E-Commerce Website and Inventory System

BY

Yong Yi Kang

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

Feb 2026

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ts Lim Jit Theam who has given me this bright opportunity to engage in Final Year Project on an Integrated E-Commerce Inventory System. It is my first step to establish a career in information and communication technology filed. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Online grocery platforms have grown rapidly, but many systems still face inventory management issues when handling perishable goods. Common problems include the lack of expiry-aware stock deduction, spoilage and damage logging, predictive restocking support, and efficient bulk inventory updates. These issues may lead to inaccurate stock records, overstocking, stockouts, expired products, and food waste.

This project developed a web-based integrated e-commerce inventory system for small and medium-sized grocery businesses. The system includes standard e-commerce functions such as user authentication, product browsing, cart management, checkout, and order management. It also provides inventory-focused features, including First-Expired-First-Out (FEFO) batch deduction, spoilage logging, bulk CSV upload, stock movement tracking, financial reporting, and predictive restocking recommendations.

The system was implemented using Next.js, Spring Boot, and MySQL, with Agile Scrum adopted as the development methodology. The predictive restocking algorithm considers demand momentum, demand variability, shelf life, and spoilage risk to support better inventory decisions. System testing was conducted across major modules, and all documented test cases passed successfully.

Overall, the developed system improves stock accuracy, supports expiry-aware inventory control, reduces manual inventory workload, and provides useful restocking decision support for SME grocery retailers.

Area of Study: Web-Based System Development, Inventory Management Systems

Keywords: Grocery E-commerce, FEFO Deduction, Spoilage Logging, Predictive Restocking, Bulk Inventory Upload, Perishable Stock Management

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

The retail sector's digital transformation has boosted online grocery shopping due to convenience and accessibility. Yet, grocery e-commerce platforms struggle with perishable inventory management—challenges like poor stock tracking, no expiry logic, spoilage logging gaps, weak predictive restocking, and inefficient updates cause waste and inefficiency. This chapter introduces the motivation, background, and problem areas that form the foundation of the proposed project, which aims to develop a cost-effective, scalable, and inventory-focused e-commerce system tailored for small and medium-sized grocery businesses.

1.1 Project Background

The exponential increase in online store platforms has transformed the dynamics in how food companies operate, particularly with the increased necessity to deliver home deliveries and online orders. Store control remains critical in maintaining the efficiency in operations, product availability, and customer satisfaction. However, most current e-commerce grocery systems are yet to rely on antiquated tracking procedures, lack effective expiry control, and need greater levels of automation. This project provides a single-vendor, website-based, grocery e-commerce system specifically tailored to eliminate the above challenges of store through functionalities like real-time stock observation, expiry-dependent logic, recordation of spoilage, and anticipation on restocking. The mission remains offering the small and medium food traders a scalable, cost-friendly system capable of promoting enhancement in stock accuracy and minimising losses.

1.2 Problem Statement

1.2.1 Lack of Expiry-Aware FEFO Integration in E-Commerce Platforms

While physical grocery stores typically apply FIFO (First-In, First-Out) or FEFO (First Expired, First Out) practices to manage perishable inventory, many grocery e-commerce platforms have yet to fully integrate expiry-aware stock handling into their online systems. FIFO focuses on selling items based on arrival order, while FEFO prioritizes items with the earliest expiry dates, making it more suitable for products with short shelf life such as groceries. However, platforms

like Walmart Grocery, Tesco (Lotus's), RedMart, and Village Grocer often do not fully implement FEFO in their online checkout or stock deduction processes.

According to DCL Corp [1], many e-commerce businesses continue to default to FIFO practices, which risks newer stock being sold before older, expiring inventory. A study published in ScienceDirect also confirms that poor expiry tracking is a major hurdle in the perishables sector, preventing full FEFO adoption [2]. Anchanto adds that FEFO plays a critical role in ensuring customers receive products with longer shelf life, directly affecting satisfaction and brand trust [3]. Without integrated FEFO logic tied to batch management and checkout flows, grocery e-commerce platforms face persistent issues such as unnecessary spoilage, financial losses, and weakened customer loyalty.

1.2.2 Absence of Spoilage or Damage Logging

Many grocery e-commerce platforms lack dedicated systems for logging spoiled, damaged, or expired inventory items. Without a spoilage logging mechanism, businesses struggle to maintain real-time stock accuracy, often continuing to list expired or unsellable goods as available inventory. This not only misleads purchasing decisions and reordering processes but also undermines inventory management integrity.

Recent studies emphasize the importance of effective spoilage tracking for perishable goods. Research by Cashflow Inventory [4] highlights that improper handling of perishables leads to significant waste and financial losses, and stresses that robust spoilage logging is essential for maintaining inventory health. Furthermore, a study conducted at H Supermarket in Riau [5] demonstrated that improving inventory management policies for perishables, particularly through spoilage and expiry tracking, directly increased operational efficiency and reduced unnecessary losses. Without systematic spoilage logging integrated into inventory systems, businesses face elevated operational risks, potential food safety concerns, and diminished customer trust.

1.2.3 Lack of Predictive Restocking Mechanisms

Many grocery e-commerce platforms still rely on manual or reactive stock replenishment methods, without the use of predictive analytics to forecast inventory needs. The absence of predictive restocking subjects' firms to periodic build-up and stockouts, both effects that are negative in their effects on profitability, supply-chain efficiency, and client satisfaction levels. Overstocking leads to wasted storage costs and increased perishable spoilage, and stockouts in business losses and branding trust erosion.

Recent research highlights the growing importance of predictive analytics in inventory management. According to a study by DataPilot [6], predictive models using sales history and seasonal trends significantly reduce stockouts and improve inventory turnover rates. By analysing past purchasing behaviours and sales patterns, businesses can forecast inventory requirements more accurately and maintain optimal stock levels. Without integrating predictive restocking tools, grocery e-commerce platforms remain reactive rather than proactive, risking inefficiencies and lost competitive advantage.

1.2.4 Lack of Bulk Inventory Update Support

Managing large quantities of stock manually remains a major challenge for many grocery e-commerce platforms. Without bulk inventory update capabilities, such as CSV uploads or batch processing tools, businesses must update stock item-by-item, which is time-consuming, error-prone, and inefficient—especially for grocery businesses managing hundreds of SKUs with frequent restocks. Manual updating increases the risk of data entry errors, delays stock visibility, and reduces operational efficiency.

Recent industry best practices emphasize the importance of bulk inventory management solutions. Intellect Outsource [7] highlights that supporting bulk product uploads helps businesses manage large catalogues efficiently, ensuring quicker updates and minimising operational bottlenecks. Implementing bulk upload features directly enhances inventory accuracy, update speed, and overall platform scalability. Without it, grocery platforms struggle to maintain real-time stock visibility and adapt to high-volume stock movement needs.

1.3 Motivation and Contribution

The motivation here is the direct observation of the inefficiencies in existing grocery infrastructure, such as surprising stockouts, delayed stock restock notification, and expired perishables being distributed to the customers. These are usually precipitated by the lack of streamlined stock deduction logic, current stock levels, and clarity in perishable material handling. Neither independent nor mom-and-pop groceries enjoy the luxury of costly enterprise systems, as in Walmart or RedMart. This offering provides a central, inventory-centred e-commerce platform with six major features. These include expiry-aware FEFO deduction in a bid to use first the oldest stocks; real-time logging of spoilage/waste in stock accuracy; predictive restock notification in sync with selling trends; price optimisation logic as concerns near-expiry stocks; real-time stock views; and a bulk CSV upload tool in a bid to fast-track stock updates. These are developed in a bid to reduce wastes, increase business transparency, and establish higher customer trustiness — all in a cost-effective and small grocery store business owner-friendly reach.

1.4 Objective

The project involves the development of a web-based Grocery Inventory Management System integrated within an e-commerce website, tailored particularly for small and medium-sized grocery store business retailers. The system is aimed at remedying the gaps in the presently established grocery sites by providing certain inventory attributes that are concerned with perishable stock administration and business productivity. The following are the objectives of the project.:

1. To develop an expiry-aware FEFO (First Expired, First Out) deduction system that prioritizes products nearing expiry during checkout, with the aim of reducing inventory waste and improving turnover efficiency for perishable goods.
2. To integrate a real-time spoilage and damage logging mechanism that enables administrators to accurately record expired, spoiled, or damaged items, thereby enhancing inventory accuracy and supporting reliable stock data management.
3. To design a bulk inventory upload feature using CSV file processing that allows for efficient, large-scale stock entry and batch tracking, minimising manual entry errors and improving operational speed.

4. To implement a predictive restocking algorithm that analyses recent sales trends to generate intelligent restock alerts, reducing the risk of overstocking or stockouts and enabling data-driven inventory planning.

1.5 Project Scope

This project focuses on the development of a web-based grocery e-commerce inventory management system, specifically designed to address key inventory issues faced by small and medium-sized grocery retailers. The system is limited to a single-vendor environment and aims to improve the accuracy, efficiency, and reliability of managing perishable goods online. The primary scope of the system includes implementing an expiry-aware FEFO (First Expired, First Out) deduction mechanism, enabling administrators to log spoiled or damaged items, generating predictive restocking alerts based on recent sales trends, and supporting bulk inventory updates through CSV file uploads. This project will not cover logistics, payment gateway integration, or multi-vendor capabilities, as the core focus remains on enhancing inventory control for perishables in an SME e-commerce context.

1.6 Report Organization

This report is organized into seven chapters, with each chapter presenting a specific part of the project development, implementation, and evaluation process.

Chapter 1: Introduction introduces the background of the project, problem statements, motivation, project objectives, project scope, and report organization. This chapter explains the need for an integrated grocery e-commerce inventory system that can address expiry-aware stock deduction, spoilage logging, predictive restocking, and bulk inventory updates.

Chapter 2: Literature Review reviews existing grocery e-commerce platforms and related inventory management concepts. It compares platforms such as Walmart, Lotus's, RedMart, Village Grocer, and Jaya Grocer to identify existing limitations. This chapter also discusses the theoretical foundation of the proposed restocking algorithm, including baseline inventory models, demand forecasting, perishability constraints, and demand volatility.

Chapter 3: Methodology and System Planning explain the development methodology used in this project. Agile Scrum is adopted to divide the project into multiple sprints across FYP1 and

FYP2. This chapter also describes the system requirements, sprint planning, verification and validation plan, and overall development timeline.

Chapter 4: System Design presents the design of the proposed system. It includes the system architecture, use case diagram, high-level flow diagrams, entity-relationship diagram, and activity diagrams. This chapter explains how customers and administrators interact with the system and how the major modules are structured.

Chapter 5: System Implementation describes the actual implementation of the system. It explains the hardware and software setup, system configuration, backend libraries, and implemented modules. The chapter covers authentication, checkout, bulk CSV upload, FEFO deduction, spoilage logging, predictive restocking, financial reports, and implementation challenges.

Chapter 6: System Evaluation and Discussion present the testing and evaluation of the developed system. It includes functional testing, integration testing, security testing, performance evaluation, system testing evidence, user acceptance validation, objectives evaluation, and limitations of evaluation. This chapter verifies whether the system successfully meets the project objectives.

Chapter 7: Conclusion concludes the project by summarizing the main findings, achievements, contributions, limitations, and future work. It highlights how the proposed system supports better inventory accuracy, waste reduction, and restocking decisions for grocery e-commerce businesses.

1.7 Summary

This chapter put forth the background, motivation, and essential issues that comprise the rationale for a smarter inventory management system designed as a response to the grocery e-commerce industry. It highlighted four fundamental issues common among existing platforms: the lack of expiry-aware FEFO deduction, the lack of logging of damages and spoilages, the lack of predictive restocking, and the lack of efficacy by the absence of bulk support in uploading inventories. Challenged by business inefficiencies and customer dissatisfaction created by gaps, the work aims to design a practical, cheap, and scalable web-based system custom-built specifically to enhance the accuracy of inventories, reduce losses, and broadly inspire the effective management of perishable goods in online groceries.

Chapter 2: Literature Review

This chapter reviewed the previous studies and made comparisons between major grocery e-commerce platforms. The limitations of previous studies were outlined and a proposed solution aimed at resolving the limitations.

2.1 Review of Previous Work

Previous studies have shown that while major grocery e-commerce platforms such as Walmart, Lotus's, RedMart, Village Grocer, and Jaya Grocer have adopted certain inventory management practices, they still face significant limitations.

2.1.1 Walmart

Walmart is a pioneer in real-time inventory tracking and AI-driven restocking [8]. It uses artificial intelligence to analyse customer behaviour and predict demand, ensuring that popular items are never out of stock while minimising excess inventory. According to Redress Compliance [9], Walmart's predictive inventory system reduces operational inefficiencies by implementing real-time inventory tracking, automated replenishment, perishable waste management and price optimisation. Besides, Walmart provides a clear perishable product tagging in several categories and allow product substitutions.

However, these systems are tightly integrated with Walmart's large-scale infrastructure, making them impractical for small businesses. Additionally, Walmart's front-end ordering process lacks expiry-aware FEFO deduction, which can lead to perishables being sold inefficiently. Furthermore, features such as spoilage logging and bulk inventory uploads are not available through their public platform. These gaps present an opportunity to create a simplified, cost-effective web-based system tailored to the operational scale and needs of SME grocery vendors.

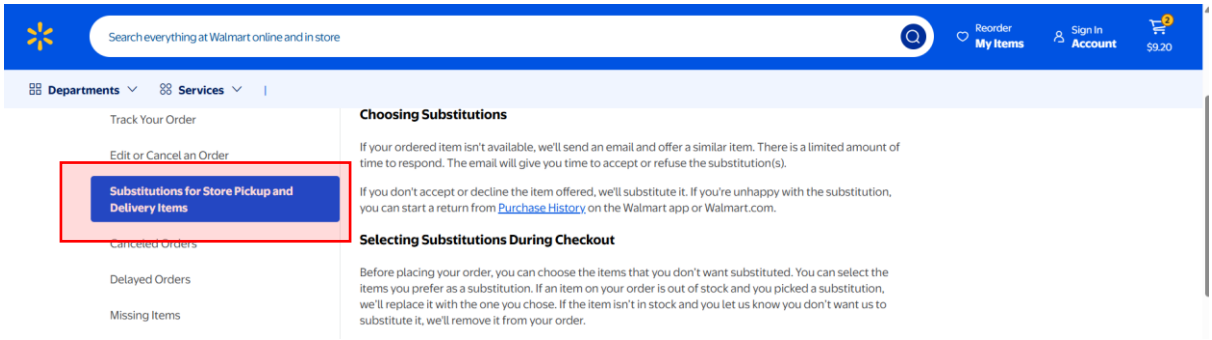


Figure 2.1.1.1 Product Substitution (Walmart)

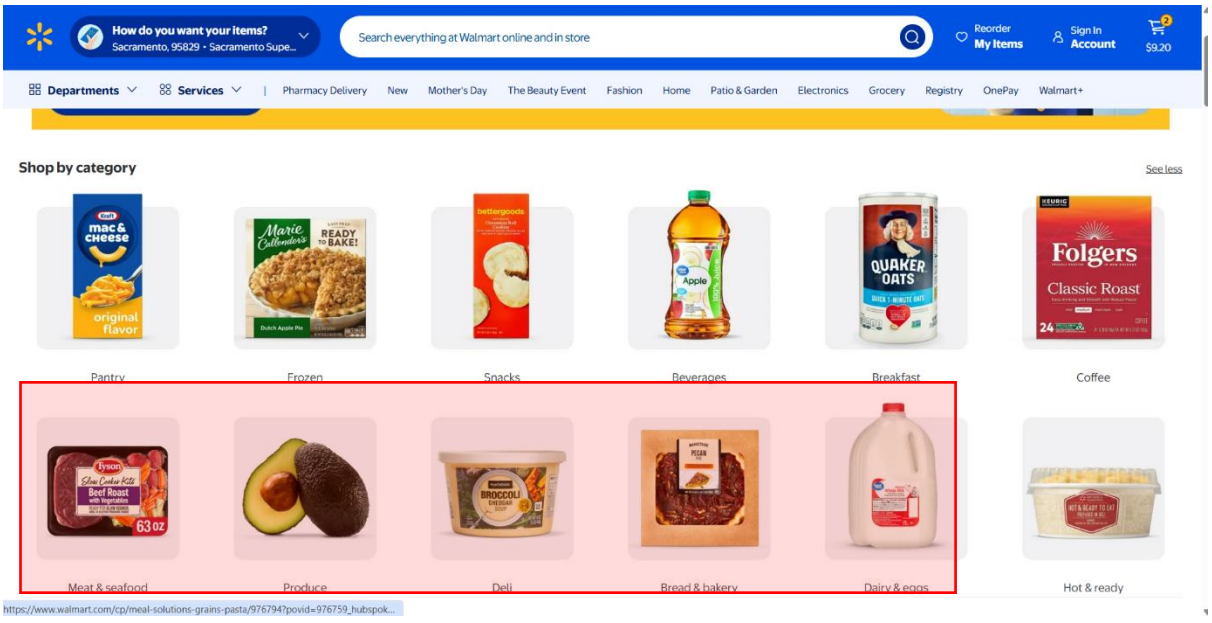


Figure 2.1.1.2 Perishable Product Tagging (Walmart)

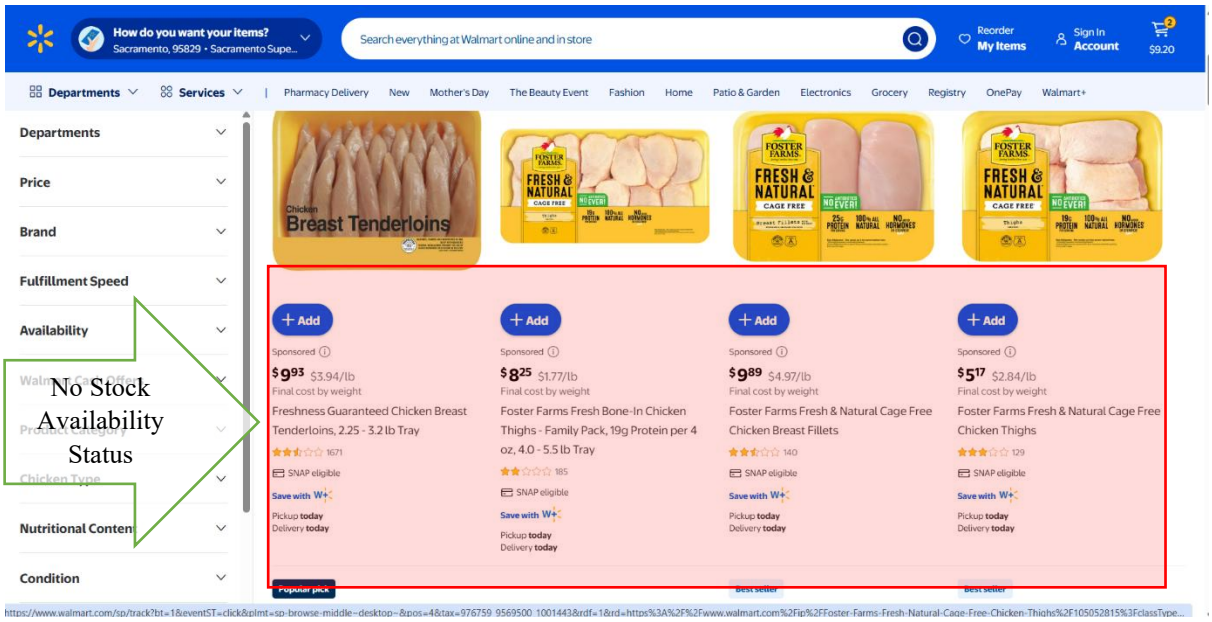


Figure 2.1.1.3 No Stock Availability Status (Walmart)

2.1.2 Lotus's

Tesco, branded as Lotus's [10] in Malaysia, provides a user-friendly online shopping experience with basic product categorisation for perishable items such as "Fresh Produce" and "Meat and Poultry". These tags help customers differentiate fresh and temperature-sensitive products but are primarily front-end labels with no evidence of backend inventory automation. The website does not show real-time stock availability and expiry dates during product browsing or checkout.

Additionally, there is no visible implementation of FEFO-based stock deduction or predictive restocking logic for perishables. Product substitution is not offered explicitly, nor is there a system for logging spoilage or managing inventory uploads in bulk. These missing features indicate that while Lotus's offers standard e-commerce functionality, its current system lacks the robust inventory controls needed to manage perishables efficiently. For smaller businesses seeking affordable and flexible alternatives, a web-based system with expiry and restocking intelligence would be more practical and scalable.

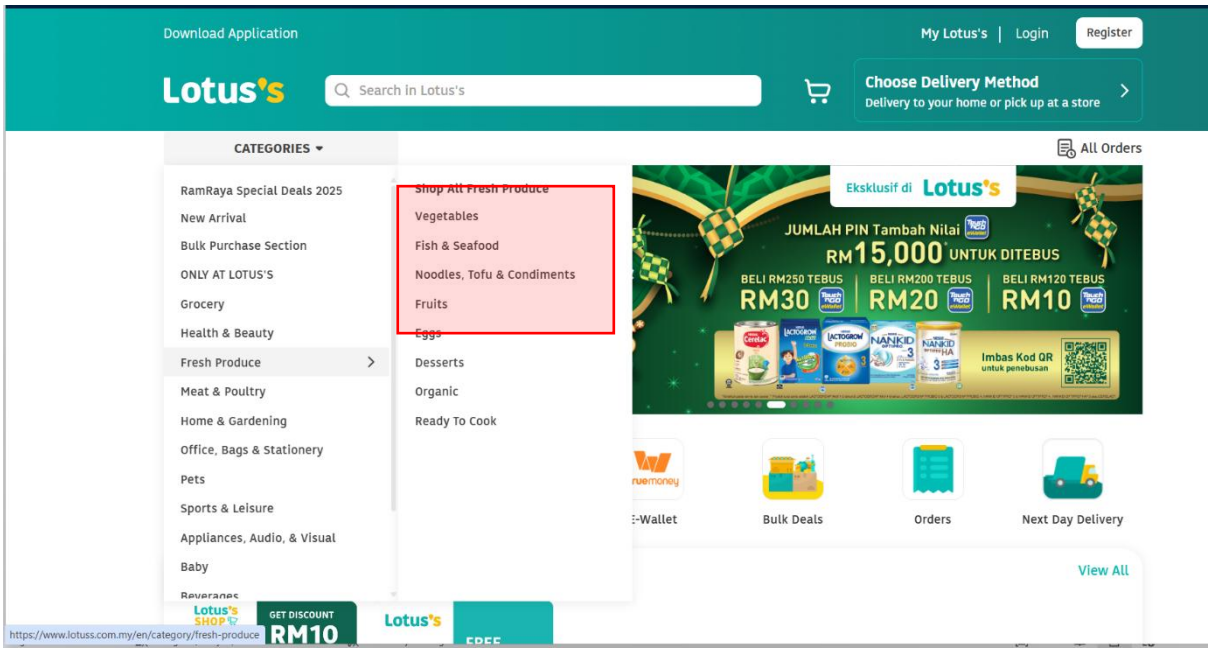


Figure 2.1.2.1 Perishable Product Tagging (Lotus)

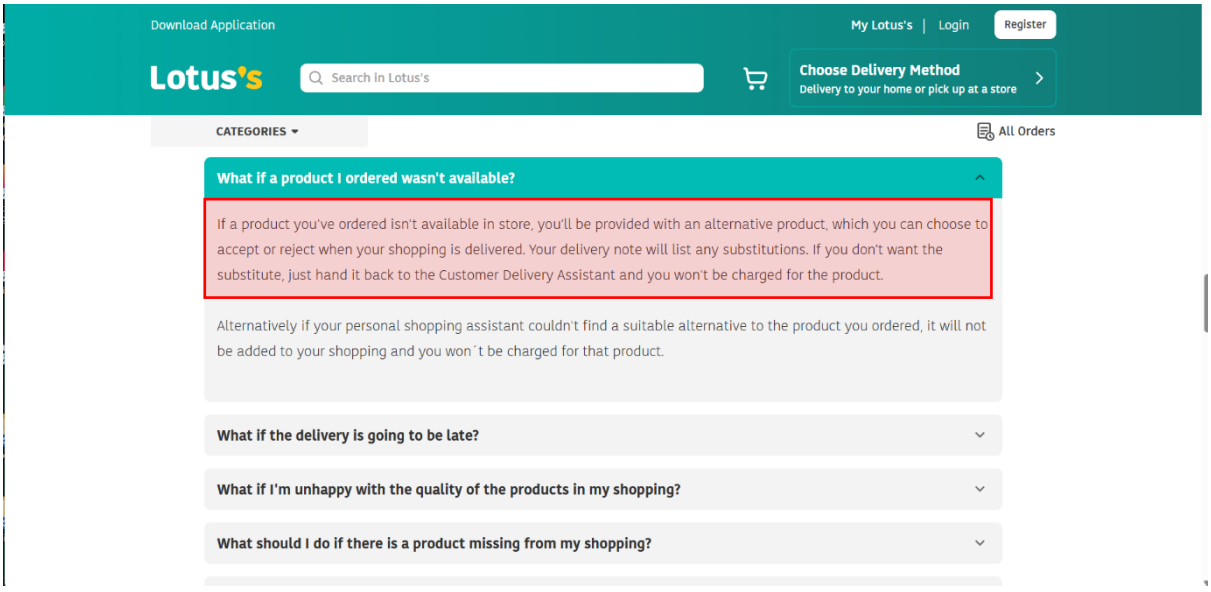


Figure 2.1.2.2 Availability of Product Substitution (Lotus)

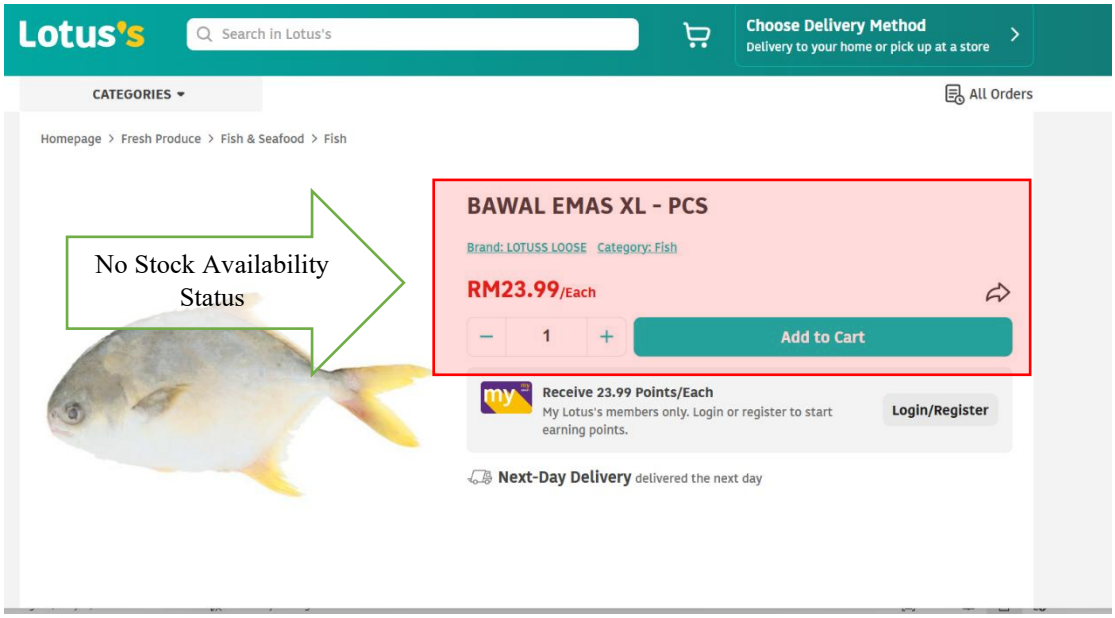


Figure 2.1.2.3 No Stock Availability Status (Lotus)

2.1.3 RedMart by Lazada

RedMart [11], an online-first grocery platform in Singapore, is known for its backend expiry and batch tracking. It leverages centralized warehousing to enable detailed inventory control and food recall capabilities. The Singapore Food Agency [12] confirms RedMart's compliance with food safety through batch-level tracking and recall procedures. It provides clear perishable product tagging and allows product substitution. Besides, RedMart allow users to track the stock availability by displaying “Only XX stock left”.

Despite this, the system does not extend expiry control to the customer experience. Orders are not fulfiled based on product expiry, and no expiry date visibility is offered at checkout. Additionally, RedMart does not support predictive restocking logic based on sales trends or data analysis. There is also no system in place for logging spoilage or expired stock, which limits inventory accuracy. Bulk inventory upload functionality is not visible, and the platform lacks automated pricing mechanisms for near-expiry goods. This disconnect can lead to spoilage if older stock isn’t prioritized—an area this project seeks to address with customer-facing FEFO deduction logic.

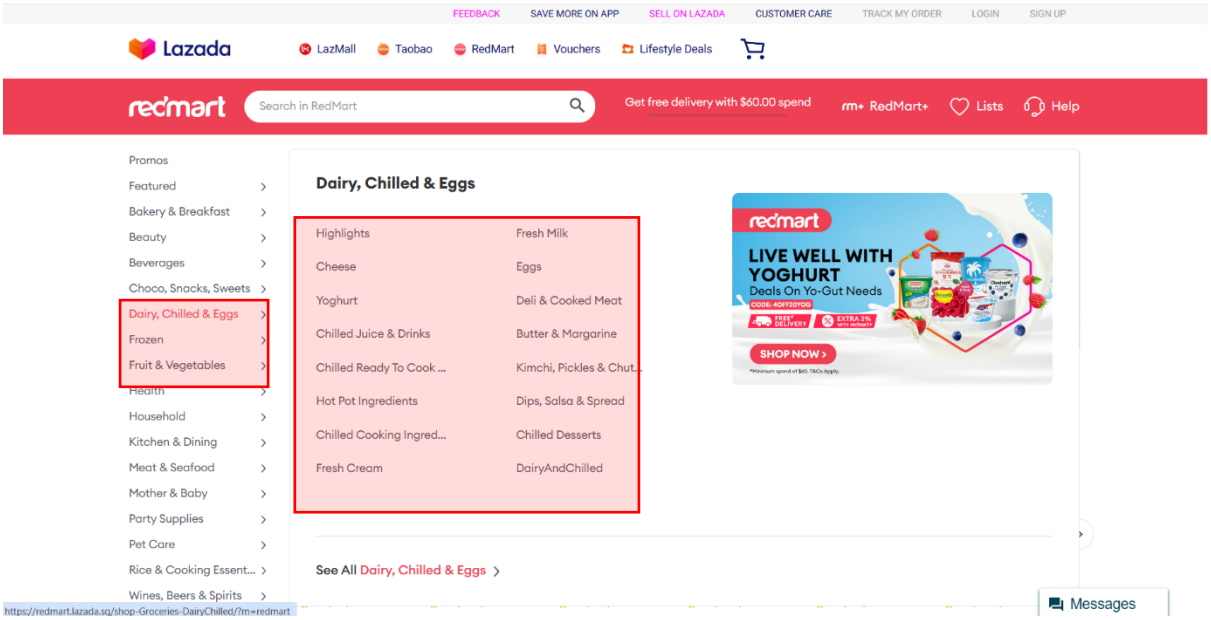


Figure 2.1.3.1 Perishable Product Tagging (RedMart)

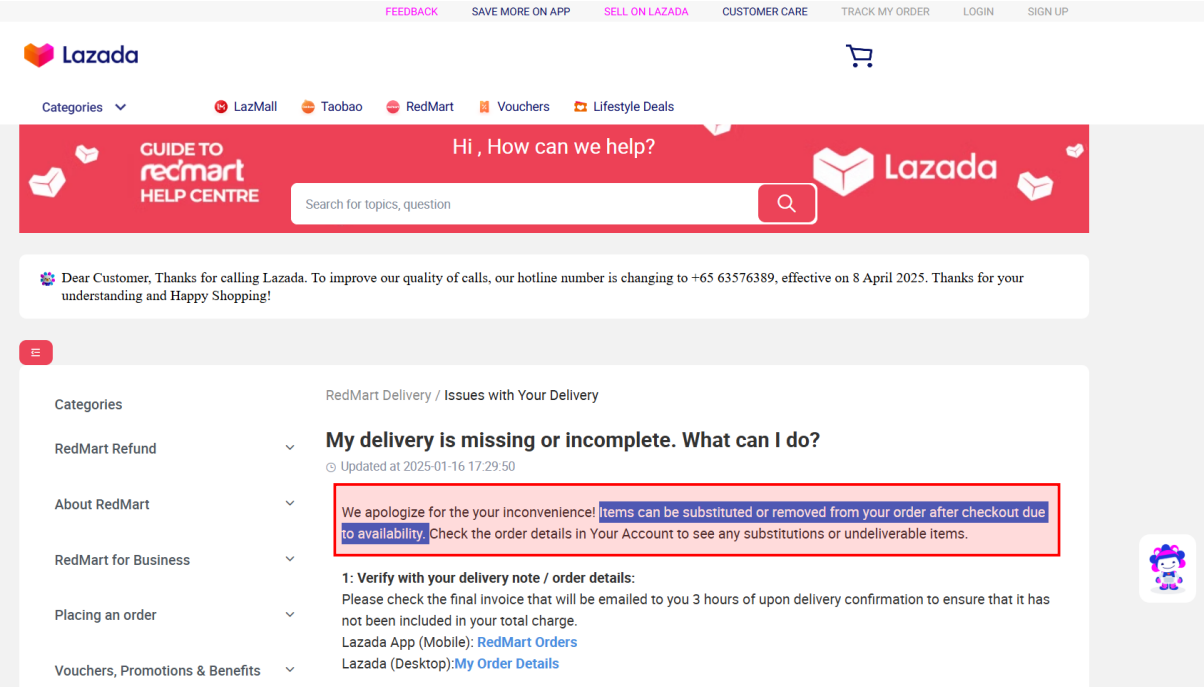


Figure 2.1.3.2 Product Substitution (RedMart)

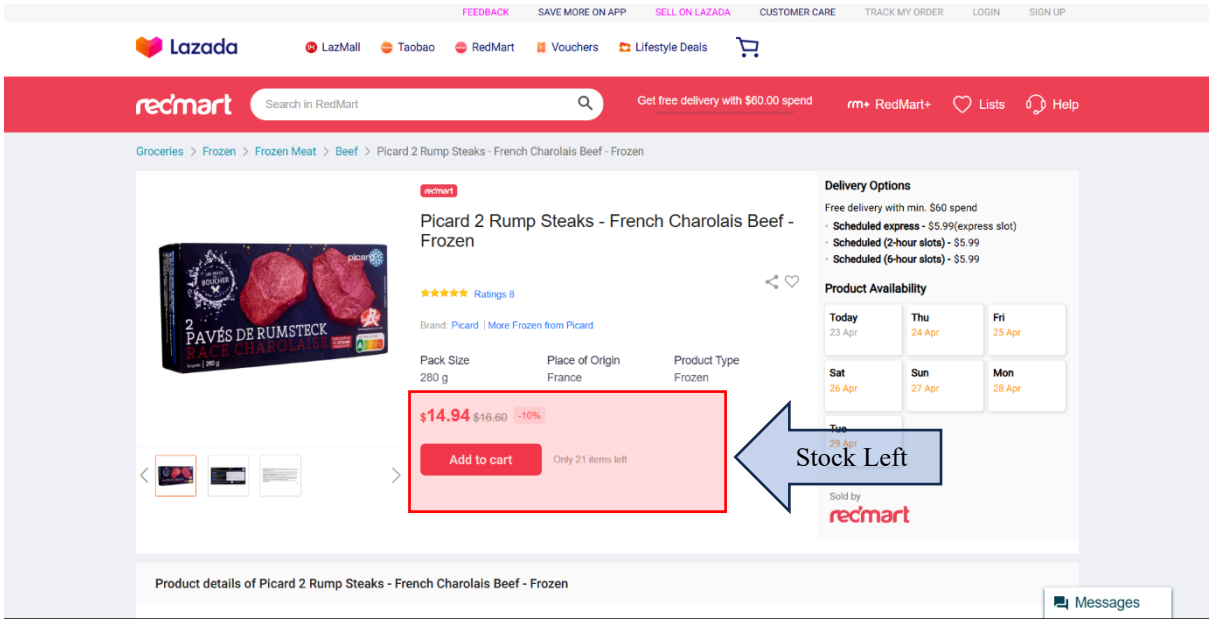


Figure 2.1.3.3 Real-Time Inventory Tracking (RedMart)

2.1.4 Village Grocer (Bites Shop)

Village Grocer [13] has made notable progress in digitizing its operations by integrating online ordering with in-store fulfilment. The company uses a picker system that allows staff to accurately process online orders while maintaining inventory visibility at the branch level. It provides clear perishable product tagging and allows product substitution. Although the exact stock quantity is not displayed, product availability is clearly indicated.

However, the inventory system remains manual for most backend processes, including stock adjustments and spoilage logging. Village Grocer lacks an expiry-aware FEFO system and has no predictive restocking or dynamic discounting for near-expiry products. The system also lacks a bulk upload tool for entering large quantities of inventory efficiently. Similarly, there is no automated pricing logic to offer discounts for products approaching their expiry date, which could help reduce waste and recover revenue. These gaps limit the platform's efficiency in reducing waste and optimizing shelf life—key issues this project aims to resolve through intelligent backend automation.

