Business Process Automation for Inventory Management

: Purchase Reordering, Supplier Management & Notification Module

By

Lee Siaw Jing

FACULTY OF COMPUTING AND
INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY OF
MANAGEMENT AND TECHNOLOGY
KUALA LUMPUR

ACADEMIC YEAR
2025/26

Business Process Automation for Inventory Management

By

Lee Siaw Jing

Supervisor: Dr Ng Yen Phing

A project report submitted to the
Faculty of Computing and Information Technology
in partial fulfillment of the requirement for the
Bachelor of Information Systems (Honours).

Faculty of Computing and Information Technology
Tunku Abdul Rahman University of Management and Technology
Kuala Lumpur

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

Jing

_____

Lee Siaw Jing

Bachelor of Information Systems (Honours) in Enterprise Information Systems

ID: 24WMR0891

Abstract

This project addresses the common inventory management challenges faced by small and medium-sized enterprises (SMEs), such as stockouts, overstocking, and inefficient procurement processes. Many SMEs still rely on outdated or manual systems, leading to operational delays and errors. To resolve these issues, a cloud-based inventory management system was developed to automate stock monitoring, reordering, supplier coordination, and notifications.

The system includes modules for threshold-based stock alerts, automated purchase order generation, supplier profile management, real-time reporting, and role-based access control. Built on Amazon Web Services (AWS), it utilizes services like Lambda, RDS, SNS, S3, and CloudWatch to support automation, scalability, and secure data handling. The solution is designed to offer SMEs an affordable and streamlined alternative to complex ERP systems.

The development approach follows modular design and cloud integration using PHP, JavaScript, and AWS technologies. Serverless architecture allows event-driven backend automation, while centralized databases and secure storage ensure data reliability and accessibility.

Functional testing and user acceptance testing were conducted using simulated inventory scenarios. Key evaluation criteria included accuracy of stock alerts, purchase order logic, and notification delivery timeliness.

The system effectively reduces manual workload, improves procurement accuracy, and enhances operational visibility. While advanced features like demand forecasting are out of scope, the project sets a solid foundation for future enhancement and contributes to the digital transformation of SMEs.

The system successfully achieved its objectives by automating critical inventory management tasks, providing real-time insights, and ensuring seamless integration with supplier and procurement processes. In conclusion, this project demonstrates the potential for SMEs to improve their operational efficiency and decision-making by leveraging cloud-based technologies. Further enhancements, including demand forecasting and more advanced analytics, are recommended for future versions.

Acknowledgement

I would like to express my deepest appreciation to my project supervisor, Dr. Ng Yen Phing, for her invaluable support, insightful advice, and continuous encouragement throughout the course of this project. Her guidance helped shape my ideas and strengthened the direction of this system's development.

I am also sincerely thankful to Ms. Kong Hooi Ming, my project moderator, for her constructive feedback, evaluations, and the time she dedicated to reviewing the work in progress. Her observations and questions helped refine both the technical and documentation aspects of the project.

Special thanks to my teammate, Ms. Lim Yean Yee, for her cooperation and efforts in developing the Stock Monitoring, Inventory Dashboard, and Cloud Reporting modules. Our teamwork and mutual support played a vital role in achieving a fully integrated and reliable cloud-based inventory management system.

I am especially grateful to my family for their unconditional support, patience, and motivation throughout the project duration. Their encouragement has been vital in helping me overcome challenges and remain focused.

Lastly, I would like to acknowledge the lecturers, peers, and testers who contributed feedback during the testing and evaluation phase. Their input was instrumental in identifying areas for improvement and ensuring the reliability of the final system. This project would not have been possible without the support and contributions of all these individuals.

Table of Contents

Chapter 1

Introduction

1 Introduction

This chapter introduces a project titled Business Process Automation System for Inventory Management, designed to modernize and digitize inventory processes for small and medium-sized enterprises (SMEs). The system addresses common challenges SMEs face, such as inaccurate stock records, inefficient procurement processes, and a lack of real-time data visibility, by automating essential functions through a cloud-native solution. It integrates real-time stock monitoring, predictive purchase order automation, supplier coordination, cloud-based alerting services, and user access control.

Built on Amazon Web Services (AWS), the system employs a fully serverless backend architecture, supporting scalability, reliability, and operational efficiency while minimizing infrastructure overhead. The system provides secure, role-based access and centralized reporting, enabling SMEs to improve decision making, streamline operations, and reduce dependency on manual inventory practices. This chapter outlines the overall project objectives, development background, scope definition, technological context, module contributions, development schedule, and team structure.

1.1 Project Objectives

The objective of this project is to develop a cloud-based inventory management system for Small and Medium-sized Enterprises (SMEs), focused on automating core inventory operations and enhancing overall visibility and control of stock-related activities. The system leverages serverless cloud technologies to reduce manual workloads while promoting operational scalability, accessibility, and data-driven decision-making.

1.1.1 Automate Stock Reordering Based on Thresholds

This objective focuses on implementing a predictive reordering mechanism that automatically generates purchase orders when stock levels fall below predefined thresholds. This eliminates the need for manual stock checks and prevents stockout events, particularly in businesses with high-volume or fast-moving inventory. By integrating auto-triggered workflows, the system ensures that replenishment is timely and consistent with business needs (Sharma & Guleria, 2021).

1.1.2 Streamline Supplier Coordination and Communication

The system is designed to improve supplier-related operations by centralizing supplier profiles and automating the purchase lifecycle. The system stores supplier details, tracks PO history, and automates communication between buyers and vendors through approval flows

and instant alerts. This creates a seamless purchasing process that minimizes delays and ensures traceability.

1.1.3 Provide Real-Time Alerts and Notifications

The system incorporates a notification module to deliver real-time alerts for critical events, such as low stock detection, new purchase order generation, and pending approvals. These alerts allow relevant users to respond quickly to inventory-related issues, reducing the risk of disruptions and enhancing coordination across departments (Naveed S et al., 2024).

1.1.4 Enable Cloud-Based Data Monitoring and Access

The system enhances data visibility by hosting inventory and order data on a secure cloud platform. This allows authorized users to access inventory dashboards, view stock trends, and monitor purchasing activity remotely, supporting mobile work environments and off-site operations.

1.1.5 Minimize Manual Effort through Business Process Automation

By shifting key tasks such as stock tracking, reorder generation, and notification dispatch into automated workflows, the system aims to reduce repetitive administrative work. This supports staff efficiency and frees up resources to focus on more strategic tasks, enabling SMEs to scale without investing in costly IT infrastructure (Benedict et al., 2023).

1.2 Project Background

Inventory management is a core part of daily operations for SMEs, but many still rely on manual processes such as stock cards to monitor their inventory levels. These outdated methods often result in stock shortages, human errors, and delays in reordering. Without a proper system in place, businesses face difficulties in knowing when to restock, which can lead to lost sales or overstock situations. This project was developed to help automate the reordering process by introducing a threshold-based stock control mechanism that can generate purchase orders automatically when stock levels are too low.

In addition to stock monitoring, communication with suppliers is also one of the challenges faced by SMEs. Many businesses use different platforms or manual paperwork to handle purchase orders and supplier details, which causes delays and lack of traceability. To solve this, the system aims to centralize all supplier information and automate the communication flow. This includes storing supplier profiles, tracking order history, and sending instant notifications when a PO is created or needs approval.

Another issue in current SME operations is the absence of a real-time alert system to inform users of urgent inventory events. For example, many businesses do not know when their stock is about to run out or when a PO is still pending for approval. Without timely alerts, staff may miss important actions that affect stock availability. To overcome this, the proposed system integrates a real-time notification module to alert relevant users about low stock, new purchase requests, and PO status updates.

Moreover, most traditional inventory systems require users to be physically present in the office to access stock records. This is inconvenient, especially for businesses with remote staff or multiple branches. By shifting the system to a cloud environment, users can now access the dashboard and inventory data from any device with internet connection. This makes stock tracking and order management more flexible and responsive, even when working outside the office.

Lastly, many SMEs lack manpower or technical staff to manage their inventory system efficiently. Repetitive tasks such as checking stock levels, generating POs, and updating records can take up a lot of time. This project helps to reduce manual work by introducing automation for key business processes. With cloud services like AWS Lambda and SNS, the system can handle these operations in the background, allowing staff to focus on more important tasks without worrying about technical complexity.

1.2.1 Technology Context

The system is powered by a modern serverless architecture using various Amazon Web Services (AWS) to achieve high availability, automation, and security. AWS Lambda serves as the core automation engine, executing backend logic such as inventory checks and purchase order generation without the need for dedicated servers. This improves scalability and cost-efficiency, particularly important for small businesses.

The user interface is built using standard web development technologies including HTML, CSS, JavaScript, and PHP, allowing dynamic and responsive page rendering for modules such as supplier management, PO approval, and login access. PHP is used to handle server-side scripting and integrates with AWS services for tasks such as user authentication and database interaction.

Data is stored in Amazon RDS, which hosts a MySQL database managing inventory records, supplier profiles, and transaction histories. Supplier documents and generated purchase orders are securely stored in AWS S3, ensuring centralized document management. Real-time notifications such as PO confirmations and login alerts are sent via AWS SNS through email.

Access control is implemented using application-level logic, allowing role-based permissions for Admin, Manager, and Staff users. AWS CloudWatch is used to monitor database

performance and services logs. Combined, these technologies provide a reliable and secure infrastructure that supports the system's automation, scalability, and maintainability.

1.3 Advantages and Contributions

This project delivers meaningful contributions to the SME sector by addressing limitations found in traditional, manual inventory methods. Through automation, real-time visibility, and integration of cloud services, the system provides a smarter and more responsive approach to inventory management.

1.3.1 Predictive Reordering

The predictive reordering module ensures that inventory replenishment is timely and responsive to actual demand. By generating POs automatically based on stock thresholds, the system prevents stock shortages and reduces the risk of business interruptions. This also cuts down procurement time and supports lean inventory strategies.

1.3.2 Centralized Cloud Access

With all key inventory operations such as stock tracking, supplier records, and reporting centralized in the cloud, users benefit from real-time access and streamlined coordination. Cloud storage and computation eliminate the need for on-premises infrastructure, making the system cost-effective and scalable for small businesses (Benedict et al., 2023).

1.3.3 Real-Time Alerts and Notifications

The integrated alert system notifies relevant users of low stock levels, approval requests, or system issues, enabling immediate action. This transparency reduces communication gaps and supports faster decision-making. It also creates a historical log of actions, improving accountability and traceability (Naveed S et al., 2024).

1.3.4 Increased Operational Efficiency

By automating repetitive tasks like stock scanning, supplier matching, and PO generation, the system reduces the workload on staff. This allows businesses to operate with smaller teams while maintaining or improving accuracy. SMEs can better allocate resources and reduce time lost to manual errors (Pangaribuan et al., 2022).

1.3.5 Scalable Serverless Architecture

The use of serverless technologies enables the system to scale automatically with workload changes. Functions such as report generation or stock checks run only when triggered, which

minimizes costs and improves responsiveness. This architectural design is well-suited for SMEs that require flexible systems without the burden of high IT maintenance.

1.3.6 Integrated Inventory Workflow

Unlike traditional inventory systems where procurement, supplier, and alert functions operate separately, this system integrates all core processes into one cohesive flow. For instance, when inventory drops below a certain level, a PO is automatically generated, matched with a supplier, and sent for approval without user intervention.

1.3.7 Supporting SME Digital Transformation

This project supports the digitalization of inventory processes in SMEs, which often lack the resources to adopt large-scale ERP systems. By offering a lightweight, cloud-integrated, and automated solution, the system enables local enterprises to compete more effectively in a data-driven market.

1.4 Project Plan

1.4.1 Project Scope

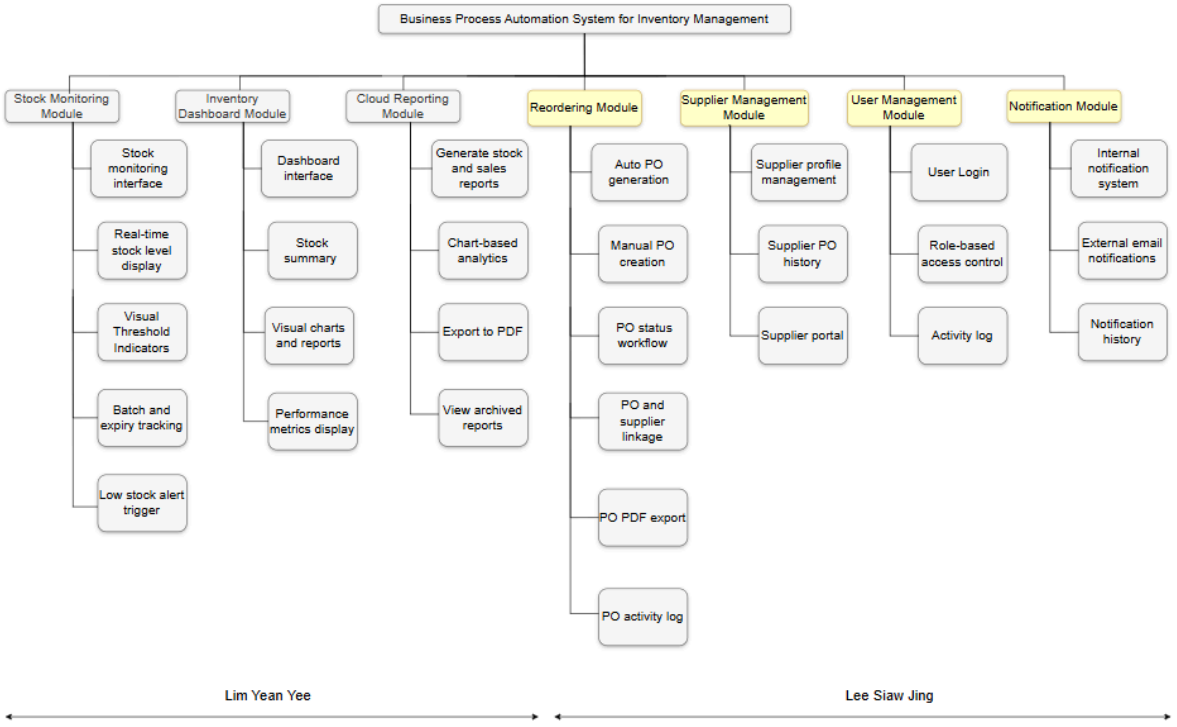


Figure 1.1 Functional Decomposition Diagram

The Reordering Module, Supplier Management Module, User Management Module, and Notification Module are developed and managed by Lee Siaw Jing as part of the Business Process Automation System for Inventory Management. The Reordering Module is designed

to automate and streamline the purchase order (PO) process, allowing the system to generate POs automatically when stock drops below a set threshold, while also enabling users to manually request POs, go through an approval process, link POs with suppliers, export them as PDFs, and log all related actions for tracking purposes. The Supplier Management Module handles supplier-related tasks, including maintaining supplier profiles, viewing PO history, and managing delivery contacts and records to ensure smooth procurement operations. The User Management Module provides a secure login and registration interface for Admin, Manager, and Staff users, supports role-based access control, and maintains an activity log to monitor all user actions within the system. The Notification Module is responsible for sending real-time alerts for low stock, item expiry, and PO status updates, while also storing all notification history for reference and audit purposes. These modules collectively support seamless procurement flow, supplier coordination, and efficient inventory handling for SMEs.

1.4.2 Key Milestones

The key phases, activities, and development timeline for this project, specifically under the scope handled by Lee Siaw Jing, are outlined in Table 1.1. These milestones represent the individual contributions made throughout the project lifecycle from planning and requirements analysis to implementation, integration, testing, and final documentation.

Table 1.1: Development Milestones

Phase	Activities	Expected Completion Date
1. Planning	Finalize module scope: PO automation, supplier management, user management notification.	17 April 2025
2. Requirements Analysis	Identify detailed functional and cloud-related requirements for PO flow, supplier records, notifications, and user roles	30 June 2025
3. System Design	Design workflows for PO automation, approval, access control structure, audit trail, and CloudWatch integration	18 August 2025
4. Implementation	- Develop supplier management module - Implement PO automation via Lambda - Add PDF generation & audit trail	20 September 2025

5. Integration	- Integrate SNS for system alerts - Display notification logs - Configure CloudWatch metrics	10 October 2025
6. Testing	- Conduct unit testing on each developed module - Perform joint system integration testing with teammate	28 October 2025
7. Deployment & Monitoring	Finalize S3 integration, enable CloudWatch logging.	5 November 2025
8. Documentation	Document module design, workflows, testing results, and diagrams for PO lifecycle and cloud integration	19 December 2025

1.5 Project Team & Organization

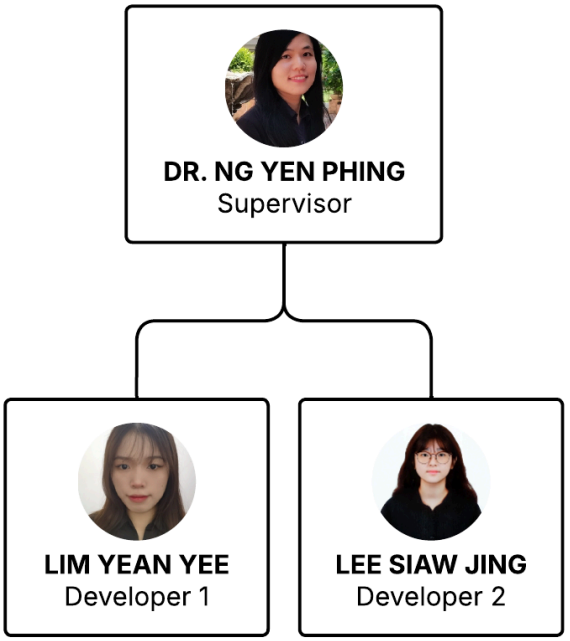


Figure 1.2 Project Organization Chart

A. Supervisor: Dr. Ng Yen Phing
   - ❖ Provides overall supervision, offers technical direction, and ensures that the project meets academic standards and objectives.

B. Developer 1: Lim Yean Yee
   - ❖ Responsible for the Stock Monitoring & Threshold Alerts Module and the Inventory Dashboard & Cloud Reporting Module.
   - ❖ Tasks include stock tracking UI and expiry monitoring, threshold alerts, visual dashboards, and exportable inventory reports.

C. Developer 2: Lee Siaw Jing
   - ❖ Responsible for the Purchase Reordering & Supplier Management Module and the User Management & Notification Module.
   - ❖ Tasks include supplier profile management, automated purchase order generation, approval workflows, PDF generation, role-based user access, and system notifications.

1.6 Chapter Summary and Evaluation

This project has introduced a cloud-based inventory management system designed to automate critical business processes for small and medium-sized enterprises. The main goals include predictive reordering, supplier coordination, real-time alerts, and secure user access, all built on a serverless AWS infrastructure. The scope of the system has been clearly defined through a modular approach, with each team member fully responsible for the development of specific functional areas. My individual focus is on the Reordering, Supplier Management, and Notification modules, which are tightly integrated with AWS services to enable automation and reliability.

The operational gaps in traditional SME inventory practices, reinforcing the need for real-time, scalable solutions. The technology stack, especially the use of AWS Lambda, RDS, SNS, and CloudWatch, supports efficient automation and secure data handling. The division of tasks and roles between me and my teammate Ms. Lim Yean Yee ensures parallel development and smooth integration. The milestone schedule further outlines how the project progresses through planning, implementation, testing, and deployment.

Overall, this project lays the groundwork for a practical, cost-effective solution that meets the digital needs of SMEs. It sets a clear direction for the chapters ahead, which will explain the detailed system design, implementation strategies, and technical contributions for the modules under my responsibility.

Chapter 2

Literature Review

2 Literature Review

This chapter explores the relevant studies, existing systems, and supporting technologies that justify the development of a cloud-based business process automation system for inventory management. It begins by reviewing the current challenges SMEs face in managing inventory manually, followed by an in-depth review of automation technologies and serverless cloud architecture. Comparisons are drawn between existing solutions and the proposed system to highlight areas of improvement. Additionally, this chapter covers the feasibility analysis across economic, technical, and operational dimensions to evaluate the practicality of system implementation for SMEs.

2.1 Company Background

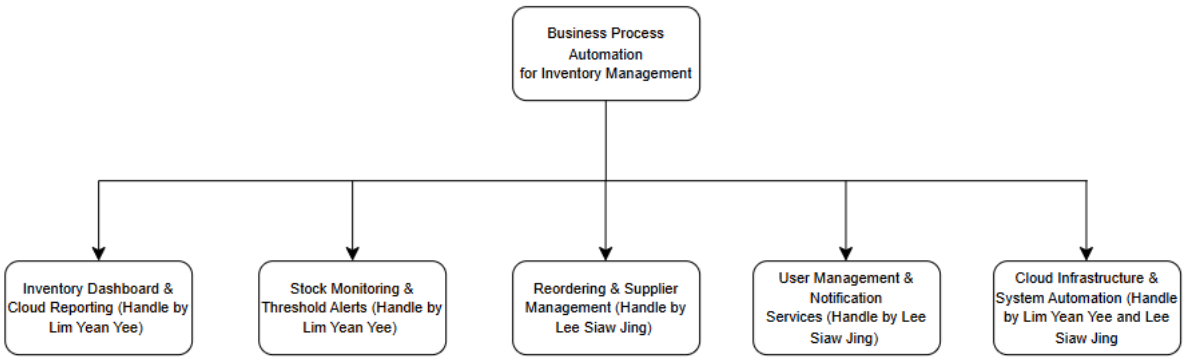


Figure 2.1 Company Background Chart

The project team is composed of two members, each assigned to distinct functional modules within the Business Process Automation System for Inventory Management. Lim Yean Yee oversees the Inventory Dashboard & Cloud Reporting and Stock Monitoring & Threshold Alerts modules, with a focus on enabling real-time stock tracking, monitoring expiry dates, and generating analytical reports. Meanwhile, Lee Siaw Jing is responsible for the Reordering & Supplier Management and User Management & Notification Services modules, which involve automating procurement processes, maintaining supplier records, and configuring system alerts. Both members collaborate on the Cloud Infrastructure & System Automation module to ensure seamless integration of AWS services, including Lambda, RDS, S3, and CloudWatch.

2.1.1 Development Team

Currently, the system is not backed or adopted by any external organization. It was developed as part of an academic initiative to support local small and medium-sized enterprises (SMEs) in enhancing their inventory operations. The system is designed to automate essential tasks such as stock tracking, reorder management, and report generation. Additionally, its

architecture allows for future scalability, making it suitable for larger enterprises seeking flexible, cloud-based inventory solutions.

This project was conceptualized and implemented by two final-year students under the supervision of a faculty advisor. It aims to modernize traditional inventory practices by introducing automated workflows powered by cloud computing. The primary focus is on minimizing human error, ensuring data accuracy, and improving operational responsiveness through real-time insights and automated notifications.

The solution is a web-based platform hosted on Amazon Web Services (AWS), offering secure, role-based access for multiple types of users. Its main objective is to centralize inventory processes, allowing businesses to improve operational efficiency and reduce costs. Although initially developed for local SMEs, the system is designed to accommodate future growth and adoption by a wider user base, including regional and international markets.

2.1.2 Potential Competitor

There are several existing inventory management systems in the market that offer features such as cloud-based automation and integrated reporting, which are comparable to the proposed system.

A. Zoho Inventory

Zoho Inventory is a well-known cloud-based inventory management solution that enables businesses to monitor stock levels, handle order processing, and produce sales analytics. It is particularly favored by small to medium-sized enterprises due to its intuitive user interface and compatibility with other Zoho tools. Nonetheless, the platform operates on a subscription model, and many advanced features are locked behind premium pricing tiers (Inventory Management Software | Online Inventory Management for Businesses - Zoho Inventory, n.d.).

B. inFlow Inventory

inFlow Inventory offers both desktop and cloud-based functionality, assisting businesses in managing inventory, issuing invoices, and maintaining order histories. It is equipped with features such as barcode support and real-time synchronization, making it suitable for SMEs. However, the system may lack flexibility and customization options, especially for users requiring sophisticated automation or scalable cloud capabilities (inFlow Inventory, 2025b).

C. Oracle NetSuite

Oracle NetSuite is a comprehensive enterprise-grade ERP platform that integrates inventory control, financial management, and customer relationship functionalities. It is primarily

designed for large organizations and corporations. While it delivers a wide range of powerful tools, its complexity and high implementation costs make it less accessible for small and medium enterprises with limited financial and technical resources (Hypernix Sdn Bhd, 2025).

2.2 Review of Existing Solutions and Technologies

2.2.1 Overview of Inventory Management in SMEs

Inventory management is essential for SMEs, but many still rely on manual systems that cause inefficiencies and errors. For example, Toko Rempah Kristal managed sales, purchases, and stock manually, leading to issues like miscalculations, lost invoices, and delayed decisions (Pangaribuan et al., 2022). These challenges are common in SMEs with high daily transactions.

Manual processes make it difficult to maintain accurate records and require time consuming checks. This affects overall business performance. The study shows that implementing an automated system helps overcome these problems by streamlining invoice creation, stock updates, and profit tracking. As a result, SMEs can improve data accuracy, reduce errors, and make faster, more informed decisions.

2.2.2 Role of Business Process Automation in Inventory Control

Business Process Automation (BPA) is essential in inventory management to handle repetitive and rule-based tasks with speed and accuracy. In an automated inventory control system, processes such as email ingestion, data extraction, ticket assignment, and task completion can be executed without human intervention, ensuring consistency and reducing the risk of human error (Sharma & Guleria, 2021).

These automation techniques, though applied in the hotel and travel industry, are highly relevant to SMEs managing inventory manually. Automation of stock monitoring, supplier coordination, and documentation improves real-time data handling and streamlines operations. With tools like AWS Lambda and SNS, inventory systems can trigger purchase orders and notifications automatically based on stock thresholds.

BPA enhances overall control, reduces manual workload, and enables faster decision making in procurement processes. It shifts routine tasks to automated workflows, allowing SMEs to operate more efficiently while maintaining accuracy in their inventory records.

2.2.3 Cloud-Based Solutions and Serverless Architecture

Serverless architecture provides SMEs with a scalable and cost-effective way to manage inventory systems by executing functions only when triggered, which reduces the need for always-on servers and lowers infrastructure costs (Benedict et al., 2023). This allows developers to focus on building system features while backend operations like scaling and maintenance are handled by cloud providers.

Platforms such as AWS Lambda support event-driven execution and automatic scaling, making them suitable for systems with varying workloads. By applying appropriate serverless patterns like event-driven or flow-driven, SMEs can further optimize cost and performance. Overall, serverless cloud architecture supports efficient, flexible, and affordable solutions for inventory management.

2.2.4 Comparative Review of Existing Inventory Systems

Many inventory systems have been developed to address the limitations of manual methods such as spreadsheets or paper-based records. Web-based platforms are commonly used to automate stock updates, reduce errors, and provide better data accessibility. Mobile-based inventory systems are also rising in popularity among SMEs due to their real-time features and user-friendly interfaces (Madamidola et al., 2024).

Some systems use technologies like barcode scanning, RFID, and IoT to improve tracking accuracy and minimize human error. However, despite these advancements, many systems still operate as standalone solutions without intelligent features or ERP integration. Common limitations include a lack of predictive analytics, data security concerns, and poor scalability in larger environments.

A comparative review shows that while existing systems offer improvements in usability and automation, most are not fully optimized for SMEs with cloud-based, serverless needs. This project aims to bridge that gap by implementing a scalable, cost-effective solution using AWS services tailored for SME inventory processes.

2.2.5 Importance of Real-Time Alerts and Notifications in Inventory Systems

Real-time alerts play a crucial role in improving the efficiency and responsiveness of inventory management. By integrating sensors and IoT platforms, systems can instantly notify users of low stock levels or abnormal consumption patterns, helping to prevent shortages and reduce manual tracking errors. In the hostel inventory system developed using the Blynk platform and ESP32 microcontroller, administrators receive immediate email notifications when stock levels drop below a set threshold (Naveed S et al., 2024).

This feature allows staff to respond quickly to inventory changes, ensuring timely restocking and better resource planning. The system also includes an LCD display for visual updates and uses load cells for accurate measurement. Such real-time alert mechanisms minimize the risk of supply disruptions and enable more proactive decision-making in managing consumables like food and cleaning supplies.

2.3 Economic Feasibility

2.3.1 Costs

The total estimated cost of the project is categorized into four main areas: development cost, hardware cost, software cost, and cloud infrastructure cost. These costs are designed to simulate real world implementation within a constrained SME budget.

A. Development Cost

Development efforts are aligned with the project milestones outlined in Table 2.1. Each phase involves system planning, design, coding, integration, and testing. As the project is developed by two student members, the cost is estimated based on a modest IT internship monthly rate of RM1,453 per person (Indeed, 2025). Refer to Table 2.1 for the estimated development cost across the eight project phases.

Table 2.1: Development Cost Estimation

Development Task	Team Members	Duration (Months)	Monthly Cost per Member	Estimated Cost (RM)
Project Planning & Research	2	1	RM1,453	RM2,906
System Design	2	1	RM1,453	RM2,906
System Development	2	3	RM1,453	RM8,718
Testing & Implementation	2	1	RM1,453	RM2,906
Total Development Cost				**RM17,436**

B. Hardware Cost

Each developer uses a personal laptop that meets basic development needs. The estimated hardware cost is outlined in Table 2.2. The average unit cost of RM2,500 for an Intel Core i5, 8GB RAM, 512GB SSD laptop is based on market research (Harvey Norman, 2025). This

table also includes RM199/month for a reliable Time Internet 1Gbps plan (Time, 2019), the internet is used throughout the 6-month project period. A complete breakdown of the total estimated operational cost can be found in Table 2.2.

Table 2.2: Hardware Cost Estimation

Hardware	Specifications	Unit Cost	Quantity	Duration (Months)	Total Cost (RM)
Laptop/Desktop	Intel Core i5, 8GB RAM, 512GB SSD	RM2,500	2	-	RM5,000
Internet	WiFi 7 router, 1Gbps	RM199	1	6	RM1,194
Total Hardware Cost					**RM6,194**

C. Software Cost

The software tools used in this project include open-source IDEs, free diagramming tools, and student-licensed productivity software. These significantly reduce the overall software budget. The unit cost for Windows OS is estimated at RM499 (PC Image, 2025). Refer to Table 2.3 for the complete breakdown of software costs.

Table 2.3: Software Cost Estimation

Software	Minimum Requirements	Unit Cost	Estimated Cost (RM)
Windows OS	Microsoft Windows 10 or above	RM499	RM998
Coding Application	Visual Studio Code (Open Source)	RM0	RM0
Database Tool	MySQL	RM0	RM0
Office Suite	Microsoft Word (Student License)	RM0	RM0
Diagram Tool	Draw.io & Lucidchart (Free version)	RM0	RM0
Total Software Cost			**RM998**

D. Cloud Infrastructure Cost

The system will utilize Amazon Web Services (AWS) as its cloud infrastructure. This section provides an estimated monthly cost for AWS services, and a projected total cost for a six

month period, specifically tailored to a medium-sized company's operational needs during development and testing. These figures are approximations because the pay-as-you-go cloud pricing model means actual costs will vary directly with usage. All estimations use an exchange rate of 1 USD = 4.25 MYR. Refer to Table 2.4 for a complete breakdown of the estimated AWS service costs.

- **Scenario: Medium-Sized Company Inventory Management System Usage**
- For this cost projection, we assume a medium-sized company with 50-100 active users and 50,000-100,000 monthly transactions. This includes moderate data storage for product images and historical data, several thousand automated notifications, continuous system monitoring, and regular data backups to ensure reliability.

Table 2.4: Cloud Infrastructure Cost Estimation

No.	AWS Service	Monthly Cost (RM)	6 Month Cost (RM)
1	AWS Lambda	RM 25	RM150
2	AWS RDS	RM190	RM1,140
3	AWS S3	RM30	RM180
4	AWS SNS	RM10	RM60
5	AWS CloudWatch	RM55	RM330
6	AWS EC2	RM60	RM360
Total Cloud Infrastructure Cost		**RM370 (Monthly)**	**RM2,220 (Six month)**

1. AWS Lambda

The estimated monthly cost is RM 25, based on 1.5 million requests (0.5 million paid) at $0.20 per million requests (approx. RM 0.85) and 600,000 GB-seconds of execution (200,000 paid) at $0.0000166667 per GB-second (approx. RM 0.0000708). The 6-month cost is RM 150 (AWS, 2024c).

2. AWS RDS

A db.t3.medium instance with 100 GB of gp2 storage is estimated to cost RM 190 monthly, factoring in instance type ($30-$50 USD/month or RM 127.50 - RM 212.50) and storage ($0.115 per GB-month or RM 0.489 per GB-month for 100 GB = RM 48.90), along with I/O. The 6-month cost is RM 1140 (AWS, 2025).

3. AWS S3

Storing 200 GB of Standard storage ($0.023 per GB or RM 0.09775 per GB) and handling 1 million PUT/GET requests ($0.005 per 1,000 PUT/COPY/POST or RM 0.02125 per 1,000, and $0.0004 per 1,000 GET/SELECT or RM 0.0017 per 1,000) results in an estimated monthly cost of RM 30. The 6-month cost is RM 180 (AWS, 2024a).

4. AWS SNS

For 50,000 notifications, a modest monthly cost of RM 10 is estimated to cover potential charges for specific delivery types, with requests beyond the free tier at $0.50 per million (RM 2.125). The 6-month cost is RM 60 (AWS, 2024b).

5. AWS CloudWatch

Monitoring costs for 50 custom metrics ($0.30 per metric or RM 1.275 per metric), 20 GB of logs ($0.50 per GB or RM 2.125 per GB), and 10 alarms ($0.10 per alarm or RM 0.425 per alarm) are estimated at RM 55 monthly. The 6-month cost is RM 330 (AWS, n.d.-a).

6. AWS EC2

Running a t3.micro instance for hosting the web application is estimated to cost RM 60 per month. This calculation considers the on-demand instance rate of approximately $0.0104 per hour (≈ RM 0.0442), which amounts to roughly RM 32–35 monthly, along with EBS storage charges for a 20–30 GB gp2 volume at $0.10 per GB-month (≈ RM 0.425 per GB). Including storage and data transfer estimation, the overall monthly cost is rounded to RM 60. Over six months, the projected cost is RM 360 (*Pricing*, n.d.-b).

2.3.2 Tangible Benefits

A. Reduced Operational Spending on Inventory Management

The system significantly decreases the financial burden associated with manual inventory tracking. Automated stock monitoring and order alerts reduce unnecessary overstocking and avoid losses from expired or slow-moving products. This leads to a more balanced and cost-efficient inventory level, which in turn improves working capital for SMEs.

B. Elimination of Printing and Storage Requirements

By fully digitizing purchase orders, supplier records, and inventory reports, the system eliminates the need for physical documentation. Companies can cut costs on printing, filing,

and renting physical storage spaces. With data stored securely on AWS S3 and RDS, documents remain accessible and organized without occupying office space.

C. Lower Workforce Costs for Reordering Operations

Manual reordering processes typically require several employees to handle supplier follow-ups and paperwork. With automated PO generation and supplier status tracking, fewer staff are needed to manage procurement. This frees up resources that can be reallocated to business development or customer service tasks.

D. Reduction in Mistakes That Lead to Costly Errors

Human errors in order quantity, pricing, or supplier selection can result in wasted purchases. By integrating rule-based automation through AWS Lambda and predefined thresholds, the system minimizes such errors. This reduces the risk of duplicate orders, miscommunication, or stock imbalances.

2.3.3 Intangible Benefits

A. Faster Order Turnaround and Increased Responsiveness

The system streamlines the ordering process by instantly triggering purchase orders based on real-time stock data. This reduces lead time and enhances supplier responsiveness, which helps ensure continuity in product availability.

B. Better Visibility for Strategic Procurement Decisions

With dashboard analytics and visual reports, managers gain improved insights into stock movement and supplier performance. These tools support smarter decisions-related to budget allocation, seasonal planning, and supplier negotiation.

C. Strengthened Data Security and Auditability

The system leverages secure session handling and logging services to track user activity and secure sensitive procurement records. This builds trust and transparency across departments, ensuring accountability in purchasing decisions and access control.

D. Enhanced Brand Reliability Through System Consistency

Automated alerts and timely restocking help maintain consistent service levels. This enhances the company’s reliability to customers by minimizing out-of-stock situations or order delays, indirectly improving customer satisfaction and brand perception.

2.3.4 Costs & Benefit Analysis

The cost and benefit analysis evaluates whether the long-term benefits justify the initial investment. While the initial year requires the highest investment for development and cloud setup, operational savings and automation benefits grow over time. Refer to Table 2.5 for the projected costs and benefits over a five year period.

Table 2.5: Five Year Cost and Benefit Projection

Year	Total Cost (RM)	Tangible Benefits (RM)	Intangible Benefits (RM)	Total Benefits (RM)	Net Value (RM)
0	-26,968	10,000	5,000	15,000	-11,968
1	-1,200	16,000	8,000	24,000	22,800
2	-1,200	24,000	10,000	34,000	32,800
3	-1,200	31,000	12,000	43,000	41,800
4	-1,200	38,000	14,000	52,000	50,800
5	-1,200	45,000	16,000	61,000	59,800

As shown in Table 2.5, the system begins generating positive net value from the first year after implementation. By the fifth year, the cumulative net gain reached RM59,800, confirming the project's long-term financial viability and contribution to operational efficiency.

2.4 Operation Feasibility

2.4.1 Internal Operations: Staff and Inventory Users

On a daily basis, general staff members involved in inventory handling will use the system to monitor stock levels, track low stock alerts, and record expiry data. They are responsible for updating inventory records in real-time, reviewing system generated alerts, and preparing for upcoming purchase orders. The stock monitoring dashboard enables staff to view item availability, expiry status, and threshold indicators, simplifying their operational workflow.

2.4.2 Middle Management: Inventory and Purchasing Supervisors

Managers or supervisors have access to more advanced features such as reviewing and approving purchase orders generated by the system. They are able to review supplier records, analyze reorder suggestions, and approve or reject system generated purchase orders before they are finalized. Additionally, they can monitor supplier performance and access notification logs for tracking important system alerts. Managers are also responsible for adjusting threshold levels, setting approval workflows, and overseeing cloud reports to support their decision making.

2.4.3 Upper Level Management: Business Owners or Directors

Higher-level users, such as business owners or department heads, use the system to extract reports and analyze trends from the inventory dashboard. They rely on visual charts and exportable data to evaluate stock performance, supplier reliability, and overall procurement efficiency. These insights allow upper management to make strategic decisions such as supplier switching, budget planning, or sales forecasting, all based on real-time data.

2.4.4 External Users: Suppliers

Once a purchase order is generated and approved, the system sends a notification to the relevant supplier using AWS SNS. Suppliers will receive these alerts via email or SMS, prompting them to prepare and deliver the required goods. Supplier profiles are also managed within the system, allowing for easy updates to contact details and business documents. This streamlines communication and minimizes delays in procurement transactions.

The system is designed to be intuitive, requiring minimal training for each user role. Each module is accessible based on user permission levels defined through system role logic. This ensures that every operation, from inventory checks to procurement approvals and external supplier interactions, is handled securely and efficiently within the system environment.

2.5 Chapter Summary and Evaluation

This chapter shows how inventory systems are important for SME and what problems they still face today. Many SMEs are still using manual methods, causing human error and slow process. The literature also highlights how automation, like using AWS Lambda and SNS, can make processes faster and reduce mistakes. Reviews also show that serverless cloud solutions give big help in cost-saving and easy scaling.

After that, this chapter compares some existing systems like Zoho, inFlow and NetSuite. Most of them are good but too expensive or too complex for small businesses. This project targets SME by giving more simple and cheaper options. The study from another journal also shows how real-time alert and supplier notification can make operation more smooth and responsive.

Feasibility studies prove that this project is realistic and useful. Cost and benefit analysis show long-term value. Technical check show systems can run smoothly with cloud tools and shared teamwork. This chapter help to give strong support and reason why this system is needed, and ready to build and test in the next phase.

Chapter 3

Methodology and Requirements Analysis

3 Methodology and Requirements Analysis

This chapter explains the development approach used to build the Business Process Automation System for Inventory Management. It covers the chosen development methodology, methods used to collect system requirements, and a breakdown of the functional and non-functional needs. In addition, this chapter explains the hardware and software needed, team responsibilities, system integration readiness, and overall project cost. By following this structured approach, the development process remains systematic and well-managed from both technical and operational perspectives.

3.1 Methodology

3.1.1 Proposed Methodology

For this project, the Waterfall Model has been selected as the primary development methodology. The Waterfall model is a linear and sequential approach that emphasizes thorough documentation and structured phase transitions, where each stage must be completed before moving on to the next. This model is suitable for this project because all the functional and non-functional requirements were clearly identified and confirmed during the early planning stage.

The system development process did not require frequent changes or iterative modifications, making a well-structured model like Waterfall a practical choice. Additionally, the development was planned within a fixed timeline with specific deliverables at each stage, which aligns well with the Waterfall Model's emphasis on phase-by-phase completion. This approach also supports better project tracking, documentation, and accountability, especially in academic or client-based settings where changes during development are minimal. The phases involved in the Waterfall methodology applied for this system development are illustrated in Figure 3.1.

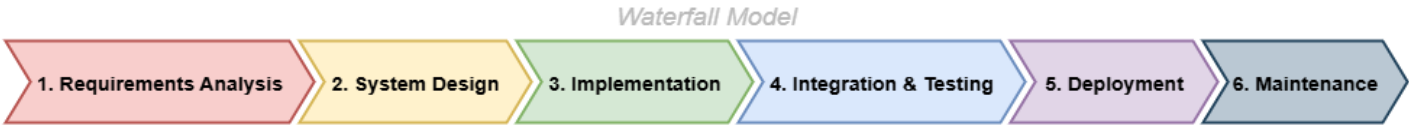


Figure 3.1: Waterfall Model Diagram

3.1.2 Description of Each Phase

Phase 1: Requirements Analysis

In this phase, the project team gathered all necessary requirements related to the system. Information was collected through observation and discussion to define the features needed

for stock monitoring, supplier coordination, automated purchasing, and cloud-based reporting. The outcome was a clear documentation of system expectations to guide the next stages.

Phase 2: System Design

Based on the gathered requirements, the technical structure of the system was planned. This included designing database schemas, UI, and defining how AWS services like Lambda, RDS, and SNS would be used. Proper design ensured that each module could work smoothly with the cloud infrastructure.

Phase 3: Implementation

The development stage involved coding and building all modules using PHP, JavaScript, HTML/CSS, and AWS components. Developed supplier management, PO automation, and notification modules, while Lim Yean Yee worked on stock monitoring, dashboard visualizations, and cloud reporting.

Phase 4: Integration & Testing

After development, all modules were combined into a single system. Testing was done to check individual features and how well they worked together. Automated alerts, PO generation, and role-based access were tested thoroughly. Cloud tools like AWS CloudWatch were used to verify system stability.

Phase 5: Deployment

The system was then deployed to the AWS environment. All necessary services such as user roles, notification delivery, cloud storage, and monitoring were fully configured. The web system was made accessible through a secure login for Admin, Manager, and Staff users.

Phase 6: Maintenance

Even though this is an academic project, maintenance planning was included. Cloud-based tools allow for ongoing data backup, performance monitoring, and easy upgrades. The use of serverless architecture ensures the system can scale and evolve with minimal effort in the future.

3.2 Requirement Gathering Techniques

3.2.1 Interview

An interview session was carried out with Mr. Nick Lee, the founder of Green Life Design Sdn Bhd, who has over 20 years of experience working as an operation manager at Timeless Design Sdn Bhd. He has been actively involved in inventory management, supplier handling, logistics, and sales operations. The purpose of this interview was to understand the current

system in use, gather opinions about inventory-related workflows, and collect suggestions for system improvement based on real business experiences. The interview was conducted by Lee Siaw Jing, while Lim Yean Yee was responsible for recording and taking down the key points of the discussion.

The company is currently using a CRM system provided by Ofcom Solution, which they subscribe to for RM1500 per year. While the system supports basic order management and customer handling, the interviewee shared that there are still many areas that can be improved especially in stock handling, automation, and reporting features.

Below are some of the key questions asked during the interview and the responses provided:

Q1: What system is your company using now?

A1: "We're using the CRM from Ofcom Solution. It helps with inventory, suppliers, and logistics management."

Q2: Any suggestions to improve how POs and suppliers are managed?

A3: "Now every time we want to create a PO, we have to manually pick the product and supplier one by one. It's very easy to make mistakes. Sometimes we forget which supplier we normally use. If the system can save supplier info and link to previous orders, that would make things a lot easier. At least everything is in one place, no need to check here and there."

Q3: How do you feel about the idea of auto-generating POs and sending them to suppliers?

A4: "I think it will be a very useful function, sometimes we're too busy and forget to reorder things in time. If the system can auto-generate a PO when stock is low and even send it straight to the supplier, maybe after approval that would be efficient."

Q4: What about showing stock levels using color indicators, like red for low stock?

A5: "That will be really helpful, especially when there are a lot of items to manage. If I can just see which items are running low based on color, I won't have to check one by one."

Q5: Do you think system notifications and logs would help your team?

A2: "Yes, sometimes orders or deliveries get delayed and we don't realise until it's too late. If the system can notify us immediately when something changes like PO status, low stock, or expired items it would save a lot of time. And having a log to check back what happened would help when we need to explain the issue."

Q6: How do you currently handle reporting and reviews?

A6: “Not great, to be honest. The reports we get are quite limited. If the system can generate proper reports like best selling items, low stock lists, or monthly summaries and allow us to export them to PDF, that would be very useful for management meetings or audits.”

From this interview, several useful insights were gathered. The current system lacks features like supplier tracking, automatic purchase orders, visual stock alerts, and meaningful reports. The interviewee's suggestions played a big role in shaping the system design, particularly the inclusion of real-time notification services, auto-PO generation via AWS Lambda, supplier management with PO history, threshold-based alerts, and exportable cloud-based reports. These align with the project modules such as Stock Monitoring, Reordering, Notification Services, Inventory Dashboard, Supplier Management, and Cloud Reporting.

Figure 3.2: Lee Siaw Jing (me) is conducting an interview with Mr. Nick Lee.

3.2.2 Observation

An observation session was also carried out to better understand the existing workflow and identify inefficiencies in the current system of Green Life Design and Timeless Design. By observing how staff perform tasks using their current platform, it becomes easier to see where the process slows down and where manual work is still heavily involved, despite having a digital system in place.

It was noticed that while the system helps with basic order entries and customer data, the process of creating a purchase order still requires users to go through multiple manual steps. The staff must manually select each item, choose the supplier, input the required quantity, and fill in details before generating the PO. Once it is prepared, the PO still needs to be emailed to the supplier manually, as there is no integration or automation built into the current system for direct supplier communication.

After the PO is sent, staff are required to manually update the order status in the system to reflect changes like "Pending", "Shipped", or "Received". This status is not automatically tracked or triggered by supplier actions, which creates a risk of outdated or forgotten records. There are also no system-generated alerts or updates to inform the team when something is delayed, approved, or completed. This lack of timely updates may lead to confusion or duplicated work, especially when multiple team members are involved in the process.

Another issue observed is the absence of low-stock alerts or expiry tracking. If certain products are running low or nearing their expiry date, there is no system-based warning to notify the team. Staff are required to check inventory levels manually on a regular basis to ensure that nothing important runs out or expires, which is not only time-consuming but also increases the chance of human error.

Reporting is another weak point. The system does not provide a clear overview of stock performance, reorder trends, or key inventory metrics. Any report required for review must be compiled manually using raw data, which takes extra time and may not be fully accurate or up-to-date.

From the observation, it became clear that the current system lacks automation, real-time updates, and visibility in key areas of inventory operations. These findings supported the need for a more intelligent and integrated solution, one that includes threshold alerts, PO automation, expiry management, and a dashboard with exportable reports. These insights directly informed the design of the proposed system, guiding the development of features that would reduce manual workload and improve the overall efficiency of the inventory process.

3.3 Requirements Analysis

3.3.1 Project Chart

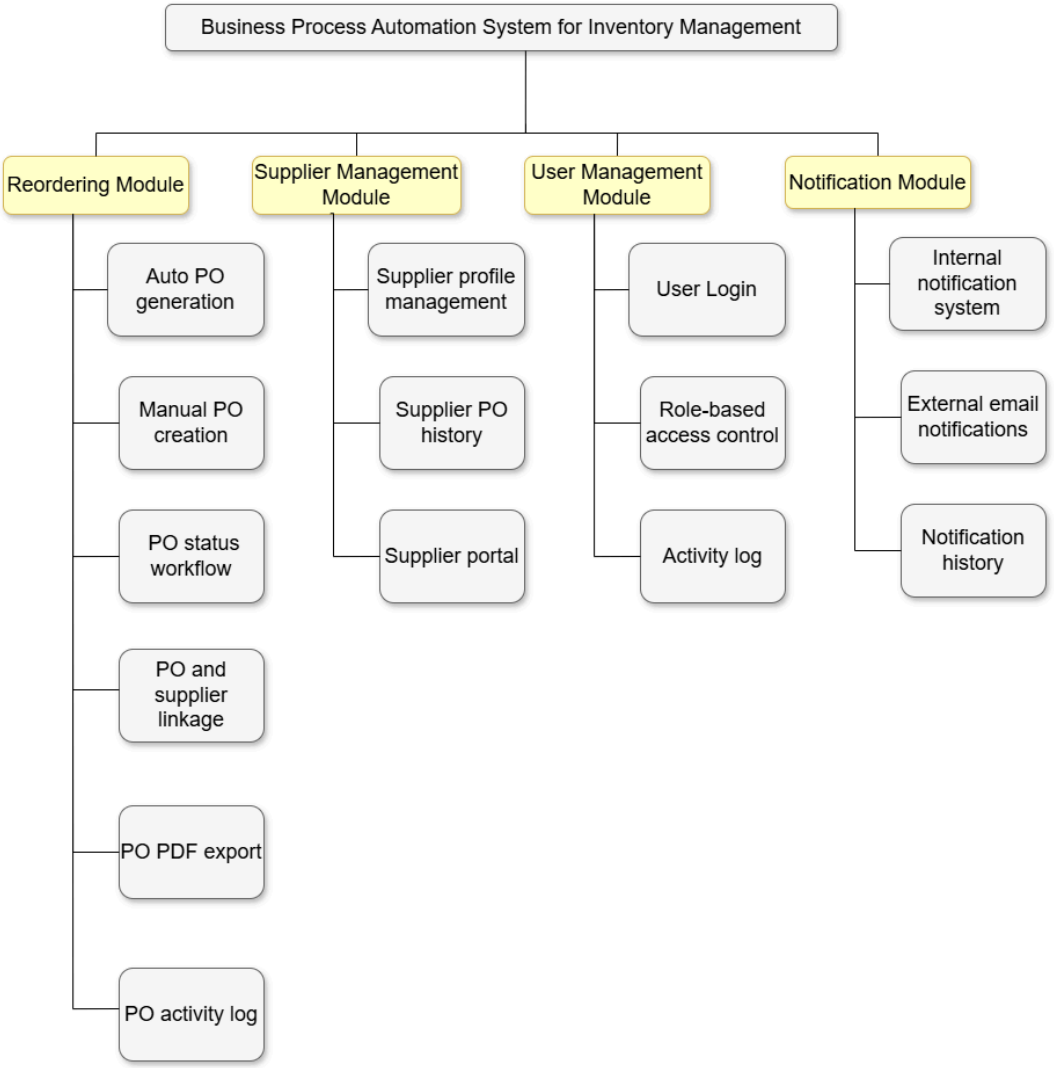


Figure 3.3: Project Chart

3.3.2 Functional Requirements

1. **Reordering Module**

   A. Auto PO Generation

   Function: Automatically generates purchase orders when stock falls below threshold, checking existing active POs to avoid duplicates.

