered the invalid data that does not satisfy the Validation schema.	System should not allow admin to continue. And show error message.	As Expected	Pass
Test 13 –Admin / Received Order Check Received order function. Admin can mark the product that are received	System should show the list of purchase order that were ordered by admin and now admin can mark them as received. Purchase order should change the status from Ordered to Received. Redirect to the received list	As Expected	Pass
Test 14 –Admin / Received Order Check Received order function. Admin Can't add any purchase order without ordering them. Or once the purchase is marked as received it shouldn't be marks as received again.	System should not show the purchase order which are marked received previously. If admin enters any wrong purchase id then admin shouldn't be allowed to mark this product as received.	As Expected	Pass
Test 15 –Admin / Billing Check Billing function. Admin can mark the bills as paid when the products are purchase and bill is generated. Or after the receiving the product.	System should show the list of unpaid bills to admin and then admin and pay the bills. And bills should be marked as paid. Redirect admin back to Billing list.	As Expected	Pass
Test 16 –Admin / Sales Check sales order function. Admin can deliver the order product from the sales order list.	System should show the sales order that are not delivered by the admin. Admin can only deliver the product which are not delivered.	As Expected	Pass
Test 17 –Admin / Sales Check sales order function. Admin Cannot deliver the order products if the product quantity in inventory are not enough to fulfil the delivery	System should not allow the admin to deliver the product which quantity are less the ordered product	As Expected	Pass
Test 18 –Admin / Sales Check refund function. Admin can refund the product or decline the refund	System should allow the admin to either decline or refund the payment back to user. If decline admin can't refund it and vice versa. Admin can't refund	As Expected	Pass

	twice. This is one-time process.		
Test 19 –Admin / Sales Check refund function. Admin can try to refund the same request twice.	System will not allow admin to make the payment twice for the same request	As Expected	Pass
Test 20 – Admin / Inventory Check Add product function. Admin can add the products only when they are received otherwise admin can't add any new product.	System will show the product that are received by the admin. And admin is allowed to use this quantity for new product. And enter all the details and save to inventory	As Expected	Pass
Test 21 –Admin / Inventory Check Add product function. Admin need to enter all the valid details of product along with the images.	System should check the details enter by admin. If the details do not follow validation rules then system should show error message and show admin to enter all the required fields and should save data in database.	As Expected	Pass
Test 22 – Customer / Shop Check shopping list function. User can access the shopping list and click to check the product details.	System should show the shopping list to the registered user. User can click on the button to view the product details on another page. System should allow user to filter the product.	As Expected	Pass
Test 23 – Customer / shop Check shopping list function. User should be navigated to product details page if quick view button is clicked	System should allow the register user to navigate to the product details page.	As Expected	Pass
Test 24 – Customer / Product Details Check the product details function. User should be able to add or remove the quantity and zoom the image of product.	System should allow the register user to zoom the product images. And Update the quantity but minimum quantity should be 1 and it should not go below 0 or in minus. User can click the button add to cart to save the product and quantity in cart	As Expected	Pass
Test 25 – Customer / Cart Check Cart function. User can add the product and quantity	System should keep the track of the product and its quantity, user wish to buy.	As Expected	Pass

that he selected. And should be saved in the cart. Until user place the order. User can modify the quantity in cart.	And save this data until the user buy this product. User is allowed to modify the product quantity or remove them.		
Test 26 – Customer / Order Confirmation Check the order confirmation progress. User click on proceed to buy. User will see the confirmation page. It will show the order summary of cart.	System should show the cart summary, and it should be the same product and quantity user has saved. And user can check the total price of all the product. If still user is not sure to buy he is allowed to go back.	As Expected	Pass
Test 27 – Customer / Address Check the address function. User will enter the delivery address. And then progress forward. Or user can select previous address.	System should not allow user to move forward without proper address. System will save the previous address. User is allowed to select the previous address or enter new address	As Expected	Pass
Test 28 – Customer / Stripe Check the Stripe Function. User enter the correct card details.	System Should not all user to enter the wrong card details. System will check the details and then validate them. Then only user can make payment.	As Expected	Pass
Test 29 – Customer / Order track Check the order track function. Register user tracking the order that he ordered.	System should mark the product as ordered is the payment Is done. And user can track if product.	As Expected	Pass
Test 30 – Customer / Order Refund Check the refund function. User click on the refund button.	System should enable the refund button once product is delivering and after that user can request for refund only one time. User need to wait for admin response.	As Expected	Pass