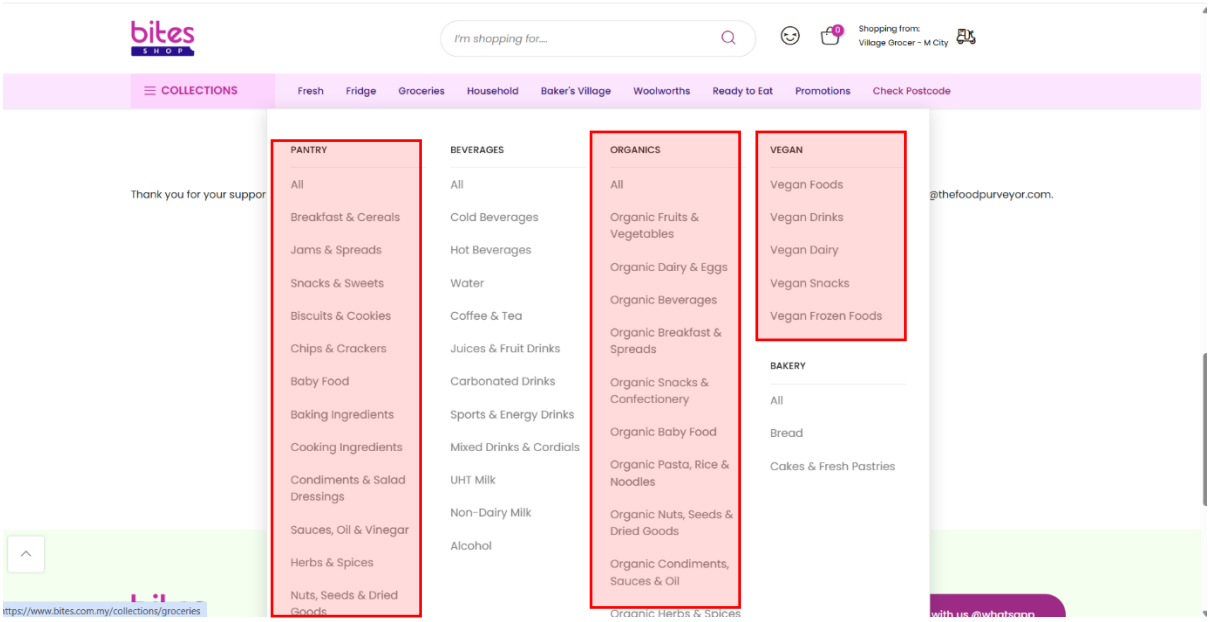


Figure 2.1.4.1 Perishable Product Tagging (Village Grocer)

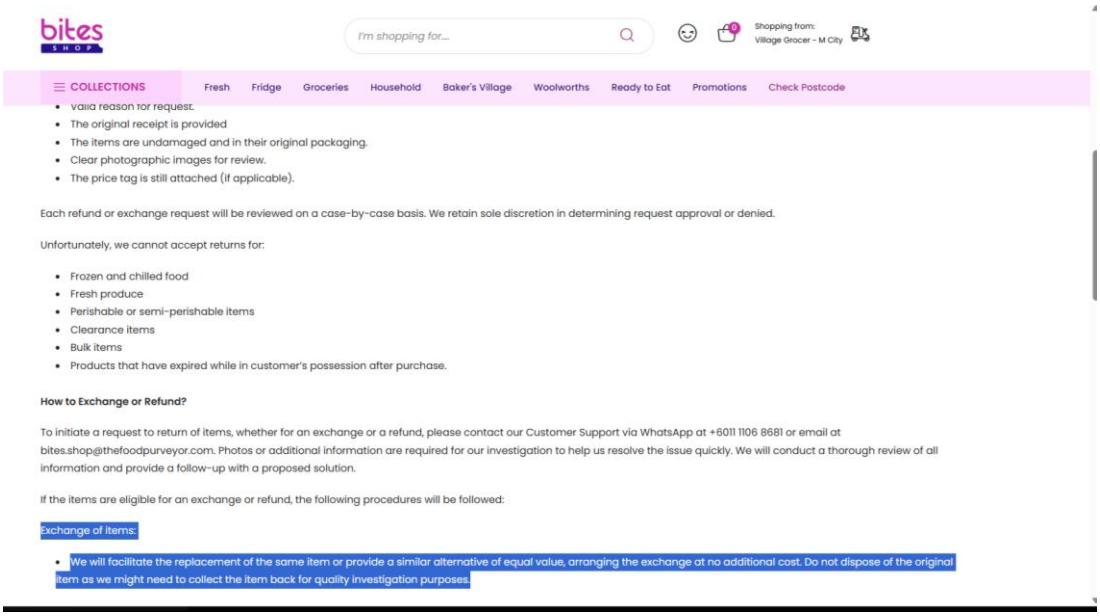


Figure 2.1.4.2 Product Substitution (Village Grocer)

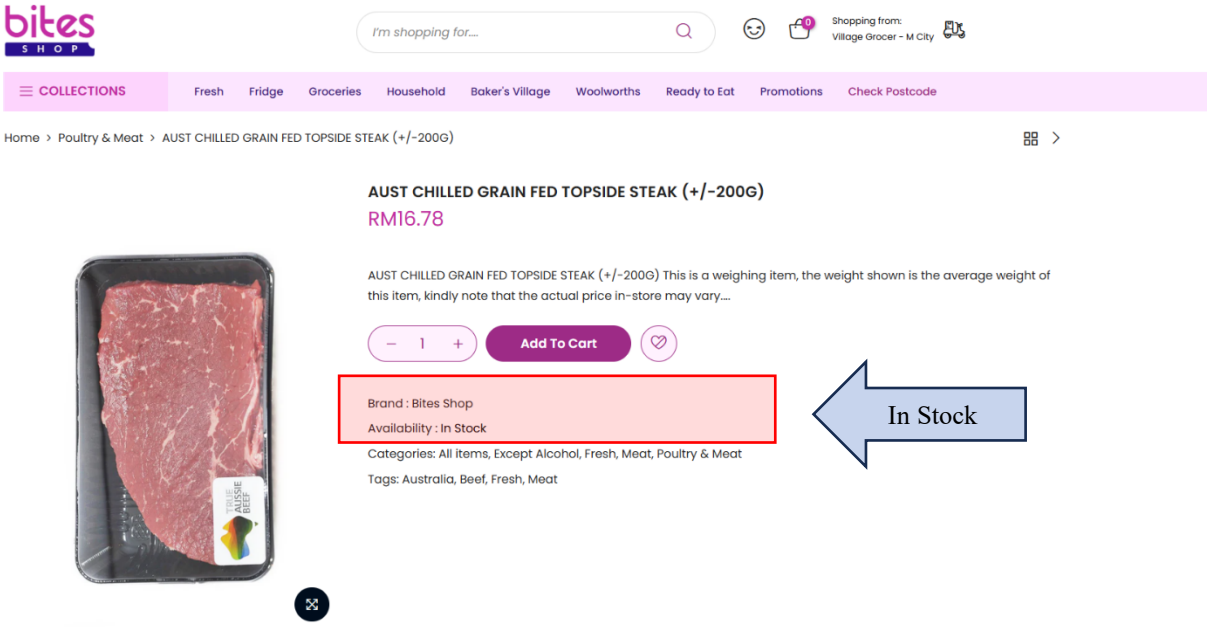


Figure 2.1.4.3 Stock Availability (Village Grocer)

2.1.5 Jaya Grocer

Jaya Grocer has adopted a digital picking app integrated with its in-store inventory. The automation syncing the product catalogue and inventory systems with GrabMart helps ensure that staff don't have to manually update the online inventory as the day goes, to keep Grab Mart's listing up to date in real-time and avoid human errors and delays [15]. Grab highlights that this tool improves order fulfilment accuracy and reduces complaints during online-to-offline order transitions. This helps bridge the digital gap for traditional retailers moving into e-commerce. Besides, Jaya Grocer provides a clear perishable product tagging in several categories.

However, the system lacks automated tools for handling spoilage and predictive pricing. Jaya Grocer does not include product substitution and provides stock availability too. Additionally, Jaya Grocer's front-end ordering process lacks expiry-aware FEFO deduction, which can lead to perishables being sold inefficiently. Moreover, the system does not include any bulk inventory upload support, which would ease the management of large restocking events. It also lacks price optimisation functionality, so near-expiry items are not automatically discounted or prioritized for sale.

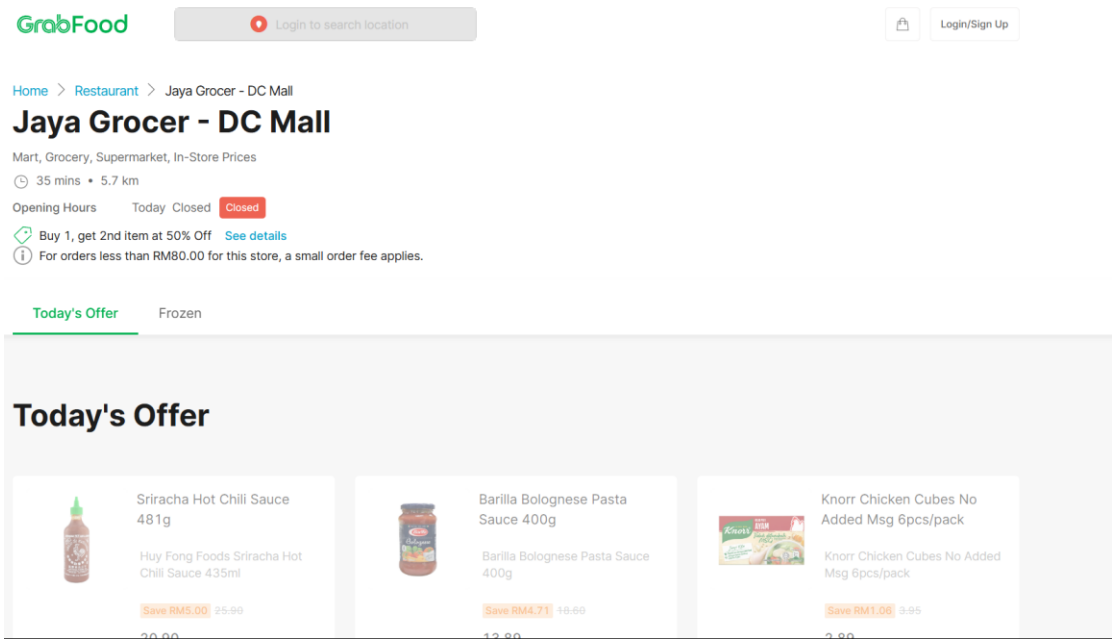


Figure 2.1.5.1 Jaya Grocer Integrated in Grab Mart

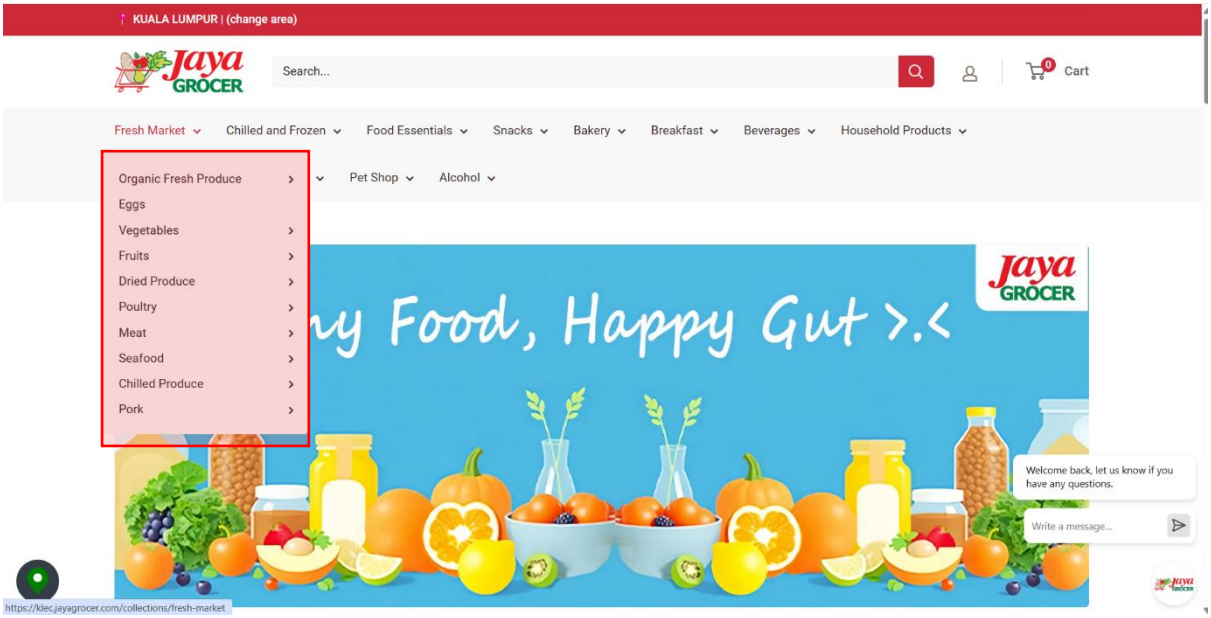


Figure 2.1.5.2 Perishable Product Labelling (Jaya Grocer)

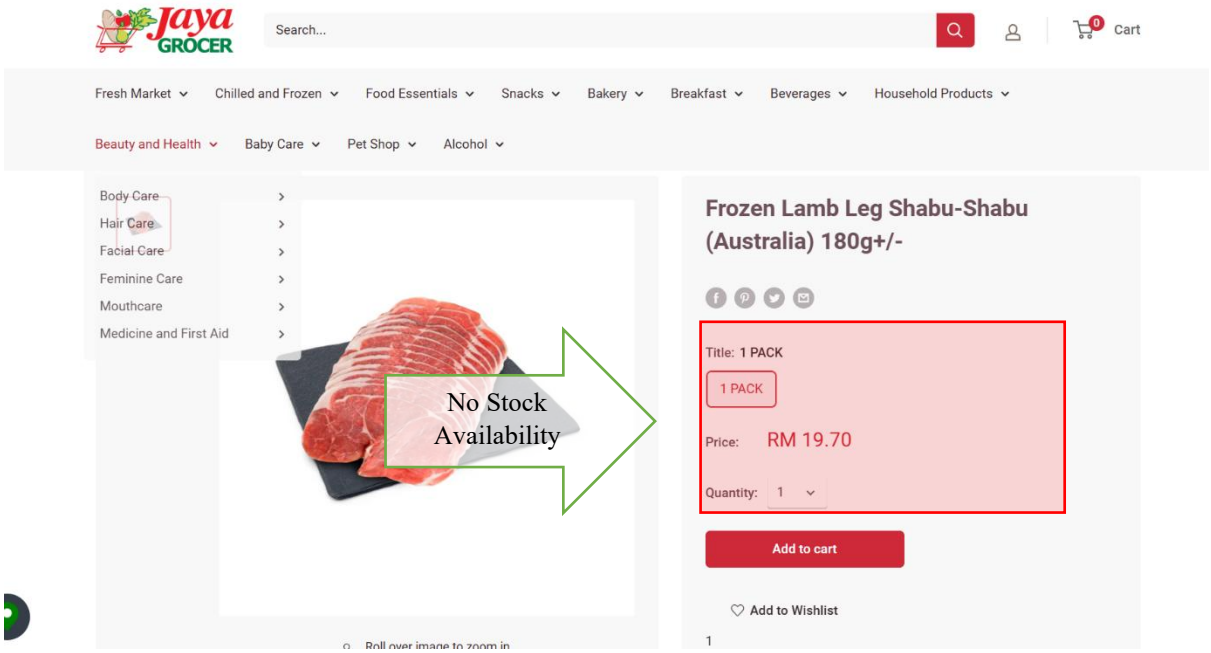


Figure 2.1.5.3 No Stock Availability Status (Jaya Grocer)

Table 2.1 Comparison between Reviewed Platforms

Feature	Walmart	Lotus's	RedMart	Village Grocer	Jaya Grocer	My Project
Real-Time Inventory Tracking	❌	❌	✅	✅	❌	✅
Expiry-Aware FEFO Deduction	❌	❌	❌	❌	❌	✅
Spoilage/Waste Logging	❌	❌	❌	❌	❌	✅
Predictive Restocking (AI/basic)	✅ (AI)	❌	❌	❌	❌	✅
Product Substitution	✅	✅	✅	✅	❌	✅
Bulk Inventory Upload	❌	❌	❌	❌	❌	✅
Perishable Product Tagging	✅	✅	✅	✅	✅	✅
Price Optimisation Logic	✅ (AI)	❌	❌	❌	❌	✅

2.2 Limitations of Previous Studies

Although main e-commerce grocery platforms are well-established stock systems, they are typically big, infrastructure-dependent systems, which are too costly and impractical to be implemented among a group of small companies. Primary limitations are absence of expiry-aware FEFO deduction, thereby bringing unwarranted product wastage; absence of live logging of spoilage or waste, thereby providing inaccurate stock positions; and absence of dynamic price logic or restocking, thereby generating higher costs of operation. Also, most platforms don't support bulk upload, thereby impairing restocking and batch entry procedures. These limitations clearly indicate a need for a pragmatic, web-based, SME-friendly e-commerce grocery stock management solution.

2.3 Proposed Solutions

To fill the above gaps, this project introduces a collection of practical features tailored exclusively to small and medium-sized grocery enterprises. The system shall apply expiry-aware FEFO deduction, ensuring that products nearest expiry are checked out first in a bid to lower wastage. The system shall also apply spoilage and waste logging, whereby administrators shall be able to mark items as expired or damaged, automatically adjusting stock levels in response. Predictive restocking notifications shall be made based on basic sales trend analysis within 7–14 days, enabling vendors to keep stocks at the optimal level. The system shall also incorporate price optimisation logic, whereby discounts shall automatically be applied on near-expiry goods, thus securing potential revenues while incurring minimum losses. Bulk stocks updates shall be accommodated through CSV uploading in a bid to simplify stocks entry during massive restocking occasions. Lastly, all changes in stocks either through customer orders or through administrators shall be updated in real-time, ensuring that stocks are displayed in a consistent and accurate basis throughout the platform.

2.4 Algorithm Evaluation and Theoretical Framework

The current study involves a comparative study to evaluate inventory management techniques on perishable grocery items where a baseline inventory algorithm is compared to a suggested adaptive predictive algorithm. The baseline algorithm is a standard reorder-based inventory model whereas the proposed algorithm builds on the base algorithm by adding demand momentum, demand variability, and perishability requirements.

This comparison aims to answer the question: can an adaptive inventory algorithm be used to give more appropriate restocking advice to grocery e-commerce systems? Groceries can be perishable goods, so inventory decisions need to consider not only demand and stock levels, but also shelf life, risk of spoilage and demand variability. The management of perishable inventory is a special case since the products have a short shelf life, poor matching of supply and demand can result in food waste, stockout risk and low product value [18], [22].

Hence, the study contrasts the baseline algorithm with the proposed adaptive algorithm in terms of their capability to assist in more realistic restocking decisions, minimise overstocking, minimise stockout risk, and enhance inventory decision-making to perishable grocery products.

2.4.1 Baseline Algorithm (Standard Industry Practice)

The baseline model incorporates a continuous review reorder point approach, which is a popular inventory control approach. This model involves monitoring of inventory and stock is replenished when the inventory level attains a certain reorder point. There is a lot of inventory management literature on reorder-based models of inventory because these models are considered a viable method of determining when and how much inventory is to be replenished [17].

In the baseline model, a 30-day Simple Moving Average (SMA) is used to forecast demand. The SMA uses historical sales data of the last 30 days to compute the average daily demand. All observations are given equal weight and hence the method is easy to comprehend. Nevertheless, since all observations are equally handled SMA can be slow to react to abrupt changes in demand. Moving averages are also effective in averaging out short-term variations in time-series data, although this process of averaging can diminish responsiveness when demand is suddenly changing [19].

The fixed service-level Z-score is used to calculate the safety stock. As an example, the approximate service level of 95 percent is generally related to a Z-score of 1.65. The baseline model takes the average demand, variability of demand and lead time into consideration without explicitly considering the product shelf life, spoilage risk, and short-term demand momentum.

Consequently, despite the simplicity and ease of calculation, the baseline model might not be appropriate with perishable grocery stock. As an illustration, it can indicate overstocking of products that have a short shelf life, or it can be very slow when there is a sudden rise in demand. This drawback gives the ground to come up with a more adaptive predictive restocking algorithm.

2.4.2 Proposed Predictive Algorithm (Conceptual Model)

This study presents an adaptive heuristic inventory model to overcome the shortcomings of the baseline model. The proposed model instead places the same model using complex machine learning algorithms, but it adds three more elements to the conventional inventory formula,

namely demand momentum adjustment, perishability constraint, and volatility-based safety stock adjustment.

The rationale behind the choice of this approach is that it is computationally efficient, interpretable and applicable to small and medium-sized grocery e-commerce businesses. Because the target users are SME grocery retailers, the model must be capable of producing valuable restocking recommendations without the need to have large datasets, sophisticated machine learning infrastructure, or a high cost of computation.

The proposed model will enhance the decision-making process in restocking by considering new demand behaviour, reducing unnecessary stock in case of perishable goods, and increasing safety stock according to the uncertainty in demand. It becomes crucial especially in the case of grocery retail where shelf life, demand, service level, customer-picking behaviour, and replenishment constraints determine the level of inventory policy effectiveness and food waste minimization [22], [24].

Moreover, inventory-forecasting studies also note that forecasting cannot be viewed as a stand-alone process since forecast results have a direct impact on replenishment decisions and inventory control performance [20]. Thus, the algorithm suggested can be regarded as a direct linkage between the demand analysis and the logic of restocking recommendations.

2.4.3 Momentum and Demand Adjustment

The initial improvement is a demand momentum adjustment to represent the short-term fluctuations of the customer demand. The system compares short-term performance in sales against the longer-term average performance. When the recent demand exceeds the long-term average, the adjusted demand estimate goes up. When the recent demand is less than average long-term demand, then the adjusted estimate of demand is less.

This enables the system to react faster to the changing demand patterns than a straightforward moving average. Exponential smoothing as a forecasting method is premised on the notion that time like responses can be enhanced using observations that are made in the near past [19]. This helps to rationalize the adjustment of the projected demand figure by recent demand behaviour.

To illustrate the point, when a product has been subjected to such a sharp rise in sales over the last few days, the suggested algorithm can suggest a larger number of units to be restocked sooner than the baseline algorithm. This can be applied in grocery e-commerce where the demand may fluctuate because of promotions, seasonal demand, customer preferences, or product popularity.

The momentum component thus enhances responsiveness of the inventory model, that is, being interpretable. This renders it appropriate with an SME oriented system where the algorithm must be comprehensible and feasible to administrators.

2.4.4 Perishability Constraint and Decay Factor

The second improvement adds a perishability constraint to consider the shelf life and risk of spoilage of products. Basic reorder models tend to concentrate on the demand in volume and the quantity of inventory. But in case of grocery products, also it must be taken into consideration whether the suggested restock level can be practically sold before the product goes out of date.

The suggested model uses a constraint of maximum sellable quantity according to the adjusted demand and useful shelf life. This makes sure that the system does not suggest a larger reorder quantity than the estimated sales volume within the effective shelf life of the product. As an illustration, when a product has a low shelf life, the system is not supposed to indicate a large order quantity in case the base model indicates a large restock.

A decay factor is also used to represent the risk of spoilage, thus lowering the raw order quantity of those products with a high risk of spoilage. This method is in line with the theory of perishable inventory and recent perishable reordering research whereby shelf life, target stock level, demand variability, and First-Expired-First-Out (FEFO) issuing policies are significant in inventory decisions [18], [23].

The proposed algorithm can be made more applicable to perishable grocery products by including the shelf life and spoilage risk in the calculation of the restocking. This will minimise waste that is unnecessary and still maintain the availability of stock.

2.4.5 Dynamic Safety Stock via Volatility

The third improvement deals with uncertainty in demand via dynamic adjustment of safety stocks. The offered algorithm computes the coefficient of variation (CV) when it does not utilise a fixed level of safety stock, but the volatility of the demand of the products.

Coefficient of variation is determined by dividing the standard deviation of demand with the average demand. The greater the CV, the more unstable or unpredictable is the demand, the less stable is the demand the greater is the CV. XYZ inventory analysis categorizes the products based on the variability of demand where the mean consumption, standard deviation, and coefficient of variation are employed to identify the variability category [21].

According to this measure of volatility, the system dynamically changes the level of stock safety. The products that have stable demand have fewer safety buffers, whereas highly variable demand products have more safety buffers to reduce risks of stockouts. This enhances the efficiency of inventory by not stocking too much safety inventory on predictable items but offers extra buffer to products with unpredictable demand.

Dynamic safety stock is thus applicable in the e-commerce of groceries since various products can possess different demand patterns. An illustration of this is that the demand of staple goods might remain relatively stable and suddenly fluctuate in seasonal or promotional goods.

2.4.6 Summary of Algorithm Improvements

Altogether, the suggested predictive algorithm enhances the baseline reorder-based model by adding demand momentum, perishability and demand volatility. These improvements enable the system to shift its inventory model to a more adaptive decision-support model, rather than a reactive model.

The baseline algorithm offers an easy and handy comparison point due to its averages in demand and constant safety stock. But it also fails to completely tackle the issue of perishable groceries. It does not explicitly address either the question of increasing demand, of whether the product is sellable before it goes out of date, or whether the sales behaviour of the product is stable or volatile.

The proposed algorithm attempts to overcome these shortcomings by modifying demand through short-term momentum, fixing recommended restock levels through usable shelf life

and altering safety stock through demand volatility. Such advancements render the restocking process more applicable to e-commerce systems of grocery e-commerce that deal with perishable items.

Thus, it is likely that the suggested algorithm will be more useful in terms of its ability to offer more practical restocking proposals to grocery e-commerce systems. The evaluation chapter will also confirm its effectiveness by implementing it and comparing it to the baseline algorithm.

2.5 Summary

This chapter reviewed existing grocery e-commerce platforms, including Walmart, Lotus's, RedMart, Village Grocer, and Jaya Grocer, to identify current inventory management limitations. The comparison showed that although these platforms provide basic online grocery functions, many still lack expiry-aware FEFO deduction, spoilage logging, predictive restocking, and bulk inventory update support.

The chapter also discussed the theoretical foundation of the proposed restocking mechanism by comparing a baseline reorder-based model with an adaptive predictive algorithm. The review showed that traditional inventory models are useful but may not fully address the challenges of perishable grocery products.

Overall, this chapter supports the need for an integrated grocery e-commerce inventory system that can improve stock accuracy, reduce waste, and support better restocking decisions.

Chapter 3: Methodology & System Planning

This chapter describes the methodology and design approach adopted for the development of the Integrated E-Commerce Inventory System. The project applies the Agile Scrum framework to ensure iterative and incremental delivery of system features, enabling continuous feedback, early detection of issues, and adaptability to changes. The chapter begins with the design specifications, tools to use, and verification plan that guided the project. It then explains the Agile development methodology in detail, including Scrum roles, artifacts, and events, followed by the sprint planning and timeline used to manage the twelve sprints across FYP1 and FYP2. The chapter also presents the system design, covering high-level flow diagrams, architecture, activity diagrams, and database models. Finally, the chapter concludes with a reflection on implementation issues and retrospectives, summarizing lessons learned from the first six sprints in FYP1.

3.1 Agile Development Methodology (Scrum)

An iterative agile methodology called Scrum methodology will be used for this whole project to meet the needs of structures and flexibility within this project. KVT Technology [16] published an article highlighting the transition from the traditional Waterfall methodology to Agile and Scrum frameworks, providing multiple case studies that demonstrate the advantages of adopting Agile in software development. Work Breakdown Structure (WBS) is established to decompose the system into manageable tasks and modules, providing a smooth and flexible development environment. Therefore, the methodology selected divides the development of the inventory system into multiple sprints. Each sprint lasts approximately 1-2 weeks and delivers s working feature or module related to inventory management. Each sprint includes planning, development, testing, and review activities. This ensures that critical inventory features are delivered incrementally, allowing for continuous validation, improvement, and alignment with user expectations. Figure 4.2 shows a cycle within a sprint.

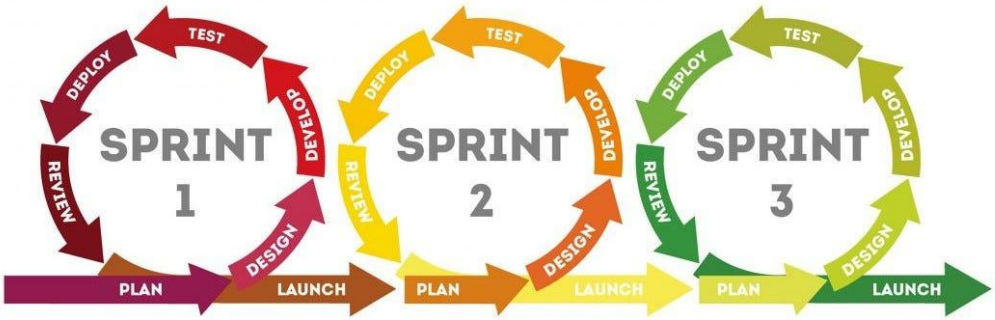


Figure 3.1 Sprint Cycle Example

3.1.1 Scrum Roles

In Scrum, three primary roles are set to ensure the smooth development and management of the project. The roles are Product Owner, Scrum Master and Development Team. In this project, these roles were adopted to the context of a single developer.

Product Owner: The student developer, responsible for defining and prioritizing the backlog.

Scrum Master: The student developer, ensuring Scrum principles are followed.

Development Team: The student developer, performing coding, testing, and integration.

3.1.2 Scrum Artifact

A comprehensive list of desired features expressed as user stories. The priority is defined for each item to ensure the essential functionality can be developed in earlier stage of development.

Table 3.1.1 Product Backlog

ID	User Story	Status	Priority
PB1	As a user, I want to register/login for secure access.	✅ Completed	High
PB2	As a customer, I want to checkout products to place orders.	✅ Completed	High
PB3	As an admin, I want to upload inventory via CSV for bulk management.	✅ Completed	Medium
PB4	As an admin, I want stock deducted using FEFO logic to minimise expiry risk.	✅ Completed	High
PB5	As an admin, I want to log spoilage and damage for stock accuracy.	✅ Completed	Medium
PB6	As an admin, I want predictive restocking alerts to plan inventory proactively.	✅ Completed	Low

3.1.3 Scrum Events

Scrum organizes development activities through a series of structured events which are Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective. Each events have its own activity flow and deliverables. Below is the detailed description of the Scrum Events:

- Sprint Planning: At the beginning of each sprint, backlog items were selected based on its feasibility and priority. Sprint 1 focused on user authentication modules that include login and registration functionality. Sprint 2 delivered the checkout module for customers use and bulk upload for administrators use. The remaining features such as FEFO duction, spoilage logging, and restocking alerts mechanisms are postponed.
- Daily Scrum: The daily scrums were adapted into self-managed progress tracking as this project was developed individually. The daily progress such as task completed, encountered issues and new ideas were logged in notepad for next day reference
- Sprint Review: At the end of each sprint, the completed increments were tested and presented to supervisor. Reviews and feedback from supervisors were considered as the enhancements of each sprint.

- Sprint Retrospective: Reflection sessions were conducted to evaluate successes and challenges. For instance, Sprint 2 identified CSV validation failures and cross-origin errors during integration. Solutions were applied in the same sprint, including row-level validation for CSV files and enabling CORS configuration, demonstrating Scrum's ability to support continuous improvement.

3.2 System Requirements

The system must achieve the following functional requirements and non-functional requirements

3.2.1 Functional Requirements

Based on the product backlog, the system is required to

i. Provide secure authentication
ii. Allow customers to browse and purchase goods
iii. Enable administrators to manage stock both manually and via bulk uploads
iv. Allow administrators to deduct inventory using FEFO logic
v. Allow administrators to log the spoilage and damage
vi. Provides predictive restocking alerts for administrators to update inventory earlier

3.2.2 Non-Functional Requirements

1. Security: All passwords must be hashed, and APIs secured with JWT tokens.
2. Performance: Checkout operations should complete within one second under normal load.
3. Scalability: The system must support a growing product catalogue of at least 500 items without performance degradation.
4. Maintainability: The architecture must follow layered design principles to allow future feature extensions.

3.3 Sprint Planning and Timeline

3.3.1 Sprint Planning

The product backlog has been divided into 12 sprints that lasts 2 weeks for each. Sprint 1-6 were conducted and completed while sprint 7-12 are marked as pending sprint that and will be conducted in future development.

Table 3.3.1 Sprint Planning Table

Phase	Sprint	Duration	Backlog Items (PB)	Deliverables (Increment)	Status
FYP1	S1	Week 1–2	PB1 (Identity – Login)	Basic login functionality, authentication form, database schema setup	Completed
	S2	Week 3–4	PB1 (Identity – Registration)	Registration module with role assignment (customer/admin), profile management	Completed
	S3	Week 5–6	PB2 (Shopping – Cart Setup)	Product listing, cart management, add/remove items, cart validation	Completed
	S4	Week 7–8	PB2 (Shopping – Checkout)	Basic checkout flow, order confirmation, payment placeholder	Completed
	S5	Week 9–10	PB3 (Onboarding – Bulk Upload Prototype)	CSV parser prototype integrated with product database	Completed
	S6	Week 11–12	PB3 (Onboarding – Validation)	Enhanced bulk upload with validation & error reporting	Completed
FYP2	S7	Week 13–14	PB4 (Expiry Control – FEFO)	FEFO deduction algorithm implementation	Completed
	S8	Week 15–16	PB4 (Expiry Control – Optimisation)	Improved FEFO logic with edge case handling	Completed
	S9	Week 17–18	PB5 (Audit Control – Tracking)	Spoilage tracking system	Completed

	S10	Week 19–20	PB5 (Audit Control – Logs)	Inventory movement logs and audit trail	Completed
	S11	Week 21–22	PB6 (Forecast – Model Design)	Predictive restocking model (baseline + adaptive)	Completed
	S12	Week 23–24	PB6 (Forecast – Analytics)	Forecast dashboard, analytics visualization, evaluation metrics	Completed

3.3.2 Timeline

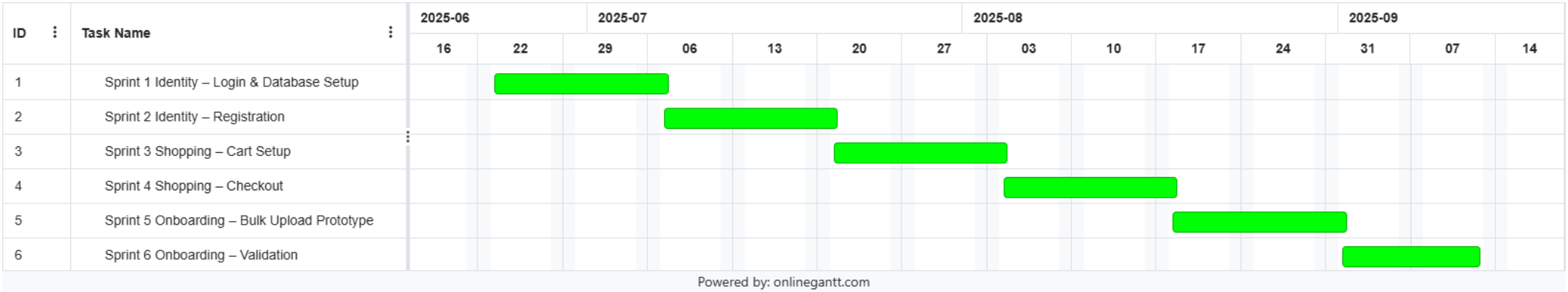


Figure 3.3.2.1 Gannt Chart for FYP 1 Task list

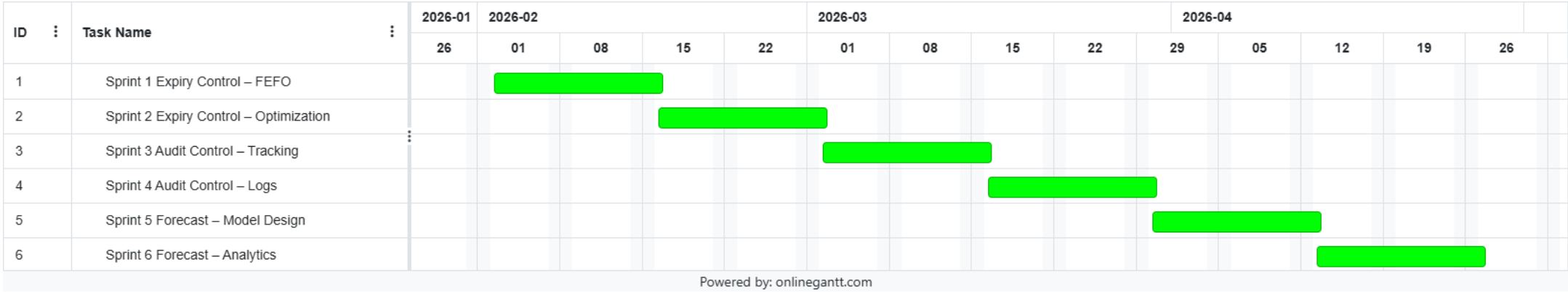


Figure 3.3.2.2 Gannt Chart for FYP 2 Task list

3.4 Verification and Validation of System Modules

Verification activities will be conducted to ensure that the developed modules meet both functional and non-functional requirements. Testing will be carried out incrementally at the end of each sprint, following Agile Scrum practices, with black-box, white-box, and integration testing approaches applied as appropriate. The following outlines the verification plan for each major module.

3.4.1 Authentication

- Inputs to Test: Valid credentials, invalid credentials, and duplicate registration attempts.
- Expected Outputs: Successful login with secure JWT token issuance, rejection of invalid login attempts, and enforcement of unique user registration constraints.
- Verification Method: Unit testing is conducted to validate password hashing using BCrypt and to ensure the integrity and correctness of JWT token generation.

3.4.2 Customer Checkout

- Inputs to Test: Cart items with sufficient stock, items exceeding available stock, empty carts, and cancelled transactions.
- Expected Outputs: Successful order placement with accurate stock deduction, appropriate error messages for insufficient stock, and proper recording of order history.
- Verification Method: Integration testing between the frontend cart module and backend checkout API is performed. Additionally, transactional testing is used to verify database consistency after each operation.

3.4.3 Bulk Inventory Upload

- Inputs to Test: Valid CSV files, files with missing fields, incorrect data types, and large datasets (e.g., 500+ SKUs).
- Expected Outputs: Successful insertion of valid records, row-level error reporting for invalid entries, and stable system performance during high-volume uploads.

- Verification Method: Algorithmic validation of the CSV parsing function and associated price computation logic is conducted to ensure data integrity.

3.4.4 FEFO Deduction Logic

- Inputs to Test: Cart items associated with multiple inventory batches having different expiry dates.
- Expected Outputs: The system prioritizes deduction from the earliest expiry batch (FEFO principle), and correct batch identifiers are recorded in transaction logs.
- Verification Method: SQL-level database audits are performed to confirm accurate batch-level quantity updates and consistency after each transaction.

3.4.5 Spoilage Logging

- Inputs to Test: Administrative inputs for expired stock, damaged goods, and invalid or negative quantity entries.
- Expected Outputs: Spoilage records are correctly created, inventory levels are updated in real time, and invalid inputs are rejected.
- Verification Method: Cross-verification is performed between spoilage log entries and the stock_movements audit table to ensure data consistency and traceability.

3.4.6 Predictive Restocking

- Inputs to Test: Historical sales datasets with varying demand trends and perishability factors.
- Expected Outputs: The system generates reorder alerts when stock falls below calculated thresholds, and recommended restocking quantities align with the adaptive forecasting model.
- Verification Method: Algorithmic verification is conducted using test datasets, along with manual comparison between predicted outputs and expected reorder points.

3.4.7 Expired Batch Cleanup

- Inputs to Test: Administrative execution of the "Cleanup Expired" function for products containing multiple expired batches.

- Expected Outputs: Expired batches are automatically deactivated, corresponding spoilage records are generated, and available stock levels are updated immediately.
- Verification Method: Database auditing is performed to verify updates to the is active flag and the correct insertion of records into the spoilage log table.

3.4.8 Inventory Reporting and Export

- Inputs to Test: Administrative requests to export stock reports in CSV or PDF format.
- Expected Outputs: Properly formatted reports are generated, containing real-time stock levels, low-stock alerts, and recent spoilage summaries.
- Verification Method: Visual inspection is conducted by comparing the exported reports against the current state of the MySQL database.

3.5 Summary

This chapter outlined the methodology and system planning for the Integrated E-Commerce Inventory System using the Agile Scrum approach. The project was developed iteratively across twelve sprints, ensuring continuous improvement and alignment with system requirements.

Functional and non-functional requirements were defined to guide the development of core features such as authentication, checkout, FEFO-based inventory control, spoilage logging, and predictive restocking. A structured sprint plan ensured systematic delivery of these modules.

A comprehensive verification and validation plan was also established, applying unit, integration, and database testing. Critical components, particularly the FEFO algorithm, underwent multiple testing iterations to ensure accuracy, prevent ghost stock, and maintain data consistency. Logical traces and database audits were used to validate transaction correctness and inventory integrity.

Overall, this chapter provides a clear and structured foundation for the system's implementation and evaluation.

Chapter 4 System Design

This chapter describes the system design for Web-Based Grocery E-Commerce and Inventory System, translating requirements into structured architecture. The system adopts a layered design consisting of frontend, backend, and database components to ensure scalability and maintainability. Key modules include authentication, checkout, bulk inventory upload, FEFO-based deduction, spoilage logging, and predictive restocking. Design diagrams such as architecture, activity, and database models are used to illustrate system flow and interactions. Overall, this design provides a clear blueprint to ensure all components function cohesively and maintain accurate inventory management.