   User Interaction: Admin and Manager receive system notifications and can review the auto-generated PO in the system.

   B. Manual PO Creation

Function: Allows authorised users (Staff/Admin/Manager) to manually create purchase orders, select items, adjust quantities, and set expected dates.
User Interaction: Users fill out a form interface to build a PO draft.

C. PO Status Workflow

Function: Controls full PO lifecycle including: Created → Pending → Approved/Rejected → Confirmed → Shipped → Received → Completed/Issue → Inventory Update.
User Interaction:

- Admin/Manager: approve, confirm, and complete POs.
- Staff: update receiving status and mark Issue/Received.
- Supplier: approve, reject, or mark Shipped.

D. PO and Supplier Linkage

Function: Automatically assigns PO and items to the correct supplier based on predefined supplier-item relationships.
User Interaction: Users view auto-linked supplier details; Admin and Manager may adjust manually before confirmation.

E. PO PDF Export

Function: Generates a structured Purchase Order PDF, including company details, supplier details, items, totals, and tax.
User Interaction: Users click "Export" to instantly download the generated PDF.

F. PO Activity Log

Function: Records all user actions related to POs, including creation, updates, approvals, and rejections.
User Interaction: Admin and Managers can view the full PO activity log for auditing purposes.

2. **Supplier Management Module**

A. Supplier Profile Management

Function: Handles creation, editing, and maintenance of supplier details including company name, contacts, address, email.
User Interaction: Admin and Manager can update supplier records; suppliers may update their own profile through the supplier portal.

B. Supplier PO History

Function: Displays all purchase orders linked to a supplier, including status, totals, and issue/received history.

User Interaction: Users select a supplier to view its complete PO transaction record.

C. Supplier Portal

Function: Provides suppliers with access to their POs, allowing them to update statuses (Approved, Rejected, Shipped) and manage contact details.

User Interaction: Supplier logs in with supplier credentials and performs required updates.

3. **User Management Module**

A. User Login

Function: Authenticates staff and supplier accounts with secure password hashing.

User Interaction: Users enter credentials and are redirected to their role-based dashboard.

B. Role-Based Access Control

Function: Enforces permissions, ensuring Admin, Manager, and Staff have different capabilities.

User Interaction: Admin can manage user roles; Staff see limited menu options.

C. Activity Log

Function: Logs all significant user actions (e.g., PO updates, supplier edits, item changes).

User Interaction:

- Admin/Manager: full Activity Log viewer
- Staff: actions are logged but cannot view logs

4. **Notification Module**

A. Internal Notification System

Function: Sends in-system notifications stored in the database (e.g., PO updates, supplier changes, low/no stock).

User Interaction: Users view notifications via the dashboard bell icon and notification page.

B. External Email Notifications

Function: Sends email notifications for PO status updates, low stock alerts, supplier profile changes, user updates, and item expiry.

User Interaction: Users receive email alerts triggered by system events.

C. Notification History

Function: Stores all internal notifications for reference, enabling users to review past alerts.

User Interaction: Users can open the notification list to review older entries.

3.3.3 Non-Functional Requirements

1. **Performance**

   The system is expected to respond quickly to user actions such as loading dashboards, submitting purchase orders, or sending stock alerts. All major functions should be completed within a few seconds under normal usage. This helps users complete their tasks without unnecessary delay, especially during peak business hours. The system uses lightweight serverless functions and optimized database queries to ensure smooth performance.

2. **Reliability**

   A reliable system is important to make sure that users can perform inventory-related tasks anytime without interruption. The system should remain available with minimal downtime. To achieve this, cloud-based services such as services backup and CloudWatch are used to monitor the system and recover data in case of any technical issues. This ensures business operations are not affected by system failures.

3. **Security**

   The system handles important business data including supplier details, purchase records, and user access logs. To keep this data safe, the system applies user role control and secure login. All sensitive information is stored in encrypted databases, and access is only given to authorized users. This protects the system from unauthorized access and data leaks.

4. **Usability**

   The interface is designed to be simple and easy to use by different levels of staff. Each function, such as creating a PO or viewing supplier info, is presented clearly with proper labels and buttons. Users do not need advanced technical skills to use the system. Clear instructions and user feedback are provided to avoid confusion and errors.

5. **Scalability**

   The system should be able to handle more users, products, and suppliers as the company grows. It is built using a flexible cloud structure that can expand when needed. This means the system will continue to run smoothly even if more features or users are added in the future, without needing to rebuild the whole system.

3.4 Development Environment

3.4.1 Hardware

The system requires minimal hardware investment. The development team uses personal laptops equipped with standard specifications such as Intel Core i5 processors and 8GB RAM, which are sufficient for programming, testing, and deploying to AWS. Since the system is hosted entirely on the cloud, there is no need for physical servers.

End users, such as managers and staff, can access the system using web browsers on desktops, laptops, or tablets. A stable internet connection is necessary for accessing cloud-based features. No software installation is required on the client side, as the system runs entirely on a secure web interface.

3.4.2 Software

The system development relies heavily on Amazon Web Services (AWS) for its backend, data handling, and cloud deployment. Table 3.1 summarizes the AWS services used and their specific roles in the system.

Table 3.1: AWS Cloud Services and Their Usage

AWS Service	Purpose in the System
AWS Lambda	Executes backend logic for automation tasks such as auto PO generation and alert handling.
AWS RDS	Hosts the MySQL database that stores inventory records, supplier data, and user accounts.
AWS S3	Stores documents such as PO files, supplier attachments, and exported reports.
AWS SNS	Sends real-time notifications for events like low stock, PO approval, and item expiry.
AWS CloudWatch	Monitors system health, logs function activity, and triggers alarms during failure.
AWS EC2	Hosts the web application and API endpoints, allowing users to access the inventory system through a stable and scalable server environment.

In addition to AWS services, the backend was developed using PHP, while MySQL served as the relational database. The frontend interface was built with HTML, CSS, and JavaScript, ensuring accessibility and responsiveness across various browsers and devices. Development work was supported by open-source or student-licensed tools such as Visual Studio Code for coding, Draw.io for system diagrams, and Microsoft Word for documentation. Together, these

tools provided a flexible and cloud-ready environment that aligned with the Waterfall model used for the project.

3.4.3 Human Resource Allocation

The project is handled by two developers. Lim Yean Yee focuses on the stock monitoring, dashboard, and reporting modules, while Lee Siaw Jing handles supplier management, purchase order automation, and notification integration. Both work together on testing, cloud configuration, and documentation to ensure smooth development and integration.

3.4.4 Integration Readiness

All selected technologies are open-source, cloud-based, and compatible with one another. The system uses APIs for seamless communication between modules and AWS services. This architecture supports future expansion without requiring major redesigns. Based on available resources and team proficiency, the system is considered technically feasible and ready for implementation.

3.4.5 Estimated Project Cost

The development environment includes the essential tools, equipment, and cloud services used to build and test the system. The cost breakdown is categorized into four components: development effort, hardware, software, and cloud infrastructure.

The development effort covers all phases: planning, design, coding, and testing, conducted by two student developers. With a reference rate of RM1,453 per person per month, the total cost is:

- RM1,453 × 2 × (1 + 1 + 3 + 1) months = RM17,436

The hardware cost includes two development laptops (Intel i5, 8GB RAM, 512GB SSD) and a high-speed internet plan. The internet is used continuously throughout the 6-month project period.

- Laptops: RM2,500 × 2 = RM5,000
- Internet: RM199 × 6 = RM1,194
- Total Hardware Cost = RM6,194

The software tools used are mostly open-source or student-licensed, except for the Windows OS:

- Windows OS: RM499 × 2 = RM998
- Other tools (VS Code, MySQL, Draw.io, Word) = RM0
- Total Software Cost = RM998

The system runs on a cloud-based infrastructure using AWS services, estimated over six months of development/testing:

- Lambda (RM150), RDS (RM1,140), S3 (RM180), SNS (RM60), CloudWatch (RM330), EC2 (RM360)
- Total Cloud Infrastructure Cost = RM2,220

Below is the updated cost summary in Table 3.2:

Table 3.2: Total Estimated Project Cost Summary

Category	**Total Cost (RM)**
Development Cost	RM17,436
Hardware Cost	RM6,194
Software Cost	RM998
Cloud Infrastructure	RM2,220
Grand Total	**RM26,848**

3.5 Requirements Analysis of Entire System (Flowchart)

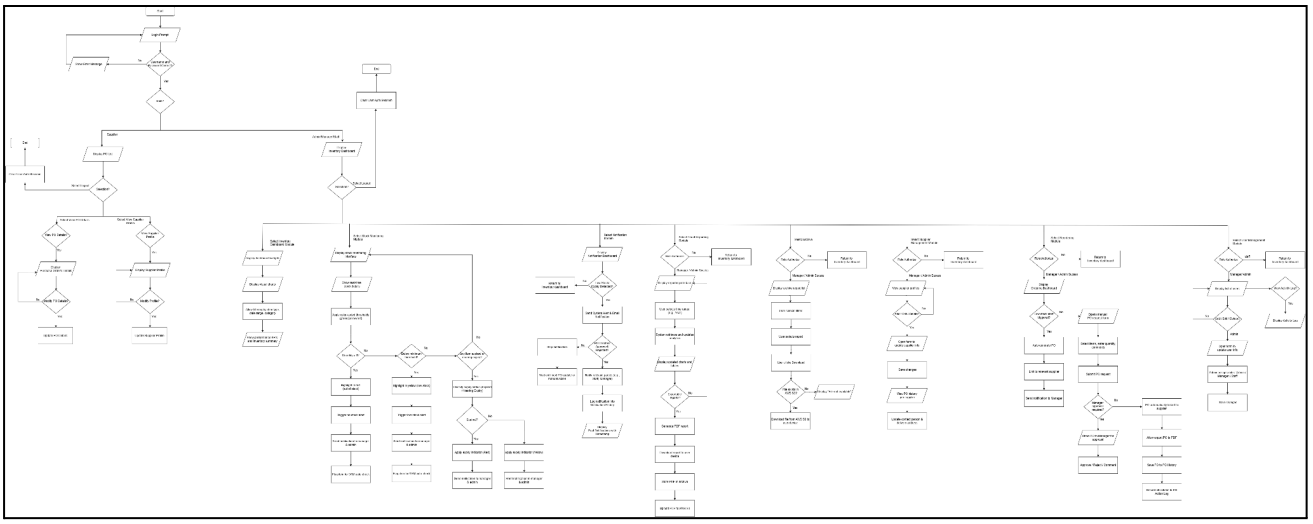


Figure 3.4: Flowchart

3.6 Chapter Summary and Evaluation

This chapter outlined the methodology and requirements analysis for the Business Process Automation System for Inventory Management. The Waterfall Model was chosen as the development approach due to its structured flow and suitability for clearly defined requirements. Each development phase was explained to show how it guided the system from planning to deployment.

Requirement gathering techniques such as interviews and observations were conducted to identify user needs and system weaknesses. These findings helped shape the core modules including stock monitoring, PO automation, reporting, supplier management, and notification features.

Functional and non-functional requirements were clearly defined, supported by descriptions of the development environment, tools, team roles, and estimated project cost. Overall, this chapter laid a strong foundation for the system's development, ensuring it is well-planned, technically feasible, and ready for implementation.

Chapter 4

System Design

4 System Design

This chapter documents the system design as it was actually implemented. It opens with the database perspective through the Entity Relationship Diagram and the corresponding class diagram, then presents a complete data dictionary for the core tables used by the application, namely User, Supplier, Item, PurchaseOrder, PurchaseOrderDetails, GoodsReceipt, GoodsReceiptDetails, StockAlert, Notification, ExportHistory, and ActivityLog. The design of behaviour is then conveyed through an overall use case diagram and detailed use case diagrams for four modules: Reordering, Supplier Management, User Management, and Notification. To make the runtime logic explicit, the activity diagrams are provided for each key function and are followed by sequence diagrams that mirror the final method names and flows used in the code. A layered software architecture diagram summarises how the presentation, control, and data/domain layers are separated, and the chapter closes with the user interface designs for the main pages that were built and tested.

4.1 Database Design/Entity Relationship Diagram (ERD)

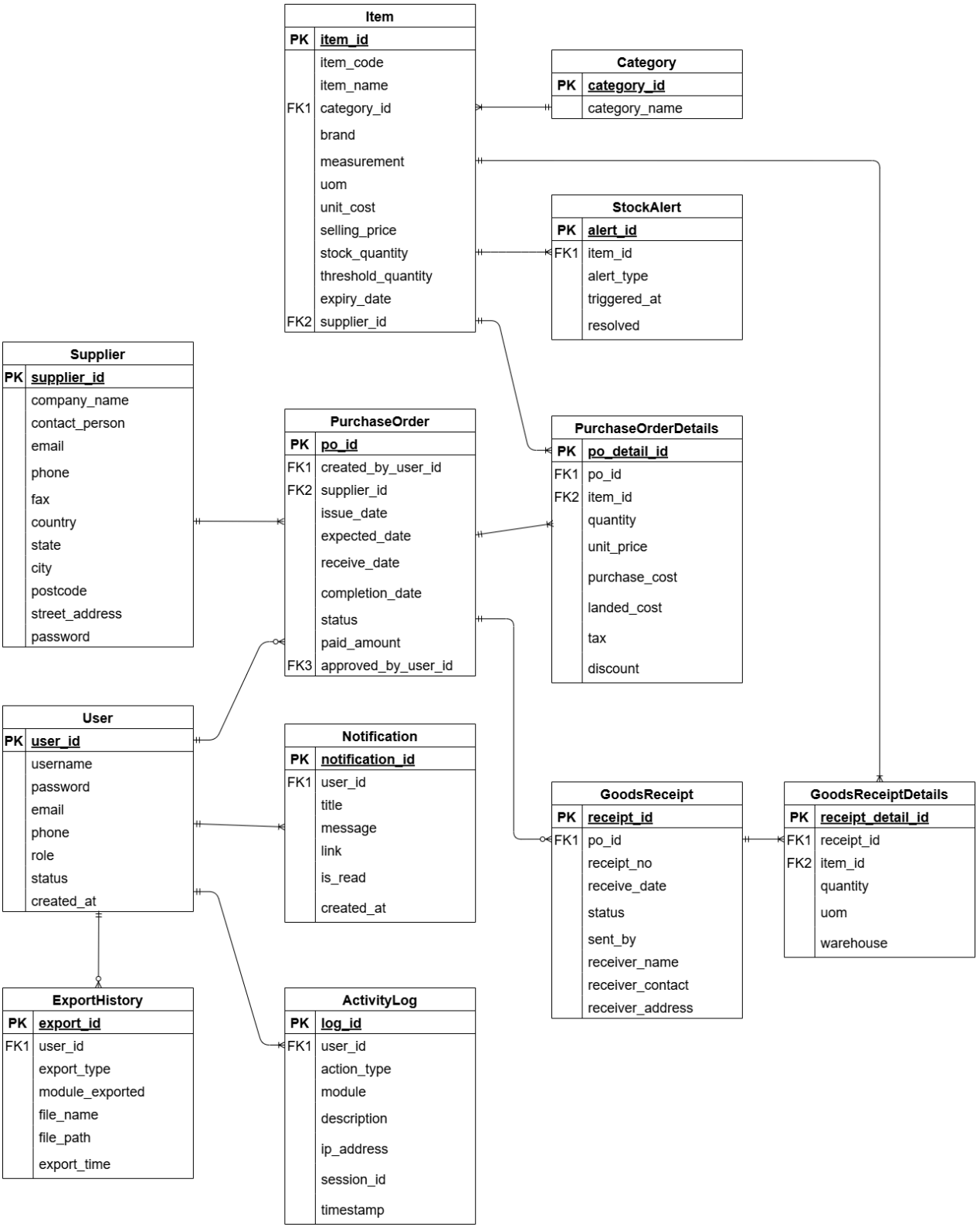


Figure 4.1: Entity Relationship Diagram (ERD)

4.2 Class Diagram

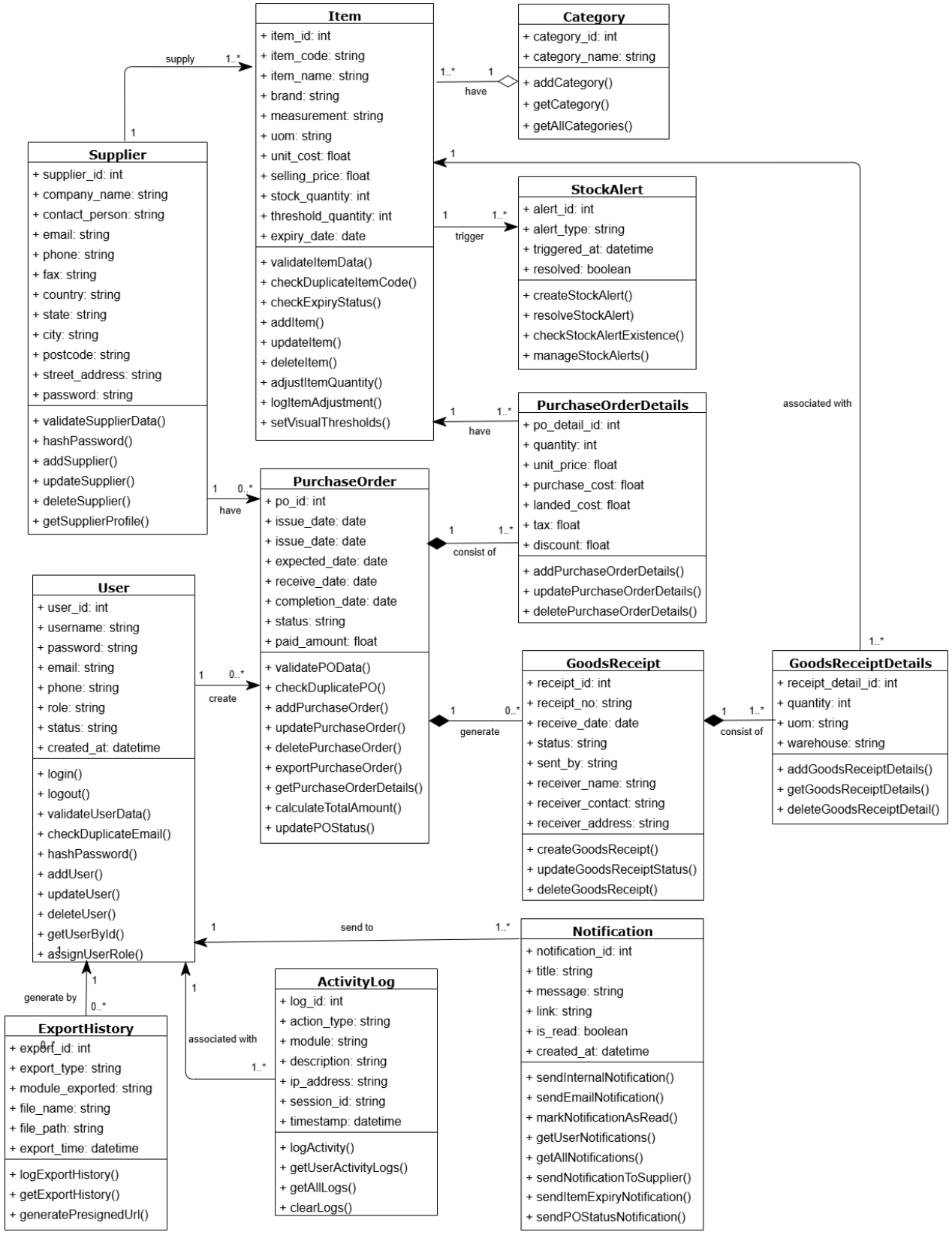


Figure 4.2: Class Diagram

4.3 Data Dictionary

4.3.1 User Table

Table 4.1: Data Dictionary for User Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	user_id	INT	–	Unique ID for each user
	username	VARCHAR	50	User login name
	password	VARCHAR	255	Hashed password
	email	VARCHAR	100	User email address
	phone	VARCHAR	20	User phone number
	role	VARCHAR	20	Role in system (Admin, Manager, Staff)
	status	VARCHAR	10	Account status (Active/Inactive)
	created_at	DATETIME	–	Date and time account was created

4.3.2 Supplier Table

Table 4.2: Data Dictionary for Supplier Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	supplier_id	INT	–	Unique ID for each supplier
	company_name	VARCHAR	100	Supplier company name
	contact_person	VARCHAR	100	Person-in-charge at supplier
	email	VARCHAR	100	Supplier's email
	phone	VARCHAR	20	Supplier phone number
	fax	VARCHAR	20	Fax number
	country	VARCHAR	50	Country of supplier
	state	VARCHAR	50	State in which supplier is located
	city	VARCHAR	50	City of supplier
	postcode	VARCHAR	10	Postal code
	street_address	VARCHAR	150	Supplier's street address

	password	VARCHAR	255	Hashed password for supplier portal login

4.3.3 Category Table

Table 4.3: Data Dictionary for Category Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	category_id	INT	–	Unique ID for category
	category_name	VARCHAR	50	Name (e.g., Food, Beverage)

4.3.4 Item Table

Table 4.4: Data Dictionary for Item Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	item_id	INT	–	Unique ID for each item
	item_code	VARCHAR	20	System code for the item
	item_name	VARCHAR	100	Name of the item
FK	category_id	INT	–	Links to Category table
	brand	VARCHAR	50	Item brand
	measurement	VARCHAR	50	Size or measurement description
	uom	VARCHAR	50	Unit of measurement
	unit_cost	DECIMAL	10,2	Unit cost price
	selling_price	DECIMAL	10,2	Retail selling price
	stock_quantity	INT	–	Quantity available in stock
	threshold_quantity	INT	–	Reorder threshold level
	expiry_date	DATE	–	Expiry date
FK	supplier_id	INT	–	Links to Supplier table

4.3.5 PurchaseOrder Table

Table 4.5: Data Dictionary for PurchaseOrder Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	po_id	INT	–	Unique ID for each purchase order
FK	created_by_user_id	INT	–	User who created the purchase order
FK	supplier_id	INT	–	Supplier associated with the order
	issue_date	DATE	–	Date the purchase order was issued
	expected_date	DATE	–	Target date for delivery
	receive_date	DATE	–	Actual date goods were received
	completion_date	DATE	–	Date PO flow was finalized
	status	VARCHAR	20	Status (Created, Approved, Completed, etc.)
	paid_amount	DECIMAL	10,2	Amount paid
FK	approved_by_user_id	INT	–	User who approved the order

4.3.6 PurchaseOrderDetails Table

Table 4.6: Data Dictionary for PurchaseOrderDetails Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	po_detail_id	INT	–	Unique ID for each order line item
FK	po_id	INT	–	Associated purchase order
FK	item_id	INT	–	Item being ordered
	quantity	INT	–	Number of units ordered
	unit_price	DECIMAL	10,2	Price per unit
	purchase_cost	DECIMAL	10,2	Total cost per line item
	landed_cost	DECIMAL	10,2	Final cost including all charges
	tax	DECIMAL	5,2	Tax percentage
	discount	DECIMAL	5,2	Discount applied

4.3.7 GoodsReceipt Table

Table 4.7: Data Dictionary for GoodsReceipt Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	receipt_id	INT	–	Unique ID for the goods receipt
FK	po_id	INT	–	Related purchase order
	receipt_no	VARCHAR	50	Official receipt number
	receive_date	DATE	–	Date the goods were received
	status	VARCHAR	20	Receipt status (Created, Confirmed, etc.)
	sent_by	VARCHAR	100	Person or company that sent the goods
	receiver_name	VARCHAR	100	Name of person receiving the goods
	receiver_contact	VARCHAR	20	Contact of receiver
	receiver_address	VARCHAR	150	Delivery address

4.3.8 GoodsReceiptDetails Table

Table 4.8: Data Dictionary for GoodsReceiptDetails Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	receipt_detail_id	INT	–	Unique ID for each goods received line
FK	receipt_id	INT	–	Associated goods receipt
FK	item_id	INT	–	Item received
	quantity	INT	–	Quantity received
	uom	VARCHAR	10	Unit of measurement
	warehouse	VARCHAR	50	Receiving warehouse

4.3.9 StockAlert Table

Table 4.9: Data Dictionary for StockAlert Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	alert_id	INT	–	Unique ID for stock alert
FK	item_id	INT	–	Item related to the stock alert
	alert_type	VARCHAR	50	Type of alert (e.g., Low Stock, Expired)
	triggered_at	DATETIME	–	Time the alert was triggered
	resolved	BOOLEAN	–	Whether the alert has been resolved

4.3.10 Notification Table

Table 4.10: Data Dictionary for Notification Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	notification_id	INT	–	Unique ID for notification
FK	user_id	INT	–	User receiving the notification
	title	VARCHAR	255	Short header of the notification
	message	TEXT	–	Full notification content
	link	VARCHAR	255	URL to related page (e.g., PO View)
	is_read	TINYINT	–	0 = Unread, 1 = Read
	created_at	TIMESTAMP	–	Time notification was sent

4.3.11 ExportHistory Table

Table 4.11: Data Dictionary for ExportHistory Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	export_id	INT	–	Unique ID for export history
FK	user_id	INT	–	User who performed the export
	export_type	VARCHAR	50	Type of data exported
	module_exported	VARCHAR	50	Module from which data was exported
	file_name	VARCHAR	100	Name of the exported file
	file_path	TEXT	–	File location path
	export_time	DATETIME	–	Date and time of export

4.3.12 ActivityLog Table

Table 4.12: Data Dictionary for ActivityLog Table

P/F	Attribute	Data Type	Field Size	Explanation
PK	log_id	INT	–	Unique ID for activity log
FK	user_id	INT	–	User who performed the action
	action_type	VARCHAR	50	Action performed (e.g., Add, Delete)
	module	VARCHAR	50	Module name where the action occurred
	description	TEXT	–	Description of the action
	ip_address	VARCHAR	45	User's IP address
	session_id	VARCHAR	128	PHP Session ID for tracking
	timestamp	DATETIME	–	Date and time of the activity