9.2 Integration Testing

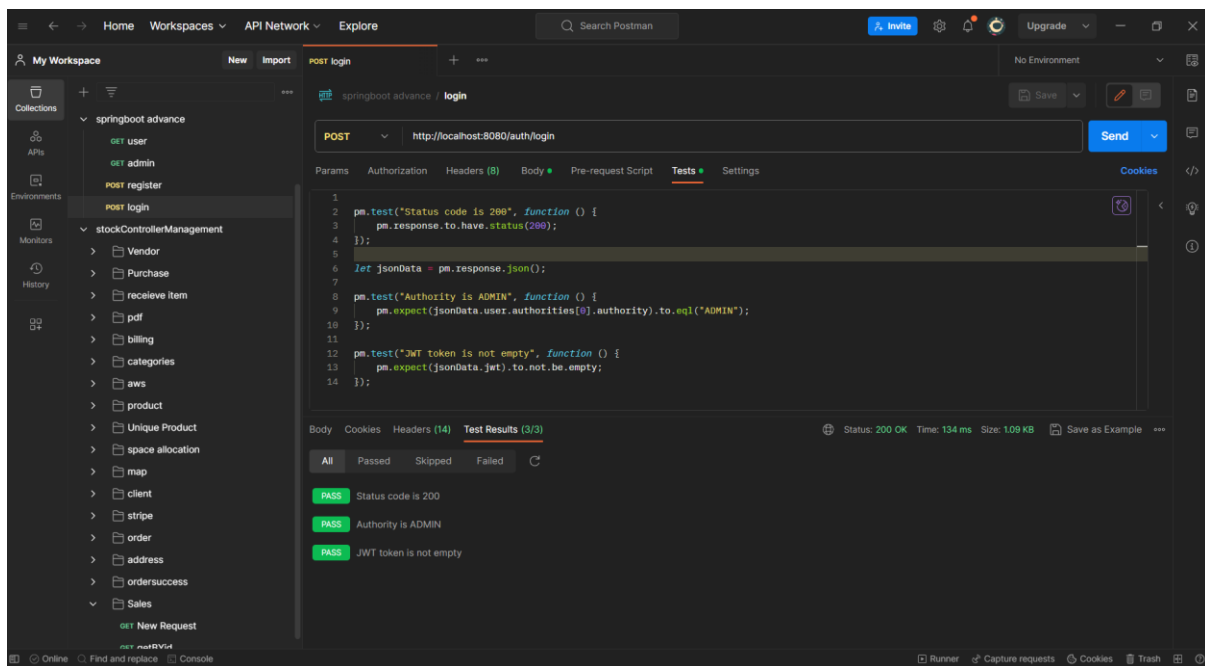


Figure 23 Postman Integration API Testing

For complicated applications, integration testing is essential to ensure that various components function properly together. I set up parameters, define test scenarios, write test suites with scripts to validate responses, set request dependencies, use variables for dynamic data, run manually or automatic tests, examine the results to find errors, and iterate to increase coverage using Postman. By imitating real-world circumstances, I can execute integration testing in Postman efficiently, validating end-to-end behaviour and spotting integration flaws early on [25].

10 Conclusion

10.1 Future improvement

10.1.1 Integrating Chat Bot

I intend to use Google Dialogflow to incorporate a chatbot into the web application to give consumers a conversational experience. Building the chatbot logic in Dialogflow in the first step includes creating intents, entities, dialogue flows, and training words. This will enable me to develop a model of natural language comprehension that can relate user inquiries to predetermined intents [26].

After the chatbot has been trained, I will use Dialogflow's webhook connection to link it to the web application's backend. Dialogflow will contact my webhook with the determined intent and entities if a user inquiry is received. These intents will be mapped to API endpoints in my backend, which will perform the logic and deliver responses [26].