4.1 System Design Architecture

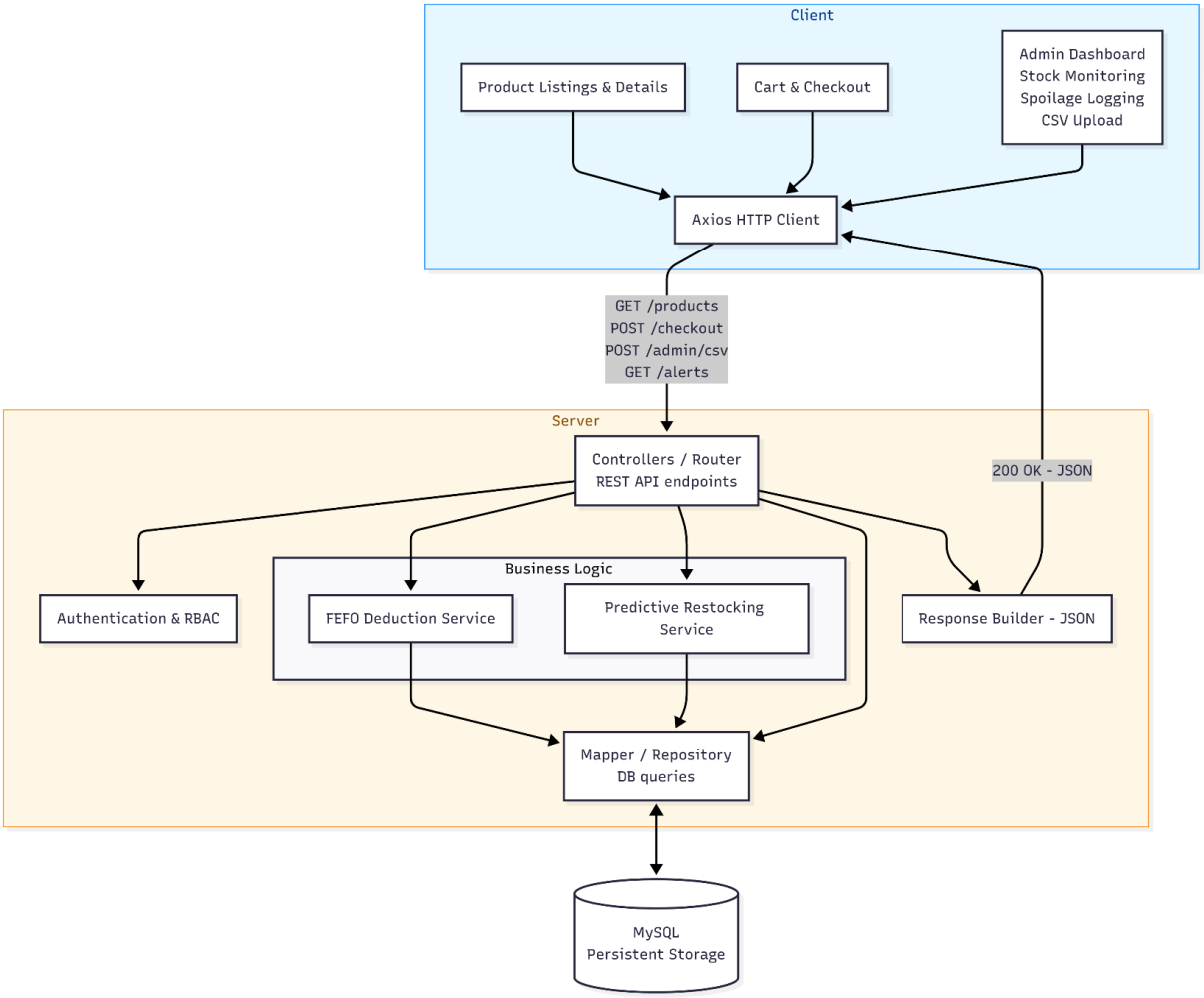


Figure 4.1 System Design Architecture Diagram

Figure 4.1 illustrates a system design architecture called Client-Server Architecture that suits the proposed system design. This design architecture involved two roles which are client and server. Below is the description of each role:

Client

- Displays:
    - Product listings, product details
    - Cart, checkout
    - Admin dashboard (stock monitoring, spoilage logging, CSV upload)
- Sends HTTP requests via Axios to the server to:
    - Fetch product data
    - Send checkout orders
    - Upload CSV files
    - Retrieve restocking alerts
- Receives JSON responses from the server.

Server

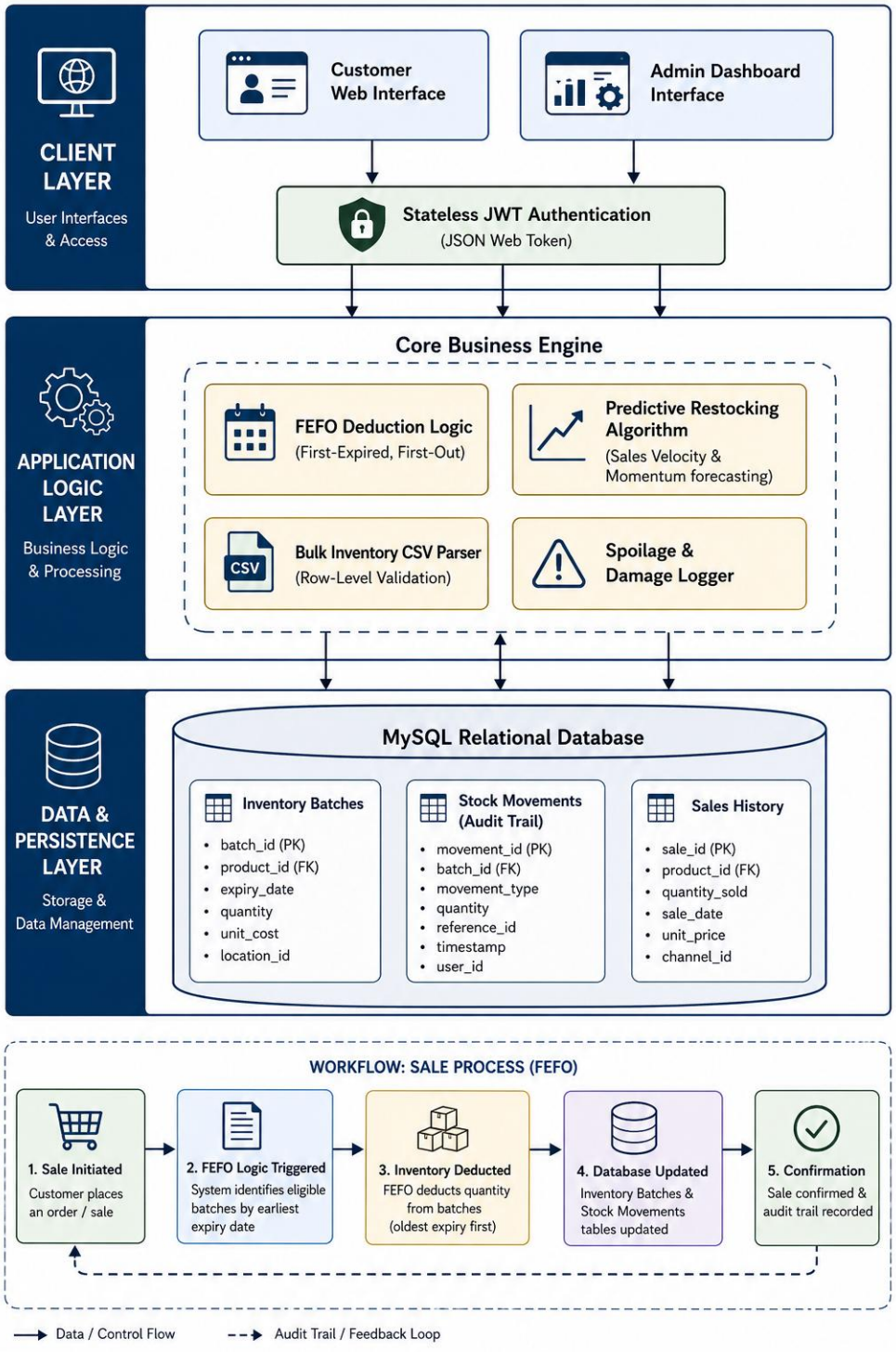
- Listens for HTTP requests from the client.
- Processes requests using:
    - Controllers (API endpoints)
    - Services (business logic: FEFO deduction, predictive restocking)
    - Mapper (database queries)
- Sends JSON responses back to the client.
- Handles security (authentication, role-based authorization).

Connects to MySQL for persistent data storage.

4.1.1 System Block Diagram

The System Block Diagram illustrates the interaction between the user interfaces, the core application services, and the data persistence layer.

Figure 4.1.1 System Block Diagram



4.1.2 Technology Stack

The project requires both hardware and software tools to support full-stack development. The hardware consists of a personal laptop (Acer Nitro 5 AN515-45) equipped with an AMD Ryzen 5 5600H processor, 16GB DDR5 RAM, and 512GB NVMe SSD storage, which is sufficient for both backend and frontend development, database hosting, and testing.

The software stack includes:

- Frontend Development: Next.js for building dynamic and responsive user interfaces.
- Backend Development: Spring Boot (Java) for implementing server-side business logic and RESTful APIs.
- Database Management: MySQL for persistent storage of user, product, order, and inventory data.
- Version Control: Git and GitHub for version management and collaboration.
- CSV Processing: OpenCSV library in Java for parsing bulk upload files.
- Development Tools: IntelliJ IDEA for backend coding, Cursor/Visual Studio Code for frontend development, and MySQL Workbench for schema design and queries.

These tools were selected based on their robustness, scalability, and compatibility with the project objectives.

4.2 Use Case Diagram

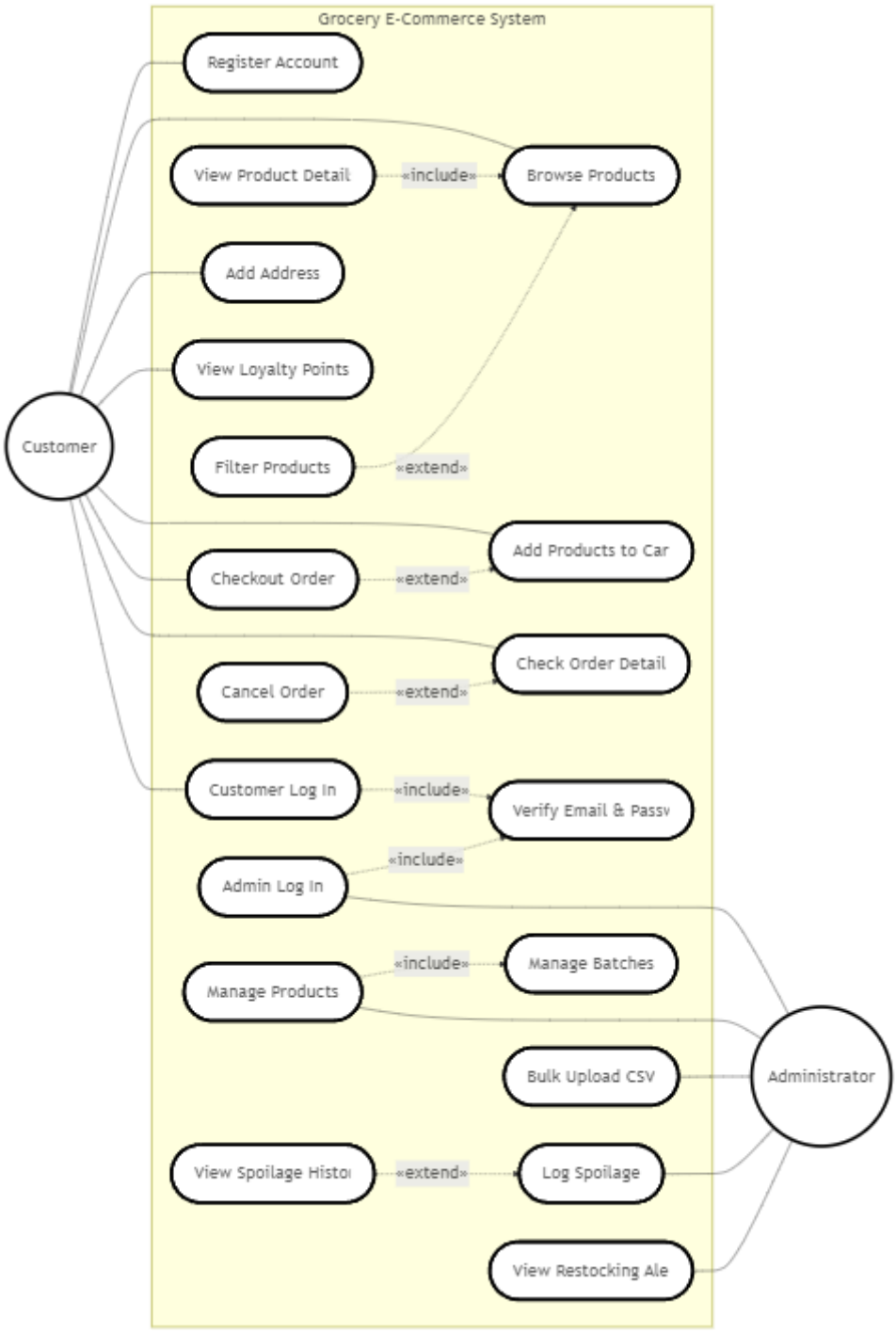


Figure 4.2.1 Use Case diagram

4.3 High-level Flow Diagram

High-level Flow Diagram provided an overview of the action flow for each role. Different roles have different access to interact with the system. Administrators have higher access compared to customer role. It abstracts away low-level details to emphasize the sequence of core activities, key decision points, and data hand-offs.

4.3.1 Administrator Flow Diagram

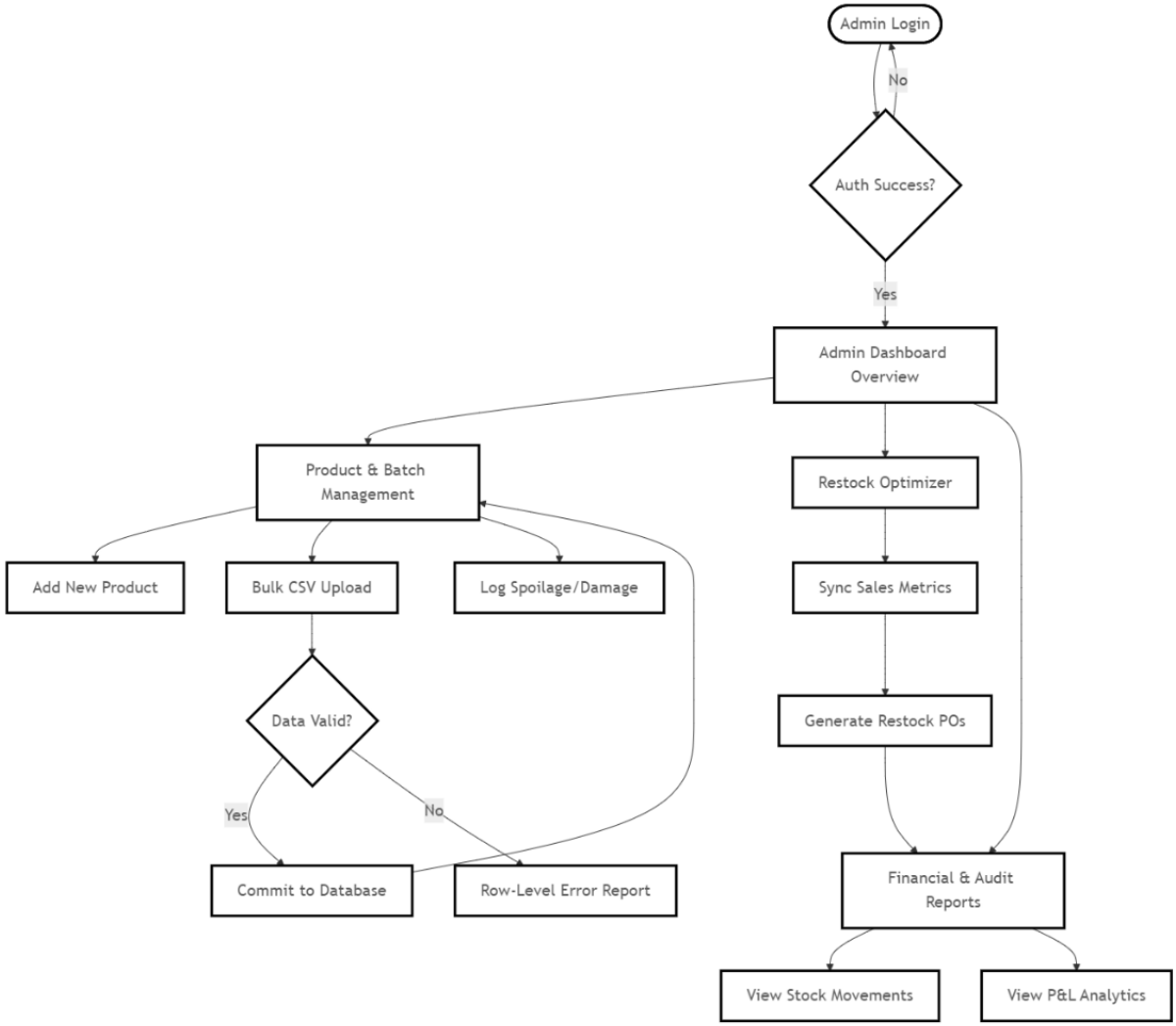


Figure 4.3.1 Administrator Flow Diagram

4.3.2 Customer Flow Diagram

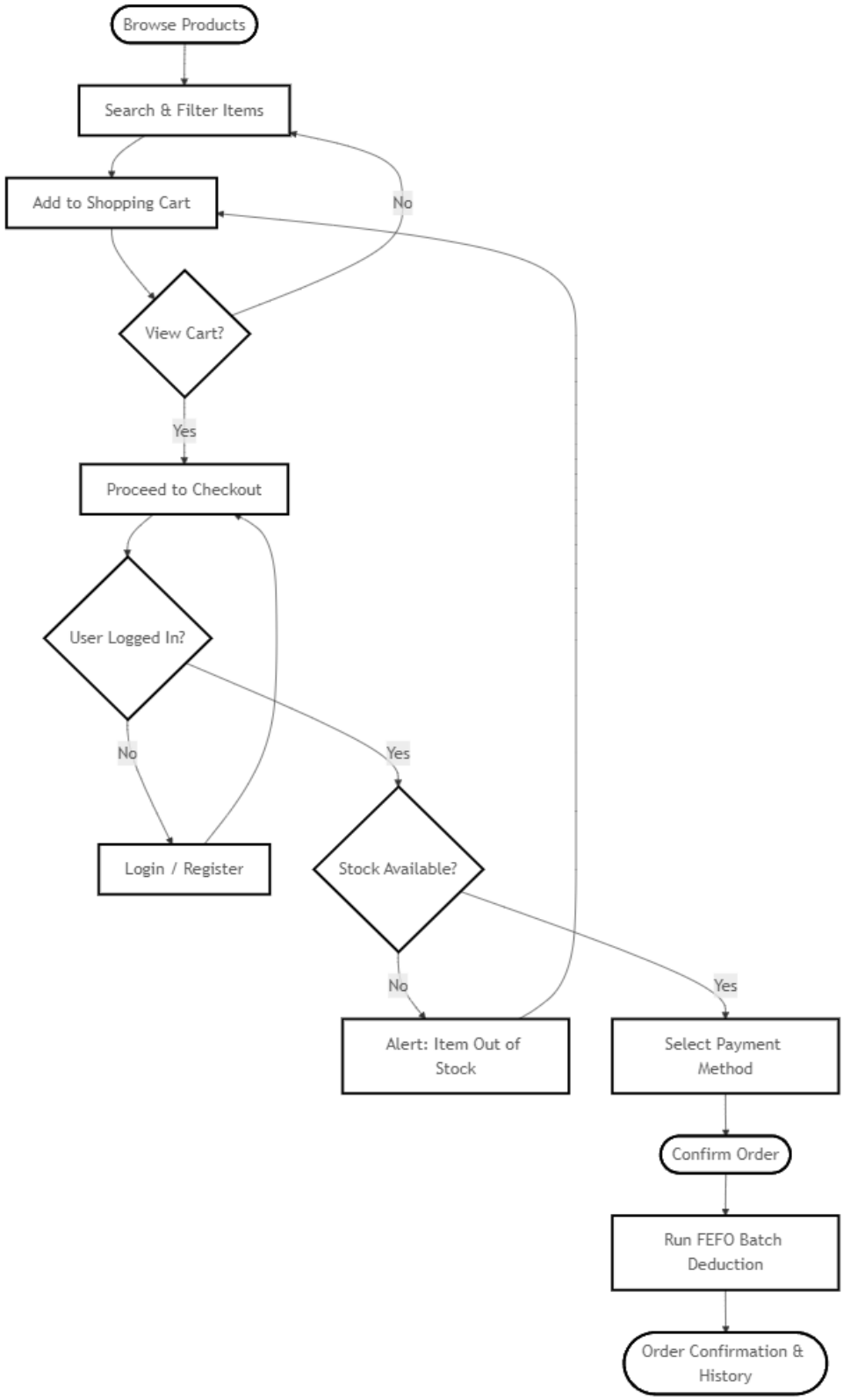


Figure 4.3.2 Customer Flow Diagram

4.4 Entity-Relation Diagram (ERD)

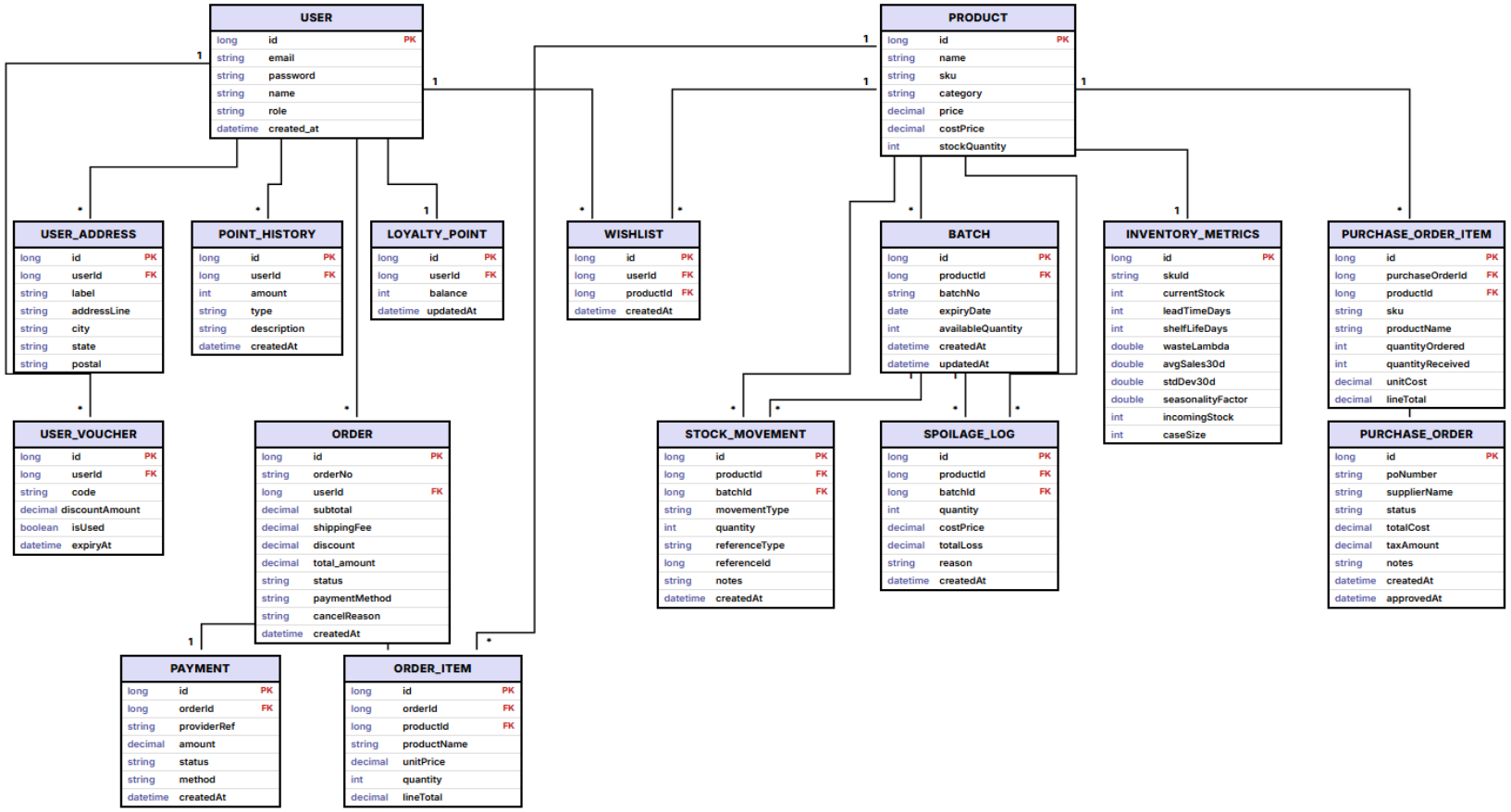


Figure 4.4 Entity-Relation Diagram

4.5 Activity Diagram

Activity diagrams are used in system design to model the dynamic behaviour of the system by showing workflows of activities and actions. They illustrate how different processes are triggered, the decision points involved, and the sequence of operations carried out by the system. In this project, activity diagrams are applied to represent the functional flows of both customer and administrator modules.