4.4 Use Case Diagram

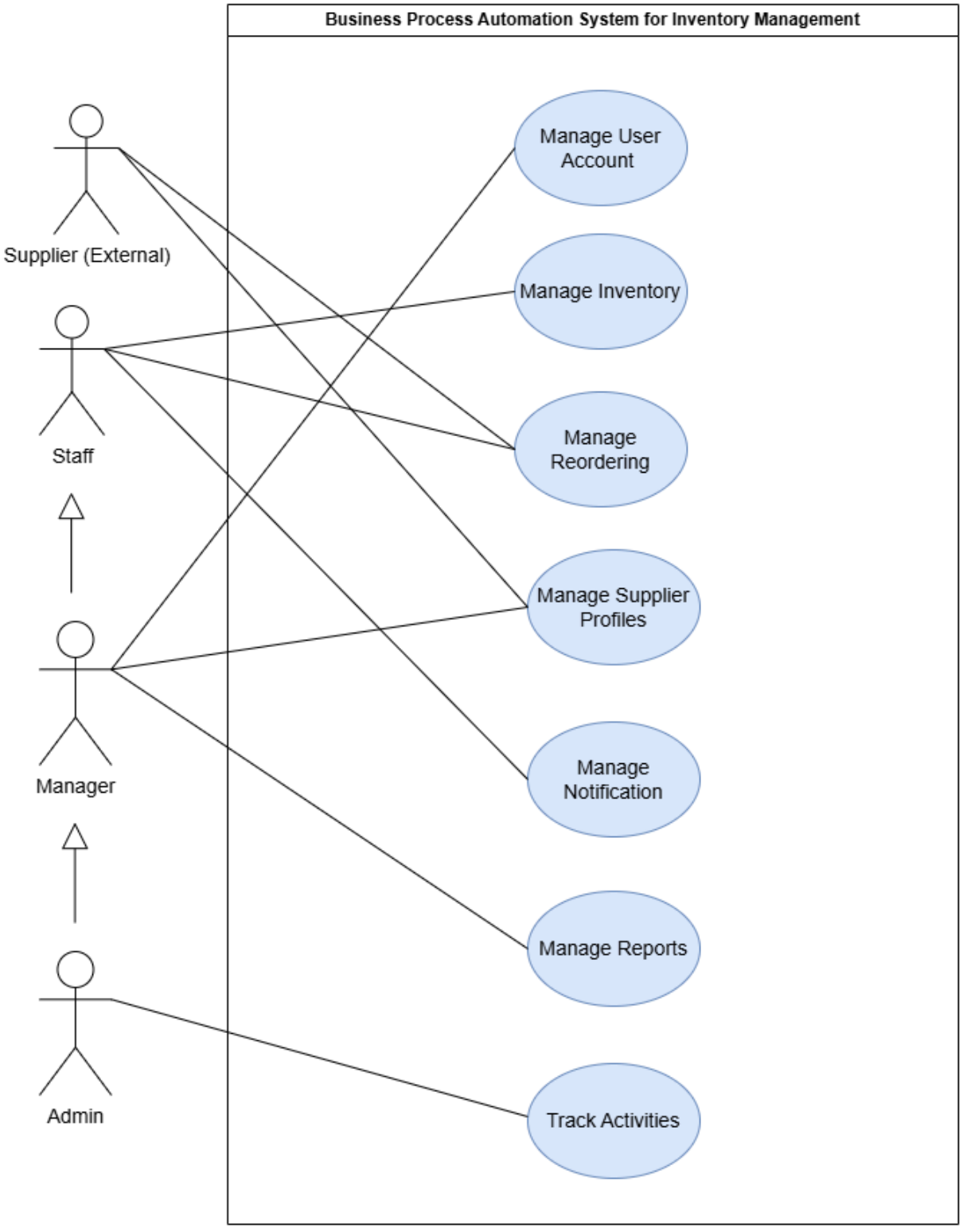


Figure 4.3: Overall Use Case Diagram

4.5 Detail Use Case Diagram

4.5.1 Reordering Module

Figure 4.4: Detail Use Case Diagram: Reordering Module

4.5.2 Supplier Management Module

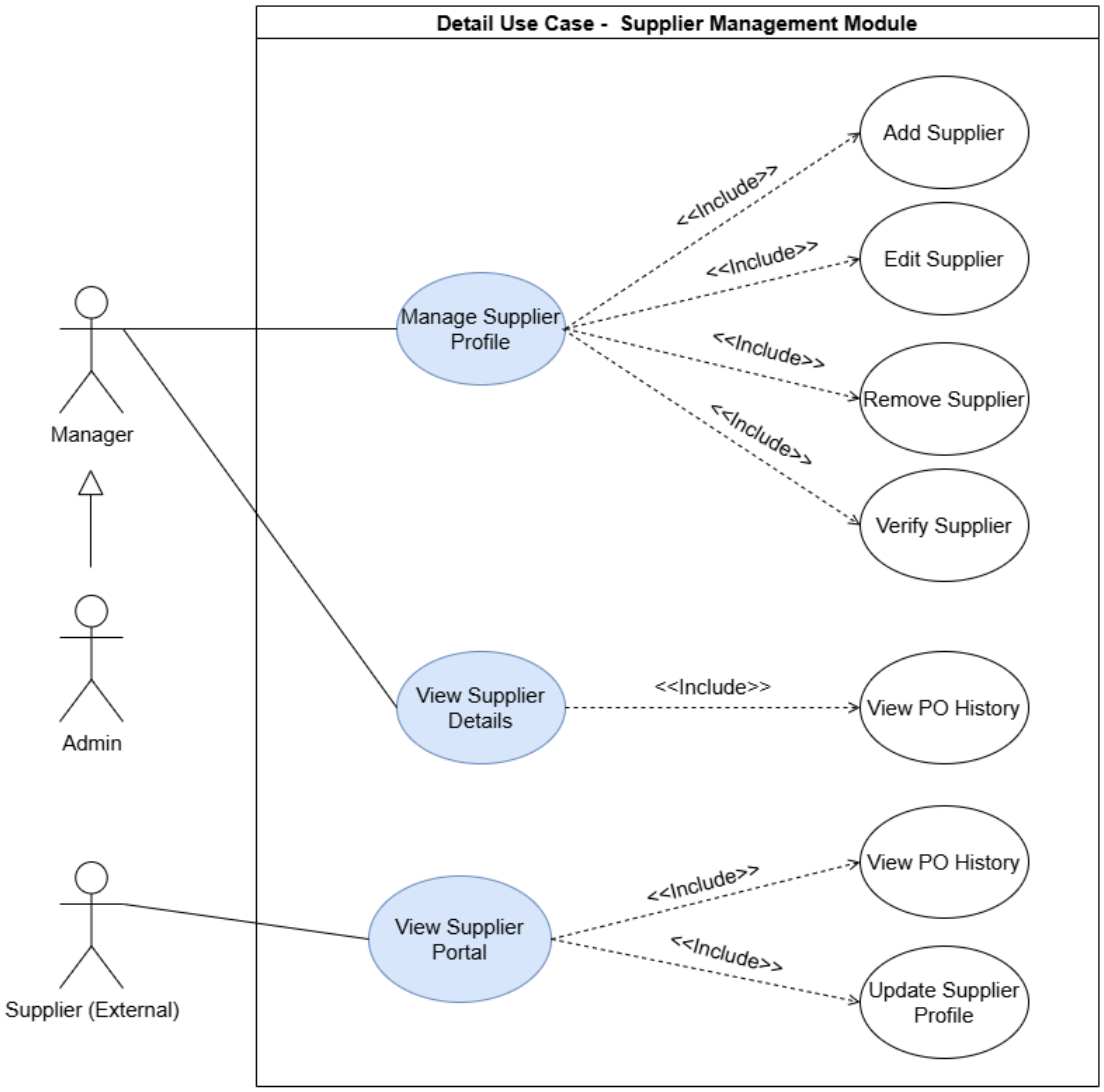


Figure 4.5: Detail Use Case Diagram: Supplier Management Module

4.5.3 User Management Module

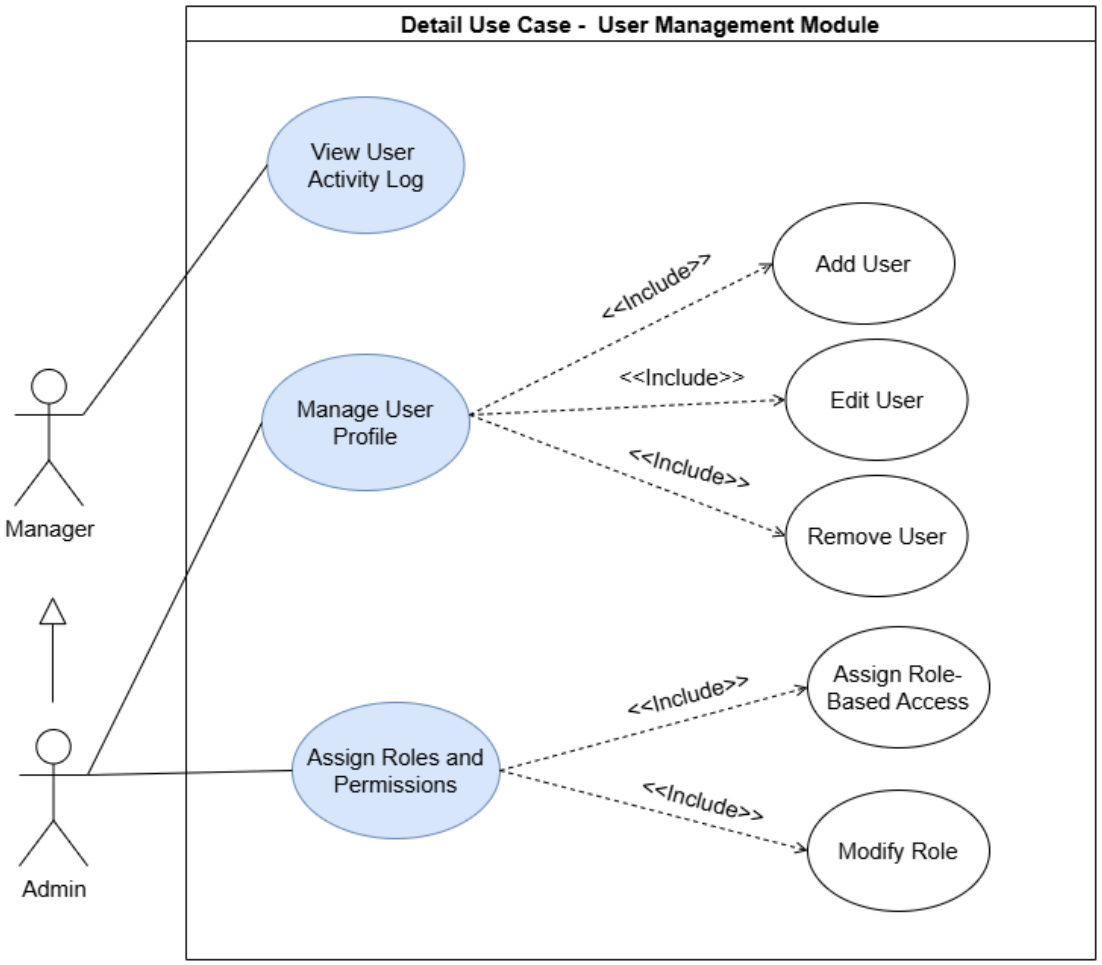


Figure 4.6: Detail Use Case Diagram: User Management Module

4.5.4 Notification Module

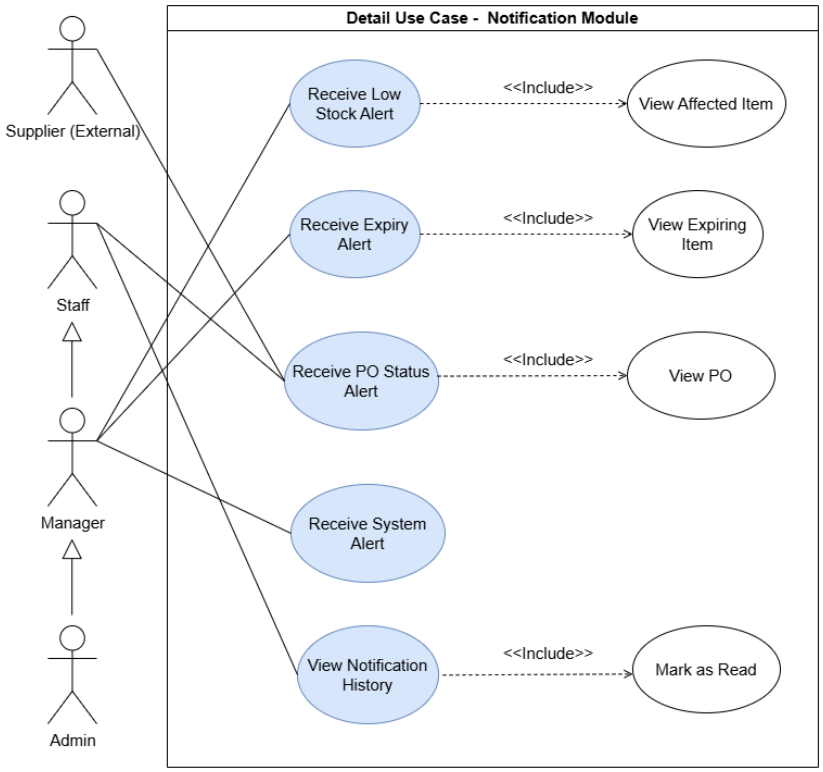


Figure 4.7: Detail Use Case Diagram: Notification Module

4.6 Activity Diagram

4.6.1 Activity Diagram for Create Manual Purchase Order

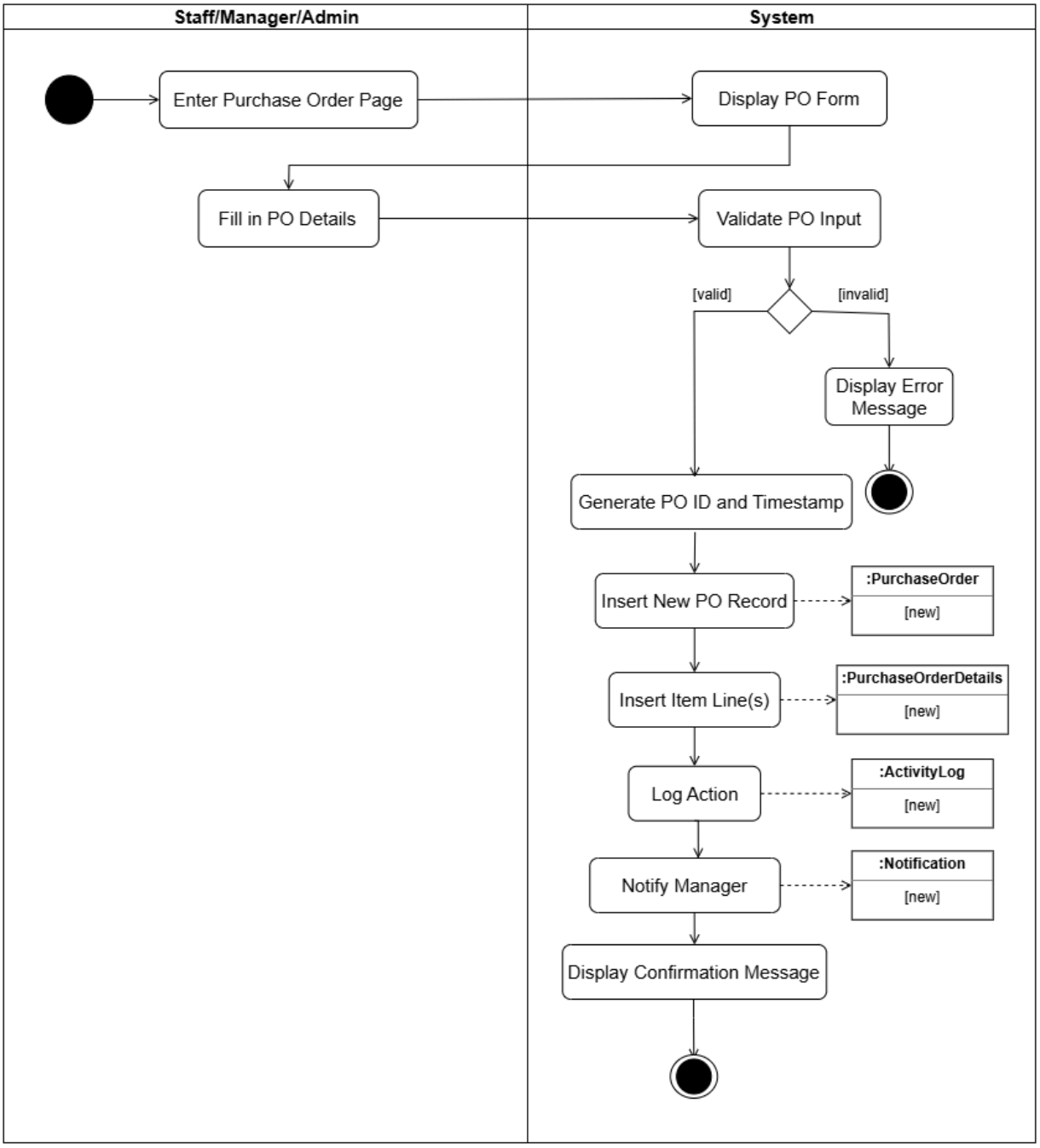


Figure 4.8: Activity Diagram for Create Manual Purchase Order

4.6.2 Activity Diagram for Auto Generate Purchase Order

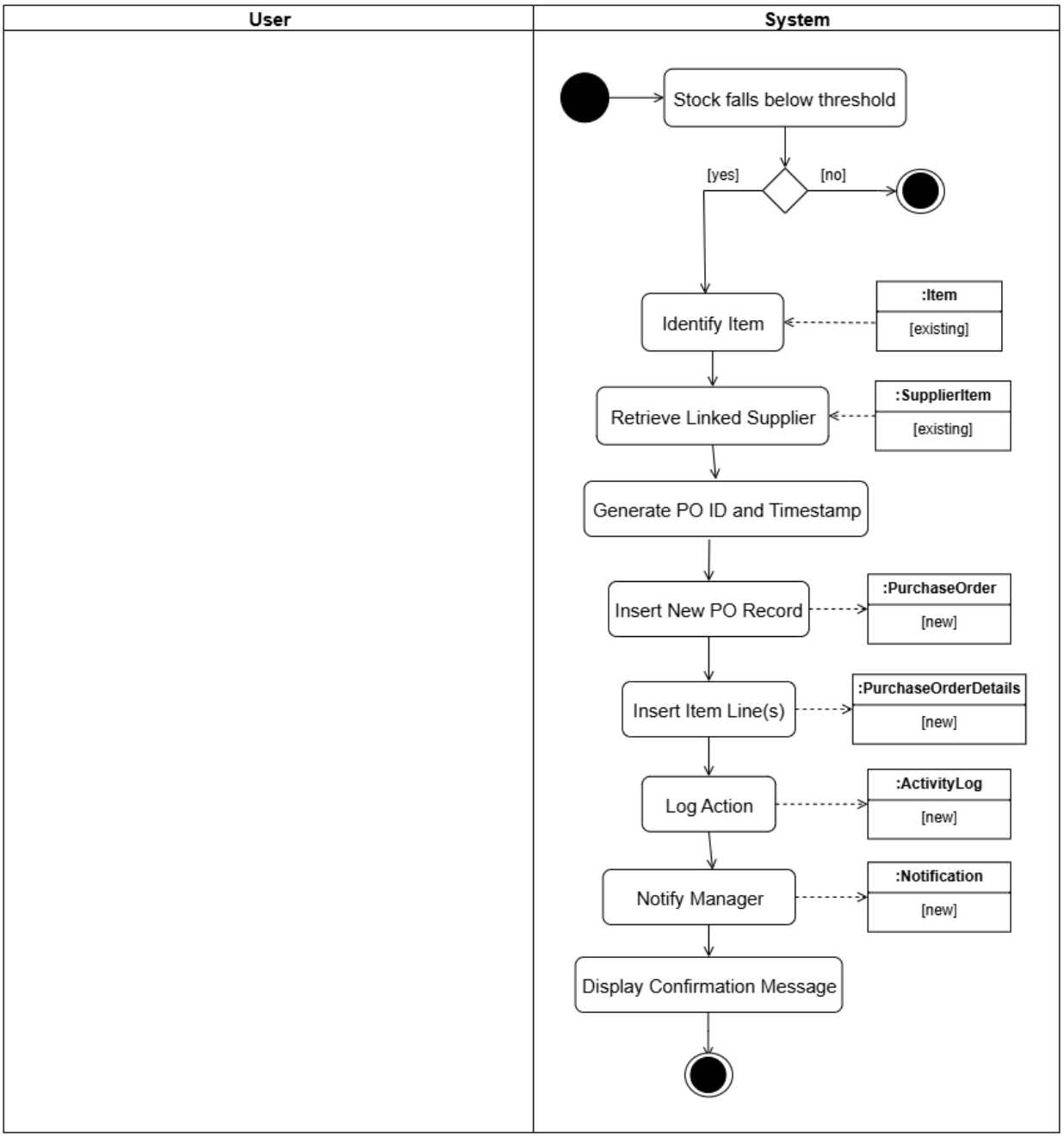


Figure 4.9: Activity Diagram for Auto Generate Purchase Order

4.6.3 Activity Diagram for Approve or Reject Purchase Order

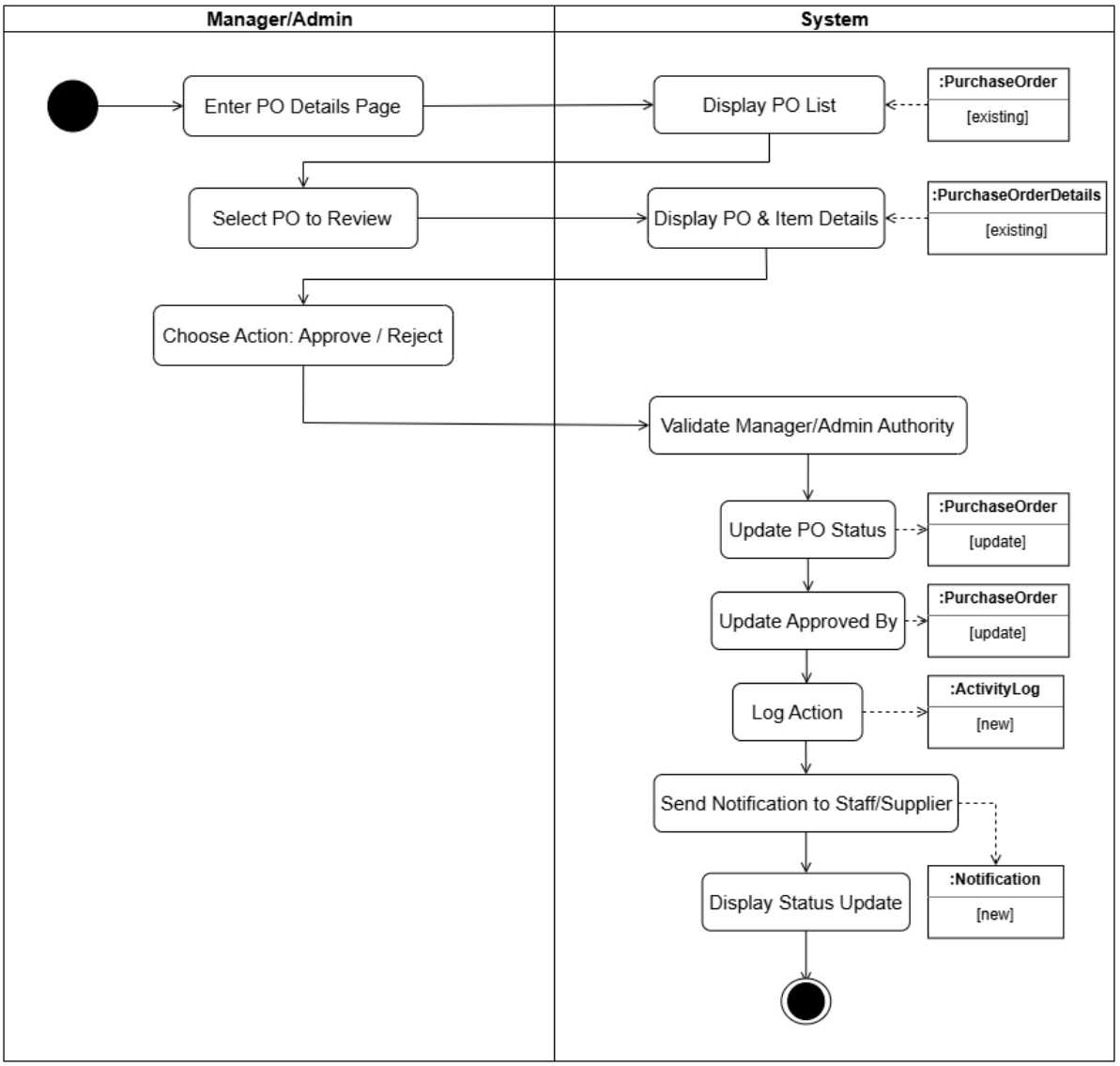


Figure 4.10: Activity Diagram for Approve or Reject Purchase Order

4.6.4 Activity Diagram for View PO History

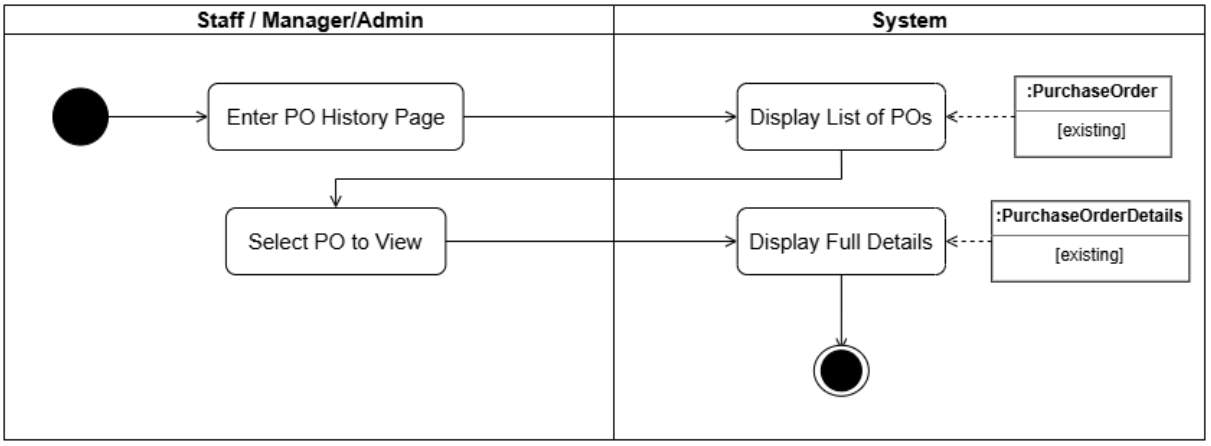


Figure 4.11: Activity Diagram for View PO History

4.6.5 Activity Diagram for Confirm Goods Receipt

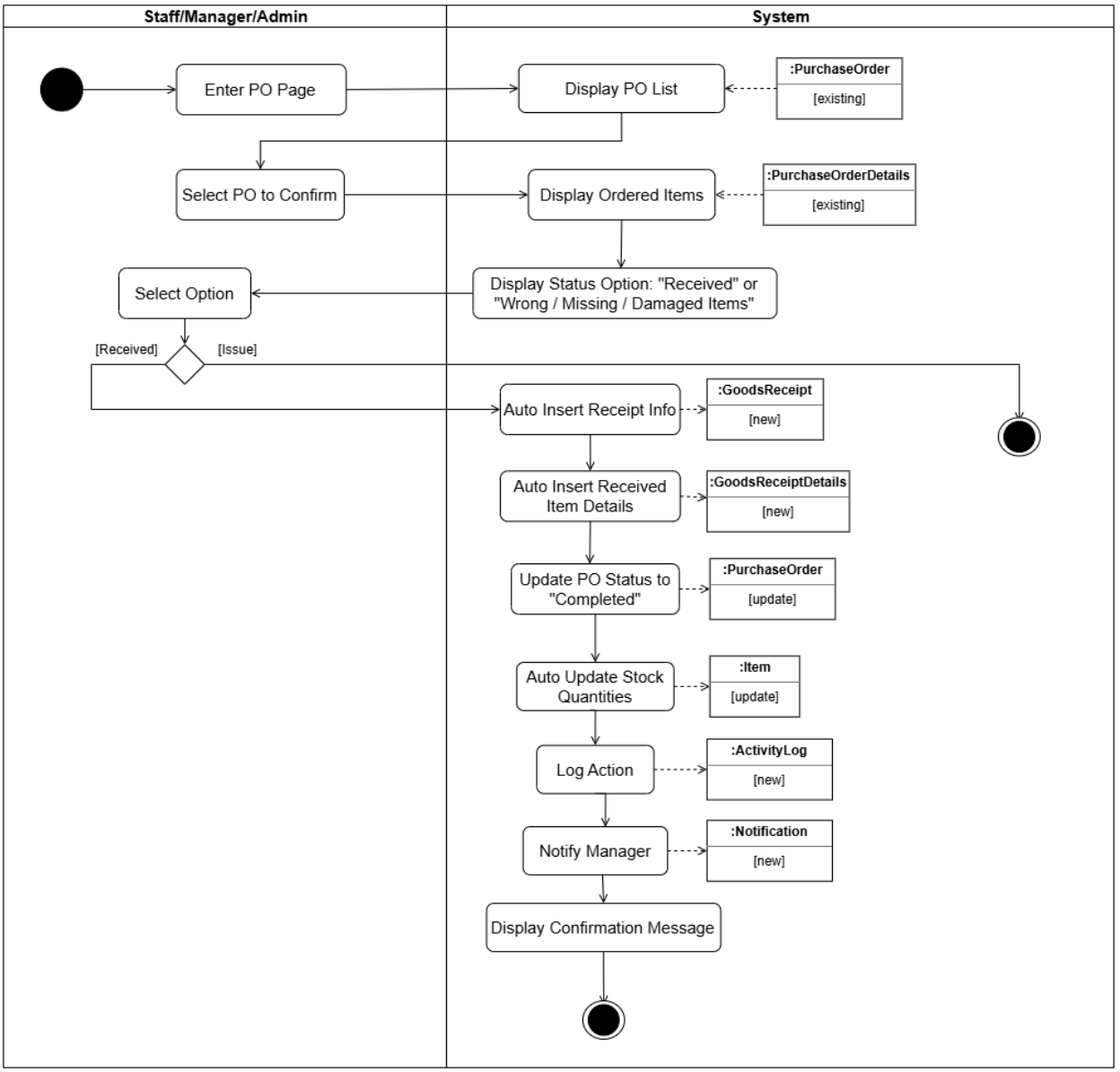


Figure 4.12: Activity Diagram for Confirm Goods Receipt

4.6.6 Activity Diagram for Add Supplier

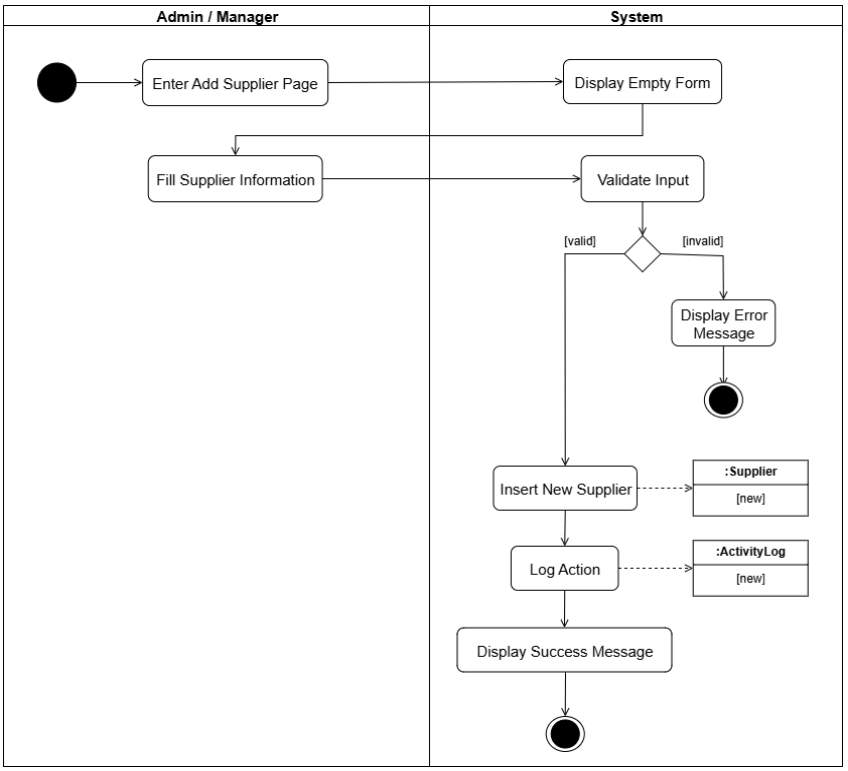


Figure 4.13: Activity Diagram for Add Supplier

4.6.7 Activity Diagram for Edit Supplier

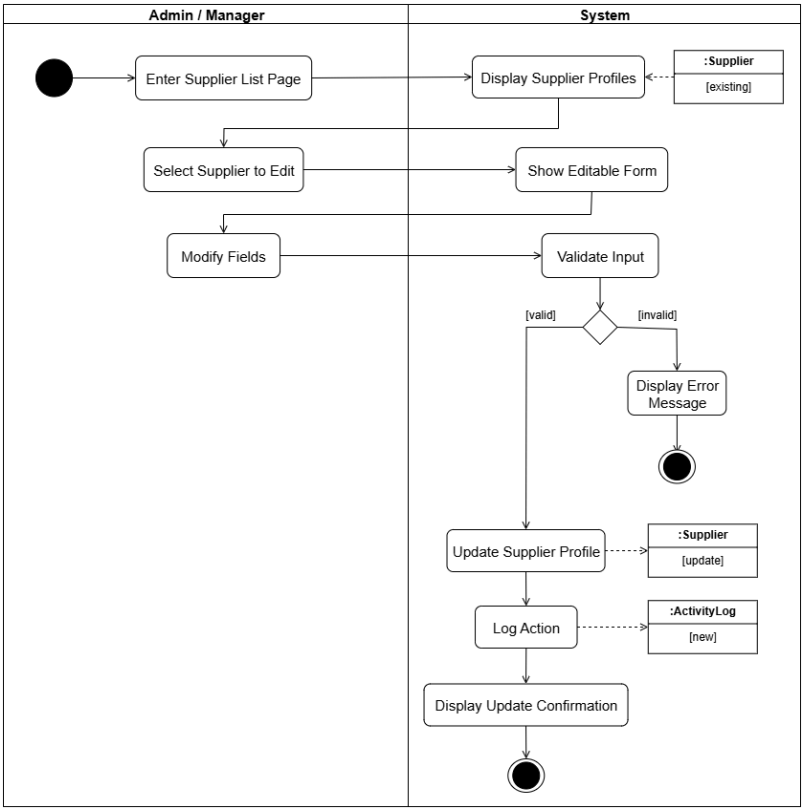


Figure 4.14: Activity Diagram for Edit Supplier

4.6.8 Activity Diagram for Remove Supplier

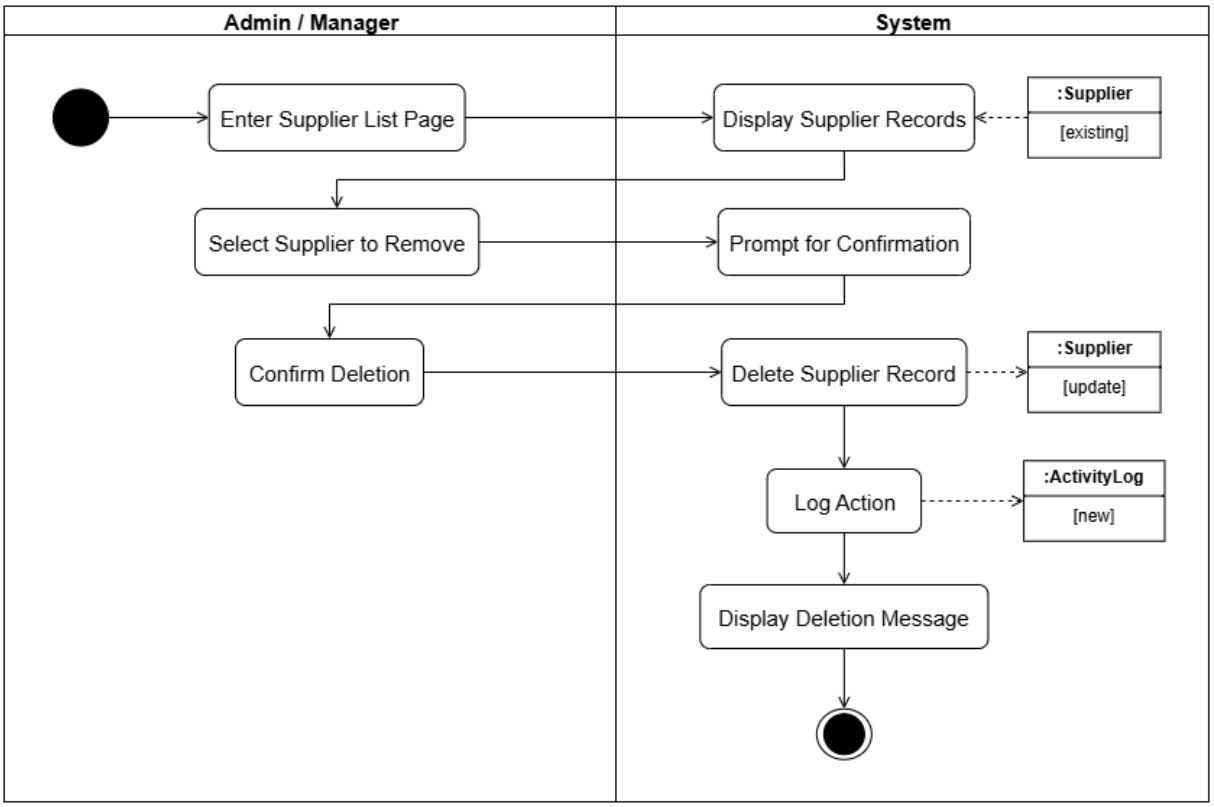


Figure 4.15: Activity Diagram for Remove Supplier

4.6.9 Activity Diagram for View Supplier

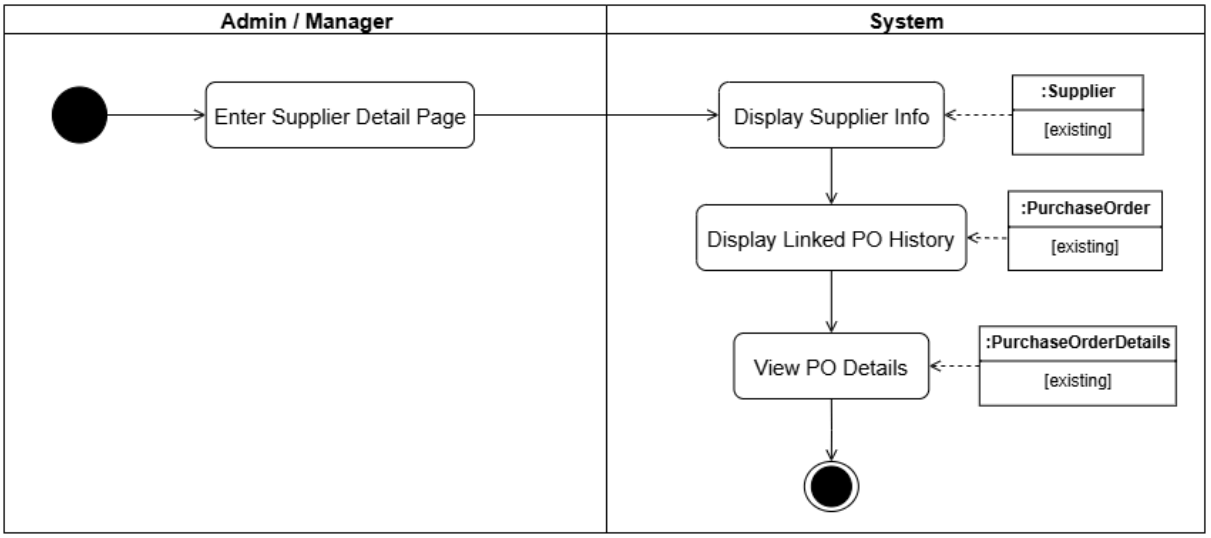


Figure 4.16: Activity Diagram for View Supplier

4.6.10 Activity Diagram for Assign/Modify Roles and Permissions

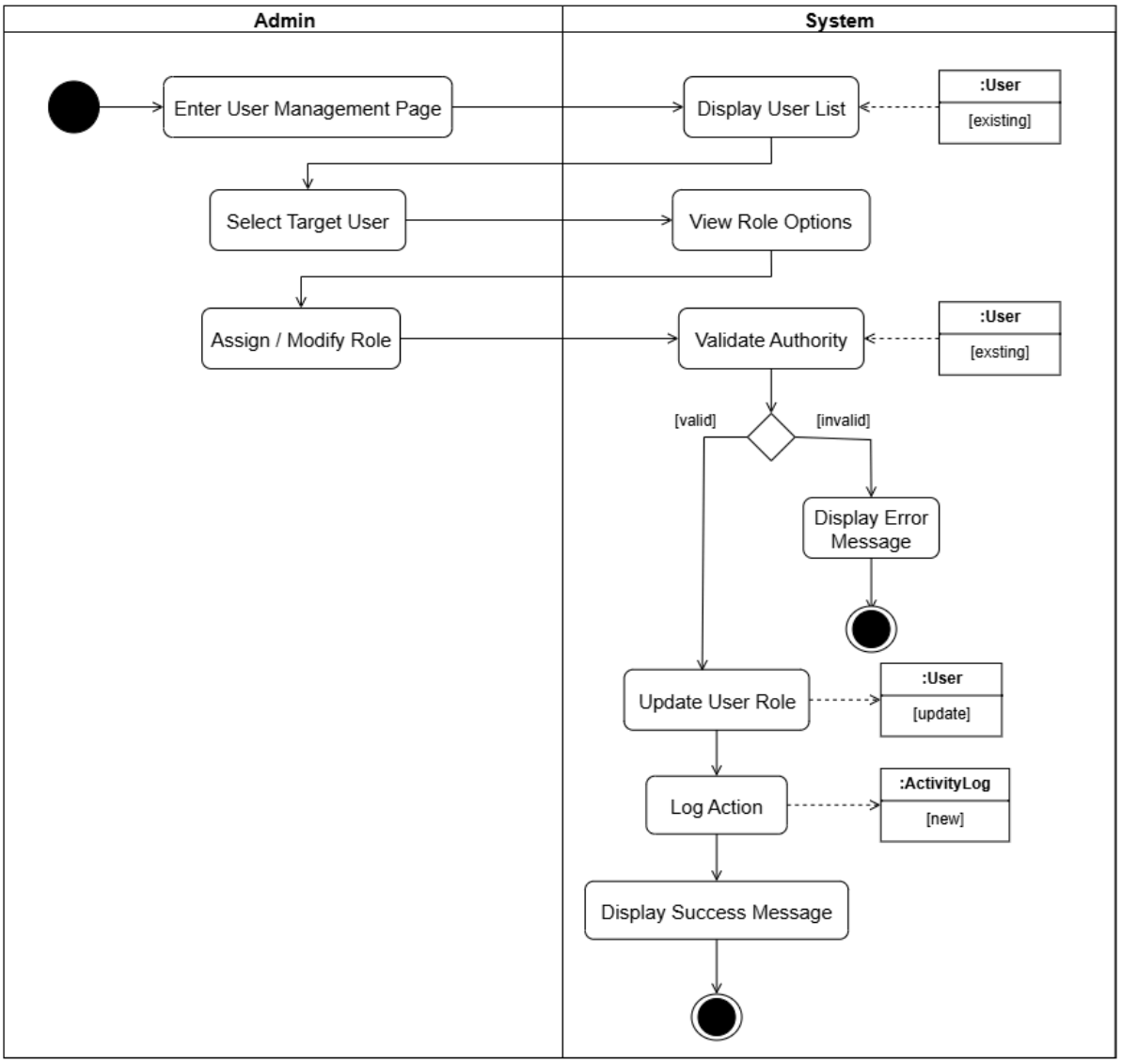


Figure 4.17: Activity Diagram for Assign/Modify Roles and Permissions

4.6.11 Activity Diagram for View User Activity Log

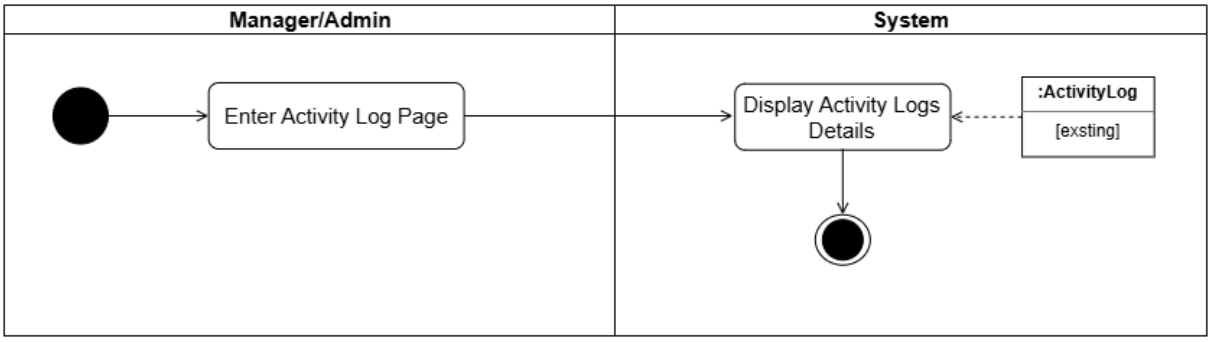


Figure 4.18: Activity Diagram for View User Activity Log

4.6.12 Activity Diagram for Add User

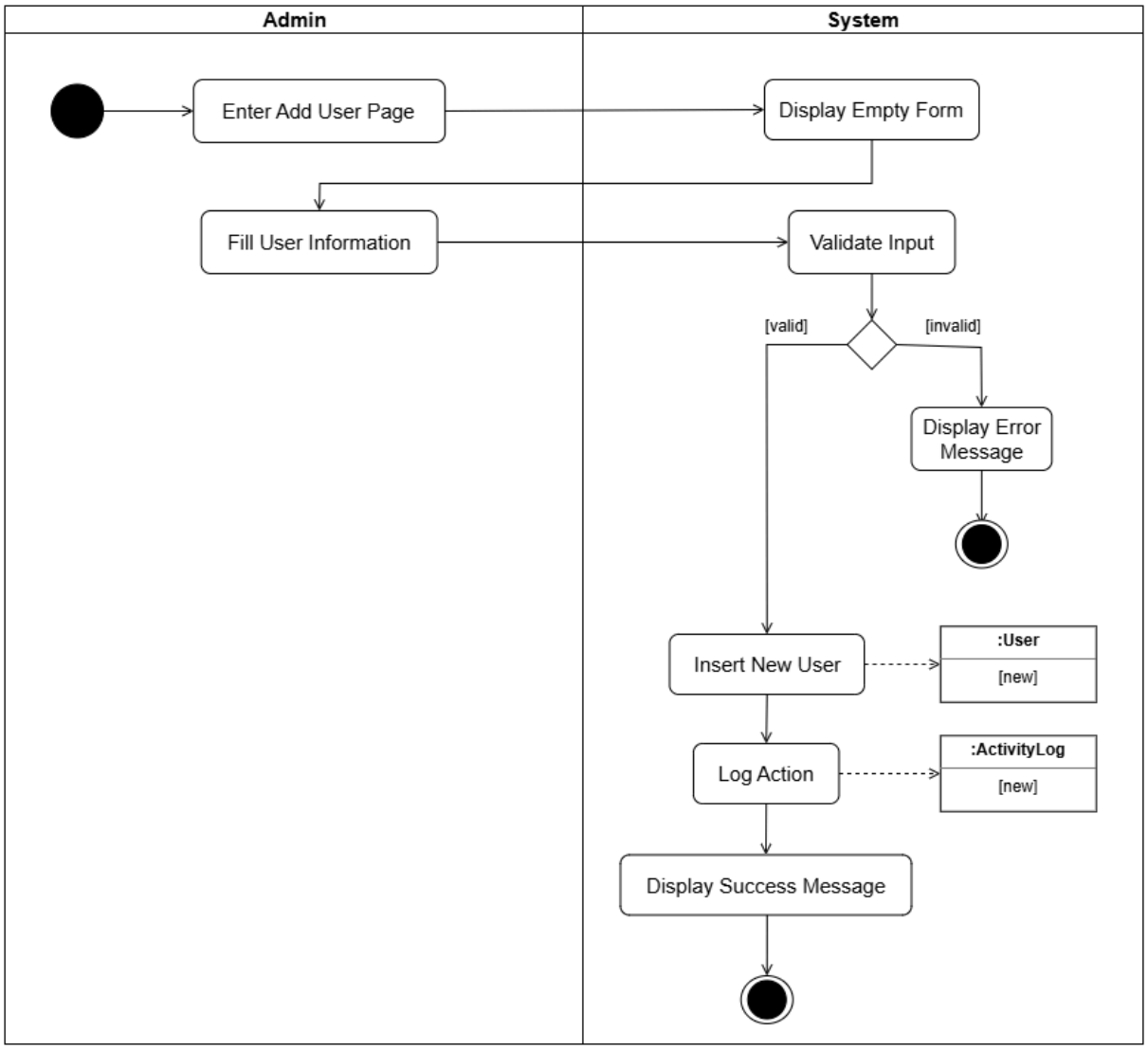


Figure 4.19: Activity Diagram for Add User

4.6.13 Activity Diagram for Edit User

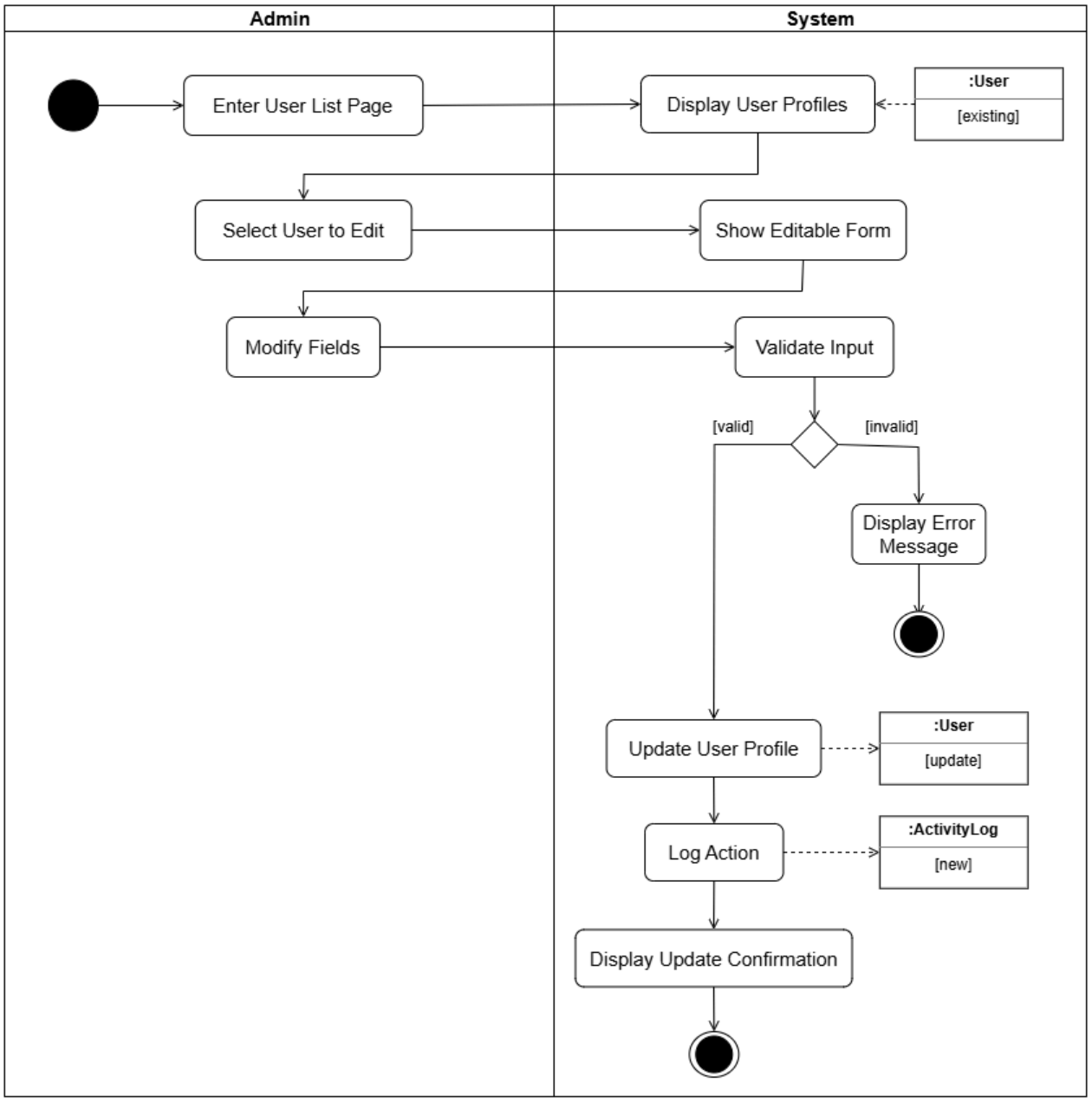


Figure 4.20: Activity Diagram for Edit User

4.6.14 Activity Diagram for Remove User

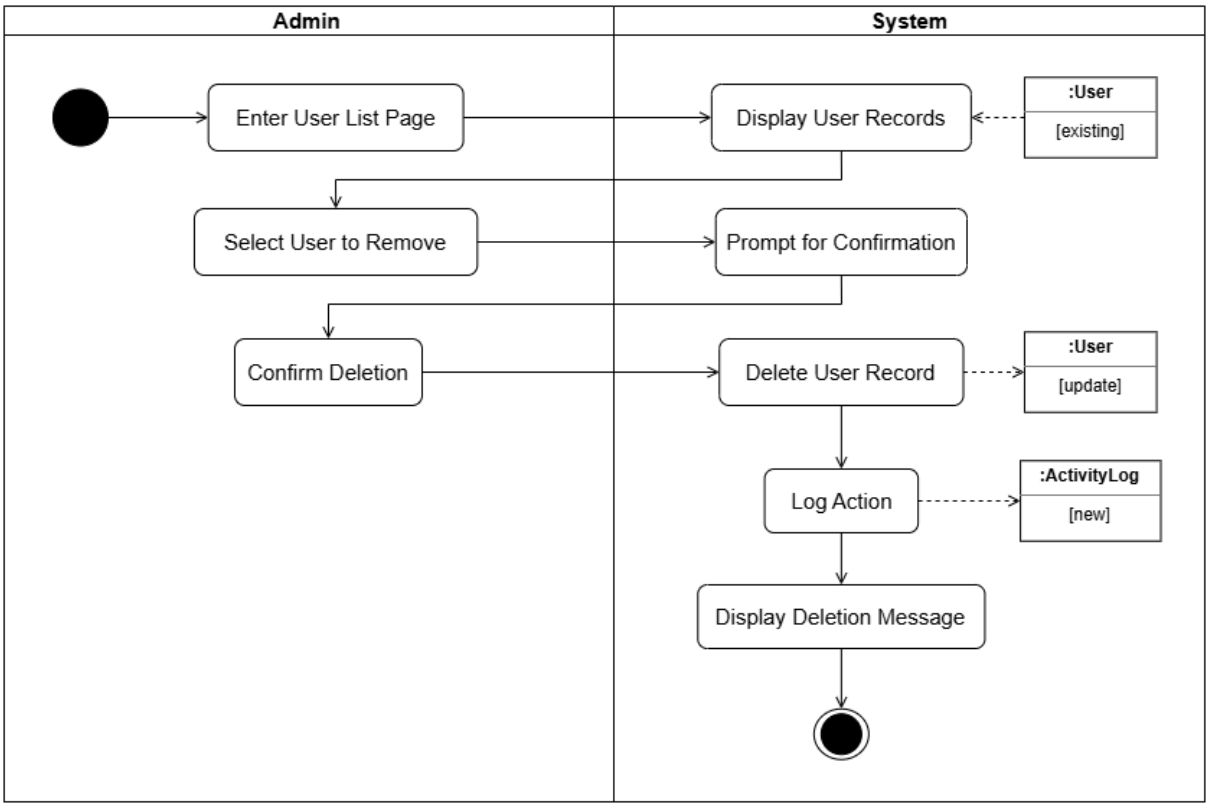


Figure 4.21: Activity Diagram for Remove User

4.6.15 Activity Diagram for Supplier Views PO

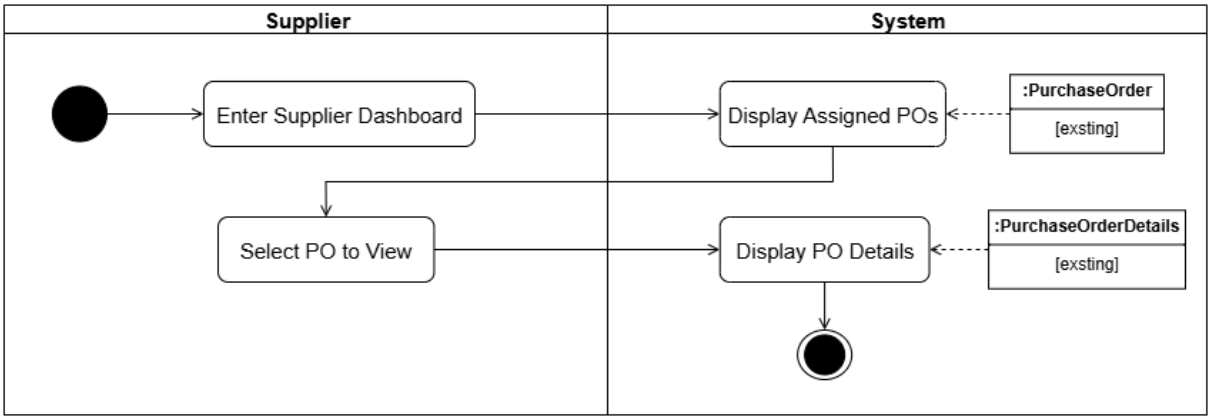


Figure 4.22: Activity Diagram for Supplier Views PO

4.6.16 Activity Diagram for Supplier Updates Contact Information

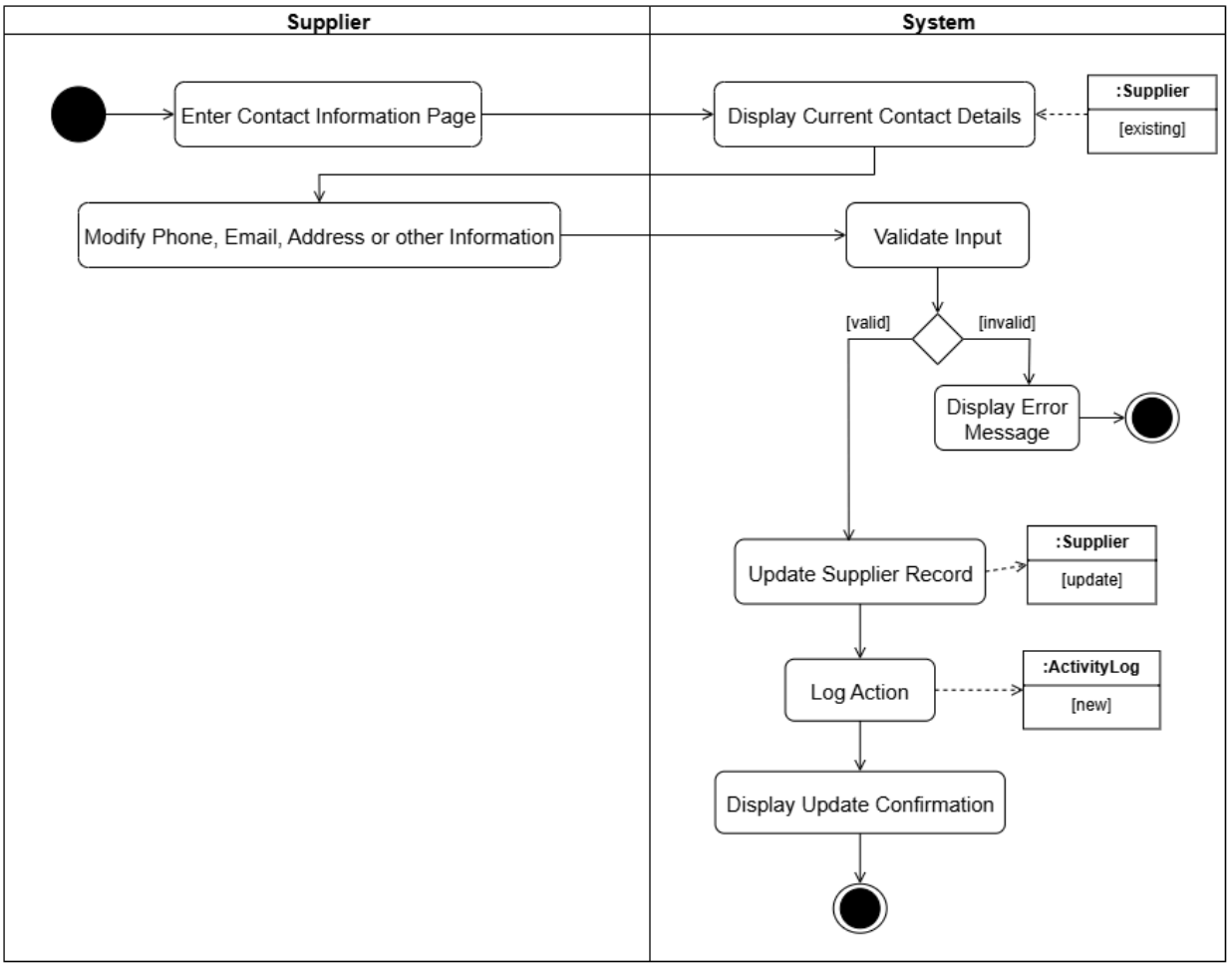


Figure 4.23: Activity Diagram for Supplier Updates Contact Information