The Dialogflow chat interface on my website is used to respond to users after Dialogflow receives the webhook response. This enables me to seamlessly integrate the chatbot with my web app while maintaining the chatbot logic's separation. I'll be able to provide users a simple means of obtaining information or carrying out activities through casual conversational engagements due to the chatbot [26].

In conclusion, Dialogflow gives me a strong foundation on which to create the chatbot's natural language understanding (NLU), the webhook integration connects it to my backend APIs, and the conversational interface on my website gives users a helpful virtual assistant.

To demonstrate how the integration might function, consider these examples of chatbot interactions:

User: "Buy two AMD Ryzen 5 CPUs"

The chatbot could identify an intent like "**purchase_product**" in this scenario and extract entities for "AMD Ryzen 5 CPU" and quantity 2. With these entities and with this aim, the webhook would be called. Two AMD Ryzen 5 CPUs would be added to the user's cart by the backend API when it was mapped to the buy intent. Following confirmation from the webhook, the chatbot would say,

Chatbot: "OK, I've added 2 AMD Ryzen 5 CPUs to your cart."

10.1.2 Suggesting the product or Autosuggest Product

This feature is the next step to previous feature. The user's request for product recommendations from the chatbot sets off a particular intent. This intent has a connection to a backend API that complies with the OpenAPI specification and can access user information like order histories and profiles [27].

Through an API call, the backend API interfaces with ChatGPT and sends user data. By using this information to analyse user preferences, ChatGPT can produce text-based responses that include product recommendations.

From ChatGPT's answer, the API extracts the product recommendations, and then it uses a webhook to send them back to the chatbot. The chatbot then formats and shows the user these recommendations. In essence, it's a sophisticated but effective integration that blends user information, AI-driven responds, and APIs to provide customised product recommendations. Also, we can create backend service independent of chatbot and we can use same logic to get the suggested product and display them on user dashboard. When website load user can see Auto suggested products [27].

10.2 Evaluation

I make thorough feature comparison tables for inventory systems that include all essential features required in areas like notifications, warehouse management, order processing, etc. It offers a clear data to assess where each alternative excels

or falls short for their objectives by extensively evaluating each solution and noting which aspects are present or absent in the table. This eliminates bias and provides an objective, side-by-side perspective of differences.

Feature	My WebApp	Zoho Inventory [7]	Inventory Planner [8]
Admin Dashboard	Yes	Yes	Yes
Customer Dashboard or website	Yes	No	No
Add Update Delete product	Yes	Yes	Yes
Add Minimum Quantity Alert	Yes	Yes	No
One Click Reorder Function	Yes	No	No
Make product active or inactive from inventory	Yes	No	
Send Email to vendor	Yes	Yes	Yes
Send Invoice to the Vendor	Yes	Yes	No
Track Purchase Order	Yes	Yes	Yes
Track Bills	Yes	Yes	Yes
Add Category	Yes	Yes	Yes
2D map of Inventory	Yes	No	No
Track dynamic sales	Yes	No	No
Track Delivery product	Yes	Yes	Yes
Stripe Integration on User Dashboard	Yes	No	No
Stripe Refund on Admin Dashboard	Yes	No	No
Save Payment tracking code	Yes	No	No
Export data to Csv or Excel	Yes	Yes	Yes
Data visualization	Yes	Yes	Yes
Chatbot	No	Yes	Yes
Third Party Integration	No	Yes	Yes
Cloud Based Application	No	Yes	Yes

11. References

- [1] H. Ford, "brainyquote," brainyquote, [Online]. Available: https://www.brainyquote.com/quotes/henry_ford_122318.
- [2] C. Waters, Inventory Control and Management, Chichester: Wiley, 2003.
- [3] S. L. a. J. B. H. D. Bloomberg, Logistics. Upper Saddle River, NJ: Prentice Hall, 2002.
- [4] M. Müller, Essentials of Inventory Management., New York: AMACOM, 2003.
- [5] Curity, "JWT Security Best Practices," 2020. [Online]. Available: <https://curity.io/resources/learn/jwt-best-practices/>.
- [6] J. e. a. Mostard, "Forecasting demand for single-period products: A case study," *International Journal of Physical Distribution & Logistics Management*, vol. 35, no. 5, pp. 320-332, 2005.