4.5.1 Log Spoilage

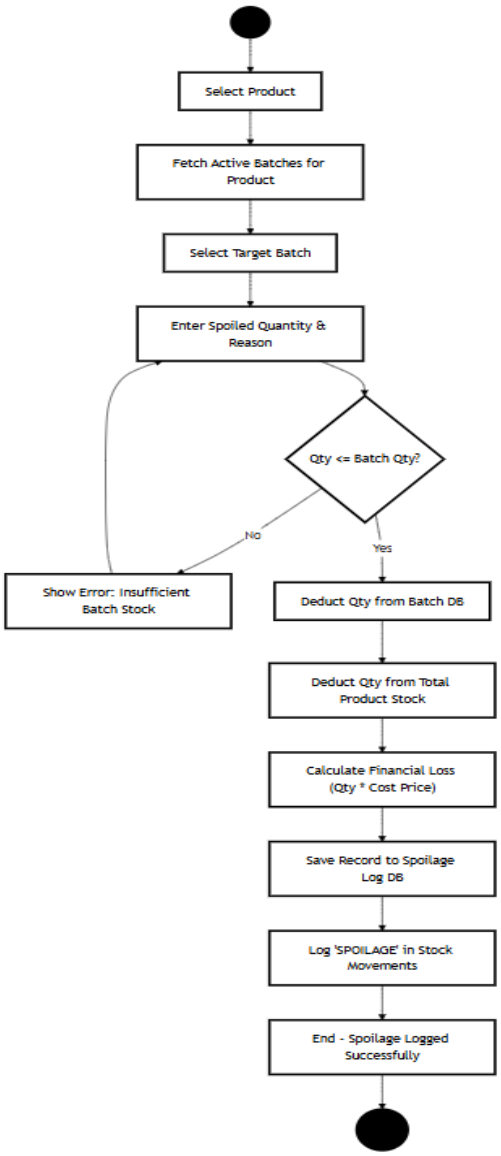


Figure 4.5.1 Log Spoilage Activity Diagram

4.5.2 Manage Purchase Orders (PO)

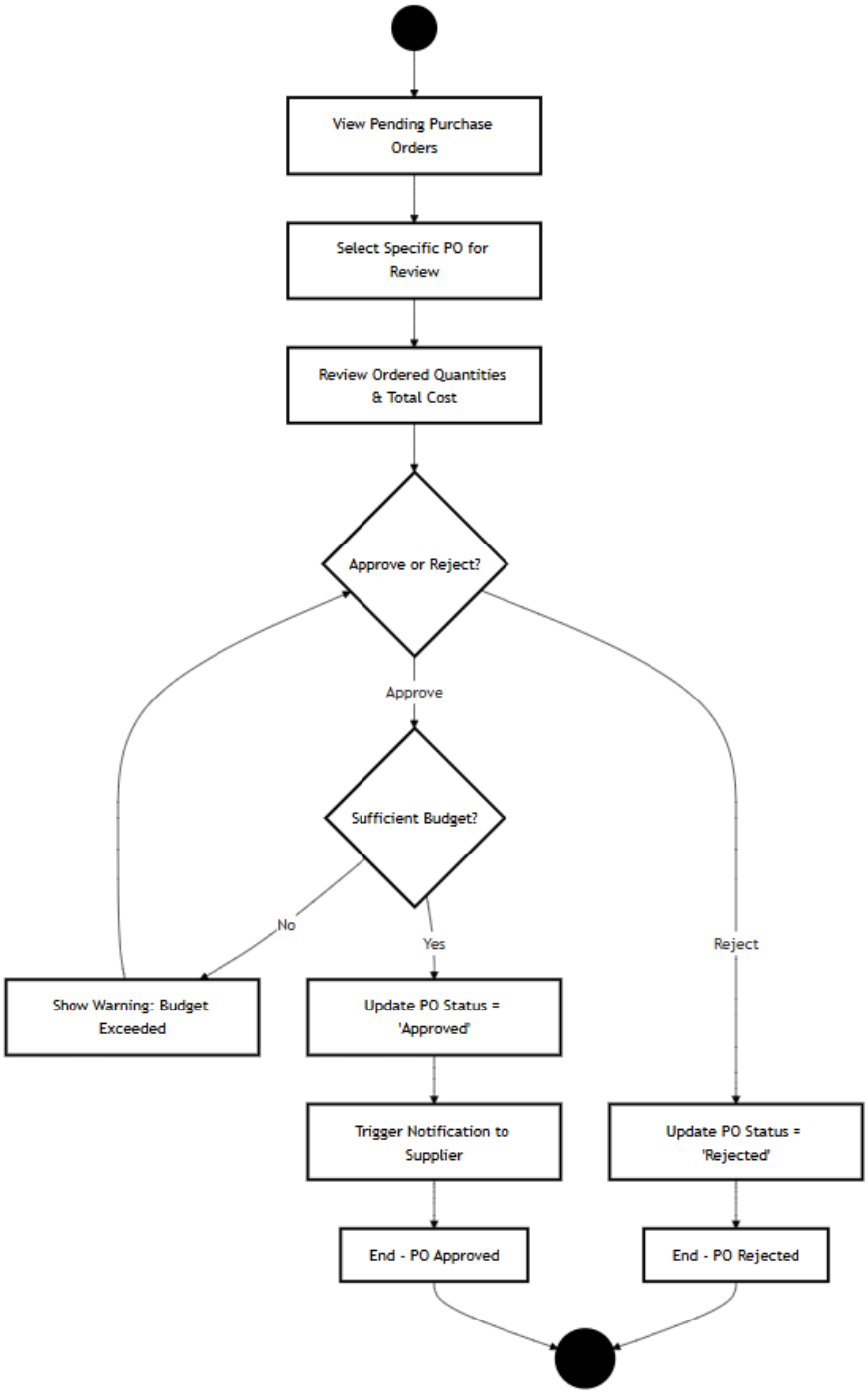


Figure 4.5.2 Purchase Order Activity Diagram

4.5.3 Financial Reports Generation

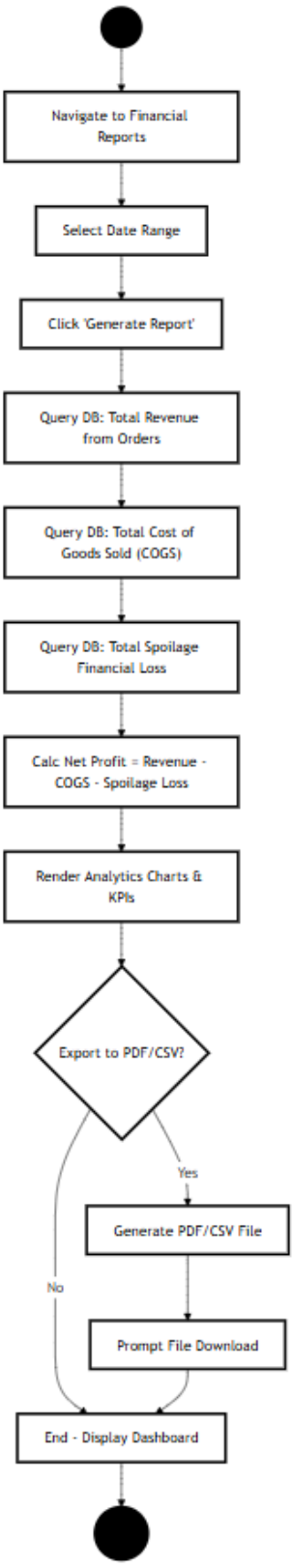


Figure 4.5.3 Financial Reports Generation Activity Diagram

4.5.4 Manage Single Product (Add/Edit)

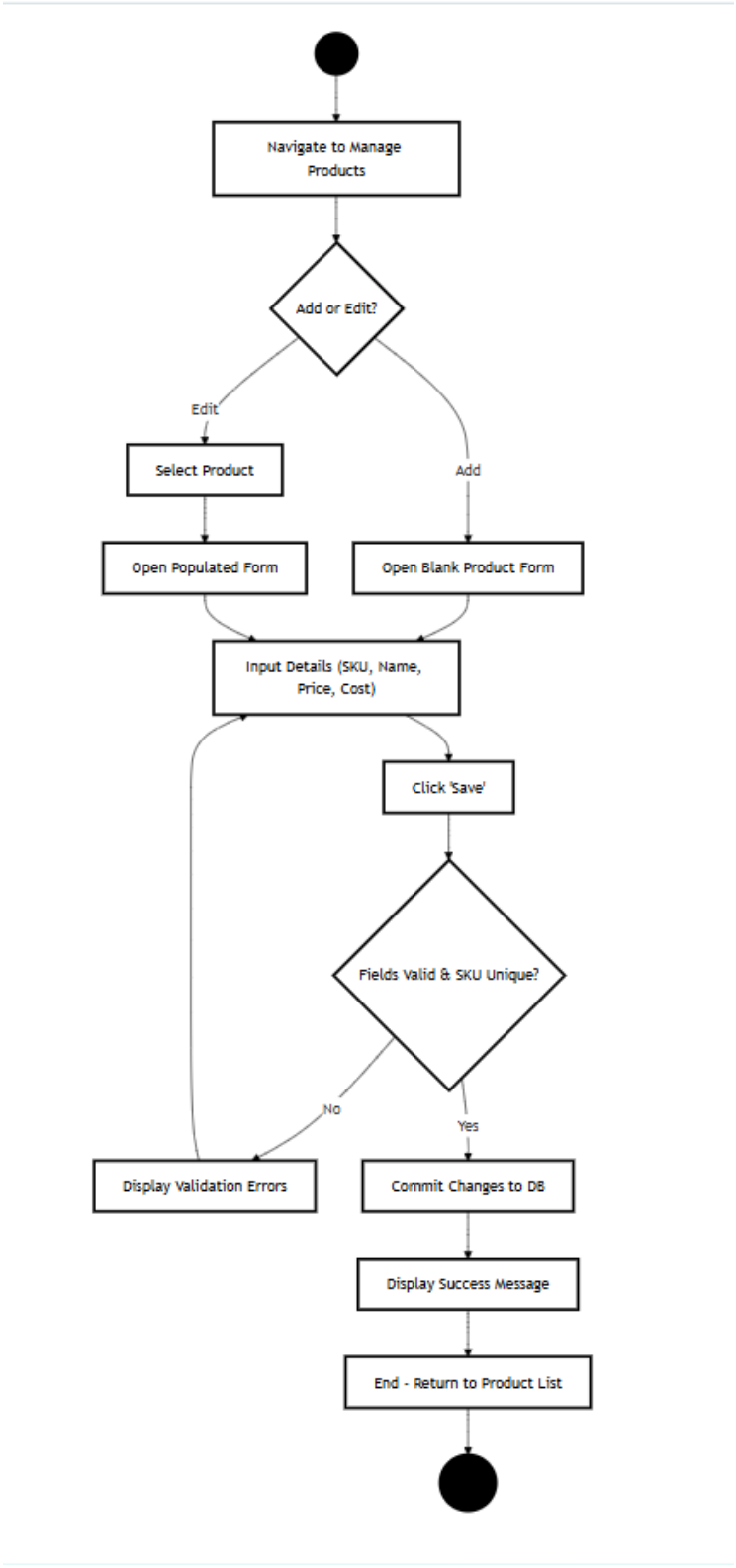


Figure 4.5.4 Single Product Addition Activity Diagram

4.5.5 Bulk Upload

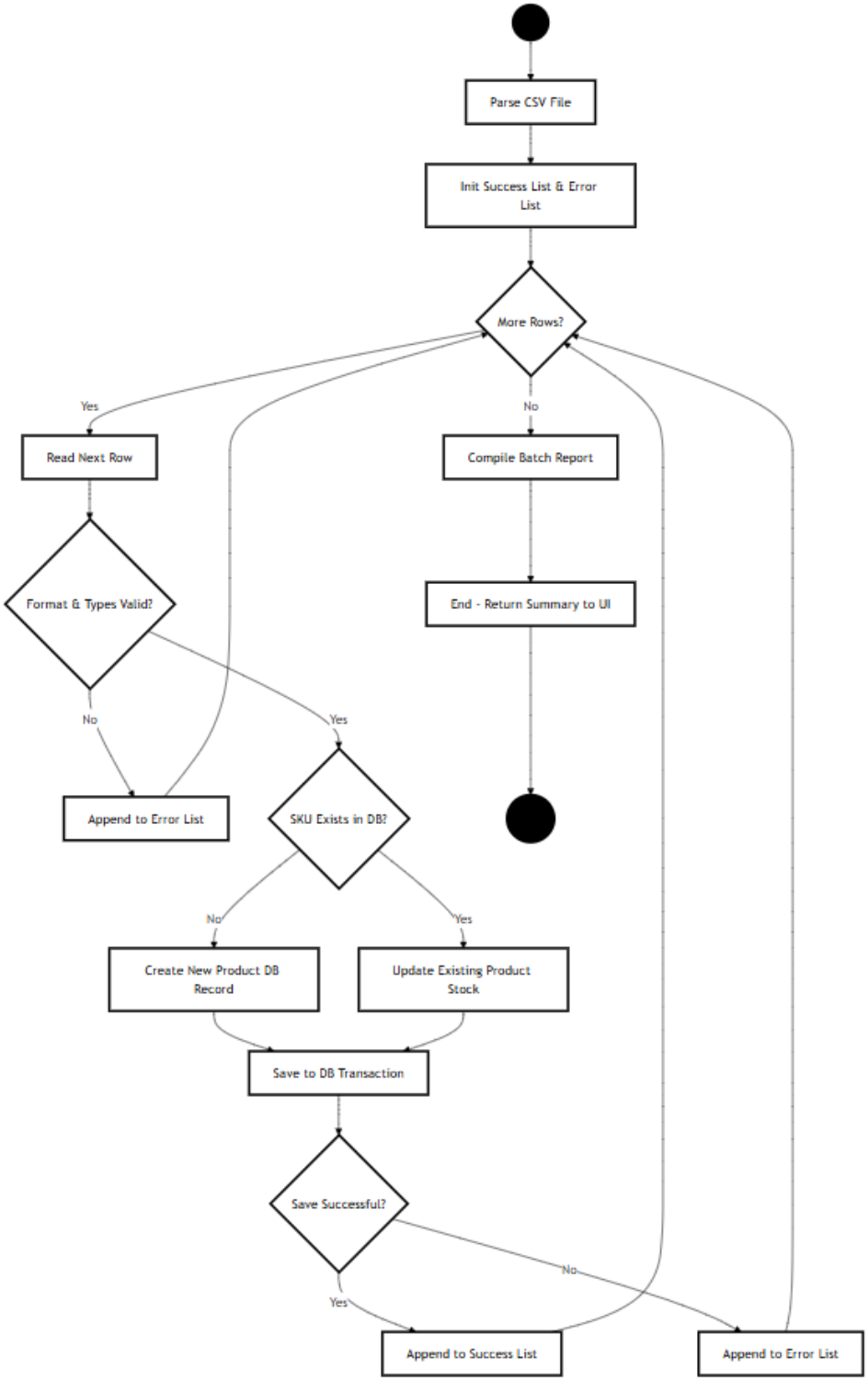


Figure 4.5.5 Bulk Upload Activity Diagram

4.5.6 User Registration

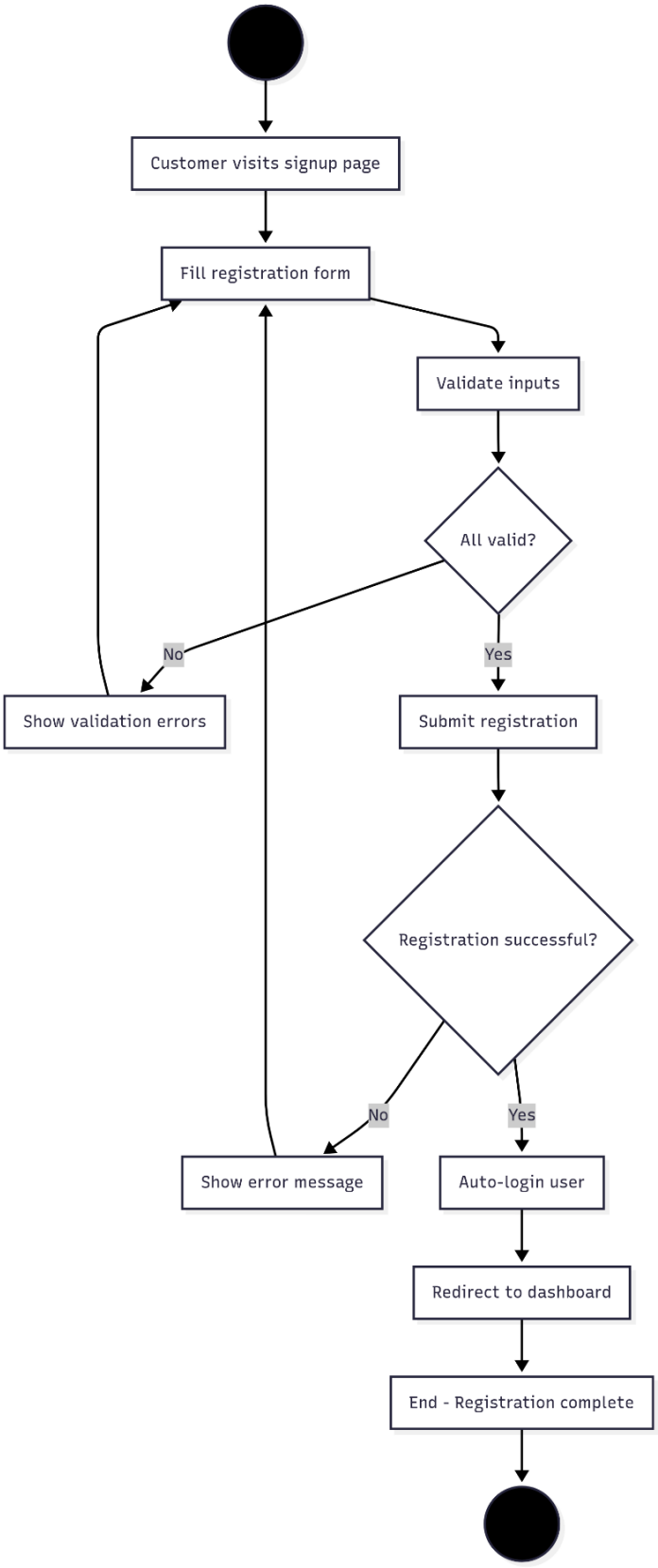


Figure 4.5.6 User Registration Activity Diagram

4.5.7 User Login with Role Check

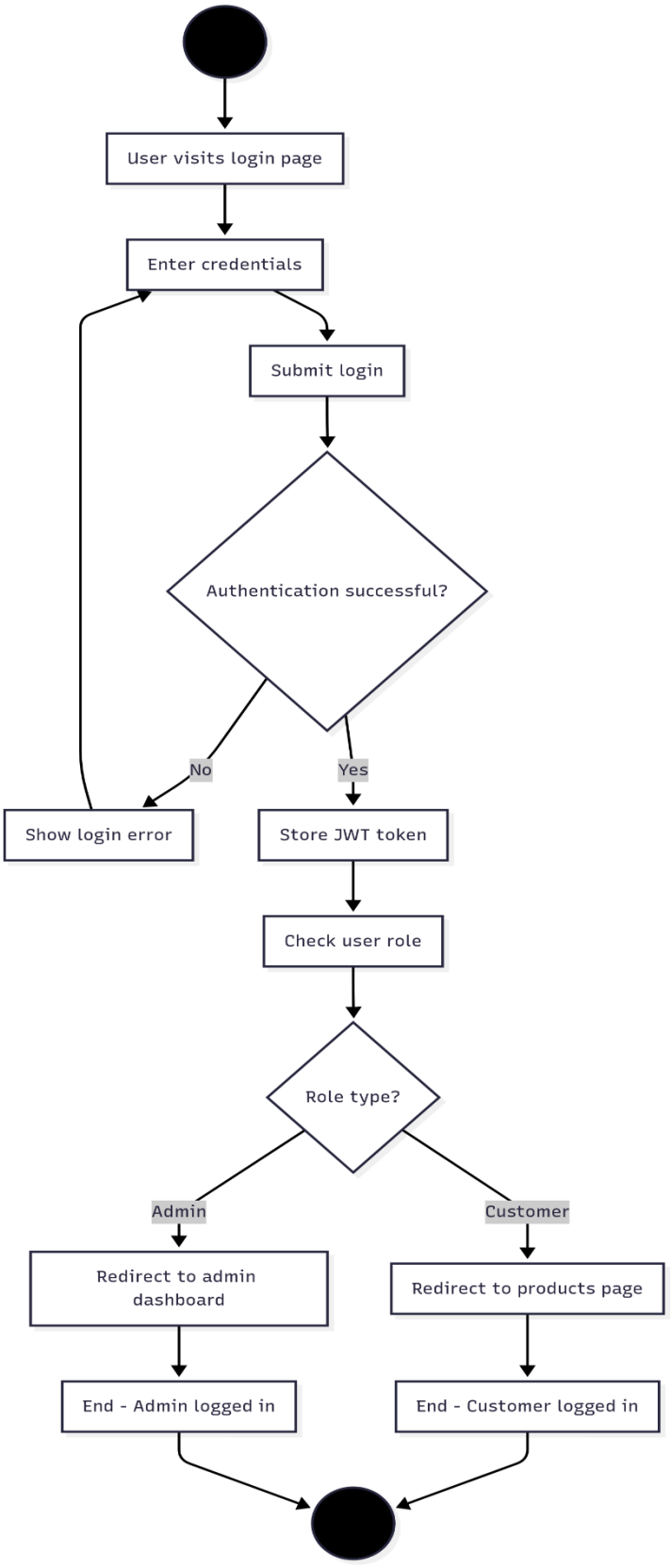


Figure 4.5.7 User Login with Role Check Activity Diagram

4.5.8 Add to Cart and Stock Validation

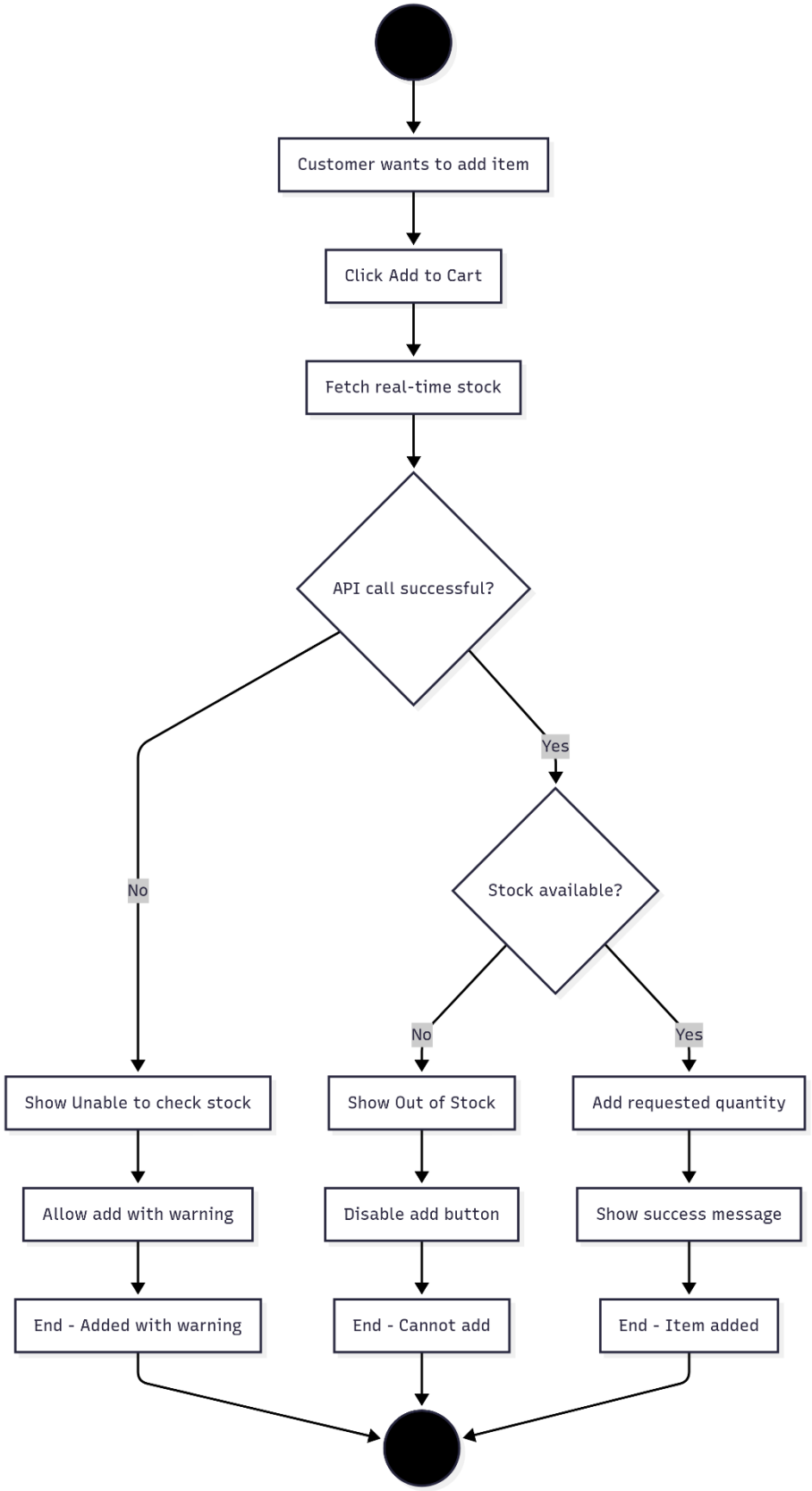


Figure 4.5.8 Add to Cart and Stock Validation Activity Diagram

4.5.9 Cart Review and Stock Issues

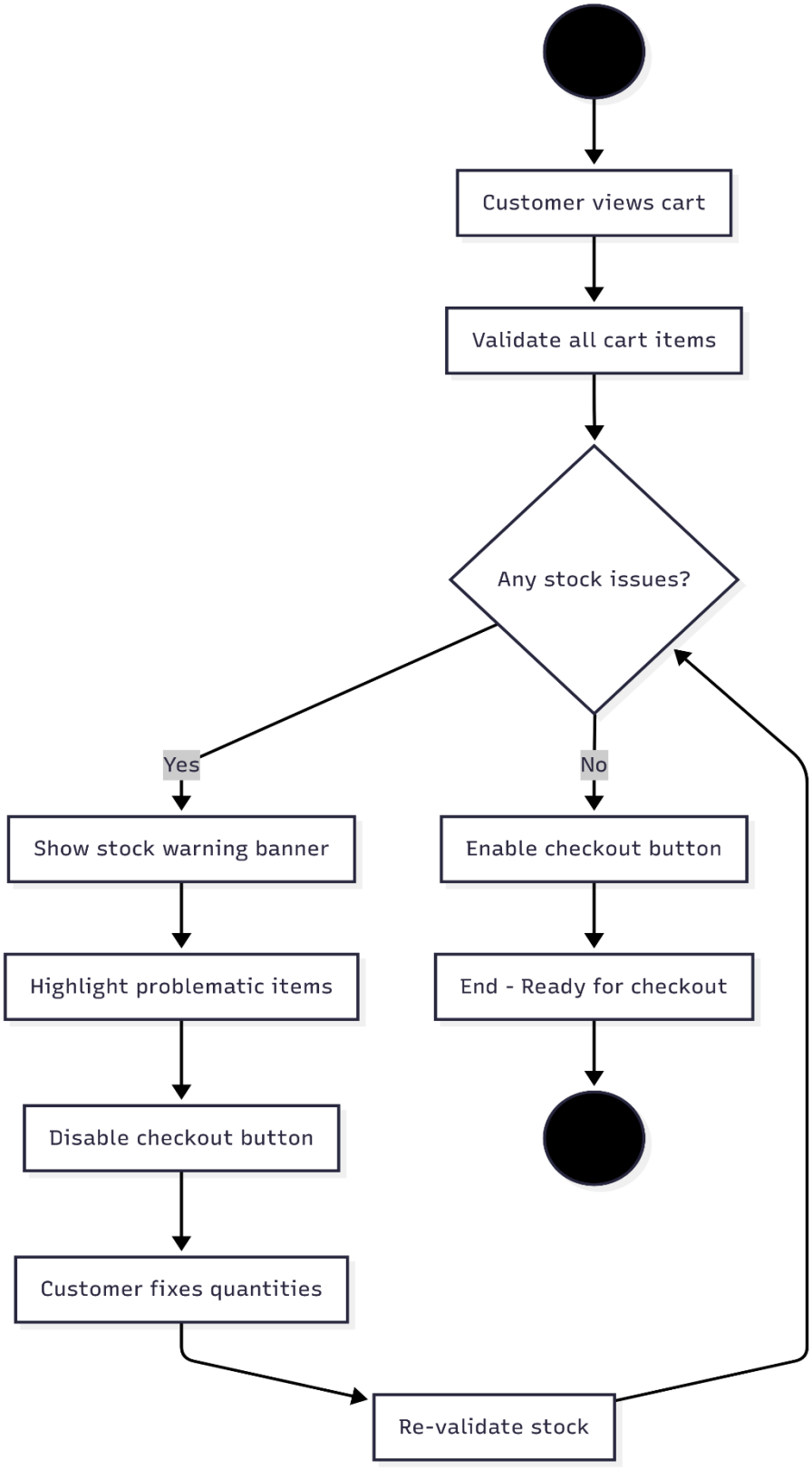


Figure 4.5.9 Cart Review and Stock Issues Activity Diagram

4.5.10 Checkout Authentication

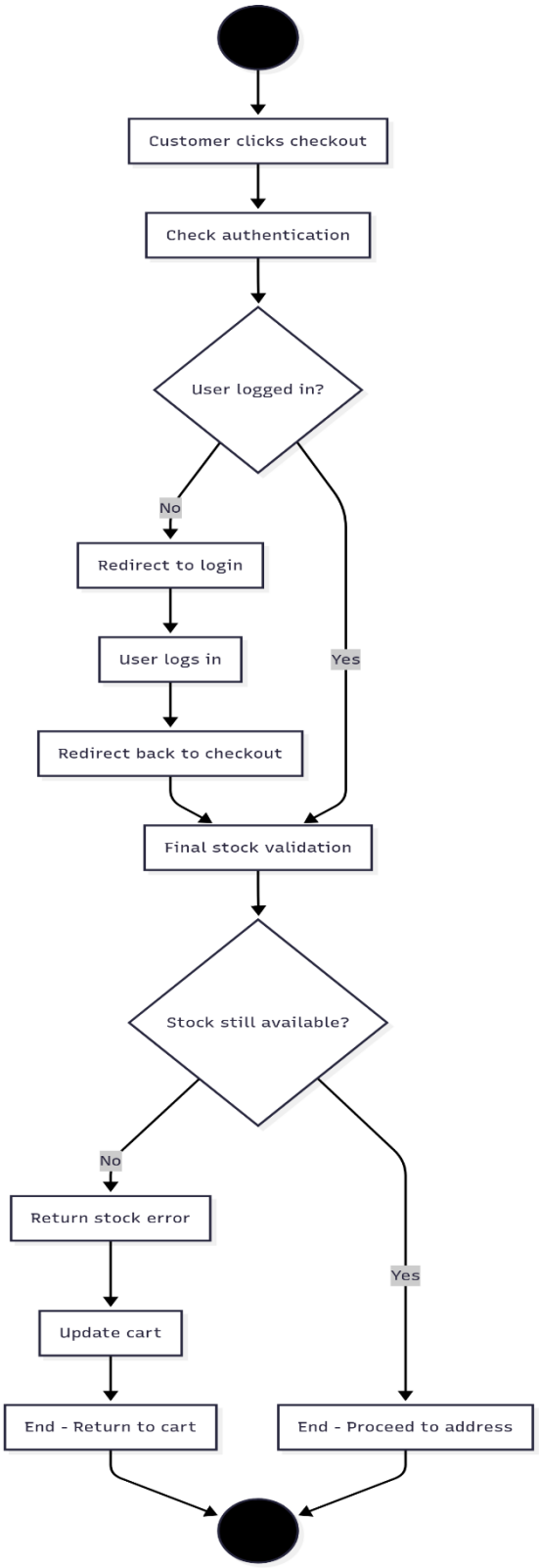


Figure 4.5.10 Checkout Authentication Activity Diagram

4.5.11 Address Management

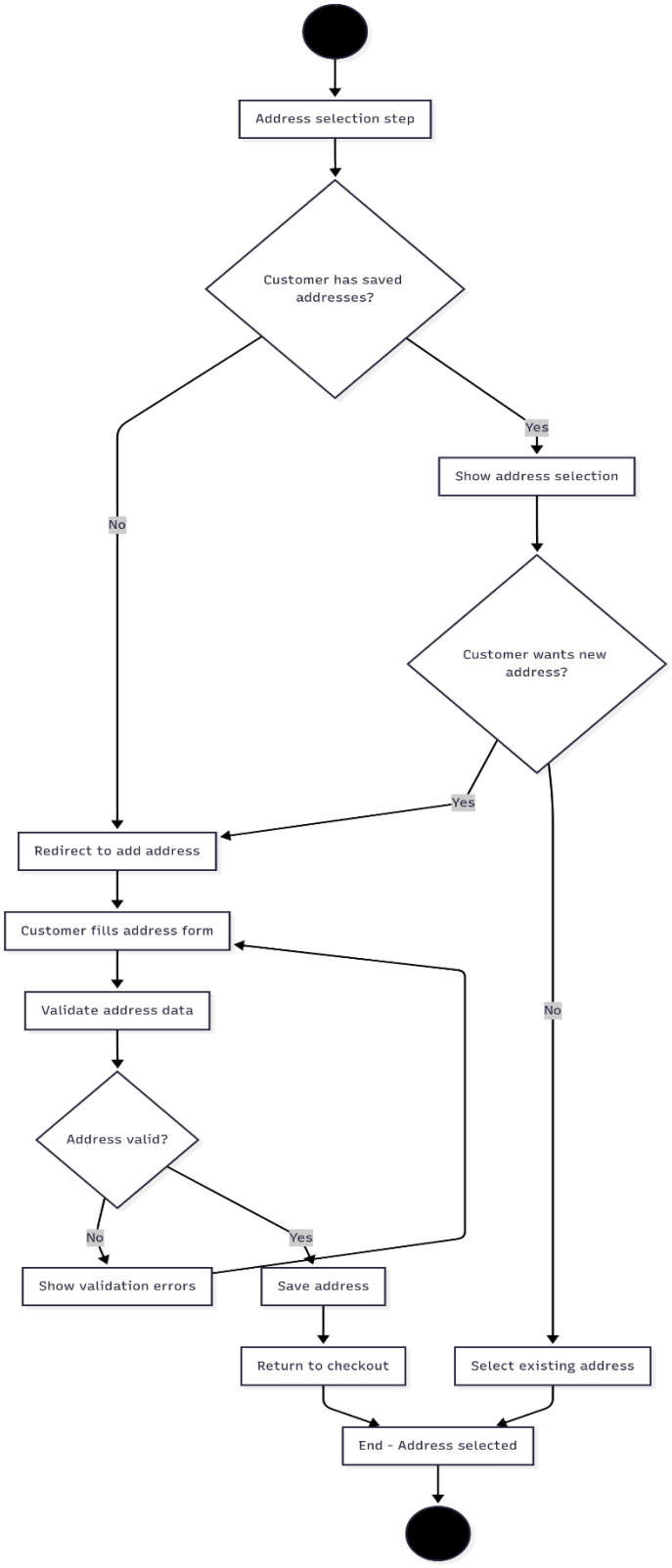


Figure 4.5.11 Address Management Activity Diagram

4.5.12 Payment Processing

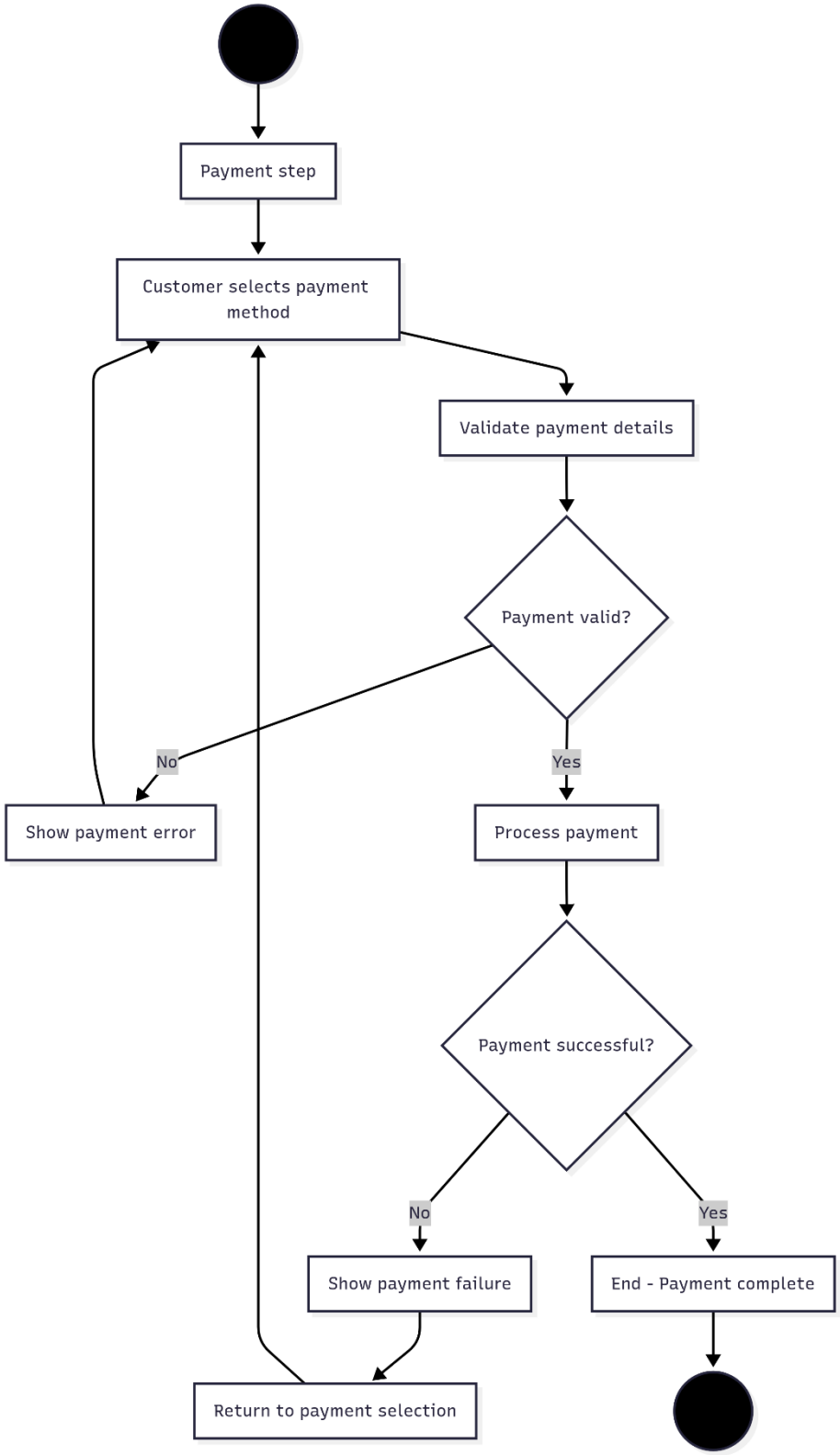


Figure 4.5.12 Payment Process Activity Diagram

4.5.13 Order Completion

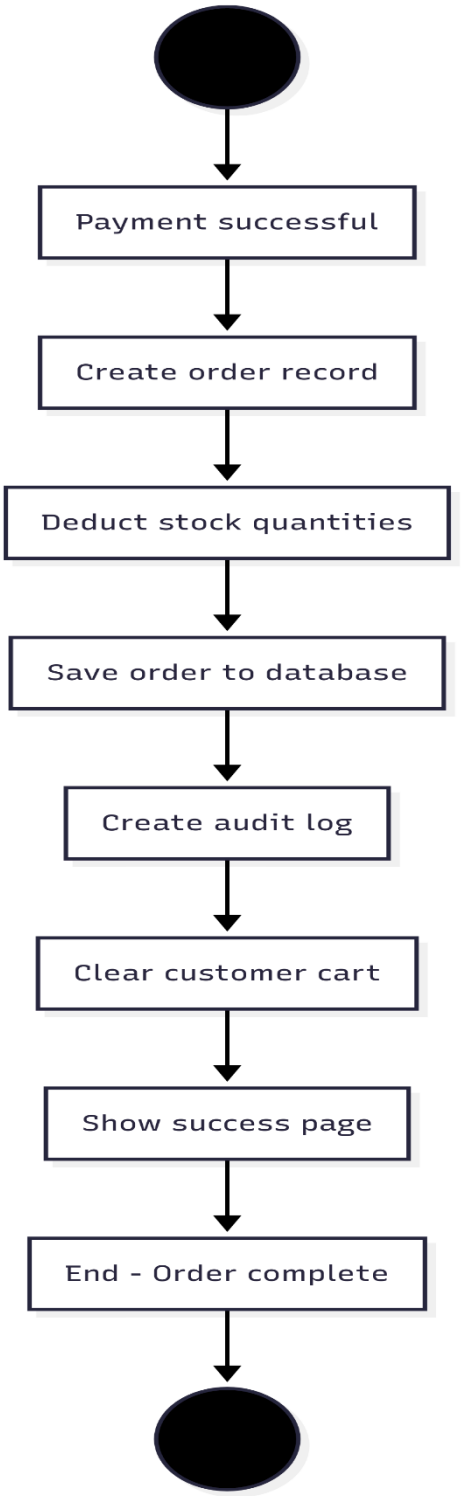


Figure 4.5.13 Order Completion Activity Diagram

4.5.14 Order Cancellation

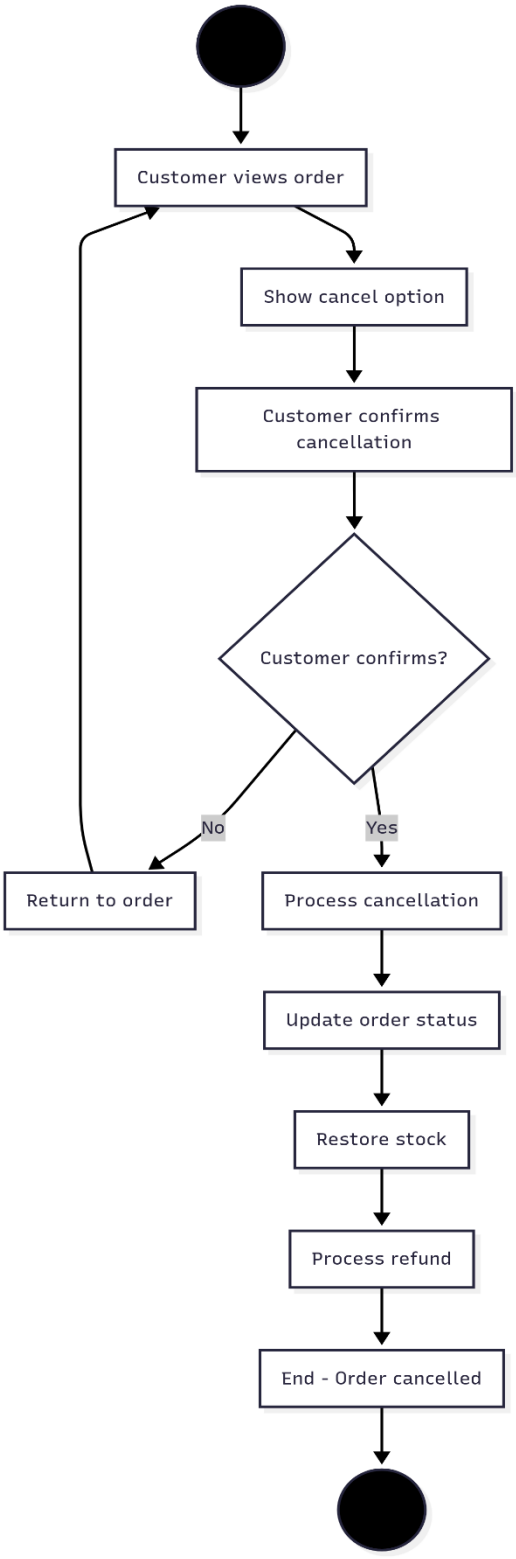


Figure 4.5.14 Order Cancellation Activity Diagram

4.6 Summary

In this chapter, we discussed the design of the E-Commerce Website and Inventory System. A client-server system design was employed, where the Next.js client makes API calls to the Spring Boot server, and MySQL serves as a persistent data store to store user, product, order, inventory batch, spoilage, and financial information.

The chapter described the system architecture, the client, the server and the database. The client layer is responsible for user interface interactions, such as product viewing, checkout and actioning the administrator dashboard, and the server layer is responsible for business processing such as authentication, first-expired-first-out (FEFO) batch stock deduction, bulk CSV upload, spoilage recording and predictive restocking. The database side provides persistent storage for system records, ensuring that they are properly stored and can be accessed when needed.

Furthermore, the chapter outlined the key system diagrams, which include the use case diagram, high-level flow diagrams, entity-relationship diagram and activity diagrams. These diagrams depict the way the system is used by the customers and administrators, and how the system functions to perform the key activities of product management, checkout, bulk upload, inventory management, order management and spoilage management.

In summary, the system design presented in Chapter 4 serves as a guide for the development of the system. The system design supports the project objectives by ensuring the system is well architected, easily maintained and capable of performing the primary e-commerce functions as well as inventory-related features such as FEFO, spoilage recording and predictive restocking.

Chapter 5 System Implementation

This chapter presents the implementation details of the Integrated E-Commerce Inventory System. It describes the hardware and software environments used during development, system configuration procedures, and the operational workflow of the system. In addition, the chapter provides a comprehensive walkthrough of the implemented modules supported by system interfaces and screenshots. Challenges encountered throughout the development sprints and their corresponding solutions are also discussed.

5.1 Hardware Setup

The development and deployment of the system were carried out on a high-performance personal development machine. This setup was required to support the concurrent execution of multiple system components, including the frontend application, backend services, and database server.

The hardware specifications are as follows:

- System Model: Acer Nitro 5 (AN515-45 Series)
- Central Processing Unit (CPU): AMD Ryzen 5 5600H with Radeon Graphics (6 Cores, 12 Threads)
- Memory (RAM): 8GB DDR4
- Primary Storage: 512GB NVMe M.2 SSD
- Display: 15.6-inch Full HD (1920 × 1080 resolution)

The multi-core processor enabled efficient parallel processing for backend services and frontend rendering. The upgraded memory capacity supported development tools, local servers, and database operations simultaneously without performance degradation. Additionally, the NVMe SSD provided high-speed data access, improving database query performance and application responsiveness.

Overall, this hardware environment ensured a stable and efficient development lifecycle, particularly when running the Next.js frontend, Spring Boot backend, and MySQL database concurrently.

5.2 Software Setup

The system was developed using a modern full-stack technology stack to ensure scalability, maintainability, and performance. The selected technologies support modular development and seamless integration between system components.

Backend Technologies

- Java 17 (OpenJDK): Core programming language for backend development
- Spring Boot 3.2.x: Framework for building RESTful APIs and business logic
- Maven 3.9: Dependency management and project build automation

Frontend Technologies

- Node.js 18.x: Runtime environment for executing JavaScript applications
- Next.js 14.x (App Router): Framework for server-side rendering and routing
- React 18: Component-based UI development
- Tailwind CSS: Utility-first CSS framework for responsive design

Database

- MySQL 8.0 Community Server: Relational database management system ensuring ACID compliance and data integrity

Development Tools

- IntelliJ IDEA Ultimate: Primary IDE for backend development and debugging
- Visual Studio Code / Cursor: Lightweight editors for frontend development
- MySQL Workbench: Database design, ERD modelling, and query execution
- Postman: API testing and validation of request-response cycles

Version Control

- Git 2.x with GitHub: Source code management, version tracking, and collaboration

This software environment enabled efficient development, testing, and debugging across all system layers while ensuring compatibility between frontend and backend components.

5.3 System Settings and Configuration

To ensure seamless integration between system components, several configurations were implemented for both frontend and backend environments.

Backend Configuration

The backend system was configured using the application.properties file. Key configurations include:

- Database Connection:
  Defined using MySQL URL, username, and password to establish connectivity between the backend and database.
- Authentication Settings:
  JSON Web Token (JWT) was implemented for secure authentication. Secret keys and token expiration durations were configured to enable stateless session management.
- CORS Configuration:
  Cross-Origin Resource Sharing (CORS) was enabled to allow requests only from the frontend domain (http://localhost:3000). This prevents unauthorised access and ensures secure communication between client and server.

Frontend Configuration

The frontend environment utilized a .env.local file to manage environment variables:

- NEXT_PUBLIC_API_URL was defined to specify the backend server endpoint (http://localhost:8080)
- This approach centralizes API configuration and simplifies deployment adjustments

System Execution

The system is executed using the following commands:

- Backend:

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

- Frontend:

npm run dev

These commands initialize the backend server and frontend development server respectively, enabling real-time interaction between system components.

5.3.1 Backend Library Setup

Table 5.3.1 List of Backend Libraries

Layer	Library	Purpose
Web/API	org.springframework.boot:spring-boot-starter-web	REST controllers, JSON, validation pipeline
Security	org.springframework.boot:spring-boot-starter-security	Security filter chain, RBAC
Security (JWT)	io.jsonwebtoken:jjwt-api + jjwt-impl + jjwt-jackson	Issue/verify JWT access tokens
Data Access	com.baomidou:mybatis-plus-boot-starter	Simplified MyBatis (CRUD, wrappers)
JDBC	org.springframework.boot:spring-boot-starter-jdbc	Datasource, transaction mgmt
MySQL Driver	com.mysql:mysql-connector-j	MySQL connectivity
Validation	org.springframework.boot:spring-boot-starter-validation	Bean validation (Jakarta Validation + Hibernate Validator)
CSV	org.apache.commons:commons-csv	Parse CSV for bulk upload
Utils	org.apache.commons:commons-text	String utils/sanitization
Hashing	org.springframework.security:spring-security-crypto (comes via starter)	BCrypt password hashing

Lombok	org.projectlombok:lombok (scope provided)	Boilerplate reduction (getters/setters)
Testing	org.springframework.boot:spring-boot-starter-test	JUnit + Spring test

5.4 System Operation

This section demonstrates the operational workflow of the system, focusing on the interaction between users and core system modules.

5.4.1 Authentication Module

Figures 5.4.1.1 to 5.4.1.10 illustrate the authentication and profile-related features developed during the first two sprints.

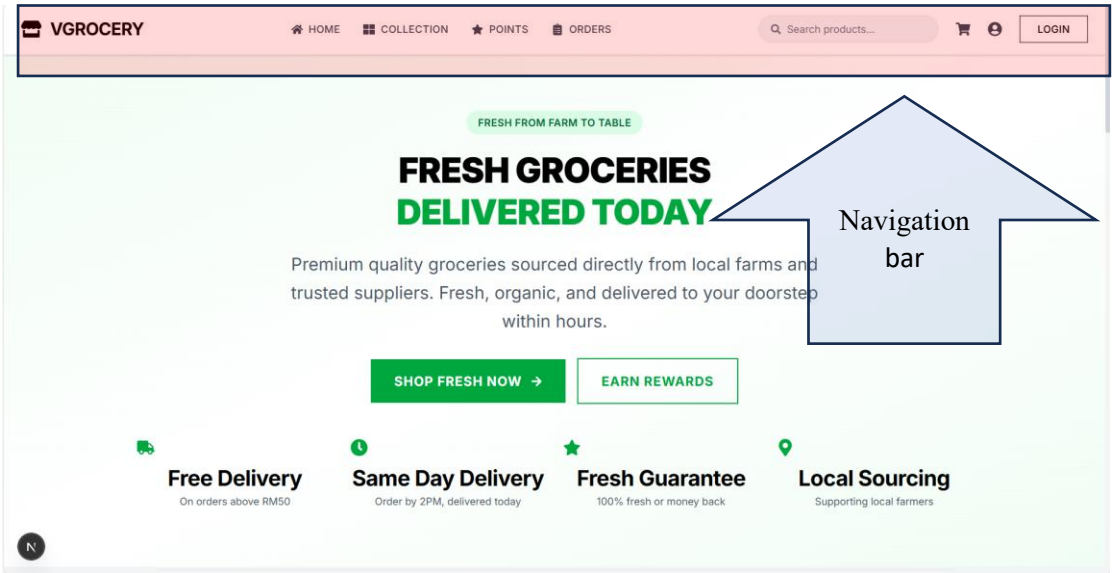


Figure 5.4.1.1 Overview of Home Page

This is the main page of the grocery store website; a navigation bar is placed top to allow user to redirect to several pages as below:

- Website logo, quick home page redirect
- Navigation links to Home, Product, Points, Orders pages
- Search bar to allow users to search for desired products
- Cart Icon, allow users to view the products in cart
- Profile Icon links to profile page

- Login buttons for users to login or create a new account

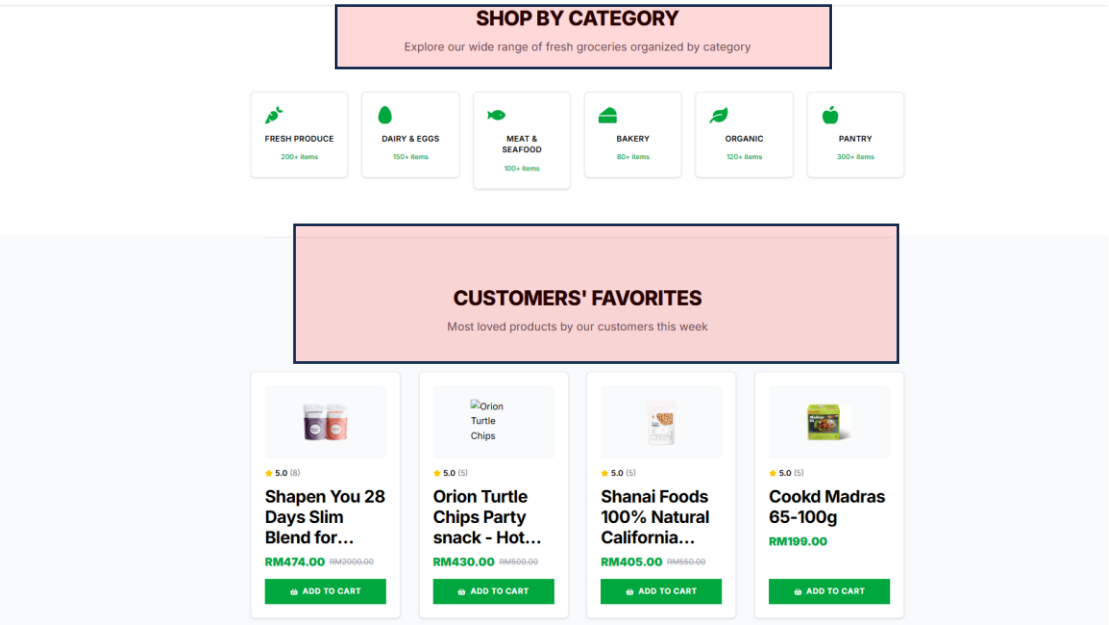


Figure 5.4.1.2 Addiction Element of Home Page

Customer can click on shop fresh now to link to the product page in quick. After clicking on Earn Rewards button, customer is redirected to point page as an alternative path to access that page. Customer can shop by category by clicking on the Shop by Category section. Besides, Customer's Favourites section shows the top 4 rate count and rating products.