4.6.17 Activity Diagram for System Generates Notification

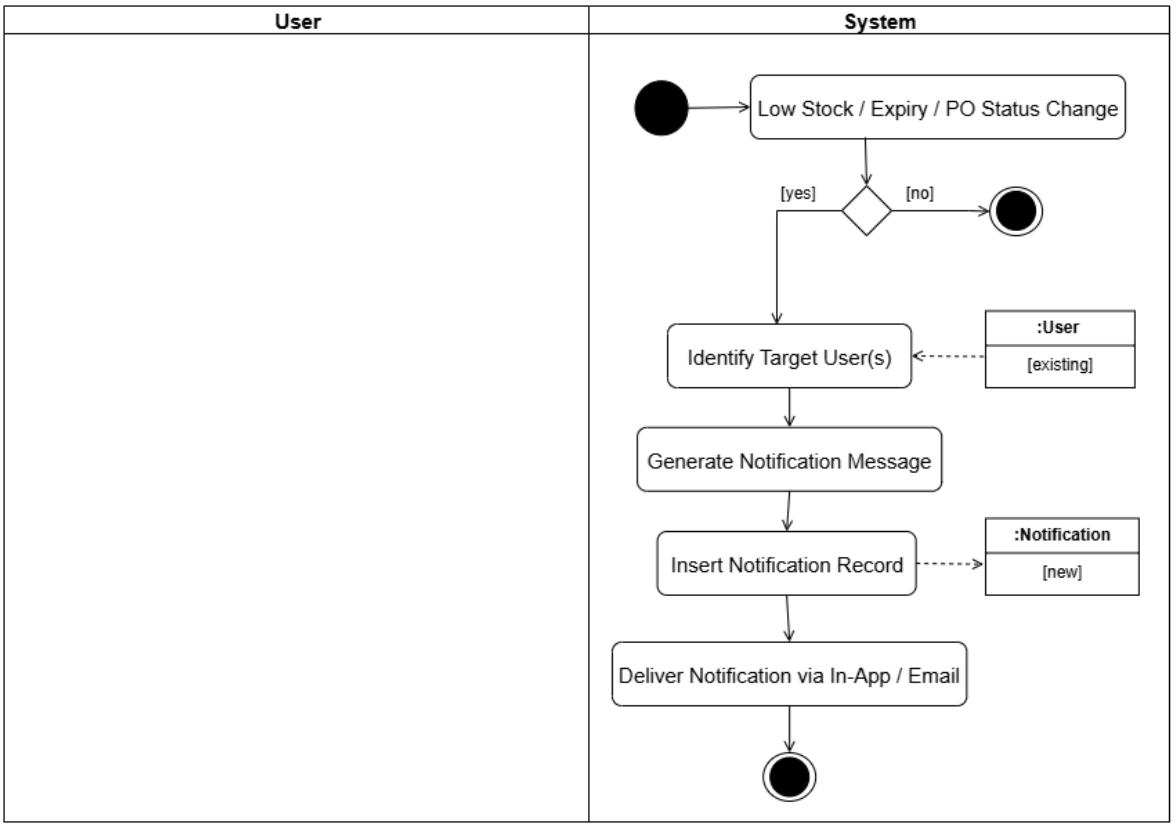


Figure 4.24: Activity Diagram for System Generates Notification

4.6.18 Activity Diagram for User Views Notification

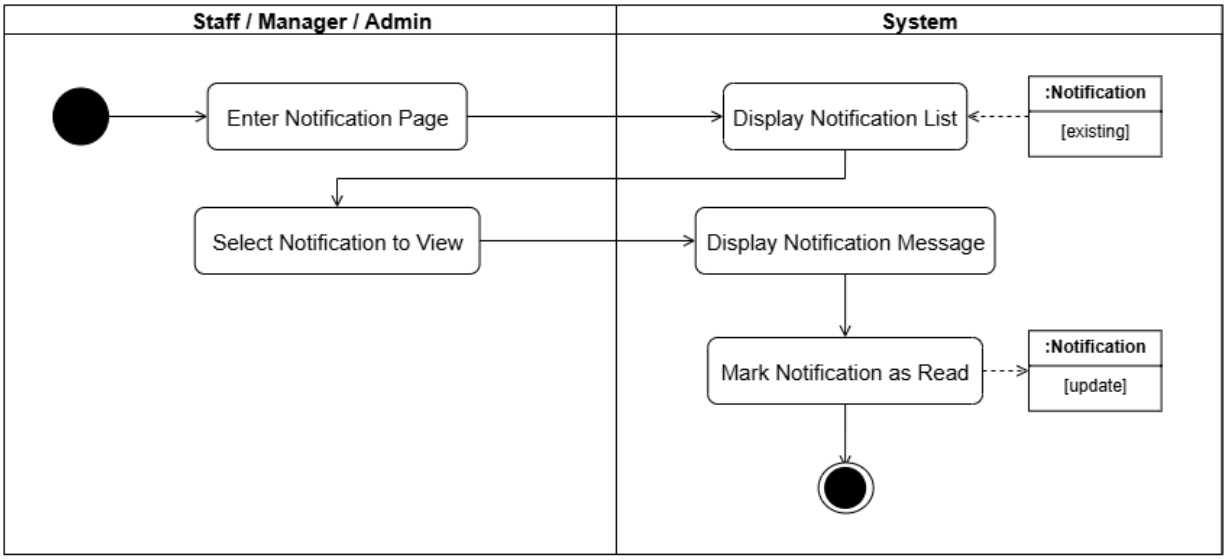


Figure 4.25: Activity Diagram for User Views Notification

4.7 Sequence Diagram

4.7.1 Sequence Diagram for Create Manual Purchase Order

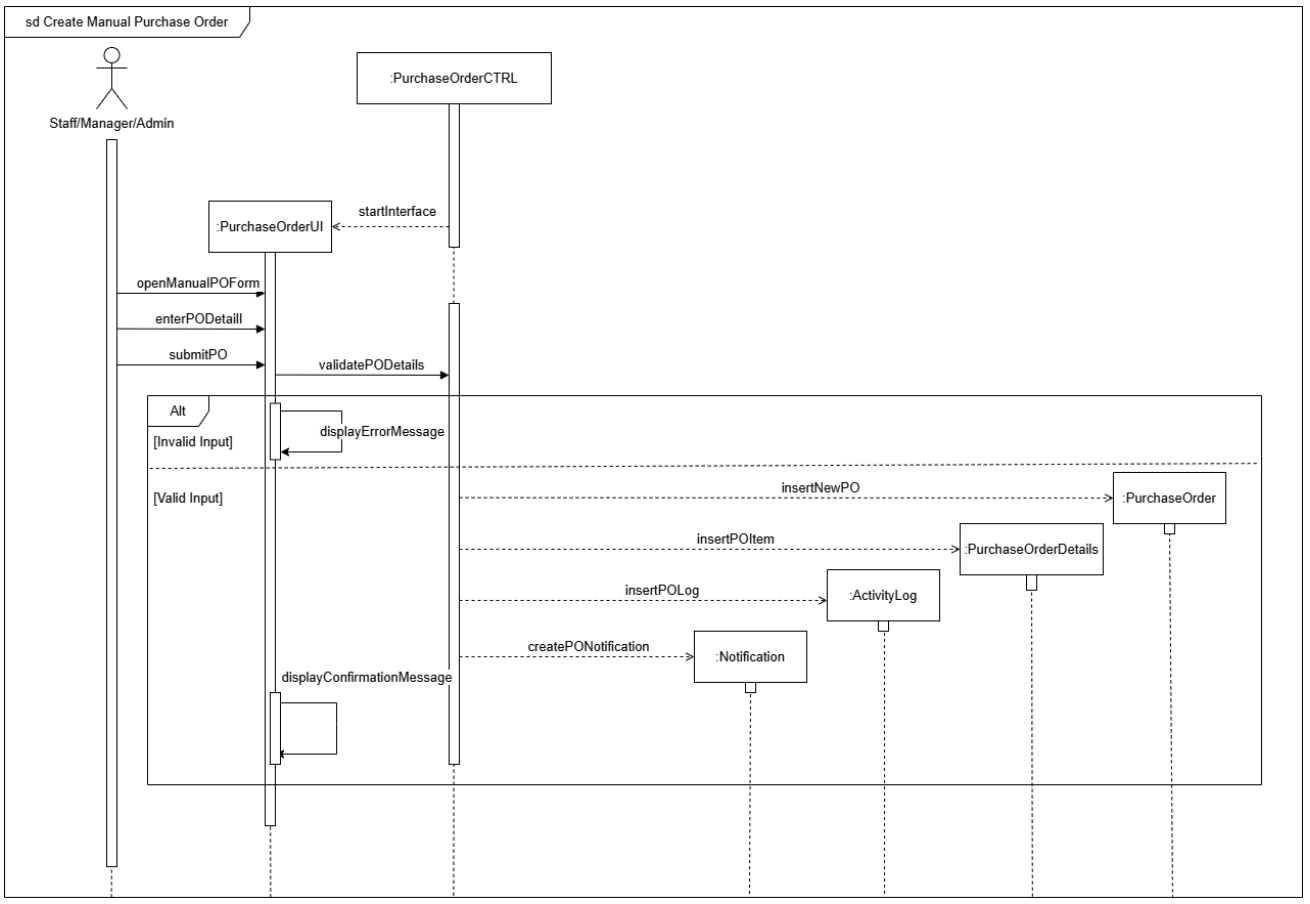


Figure 4.26: Sequence Diagram for Create Manual Purchase Order

4.7.2 Sequence Diagram for Approve or Reject Purchase Order

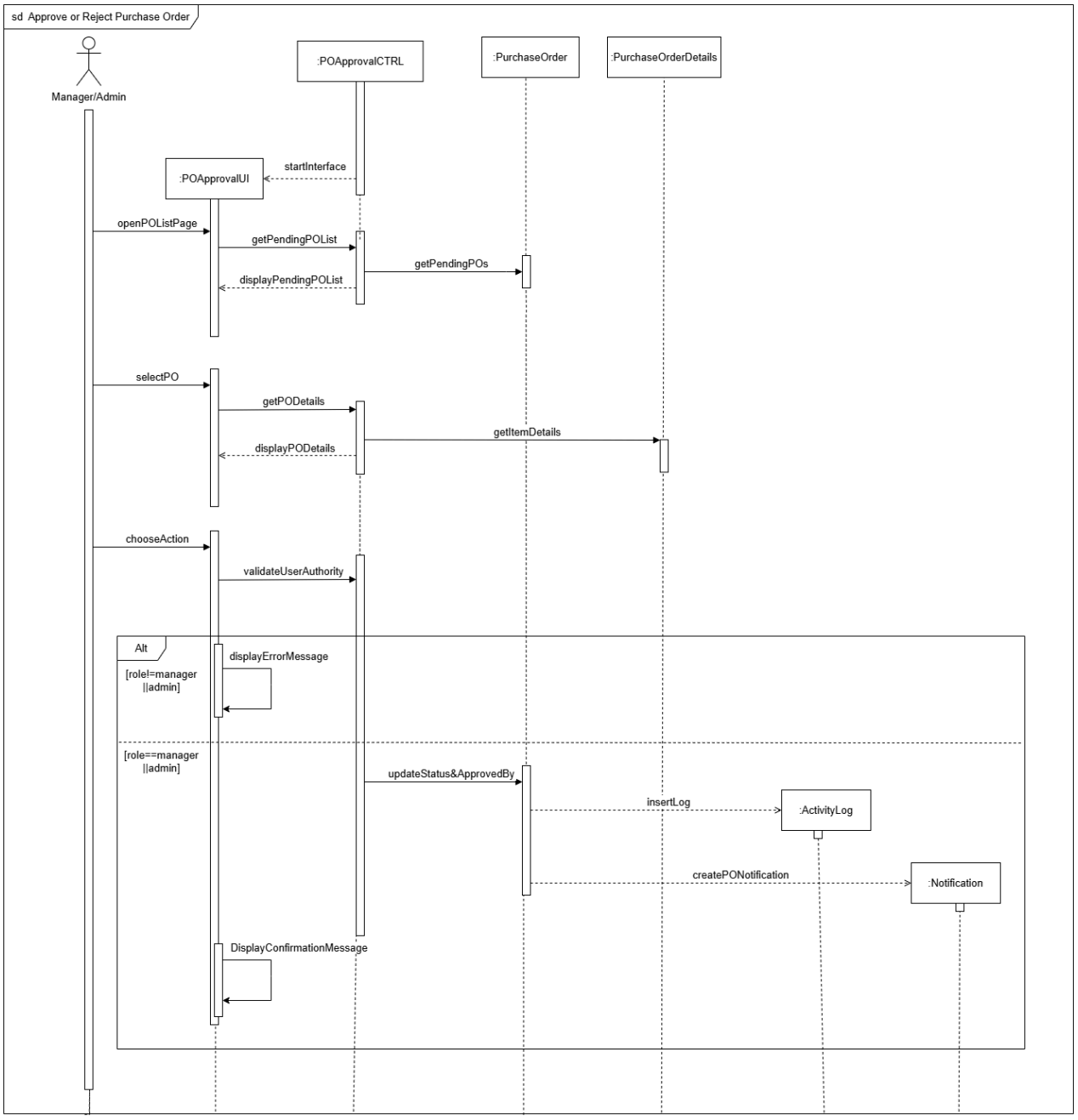


Figure 4.27: Sequence Diagram for Approve or Reject Purchase Order

4.7.3 Sequence Diagram for View PO History

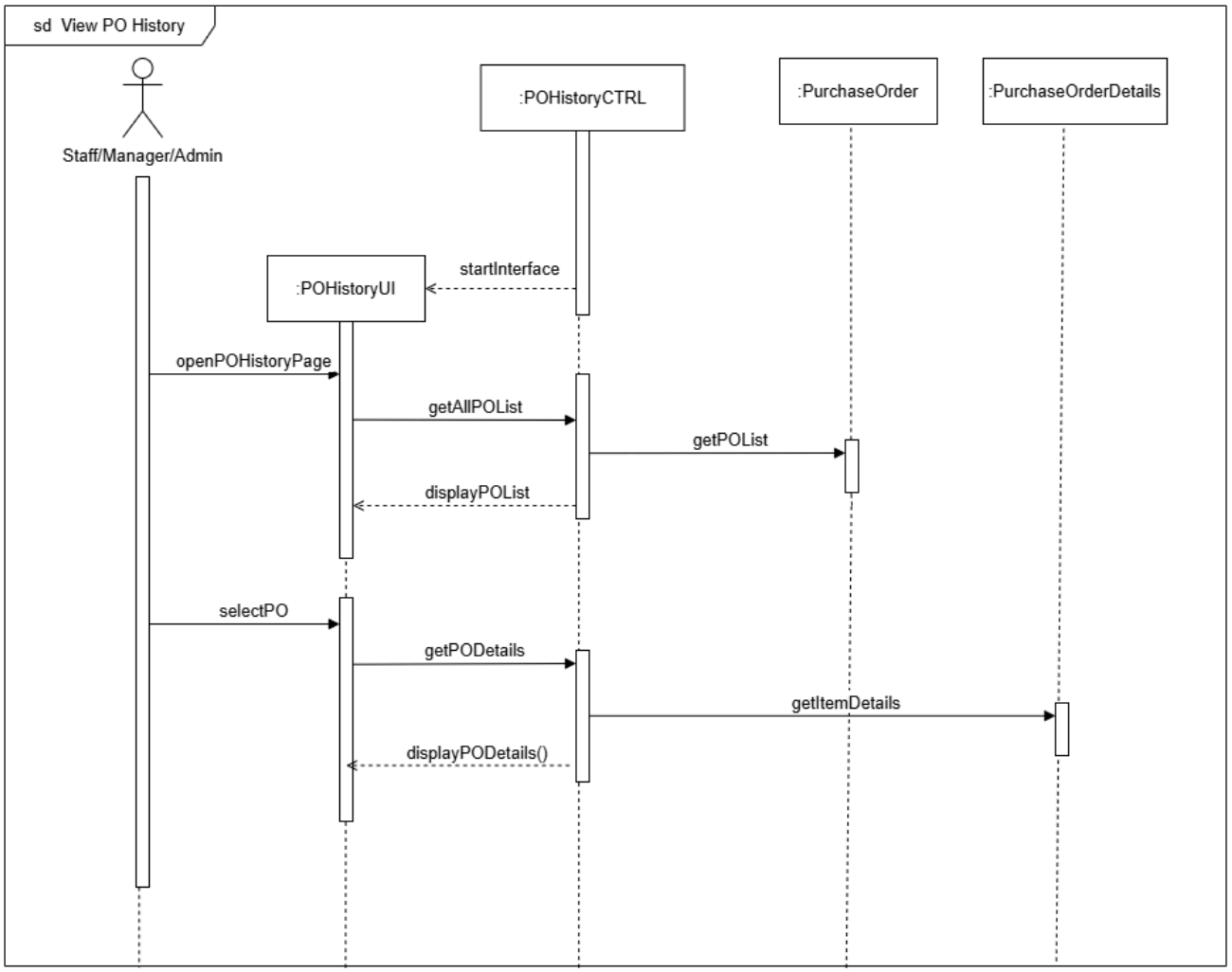


Figure 4.28: Sequence Diagram for View PO History

4.7.4 Sequence Diagram for Confirm Goods Receipt

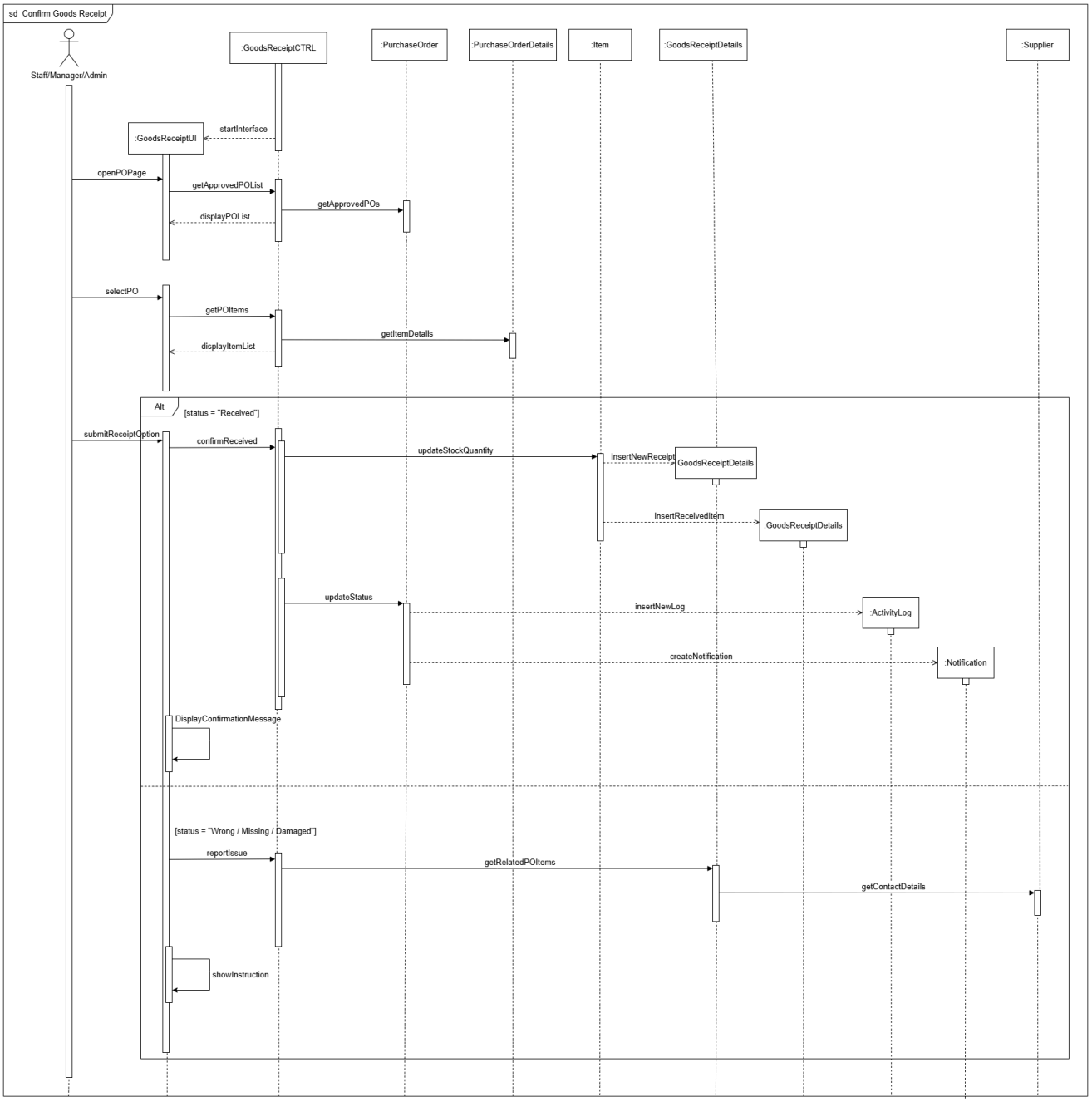


Figure 4.29: Sequence Diagram for Confirm Goods Receipt

4.7.5 Sequence Diagram for Add Supplier

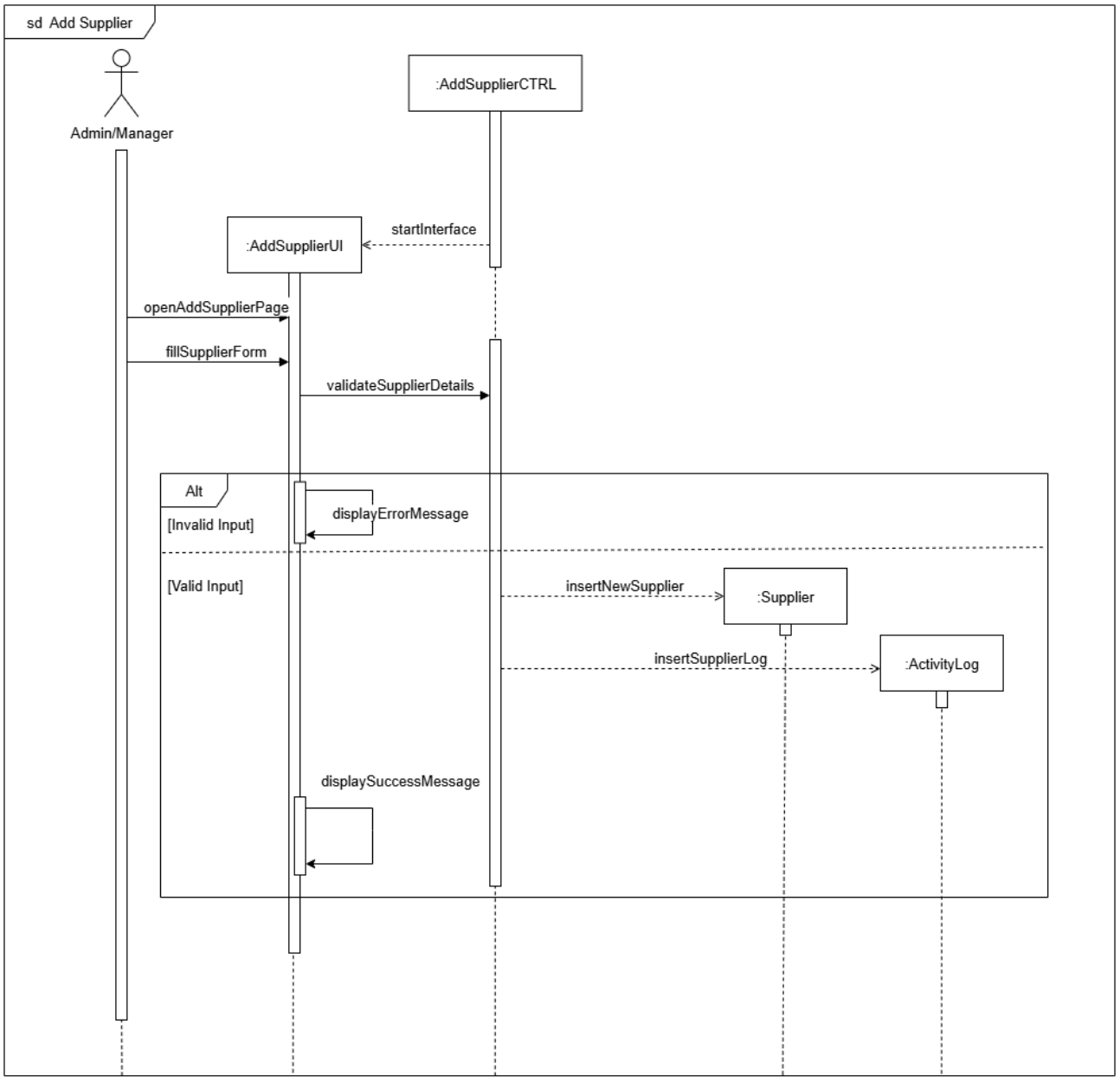


Figure 4.30: Sequence Diagram for Add Supplier

4.7.6 Sequence Diagram for Edit Supplier

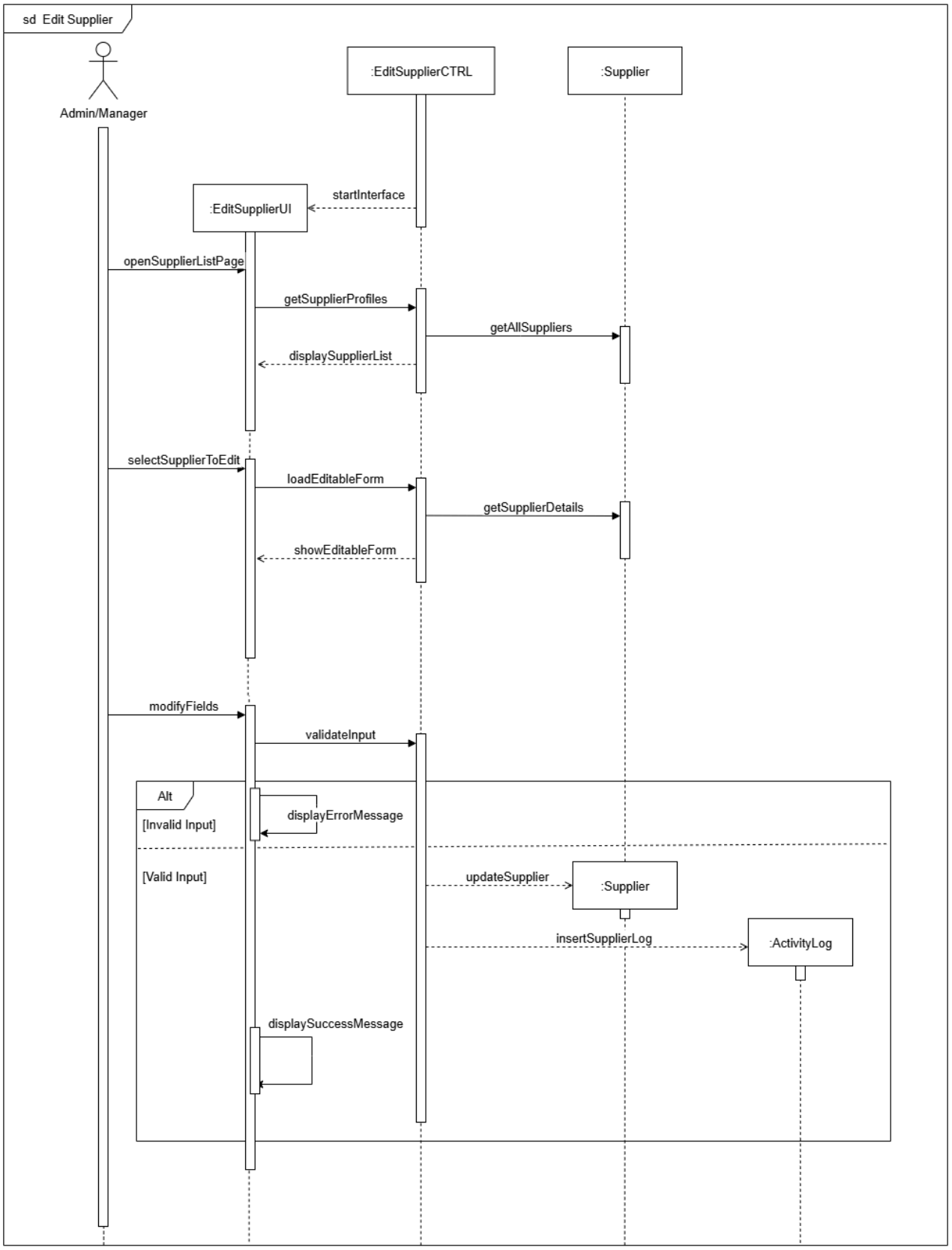


Figure 4.31: Sequence Diagram for Edit Supplier

4.7.7 Sequence Diagram for Remove Supplier

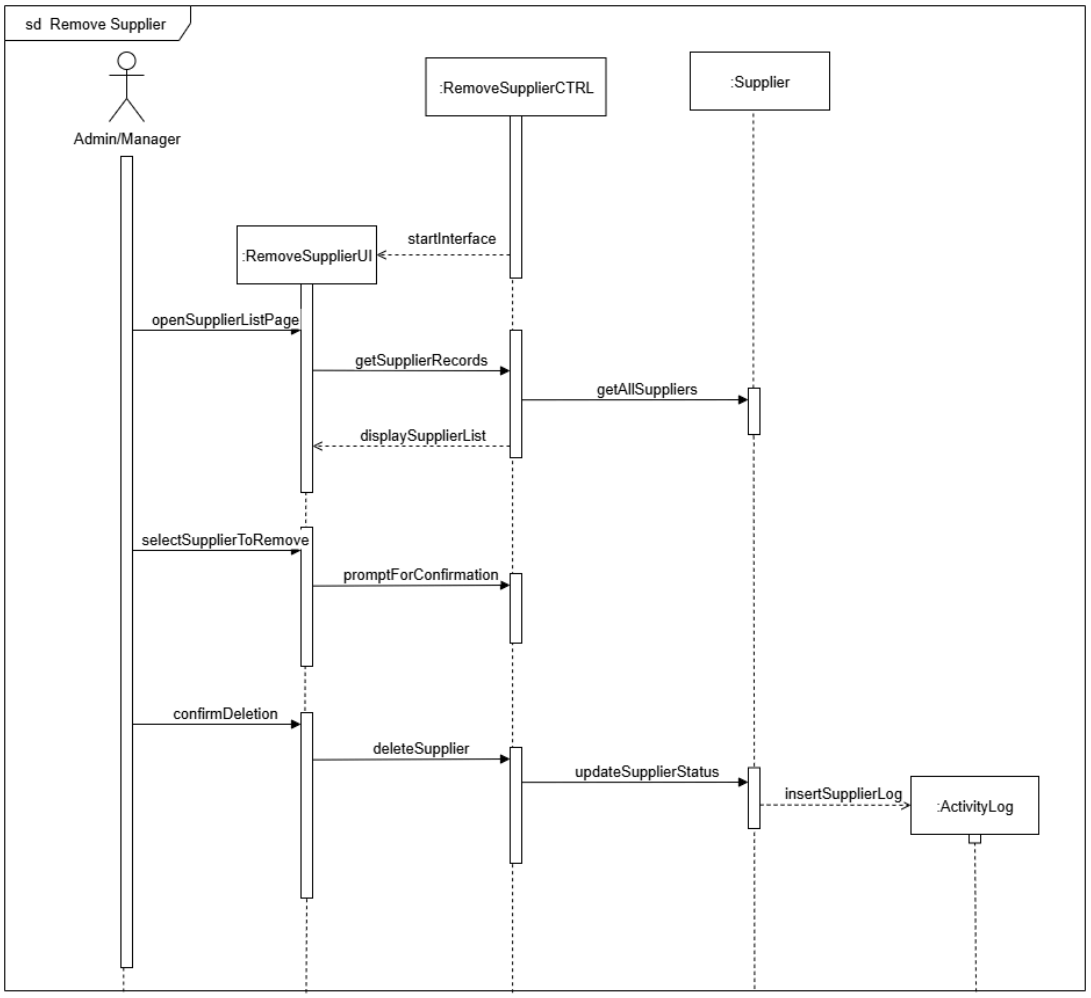


Figure 4.32: Sequence Diagram for Remove Supplier

4.7.8 Sequence Diagram for View Supplier

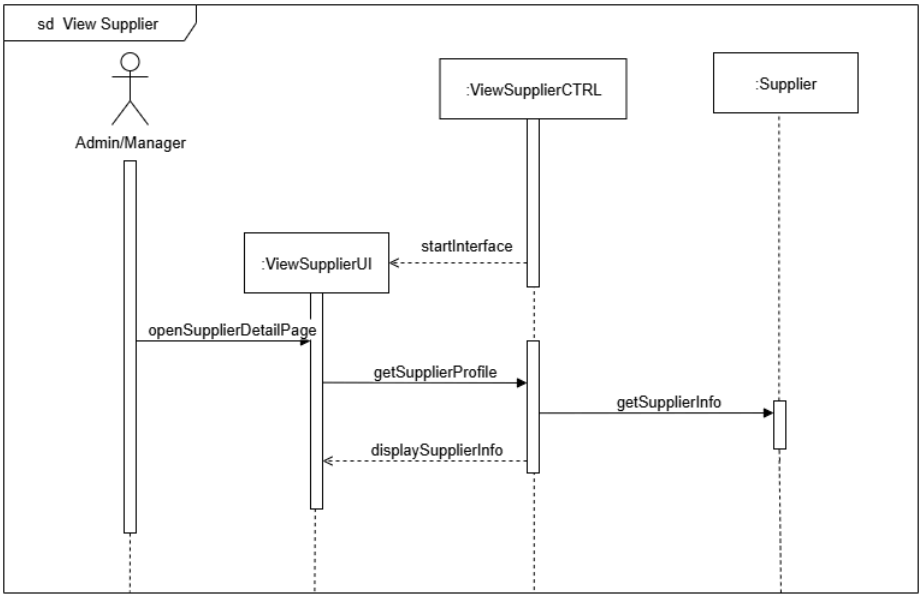


Figure 4.33: Sequence Diagram for View Supplier

4.7.9 Sequence Diagram for Assign/Modify Roles and Permissions

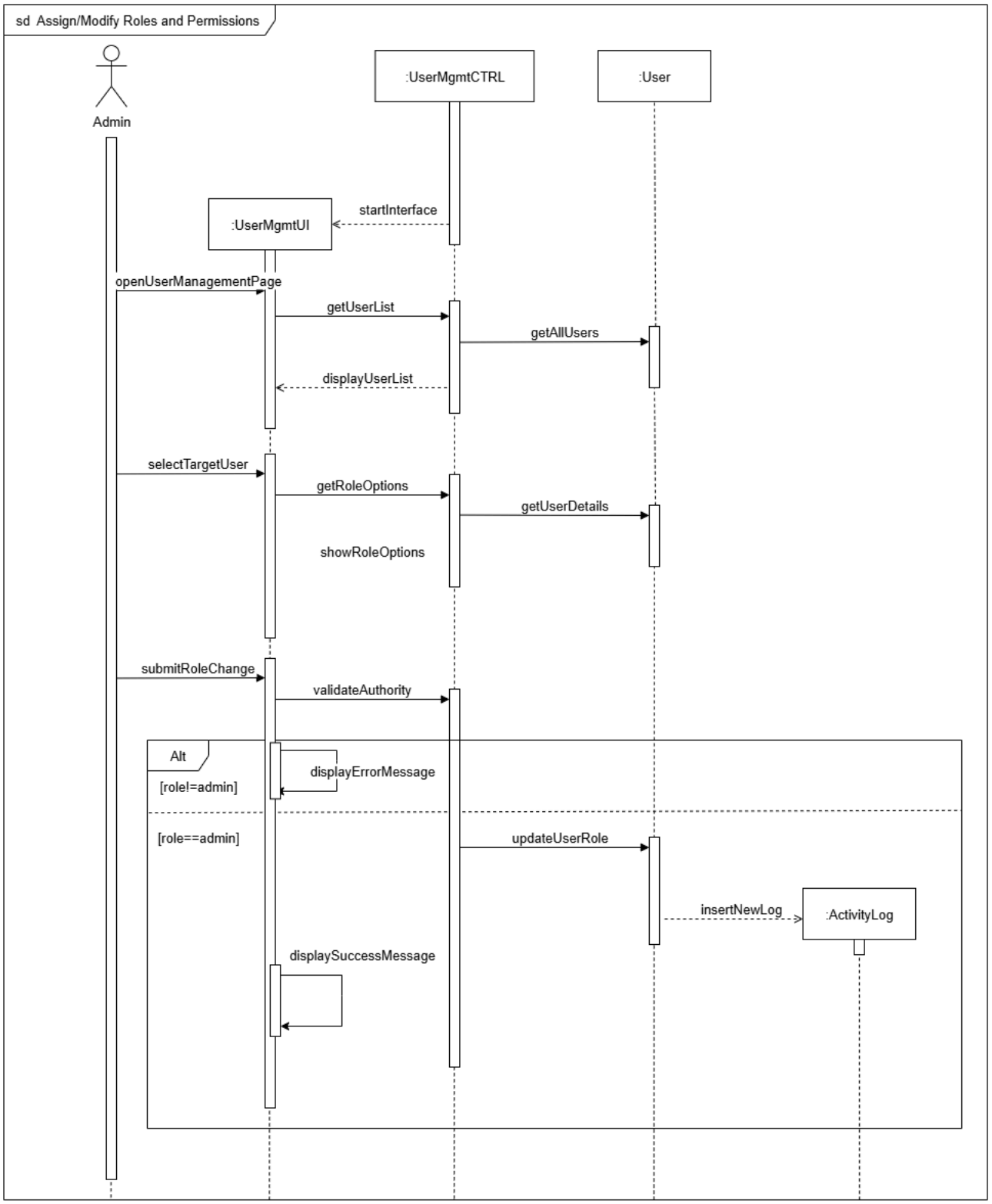


Figure 4.34: Sequence Diagram for Assign/Modify Roles and Permissions

4.7.10 Sequence Diagram for View User Activity Log

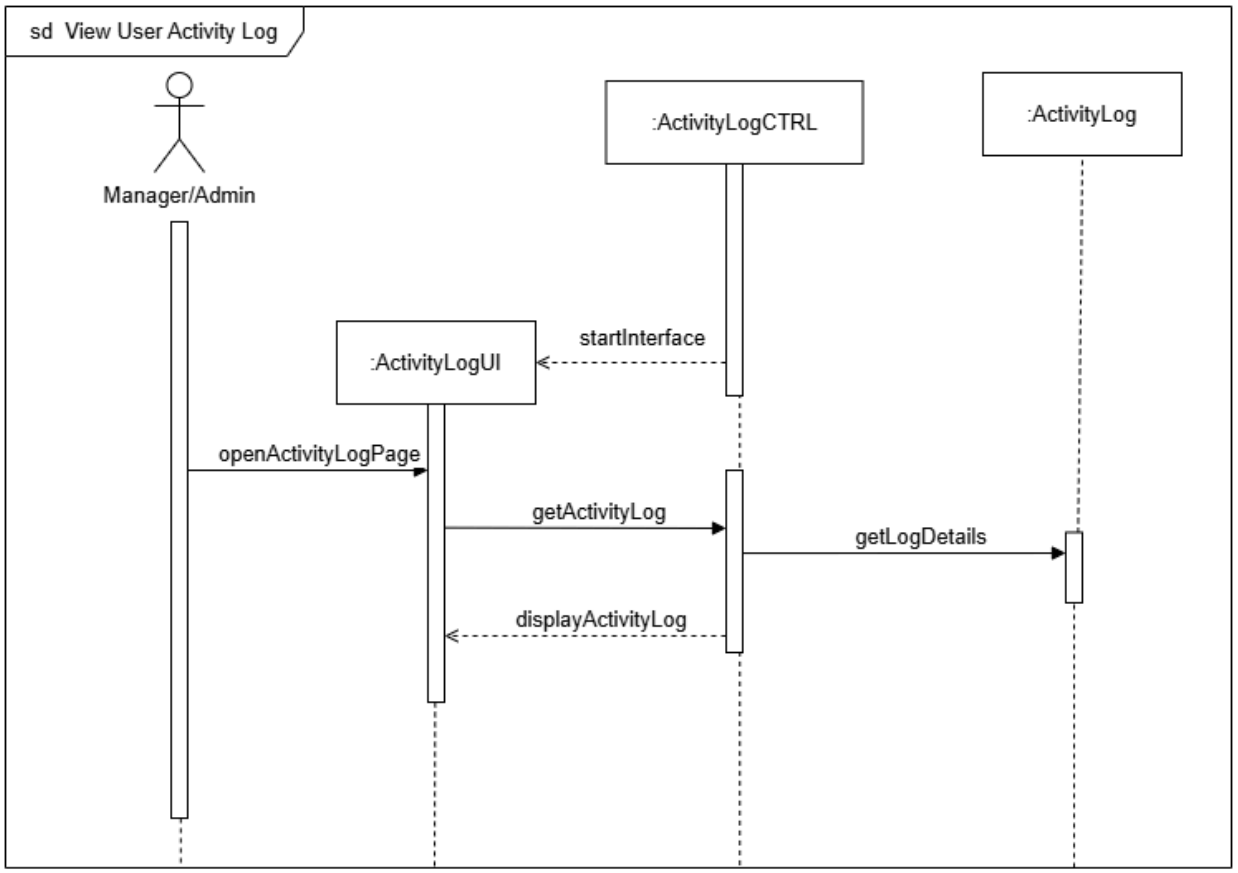


Figure 4.35: Sequence Diagram for View User Activity Log

4.7.11 Sequence Diagram for Add User

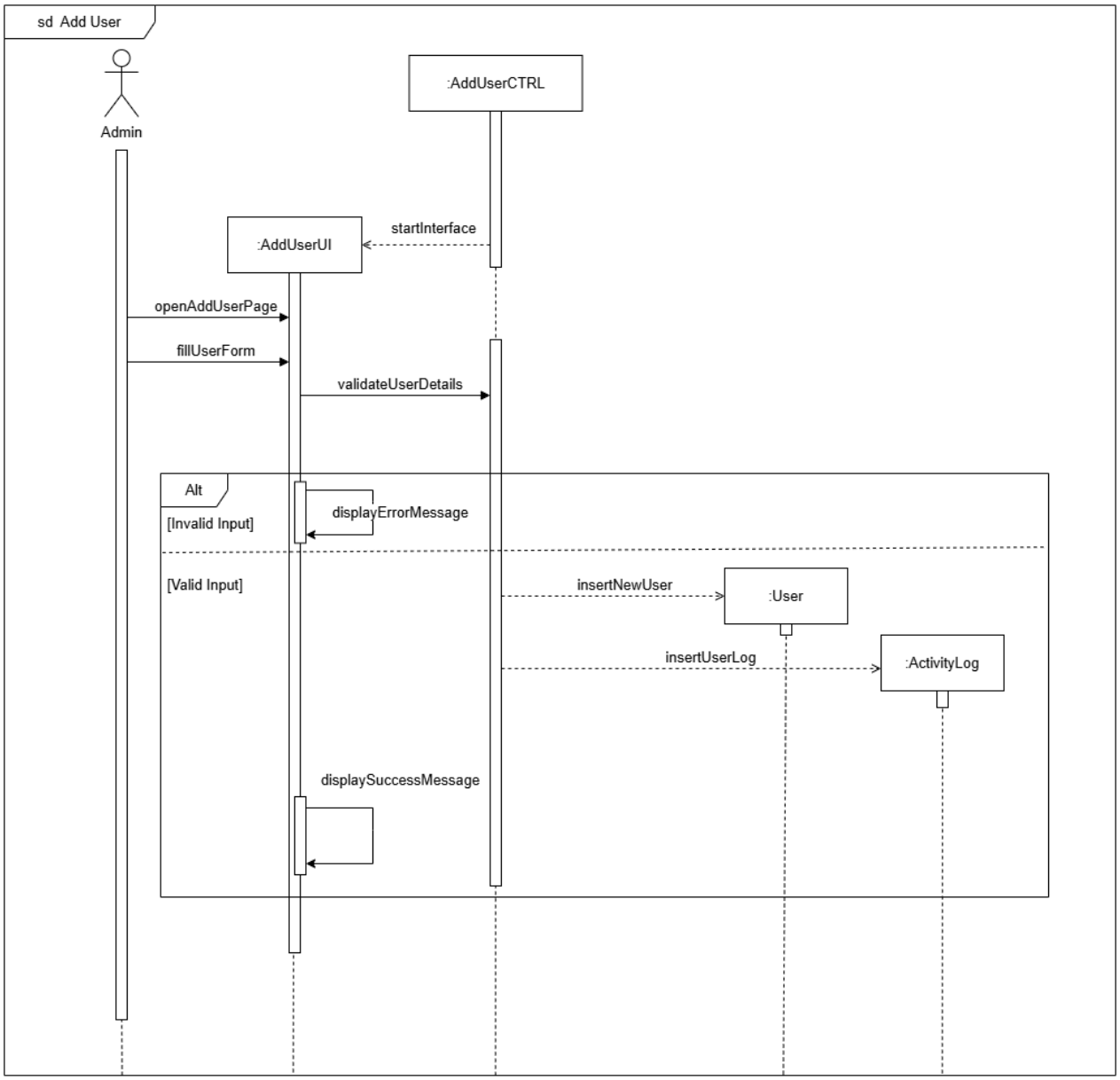


Figure 4.36: Sequence Diagram for Add User

4.7.12 Sequence Diagram for Edit User

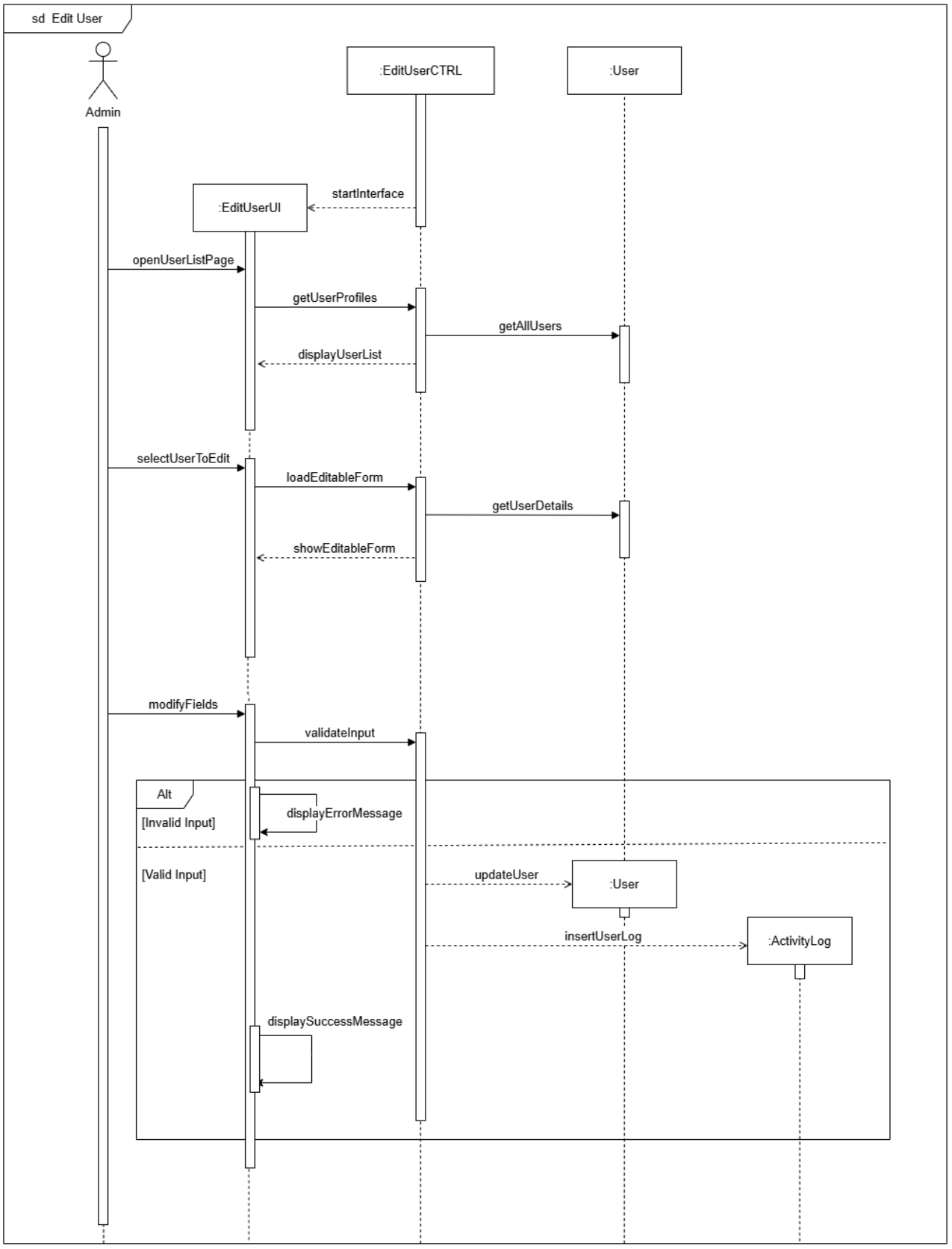


Figure 4.37: Sequence Diagram for Edit User

4.7.13 Sequence Diagram for Remove User

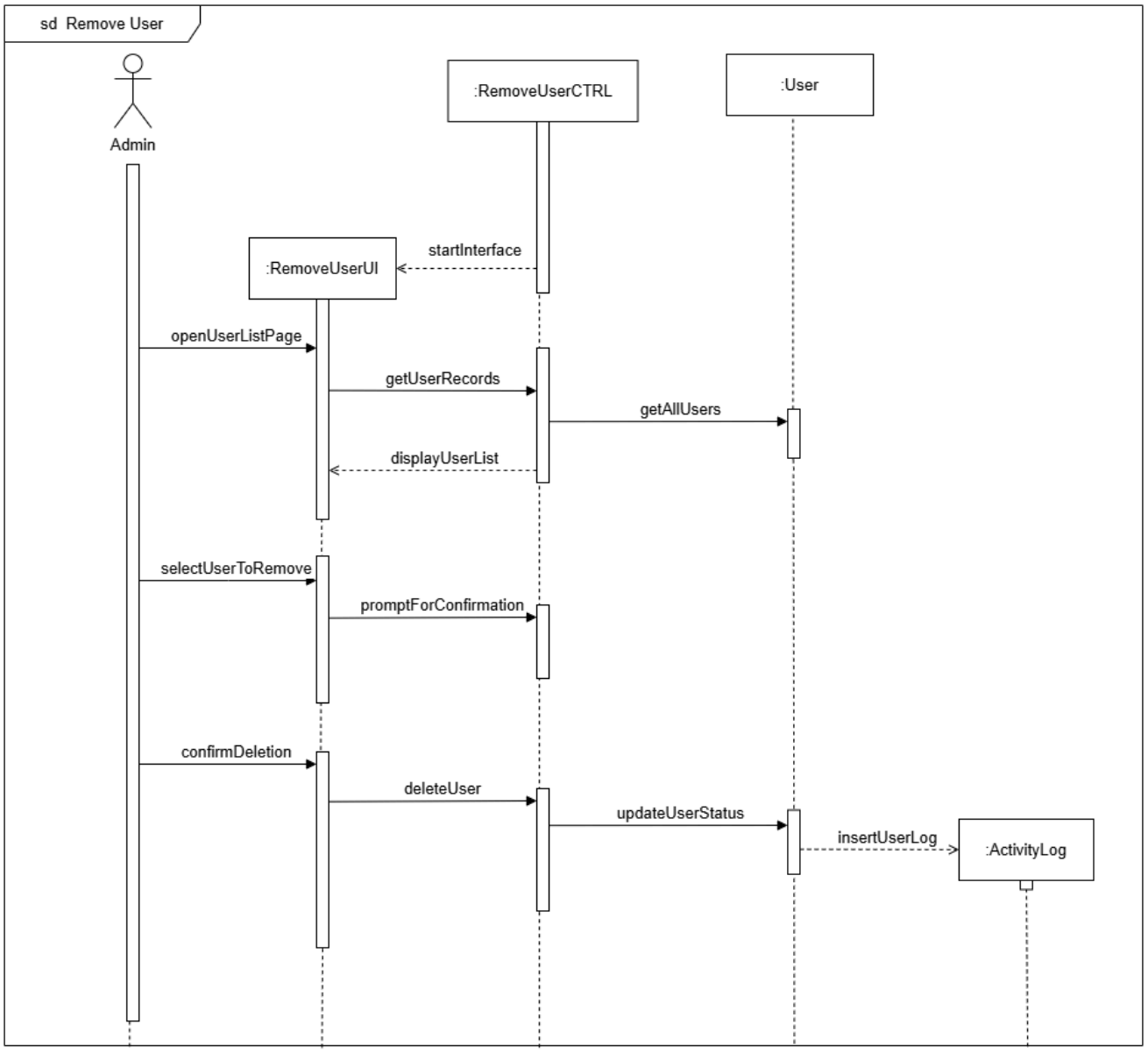


Figure 4.38: Sequence Diagram for Remove User

4.7.14 Sequence Diagram for Supplier Views PO

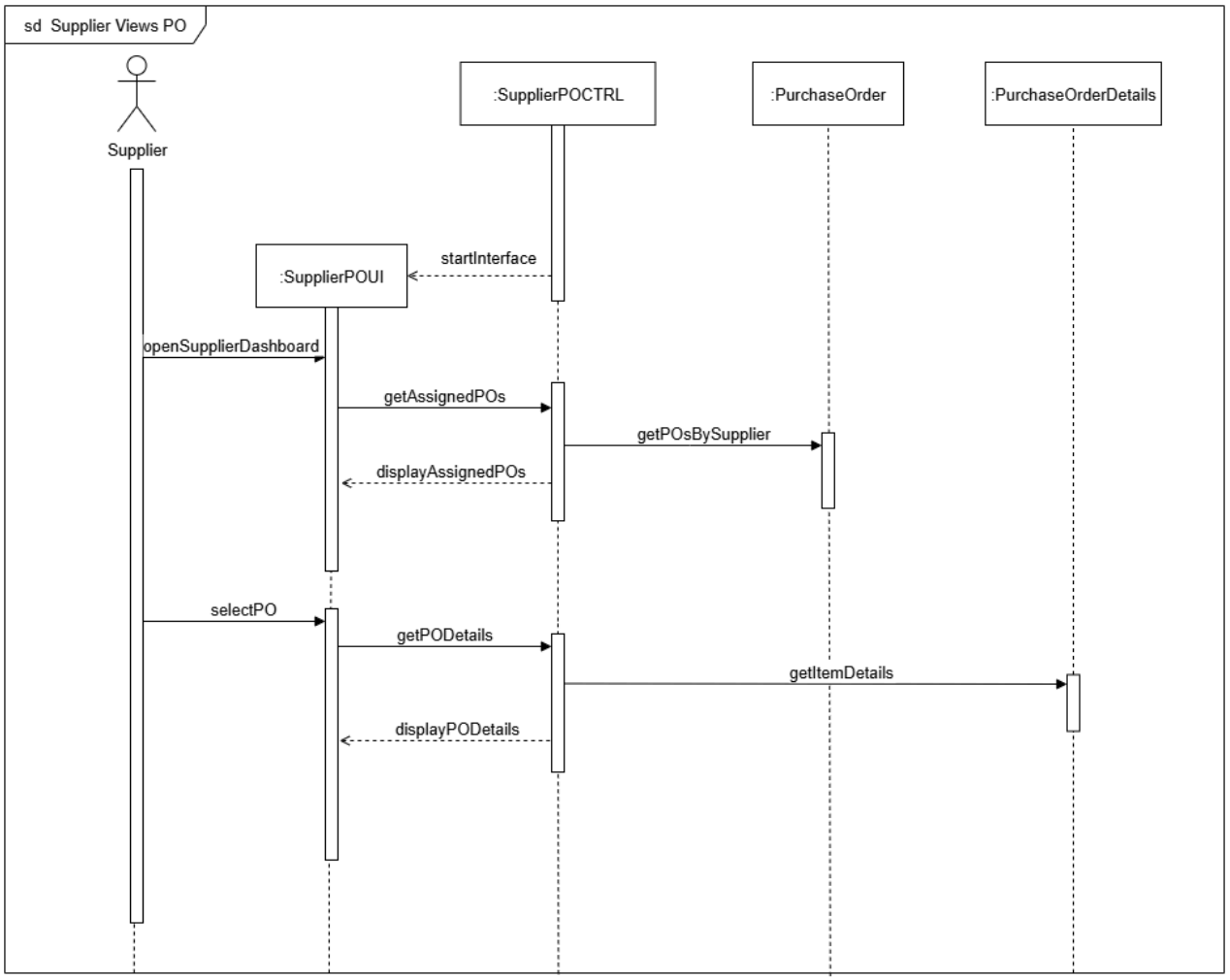


Figure 4.39: Sequence Diagram for Supplier Views PO

4.7.15 Sequence Diagram for Supplier Updates Contact Information

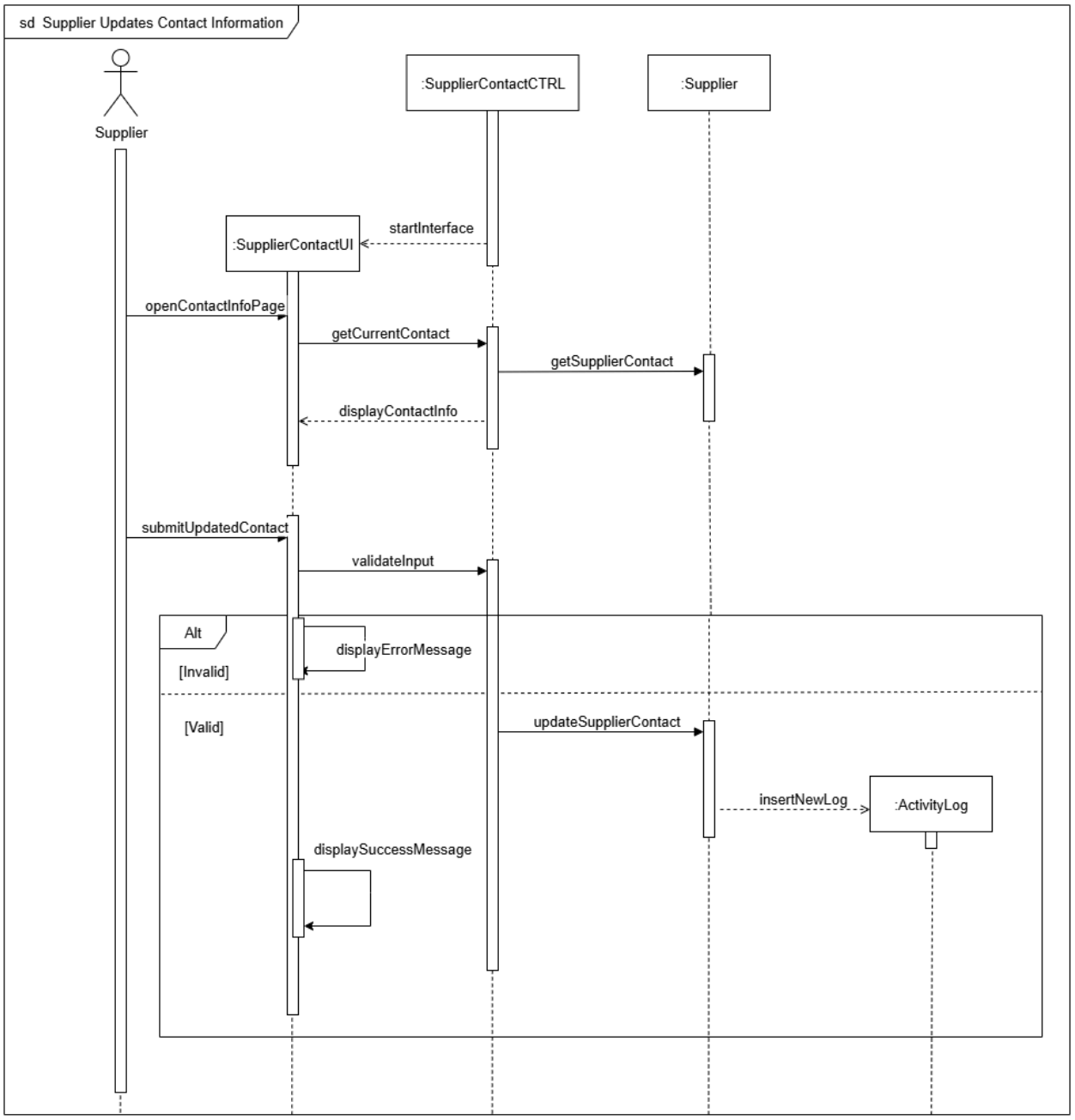


Figure 4.40: Sequence Diagram for Supplier Updates Contact Information

4.7.16 Sequence Diagram for User Views Notification

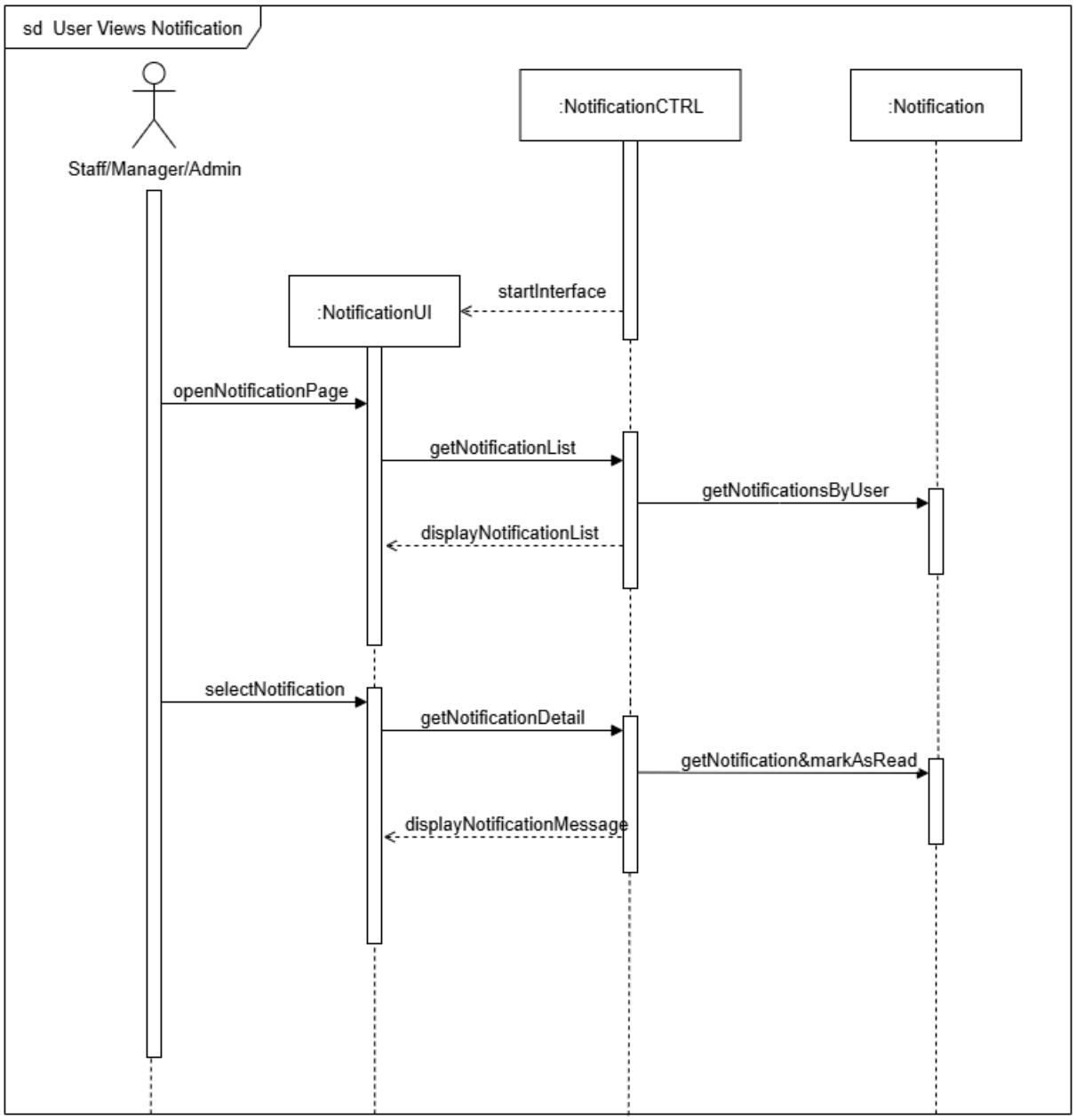


Figure 4.41: Sequence Diagram for User Views Notification

4.8 Layered Software Architecture Diagram

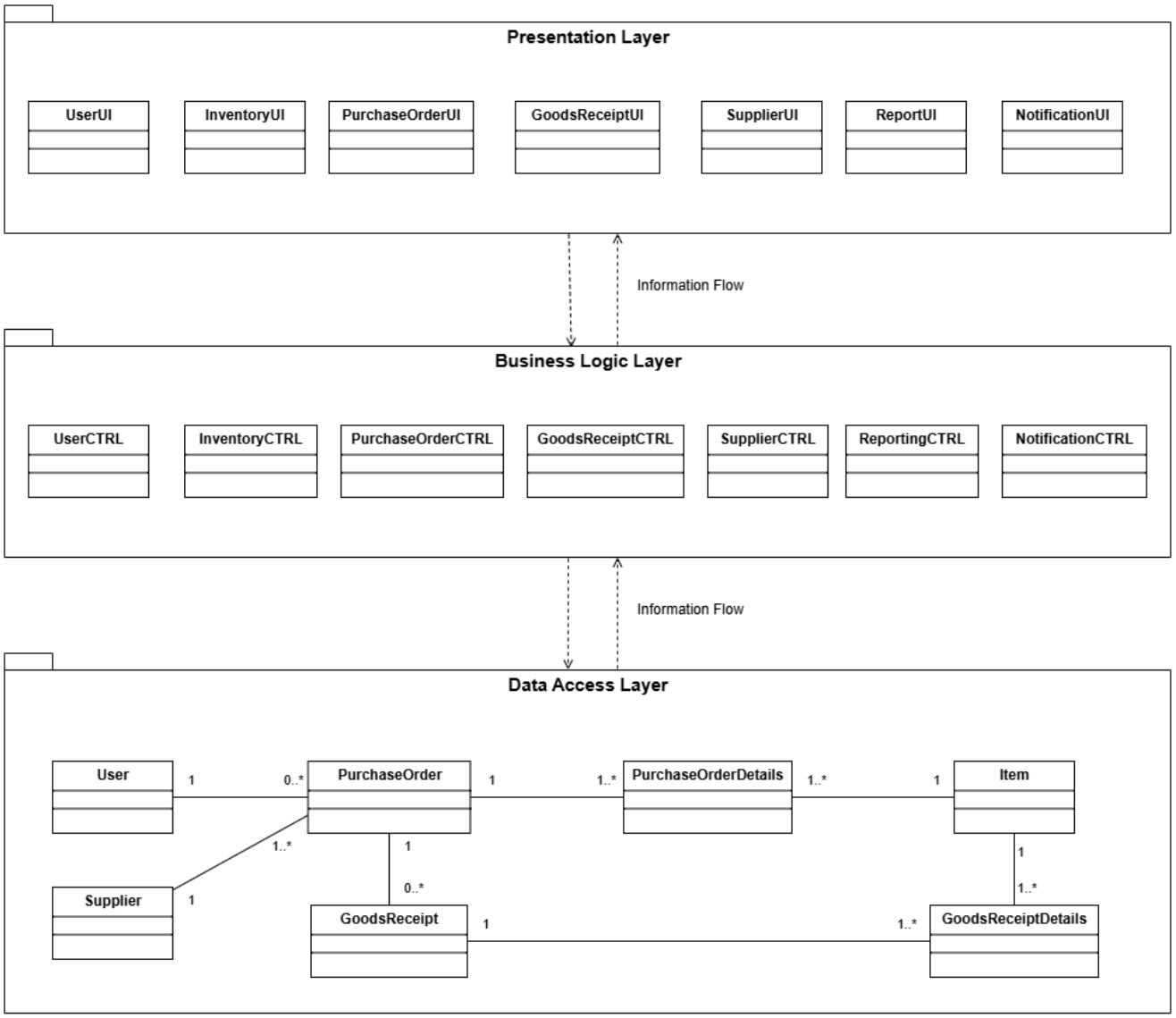


Figure 4.42: Layered Software Architecture Diagram

4.9 User Interface (UI) Design

4.9.1 Supplier List Page

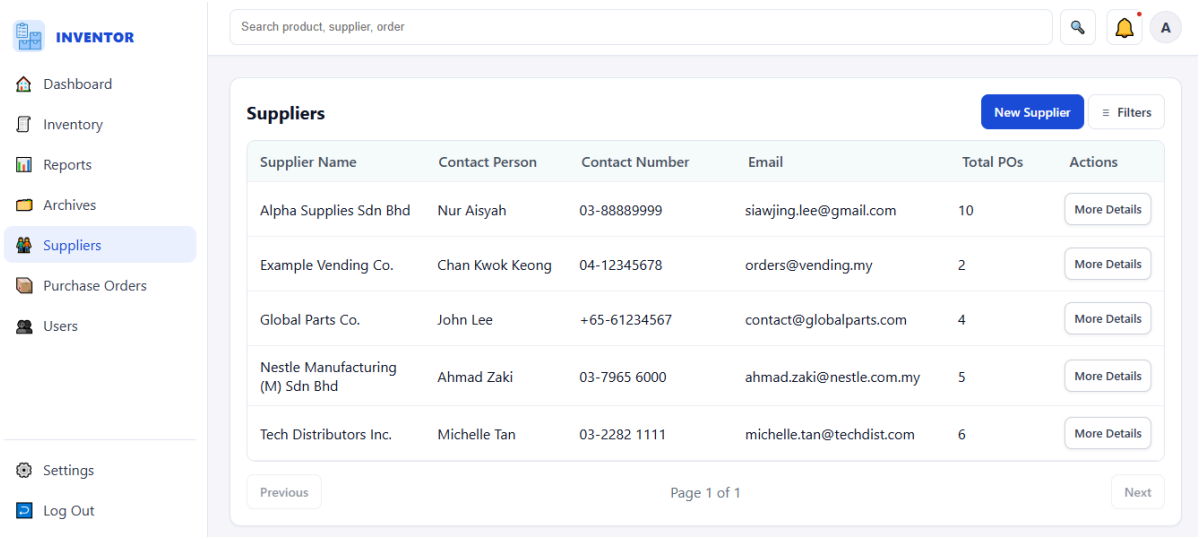