- [7] Z. C. P. Ltd., "ZOHO Inventory," Zoho Corporation Pvt. Ltd., 2003. [Online]. Available: <https://www.zoho.com/uk/inventory/>.
- [8] Sage, "Inventory Planner by sage," Inventory Planner, 2023. [Online]. Available: <https://www.inventory-planner.com/>. [Accessed 2023].
- [9] F. Std, "Create Responsive Admin Dashboard | HTML CSS JavaScript | With Source Code," Youtube, 2 October 2021. [Online]. Available: https://www.youtube.com/watch?v=CkVrmLLHmul&t=276s&ab_channel=FajarStd. [Accessed 12 june 2023].
- [10] u. Koder, "Github unknown Koder," GitHub, 3 April 2023. [Online]. Available: <https://github.com/unknownkoder/spring-security-login-system>. [Accessed 2023].
- [11] L. C. W. Durgesh, "Simple 5 Steps to Send Email using Java and Gmail | With out Less Secure Ap," youtube, 14 Jan 2023. [Online]. Available: https://www.youtube.com/watch?v=hm23MfVnkCU&ab_channel=LearnCodeWithDurgesh. [Accessed 5 july 2023].
- [12] C. Dost, "10-Hour React Tutorial 2023 - Zero to Advanced," Youtube, 8 february 2023. [Online]. Available: https://www.youtube.com/watch?v=6l8RWV8D-Yo&t=29407s&ab_channel=CoderDost. [Accessed 2023].
- [13] A. S. T. C. & P. M. K. Gillis, "Agile project management (APM) CIO," 2023. [Online]. Available: <https://www.techtarget.com/searchcio/definition/Agile-project-management>.
- [14] C. p. f. (n.d.), "Stripe Documentation.," Stripe, [Online]. Available: <https://stripe.com/docs/payments/quickstart>. [Accessed 2023].
- [15] Linnworks, "9 Inventory Management Strategies To Reduce Stockouts For Ecommerce Businesses," 05 May 2023. [Online]. Available: <https://www.linnworks.com/blog/reduce-stock-levels-stock-outs>.
- [16] M. Foundation, "mdn web docs," Mozilla Foundation, 2023. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [17] M. Foundation., "mdn web docs," Mozilla Foundation., [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [18] S. Jain, "Geeksforgeeks," Geeksforgeeks, [Online]. Available: https://www.geeksforgeeks.org/java/?ref=shm_outind.
- [19] F. Inc., "reactjs," facebook Inc, 2023. [Online]. Available: <https://react.dev/>.
- [20] Vmware, "Spring by Vmware Tanzu," Pivotal Software, 2023. [Online]. Available: <https://spring.io/projects/spring-boot>.
- [21] Mui, "MUI Core," Material UI SAS, 2023. [Online]. Available: <https://mui.com/material-ui/getting-started/>.
- [22] MySQL , "MySQL Workbench. (n.d.)," [Online]. Available: <https://www.mysql.com/products/workbench/>. [Accessed 2023].
- [23] W. M. f. (n.d.). [Online]. Available: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>.
- [24] S. a. patterns, "O'Reilly Online Learning," [Online]. Available: <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>. [Accessed 2023].
- [25] J. Unadkat, "Manual testing tutorial for beginners," BrowserStack, 2023. [Online]. Available: <https://www.browserstack.com/guide/manual-testing-tutorial>.

- [26 codebasics, "End-to-End NLP Project | Build a Chatbot in Dialogflow | NLP Tutorial | S3 E2," 23 June 2023. [Online]. Available: https://www.youtube.com/watch?v=2e5pQqBvGco&ab_channel=codebasics. [Accessed 2023].
- [27 C. Builder, "How to build the best chatbot for your business: ChatGPT Builder, OpenAI & DialogFlow (May 2023)," 31 May 2023. [Online]. Available: https://www.youtube.com/watch?v=yi6rrieozMw&t=1s&ab_channel=ChatbotBuilder. [Accessed 2023].
- [28 M. Foundation., "mdn web docs," Mozilla Foundation., 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.