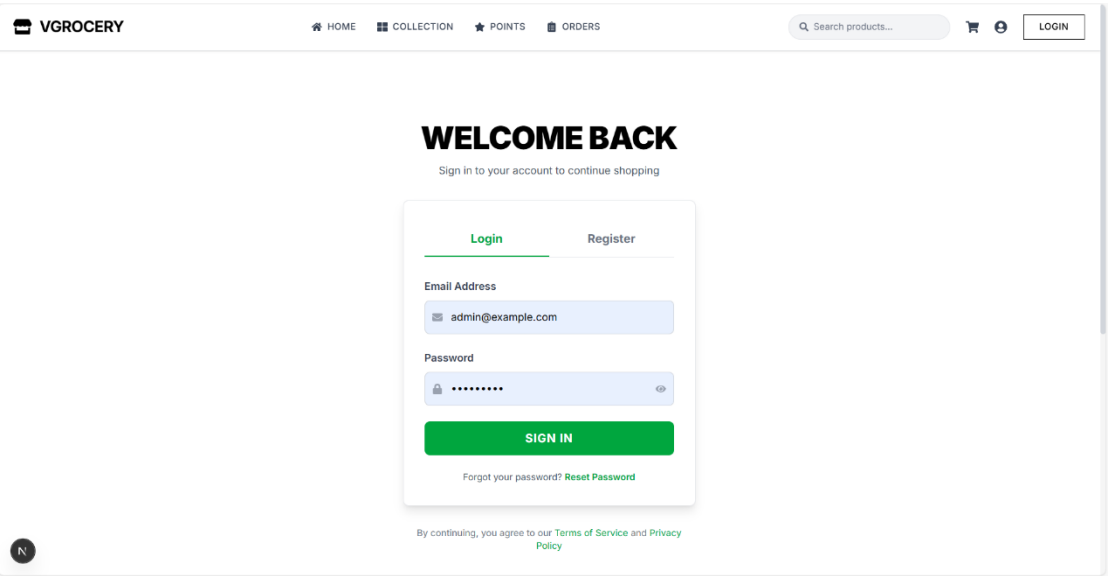


Figure 5.4.1.3 Login function page

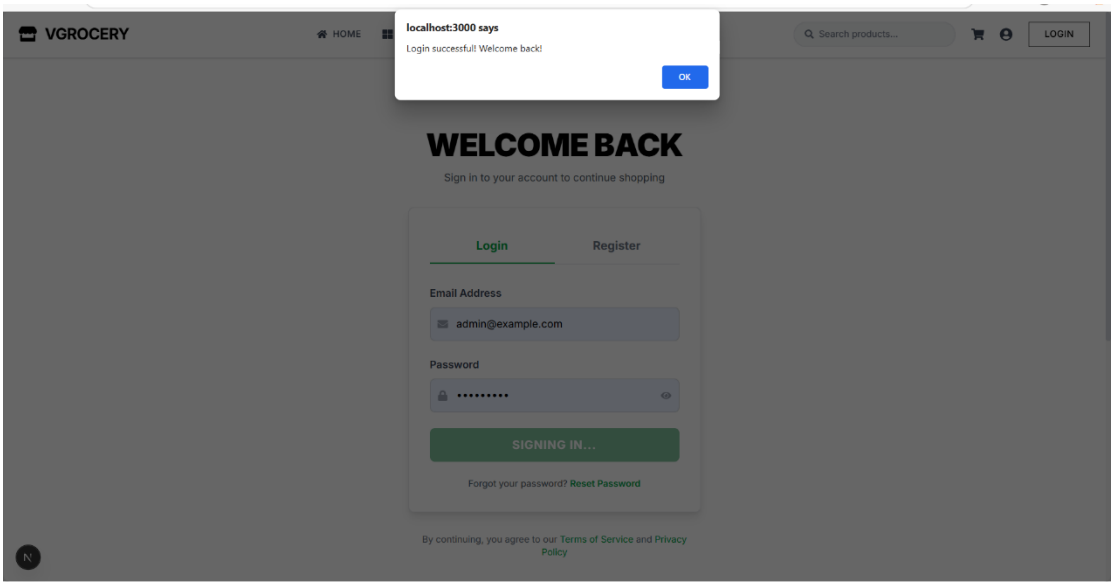


Figure 5.4.1.4 Successful Login

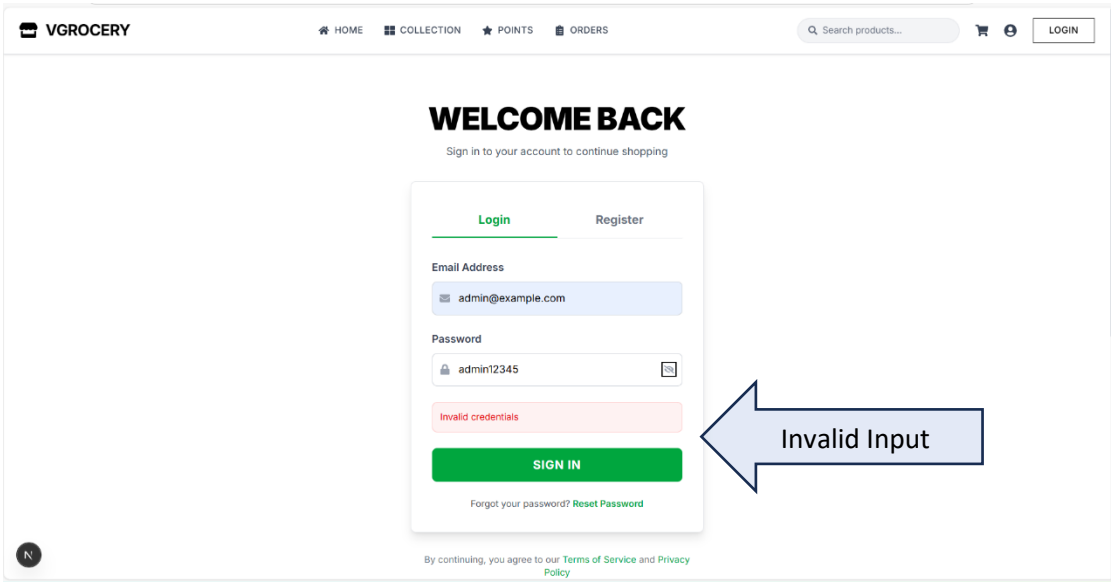


Figure 5.4.1.5 Invalid input for login

Figure 5.4.1.3 shows the login function, and a valid input was inserted to test the usability of the function. After a valid input was inserted, a successful login message was prompted and user login to their account. Figure 5.2.1.4 shows the successful login. Figure 5.4.1.5 shows an invalid input if the user signed in with incorrect email or password.

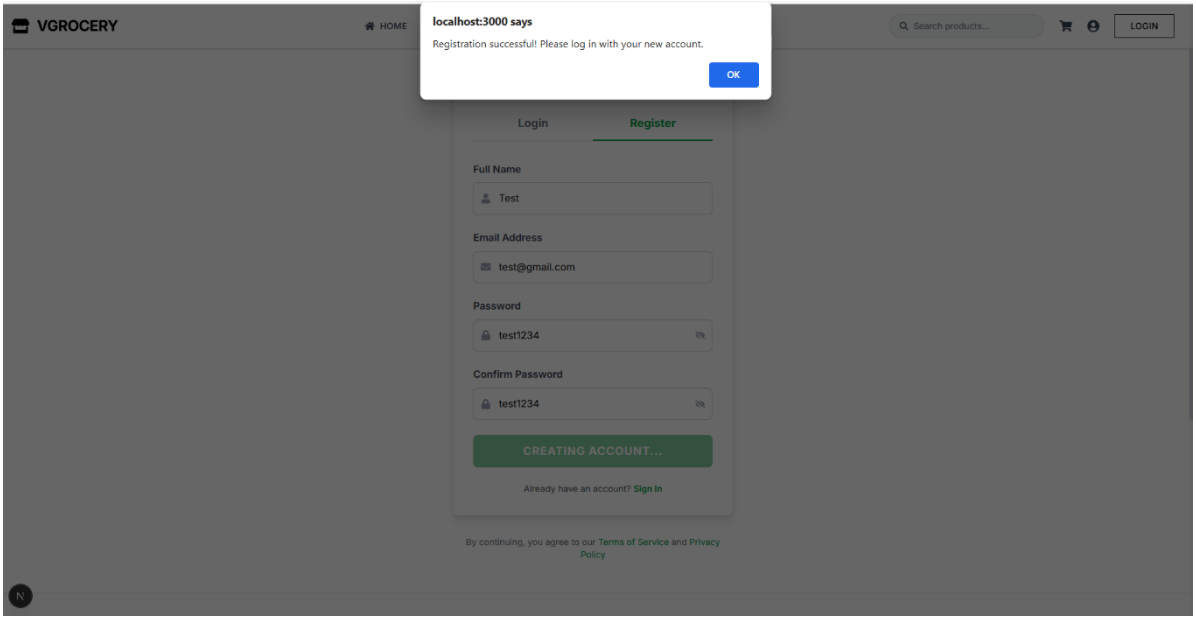


Figure 5.4.1.6 Successful Registration with valid input

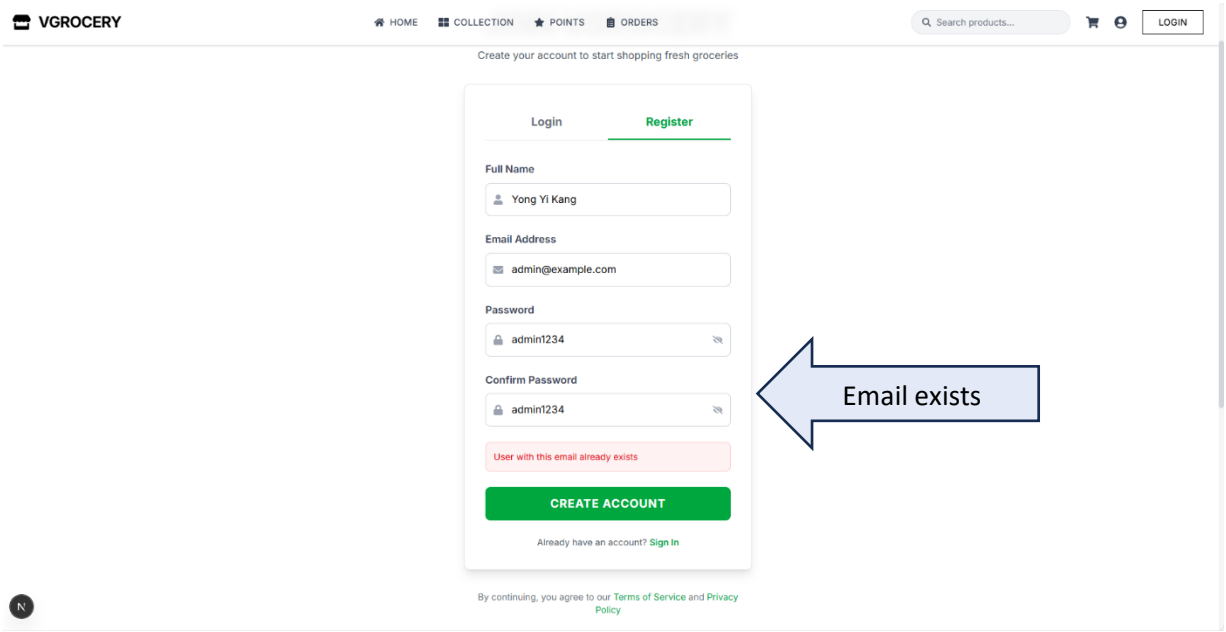


Figure 5.4.1.7 Existing Email Input

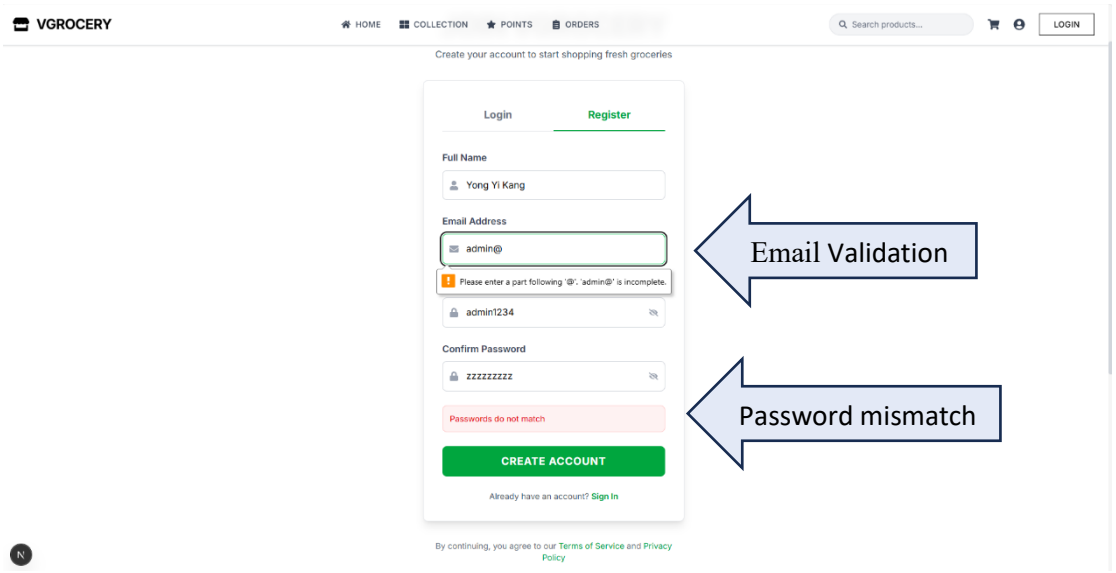


Figure 5.4.1.8 Mismatch password and Email Input Validation

Figure 5.4.1.6 shows a successful registration of account when valid input was inserted. After the input of the information, the system validates the input with the existing user data in persistent database. Figure 5.4.1.7 shows an error when the registration email existed in the system database. Figure 5.4.1.8 shows an error if the password was not exactly matching the confirm password. A valid email that followed the email format is required for the registration.

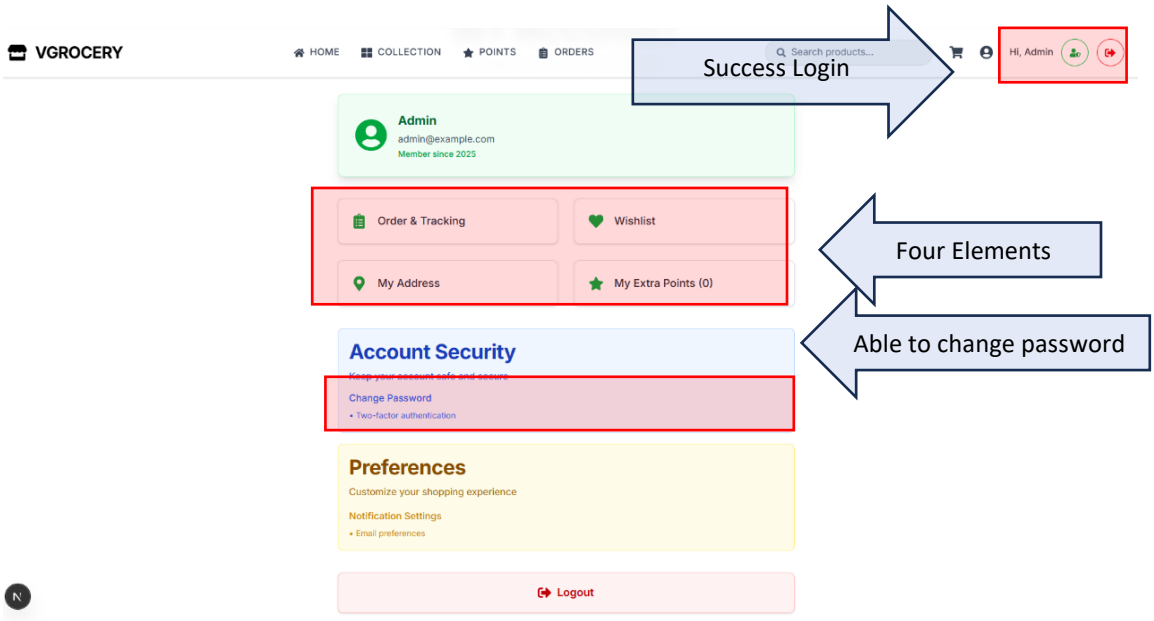


Figure 5.4.1.9 Profile Page

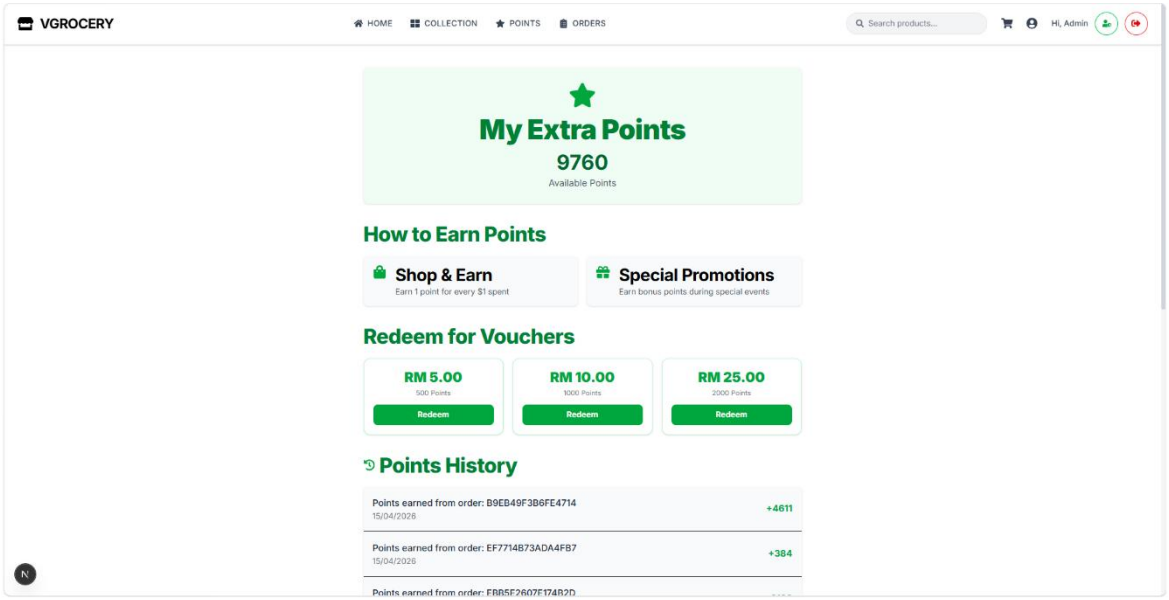


Figure 5.4.1.10 Points Page

This page has four main elements which are order and tracking, Wishlist, address and rewards, by clicking on these buttons the page will redirect to respective page as shown in Figure 5.4.1.9. Users can also view their account information and change password to ensure the security by clicking the change password option. When user click on the point, the system redirected to point page and earned point and point history displayed as shown in Figure 5.4.1.10.

5.4.2 Checkout Module

Figures 5.4.2.1 to 5.4.2.8 illustrate the checkout and product-related features developed during the sprint 3 and 4.

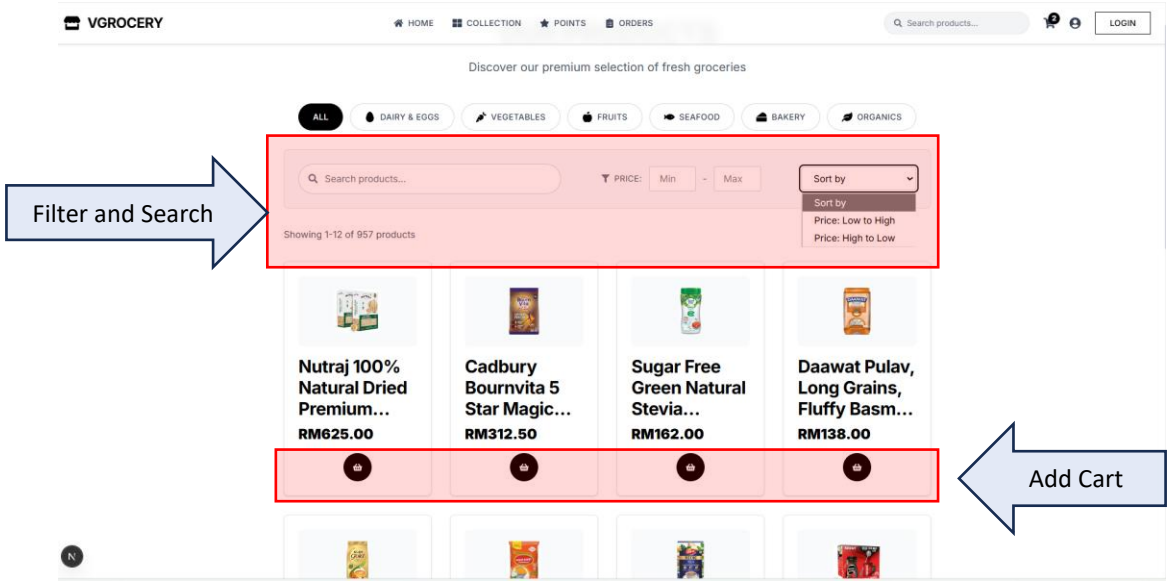


Figure 5.4.2.1 Product Page

Figure 5.4.2.1 shows a product page which include a category bar that allow customer to filter by category. A search bar is included to allow searching for desired products. Customer can choose to filter by entering minimum price and maximum price and sort by ascending or descending price. After clicking on the shopping icon, the selected product will be added into the cart and only the types of products will be shown at top right above the cart icon, instead of showing quantity of single product. After clicking on the image of product, customer will be redirected to product detail page.

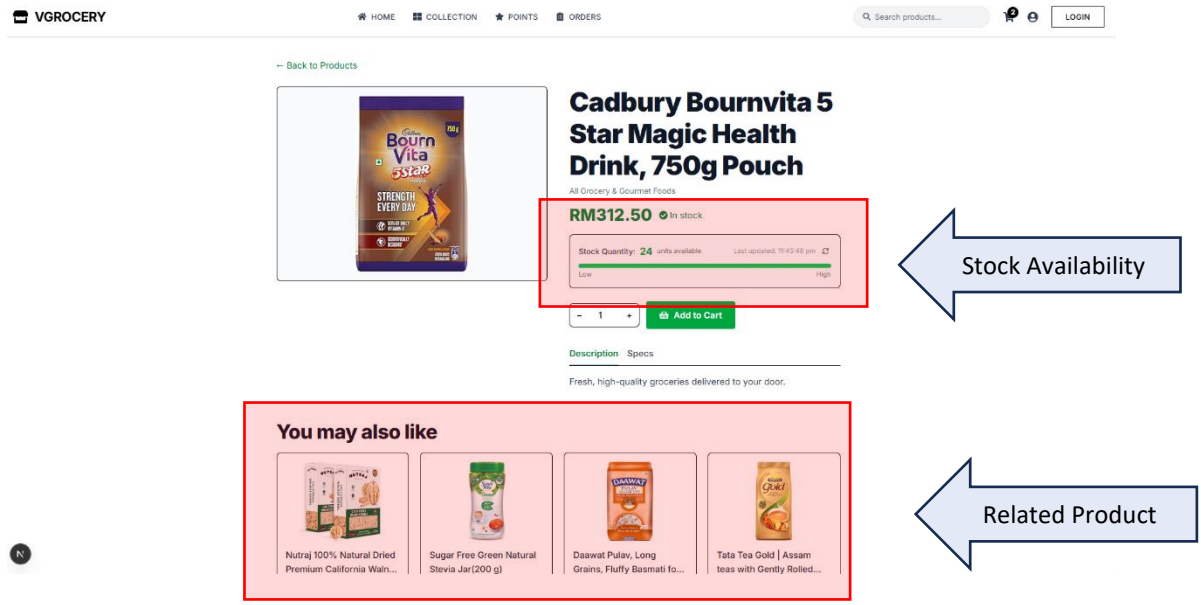


Figure 5.4.2.2 Product Description Page

Figure 5.4.2.2 shows the product details product that allows customers to view the full details of the product and check for the availability of the stock of the product. The real time stock quantity is shown and a reload icon allow customer the track the real time stock. Customer can also click on add to cart to add product and adjust the quantity by clicking on the add or minus button. Below section shows the product the related to the selected product that may be a choice for the customer.

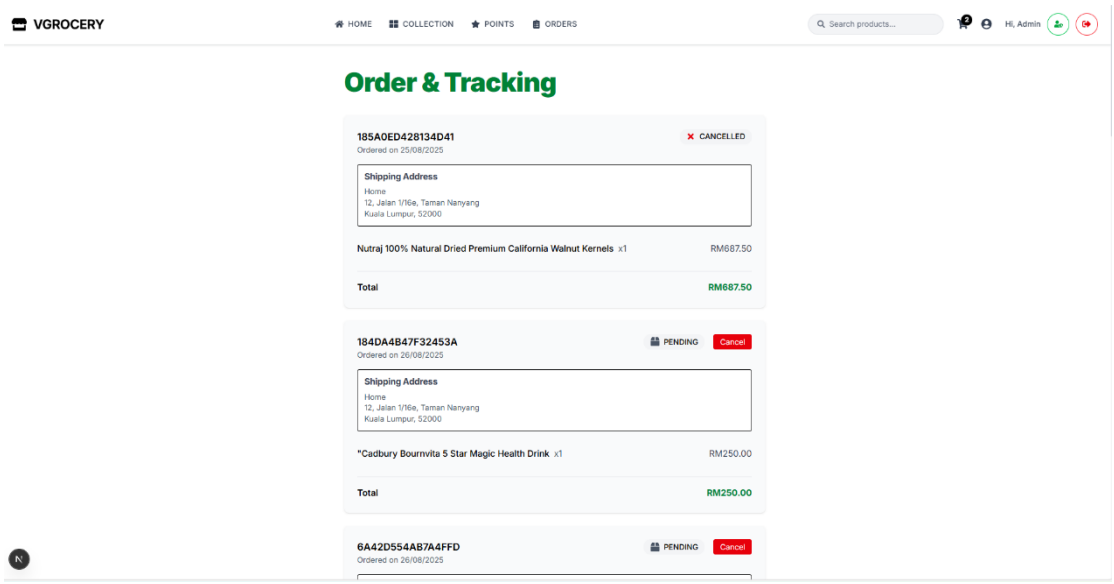


Figure 5.4.2.3 Order Page

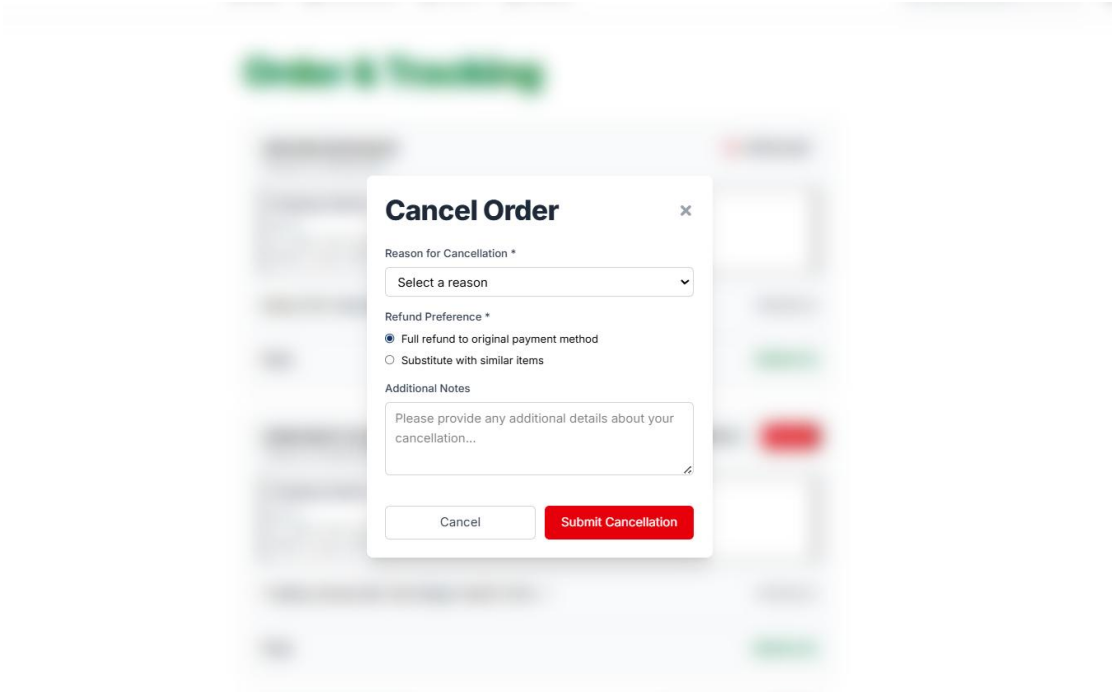


Figure 5.4.2.4 Cancellation Form in Order Page

This page allows customers to view their order and choose to cancel their order with a reason. A summary of the order is displayed to provide a clear information of the order. Customers must fill up the Cancellation form before cancelling an order.

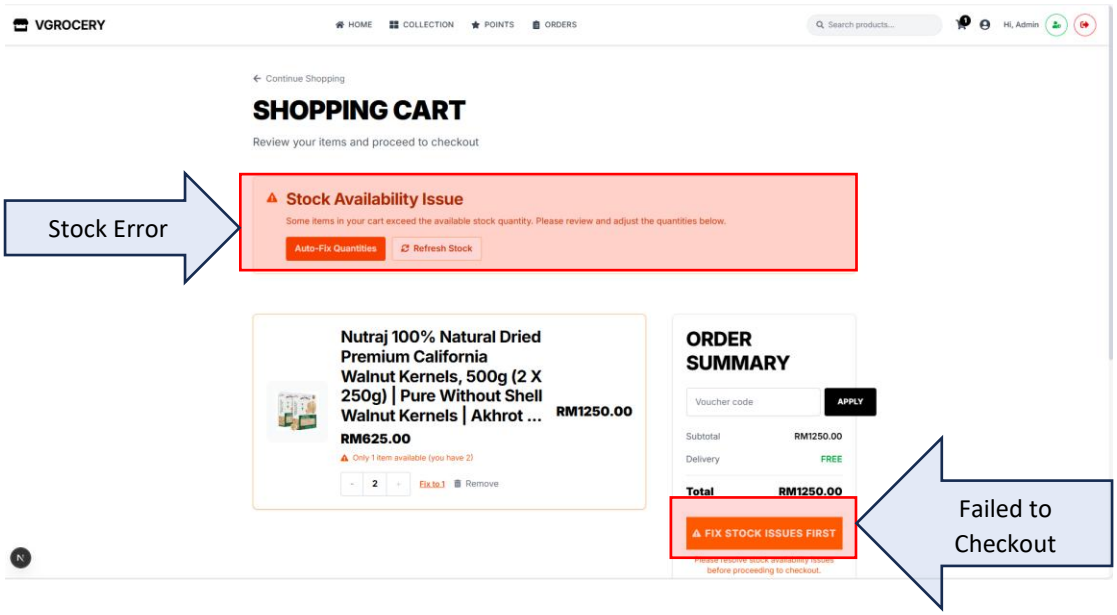


Figure 5.4.2.5 Stocks Error in Cart Page

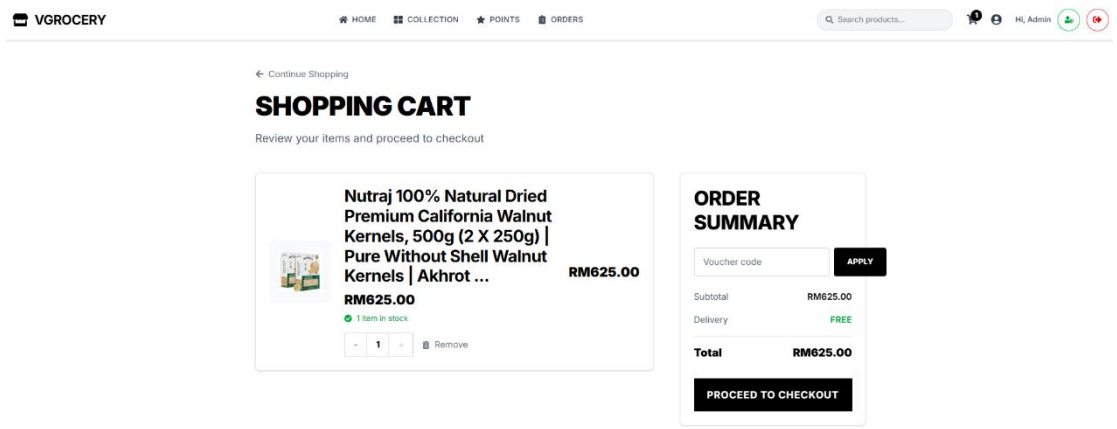


Figure 5.4.2.6 Cart Page Without Error

This page allows customers to check the product they added into cart before proceeding to checkout page. Figure 5.4.2.5 shows error if the desired quantity is larger than the in-stock quantity of the product, customers should adjust the quantity manually or auto fix by clicking the button below the warning. Figure 5.4.2.6 displayed the normal visual and customers can straightly proceed to checkout page.

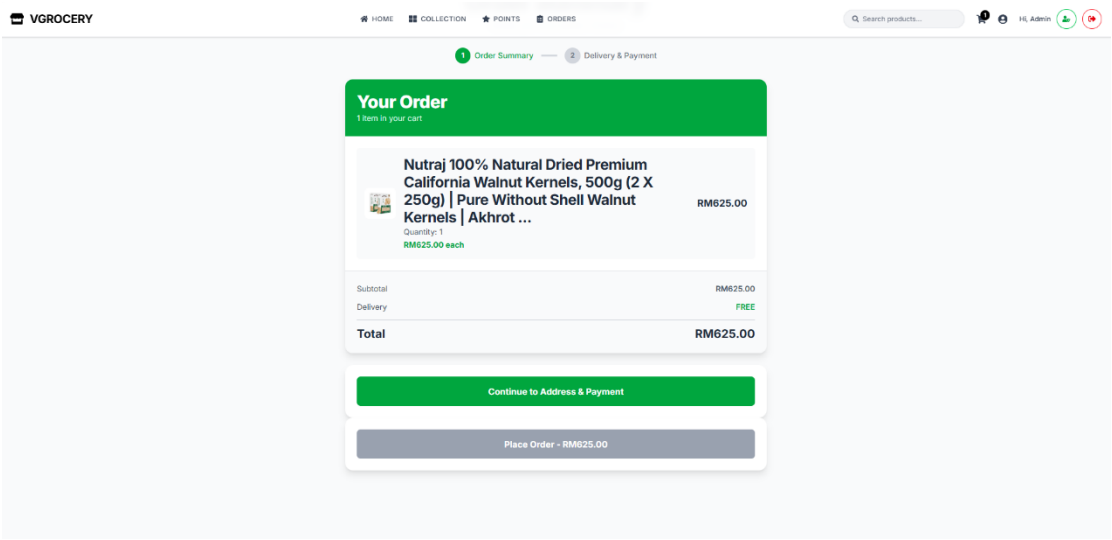


Figure 5.4.2.7 Order Overview in Checkout Page

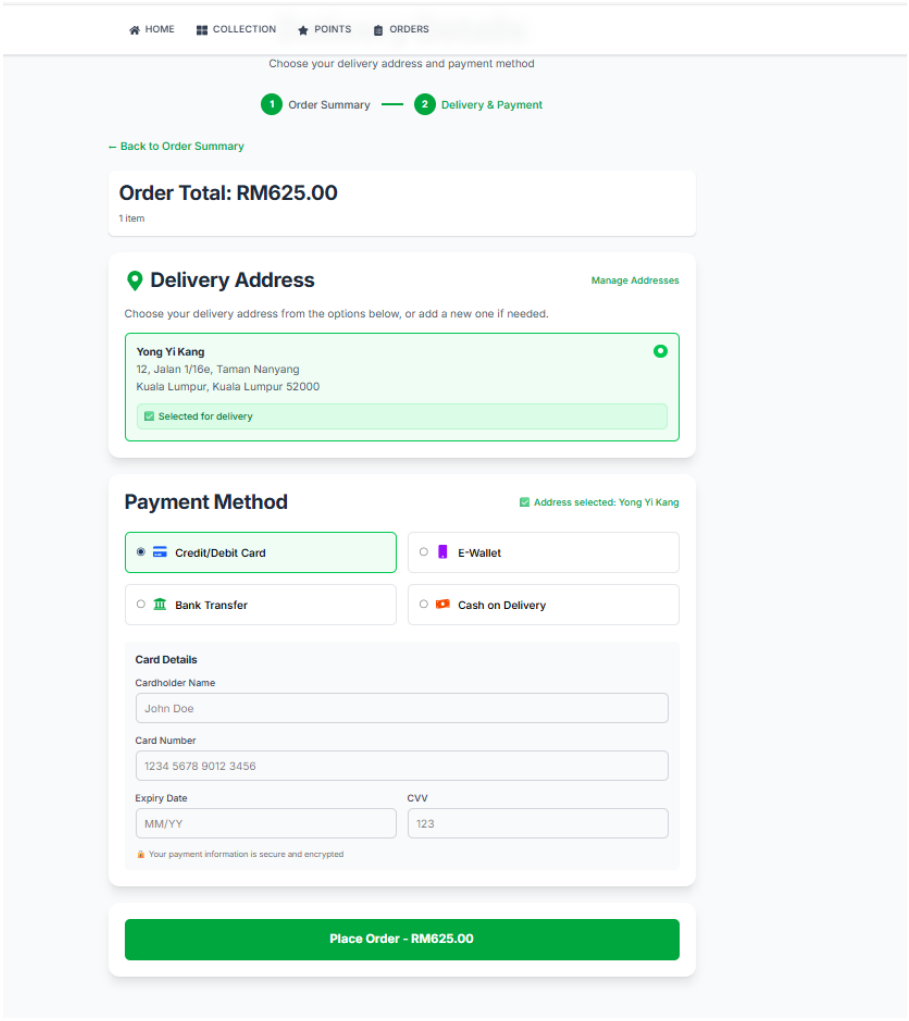


Figure 5.4.2.8 Address and Payment Method in Checkout Page

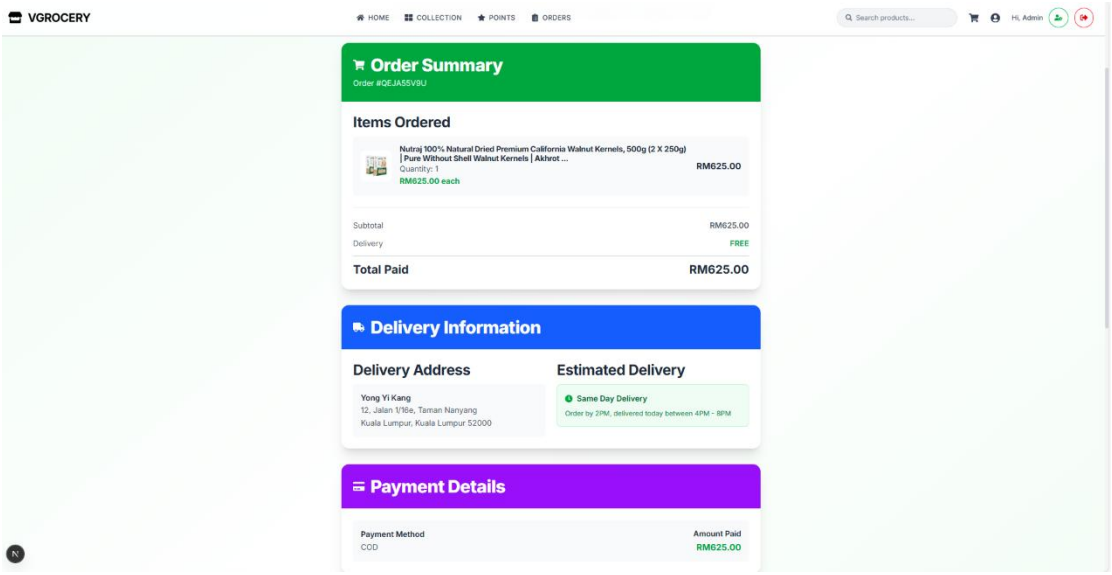


Figure 5.4.2.9 Order Information after Successful Placement

This page allows customers to view again the order summary before proceeding to payment process as shown in Figure 5.4.2.7. After that, customers are required to choose the desired address for the delivery use and choose the payment method, given option Cash on Delivery, Credit or Debit card, and E-wallet. After clicking on place order button, a summary of order, delivery information and payment details are displayed.