Figure 4.43: Supplier List Page UI

4.9.2 Supplier List Filter

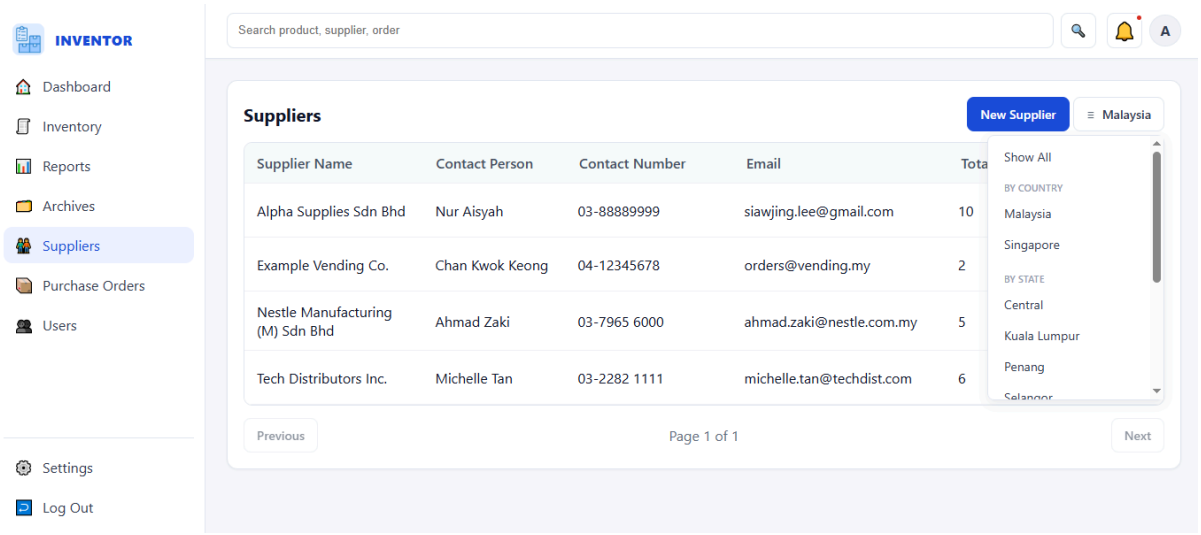


Figure 4.44: Supplier List Filter UI

4.9.3 Add New Supplier

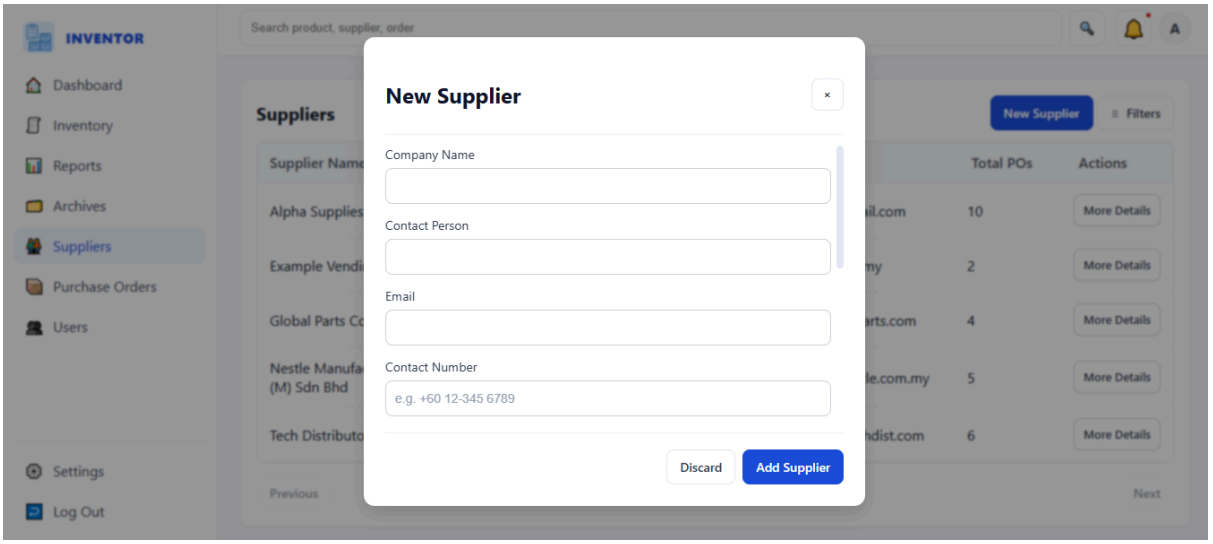


Figure 4.45: Add New Supplier UI 1

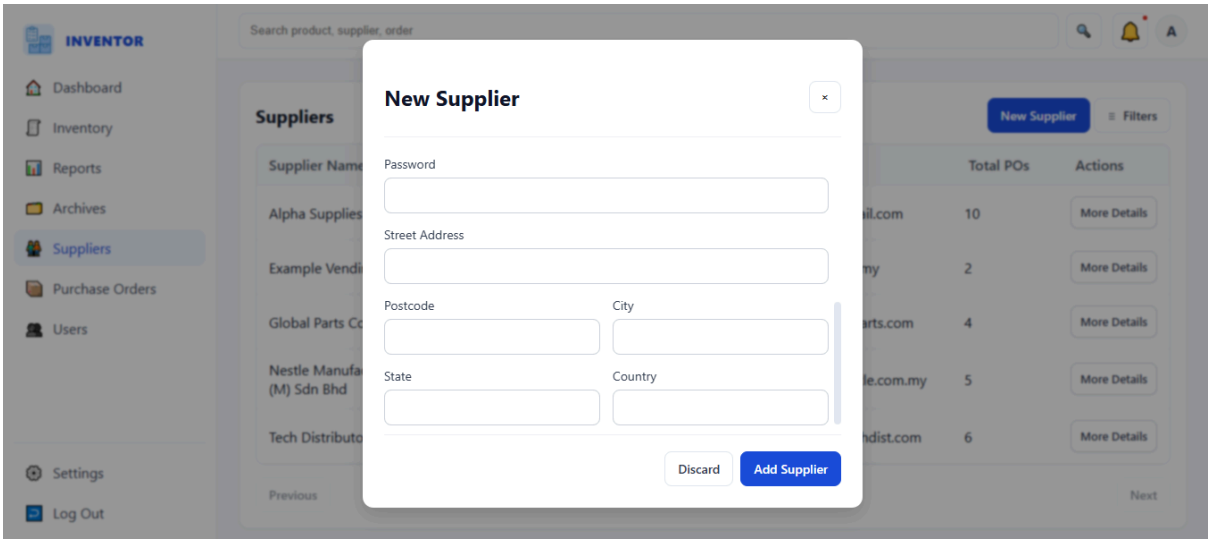


Figure 4.46: Add New Supplier UI 2

4.9.4 Supplier Details Page

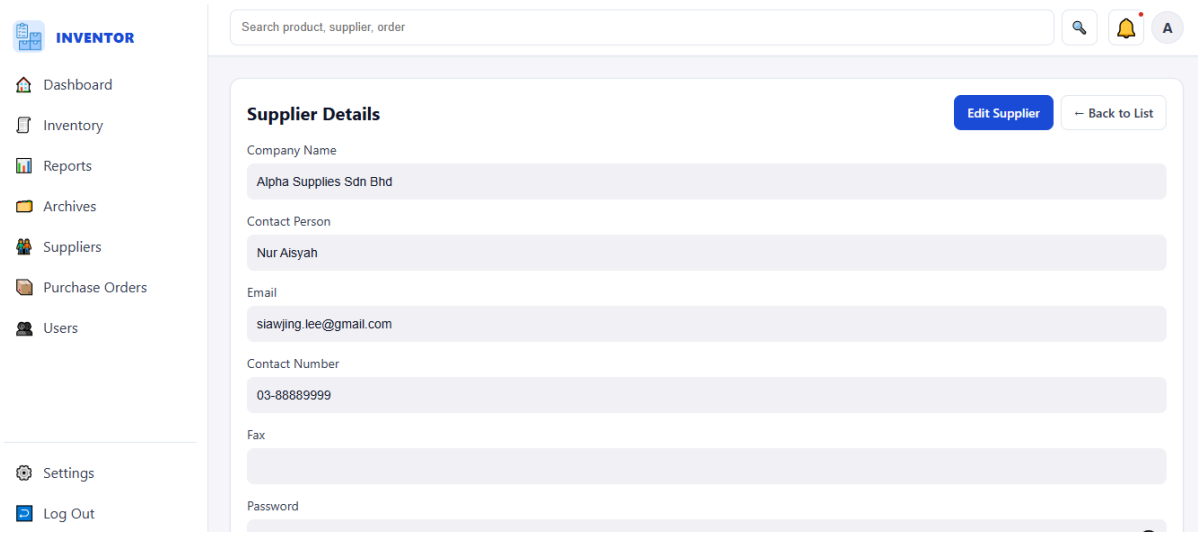


Figure 4.47: Supplier Details Page UI

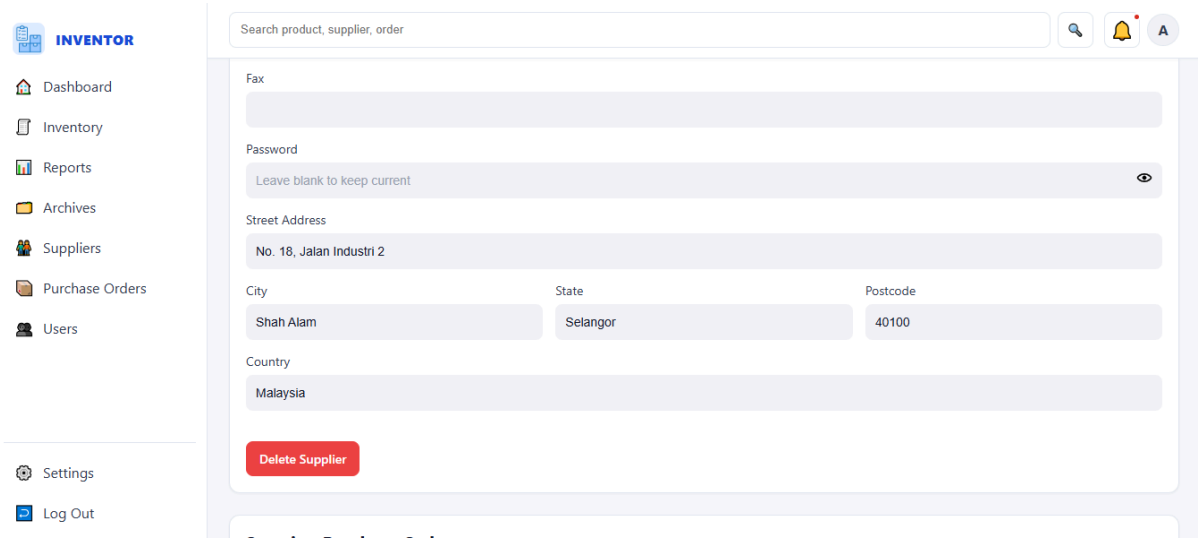


Figure 4.48: Supplier Details Page UI (Delete Supplier)

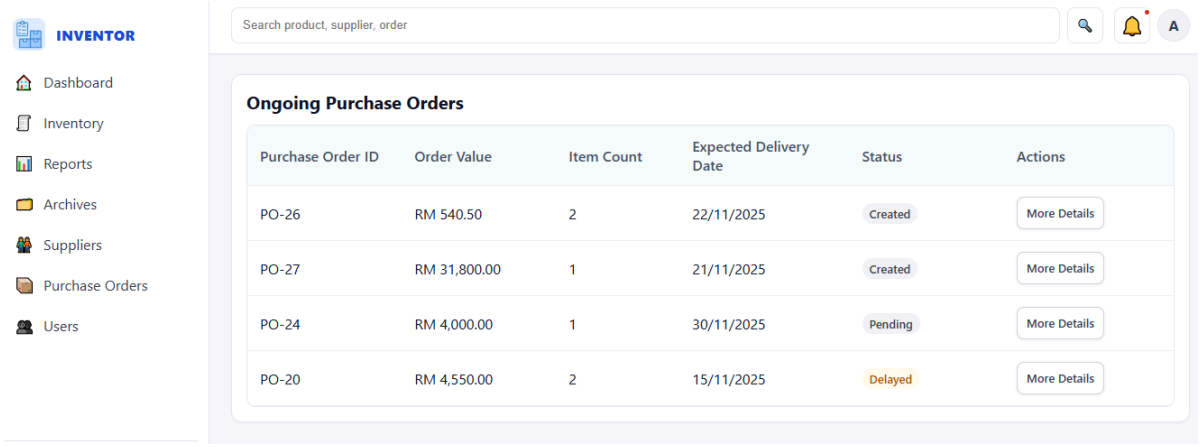


Figure 4.49: Supplier Details Page UI (Ongoing PO)

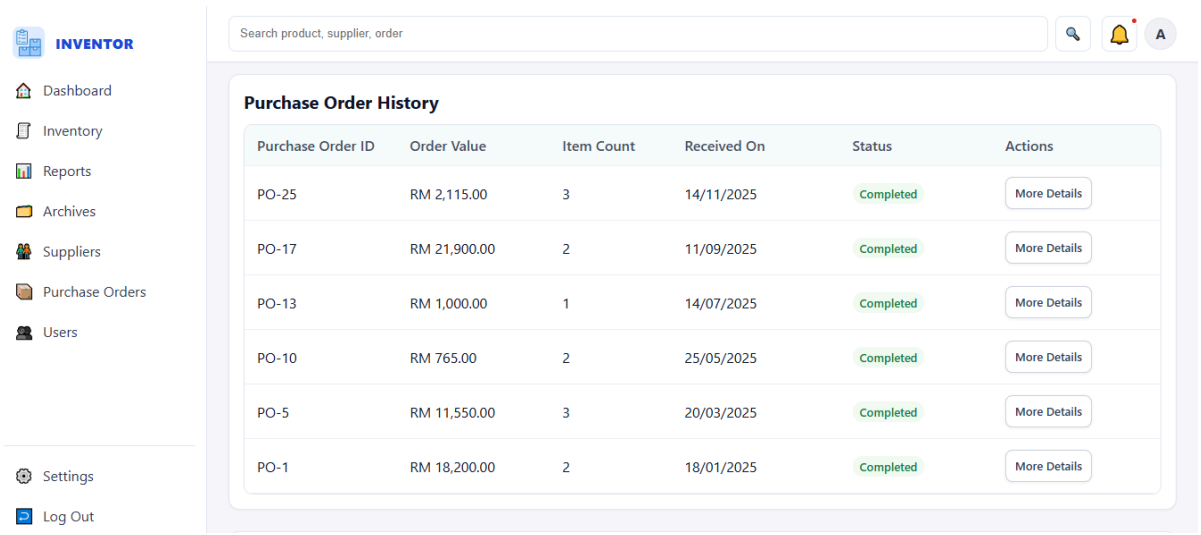


Figure 4.50: Supplier Details Page UI (PO History)

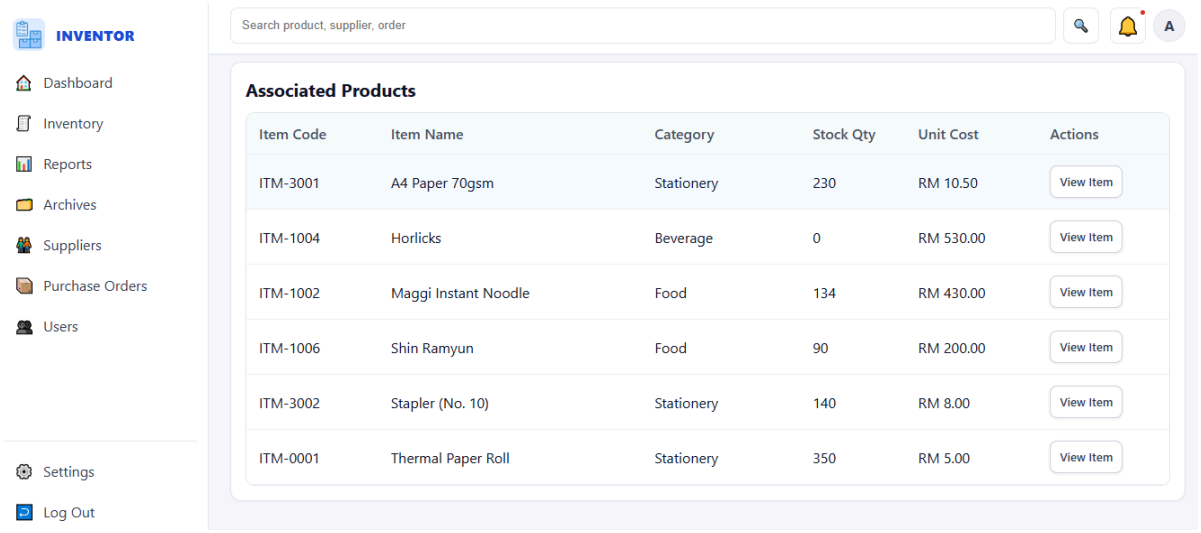


Figure 4.51: Supplier Details Page UI (Associated Products)

4.9.5 Edit Supplier Details

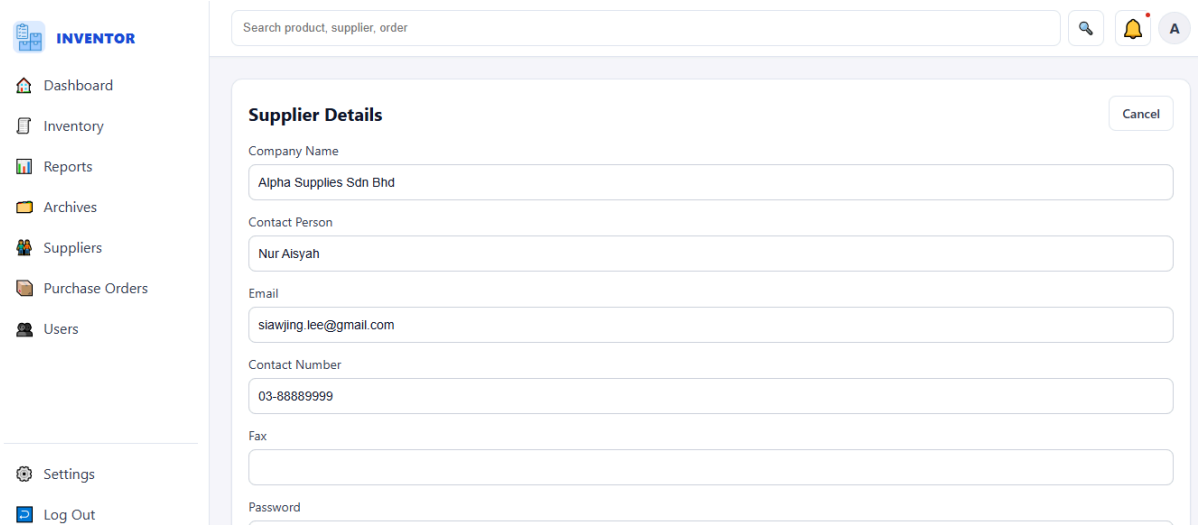


Figure 4.52: Edit Supplier Details Page UI 1

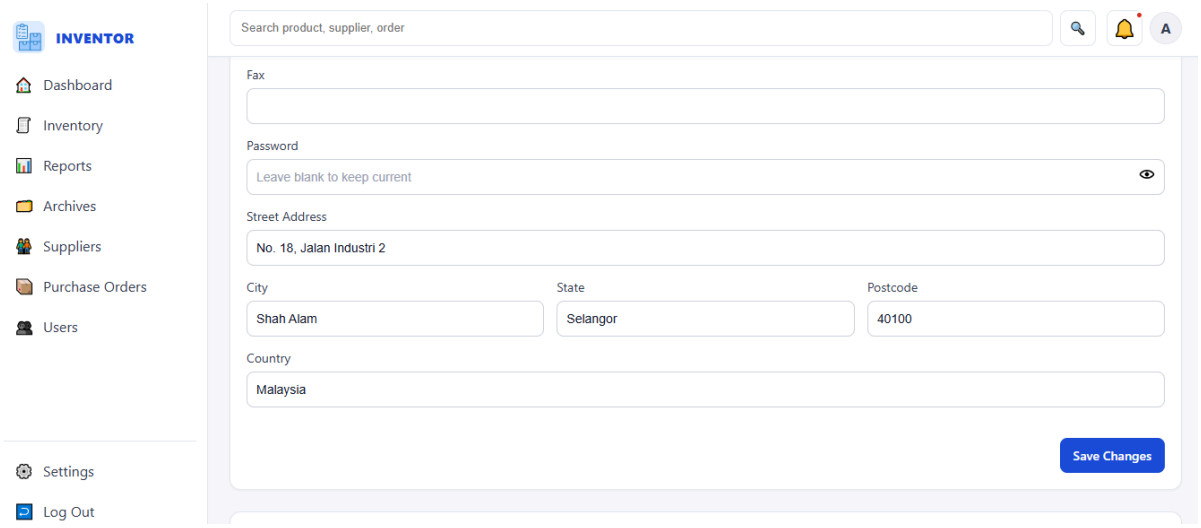


Figure 4.53: Edit Supplier Details Page UI 2

4.9.6 Supplier Portal Page

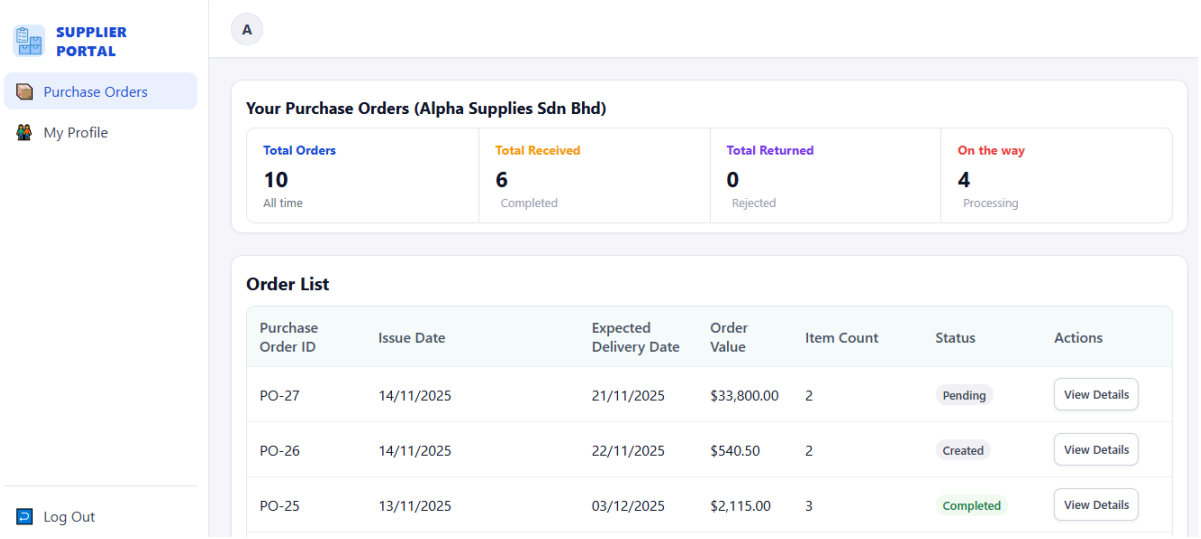


Figure 4.54: Supplier Portal PO Page UI

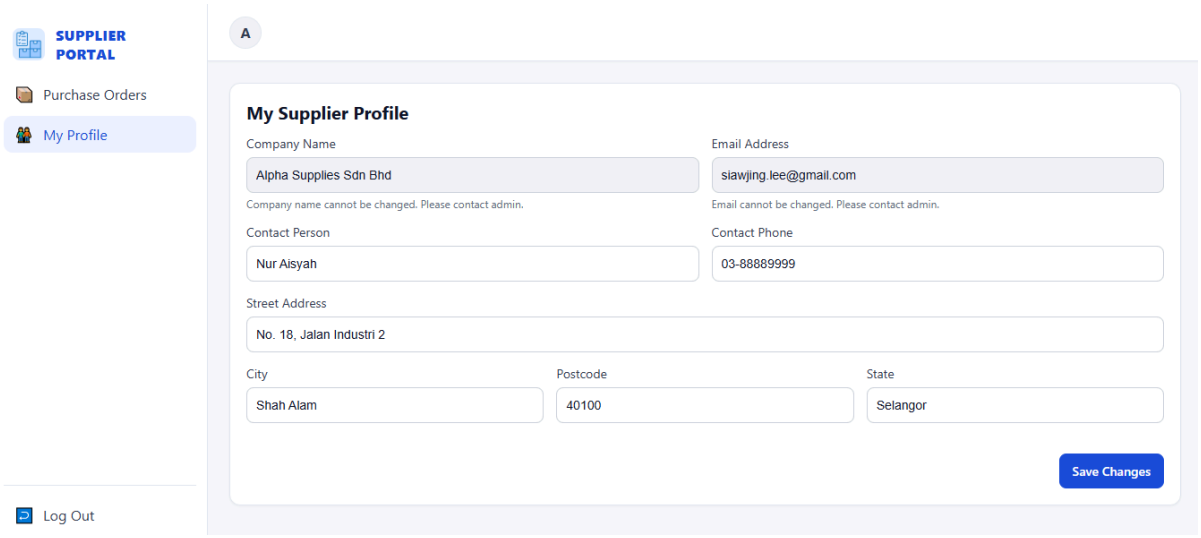


Figure 4.55: Supplier Portal Profile Page UI

4.9.7 Purchase Order List Page

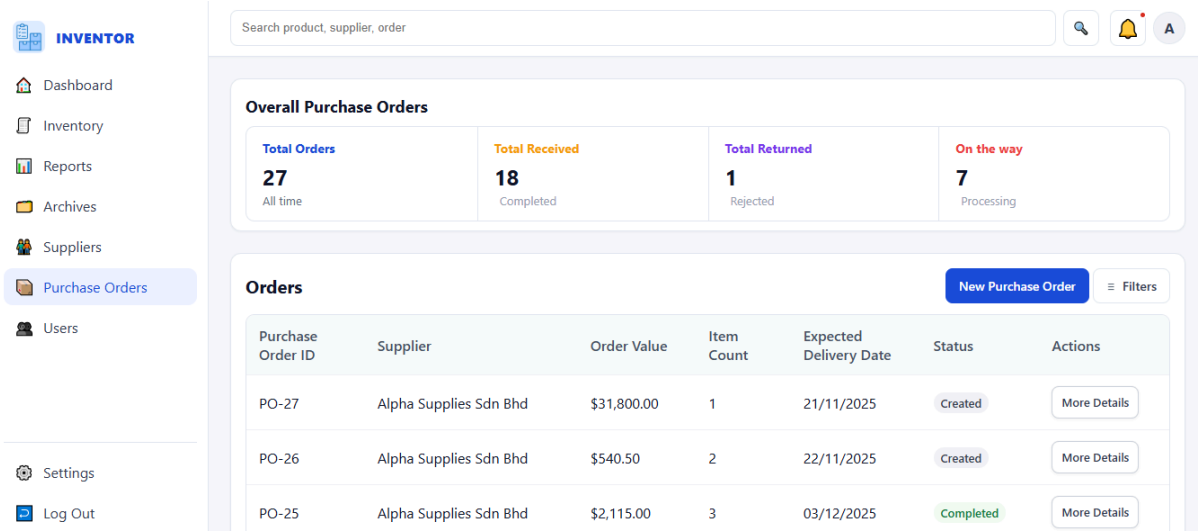


Figure 4.56: Purchase Order List Page UI

4.9.8 Purchase Order List Filter

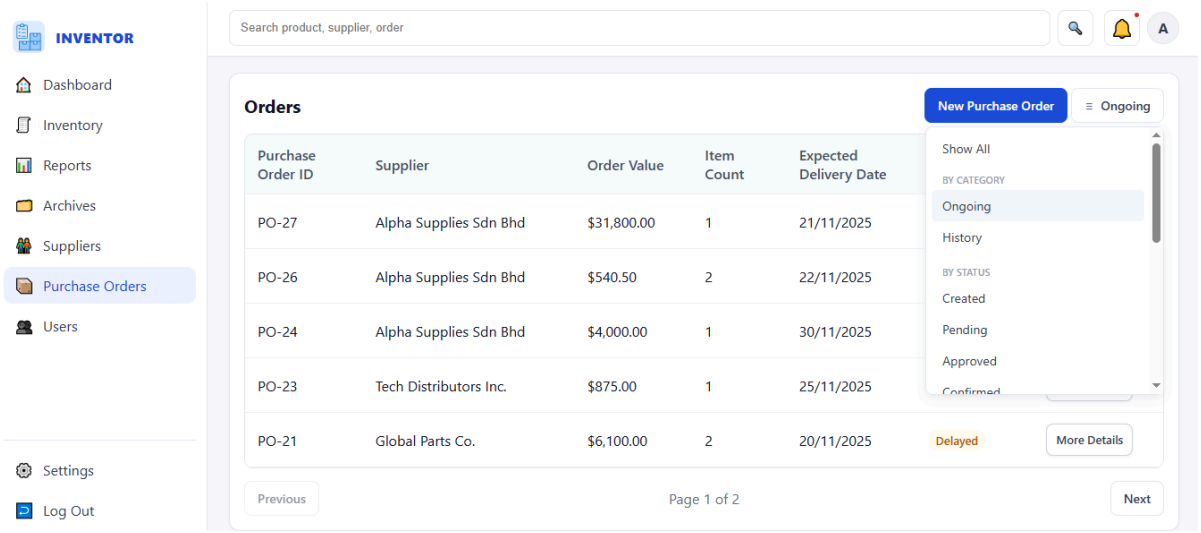


Figure 4.57: Purchase Order List Filter UI

4.9.9 Create Manual Purchase Order Page

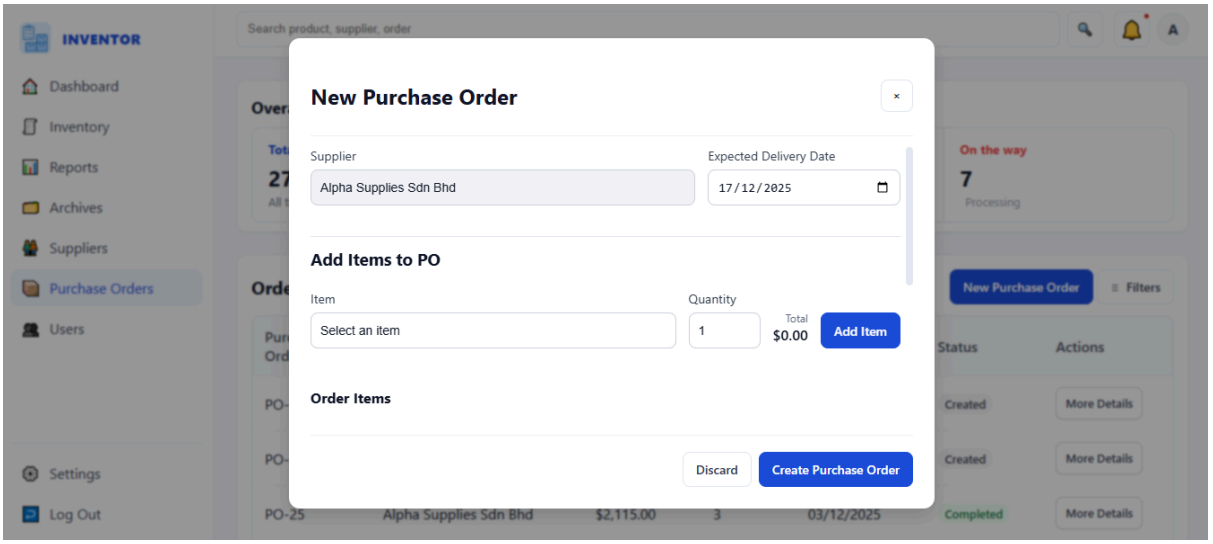


Figure 4.58: Create Manual Purchase Order Page UI 1

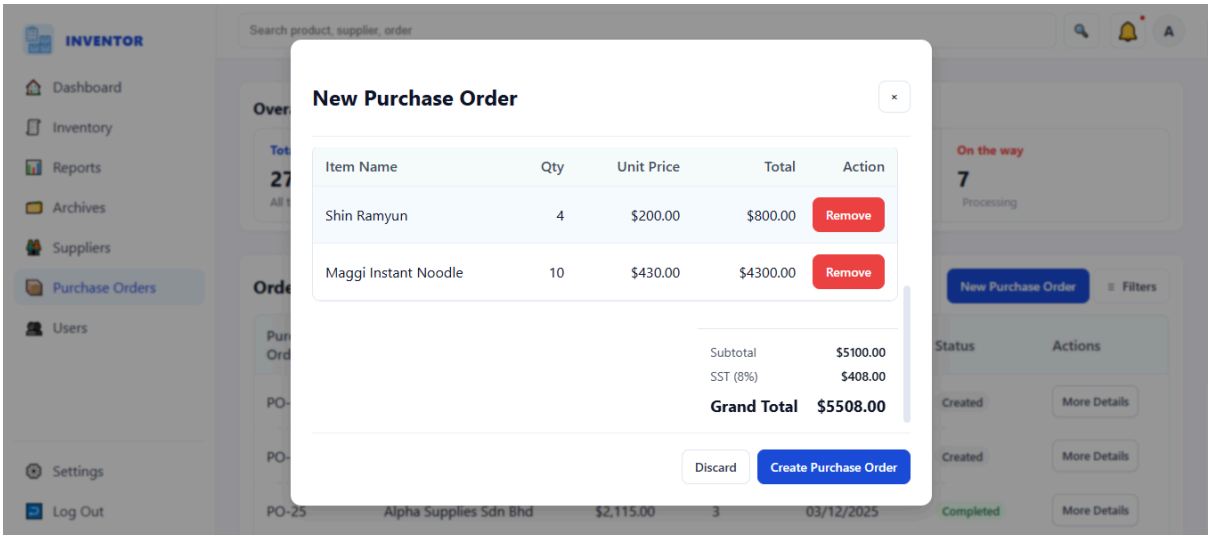


Figure 4.59: Create Manual Purchase Order Page UI 2

4.9.10 Purchase Order Details Page

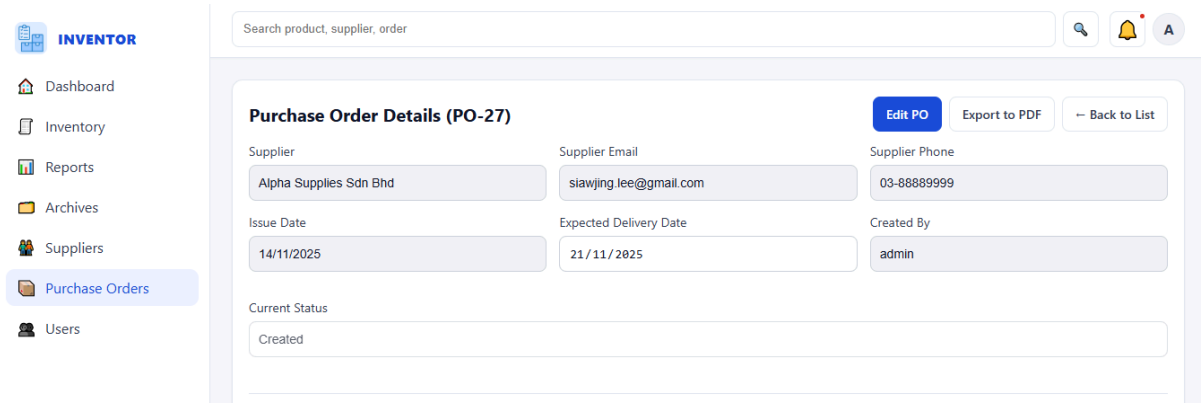


Figure 4.60: Purchase Order Details Page UI

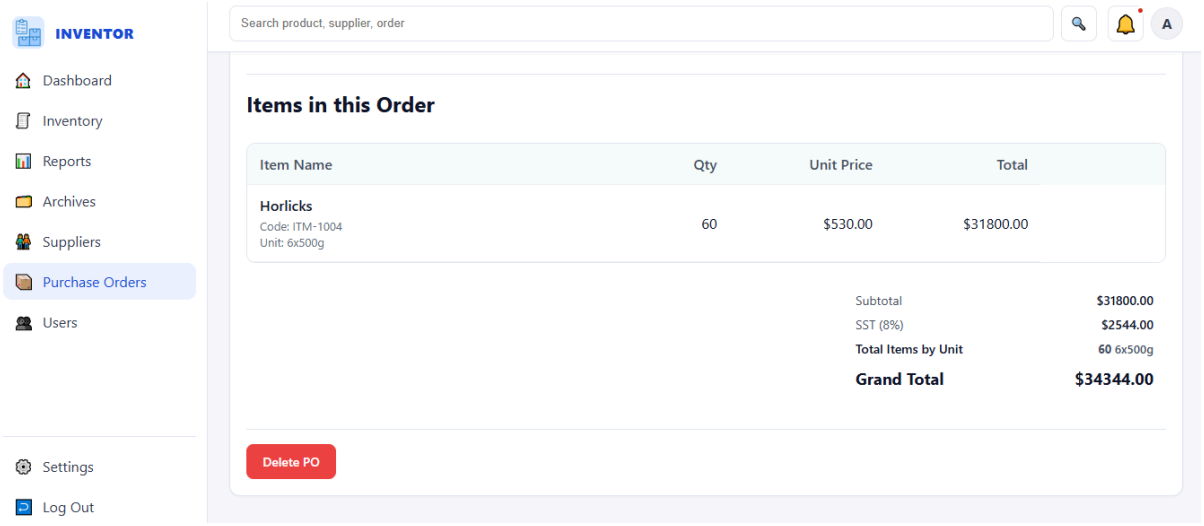


Figure 4.61: Purchase Order Details Page UI (Delete PO)

4.9.11 Edit Purchase Order Page

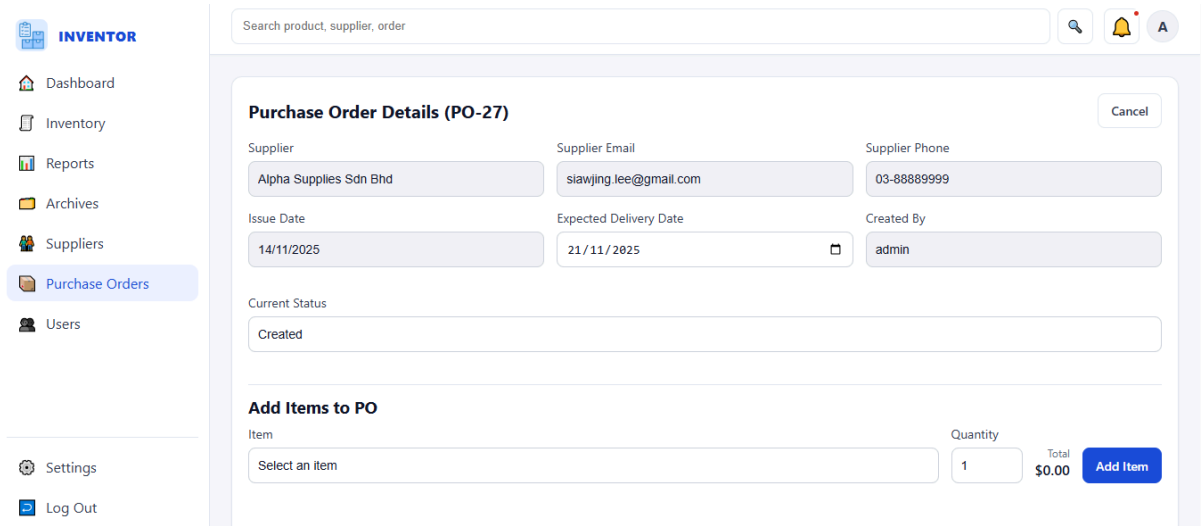


Figure 4.62: Purchase Order Details Page UI (Edit PO) 1

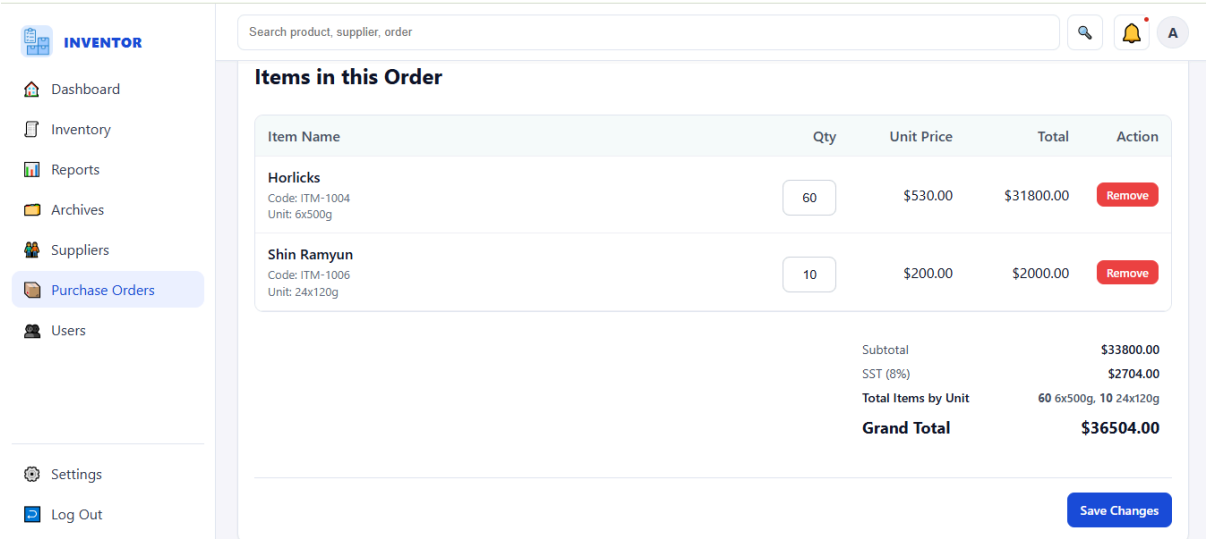
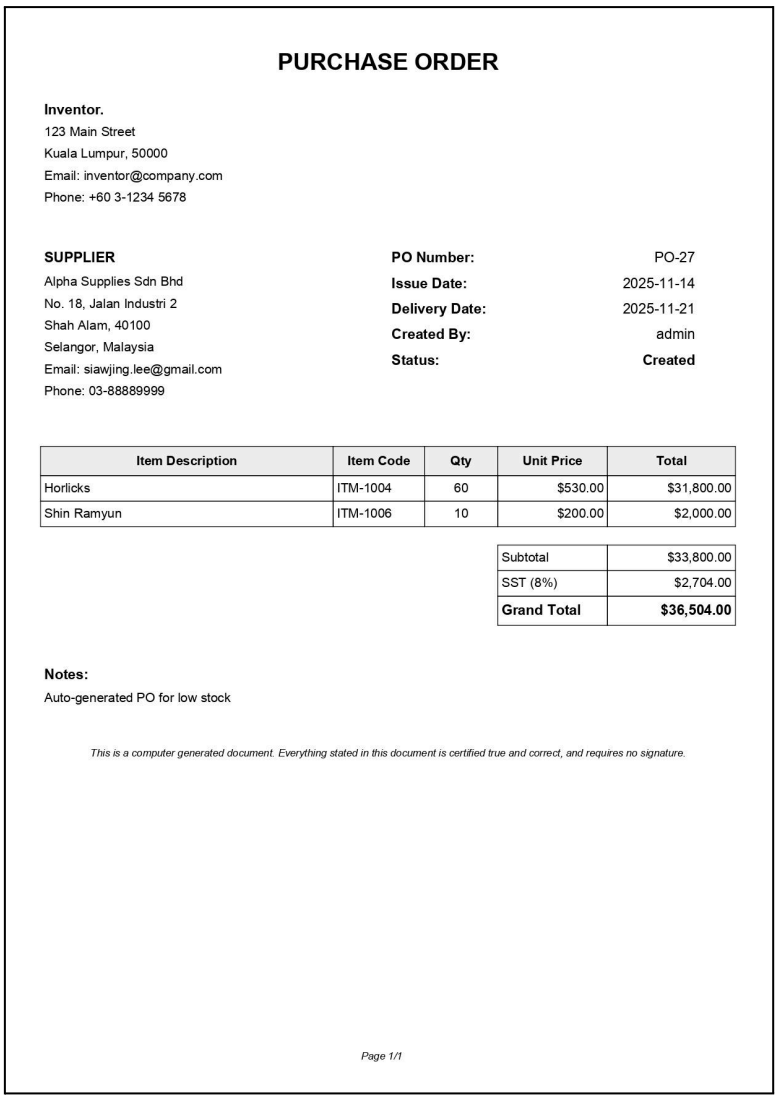


Figure 4.63: Purchase Order Details Page UI (Edit PO) 2

4.9.12 Export Purchase Order PDF



PURCHASE ORDER

Inventor.
123 Main Street
Kuala Lumpur, 50000
Email: inventor@company.com
Phone: +60 3-1234 5678

SUPPLIER
Alpha Supplies Sdn Bhd
No. 18, Jalan Industri 2
Shah Alam, 40100
Selangor, Malaysia
Email: siawjing.lee@gmail.com
Phone: 03-88889999

PO Number: PO-27
Issue Date: 2025-11-14
Delivery Date: 2025-11-21
Created By: admin
Status: Created

Item Description	Item Code	Qty	Unit Price	Total
Horlicks	ITM-1004	60	$530.00	$31,800.00
Shin Ramyun	ITM-1006	10	$200.00	$2,000.00

Subtotal	$33,800.00
SST (8%)	$2,704.00
Grand Total	**$36,504.00**

Notes:
Auto-generated PO for low stock

This is a computer generated document. Everything stated in this document is certified true and correct, and requires no signature.

Page 1/1

Figure 4.64: Purchase Order PDF Design

4.9.13 User List Page

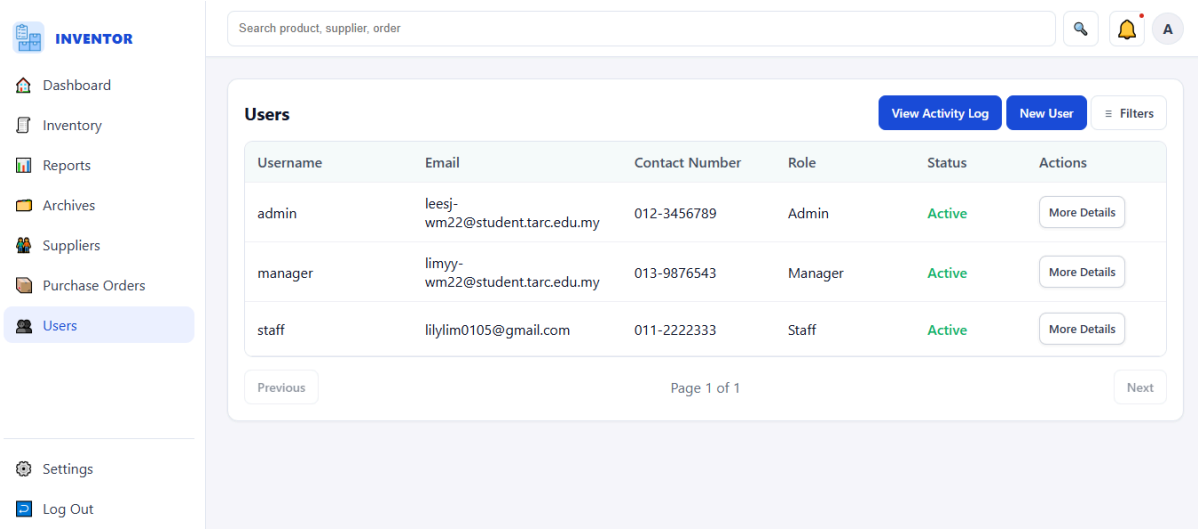


Figure 4.65: User List Page UI

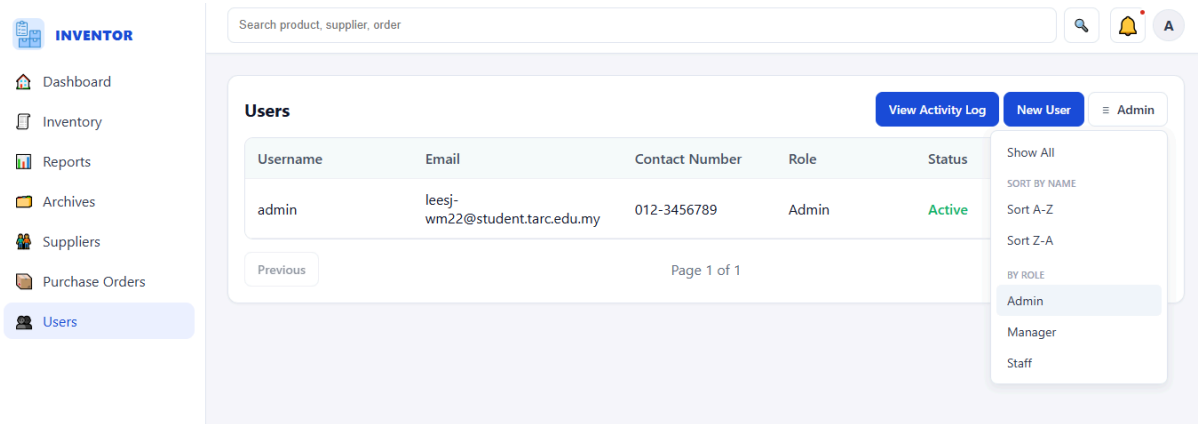


Figure 4.66: User List Filter UI

4.9.14 Add User Page

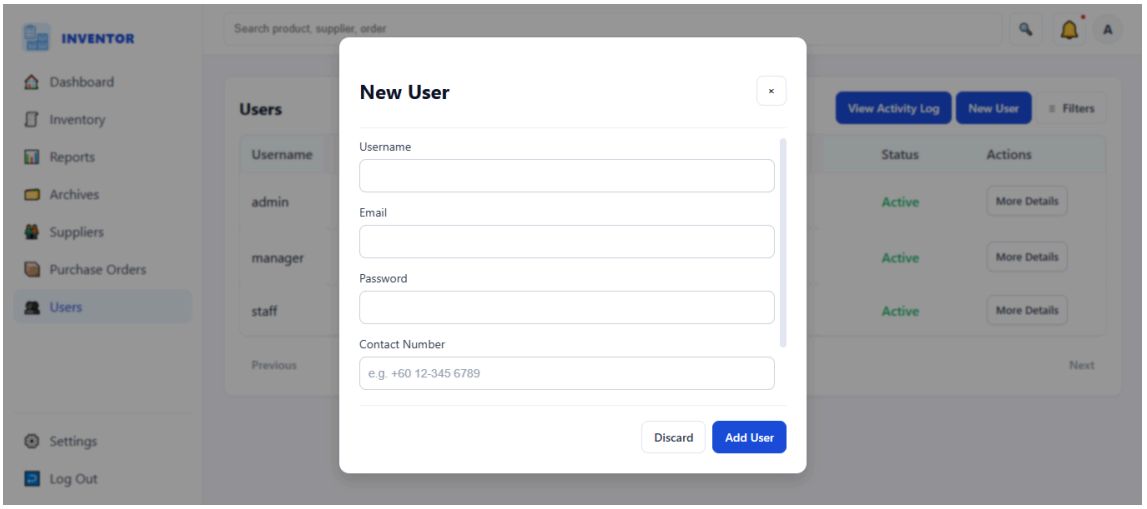


Figure 4.67: Add New User UI 1

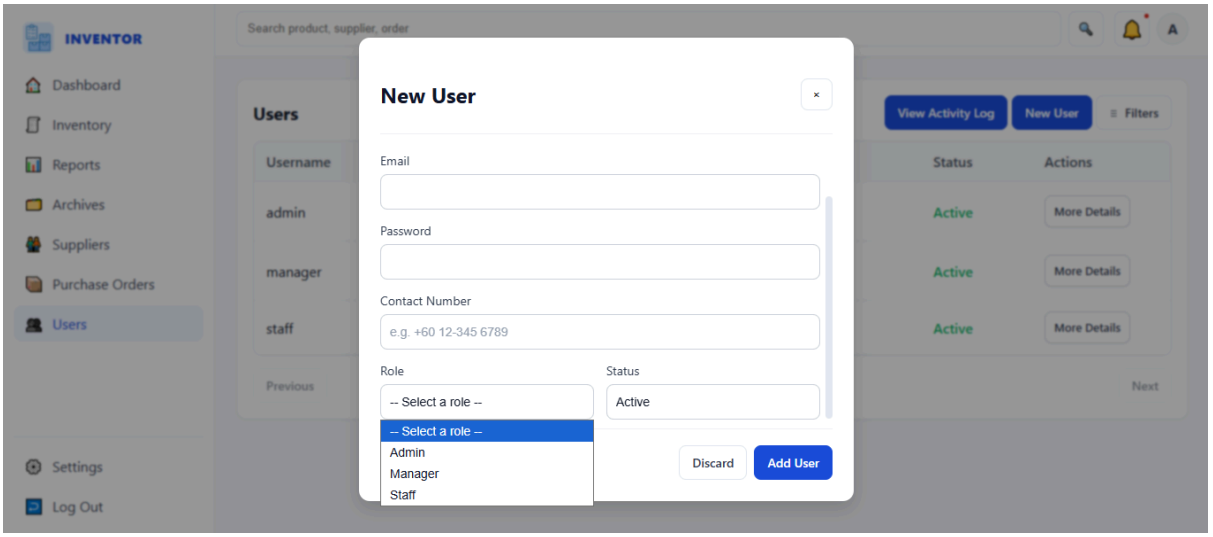


Figure 4.68: Add New User UI 2

4.9.15 User Details Page

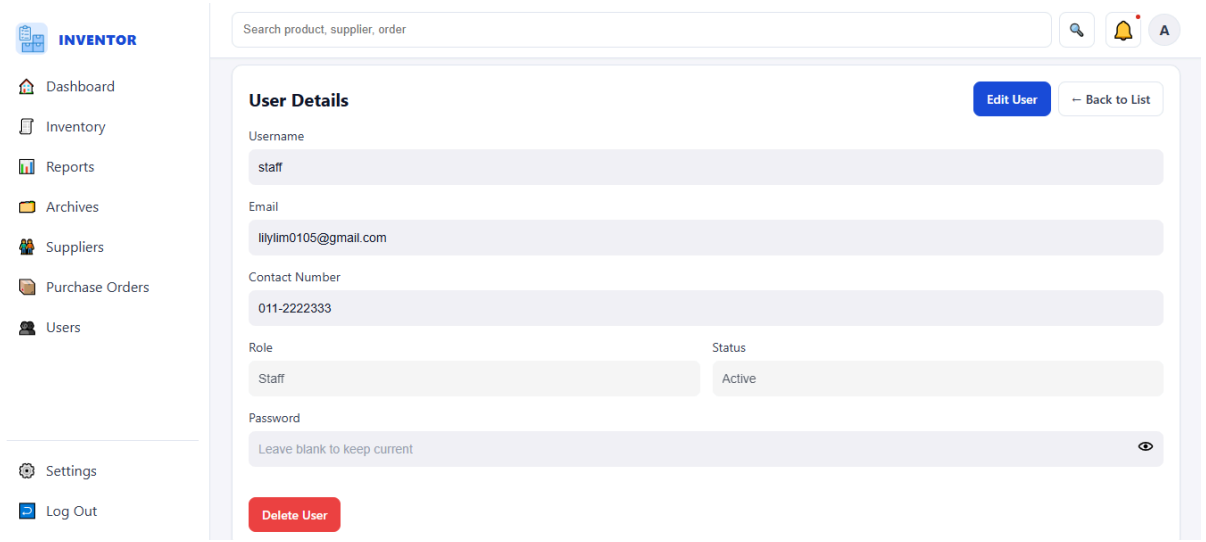


Figure 4.69: User Details Page UI (Delete Supplier)

4.9.16 Edit User Details Page

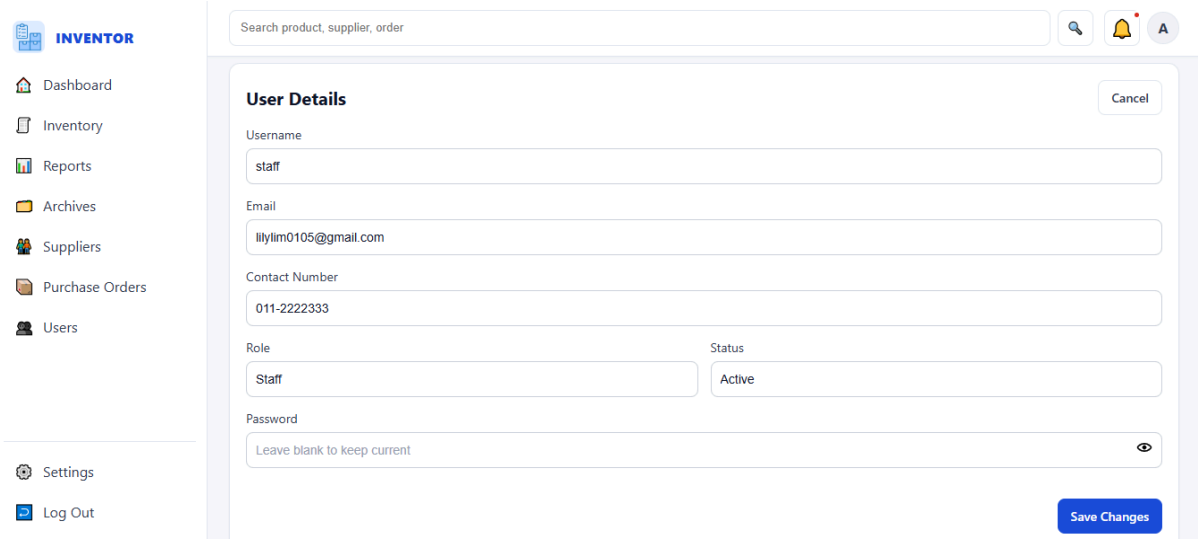


Figure 4.70: Edit User Details Page UI

4.9.17 Activity Log Page

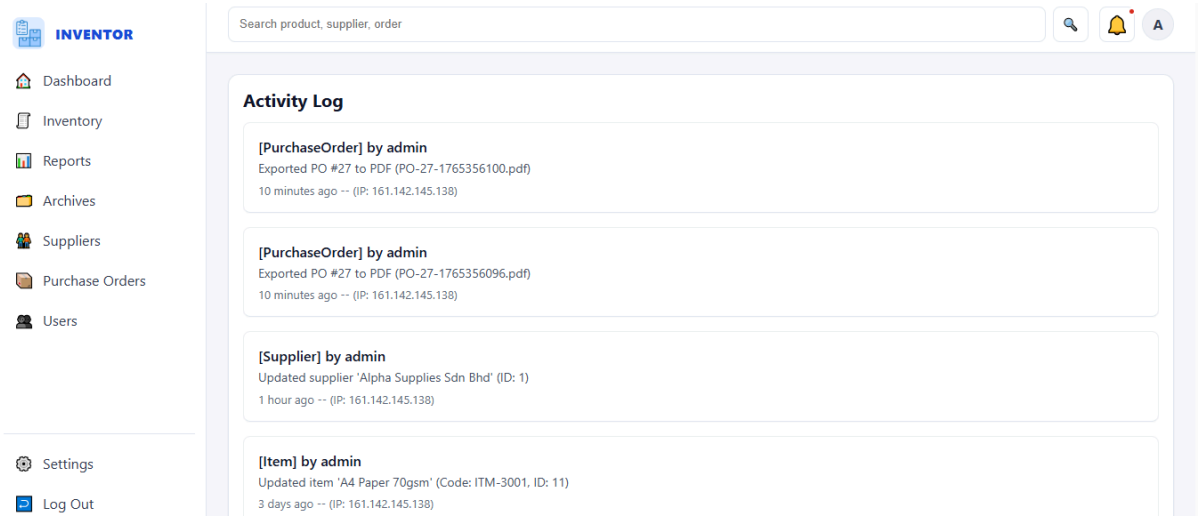


Figure 4.71: Activity Log Page UI

4.9.18 Notification Page

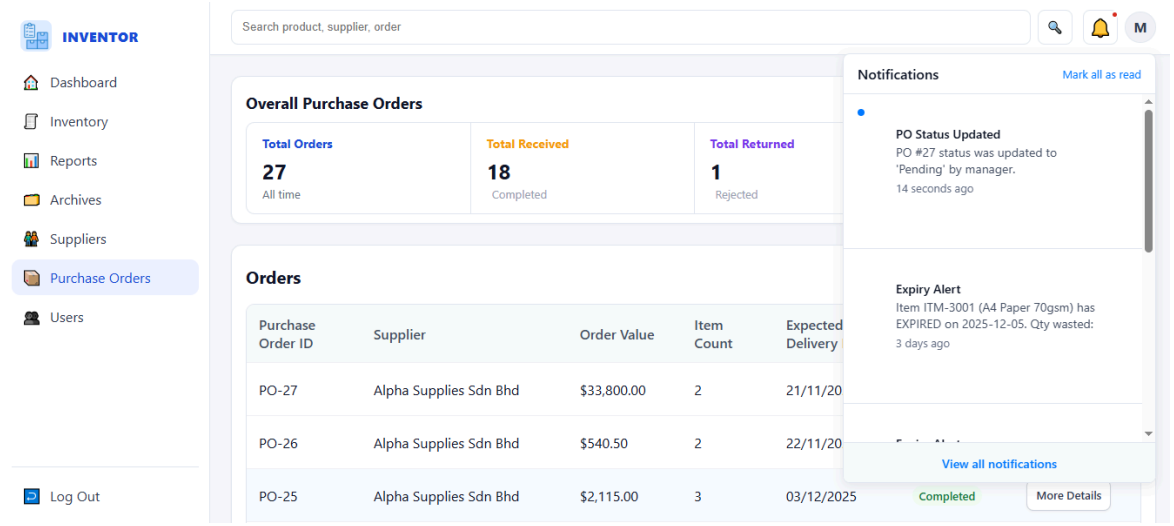


Figure 4.71: Notification Dropdown UI

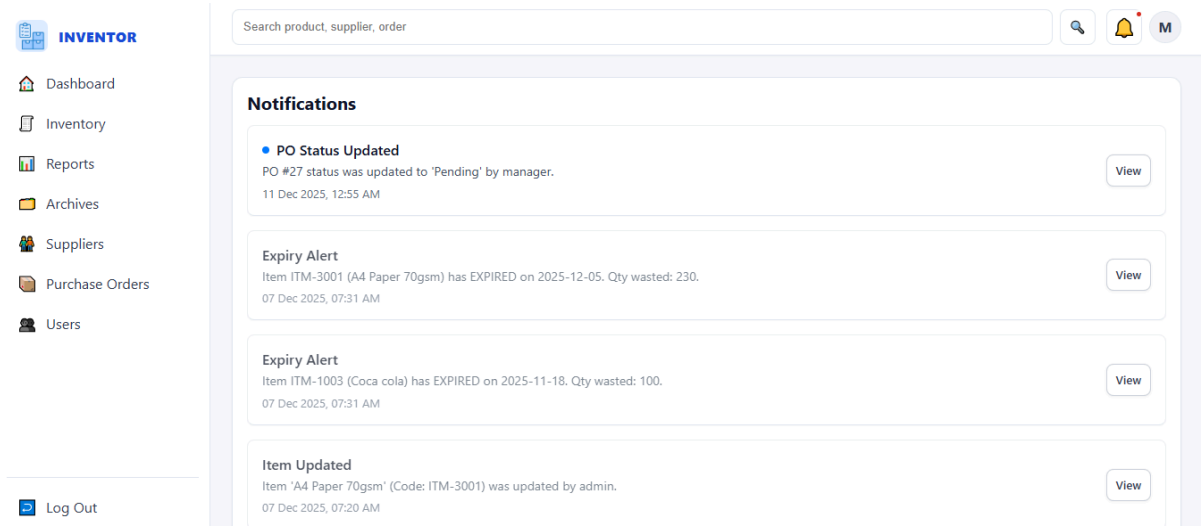


Figure 4.72: Notification Page UI

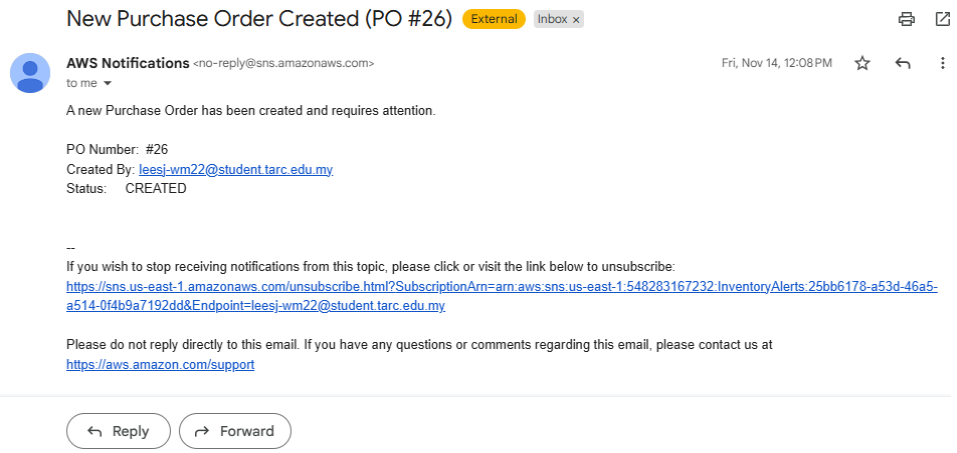


Figure 4.73: Email Notification Design

4.9.19 Company Setting Page

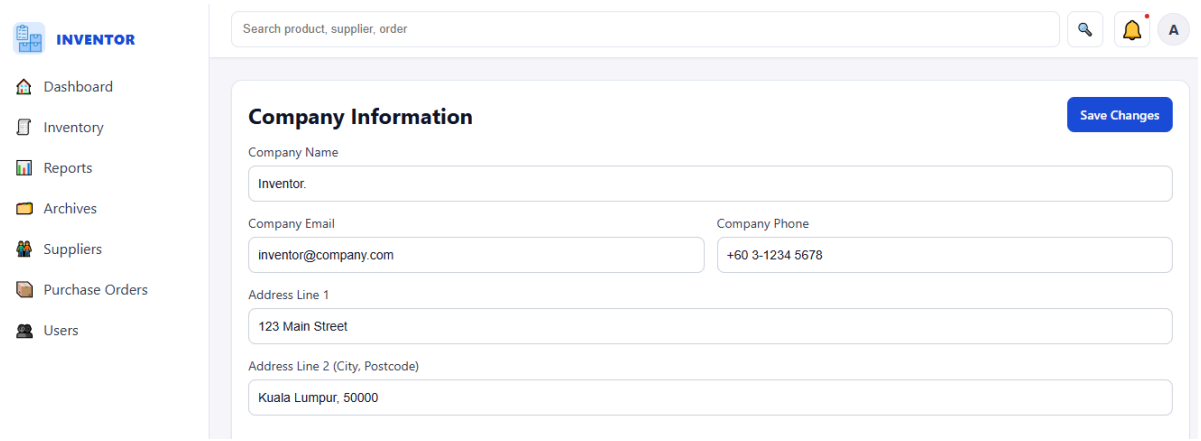


Figure 4.74: Company Settings Page UI 1

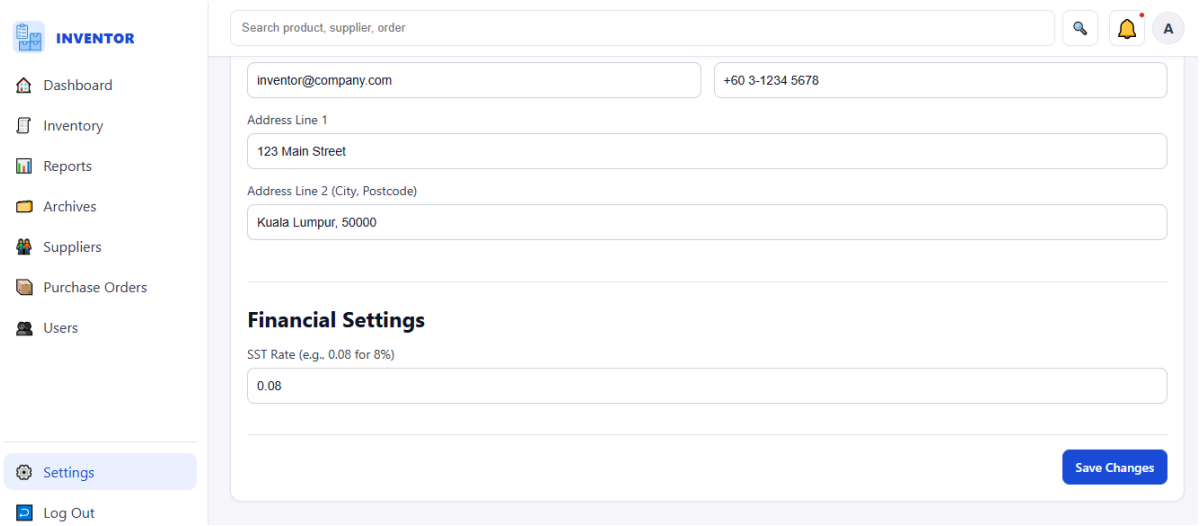


Figure 4.75: Company Settings Page UI 2

4.10 Chapter Summary and Evaluation

This chapter presents the design that was actually implemented, linking data structures, object design, process behaviour, and user experience into a coherent whole. The ERD, class diagram, and data dictionary establish a clear and consistent schema across core tables (User, Supplier, Item, PurchaseOrder, PurchaseOrderDetails, GoodsReceipt, GoodsReceiptDetails, StockAlert, Notification, ExportHistory, ActivityLog). The overall and detailed use case diagrams define system responsibilities and actor interactions, while the activity and sequence diagrams translate those responsibilities into step-by-step flows that mirror the final method names used in implementation. The layered software architecture clarifies separation of concerns between presentation, controller, and data/domain, and the UI designs confirm that the specified behaviours were realised in practical screens.

Overall, the design is consistent, traceable, and suitable for the project scope. Strengths include a clean data model, alignment between diagrams and code, and complete coverage of core workflows from reordering to supplier and user management, goods receipt, and notifications. For future improvement, the project could refine exception paths (e.g., partial deliveries/returns), strengthen validation and idempotency for critical submissions, and expand reporting and analytics as more operational data becomes available.

Chapter 5

Implementation and Testing

5 Implementation and Testing

This chapter details the technical implementation and validation of the Business Process Automation System for Inventory Management. It begins by outlining the setup of the cloud infrastructure on Amazon Web Services (AWS), which serves as the backbone for the serverless architecture. Subsequently, it describes the implementation of core modules, focusing on API development, authorization mechanisms, and business logic, followed immediately by the testing results for each module to valid functionality.

5.1 Cloud Infrastructure and Environment Setup

The system's infrastructure is built entirely on the AWS Cloud to ensure scalability, security, and high availability. The environment setup involves configuring the relational database, serverless computing functions, notification services, and monitoring dashboards. The following subsections detail the configuration of these services.

5.1.1 Cloud Architecture Diagram

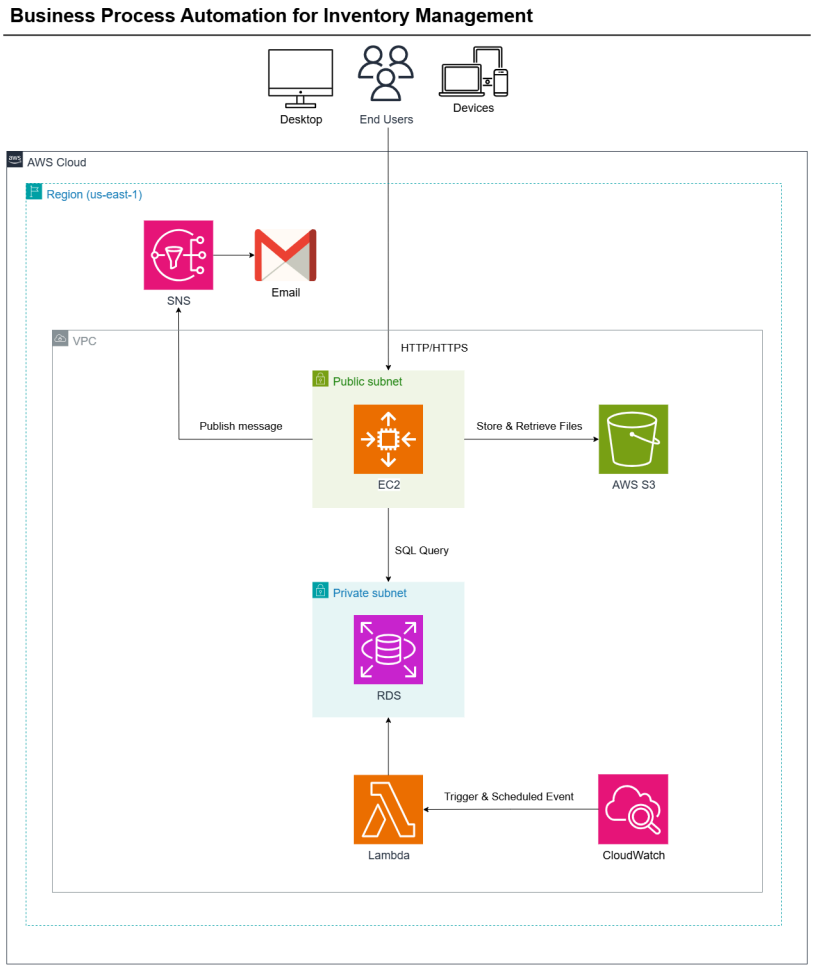


Figure 5.1.1: Cloud Architecture Diagram

5.1.2 Database Configuration (AWS RDS)

This subsection explains the process of creating the cloud database using Amazon RDS and configuring MySQL Workbench to connect to the instance. The database stores all inventory-related records such as users, suppliers, items, stock levels, purchase orders, and activity logs.

(1) Creating the RDS MySQL Instance

The first step was to provision a managed database instance through Amazon Relational Database Service (RDS). The MySQL engine was selected because it is lightweight, widely supported, and suitable for web-based systems.

A. Choosing Database Engine

- Engine type: MySQL
- Version: Default version provided by AWS.
    - This ensures compatibility with PHP and MySQL Workbench.

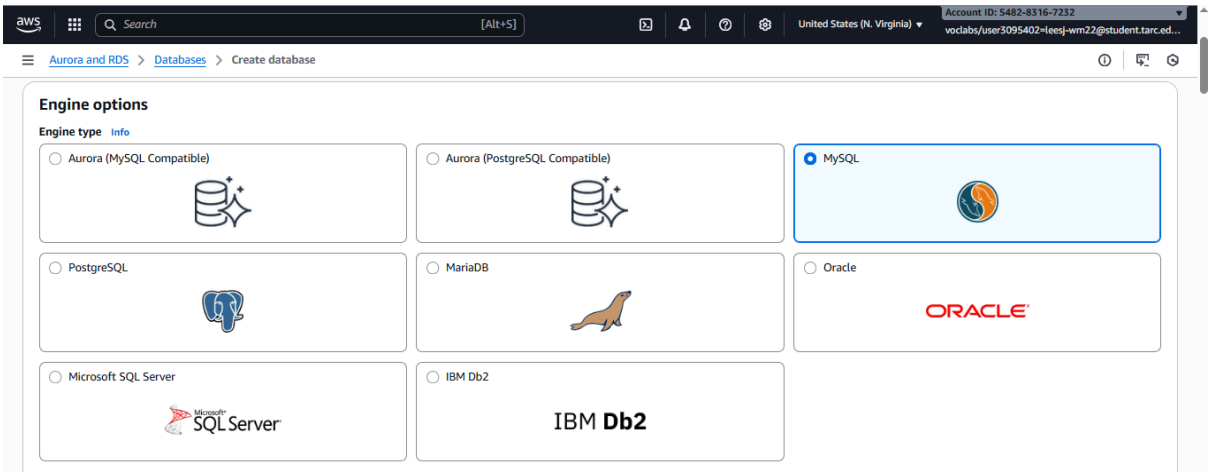


Figure 5.1.2.1: Choosing Database Engine

B. DB Instance Settings

- DB Instance Identifier: database-1
- Master Username: admin
- Master Password: 12345678 (self-managed)
- Credential Management: Self-managed
    - These credentials allow the system to authenticate from EC2 and MySQL Workbench.

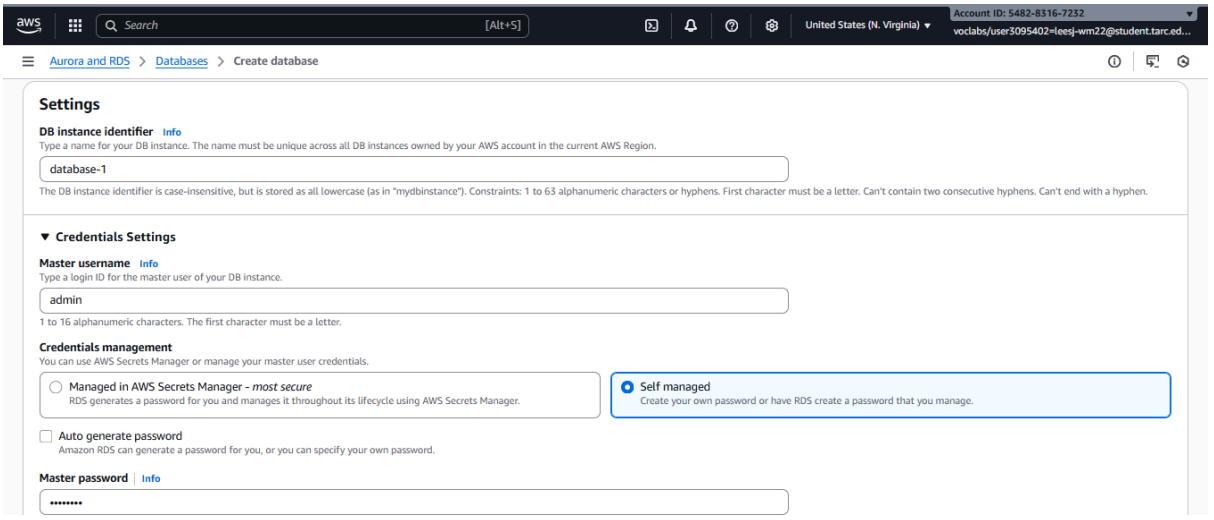


Figure 5.1.2.2: DB Instance Settings

C. Instance Class

- DB Instance Class: db.t4g.micro

  This class is cost-effective and suitable for student projects while still delivering adequate performance for CRUD operations.

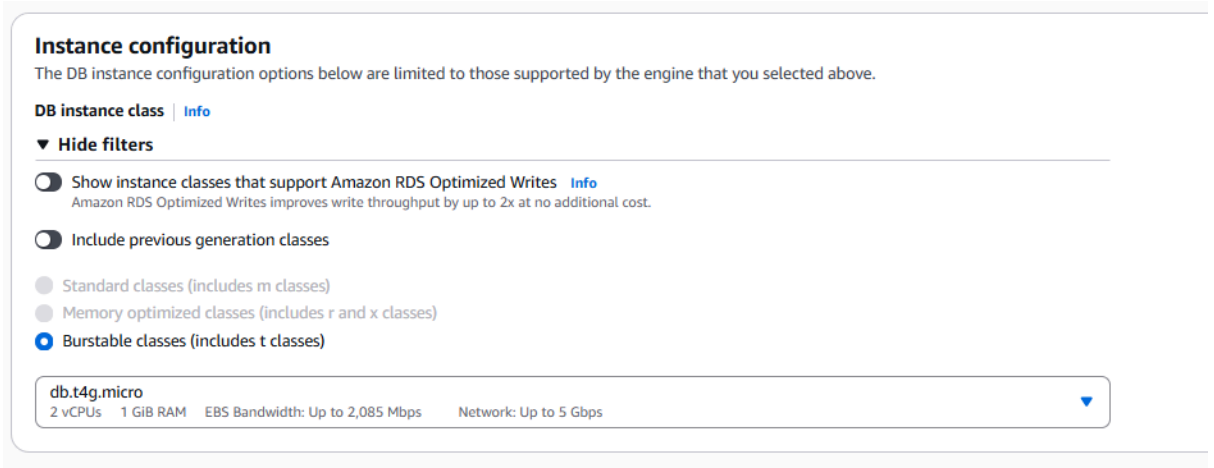


Figure 5.1.2.3: DB Instance Class

D. Storage Settings

- Storage Type: General Purpose SSD (gp2)
- Allocated Storage: 20 GB
- Storage Autoscaling: Enabled
    - Autoscaling ensures the database can automatically increase storage capacity when usage grows.

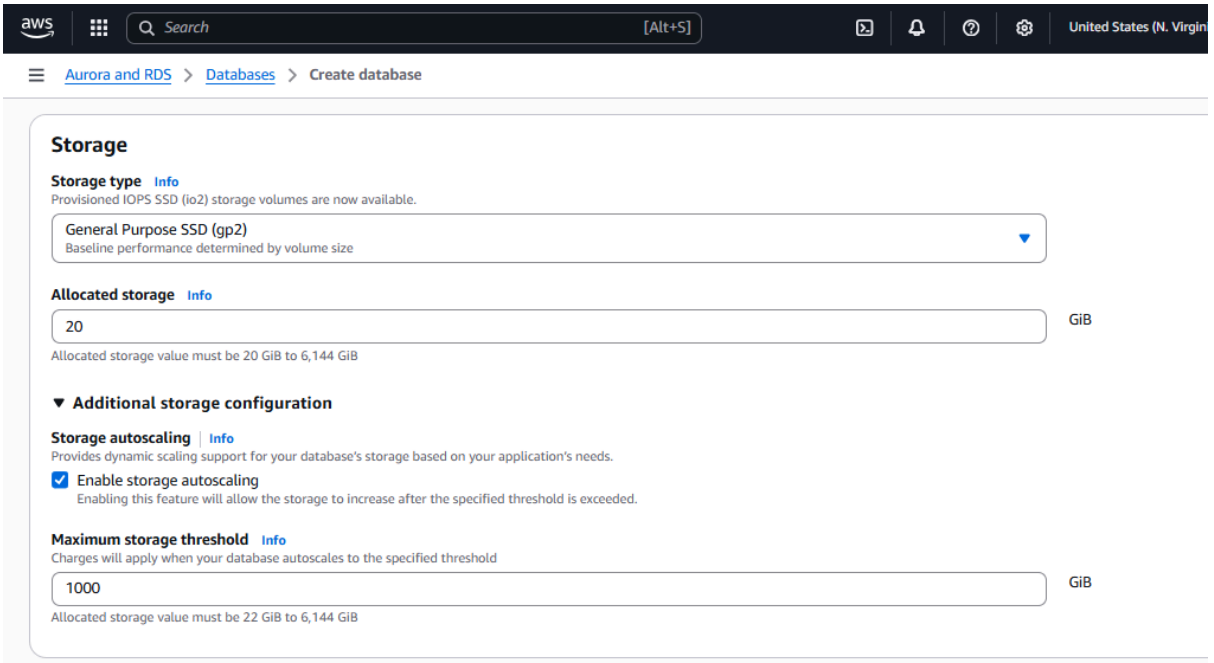


Figure 5.1.2.4: Storage Settings

E. Connectivity and Authentication

- VPC: Default AWS VPC
- Public Access: Enabled to allow connection from MySQL Workbench
- Database Authentication: “Password and IAM Authentication”
    - Password authentication is used for the project, while IAM remains available for future enhancements.

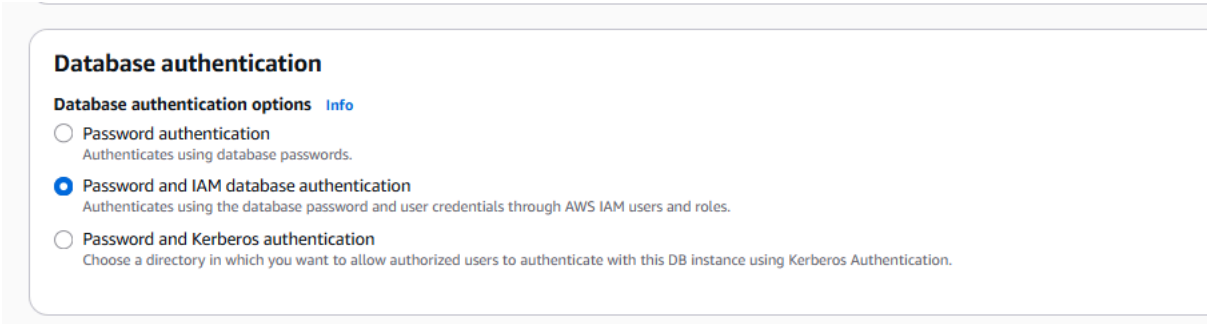


Figure 5.1.2.5: Database Authentication

F. Completion

After reviewing all settings, the RDS instance was launched. AWS automatically assigned an endpoint: database-1.cxjo8lexejtp.us-east-1.rds.amazonaws.com. This endpoint is used by the EC2 application and MySQL Workbench to connect to the cloud database.

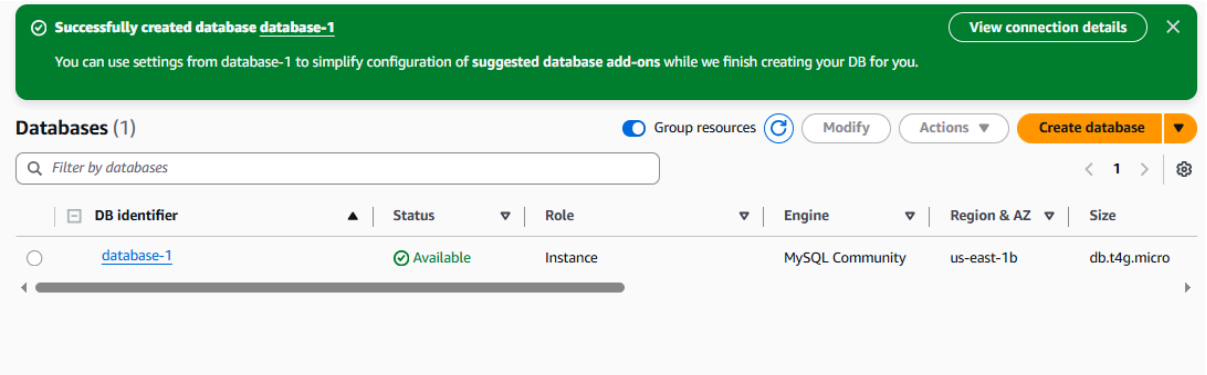


Figure 5.1.2.6: Successfully Created Database

(2) Configuring MySQL Workbench

Once the RDS instance became available, MySQL Workbench was installed locally to manage the database schema.

A. Installing Workbench

- Setup Type: Client Only
  - This installs only the client software needed to connect to external MySQL servers.

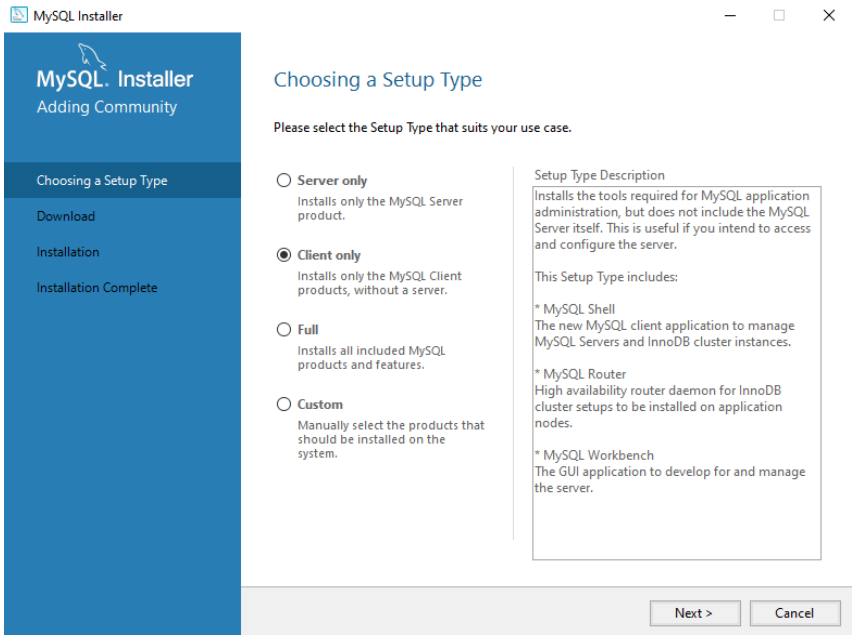


Figure 5.1.2.7: Configuring MySQL Workbench

B. Creating a New Connection

A new connection was configured using the RDS endpoint, as shown in the screenshot.

Connection settings:

- Hostname: database-1.cxjo8lexejtp.us-east-1.rds.amazonaws.com
- Port: 3306
- Username: admin
- Password: Stored in vault
- Connection Method: Standard TCP/IP

After entering the details, the "Test Connection" button was clicked to verify successful communication with the cloud database.

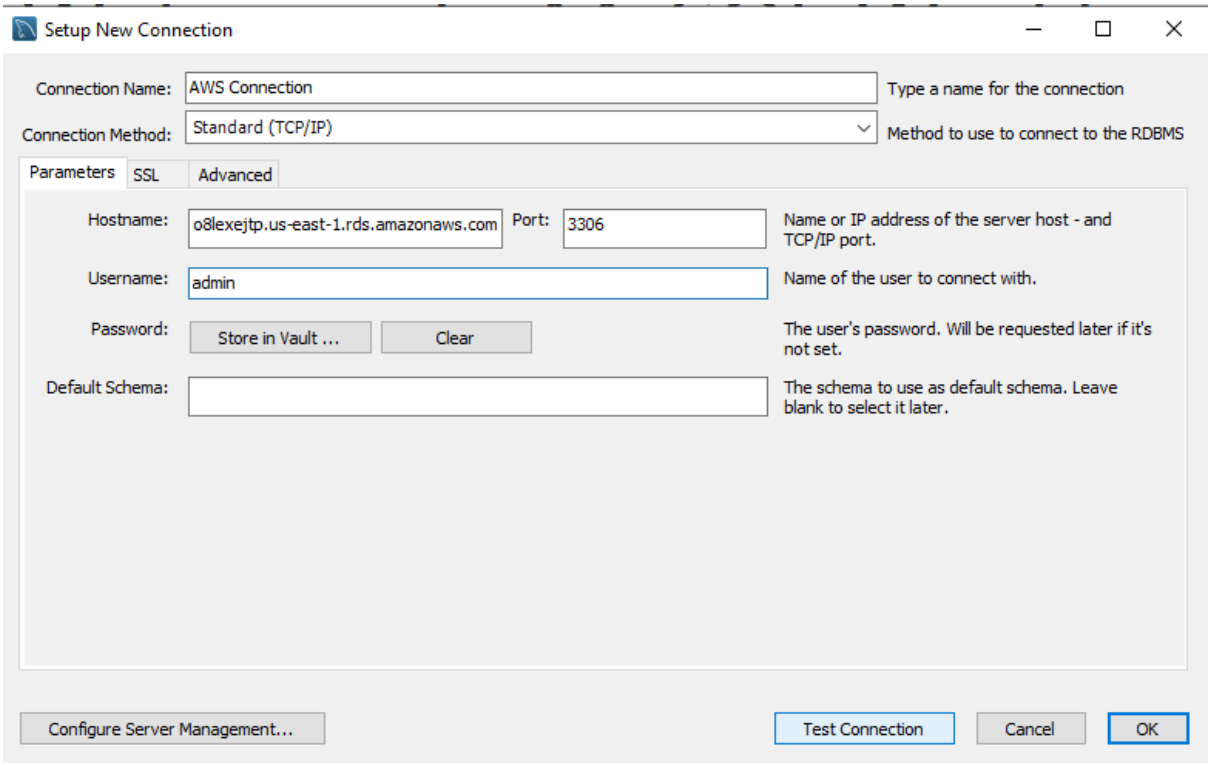


Figure 5.1.2.8: Setup New Connection

(3) Creating Database Tables and Entities

When the connection was successful, the SQL script for the inventory system was executed inside MySQL Workbench.

The SQL script includes:

- Creating the main database inventory_db
- Creating all required tables:

  user, supplier, category, item, purchase_order, purchase_order_details, goods_receipt,

goods_receipt_details, stock_alert, notification, export_history, activity_log, company_settings.

- Inserting sample records for testing
- Defining foreign keys, indexes, and constraints

This ensures that the cloud database structure matches the requirements of the Business Process Automation System.

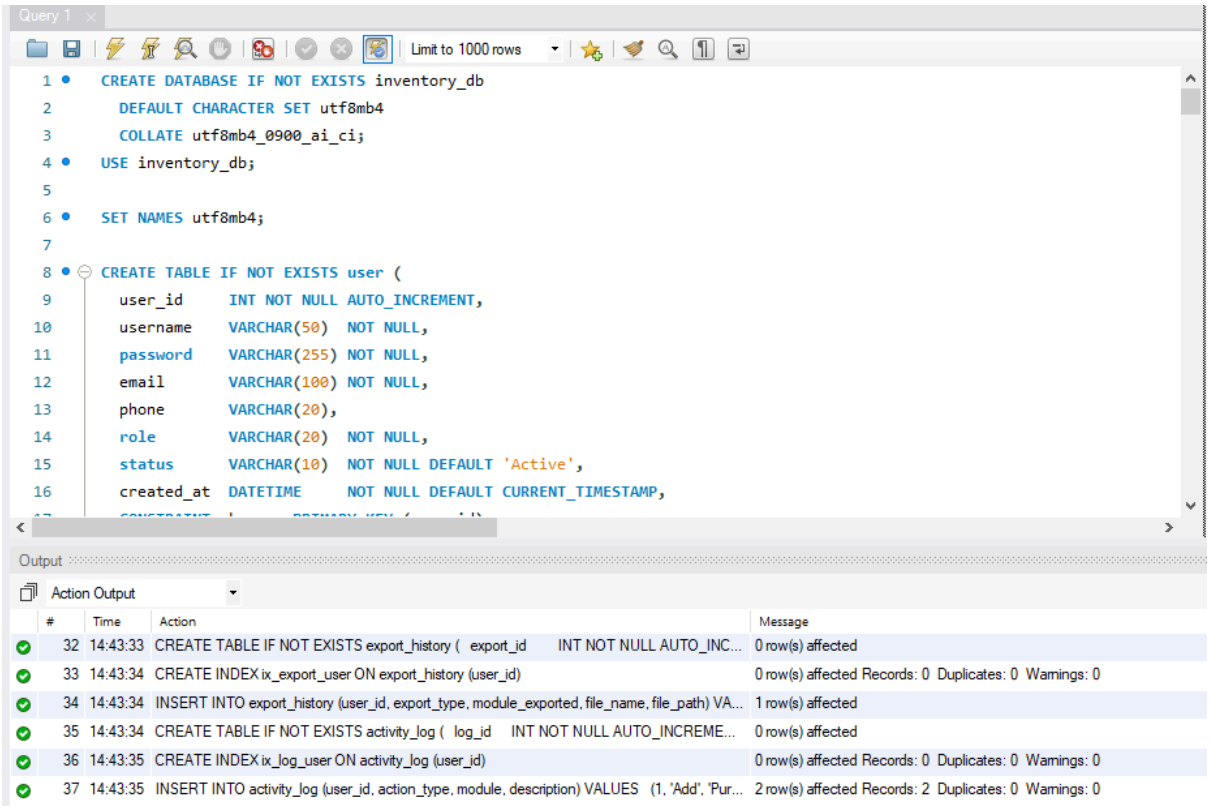


Figure 5.1.2.9: Creating Database Tables and Entities

5.1.3 Serverless Backend Setup (AWS Lambda)

The Lambda function is responsible for automatically generating purchase orders whenever an item's stock level falls below its threshold. The function connects securely to the RDS database, processes low-stock items, and inserts new purchase orders and notifications into the system. The configuration steps are detailed below.