5.4.3 Bulk Upload Module and Admin Dashboard

5.4.3.1 Admin Dashboard Page

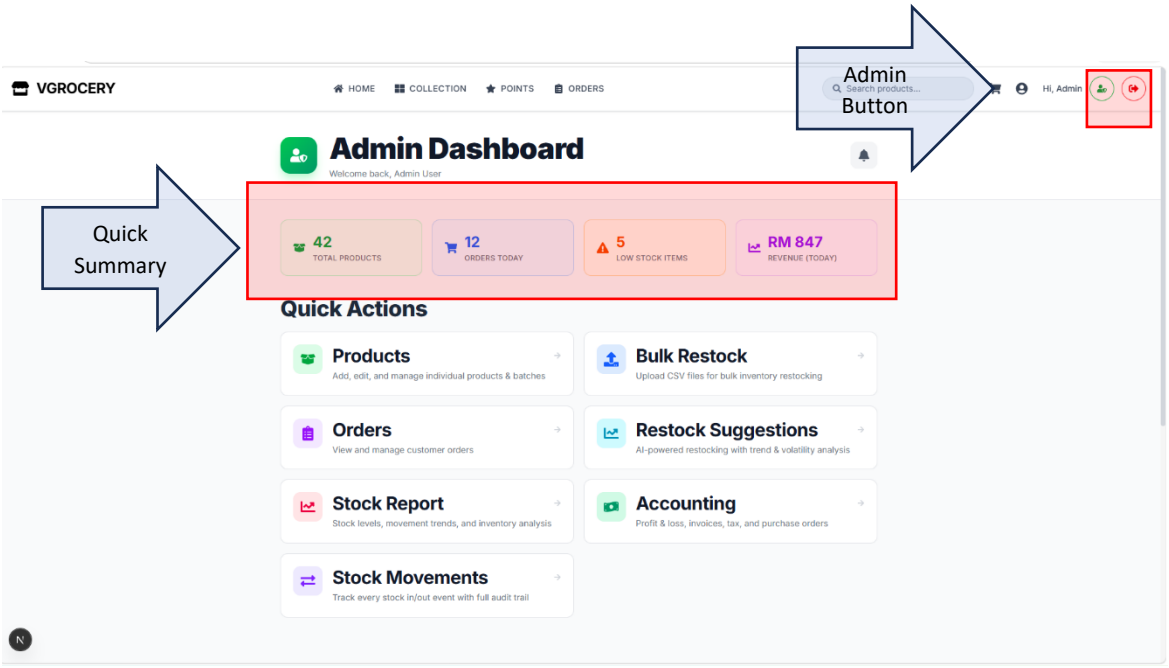


Figure 5.4.3.1 Admin Dashboard Page

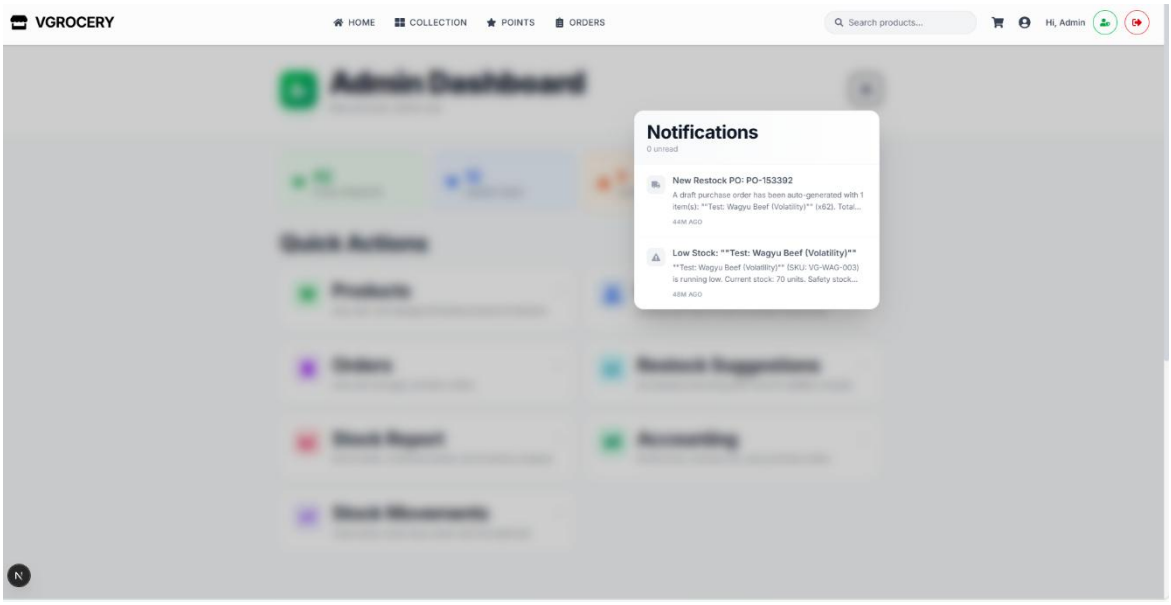


Figure 5.4.3.2 Notification Component to Alert Administrator

This page only can access if the user role is authenticated as admin role after sign in. Admin can click on the green profile icon beside the logout icon at top right of the page to redirect back to admin dashboard page. There are seven admin action buttons including product management, order management, and restocking, stock movements, stock report, accounting and bulk upload functionality as shown in Figure 5.4.3.1. After clicking on the notification icon, the background will be blurred to focus on the notification that alert the administrator to perform restocking order on low stock item as shown in Figure 5.4.3.2.

5.4.3.2 Bulk Upload Page

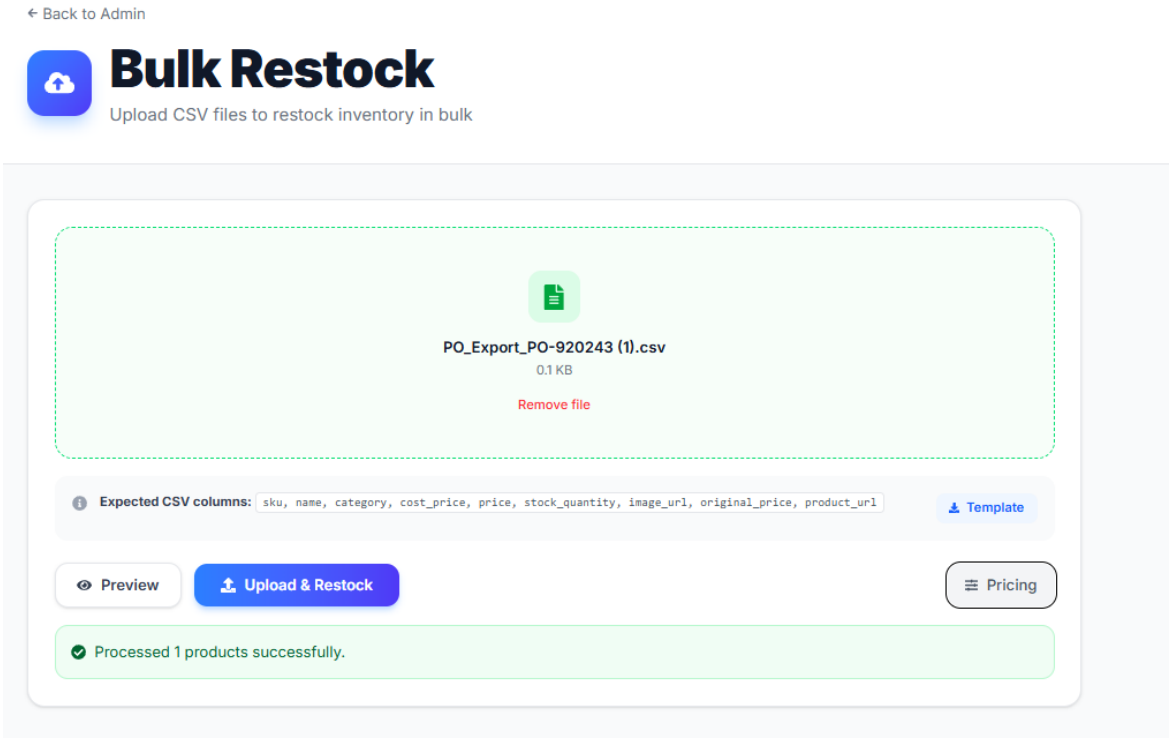


Figure 5.4.3.3 Restock Page with Bulk Upload

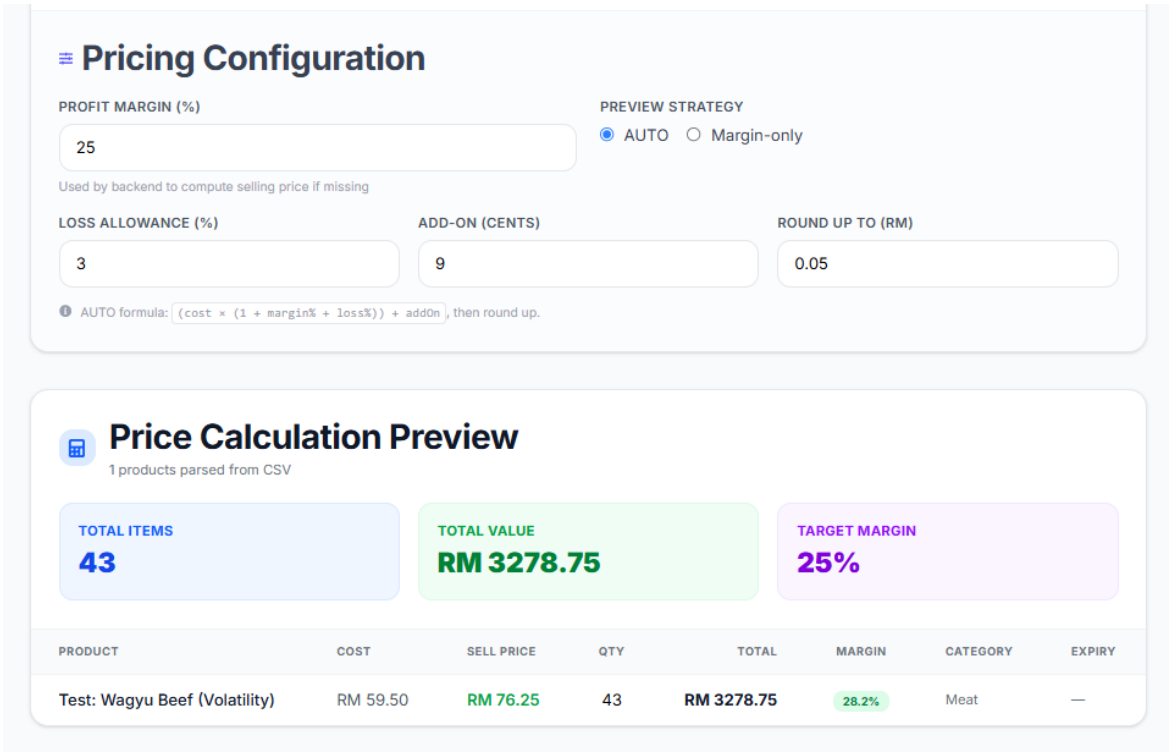


Figure 5.4.3.4 Price Configuration and Preview before Restocking

This page allows administrators to view the low stock alert and perform restocking action by using the bulk inventory upload function. Only csv file can be attached to the bulk upload function and admin can adjust the profit margin, loss allowance, add-on and rounding up option to compute the selling price. The selling price will be computed by the service provider which is Spring boot and admin can choose to preview before uploading the csv file.

5.4.4 FEFO Deduction Module (First Expired, First Out)

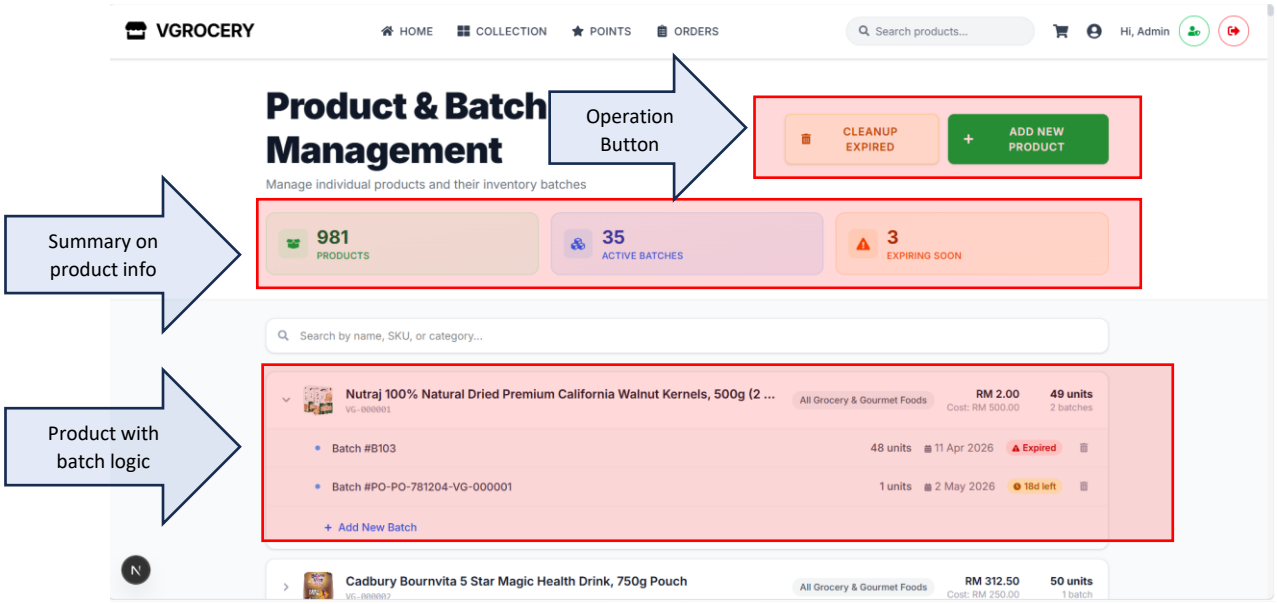


Figure 5.4.4.1 Product and Batch Management Page

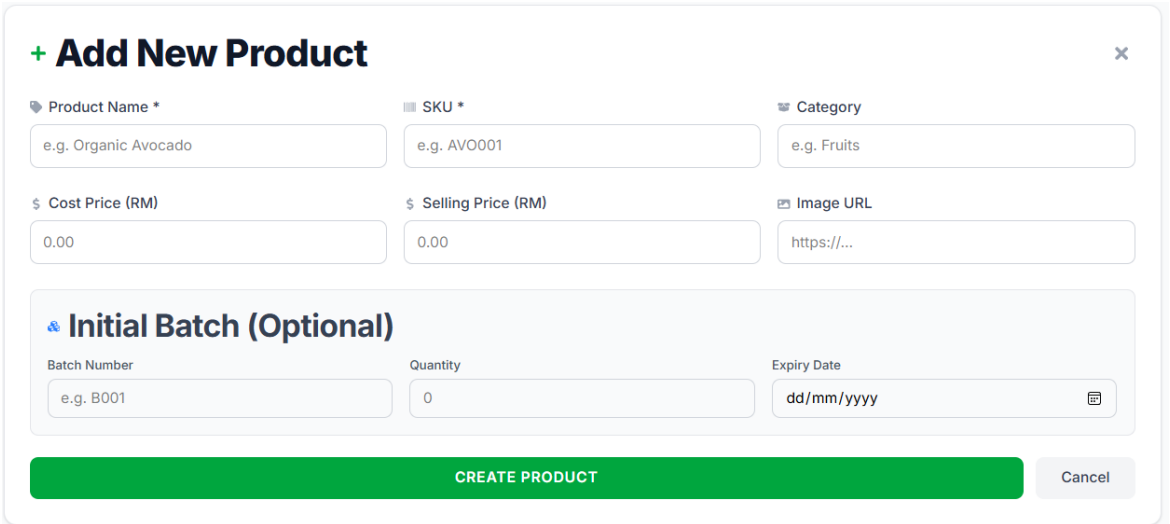


Figure 5.4.4.2 Adding Single Product with Batch Initialization

Figure 5.4.4.3 Deleting/Logging Product with Reason

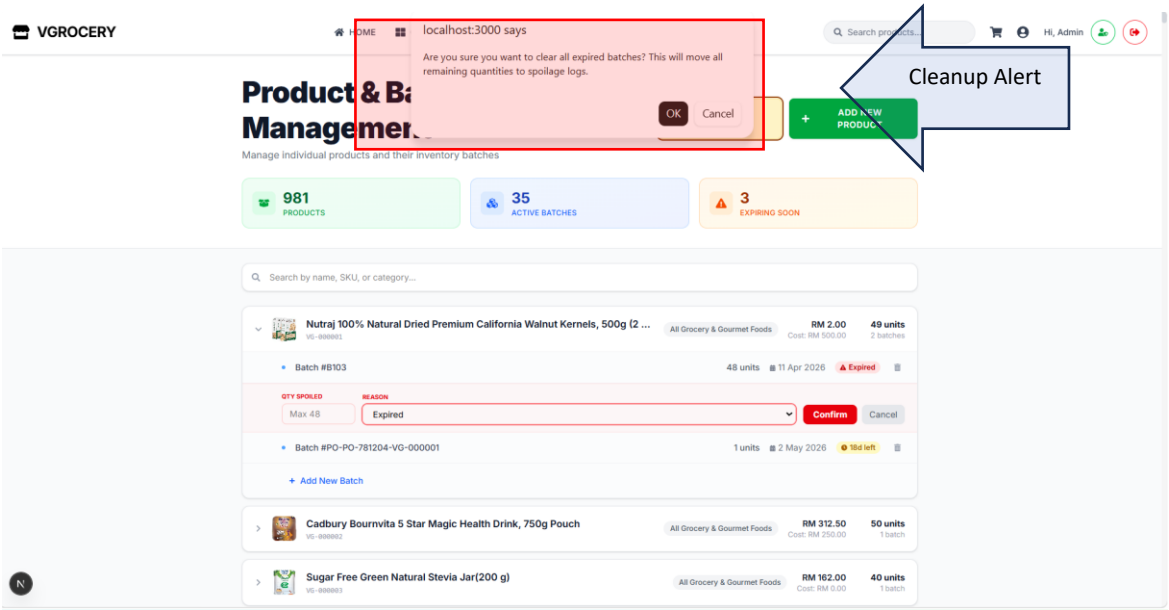


Figure 5.4.4.4 Cleanup Alert

This page allows administrators to add product by clicking the add product button and enter the product information in Figure 5.4.4.2 to add new product. A summary of the product and batches information is shown in Figure 5.4.4.1. Each product has different batches, and the expiry date is shown. Administrators can choose to delete the batches with necessary reason as shown in Figure 5.4.4.3. After clicking on the cleanup expired button, a cleanup alert and the expired batches will be move to spoilage log if confirmation has been done as shown in Figure 5.4.4.4.

5.4.5 Spoilage Logging and Stock Module

5.4.5.1 Stock Movements Page

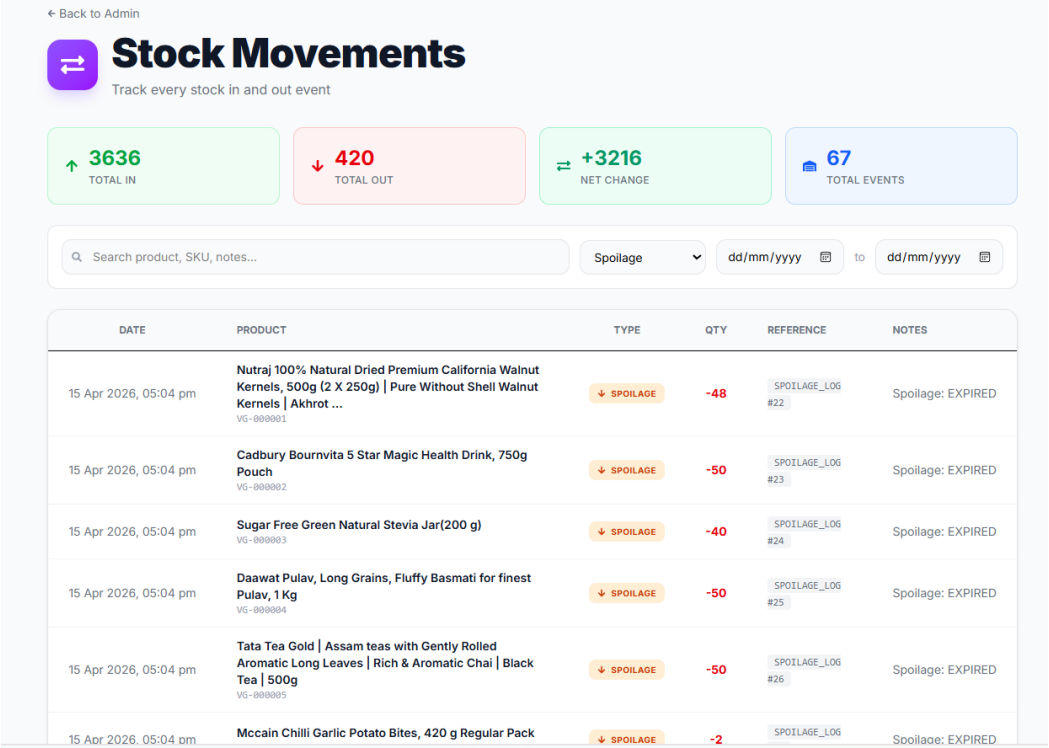


Figure 5.4.5.1 Stock Movements Page Showing Historical Spoilage Log

The Stock Movements page acts as a comprehensive audit trail, logging precisely when and why stock quantities change. Administrators can view a high-level summary of total inbound inventory, total outbound inventory, net change, and total events. The chronological table below tracks every event. Using the built-in dropdown filter, administrators can isolate specific movement types; for example, filtering by "Spoilage" instantly generates the historical Spoilage Report.

5.4.5.2 Stock Reports Page

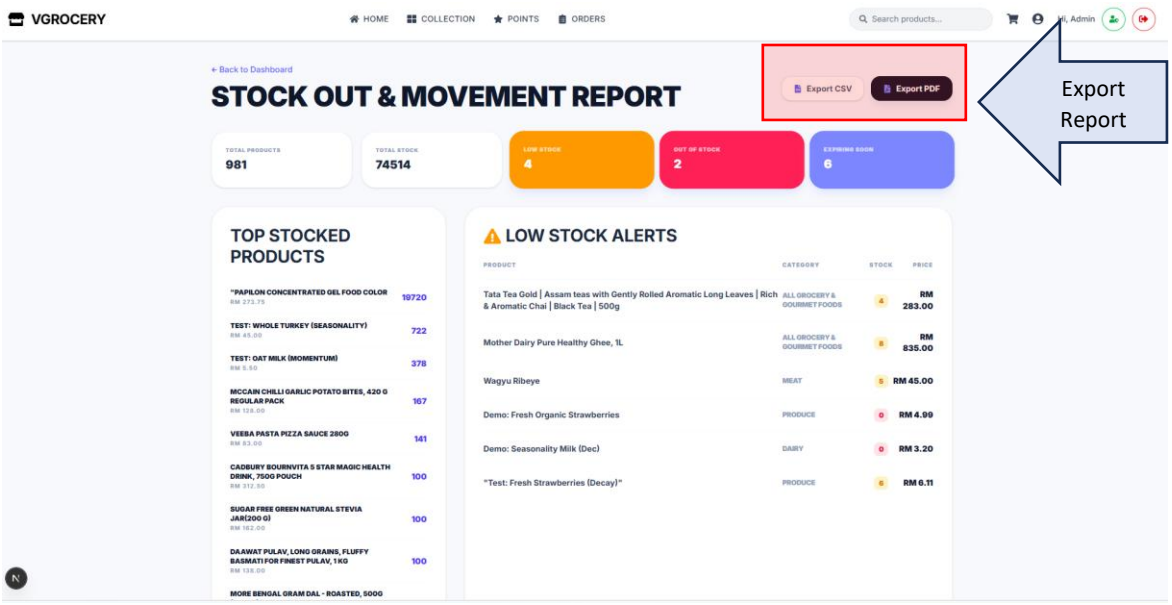


Figure 5.4.5.2 Stock Reports Page with Report Generation

The Stock Reports page provides administrators with an immediate overview of current inventory health. The dashboard calculates the total number of products, while highly emphasizing critical metrics via coloured cards such as "Low Stock," "Out of Stock," and "Expiring Soon." A dedicated panel displays "Low Stock Alerts", warning administrators of specific items that are dangerously close to depletion alongside their current quantities. Administrators can choose to export the stock report in CSV or PDF format by clicking the Export CSV, Export PDF button.

5.4.6 Financial Analytics and Predictive Restocking Module.

To provide administrators with a unified view of business health and inventory forecasting, the financial reports and predictive restocking mechanics are grouped into this comprehensive analytical module.

5.4.6.1 Restock-suggestion page

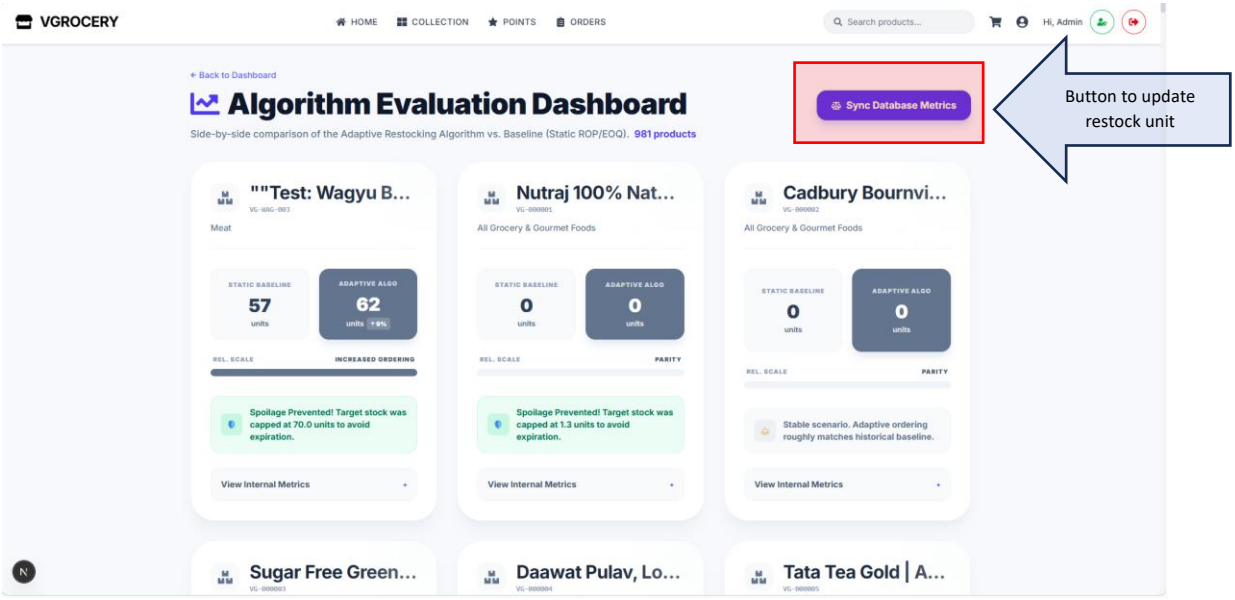


Figure 5.4.6.1 Dashboard Comparing Baseline Static Algorithm and Proposed Adaptive Algorithm

Administrators can click the "Sync Database Metrics" button to forcefully recalculate and update the suggested restock units based on real-time sales velocity and stock movements as shown in Figure 5.4.6.1 The system displays individual product cards detailing the final restocking recommendations as shown in Figure 5.4.6.2. Crucially, the adaptive algorithm incorporates a "Spoilage Prevented" cap to ensure perishable goods are not over-ordered beyond their expected shelf life. Administrators can expand a "View Internal Metrics" section on each card to audit the mathematical variables driving the recommendation, including the decay factor, dynamic Z-score, seasonality, and coefficient of variability.

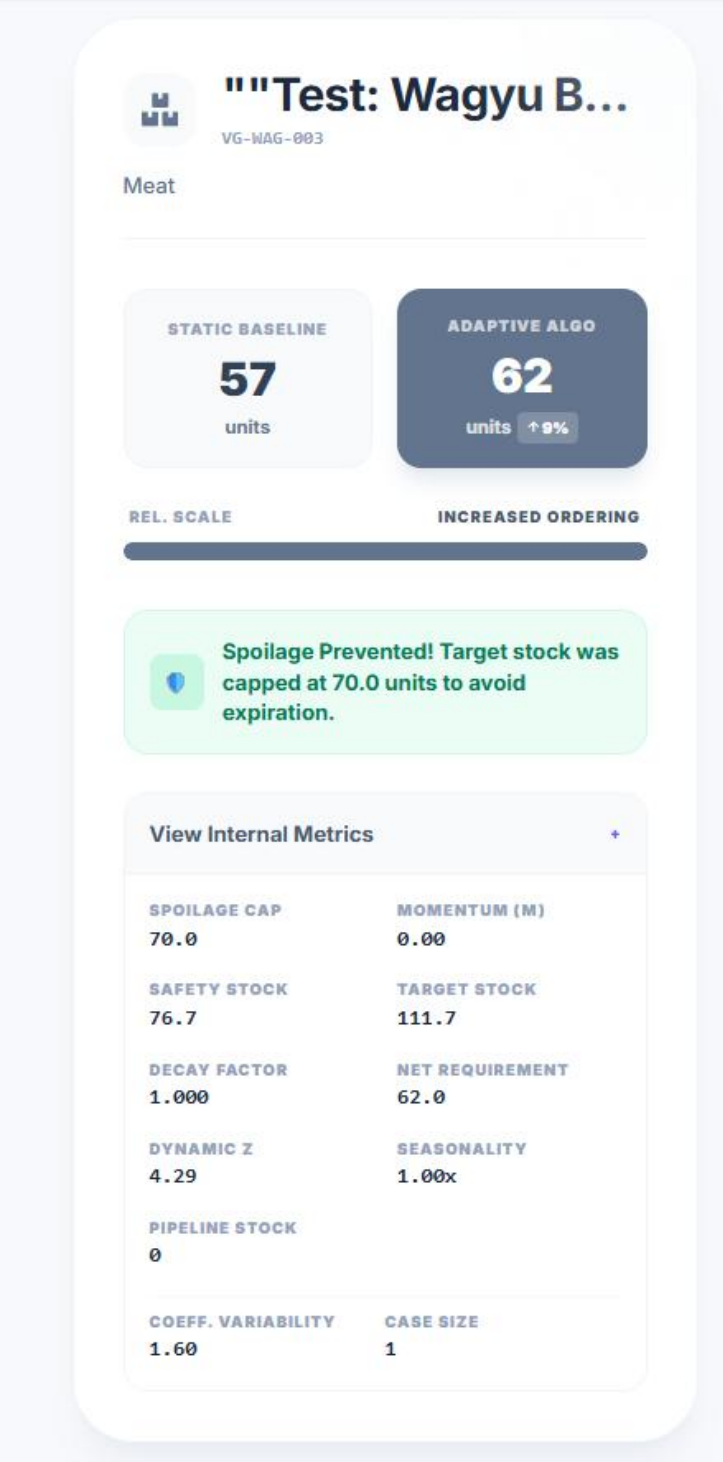


Figure 5.4.6.2 Product Card Showing Internal Metrics Calculation and Spoilage Prevention Capping

5.4.6.2 Accounting Page

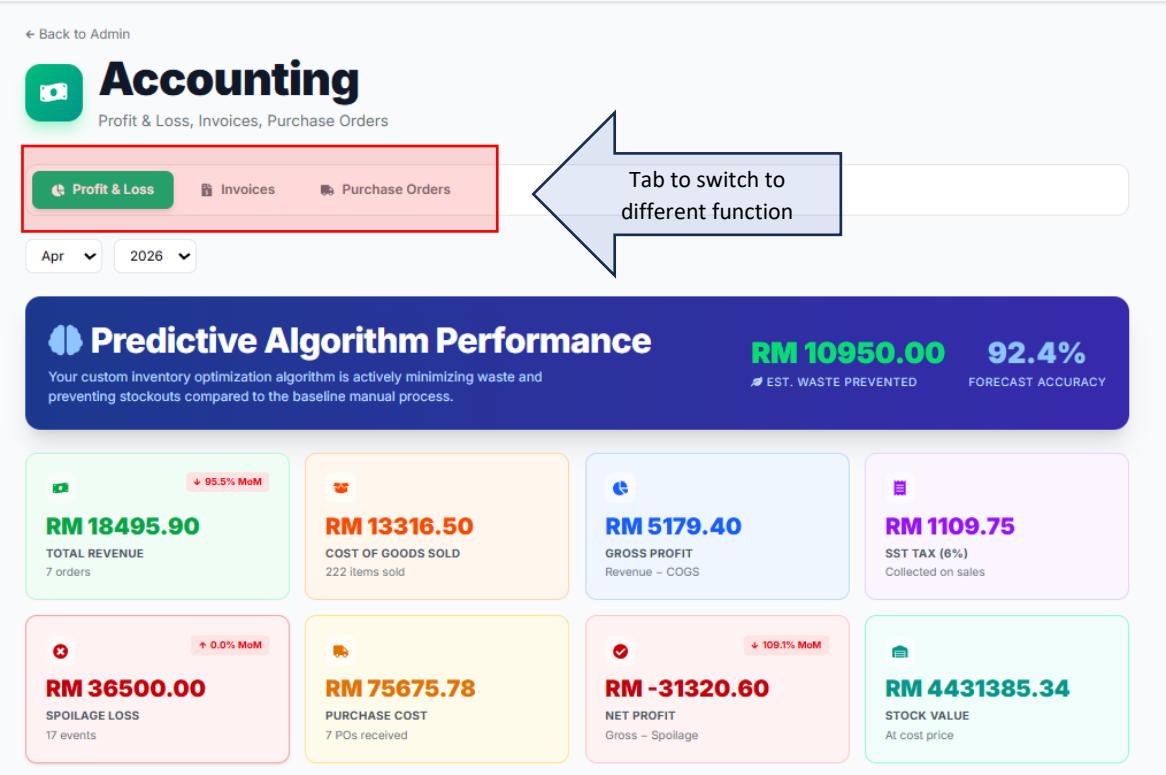


Figure 5.4.6. High-Level Financial Metrics and Predictive Algorithm Performance Tracking

Figure 5.4.6.4 Weekly Financial Trend Bar Chart

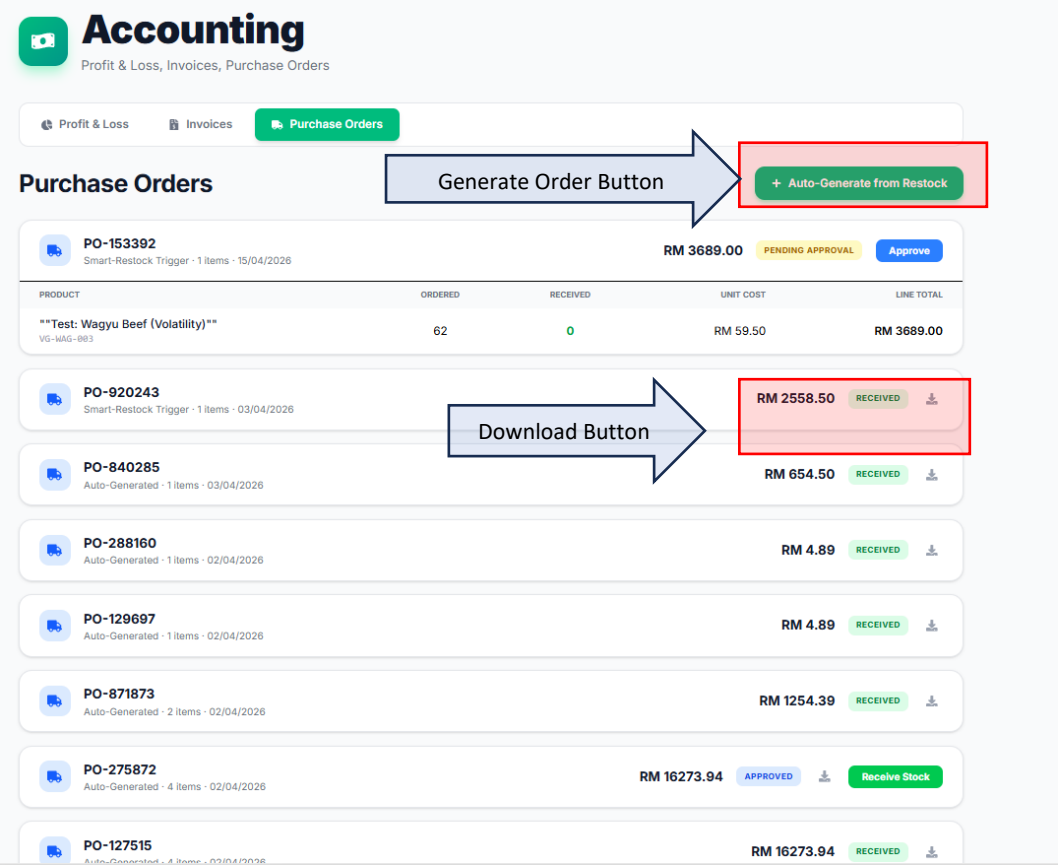


Figure 5.4.6.5 Purchase Order Tracking Interface Showing Predictive Restock Trigger Status

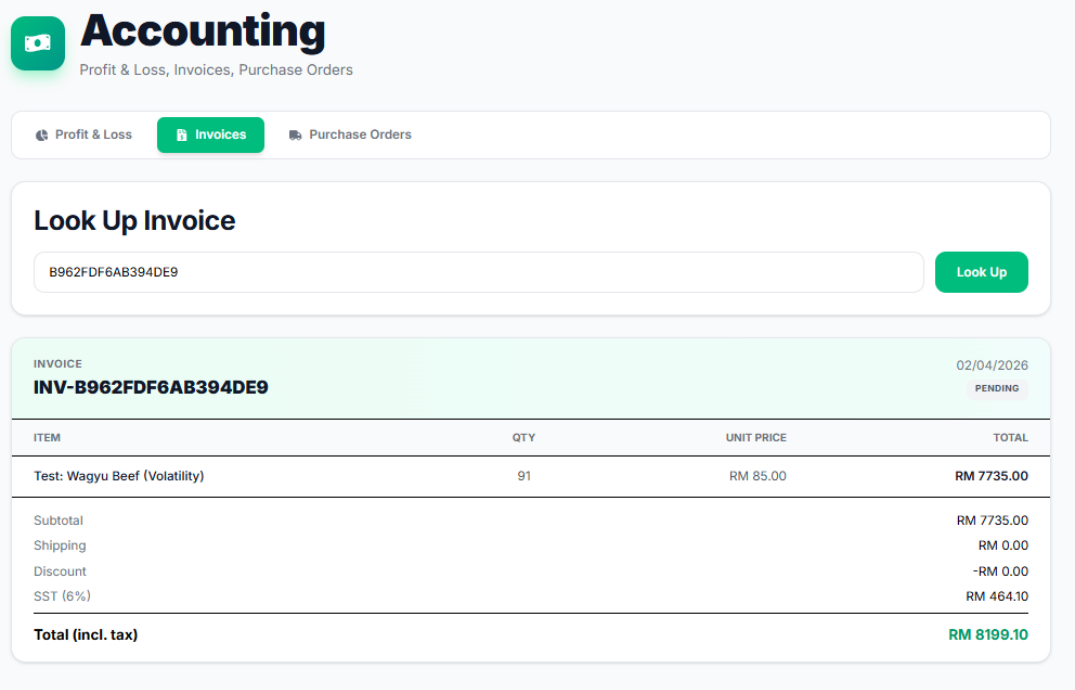


Figure 5.4.6.6 Detailed Invoice View with Subtotal and SST Tax Calculation

This page shows the financial portion of the module consists of an extensive Accounting Dashboard that focuses heavily on revenue tracking, tax compliance, and business health. The dashboard is divided into three primary tabs: Profit & Loss, Invoices, and Purchase Orders. The Profit & Loss tab provides high-level financial summary cards, including Total Revenue, Cost of Goods Sold (COGS), Gross Profit, calculated SST Tax, Spoilage Loss, and Net Profit. Notably, it includes a "Predictive Algorithm Performance" banner that calculates the estimated waste prevented and the forecast accuracy of the system's adaptive algorithm. Administrators can also view financial trends via a "Revenue vs Expenses" bar chart comparing weekly revenue against expenses and spoilage loss.