(1) Creating the Lambda Function

The function AutoPO_Generator was created using the AWS Lambda service. Python was selected as the runtime because it integrates well with MySQL through external libraries such as PyMySQL.

A. Basic Function Setup

- Function Name: AutoPO_Generator
- Runtime: Python 3.13
- Execution Role: LabRole

This role allows the function to access CloudWatch Logs and interact with RDS within the VPC.

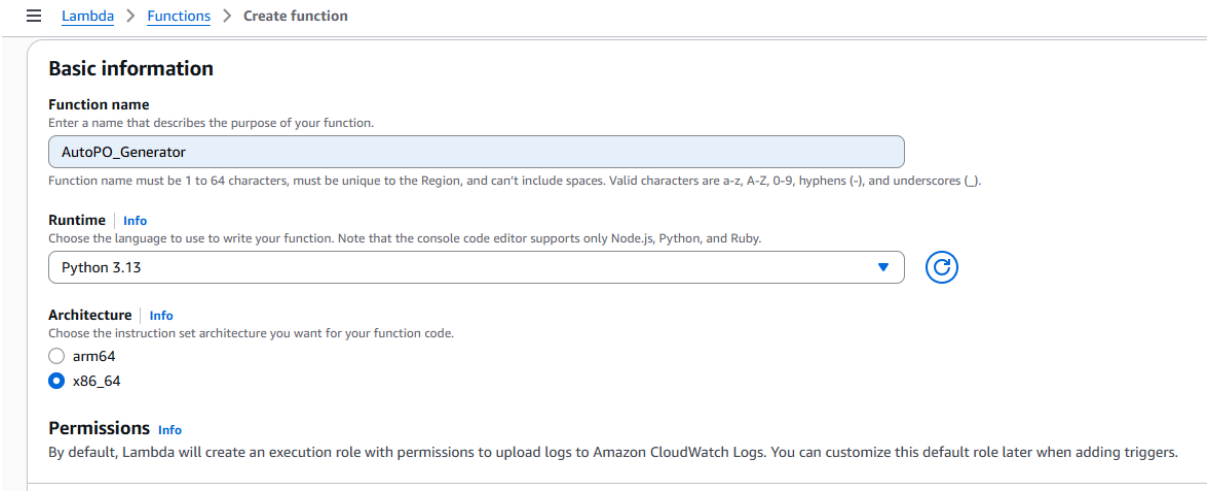


Figure 5.1.3.1: Lambda Basic Function Setup 1

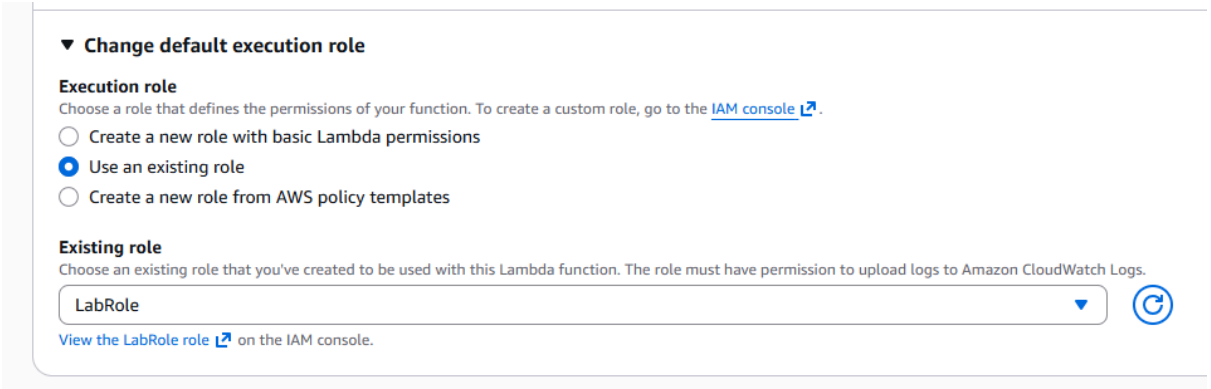


Figure 5.1.3.2: Lambda Basic Function Setup 2

B. VPC Configuration

To enable database connectivity, VPC access was configured:

- VPC: Lab VPC
- Subnets: At least two subnets selected
- Security Group: Default security group

This ensures the Lambda function can communicate privately with the RDS instance inside the VPC.

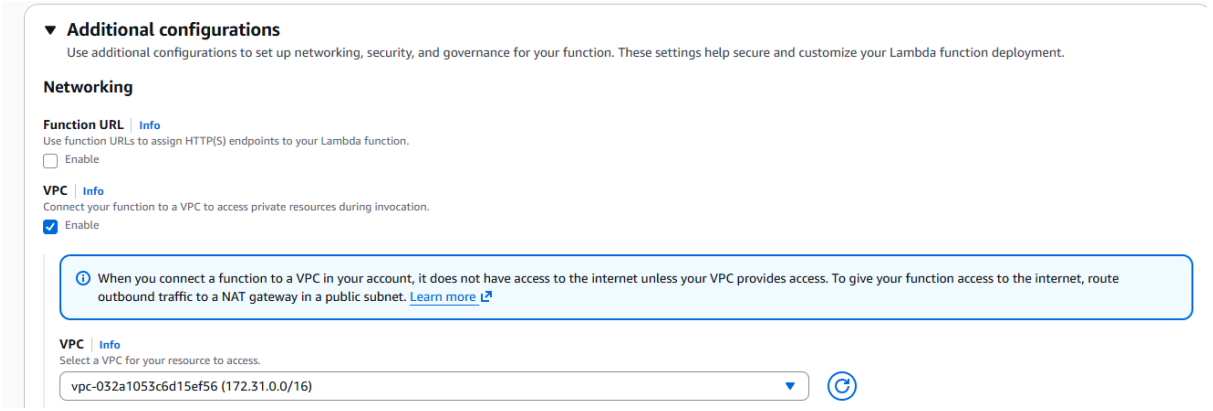


Figure 5.1.3.3: VPC Configuration

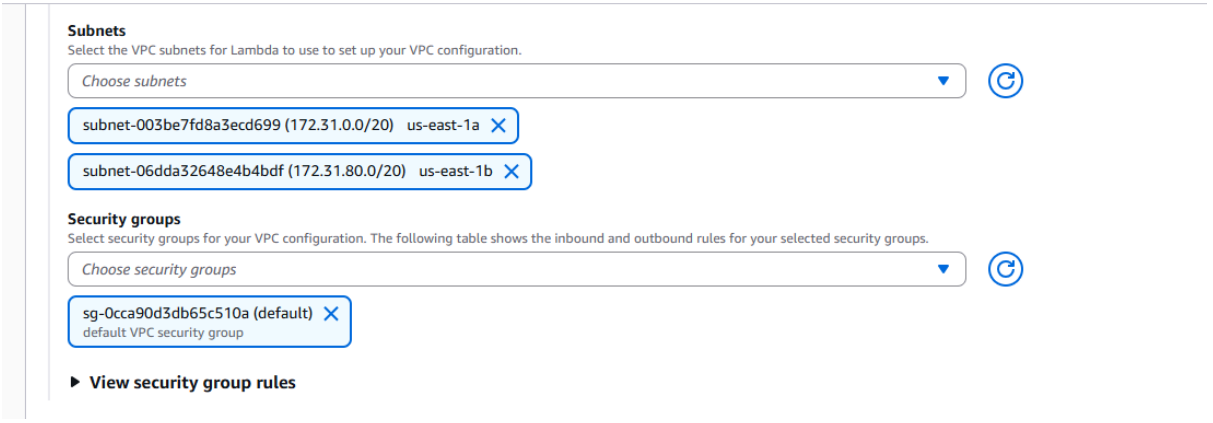


Figure 5.1.3.4: Subnet & Security Group Configuration

(2) Preparing Environment Variables

After creating the function, environment variables were added to store database credentials securely.

A. Adding Required Variables

- DB_HOST: http://database-1.cxjo8lexejtp.us-east-1.rds.amazonaws.com
- DB_USER: admin
- DB_PASS: 12345678
- DB_NAME: inventory_db

These variables allow the Python code to access credentials without hardcoding them.

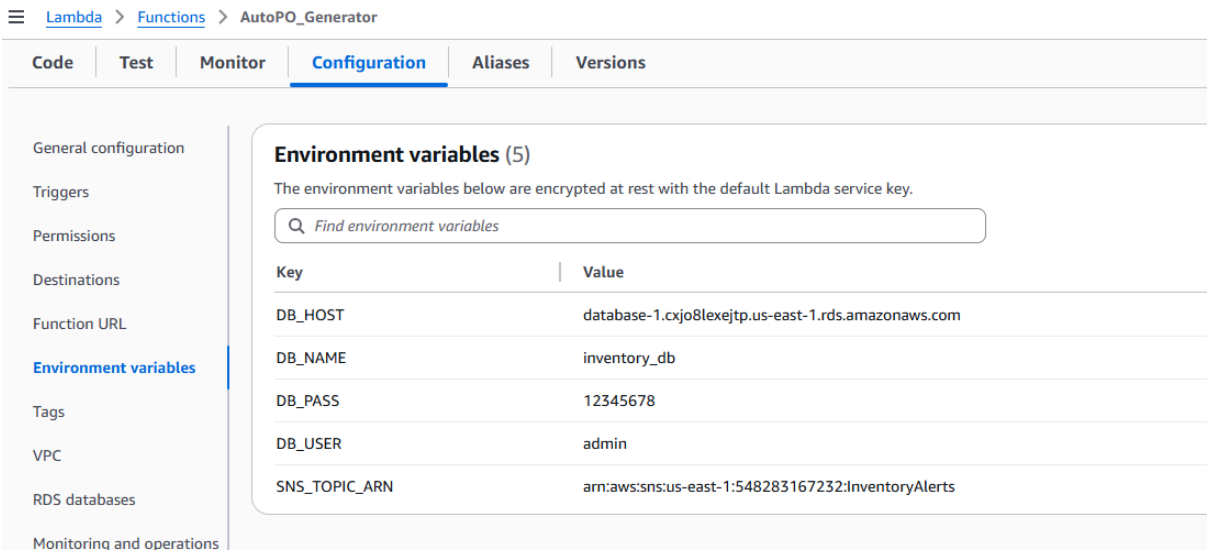


Figure 5.1.3.5: Adding Environment Variables

(3) Preparing the PyMySQL Lambda Layer

Because Lambda does not include the PyMySQL library by default, a custom layer was created.

A. Installing Python and Dependencies

- Installed Python 3.13 (64-bit) with Add to PATH enabled.
- Verified installation using: python -m pip --version

B. Creating the Layer Folder

A folder was created for the PyMySQL library:

python -m pip install pymysql -t ./python

C. Packaging the Layer

The folder was compressed into a ZIP file:

Compress-Archive -Path .\python -DestinationPath pymysql-layer.zip -Force

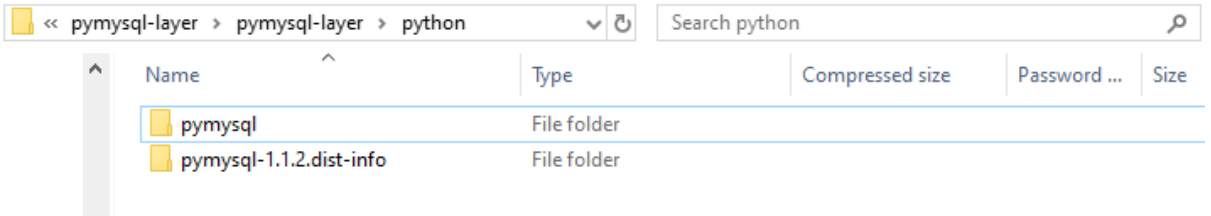


Figure 5.1.3.6: Folder Inside pymysql-layer.zip

(4) Uploading and Attaching the Layer

A. Creating the Layer in AWS

- Layer Name: pymysql-layer
- Upload File: pymysql-layer.zip
- Runtime: Python 3.13

Figure 5.1.3.7: Layer Configuration 1

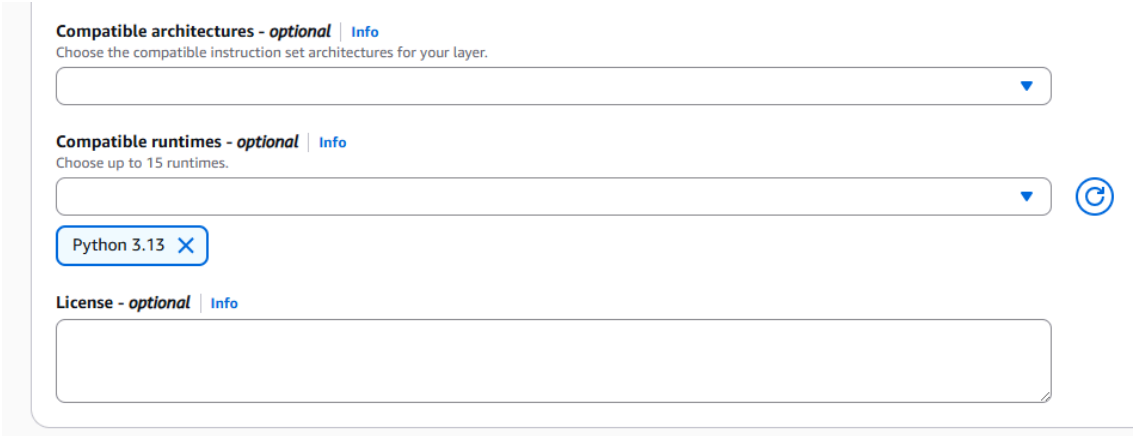


Figure 5.1.3.8: Layer Configuration 2

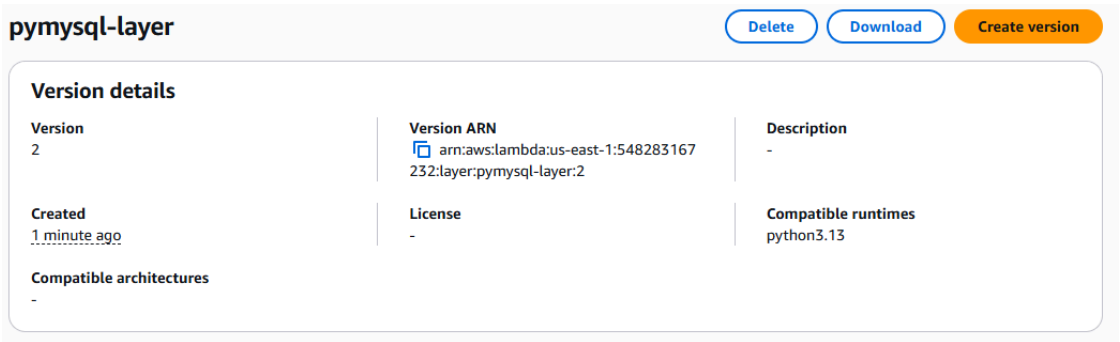


Figure 5.1.3.9: Layer pymysql-layer

B. Attaching the Layer to the Function

- Selected pymysql-layer
- Attached the latest version

This allows the Lambda function to import PyMySQL successfully.

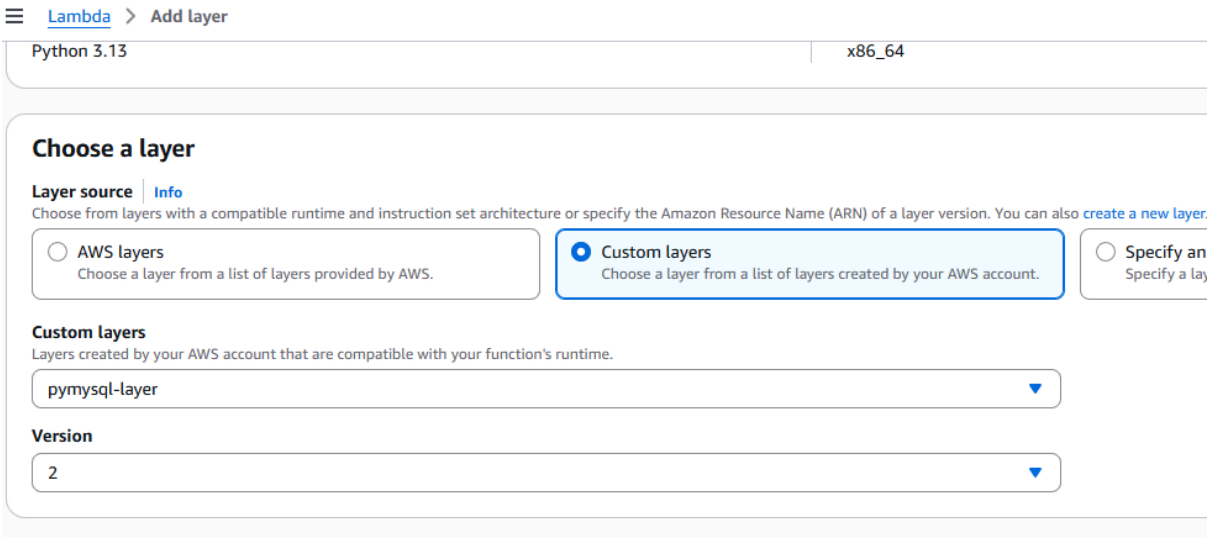


Figure 5.1.3.10: Attaching the Layer to the Lambda Function

(5) Implementing the Lambda Logic

The AutoPO_Generator function performs several automated tasks:

A. RDS Connection and Item Retrieval

- Reads environment variables
- Connects to RDS using PyMySQL
- Fetches all items below threshold
- Considers existing purchase orders to avoid duplication

B. Purchase Order Generation

- Groups items by supplier
- Creates a purchase order for each supplier
- Inserts all required purchase_order_details records
- Calculates quantities and item costs automatically

C. Internal Notifications

- Retrieves active Admin/Manager accounts
- Inserts a notification record for each user
- Includes PO link for quick access

D. Logging

Extensive logging was added:

- logger.info("Lambda triggered")
- logger.info("Database connection successful")
- logger.info("Fetched X low-stock items")
- logger.info("Created PO #XX")
- logger.info("Notifications sent")

These logs appear in CloudWatch and help track execution and troubleshoot errors.

E. Execution Result

Sample success response:

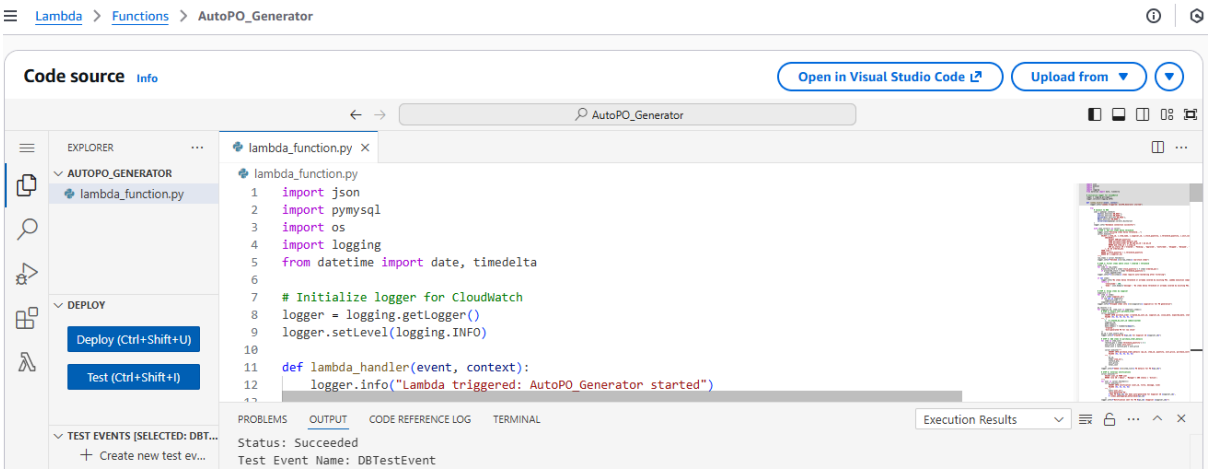


Figure 5.1.3.11: Auto-Generate PO Success (PO-22)


```
{
  "statusCode": 200,
  "body": "{\"message\": \"Auto-generated POs successfully.\",
\"created\": [{\"supplier_id\": 1, \"po_id\": 22}]}"
}
```

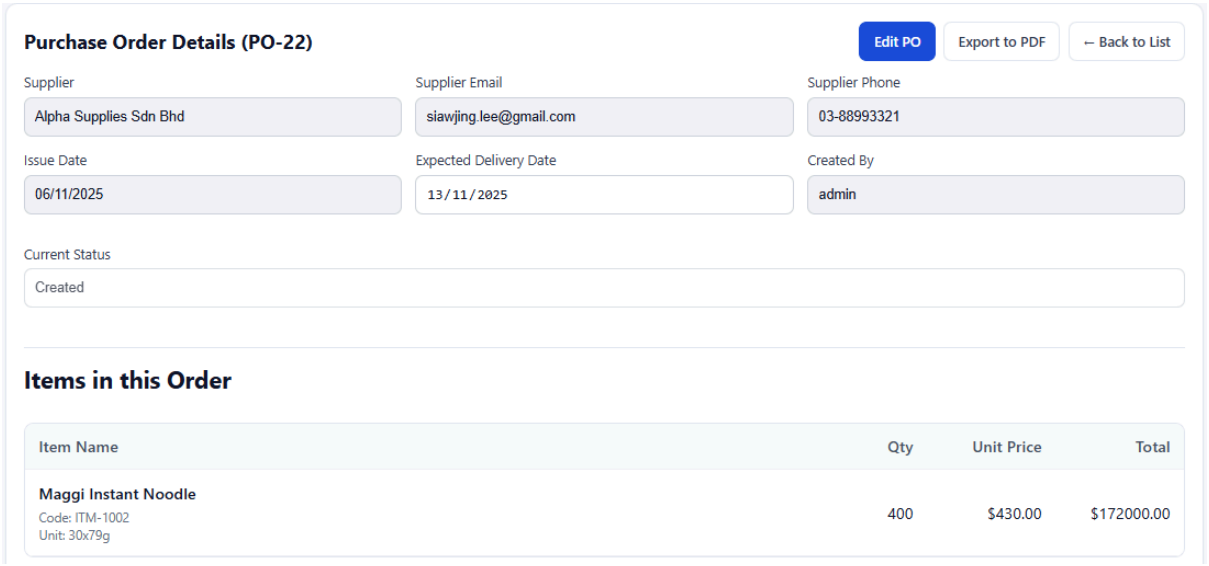


Figure 5.1.3.11: Auto-Generate PO Details (PO-22)

If no items require reordering, the function returns:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"No items below threshold or already
covered by existing POs.\"}"
}
```

(6) Scheduling the Lambda Function

To run the automation daily, Amazon EventBridge (CloudWatch Events) was used.

A. Cron Schedule

cron(0 1 * * ? *)

This triggers the function every day at 9:00 AM Malaysia Time (UTC+8).

Table 5.1.3: Cron Schedule

Purpose	Cron Expression	Malaysia Time
Daily at 9 AM	cron(0 1 * * ? *)	9:00 AM

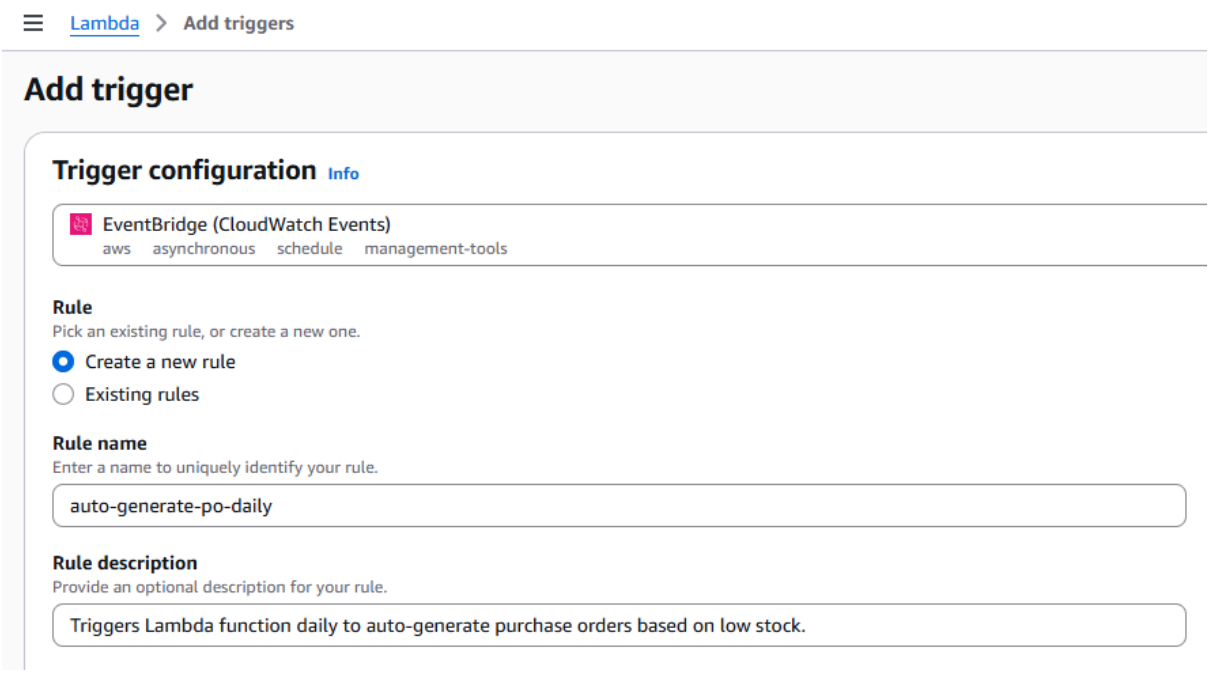


Figure 5.1.3.12: Lambda Function Trigger Configuration 1

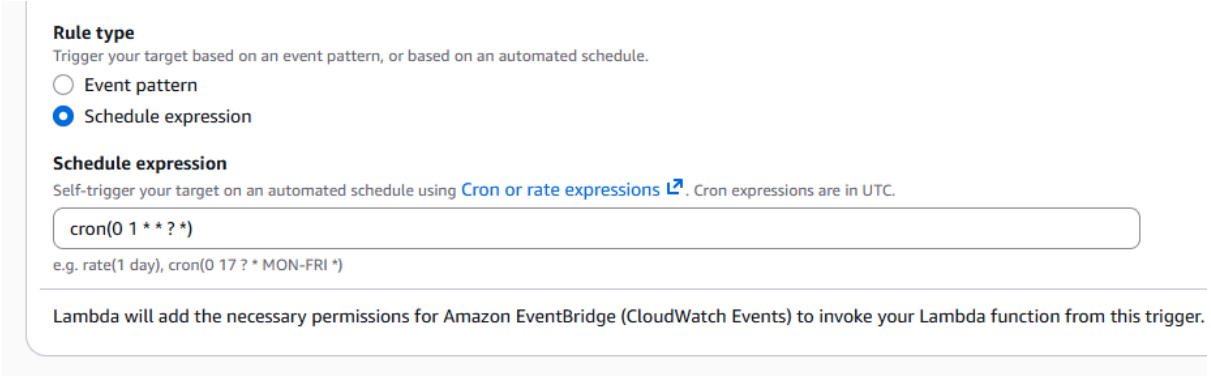


Figure 5.1.3.13: Lambda Function Trigger Configuration 2

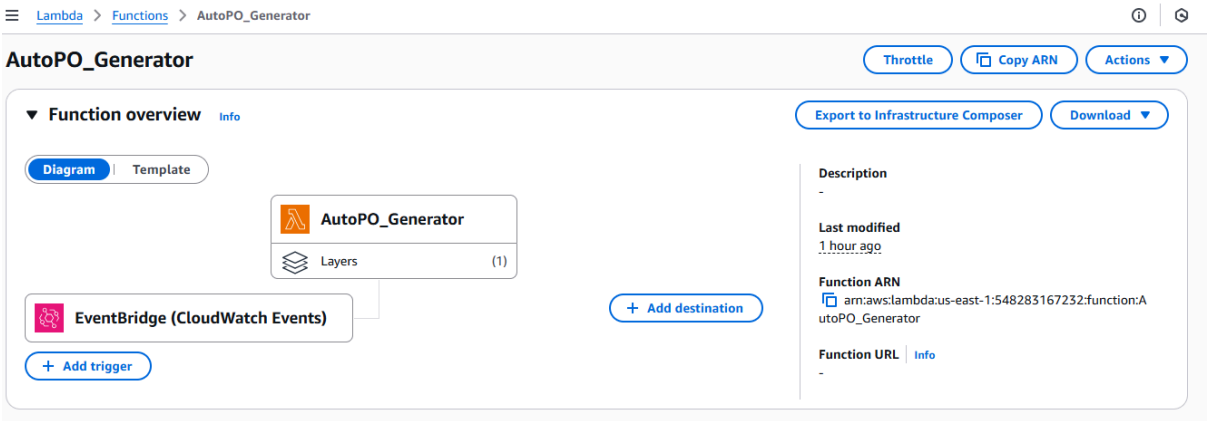


Figure 5.1.3.14: Successfully Created Lambda with EventBridge Trigger

(7) Adjusting Function Timeout and Memory

The default timeout of 3 seconds was insufficient for PO generation, so the values were updated:

- Timeout: Increased to 30 seconds
- Memory: Increased to 256 MB

This improves execution speed and reduces the risk of timeouts.

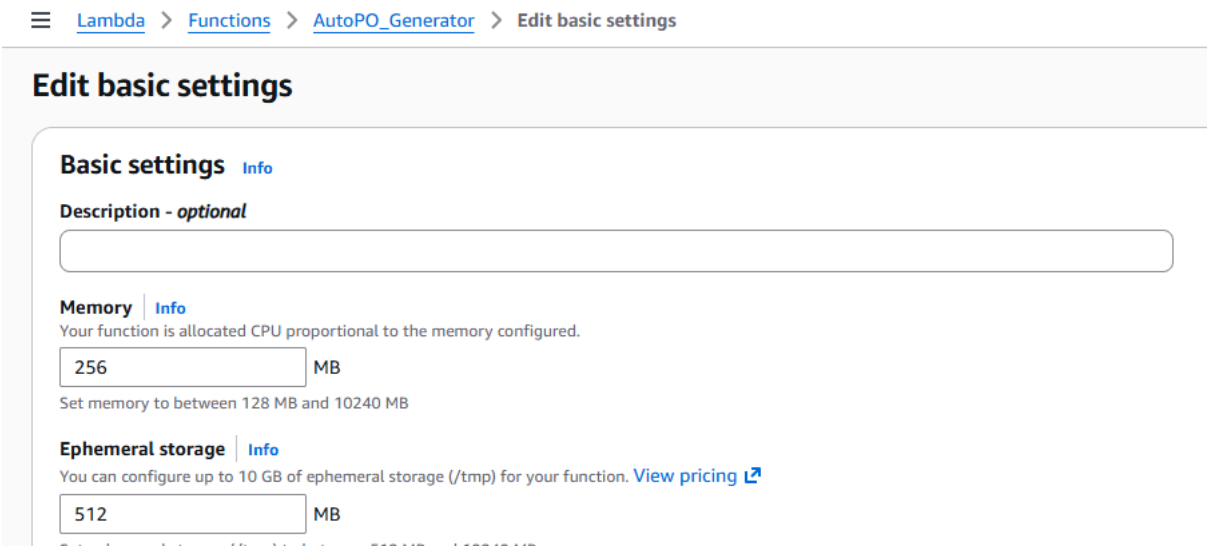


Figure 5.1.3.15: Lambda Function Settings 1

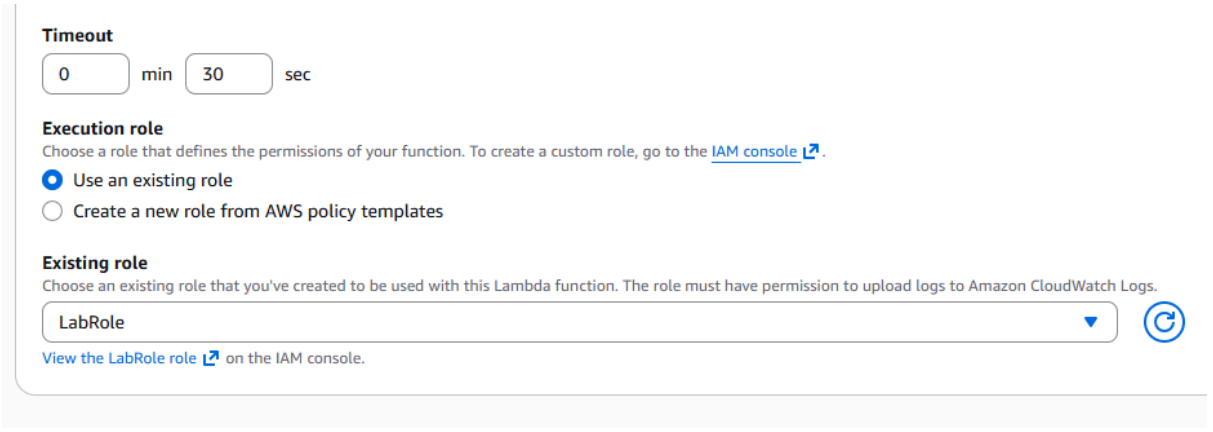


Figure 5.1.3.16: Lambda Function Settings 2

5.1.4 Notification Service Setup (AWS SNS)

To support real-time communication in the Business Process Automation System, Amazon Simple Notification Service (SNS) was configured to deliver email alerts for purchase order activities. SNS acts as the cloud-based push notification component that informs system users (Admin, Manager, Staff, Supplier) of important events such as PO creation or status changes.

A. Creating the SNS Topic

A new SNS topic was created to act as the central communication channel for system notifications.

- Service: AWS SNS
- Topic Type: Standard
- Topic Name: InventoryAlerts
- Topic ARN: Automatically generated after creation
  (e.g., arn:aws:sns:us-east-1:548283167232:InventoryAlerts)

The topic serves as the endpoint that PHP code via AWS SDK or other AWS services can publish messages to.

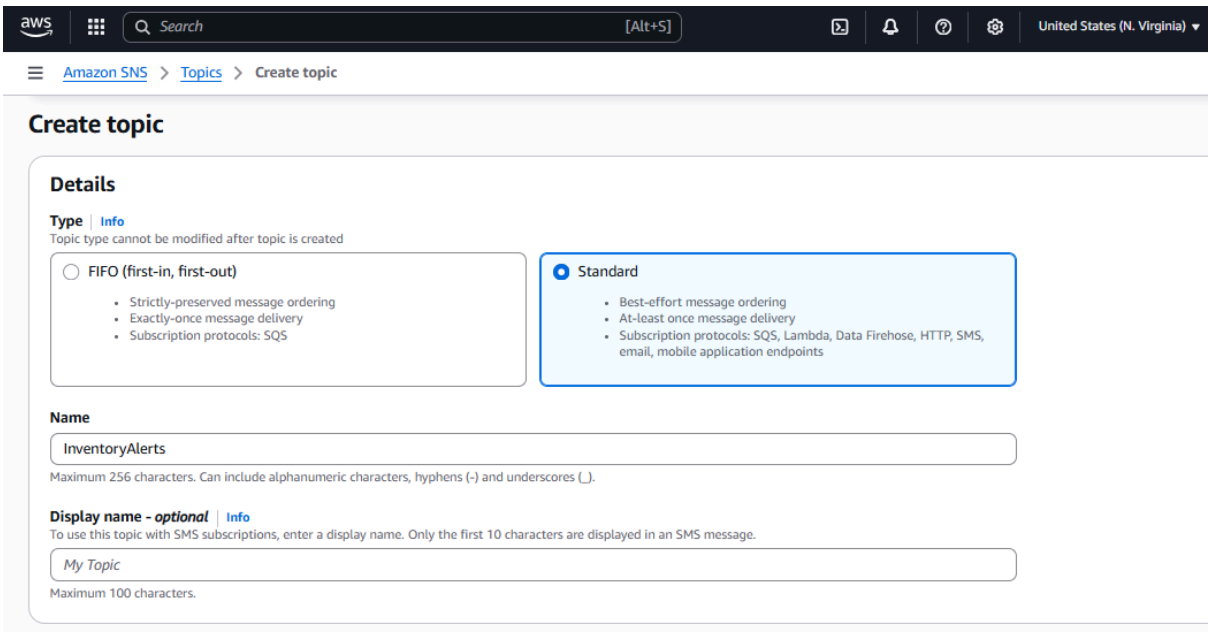


Figure 5.1.4.1: Creating the SNS Topic

B. Adding Email Subscriptions

SNS uses subscriptions to define who receives alerts. Email was selected because it is simple and suitable for testing in a student environment.

Steps performed:

1. Open the InventoryAlerts topic.
2. Click Create subscription.
3. Protocol: Email
4. Endpoint: User email address
5. Confirm the subscription through the email link sent by AWS.

This ensures the user will receive SNS alerts triggered by the system.

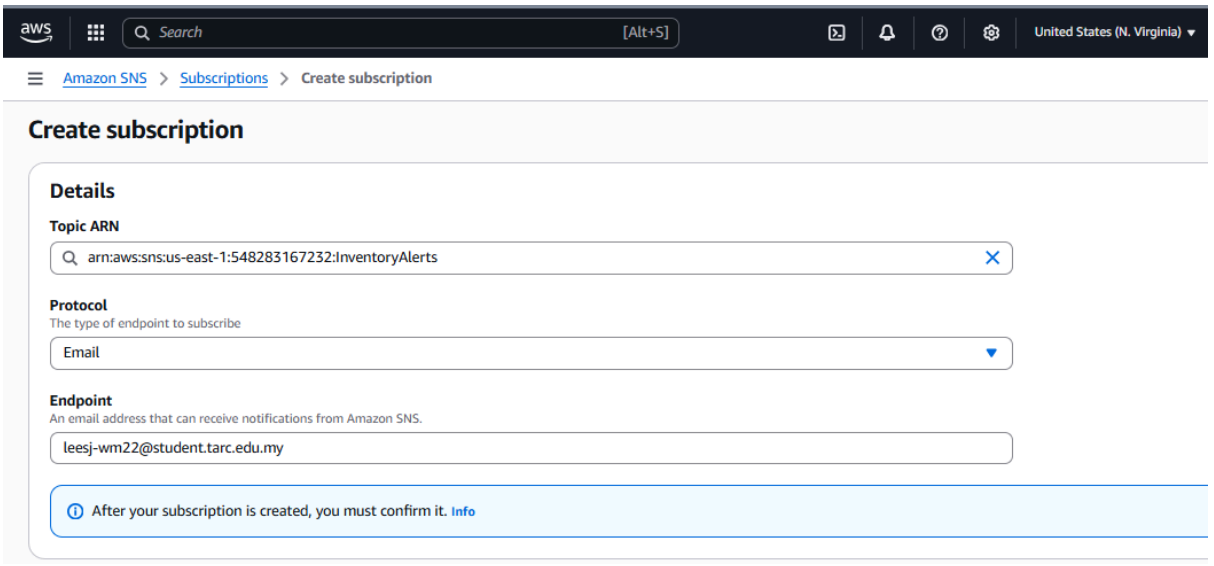


Figure 5.1.4.2: Adding Email Subscriptions

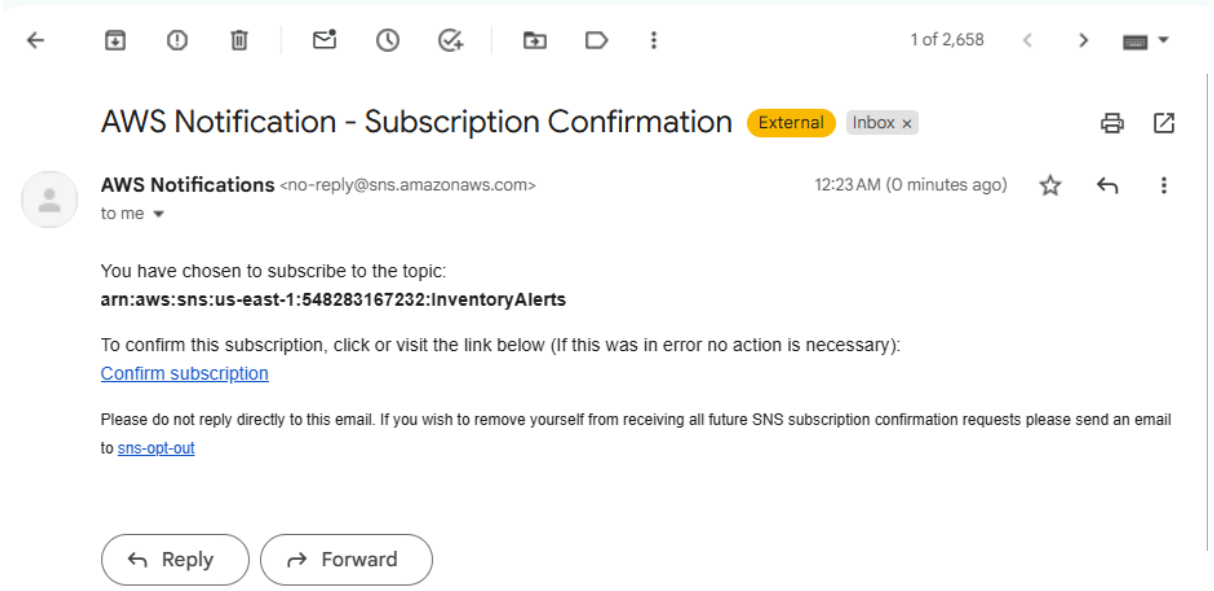


Figure 5.1.4.3: Confirm Subscription through Email Link

C. Multiple Subscriptions for System Roles

To simulate different system users, four SNS email subscriptions were created. Each subscription includes a message filter policy to ensure users only receive notifications relevant to their responsibilities.

Table 5.1.4: System Roles & Subscription Type

Role	Subscription Type	Purpose / Actual Filter Policy Behaviour
Admin	Receives all notifications	Full system overview; no filter policy applied
Manager	PO and Stock-related alerts	Receives events for: PO creation, PO updates, item management, low stock, out of stock, and expiry alerts
Staff	Operational updates	Receives only PO status changes related to receiving workflow (RECEIVED, ISSUE, COMPLETED, DELAYED, SHIPPED)
Supplier	Supplier-specific PO updates	Receives PO status notifications that match their email and supplier workflow (PENDING_APPROVAL, CONFIRMED)

All subscriptions were confirmed and appeared as Confirmed in the SNS console.

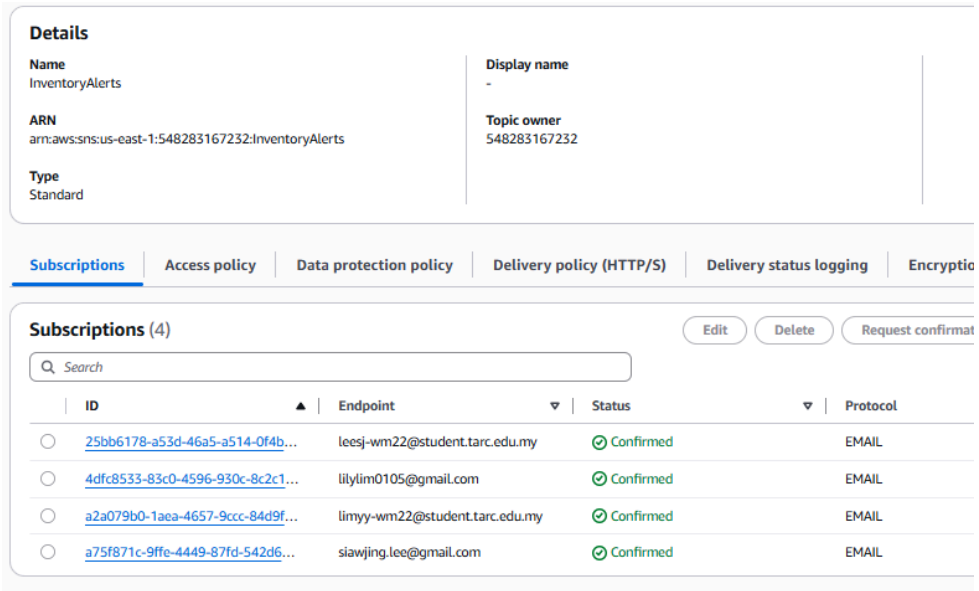


Figure 5.1.4.4: Confirmed Subscription

D. Applying Filter Policies

SNS filter policies determine which notifications each role should receive. This prevents unnecessary emails and ensures only the relevant events are delivered. Policies were applied to each subscription.

- Admin: No filter policy (receives all messages).
- Manager: Receives all purchase-order and stock-related events.

Figure 5.1.4.5: Manager Filter Policy

- Staff: Focused on operational PO status updates (items moving through the receiving process).

Figure 5.1.4.6: Staff Filter Policy

- Supplier: Only receives notifications related to their own purchase orders.

Figure 5.1.4.6: Supplier Filter Policy

These filter rules align SNS behavior with the logic of the inventory management workflow.

E. Integrating SNS with the Application

To allow the PHP system to publish messages to the SNS topic:

- The AWS SDK for PHP was installed using Composer. This is required because it manages and installs the AWS SDK for PHP along with its dependencies, ensuring the application can communicate with AWS services.
- Temporary AWS credentials from AWS Learner Lab were stored inside config/aws.php.
- A custom Publish policy was created to allow SNS publishing.

This enables purchase order actions such as create, approve, reject, and ship to automatically trigger SNS alerts.

aws.php

config > aws.php

```php
<?php
// config/aws.php
// Credentials MUST be updated every time when restart AWS Learner Lab
return [
    'region'     => 'us-east-1',
    'topic_arn' => 'arn:aws:sns:us-east-1:548283167232:InventoryAlerts',
    'bucket'     => 'my-fyp-bucket1',