The Purchase Orders tab logs all restocking actions, differentiating between "Smart-Restock Trigger" POs generated by the predictive algorithm and manual "Auto-Generated" POs. It tracks the status of each PO (e.g., Pending Approval, Approved, Received). Administrator can click on the download icon to download the csv file that can be upload for restocking purpose. Finally, the Invoices tab allows administrators to look up specific invoice IDs to generate detailed tax calculations and transaction summaries for individual sales.

5.4.6.3 Admin Order Management Page

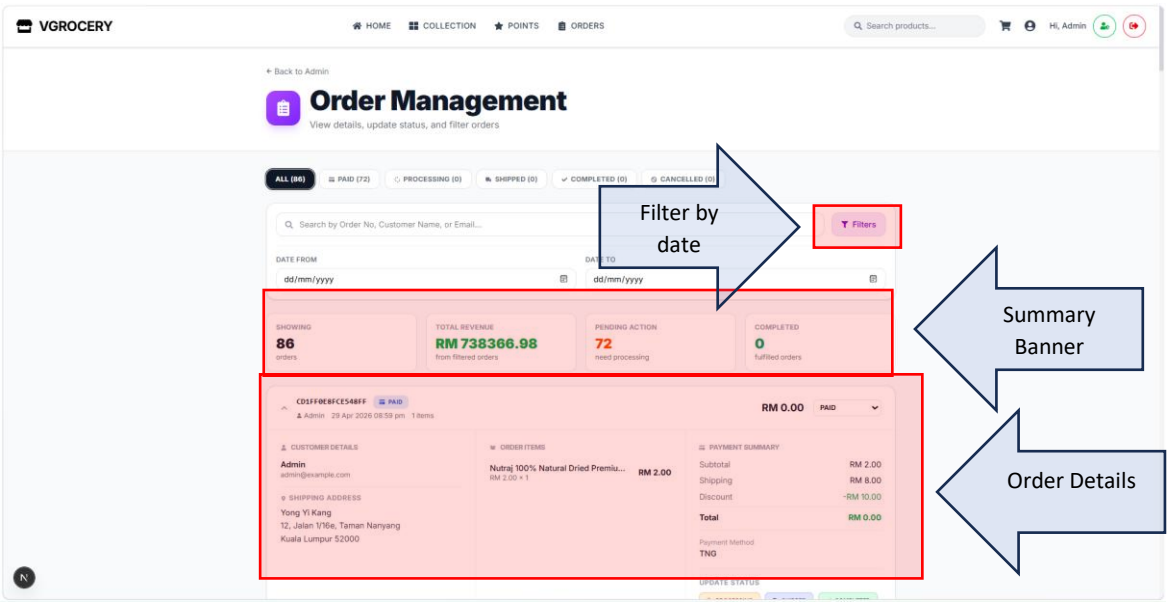


Figure 5.4.6.7 Order Management Page

The Admin Order Management page is the control room for tracking and managing transactions in real time. The user interface is built for efficiency, with an analytics banner

providing a high-level overview of total orders, revenue and actions to be taken, as well as interactive filtering options enabling administrators to search for specific orders through keywords, date ranges and order statuses. For ease of visual display, orders are displayed as a row that can be expanded to reveal detailed transaction information, including a breakdown of the customer's shipping address, a list of products, and a breakdown of the financial details. Moreover, the module is more than a passive reporting interface and allows administrators to update the status of an order (e.g., from PAID to PROCESSING to SHIPPED), which automatically triggers further database updates and moves the order along the e-commerce order processing pipeline.

5.5 Algorithm Implementation

This section presents the core algorithm implemented in the system, which provides a strong foundation for inventory management capabilities. These algorithms make the proposed systems different from conventional e-commerce platforms by using expiry-aware stock handling and data-driven restocking decisions.

Three key algorithms are implemented within the Spring Boot backend:

- FEFO Batch Deduction Algorithm
- Adaptive Restocking Algorithm
- Baseline (Static ROP) Restocking Algorithm

The adaptive restocking algorithm is evaluated against the baseline model through the administrative dashboard to demonstrate its effectiveness in reducing spoilage and preventing stockouts.

5.5.1 FEFO Batch Deduction Algorithm

The First-Expired, First-Out (FEFO) algorithm ensures that inventory is deducted based on the earliest expiry date during the checkout process. This approach is particularly important for perishable goods, as it minimises waste and ensures optimal stock rotation.

The algorithm is implemented in the InventoryService.deductStockFEFO() method at the backend level.

```
@Transactional
public void deductStockFEFO(Long productId, int quantity) {
    if (quantity <= 0)
        return;

    LocalDate today = LocalDate.now();
    List<Batch> batches = batchMapper.findValidBatchesForFefo(productId, today);

    int totalAvailable = batches.stream().mapToInt(Batch::getAvailableQuantity).sum();
    if (totalAvailable < quantity) {
        throw new InsufficientStockException("Insufficient total stock for product ID: " + productId +
                ". Requested: " + quantity + ", Available: " + totalAvailable);
    }

    int remainingDeduction = quantity;
    for (Batch batch : batches) {
        int deductFromThisBatch = Math.min(batch.getAvailableQuantity(), remainingDeduction);

        batch.setAvailableQuantity(batch.getAvailableQuantity() - deductFromThisBatch);
        batchMapper.updateById(batch);

        remainingDeduction -= deductFromThisBatch;
        if (remainingDeduction == 0)
            break;
    }

    // Synchronize product table's total stock (optional but good for display)
    productMapper.decrementStock(productId, quantity);

    // Log stock movement
    if (stockMovementService != null) {
        stockMovementService.logMovement(
                productId, null, "ORDER_DEDUCT",
                quantity, "ORDER", null,
                "FEFO deduction: " + quantity + " units");
    }

    // TRIGGER REAL-TIME RESTOCK CHECK
    if (purchaseOrderService != null) {
        purchaseOrderService.triggerRealTimeRestock(productId);
    }
}
```

Figure 5.5.1 Code snippet for FEFO deduction logic

Design Rationale

- Expiry-Aware Prioritisation:
  Batches are sorted by ascending expiry date, ensuring that products closest to expiration are always sold first.
- Greedy Deduction Strategy:
  The algorithm follows a greedy approach by fully utilizing each batch before proceeding to the next, ensuring efficient stock utilisation.
- Spill-Over Handling:
  If a single batch cannot fulfil the requested quantity, the algorithm automatically continues deduction across subsequent batches, maintaining batch-level accuracy.
- Transactional Integrity:
  The method is annotated with @Transactional, ensuring atomic execution. Any failure during the process results in a complete rollback, preserving database consistency.
- Real-Time Integration:
  After successful deduction, a restocking trigger is activated, enabling immediate evaluation of inventory thresholds.

5.5.2 Adaptive Restocking Algorithm

The adaptive restocking algorithm is designed to overcome the limitations of traditional static inventory models. It dynamically adjusts restocking recommendations based on demand trends, variability, seasonality, and product perishability.

The algorithm is implemented in the RestockOptimizer.calculateRestock() method.

Algorithm Overview

The model is composed of eight key computational steps:

Step A: Demand Momentum

Short_Term_Avg_3d = Total_Sales_Last_3_Days / 3

Long_Term_Avg_30d = Total_Sales_Last_30_Days / 30

If Long_Term_Avg_30d = 0 and Short_Term_Avg_3d = 0:

M = 0

If Long_Term_Avg_30d = 0 and Short_Term_Avg_3d > 0:

M = 1

Otherwise:

M = (Short_Term_Avg_3d - Long_Term_Avg_30d) / Long_Term_Avg_30d

To prevent extreme recommendations, the momentum value is capped between -1 and 1:

M_capped = MIN(MAX(M, -1), 1)

This step measures whether recent demand is increasing or decreasing compared with the longer-term sales average. By comparing the 3-day average demand with the 30-day average demand, the system can detect short-term demand changes. The momentum value is capped between -1 and 1 to prevent extreme restocking recommendations caused by unusual sales spikes or abnormal demand drops.

Step B: Adjusted Demand

If Long_Term_Avg_30d = 0:

Base_Demand = Short_Term_Avg_3d

Otherwise:

Base_Demand = Long_Term_Avg_30d

$$D_{adj} = Base\_Demand \times (1 + \beta \times M_{capped}) \times Seasonality\_Factor$$

where $\beta = 0.8$.

This step calculates the adjusted daily demand used by the adaptive restocking algorithm. The base demand is adjusted using the momentum factor and seasonality factor so that the system can respond more quickly to changing sales behaviour. The β value controls how strongly the algorithm reacts to recent demand changes.

Step C: Volatility-Based Safety Stock

If Base_Demand = 0:
CV = 0
Otherwise:
$CV = \sigma / Base\_Demand$
$Z_{dynamic} = 1.65 \times (1 + CV)$

To avoid excessive safety stock, Z_dynamic is capped within a reasonable range:
$Z_{dynamic} = MIN(MAX(Z_{dynamic}, 1.28), 2.33)$
$SS_{adaptive} = Z_{dynamic} \times \sigma \times \sqrt{(Lead\_Time + Review\_Period)}$

This step adjusts the safety stock based on demand uncertainty. The coefficient of variation measures how unstable the demand is by comparing the standard deviation with the average demand. Products with higher demand volatility receive a higher safety stock buffer, while stable products require less buffer stock.

Step D: Adaptive Target Stock

$$Target\_Stock_{adaptive} = D_{adj} \times (Lead\_Time + Review\_Period) + SS_{adaptive}$$

This step calculates the target stock level required to cover expected demand during the lead time and review period. The adjusted demand is multiplied by the total coverage period, and the adaptive safety stock is added as a buffer. This gives the system a recommended stock level before applying perishability constraints.

Step E: Perishability Constraint

Max_Sellable_Qty = D_adj × Usable_Shelf_Life_Days

Safe_Target_Stock = MIN(Target_Stock_adaptive, Max_Sellable_Qty)

This step prevents the system from recommending more stock than can realistically be sold before expiry. The maximum sellable quantity is calculated using adjusted demand and usable shelf life. The safe target stock is then capped so that the recommended inventory level does not exceed the expected sellable quantity within the product's shelf life.

Step F: Net Requirement

Inventory_Position = Current_Stock + Incoming_Stock - Reserved_Stock

Net_Requirement = MAX(0, Safe_Target_Stock - Inventory_Position)

This step calculates the actual stock gap after considering current stock, incoming stock, and reserved stock. Inventory position gives a more realistic view of available inventory than current stock alone. The net requirement ensures that the system only recommends restocking when the safe target stock is higher than the available inventory position.

Step G: Decay Adjustment

Expected_Storage_Days = MIN(Lead_Time + Review_Period, Usable_Shelf_Life_Days)

Decay_Factor = e^(-λ × Expected_Storage_Days)

Raw_Order_Qty = Net_Requirement × Decay_Factor

This step reduces the recommended restock quantity based on expected spoilage risk. The decay factor is calculated using the expected storage period and spoilage rate. Products with higher spoilage risk or longer expected storage duration will receive a lower raw order quantity, helping to reduce unnecessary waste.

Step H: Practical Ordering Constraints

If Raw_Order_Qty <= 0:

Final_Order_Qty = 0

Otherwise:

Final_Order_Qty = CEIL(Raw_Order_Qty)

If Case_Size is used:

Final_Order_Qty = CEIL(Final_Order_Qty / Case_Size) × Case_Size

If Supplier_MOQ is used and Final_Order_Qty < Supplier_MOQ:

If Supplier_MOQ <= Net_Requirement:

Final_Order_Qty = Supplier_MOQ

Otherwise:

Final_Order_Qty = Final_Order_Qty

MOQ_Risk_Flag = TRUEKey Strengths

This step converts the raw order quantity into a practical order quantity for business use. The system rounds the quantity upward and adjusts it according to case size or supplier minimum order quantity where applicable. This ensures that the final recommendation is not only mathematically valid, but also operationally feasible for purchasing and restocking.

Enhancements

- Accounts for uncertainty and volatility
- Prevents overstocking of perishable goods
- Integrates business constraints (MOQ, case size)
- Adapts to real-time demand changes

5.5.3 Baseline (Static ROP) Restocking Algorithm

The baseline model serves as a control mechanism for evaluating the effectiveness of the adaptive algorithm. It follows a traditional Reorder Point (ROP) approach using fixed parameters and historical averages.

The algorithm is implemented in BaselineRestockOptimizer.calculateBaseline().

Mathematical Model

Avg_Demand_30d = Total_Sales_Last_30_Days / 30

Safety_Stock_base = $Z \times \sigma \times \sqrt{(Lead\_Time + Review\_Period)}$

Target_Stock_base = Avg_Demand_30d × (Lead_Time + Review_Period) +
Safety_Stock_base

Inventory_Position = Current_Stock + Incoming_Stock - Reserved_Stock

Net_Requirement_base = Target_Stock_base - Inventory_Position

Order_Qty_base = MAX(0, CEIL(Net_Requirement_base))
where Z = 1.65 for an approximate 95% service level.

Limitations

- Does not consider short-term demand momentum
- Uses a fixed service-level approach
- Ignores product perishability constraints
- Does not apply spoilage or decay adjustment
- May lead to overstocking or stockouts for perishable products

5.6 Implementation Issue and Challenge

In Sprints 1-12, several problems were experienced during the development of the Integrated E-Commerce Inventory System. These included mainly the issues associated with combining the various technologies used for creating this system and ensuring that data consistency and reliable operation were maintained.

The first problem was the CORS restriction during Sprint 2. Since the request made by the frontend to the backend needed to be from an origin other than the one specified, the Next.js frontend had problems communicating with the Spring Boot backend. However, this was fixed via the implementation of global CORS configuration in the backend code by specifying WebMvcConfigurer. This allowed for the requests from the frontend domain (http://localhost:3000).

There was yet another problem identified in Sprint 5 of bulk uploading module. Initially, in the CSV upload process, we had an all-or-nothing process, in which the existence of a single erroneous record caused failure in the entire upload process. It caused usability problems with bulk data uploads. As a solution to this, we changed the design of InventoryService class in such a way that we used row level exception handling and thus managed to have successful processing of good records while isolating erroneous rows for further analysis.

In Sprint 7 and 8, there were several difficulties associated with the FEFO (First-Expired, First-Out) batch deduction algorithm. When the requested order quantity was more than the quantity in a batch, we had inconsistencies in our batch-wise data records. This was solved through sequential iteration-based batch deduction approach. The algorithm deducts the quantity from different batches if required based on the expiry dates of batches.

Sprint 9 brought forward concerns around transaction integrity and concurrent processing. This involved situations where several activities were going on at once, including ordering and tracking any spoilages, and there was always the potential for errors to be caused by race conditions, ultimately causing problems with inventory tracking or even causing ghost inventory to appear. To address this problem, Spring @Transactional was used to ensure that certain actions within the service layer were done atomically.

Finally, there were challenges associated with stateless authentication in Sprint 1, involving the issue of JSON Web Tokens expiring and the frontend doing nothing about it. This was resolved by using an Axios interceptor to catch unauthorised requests on the server-side and redirecting users back to the login screen.

Overall, these problems show the difficulty of creating an entire stack application which would include features like real-time work, communication encryption, and smart inventory control. In any case, dealing with these issues greatly contributed to our knowledge of concurrent processing, error handling, and user interface designing.

5.7 Summary

The implementation phase successfully translated the proposed system design into a fully functional full-stack application. All core features, including user authentication, product management, and checkout processes, were effectively developed and integrated. In addition, advanced inventory control mechanisms such as FEFO-based stock deduction, spoilage logging, and predictive restocking were successfully implemented to enhance system intelligence and operational efficiency.

Throughout the development process, various technical challenges were encountered, particularly in areas involving system integration, data consistency, and concurrent operations. These challenges were systematically addressed through appropriate design decisions and technical solutions, resulting in a stable and reliable system.

Overall, the completed implementation demonstrates that the proposed system can support real-time inventory management and decision-making. The system is therefore ready for further evaluation and performance analysis, which will be presented in the subsequent chapter.

Chapter 6 System Evaluation and Discussion

This chapter discusses the evaluation and testing of the Integrated E-Commerce Inventory System. This chapter is to ensure that the proposed system meets the functional and non-functional requirements as specified in the previous chapters. The assessment is primarily targeted at the e-commerce and the enhanced inventory management modules, which include authentication, checkout, FEFO batch deduction, spoilage logging, predictive restocking, bulk CSV upload, financial reporting, and security control. Black-box testing, integration testing, security testing and performance evaluation were implemented to carry out the testing activities. Furthermore, some user acceptance testing activities were conducted to assess the usability of the system from the point of view of the customers and the administrators. We highlight and assess the outcome based on the project requirements to assess the system's success.

6.1 System Testing and Performance Metrics

A robust testing approach was employed to test the functional and non-functional features of the Integrated E-Commerce Inventory System. The strategy involved black-box testing, integration testing, security testing and quantitative performance analysis.

Functional requirements of the system were tested using black-box testing from the viewpoints of users and administrators. It validated the input-output behaviour of the system without knowledge of its internal structure. The system tested the registration process, login, bulk upload (CSV), spoilage, checkout, and viewing financial reports. The findings verified that the system generated the right results for valid inputs and displayed errors or warnings for invalid or edge-case inputs through messages and input validation to prevent errors.

Integration testing tested the interaction between the front end, back end and database. This was critical for operations like checkout, first-expiry-first-out (FEFO) stock reduction, stock movement auditing, and stock prediction. This ensured data was being correctly passed from the Next.js frontend to the Spring Boot backend and MySQL database.

Security testing was also performed due to the use of JWTs and role-based security. This was to test whether the system was able to secure data and resources. This involved testing invalid user login, token expiry, unauthorised access to administrator features and invalid file uploads.

We conducted quantitative evaluation to assess the system's performance in terms of response time, consistency and efficiency. This involved assessing key system functions including checkout response time, processing bulk CSV data, consistency of FEFO data, and export success rate. These were used to assess whether the system can handle the anticipated workload for an SME grocery e-commerce system.

6.2 Testing Environment

Testing was performed to ensure that the system operates correctly under expected development and operational conditions. Table 6.2 outlines the hardware, software, and data environment used during the testing phase.

Table 6.2 Testing Environment Summary

Component	Specification / Tool
Operating System	Windows 11
Frontend	Next.js 14, React 18
Backend	Spring Boot 3.2.x, Java 17
Database	MySQL 8.0
API Testing Tool	Postman
Browser Used	Google Chrome
Database Validation Tool	MySQL Workbench
Test Dataset	Product, batch, order, spoilage, and sales history records
CSV Test Size	Small file, invalid file, partial-error file, and 500+ SKU file

The testing environment was selected to match the development setup used throughout the project. Postman was used to validate backend API request and response behaviour, while Google Chrome was used to test user interface interactions. MySQL Workbench was used to inspect database records after important operations such as checkout, FEFO deduction, spoilage logging, and CSV upload.

6.3 Testing Setup and Result

The system was tested module by module. The following tables outline the critical test cases executed, their expected outcomes, and the achieved results.

6.3.1 User Authentication Module Testing

Table 6.3.1 User Authentication Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-001	Valid Registration	Enter valid name, email, and password on the registration page.	Account created successfully; user redirected to the login page.	Pass
TC-002	Duplicate Email Registration	Attempt to register with an email that already exists in the system.	System rejects registration and displays “Email already exists” error.	Pass
TC-003	Valid Login	Enter correct email and password on the login page.	JWT token generated; user redirected to the home page or admin dashboard based on role.	Pass
TC-004	Invalid Login Credentials	Enter incorrect password with a valid email.	System displays “Invalid email or password” error and denies access.	Pass
TC-005	Session Expiry Handling	Allow JWT token to expire and attempt an API request.	Frontend Axios interceptor catches 401 response and redirects user to login page.	Pass
TC-006	Role-Based Access Control	A customer account attempts to access admin-only API endpoints.	System returns 403 Forbidden; admin functions are inaccessible to customer roles.	Pass

6.3.2 Product Checkout Module Testing

Table 6.3.2 Product Checkout Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-007	Checkout with Valid Stock	Complete checkout with items that have sufficient stock.	Order is created successfully; stock is deducted through FEFO; confirmation page is displayed.	Pass
TC-008	Checkout Exceeding Stock	Attempt to checkout with an item quantity exceeding available stock.	System prevents checkout and displays an “Insufficient stock” error.	Pass

6.3.3 FEFO Deduction Module Testing

Table 6.3.3 FEFO Deduction Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-009	Single-Batch Deduction	Order 5 units of a product with Batch A containing 10 units.	System deducts 5 units from Batch A only; Batch A is updated to 5 units remaining.	Pass
TC-010	Multi-Batch Spill-Over	Order 15 units. Batch A has 10 units and Batch B has 20 units.	Batch A is exhausted to 0; 5 units are deducted from Batch B.	Pass
TC-011	Insufficient Total Stock	Order 50 units when total available stock across all batches is 30.	System throws an insufficient stock error; no batch quantities are modified.	Pass

TC-012	Expired Batch Exclusion	Product has a batch with an expiry date in the past.	Expired batch is excluded from FEFO deduction.	Pass
TC-013	Audit Trail Accuracy	Verify database after FEFO deduction.	stock_movements table contains an ORDER_DEDUCT entry with correct product ID and quantity.	Pass

6.3.4 Spoilage and Damage Logging Module Testing

Table 6.3.4 Spoilage Logging Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-014	Manual Spoilage Recording	Log spoiled items from a selected inventory batch with the reason "Expired."	Batch quantity is reduced, and a "SPOILAGE" event is added to the Stock Movements log.	Pass
TC-015	Over-Quantity Spoilage	Attempt to log 20 spoiled units from a batch that only has 10 available units.	System rejects the action with an "Insufficient quantity in batch" error, and no inventory data is modified.	Pass
TC-016	Bulk Cleanup of Expired Inventory	Click the "Cleanup Expired" button when multiple batches have passed their expiry dates.	Expired batches are cleared from active inventory, spoilage records are created, a success notification is displayed, and the Profit and Loss dashboard is updated.	Pass
TC-017	Financial Impact of Spoilage	View the monetary value of total spoiled items in the financial dashboard.	Spoilage cost is accurately calculated and reflected in the Profit and Loss ledger.	Pass

6.3.5 Predictive Restocking Algorithm Testing

Table 6.3.5 Predictive Restocking Algorithm Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-018	Adaptive vs Baseline Comparison	Run both algorithms on a product with high short-term demand momentum.	Adaptive algorithm suggests a higher restock quantity than the baseline algorithm.	Pass
TC-019	Perishability Cap Activation	Product has short usable shelf life and moderate demand.	Adaptive algorithm caps the order quantity using Max_Sellable_Qty = D_adj × Usable_Shelf_Life_Days.	Pass
TC-020	Zero-Sales Edge Case	Product has 0 sales in both 3-day and 30-day windows.	Algorithm handles zero demand safely and returns 0 restock quantity without error.	Pass
TC-021	Decay Factor Impact	Product has high spoilage risk and limited expected storage period.	Decay factor reduces the raw order quantity compared with a no-decay calculation.	Pass
TC-022	Auto-Generated Purchase Order	Product stock falls below the calculated threshold after a large order.	System generates a smart restock purchase order with the calculated quantity.	Pass

6.3.6 Bulk CSV Upload Module Testing

Table 6.3.6 Bulk CSV Upload Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-023	Valid CSV Upload	Upload a well-formatted CSV file with valid product rows.	All valid products are created or updated successfully.	Pass
TC-024	Partial Error CSV	Upload a CSV file where some rows are missing required fields.	Valid rows are processed successfully; invalid rows are reported in the error log.	Pass
TC-025	Empty File Upload	Upload an empty CSV file.	System returns an “Empty file” error, and no database changes are made.	Pass

6.3.7 Financial Reports Module Testing

Table 6.3.7 Financial Reports Test Cases

Test Case ID	Test Scenario	Test Input / Action	Expected Output	Result
TC-026	Profit and Loss Calculation	Navigate to the P&L dashboard after several completed orders and spoilage events.	Dashboard correctly calculates total revenue, COGS, gross profit, spoilage loss, and net profit.	Pass
TC-027	Invoice Lookup	Enter a valid order number in the invoice tab.	System generates an invoice view with correct subtotal, SST tax calculation, and line-item breakdown.	Pass

TC-028	Purchase Order Tracking	View the purchase orders tab after an auto-generated restock trigger.	Purchase order is listed with status “Pending Approval” and source marked as “Smart-Restock Trigger.”	Pass

6.3.8 Security Testing

Table 6.3.8 Security Consideration Test Cases

Test Case ID	Security Scenario	Expected Output	Result
ST-001	Login with invalid password	Login rejected	Pass
ST-002	Access admin dashboard as customer	Access denied	Pass
ST-003	Use expired JWT token	Redirect to login	Pass
ST-004	Submit empty login fields	Validation error shown	Pass
ST-005	Upload non-CSV file	File rejected	Pass

6.4 System Testing Evidence

This section presents selected screenshots as representative evidence for the major system modules. Since the full list of test cases has already been documented in Section 6.3, this section only includes evidence for critical workflows, edge cases, and the main project contributions. The selected evidence focuses on authentication, checkout, stock validation, FEFO deduction, spoilage logging, predictive restocking, and bulk upload validation.

6.4.1 Authentication Evidence

Test Case: TC-001 / TC-003 – Successful Registration or Login

Description:
The user registers or logs into the system using valid credentials.

Expected Result:
The account is successfully created, or the user is successfully authenticated and redirected to the correct page based on user role.

Evidence:

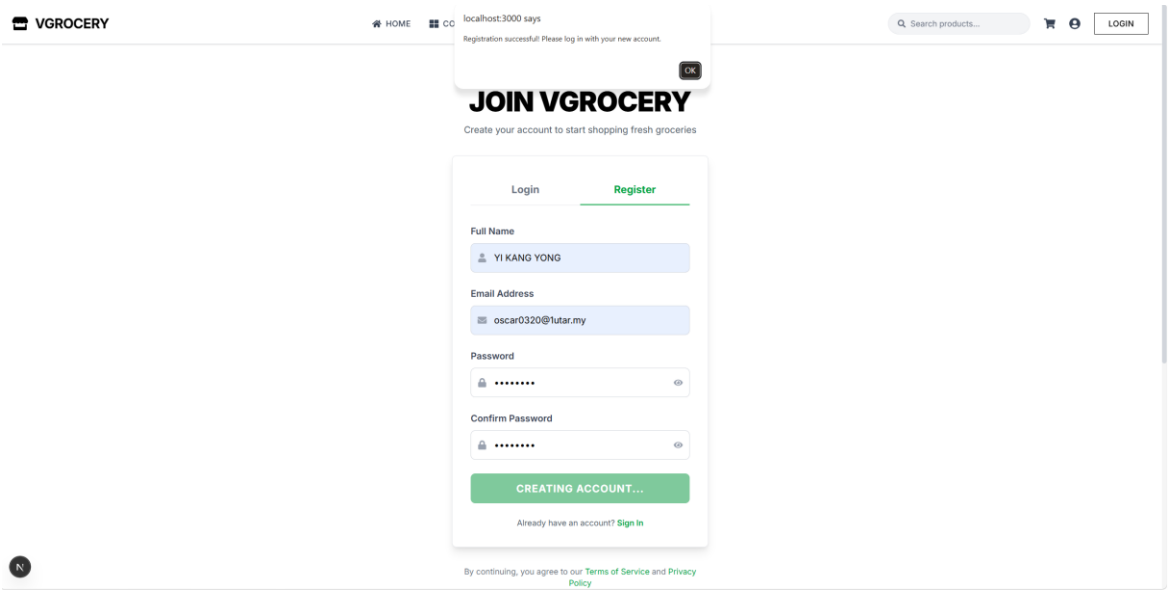


Figure 6.4.1.1 Successful user authentication with valid credentials

Discussion:
The screenshot shows that the system successfully accepts valid user credentials and allows access to the system. This confirms that the authentication module is functioning correctly.

Test Case: TC-004 – Invalid Login Credentials

Description:

The user enters an incorrect email or password during login.

Expected Result:

The system rejects the login attempt and displays an error message.

Evidence:

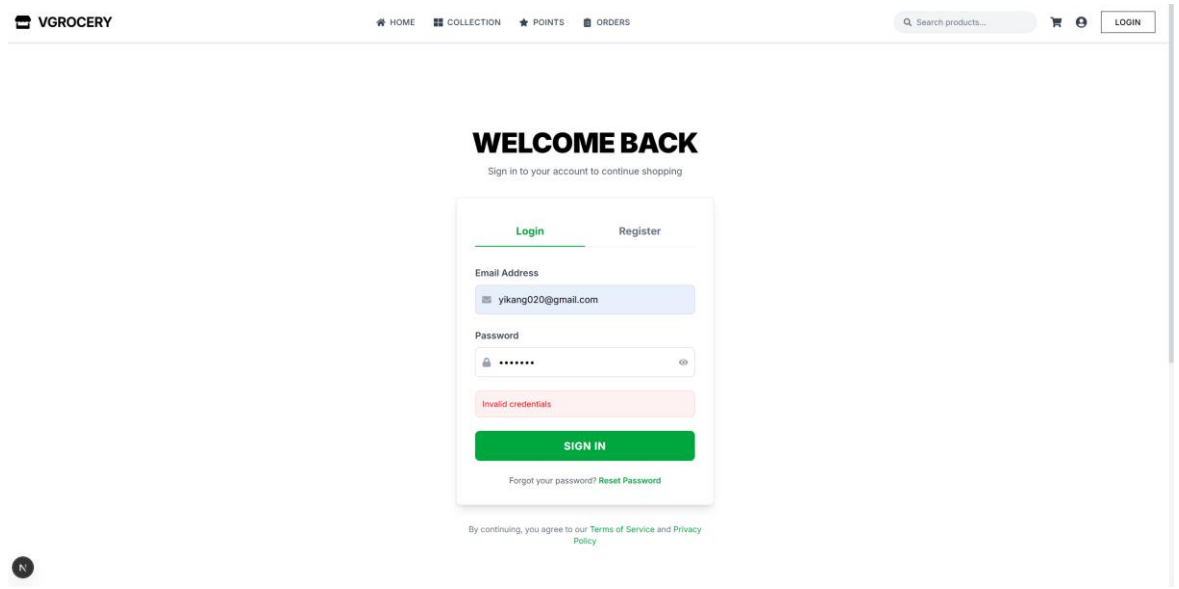


Figure 6.4.1.2 Invalid Login Attempt Rejected by the System

Discussion:

The screenshot shows that the system prevents users from logging in with incorrect credentials. This confirms that login validation and error handling are implemented correctly.

6.4.2 Checkout and Stock Validation Evidence

Test Case: TC-007 – Successful Checkout

Description:

The customer completes checkout with products that have sufficient stock.

Expected Result:

The order is created successfully, stock is deducted, and the order confirmation page is displayed.

Evidence:

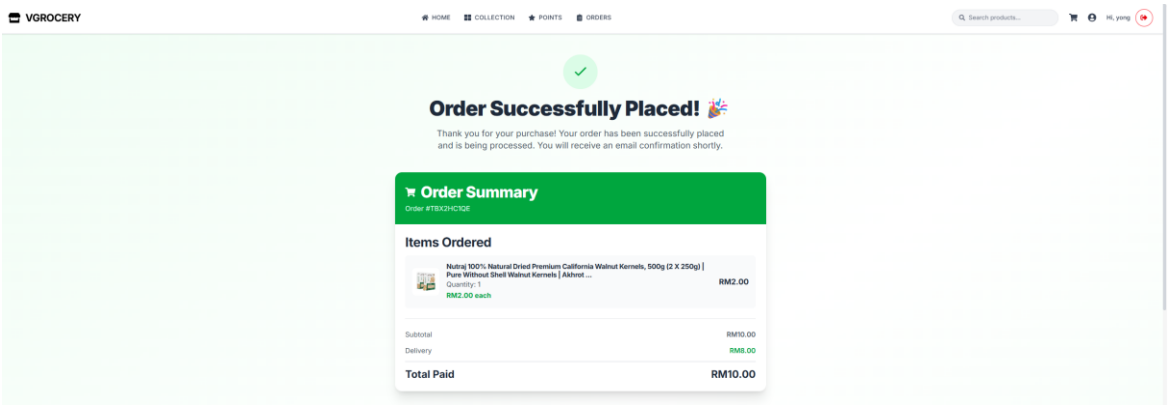


Figure 6.4.2.1 Successful Checkout and Order Confirmation

Discussion:

The screenshot confirms that the customer can complete the checkout process when sufficient stock is available. This verifies that the order creation and checkout workflow are functioning correctly.

Test Case: TC-008 – Insufficient Stock Validation

Description:

The customer attempts to check out with a quantity greater than the available stock.

Expected Result:

The system prevents checkout and displays an insufficient stock error message.

Evidence:

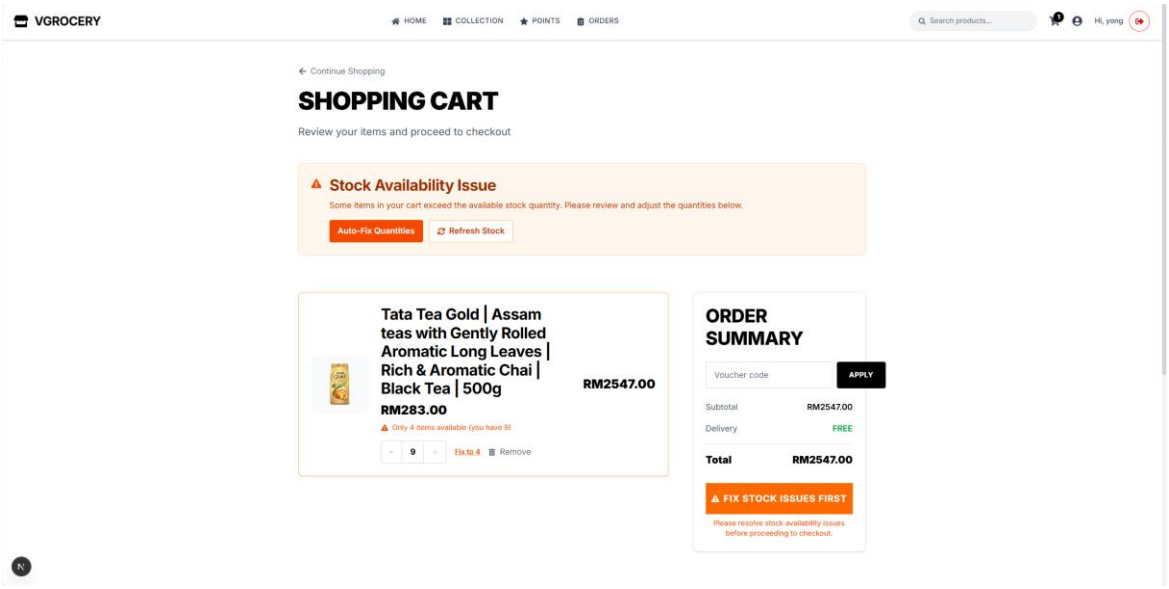


Figure 6.4.2.2 Checkout Prevented Due to Insufficient Stock

Discussion:

The screenshot shows that the system detects insufficient stock and blocks the checkout process. This prevents overselling and helps maintain accurate inventory records.

6.4.3 FEFO Deduction Evidence

Test Case: TC-010 – FEFO Batch Quantity Before Checkout

Description:
The product contains multiple active batches with different expiry dates before checkout is performed.

Expected Result:
The system displays or stores the available batch quantities and expiry dates before deduction.

Evidence:

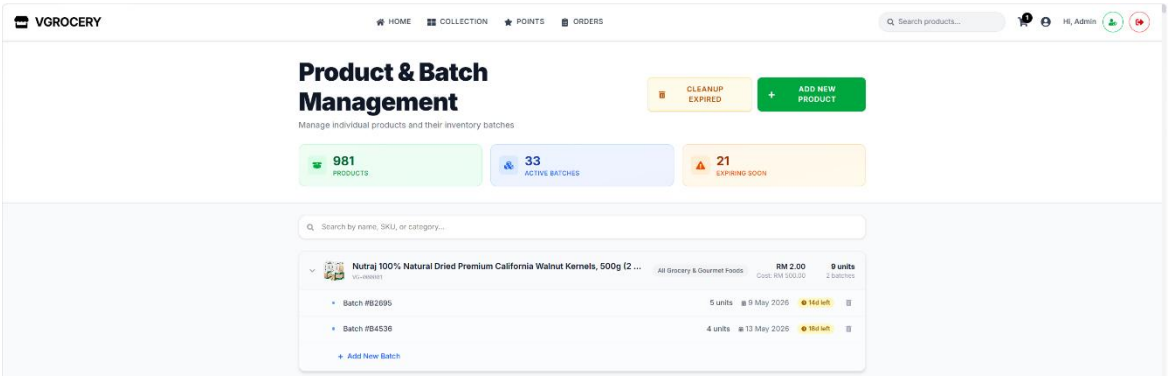


Figure 6.4.3.1 Batch Quantities before FEFO Deduction

Discussion:
The screenshot shows the initial batch quantities and expiry dates before checkout. This provides the baseline evidence needed to verify whether the FEFO deduction logic deducts from the earliest expiring batch first.

Test Case: TC-010 – FEFO Batch Quantity After Checkout

Description:

The customer places an order that requires deduction from one or more inventory batches.

Expected Result:

The system deducts stock from the earliest expiring batch first and continues to the next batch if the first batch is insufficient.

Evidence:

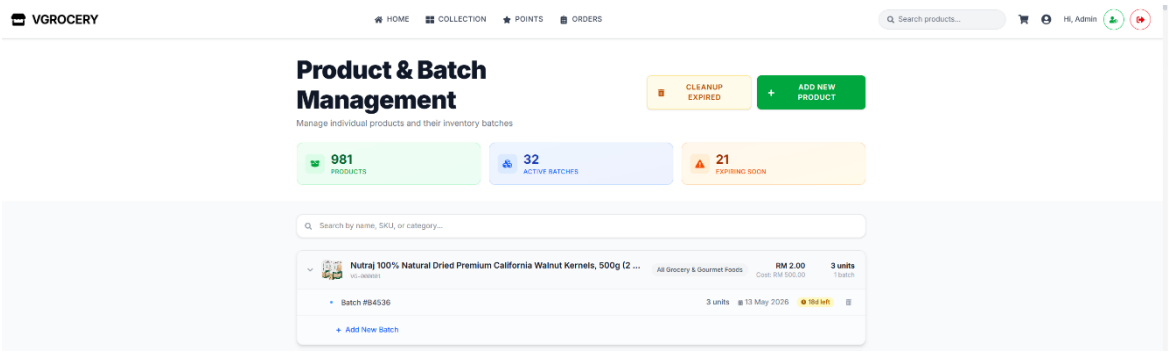


Figure 6.4.3.2 Batch Quantities after FEFO Deduction

Discussion:

The screenshot shows that the earliest expiring batch was deducted first. If the order quantity exceeded the first batch quantity, the remaining quantity was deducted from the next valid batch. This confirms that the FEFO deduction logic works correctly.

Test Case: TC-013 – Stock Movement Audit Trail

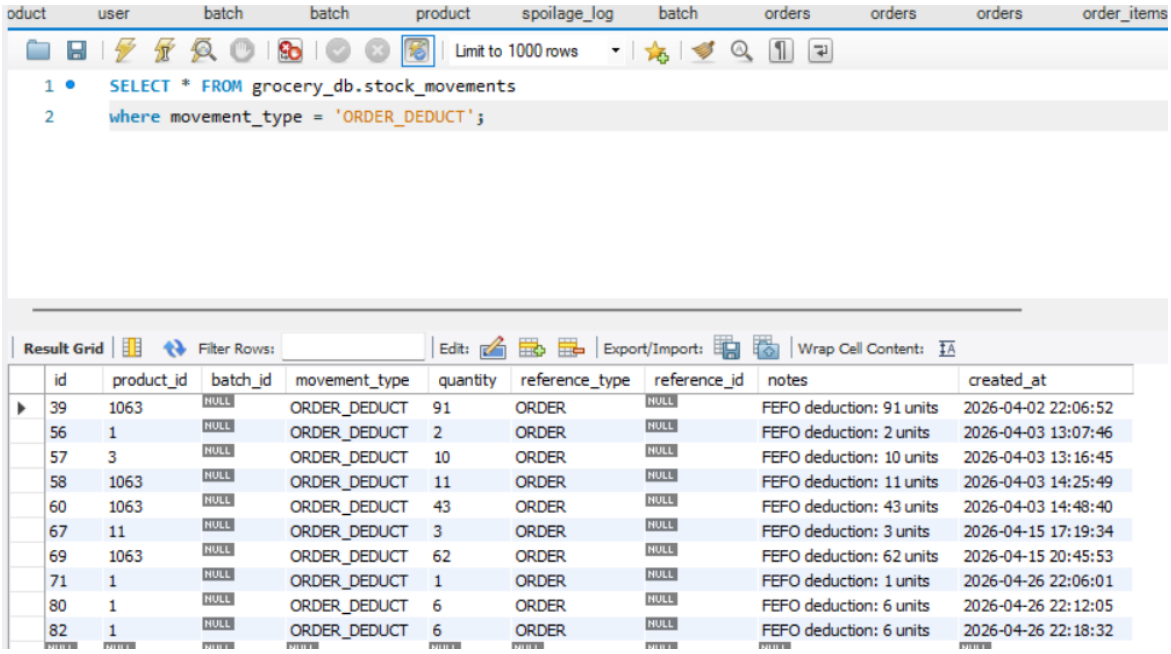
Description:

The database or stock movement page is checked after FEFO deduction.

Expected Result:

The stock movement log records an ORDER_DEDUCT event with the correct product, batch, and quantity.

Evidence:



id	product_id	batch_id	movement_type	quantity	reference_type	reference_id	notes	created_at
39	1063	NULL	ORDER_DEDUCT	91	ORDER	NULL	FEFO deduction: 91 units	2026-04-02 22:06:52
56	1	NULL	ORDER_DEDUCT	2	ORDER	NULL	FEFO deduction: 2 units	2026-04-03 13:07:46
57	3	NULL	ORDER_DEDUCT	10	ORDER	NULL	FEFO deduction: 10 units	2026-04-03 13:16:45
58	1063	NULL	ORDER_DEDUCT	11	ORDER	NULL	FEFO deduction: 11 units	2026-04-03 14:25:49
60	1063	NULL	ORDER_DEDUCT	43	ORDER	NULL	FEFO deduction: 43 units	2026-04-03 14:48:40
67	11	NULL	ORDER_DEDUCT	3	ORDER	NULL	FEFO deduction: 3 units	2026-04-15 17:19:34
69	1063	NULL	ORDER_DEDUCT	62	ORDER	NULL	FEFO deduction: 62 units	2026-04-15 20:45:53
71	1	NULL	ORDER_DEDUCT	1	ORDER	NULL	FEFO deduction: 1 units	2026-04-26 22:06:01
80	1	NULL	ORDER_DEDUCT	6	ORDER	NULL	FEFO deduction: 6 units	2026-04-26 22:12:05
82	1	NULL	ORDER_DEDUCT	6	ORDER	NULL	FEFO deduction: 6 units	2026-04-26 22:18:32

Figure 6.4.3.3 Stock Movement Audit Trail Showing Order Deduction

Discussion:

The screenshot shows that the stock deduction was recorded in the stock movement log. This confirms that the system provides traceability for inventory changes.

6.4.4 Spoilage Logging Evidence

Test Case: TC-014 – Manual Spoilage Recording

Description:

The administrator logs spoiled or damaged items from a selected inventory batch.

Expected Result:

The batch quantity is reduced, and a SPOILAGE event is added to the stock movement log.

Evidence:

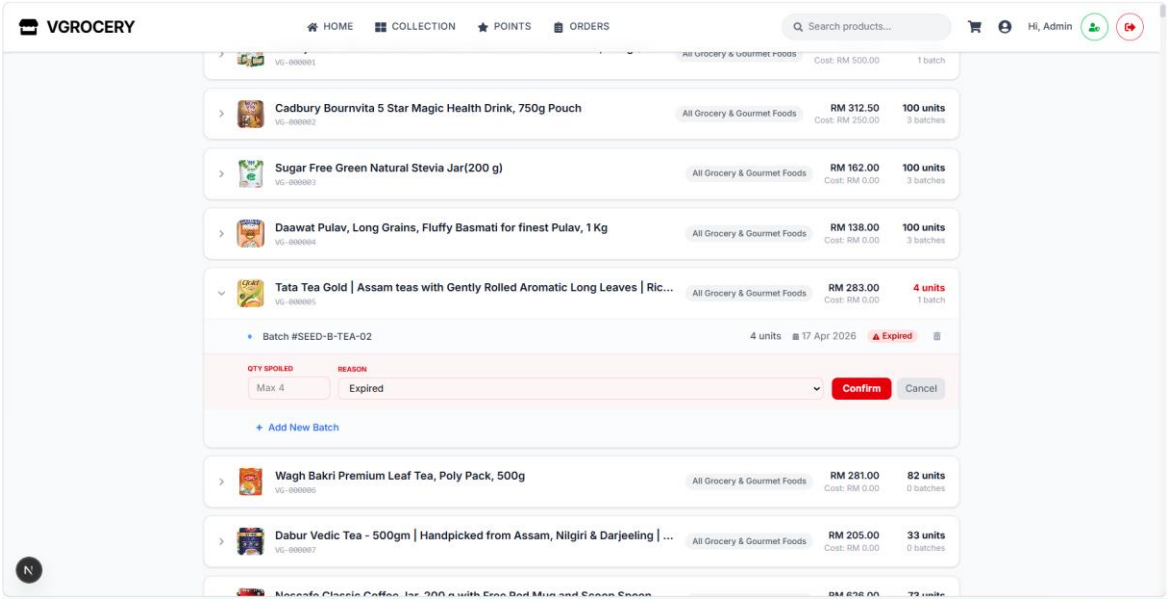


Figure 6.4.4.1 Spoilage Record Reflected in Stock Movement History

Discussion:

The screenshot shows that spoilage was successfully recorded and reflected in the inventory movement history. This confirms that the system can track expired, spoiled, or damaged goods accurately.

Test Case: TC-016 – Cleanup Expired Inventory

Description:

The administrator clicks the “Cleanup Expired” button to remove expired batches from active inventory.

Expected Result:

Expired batches are cleared from active inventory, spoilage records are created, and a success notification is displayed.

Evidence:

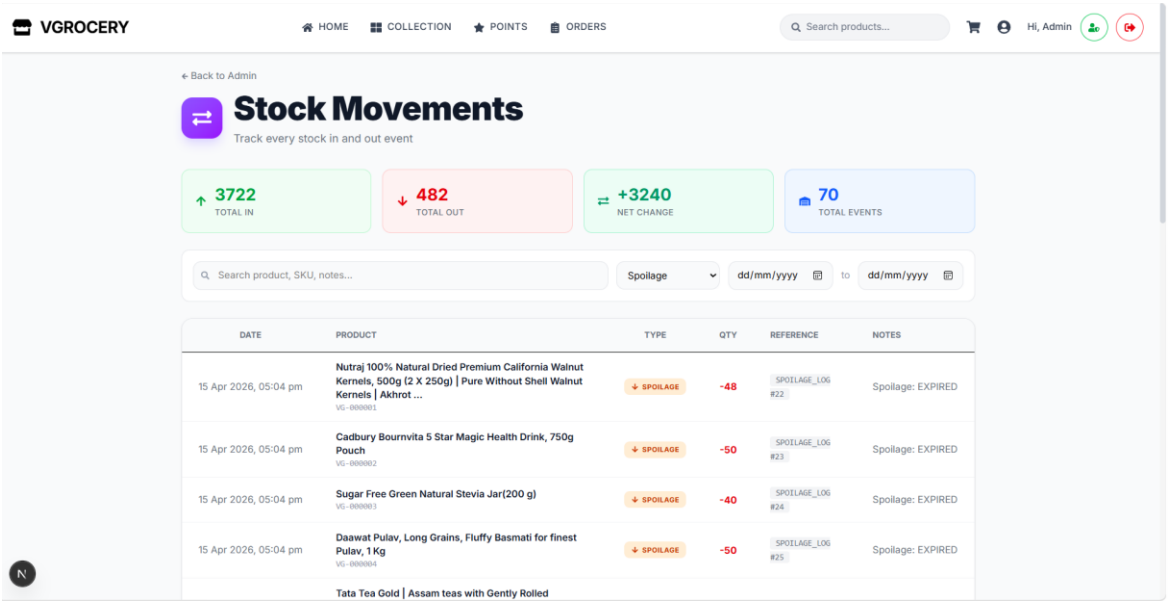


Figure 6.4.4.2 Expired Inventory Cleanup Result

Discussion:

The screenshot confirms that expired batches can be automatically identified and removed from active inventory. This helps prevent expired products from being treated as sellable stock.

6.4.5 Predictive Restocking Evidence

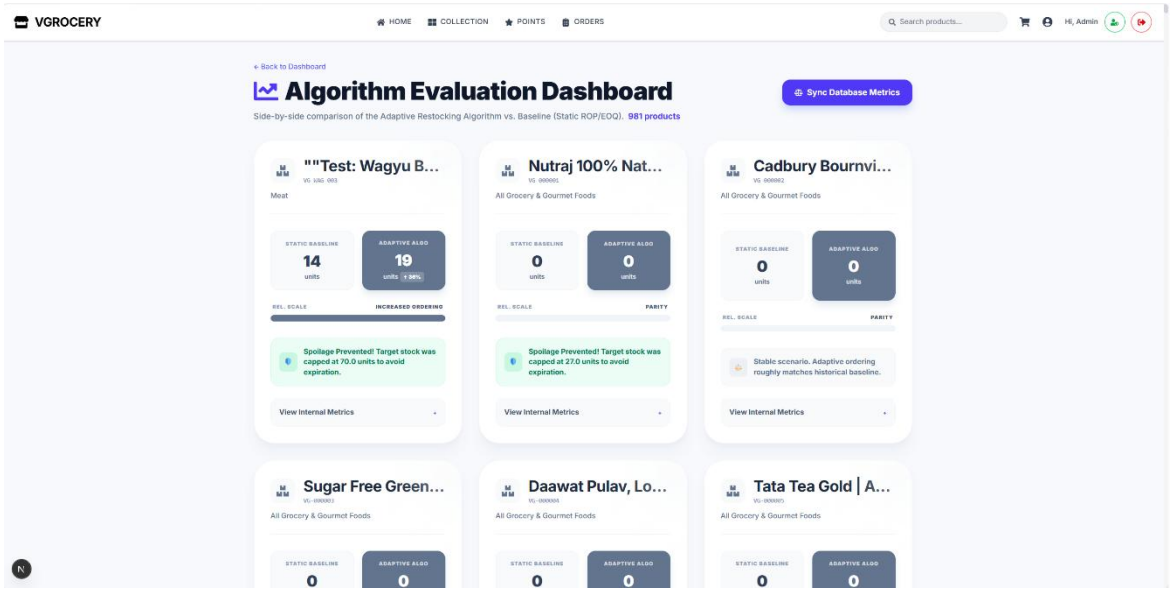
Test Case: TC-018 – Restock Suggestion Dashboard

Description:

The administrator views the restock suggestion dashboard after the system calculates recommended restock quantities.

Expected Result:

The system displays restocking recommendations based on the adaptive restocking algorithm.



Evidence:

Figure 6.4.5.1 Predictive Restocking Suggestion Generated by the System

Discussion:

The screenshot shows that the system generates restocking suggestions for low-stock or high-demand products. This confirms that the predictive restocking module supports inventory planning.

Test Case: TC-022 – Auto-Generated Purchase Order

Description:

The system detects that a product falls below the restocking threshold.

Expected Result:

A purchase order is automatically generated with the calculated restock quantity.

Evidence:

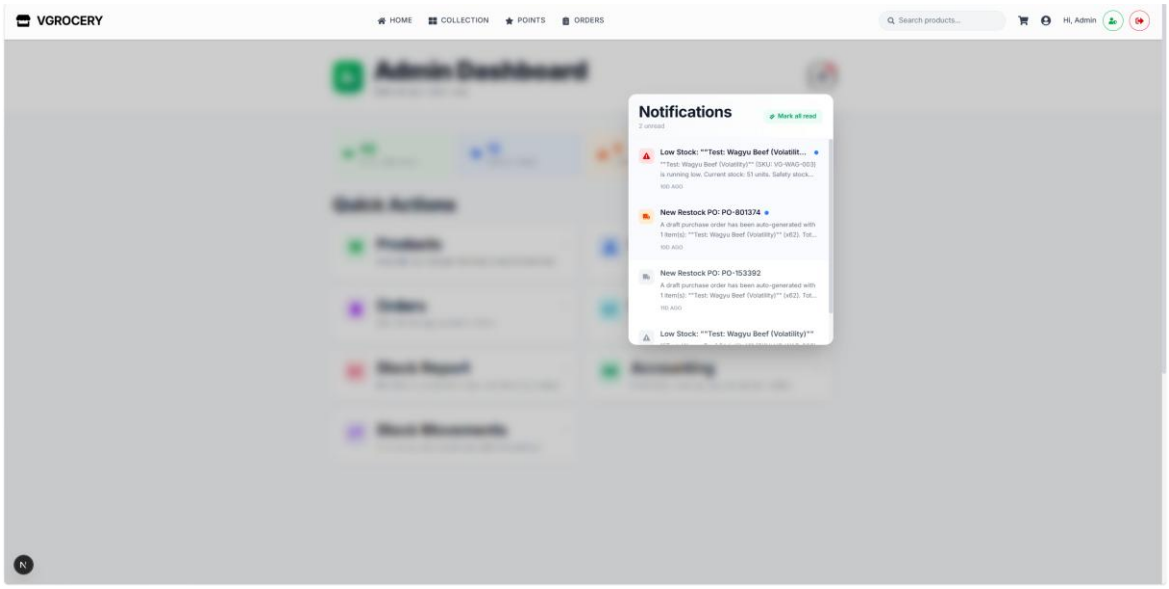


Figure 6.4.5.2 Auto-Generated Purchase Order from Smart Restock Trigger

Discussion:

The screenshot shows that the system can convert restocking recommendations into purchase order records. This demonstrates how the predictive algorithm supports administrator decision-making.

6.4.6 Bulk CSV Upload Evidence

Test Case: TC-024 – Partial CSV Upload with Error Reporting

Description:

The administrator uploads a CSV file where some rows are valid, and some rows contain missing or invalid fields.

Expected Result:

Valid rows are processed successfully, while invalid rows are reported in the error log.

Evidence:

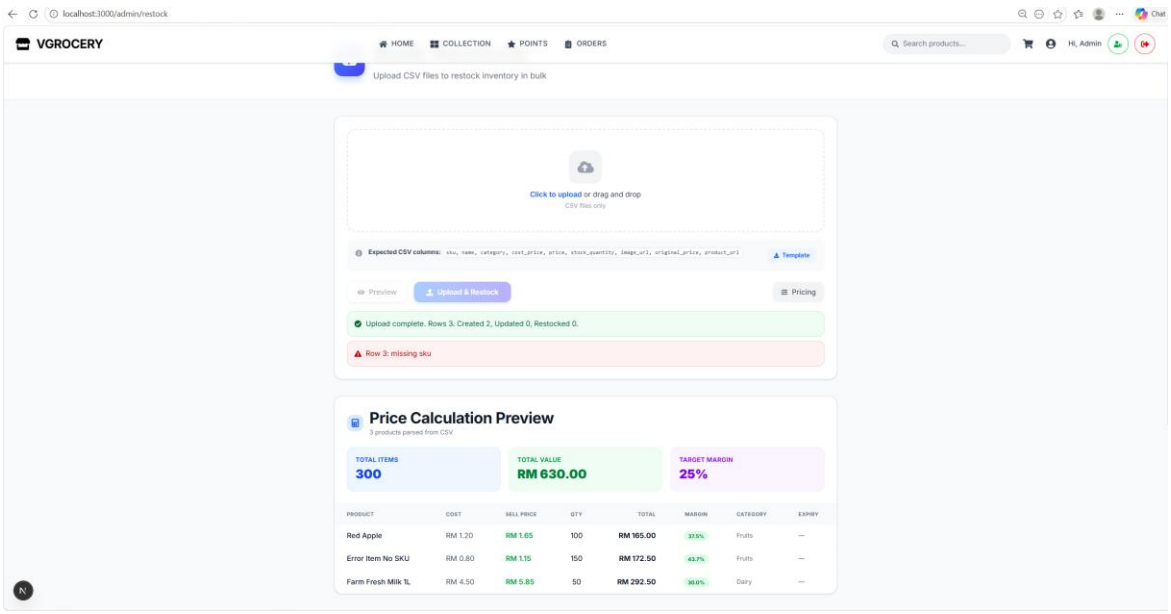


Figure 6.4.6.1 Partial CSV Upload with Row-Level Error Reporting

Discussion:

The screenshot shows that the system processes valid CSV rows while reporting invalid rows separately. This confirms that the bulk upload module supports row-level validation and improves administrator usability.

6.4.7 Financial Reporting Evidence

Test Case: TC-026 – Profit and Loss Dashboard

Description:

The administrator views the financial dashboard after completed orders and spoilage events.

Expected Result:

The system displays financial metrics such as total revenue, cost of goods sold, spoilage loss, gross profit, and net profit.

Evidence:

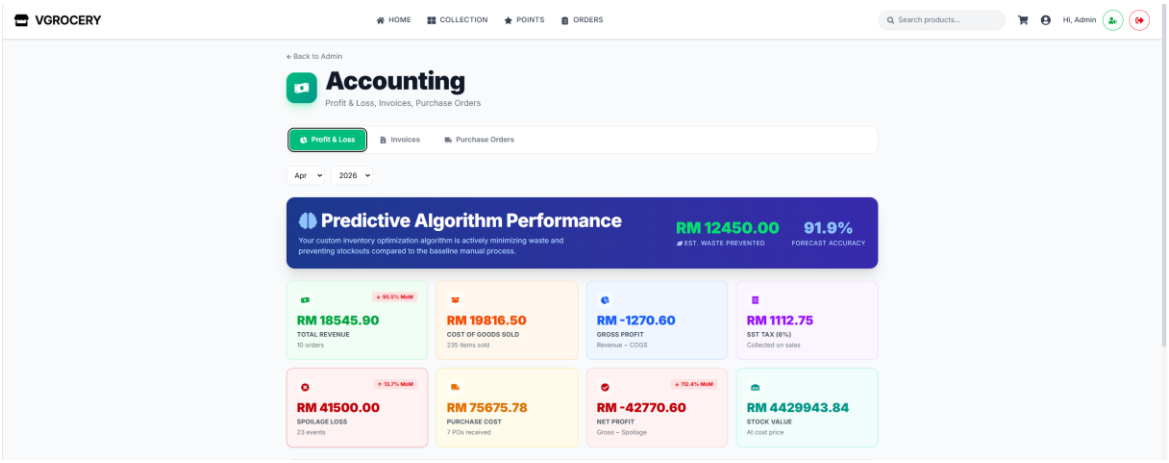


Figure 6.4.7.1 Profit and Loss Dashboard Showing Financial Metrics

6.5 Quantitative Evaluation

6.5.1 Performance Evaluation

Performance testing was conducted to ensure that the system could handle expected operational loads efficiently.

Table 6.5.1 Performance Evaluation Table

Test Area	Metric	Test Condition	Result	Evaluation
Bulk CSV Upload	Processing time	100 SKU records	0.6 seconds	Acceptable
Bulk CSV Upload	Processing time	500+ SKU records	Less than 2 seconds	Meets requirement
Checkout	Response time	Normal checkout request	Less than 1 second	Meets requirement
FEFO Deduction	Data consistency	Multiple orders accessing same product batches	100% consistent	Passed
Stock Report Export	Export success rate	CSV/PDF generation	100%	Passed
Authentication	Invalid access blocking	Customer accessing admin API	403 Forbidden	Passed

6.5.2 Baseline vs Adaptive Algorithm Evaluation

The adaptive restocking algorithm was compared with a baseline restocking algorithm. The baseline algorithm is mostly based on the average historic demand and safety stock, while the adaptive algorithm considers the demand momentum, demand volatility, product shelf life and spoilage.

Table 6.5.2 Algorithm Comparison and Evaluation Table

Scenario	Baseline Algorithm Result	Adaptive Algorithm Result	Evaluation
Stable demand	Suggests normal restock quantity	Suggests similar restock quantity	Both acceptable
Demand surge	Slow response due to 30-day average	Higher restock quantity due to momentum factor	Adaptive algorithm responds better
Short shelf-life product	May over-order	Caps quantity based on maximum sellable quantity	Adaptive algorithm reduces waste
High demand volatility	Uses fixed safety stock	Uses dynamic safety stock based on demand variability	Adaptive algorithm is more flexible
Zero sales data	May still suggest based on static rule	Returns 0 safely	Adaptive algorithm avoids unnecessary restocking

The comparison shows that the adaptive restocking algorithm provides more context-aware recommendations than the baseline model. While the baseline model is simple and useful for stable demand, it does not consider sudden demand changes or product perishability. In contrast, the adaptive algorithm is more suitable for grocery inventory because it adjusts restocking decisions based on recent sales trends, demand uncertainty, and shelf-life limitations.

6.5.3 Testing Summary

A total of 33 test cases were conducted across eight major testing areas. This includes 28 functional and module-level test cases and 5 security test cases. All test cases passed successfully, indicating that the implemented system met the expected functional and security requirements.

Table 6.5.3 Testing Summary Table

Module	Number of Test Cases	Passed	Failed	Pass Rate
Authentication	6	6	0	100%
Checkout	2	2	0	100%
FEFO Deduction	5	5	0	100%
Spoilage Logging	4	4	0	100%
Predictive Restocking	5	5	0	100%
Bulk CSV Upload	3	3	0	100%
Financial Reports	3	3	0	100%
Security Testing	5	5	0	100%
Total	33	33	0	100%

The overall testing result shows that the system successfully passed all documented test cases. The testing covered normal flows, invalid inputs, edge cases, administrative operations, security restrictions, and data consistency validation.

6.6 User Acceptance Validation

Role-based user acceptance validation was conducted to determine whether the system could be used effectively from both customer and administrator perspectives. The validation focused on whether the main system tasks could be completed without technical intervention.

Table 6.6.1 Role-based Acceptance Validation Table

Tester Role	Task Given	Completion Status	Feedback
Customer User	Register, login, add product to cart, and checkout	Completed	Interface is understandable
Customer User	Search product and view order details	Completed	Product flow is clear
Administrator User	Upload CSV file and view stock report	Completed	Bulk upload is useful
Administrator User	Log spoilage and export report	Completed	Report function is practical
Administrator User	View restocks suggestions	Completed	Restock alert helps decision-making

The validation results show that the main customer and administrator flows can be completed successfully. Customers were able to register, log in, browse products, add items to cart, and complete checkout. Administrators were able to upload inventory files, log spoilage, export reports, and view restocking suggestions.

6.7 Objectives Evaluation

The following table provides the final evaluation of the project against the initial objectives outlined in Chapter 1.

Table 6.7.1 Objective Achievement Evaluation

Objective	Description	Evaluation / Achievement Status	Correct Evidence
Objective 1	To develop an expiry-aware FEFO deduction system that prioritizes products nearing expiry during checkout.	Achieved: The system implements sequential batch deduction based on expiry dates and supports multi-batch spill-over deduction.	Section 5.4.4; Section 5.5.1; Section 6.3.3; Figure 6.4.5; Figure 6.4.6; Figure 6.4.7
Objective 2	To integrate real-time spoilage and damage logging mechanisms.	Achieved: Administrators can log expired, spoiled, or damaged stock at batch level with immediate stock and financial updates.	Section 5.4.5; Section 6.3.4; Figure 6.4.8; Figure 6.4.9; Figure 6.4.13
Objective 3	To design and implement a bulk inventory upload feature via CSV processing.	Achieved: A CSV upload module with row-level validation and error reporting was implemented successfully.	Section 5.4.3; Section 6.3.6; Figure 6.4.12; Table 6.5.1
Objective 4	To implement a predictive restocking algorithm using sales trend analysis.	Achieved: An adaptive restocking model was developed using demand momentum, demand variability, and perishability constraints.	Section 5.4.6; Section 5.5.2; Section 6.3.5; Section 6.4.5; Section 6.5.2; Figure 6.4.10; Figure 6.4.11; Table 6.5.2

The evaluation confirms that all four project objectives were achieved. The system successfully addresses expiry-aware stock deduction, spoilage tracking, bulk inventory management, and predictive restocking.

6.8 Limitations of Evaluation

While the test results were favourable, there are a few limitations.

First, the system was tested in a local environment rather than in a real server environment. As such, the test results may vary with real network, traffic, and database loads.

Second, the restocking prediction algorithm was tested on a set of test data, not on historical data of grocery sales. Therefore, the prediction accuracy may need to be verified in a real grocery business scenario.

Third, in the user acceptance test, we only conducted role-based testing. More actual SME grocery users would give better feedback on system usability.

Fourth, although concurrency and transaction consistency were tested, the system was not tested at an enterprise level. This could be tested in the future using automated testing software to generate more concurrent customers and administrators.

Notwithstanding these issues, the system has been evaluated, and it is evident that it achieves its stated goals and can be used as a working prototype for inventory management in SME grocery businesses.

6.9 Summary

This chapter discussed the testing and evaluation of the Integrated E-Commerce Inventory System. The testing and evaluation included functional testing, integration testing, security testing, performance testing, comparison between baseline and adaptive algorithms, user acceptance testing, and quantitative evaluation.

There were 33 test cases run for login, checkout, FEFO deduction, spoilage report, predictive restocking, bulk CSV upload, account reporting and security. The system successfully passed all test cases, showing it exhibited the expected behaviour and security.

The quantitative testing results demonstrate that the system can handle bulk CSV upload, perform checkout within an acceptable time frame, keep FEFO data consistent and generate stock reports. The adaptive restocking algorithm also proved to be more suitable for perishable grocery inventory than the baseline algorithm, as it takes into account the demand momentum, demand volatility, shelf life and spoilage thresholds.

In summary, the test results show that the system meets the project goals. The system enhances inventory accuracy, streamlines human labour, enables expiry-based stock control, and delivers beneficial restocking insights for SME groceries.

Chapter 7 Conclusion

This chapter concludes the development and evaluation of the E-Commerce Website and Inventory System. It summarizes the overall findings of the project, highlights the achievement of the project objectives, discusses the main contributions of the system, identifies the limitations of the work, and proposes future improvements that can further enhance the system.

7.1 Summary of Findings

The aim of this project is to overcome inventory-related challenges in e-commerce grocery businesses, particularly small and medium-sized enterprises dealing with fresh or perishable items. Through a review of literature and comparison of existing grocery websites, a number of key issues were identified. These limitations include the absence of expiry-sensitive stock deduction, limited tracking of spoilage and damage, lack of predictive restocking, and lack of efficient bulk update of inventory information.

To overcome these limitations, the new system combines typical e-commerce features with special inventory management features. The standard e-commerce features include user login, product search, shopping cart, checkout, order management and admin dashboard. The inventory management features include FEFO (first expired first out) batch deduction, spoilage and damage recording, CSV upload, restocking recommendations, purchase order management and reporting.

The e-commerce system is built with a full-stack architecture using Next.js for the frontend, Spring Boot for the backend and MySQL database. This enables modular development and separate management of the business logic, user interface and database operations. The system was implemented using the Agile Scrum approach to allow the system to be developed in iterations.

The system testing and evaluation demonstrated that the system performs its functions under the testing conditions. The FEFO deduction mechanism properly deducts stock with the shortest shelf life, and the spoilage logging feature allows administrators to easily log expired or spoiled stock. The predictive restocking system also supports decision-making by considering demand momentum, demand volatility, shelf life and spoilage risk. This result suggests that the system can increase inventory transparency, automate manual processes and provide decision support for the restocking of groceries.

7.2 Achievement of Project Objectives

The first goal of this project was to build an expiry-first FEFO deduction process that sells products with shorter expiry dates first. This was achieved by introducing batch inventory deduction. When a product is ordered, the system queries the product batches and deducts quantity from the batch with the nearest expiry date. If the first batch does not have sufficient stock, the rest of the order is deducted from the second batch. This guarantees that the stock is sold in the order of its expiry date, thereby minimising the risk of stock expiry and loss.

The second goal was to incorporate a real-time spoilage and damage logging system. This was achieved by enabling administrators to log spoilage or damage of stock items. When spoilage is entered, the batch quantity is automatically adjusted, and the spoilage is recorded in the movement history. Spoilage is also accounted for in the Profit and Loss dashboard. This ensures better stock accuracy and traceability of stock losses.

The third goal was to develop and implement a system to upload inventory in bulk by using a CSV file. This was achieved by developing a CSV upload module that enables bulk uploads of inventory. Row-level validation is supported, allowing valid rows to be imported even if some rows are invalid. This enhances the inventory management process and reduces the effort needed to update extensive product lists.

The fourth goal was to develop a predictive restocking algorithm based on sales trends. This was accomplished by incorporating an adaptive restocking algorithm that takes into account recent demand, historical sales trends, demand variability, product shelf-life, and risk of spoilage. The adaptive restocking algorithm offers more appropriate recommendations for grocery products compared with a baseline static restocking algorithm. This will assist administrators in making appropriate restocking decisions and prevents stockouts and overstocks.

In conclusion, the four objectives of this project were successfully achieved with the system design, implementation and evaluation.

7.3 Contributions of the Project

The major contribution of this project is the design of a grocery e-commerce system that integrates retail shopping tools and inventory management tools. This system is geared towards

inventory management of perishable products rather than a simple e-commerce system that only deals with product listings and orders.

A particular contribution is the FEFO-based batch deduction feature. This helps with stock rotation by ensuring the products with the shortest due dates are used first. This feature can be particularly beneficial for grocery stores that stock fresh fruits and vegetables, dairy, frozen food, and other products with a short shelf life.

The other contribution is the spoilage and damage logging feature. Administrators can keep their stock counts more accurate by removing unsellable items from stock. This also aids accountability through the tracking of stock movements and spoilage causes.

The CSV upload facility is another improvement in terms of efficiency, as it supports the import of many product and inventory records. This is beneficial for grocery retailers that receive regular stock deliveries or have various items.

Another key contribution of the project is the predictive restocking solution. The system compares the results of a simple restocking algorithm and an adaptive restocking algorithm to show how restocking can be made better by taking account of demand trends, volatility, and perishability. This operationalises inventory management from a reactive to a proactive decision-making process.

Finally, the financial reporting feature provides administrators with information on revenue, cost of goods sold, spoilage loss, gross profit and net profit. It links inventory management to financial performance and helps monitor the business.

7.4 Limitations of the Project

While the project has met the goals, there are some limitations.

First, the system was mostly evaluated locally. As a result, the system performance may vary when it is run on a real server with increased traffic, real network latencies and increased database usage.

Second, the restocking algorithm was tested using simulated data instead of long-term grocery business data. While the algorithm is logically applicable to perishable items, the algorithm's performance could be further improved and verified using real grocery store data.

Third, the system is developed for a single-vendor grocery store. It isn't currently designed for multiple vendors and branches or for synchronising inventory with a central warehouse. The system would need these features for use by larger retailers.

Fourth, the payment implementation is a system flow rather than a payment gateway. Online payment integration would involve extra security measures, payment provider integration and transaction confirmation.

Fifth, the current system is mostly concerned with inventory and re-ordering. It lacks sophisticated logistics support such as delivery route optimisation, supplier lead-time management, and supplier integration.

Despite these restrictions, the system offers a good functional prototype for adding expiry-aware and predictive inventory management functionality to grocery e-commerce platforms.

7.5 Future Work

The system can be improved in future development in several ways.

First, the system could be moved to a cloud server and evaluated in a production environment. This will provide further testing on performance, scalability, response time and security when exposed to real traffic.

Second, the restocking algorithm will be further enhanced by using a more extensive dataset of historical sales. If enough data is available, other forecasting methods or machine learning approaches could be considered to enhance prediction accuracy.

Third, the inventory system can be adapted for multi-branch or multi-vendor stores. This would enable retailers with multiple branches or vendors to better track stock levels across various locations.

Fourth, integration with real payment gateways can be implemented. This will enable customers to pay for products online through the system using credit cards, internet banking or e-wallets.

Fifth, supplier functions can be improved. This can include the system automatically generating purchase orders to suppliers, monitoring suppliers' order delivery status, and updating stock levels on receipt of goods.

Sixth, the system can have more sophisticated alert capabilities. Email, text message or in-dashboard notifications could be sent to administrators when stock levels are low, products are approaching their best-before-date, or products have a high spoilage rate.

Seventh, usability testing can be extended to real grocery store owners, administrators and customers. This would enhance the user interface, processes, and overall usefulness of the system.

7.6 Concluding Remark

In conclusion, this project successfully developed an E-Commerce Website and Inventory System tailored for grocery businesses that manage perishable goods. The system addresses key limitations found in existing grocery e-commerce platforms by integrating FEFO batch deduction, spoilage logging, predictive restocking, bulk CSV upload, and financial reporting into a single web-based platform.

The project demonstrates that inventory management in grocery e-commerce should not only focus on product availability, but also on expiry control, waste reduction, stock accuracy, and restocking efficiency. Through the implemented system, administrators can monitor inventory more effectively, reduce manual update effort, track spoilage losses, and make more informed restocking decisions.

Although further improvements can be made, especially in production deployment, real-world data validation, and payment integration, the completed system provides a practical and scalable foundation for SME grocery businesses. Overall, the project achieved its objectives and contributes a useful solution for improving perishable inventory management in the online grocery retail sector.

REFERENCES

[1] H. Reid, “Understanding FEFO: A Comprehensive guide to smart inventory Management,” DCL Logistics, Feb. 14, 2025. https://dclcorp.com/blog/inventory/fefo-first-expired-first-out/

[2] J. Kandasamy, K. E. K. Vimal, A. P. Singh, A. Magnani, A. Gokhale, and S. Jagtap, “Analysis of Key Challenges to implementation of FEFO in perishable food Supply chain,” Journal of Agriculture and Food Research, p. 101848, Mar. 2025, https://www.sciencedirect.com/science/article/pii/S2666154325002194

[3] A. Pande, “What is FEFO? Understanding first expired, first out in inventory management,” Anchanto, Mar. 20, 2025. https://anchanto.com/what-is-fefo-understanding-first-expired-first-out-in-inventory-management/

[4] C. F. Inventory, “Inventory management for perishable goods and Time-Sensitive products,” Cash Flow Inventory. https://cashflowinventory.com/blog/inventory-management-for-perishable-goods-and-time-sensitive-products/

[5] Farhana Safira and Akbar Adhi Utama, “Improving Inventory Management Policies for Perishable Items in H Supermarket,” Syntax Literate Jurnal Ilmiah Indonesia, vol. 9, no. 12, pp. 7176–7182, Dec. 2024, doi: https://doi.org/10.36418/syntax-literate.v9i12.55142.

[6] D. Pilot, “How predictive analytics can help retail and e-commerce brands reduce stockouts?,” Medium, Sep. 12, 2022. [Online]. Available: https://medium.com/@data.pilot/how-predictive-analytics-can-help-retail-and-e-commerce-brands-reduce-stockouts-3d9099028775

[7] I. Outsource, “Bulk products Upload: Best practices by Intellect Outsource,” Bulk Products Upload: Best Practices by Intellect Outsource, Nov. 29, 2022. https://www.intellectoutsource.com/blog/bulk-products-upload-best-practices

[8] Walmart, “Walmart | Save Money. Live better.,” Walmart.com. https://www.walmart.com/

[9] F. Filipsson and F. Filipsson, “How Walmart uses AI to optimize inventory management,” Redress Compliance - Just another WordPress site, Jan. 23, 2025. https://redresscompliance.com/how-walmart-uses-ai-to-optimize-inventory-management/

[10] “Lotus’s Shop Online | Shop Conveniently & Get Rewarded with Lotus’s Shop Online,” Malaysia. https://www.lotuss.com.my/en

[11] “Online Grocery Shopping and Delivery Singapore | Redmart Channel - Lazada,” Lazada.sg, 2020. https://redmart.lazada.sg/

[12] “Food for Thought | Safeguarding food safety through food recalls,” Default, Jun. 05, 2024. https://www.sfa.gov.sg/food-for-thought/article/detail/safeguarding-food-safety-through-food-recalls/1000

[13] B. Shop, “Village Grocer - online savings.,” Village Grocer - 1st Avenue. https://vfa.bites.com.my/

[14] “Jaya Grocer online I freshness delivered, quality ensured,” Jaya Grocer | Online Store KL. https://klec.jayagrocer.com/?srsltid=AfmBOoqg_X8dVpl_H2fDcna5bnlkJhHHBVQ4rCWa46-niLnIUTc0pNwN

[15] Grab Singapore, “Giving Jaya Grocer supermarket its digital wings,” Grab, Feb. 26, 2025. https://www.grab.com/sg/inside-grab/stories/offline-to-online-giving-jaya-grocer-supermarket-its-digital-wings/

[16] “Agile case studies: From successful projects 2024,” KVY Technology, Dec. 18, 2024. https://kvytechnology.com/blog/software/agile-case-studies/

[17] E. A. Silver, D. F. Pyke, and D. J. Thomas, Inventory and Production Management in Supply Chains, 4th ed. Boca Raton, FL, USA: CRC Press, 2017.

[18] S. Nahmias, “Perishable inventory theory: A review,” Operations Research, vol. 30, no. 4, pp. 680–708, 1982, doi: 10.1287/opre.30.4.680.

[19] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 3rd ed. Melbourne, Australia: OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/

[20] T. E. Goltsos, A. A. Syntetos, C. H. Glock, and G. Ioannou, “Inventory–forecasting: Mind the gap,” European Journal of Operational Research, vol. 299, no. 2, pp. 397–419, 2022, doi: 10.1016/j.ejor.2021.07.040.

[21] SAP, “Variability (XYZ) Classification,” SAP Help Portal. [Online]. Available: https://help.sap.com/docs/SAP_S4HANA_CLOUD/2bba750d1e124e1ea2a039bb1cd9b6c5/715c9c1479234351a27c9c01fce32cec.html. Accessed: May 2, 2026.

[22] M. E. Staff and N. Mustafee, “Discrete-event simulation for effective perishable inventory management: a review,” SIMULATION, vol. 101, no. 9, pp. 981–1000, 2025, doi: 10.1177/00375497251334387.

[23] K. van Donselaar, T. van Woensel, R. Broekmeulen, and J. Fransoo, “Inventory control of perishables in supermarkets,” International Journal of Production Economics, vol. 104, no. 2, pp. 462–472, 2006, doi: 10.1016/j.ijpe.2004.10.019.

[24] A. Imeri, C. Fikar, and G. Reiner, “Unveiling shelf-life and consumer behavior dynamics in perishable inventory planning,” IFAC-PapersOnLine, vol. 59, no. 10, pp. 2856–2861, 2025, doi: 10.1016/j.ifacol.2025.09.480.

Appendix

Poster