    'credentials' => [
        'key'    => 'ASIAX7KBZGIAOGO35UHC',
        'secret' => 'hBr1PvY2t7iYWjQ3KjS15TeWbZ5G+jDNlq1yZrrr',
        'token'  => 'IQoJb3JpZ2luX2VjEH4aCXVzLXdlc3QtMiJHMEUCIElVPBbpQ4x4AuZaKwhG3Z16VMsfn6z8KQcGM8oGmcFxAiE
    ],
];
```

Figure 5.1.4.7: config/aws.php file code

PO #25 Status Updated: CONFIRMED External Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> Fri, Nov 14, 12:05 PM

to me

The status of a Purchase Order has been updated.

PO Number: #25
New Status: CONFIRMED
Updated By: leesj-wm22@student.tarc.edu.my
Actor Role: ADMIN

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:548283167232:InventoryAlerts:25bb6178-a53d-46a5-a514-0f4b9a7192dd&Endpoint=leesj-wm22@student.tarc.edu.my

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support

Reply Forward

Figure 5.1.4.8: SNS Email Notification (PO Status Updated)

Out of Stock Alert: Horlicks (ITM-1004) External Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> Thu, Dec 4, 5:59 PM (2 days ago)

to me

An item is now OUT OF STOCK.

Item ID: #4
Item Name: Horlicks
Item Code: ITM-1004
Current Stock: 0

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:548283167232:InventoryAlerts:25bb6178-a53d-46a5-a514-0f4b9a7192dd&Endpoint=leesj-wm22@student.tarc.edu.my

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support

Reply Forward

Figure 5.1.4.8: SNS Email Notification (Out of Stock)

5.1.5 System Monitoring (Amazon CloudWatch)

Amazon CloudWatch is used as the central monitoring and alerting service for the Business Process Automation System. It tracks the behaviour of the Lambda function, RDS database and SNS notifications, and triggers email alerts when abnormal conditions occur.

A. Lambda Error Alarm (AutoPO_Generator)

To ensure the auto-reordering logic runs reliably, a CloudWatch alarm was created for the AutoPO_Generator Lambda function.

- Monitored resource: Lambda (AutoPO_Generator)
- Metric: Errors

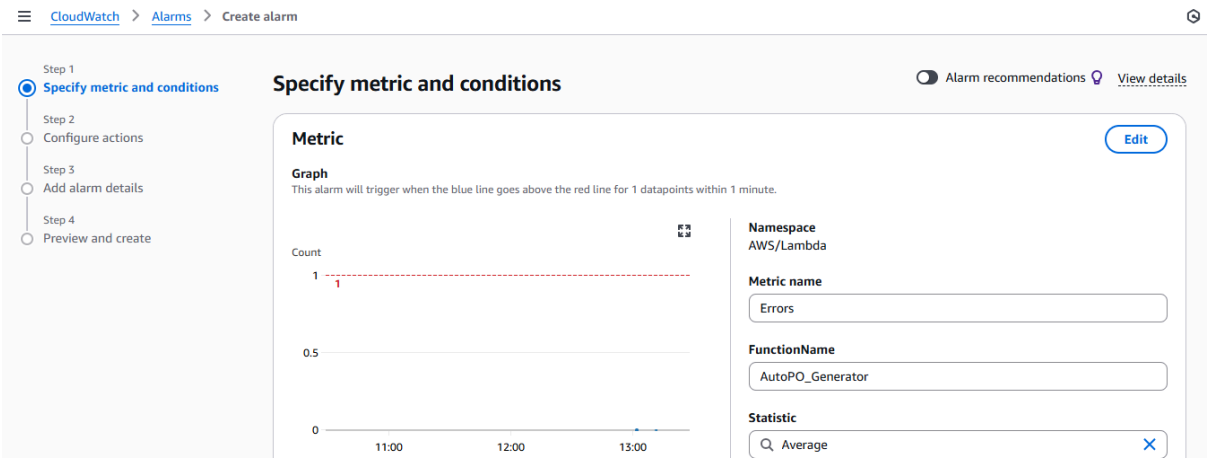


Figure 5.1.5.1: AutoPO_Generator Error Alarm Metric

- Threshold: Static – alarm when Errors ≥ 1
- Period / Evaluation: 1 minute, 1 out of 1 evaluation periods

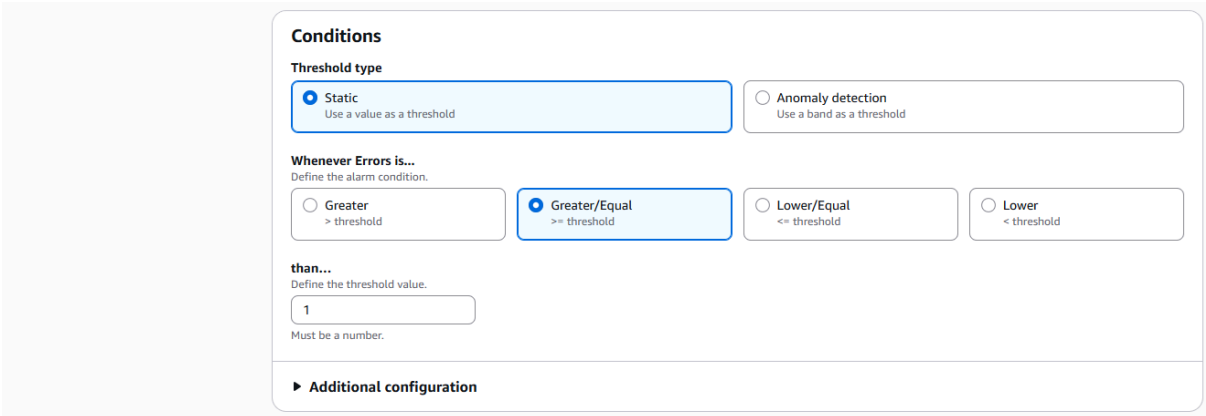


Figure 5.1.5.2: AutoPO_Generator Error Alarm Condition

- Alarm name: AutoPO_Generator_ErrorAlarm

- Notification channel: New SNS topic AutoPO_Error_Alerts with admin email subscription (e.g. leesj-wm22@student.tarc.edu.my)

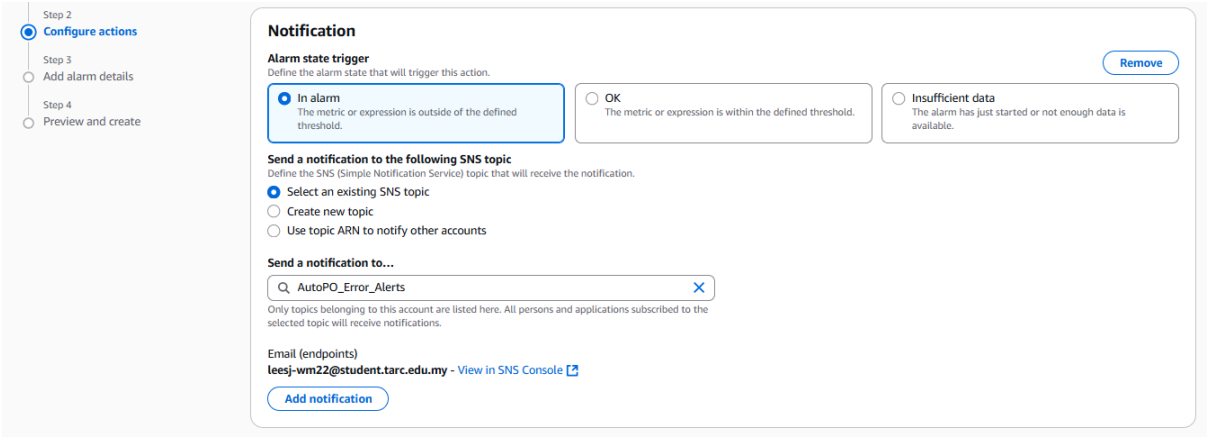


Figure 5.1.5.3: AutoPO_Generator Error Alarm Configuration

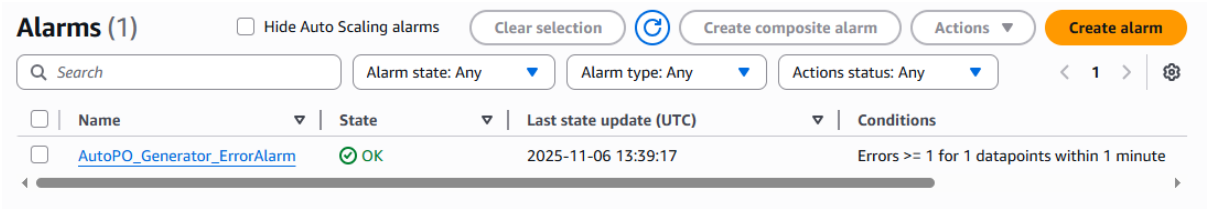


Figure 5.1.5.4: Successfully Created AutoPO_Generator Error Alarm

When the function fails, the alarm moves to ALARM state and sends an email so the developer can investigate quickly.

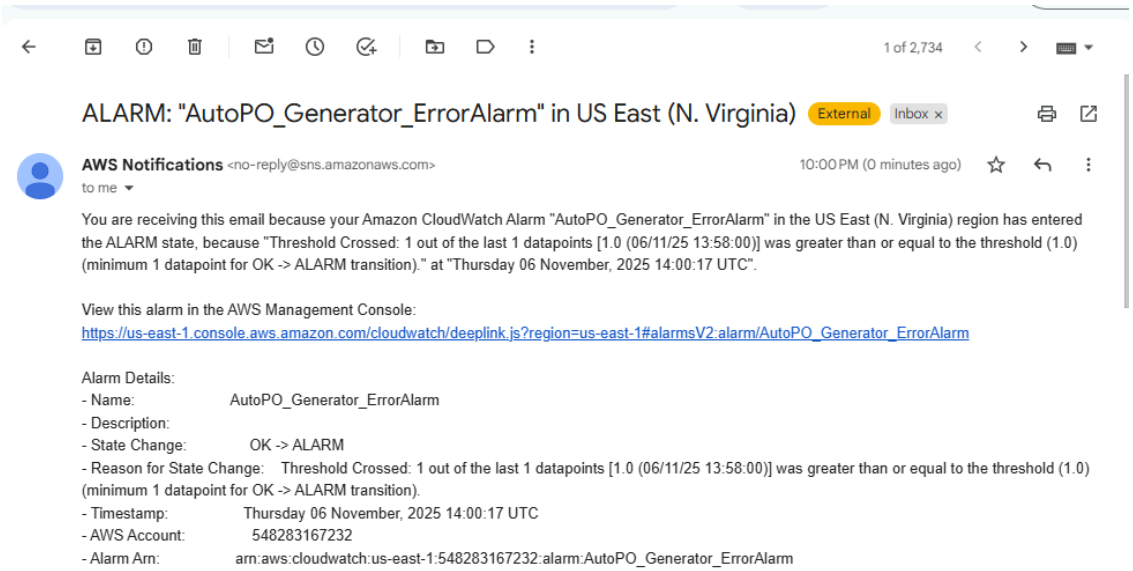


Figure 5.1.5.5: Email Alarm Notification

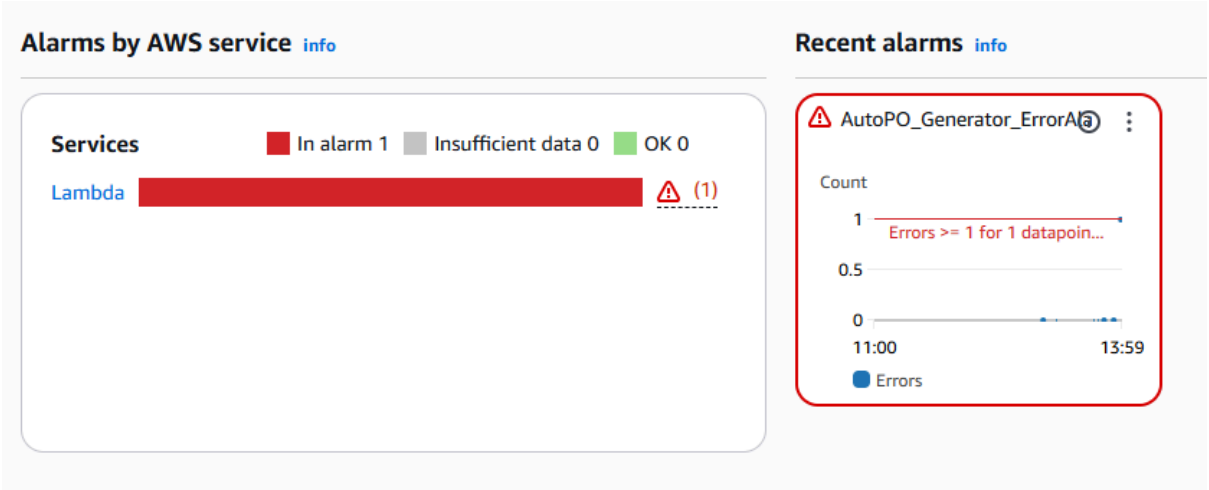


Figure 5.1.5.6: Lambda Alarm in CloudWatch

B. Lambda Monitoring Dashboard

A dedicated CloudWatch dashboard was created to visualise the behaviour of the auto-PO function over time.

- Dashboard name: AutoPO_Generator_Dashboard
- Widgets: Line charts using Lambda metrics:
  - Invocations – how often the function runs
  - Errors – number of failed executions
  - Duration – average execution time

The widgets are configured with a title “PO Auto Generator:Usage & Errors” and 1 week time ranges so that trends can be monitored.

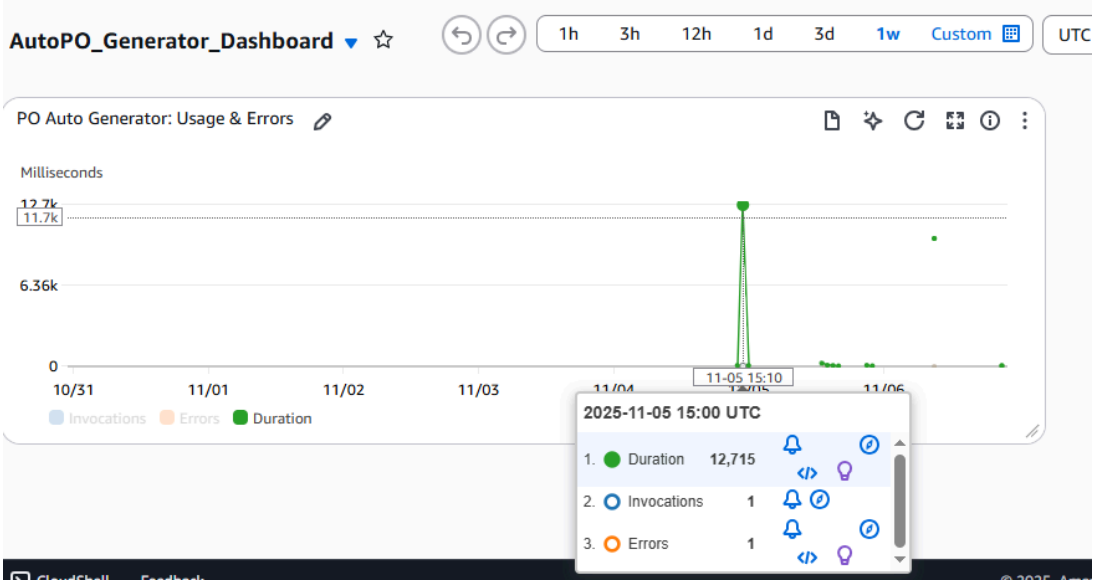


Figure 5.1.5.7: Lambda Monitoring Dashboard

C. RDS Health Dashboard and CPU Alarm

CloudWatch is also used to monitor the Amazon RDS MySQL instance that stores the inventory data.

RDS Dashboard

- Dashboard name: Inventory_RDS_Dashboard
- Key metrics visualised:
    - CPUUtilization: database CPU load
    - DatabaseConnections: number of active connections
    - FreeStorageSpace: remaining storage (displayed in GB)
    - ReadIOPS and WriteIOPS: read/write activity
    - ReadLatency and WriteLatency: query response time

These graphs are grouped under a section “Inventory System: RDS Database Health”.

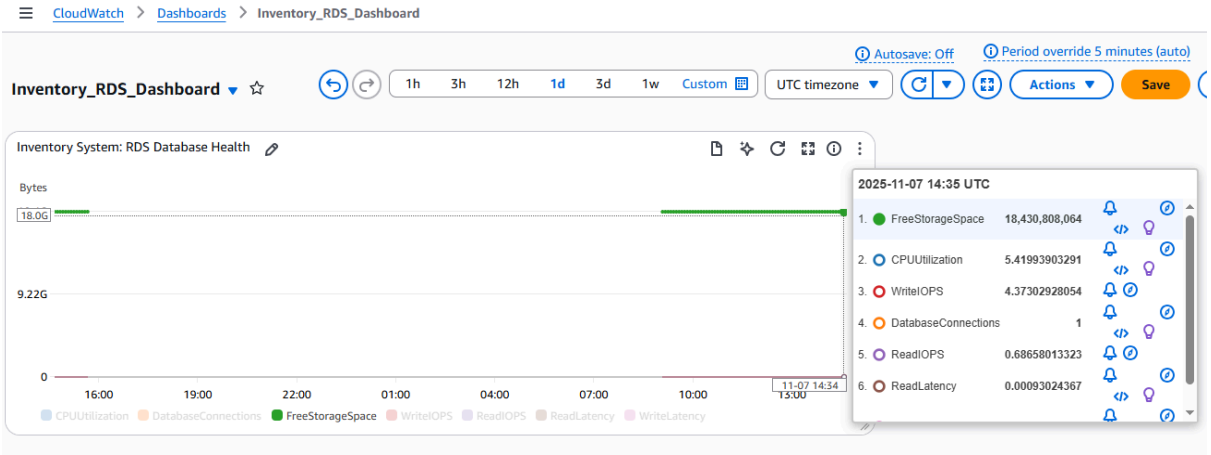


Figure 5.1.5.8: RDS Health Dashboard

CPU Utilisation Alarm

To detect resource bottlenecks, a specific alarm was created on CPU usage:

- Metric: CPUUtilization
- Threshold: CPUUtilization ≥ 70%
- Period: 1 minute, evaluation 1 out of 1
- Alarm name: RDS_CPUUtilization_High_Alarm
- Notification: SNS topic RDS_Health_Alerts with email subscription

The email subscription is confirmed so that any CPU spike will immediately notify the maintainer.

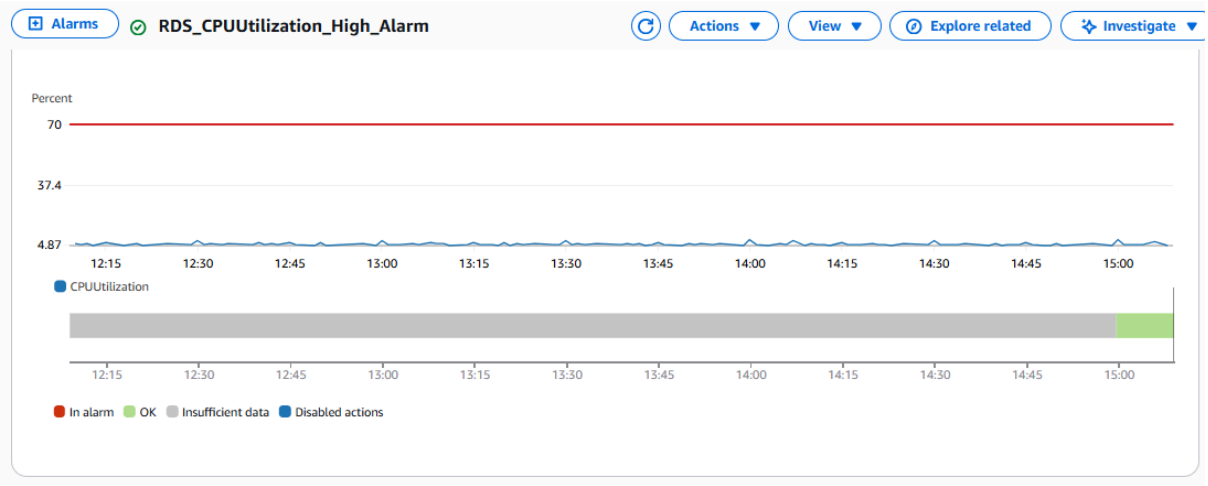


Figure 5.1.5.9: CPU Utilisation Alarm

D. Monitoring SNS Delivery Failures

Because the system relies on SNS to deliver email notifications, CloudWatch is also used to detect any delivery problems.

SNS Failure Dashboard

- Dashboard name: SNS_Notification_Failures_Dashboard
- Metric: NumberOfNotificationsFailed
- Source: AutoPO_Error_Alerts
- Widget: Line chart showing failed deliveries over time

SNS Failure Alarm

An additional alarm ensures failures are not ignored:

- Metric: NumberOfNotificationsFailed
- Condition: Static threshold, ≥ 1
- Period: 1 minute, evaluation 1 of 1
- Alarm name: SNS_NotificationFailure_Alarm
- Notification: Uses the same SNS topic (AutoPO_Error_Alerts) so the owner receives an email when SNS cannot deliver messages.

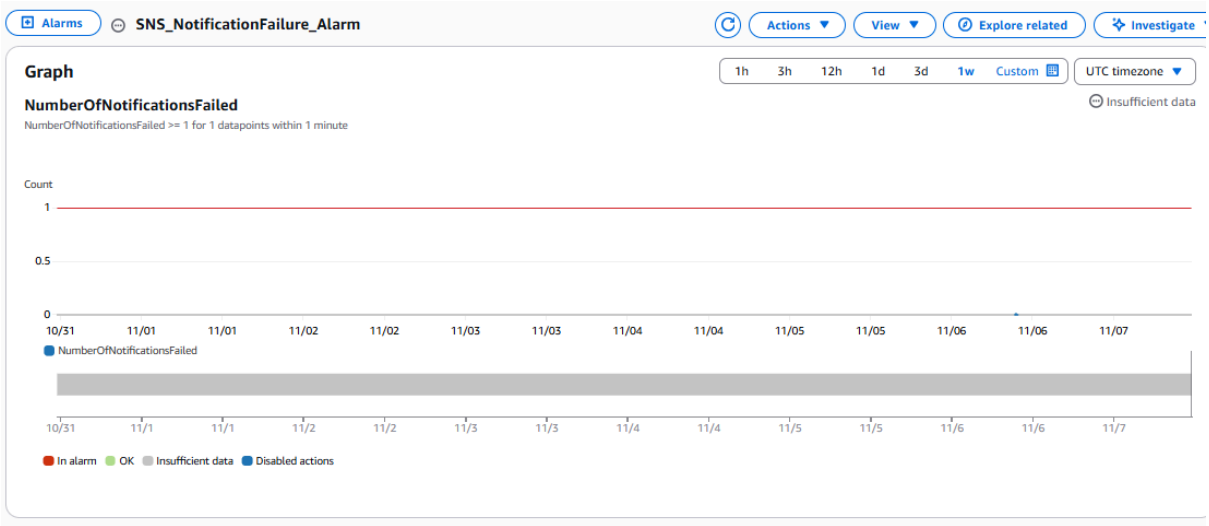


Figure 5.1.5.10: SNS Failure Alarm

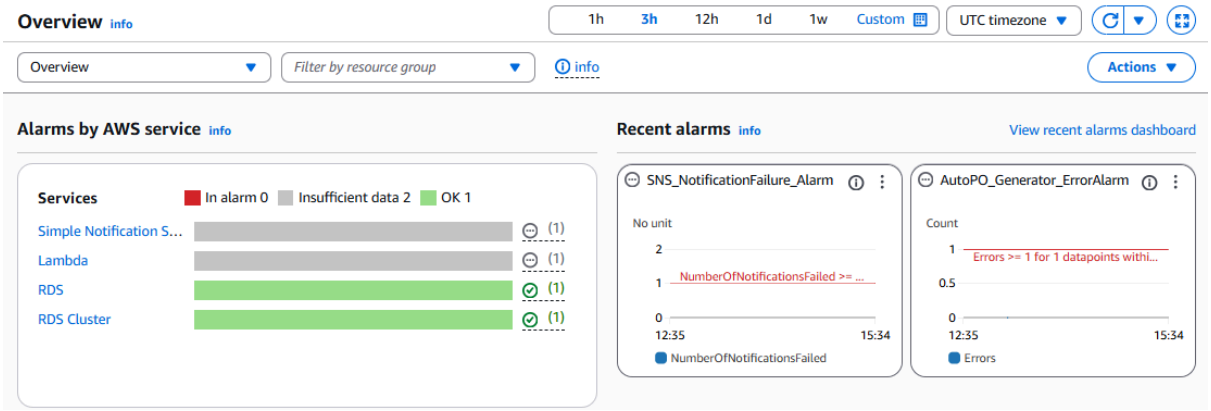


Figure 5.1.5.11: Alarm by AWS Service in CloudWatch

5.2 Module Implementation

This section presents the technical implementation of the system's core modules, explaining how the underlying logic and coding structure work together to support the functional requirements defined in Chapter 3. It describes how the server-side components interact with the database, how data is processed throughout each module, and how the overall architecture ensures consistent, secure, and reliable system performance.

5.2.1 Reordering Module

The Reordering Module manages both automated and manual purchase order processes within the inventory management system. It handles the detection of low-stock items, generates purchase orders, applies permission-controlled updates, manages status changes across the procurement lifecycle, and ensures inventory accuracy once goods are received. This section explains only the main backend logic that drives the reordering workflow.

A. Automated Purchase Order Generation

The automated reordering process runs in a serverless environment, where the system checks for items below their threshold levels. The function retrieves low-stock items and includes the quantity already ordered in active purchase orders to prevent duplicate or unnecessary reorders:

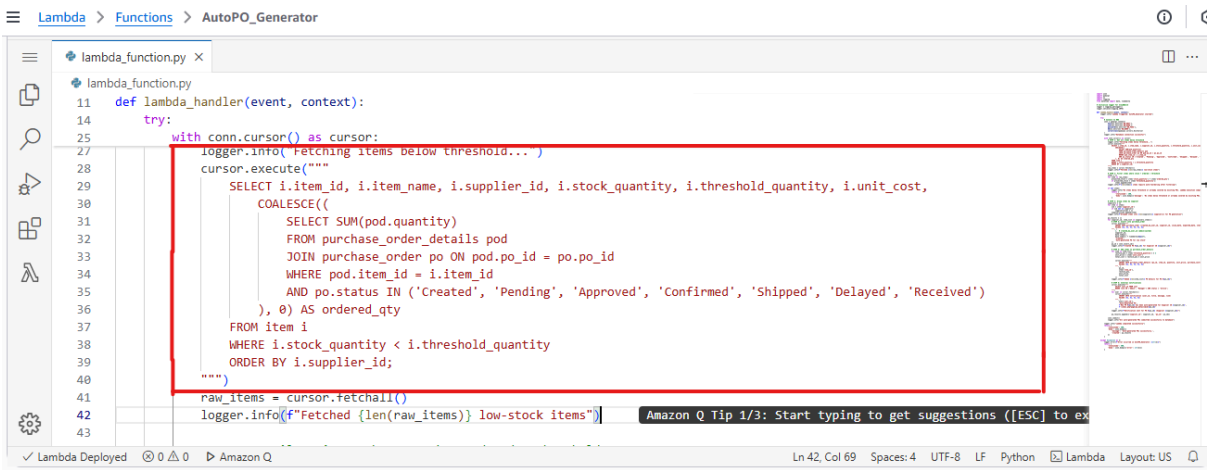


Figure 5.2.1.1: Lambda Function (Low Stock Detection)

The system calculates projected stock and only proceeds if the quantity remains below the threshold:

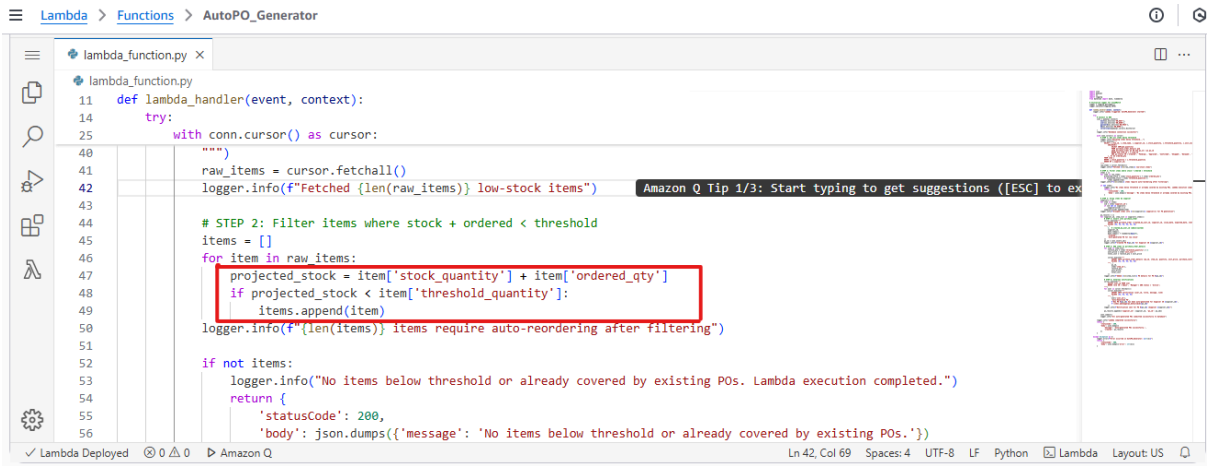


Figure 5.2.1.2: Lambda Function (Projected Stock Calculation)

Items belonging to the same supplier are grouped so a single consolidated purchase order is created per supplier:

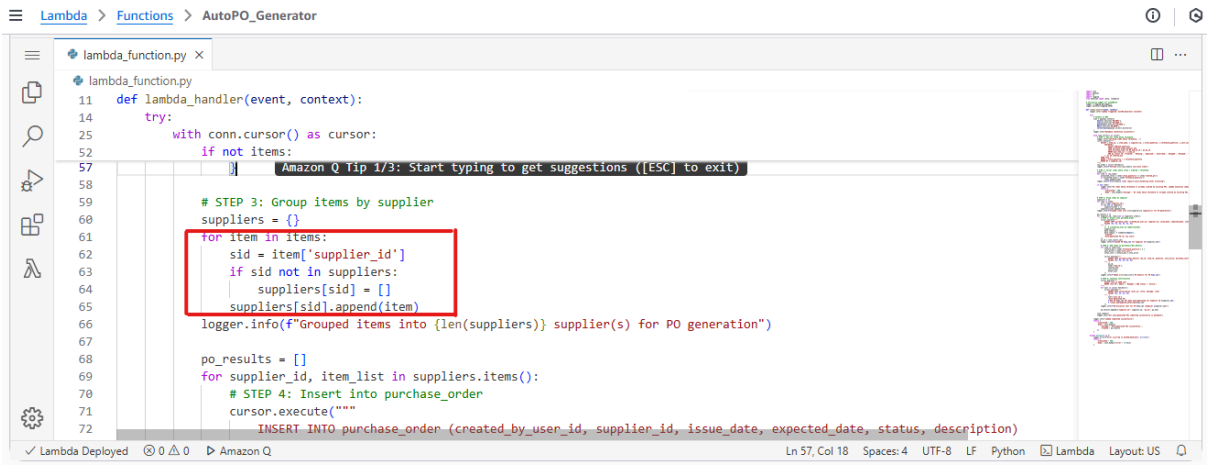


Figure 5.2.1.3: Lambda Function (Grouping Items by Supplier)

A new PO is then inserted with the status Created, indicating that the PO is ready for internal review:

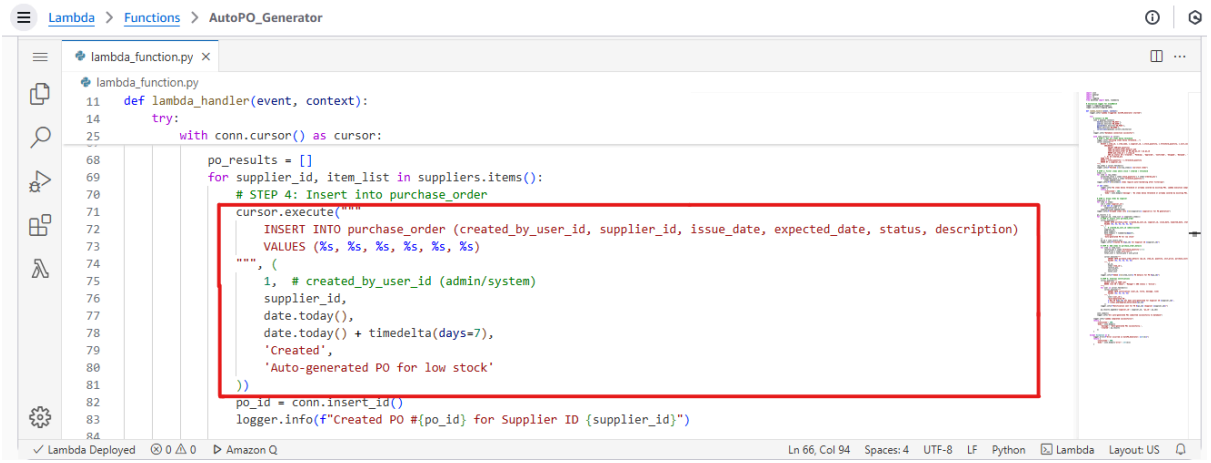


Figure 5.2.1.4: Lambda Function (Auto-Generated Purchase Order Creation)

Each item is then added into the PO details with its calculated quantity and cost, completing the automated reordering cycle.

B. Purchase Order Status Flow

The system uses a structured and realistic status pipeline based on internal actions, supplier decisions, and automated rules.

1. Internal Staff Actions

- Created: A user manually creates the PO.
- Pending: An Admin or Manager approves the PO and sends it to the supplier.
- Confirmed: After supplier approval, Admin/Manager confirms it for processing.
- Received: Staff mark the PO as received when items arrive.
- Issue: Items are marked with discrepancies or damages.
- Completed: After verification, the PO is closed and processed into inventory.

2. Supplier Actions

- Approved / Rejected: Supplier decides whether to fulfil the order.
- Shipped: Supplier dispatches items and updates the PO status.

3. System-Triggered Actions

- Delayed: Automatically triggered when the expected delivery date has passed but items are not yet received.
- Inventory Update: When the PO reaches Completed, item quantities are added to stock to finalize the procurement cycle.

Example of internal stock updates when a PO transitions to Completed:

```
if (empty($items_to_add)) {
    // No items on PO, just mark as complete.
} else {
    $item_update_sql = "UPDATE item SET stock_quantity = stock_quantity + :quantity WHERE item_id = :item_id";
    $stmt_update_item = $pdo->prepare(query: $item_update_sql);
```

Figure 5.2.1.5: Internal Stock Update upon Purchase Order Completion

This step ensures that physical receiving is correctly synchronized with the system inventory.

C. Purchase Order Workflow

The system implements a multi-stage workflow to ensure proper control and traceability across procurement activities. Role-based authorization is enforced through:

```
// Authorization Checks
if ($is_fully_editable) {
    // User is trying a full edit (items, date, or status)
    if (!Auth::can(capability: 'manage_po_status_all')) {
        jerr_redirect(message_key: 'auth_error', po_id: $po_id);
    }
} else if ($new_status !== $current_status) {
    // User is trying to change the status on a locked PO
    if (Auth::can(capability: 'manage_po_status_all')) {
        // Allow (Manager/Admin)
    } else if (Auth::can(capability: 'manage_po_status_basic')) {
        // Staff check
        $staff_allowed_statuses = ['Received', 'Issue', 'Completed'];
        if (!in_array(needle: $new_status, haystack: $staff_allowed_statuses, strict: true)) {
            // Staff is trying to set a status they are not allowed to
            jerr_redirect(message_key: 'auth_error', po_id: $po_id);
        }
    } else {
        // User has no status-management perms at all
        jerr_redirect(message_key: 'auth_error', po_id: $po_id);
    }
} else if ($new_status === $current_status && $expected_date !== $current_expected_date) {
    // User is only changing the date on a locked PO.
    // This is not allowed. Only 'create_po' users can change dates, and only on 'Created' POs.
    jerr_redirect(message_key: 'auth_error', po_id: $po_id);
}
```

Figure 5.2.1.6: Role-Based Authorization for Purchase Order Workflow

Status updates are applied through a controlled SQL update:

```
$sql_complete = "
  UPDATE purchase_order
  SET
    status = 'Completed',
    approved_by_user_id = :user_id,
    completion_date = CURDATE()
  WHERE po_id = :id
";
$pdo->prepare(query: $sql_complete)->execute(params: [
    ':user_id' => $user_id,
    ':id' => $po_id
]);
```

Figure 5.2.1.7: Purchase Order Status Update Logic

This ensures that each modification records the timestamp, preserving auditability throughout the procurement process.

D. Exporting Purchase Orders as PDF (FPDF Library)

The system provides a professional PDF export function to generate purchase order documents. Access is permission-controlled:

```
<?php if (Auth::can(capability: 'export_po')): ?>
  <a href="/api/export_po.php?id=<?= e(v: $po['po_id']) ?>"
     class="btn btn-secondary"
     target="_blank">
      Export to PDF
  </a>
<?php endif; ?>
```

Figure 5.2.1.8: Purchase Order PDF Export Authorization

The backend retrieves PO header information, supplier details, company settings, and all line items:

```
// PO line items
$sql_items = "
    SELECT
        pod.quantity, pod.unit_price,
        i.item_code, i.item_name, i.measurement
    FROM purchase_order_details pod
    LEFT JOIN item i ON pod.item_id = i.item_id
    WHERE pod.po_id = :po_id
    ORDER BY i.item_name
";
```

Figure 5.2.1.9: SQL Query for Purchase Order Line Item Retrieval

A custom FPDF layout defines the document structure, including the title, headers, and table formatting:

```
class PDF extends FPDF
{
    1 reference | 0 overrides | prototype
    function Header(): void {
        $this->SetFont(family: 'Arial', style: 'B', size: 18);
        $this->Cell(w: 0, h: 10, txt: 'PURCHASE ORDER', border: 0, ln: 1, align: 'C');
        $this->Ln(h: 5);
    }
    2 references | 0 overrides | prototype
    function Footer(): void {
        $this->SetY(y: -15);
        $this->SetFont(family: 'Arial', style: 'I', size: 8);
        $this->Cell(w: 0, h: 10, txt: 'Page '.$this->PageNo().'/{nb}', border: 0, ln: 0, align: 'C');
    }
}
```

Figure 5.2.1.10: FPDF Document Layout Customization

Line items are printed into a formatted item table:

```
$pdf->Cell(w: 80, h: 7, txt: (string)$item['item_name'], border: 1);
$pdf->Cell(w: 25, h: 7, txt: (string)$item['item_code'], border: 1);
$pdf->Cell(w: 20, h: 7, txt: (string)$qty, border: 1, ln: 0, align: 'C');
$pdf->Cell(w: 30, h: 7, txt: '$' . number_format(num: $price, decimals: 2), border: 1, ln: 0, align…'R');
$pdf->Cell(w: 35, h: 7, txt: '$' . number_format(num: $line, decimals: 2), border: 1, ln: 1, align…'R');
```

Figure 5.2.1.11: FPDF Line Item Table Rendering Logic

Once the document is generated, it is streamed directly to the browser:

```
while (ob_get_level()) { ob_end_clean(); }
header(header: 'Content-Type: application/pdf');
header(header: 'Content-Disposition: inline; filename="PO-' . $po_data['po_id'] . '.pdf"');
header(header: 'Content-Length: ' . strlen(string: $pdfBytes));
echo $pdfBytes;
exit;
```

Figure 5.2.1.12: PDF Streaming to Browser

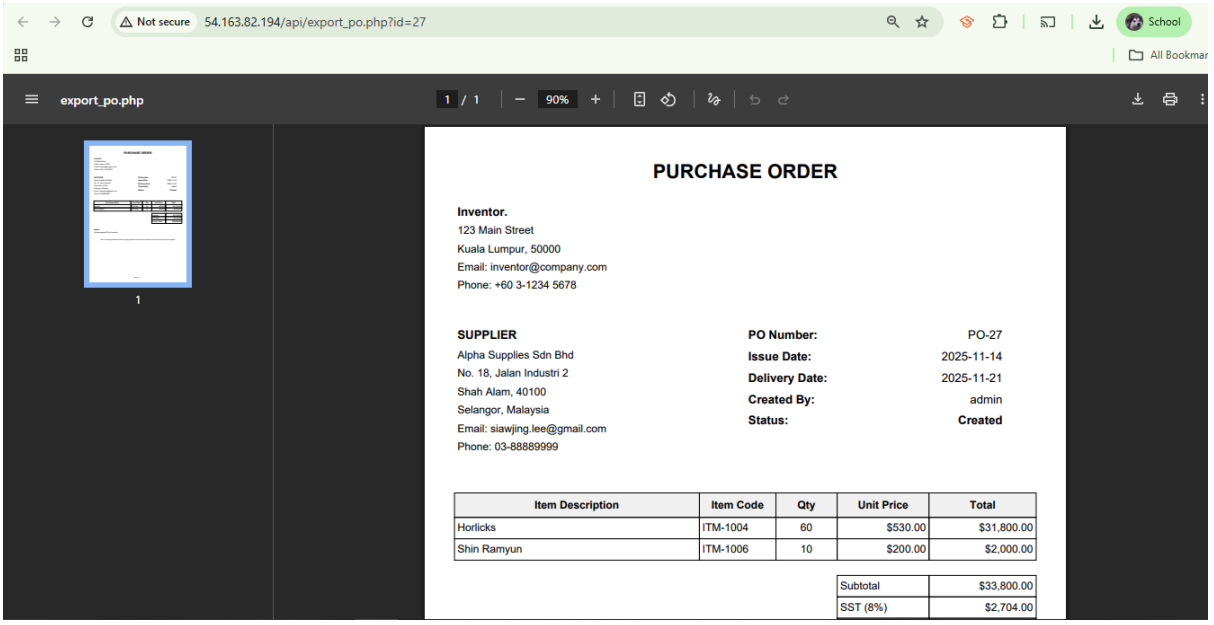


Figure 5.2.1.13: Generated Purchase Order PDF Output

This PDF export functionality ensures consistent, printable documentation suitable for suppliers, financial records, and management reviews.

5.2.2 Supplier Management Module

The Supplier Module manages the interaction between external suppliers and the procurement process. It handles supplier profile information, enforces secure access to the supplier portal, and supports controlled supplier-driven updates to purchase order statuses. This module ensures that suppliers can only access their own records and perform actions permitted within the procurement workflow.

A. Supplier Profile Management

Supplier profile information is maintained through standard PHP and MySQL operations. Administrators may update contact details, phone numbers, and address information to ensure supplier records remain accurate and usable during procurement activities.

```
$sql = "UPDATE supplier SET
            contact_person = :contact_person,
            phone = :phone,
            street_address = :street_address,
            city = :city,
            postcode = :postcode,
            state = :state
        WHERE supplier_id = :supplier_id";

$stmt = $pdo->prepare(query: $sql);
$stmt->execute(params: [
    ':contact_person' => $contact_person,
    ':phone' => $phone,
    ':street_address' => $street_address,
    ':city' => $city,
    ':postcode' => $postcode,
    ':state' => $state,
    ':supplier_id' => $supplier_id
]);
```

Figure 5.2.2.1: Supplier Profile Update Logic

This snippet shows how supplier details are updated in the database after form data is validated on the server. Administrative permission checks ensure that only authorized users may edit supplier profile records.

B. Supplier Portal Access

To ensure a strict separation between internal users and external suppliers, the supplier portal uses session-based authentication. Only suppliers with a valid session may access supplier-specific pages.

```
// Authentication check
if (!isset($_SESSION['supplier_id'])) {
    header(header: 'Location: /index.php?page=login');
    exit();
}
```

Figure 5.2.2.2: Supplier Portal Session Authentication Check

This access control prevents staff or unauthorized visitors from entering the supplier portal. When retrieving purchase orders, the system filters results using the supplier's ID to ensure confidentiality:

```php
try {
  // 1. Fetch main PO data
  $sql_po = "
    SELECT
      po.*,
      po.expected_date,
      u.username AS creator_username
    FROM purchase_order po
    LEFT JOIN user u ON po.created_by_user_id = u.user_id
    WHERE po.po_id = :po_id AND po.supplier_id = :supplier_id
  ";
  $stmt_po = $pdo->prepare(query: $sql_po);
  $stmt_po->execute(params: [':po_id' => $po_id, ':supplier_id' => $supplier_id]);
  $po = $stmt_po->fetch(mode: PDO::FETCH_ASSOC);

  if (!$po) {
    throw new Exception(message: "Purchase Order not found or you do not have permission to view it.");
  }
```

Figure 5.2.2.3: Access-Controlled Purchase Order Query for Suppliers

This query ensures that suppliers may only view purchase orders that belong to them.

D. Supplier Purchase Order Status Updates

Suppliers are permitted to update purchase order statuses according to a predefined progression. The allowed transitions are controlled through a system-defined mapping:

```php
// Flow Validation
$supplier_flow_map = [
    'Pending'   => ['Approved', 'Rejected'],
    'Confirmed' => ['Shipped'],
];
```

Figure 5.2.2.4: Supplier Status Flow Validation Logic

This mapping ensures that suppliers cannot update a purchase order outside the permitted sequence. Before applying an update, the system verifies ownership and status consistency:

```php
if (!$po || $po['supplier_id'] != $supplier_id) {
    $pdo->rollBack();
    http_response_code(response_code: 403);
    header(header: "Location: $redirect_url&status=error_permission");
    exit();
}

if ($po['status'] !== $old_status) {
    $pdo->rollBack();
    header(header: "Location: $redirect_url&status=error_sync");
    exit();
}
```

Figure 5.2.2.5: Supplier Status Update Validation Checks

These checks prevent unauthorized updates and ensure that suppliers cannot change a purchase order that has already moved to another stage internally.

Once validated, the updated status is applied:

```php
// IF SECURITY CHECK PASSES, UPDATE THE PO
$update_expected_date_clause = '';
$update_params = [
    ':status' => $new_status,
    ':po_id' => $po_id,
    ':supplier_id' => $supplier_id
];

if ($old_status === 'Pending') {
    $update_expected_date_clause = ', expected_date = :new_expected_date';
    $update_params[':new_expected_date'] = $new_expected_date;
}

$update_sql = "
    UPDATE purchase_order SET
        status = :status
        {$update_expected_date_clause}
    WHERE
        po_id = :po_id AND supplier_id = :supplier_id
";
```

Figure 5.2.2.6: Supplier Purchase Order Status Update Execution

The system also allows suppliers to update the expected delivery date when the order is still in the Pending stage. This supports accurate planning between the supplier and internal procurement staff.

5.2.3 User Management Module

The User Module manages authentication, session handling, and role-based authorization throughout the system. It ensures that only verified users can access internal functions and that every action performed within the platform is properly recorded for audit purposes through the Activity Log. This module forms the core security layer that governs access to purchase orders, supplier records, settings, and system dashboards.

A. User Authentication

User authentication is implemented through a secure login handler that validates credentials and assigns the appropriate session data upon successful login. When a user submits the login form, the system checks the credentials against the database.

```php
// Check if it's a regular user
$stmt_user = $pdo->prepare(query: "SELECT user_id, username, role, email FROM user WHERE email = :email AND
password = :password AND status = 'Active'");
$stmt_user->execute(params: [':email' => $email, ':password' => $hashed_password]);
$user = $stmt_user->fetch(mode: PDO::FETCH_ASSOC);

if ($user) {
    // It's a user, set user session and redirect to the main dashboard
    $_SESSION['user'] = [
        'user_id' => $user['user_id'],
        'username'=> $user['username'],
        'role'    => $user['role'],
        'email'   => $user['email']
    ];

    header(header: 'Location: /index.php?page=dashboard');
    exit;
}
```

Figure 5.2.3.1: User Authentication Logic

This snippet authenticates a staff user and initializes a session containing their ID, role, and username. Once the session is created, the user gains access to internal system modules based on their assigned permissions.

B. Role-Based Authorization

The system uses a capability-based authorization model that maps each role (Admin, Manager, Staff) to a set of permissions. These permissions determine which actions a user is allowed to perform.

```php
public static function can(string $capability): bool
{
    $role = self::role();

    // List of permissions for each role
    $matrix = [
        'Admin'    => [
            'manage_suppliers',
            'manage_po',
            'view_po_list',
            'view_po_details',
            'create_po',
            'manage_po_status_all',
            'delete_po',
            'export_po',
            'manage_users',
            'view_logs',
            'view_users_list',
            'view_users_details',
            'view_notifications',
            'view_dashboard',
            'manage_settings',
            'view_reports',
            'view_archives',
            'view_items',
            'edit_items',
            'adjust_stock',
            'delete_items'
        ],
```

Figure 5.2.3.2: Admin Role Capability Mapping

```php
'Manager' => [
    'manage_suppliers',
    'manage_po',
    'view_po_list',
    'view_po_details',
    'create_po',
    'manage_po_status_all',
    'export_po',
    'view_logs',
    'view_users_list',
    'view_users_details',
    'view_notifications',
    'view_dashboard',
    'view_reports',
    'view_archives',
    'view_items',
    'edit_items',
    'adjust_stock',
    'delete_items'
],
```

Figure 5.2.3.3: Manager Role Capability Mapping

```
    'Staff'    => [
        'view_users_list',
        'view_notifications',
        'view_dashboard',
        'view_items',
        'edit_items',
        'adjust_stock',
        'delete_items',
        'view_po_list',
        'view_po_details',
        'create_po',
        'manage_po_status_basic',
        'export_po'
    ],
];
```

Figure 5.2.3.4: Staff Role Capability Mapping

This method checks whether the logged-in user has the necessary capability to access a particular function. It is used across critical modules such as supplier management, purchase order approval, user management, and system settings.

A page-specific protection method ensures that only valid staff users may access internal features:

```
/* --- Authorization Check --- */
Auth::check_staff(capabilities: ['manage_users']);
```

Figure 5.2.3.5: Staff Authorization Check for Module Access

This check verifies both the user's login state and their permission to view or execute the requested module.

C. Activity Log Integration

The User Module is integrated with the Activity Log system to record actions performed by authenticated staff. Each activity entry stores the user ID, action type, module name, IP address, and session ID.

```
// Log this action
ActivityLogger::log(pdo: $pdo, action_type: 'Update', module: 'User',
description: "Updated user '$username' (ID: $user_id)");
```

Figure 5.2.3.6: User Action Logging in Activity Log

This function collects the current user information from the session, captures the request metadata, and inserts a new record into the activity_log table:

```
$sql = "INSERT INTO activity_log (user_id, action_type, module, description, ip_address, session_id)
        VALUES (:user_id, :action_type, :module, :description, :ip_address, :session_id)";
```

Figure 5.2.3.7: SQL Insert Logic for Activity Log Recording

The Activity Log page is accessible only to authorised Admin and Manager users through the view_logs capability. After the system validates access using Auth::check_staff(['view_logs']), it retrieves activity entries in reverse chronological order and joins them with the user table so supervisors can see which account performed each action. Pagination is applied to maintain performance and readability.

```
$sql = "SELECT a.*, u.username
        FROM activity_log a
        LEFT JOIN user u ON a.user_id = u.user_id
        ORDER BY a.timestamp DESC
        LIMIT :limit OFFSET :offset";
$stmt = $pdo->prepare(query: $sql);
$stmt->bindParam(param: ':limit', var: &$items_per_page, type: PDO::PARAM_INT);
$stmt->bindParam(param: ':offset', var: &$offset, type: PDO::PARAM_INT);
$stmt->execute();
$logs = $stmt->fetchAll(mode: PDO::FETCH_ASSOC);
```

Figure 5.2.3.8: Activity Log Query for Admin and Manager View

This design allows Admin and Manager users to monitor staff actions, review operational changes, and maintain a verifiable audit trail, while standard Staff users only generate log entries through normal system usage and cannot access the log viewer.

5.2.4 Notification Module

The Notification Module manages all communication channels within the system by delivering both internal web-based alerts and external email notifications. It ensures that operational events such as purchase order status updates, low-stock detection, user or supplier profile modifications, and item management actions are communicated promptly to the correct parties. This module combines database-driven notification storage with AWS Simple Notification Service (SNS) to provide a unified, reliable, and scalable notification framework.

A. Internal Web Notification System

Internal notifications are stored in the database and displayed through the system's dashboard interface. These alerts are triggered by significant system events, including administrative actions, purchase order updates, and supplier-side changes. The notification engine supports targeted delivery to a specific user or broadcast delivery to an entire role group.

```php
try {
    if ($userId !== null) {
        // 1. Notify a specific user
        $userIds[] = $userId;
    } else {
        // 2. Notify all users in a role
        $sql_role = "SELECT user_id FROM user WHERE role = :role AND status = 'Active'";
        $stmt_role = $pdo->prepare(query: $sql_role);
        $stmt_role->execute(params: [':role' => $role]);
        $userIds = $stmt_role->fetchAll(mode: PDO::FETCH_COLUMN);
    }
```

Figure 5.2.4.1: Role-Based User Selection for Notification Delivery

This query identifies all active users belonging to a particular role, enabling the system to notify Admins, Managers, or Staff depending on the event.

```php
// 3. Insert notification for each user
$sql_insert = "
    INSERT INTO notification
        (user_id, title, message, link)
    VALUES
        (:user_id, :title, :message, :link)
";
```

Figure 5.2.4.2: Notification Record Insertion Logic

Each notification is saved with a title, descriptive message, and an optional link directing the user to relevant system pages such as purchase order details. These web notifications appear immediately when users access their dashboard.

A complete internal notification operation is performed using:

```php
$notif_msg = "PO #$po_id was created by $currentUsername.";
$notif_link = "/index.php?page=po_details&id=$po_id";
InternalNotify::send(pdo: $pdo, title: "New PO Created", message: $notif_msg, link: $notif_link, userId: null, role: "Admin");
InternalNotify::send(pdo: $pdo, title: "New PO Created", message: $notif_msg, link: $notif_link, userId: null, role: "Manager");
```

Figure 5.2.4.3: Helper Function for Internal Notification Sending

This helper function encapsulates user-role detection and database insertion, ensuring consistent and efficient notification handling throughout the system.

B. External Email Notification via AWS SNS

For events requiring immediate attention or off-platform communication, the module integrates AWS SNS to send email-based notifications. The sns_notify() function acts as the core mechanism for constructing and publishing messages to the SNS topic.

```
$sns = new SnsClient(args: $client_config);

$p = ['TopicArn'=>$cfg['topic_arn'],'Subject'=>$subject,'Message'=>$message];
if ($attrs) foreach ($attrs as $k=>$v)
  $p['MessageAttributes'][$k] = ['DataType'=>'String','StringValue'=>(string)$v];
try {
  $sns->publish(args: $p);
  return true;
}
catch (AwsException $e) {
  error_log(message: '[SNS] '.$e->getAwsErrorMessage());
  return false;
}
```

Figure 5.2.4.4: AWS SNS Email Notification Publishing Logic

The SNS client publishes structured messages that include relevant attributes, enabling filtering and categorization of events such as purchase order changes, stock alerts, and account management actions.

SNS is utilized across the system for multiple event types:

1. Purchase Order Status Changes

```
2 references
function po_notify_status_change(
    string $poId,
    string $rawStatus,
    string $actorEmail,
    string $actorRole,
    ?string $supplierEmail = null
): bool {
```

Figure 5.2.4.5: Purchase Order Status Change Notification Function

```php
$subject = "PO #$poId Status Updated: $status";

$message = "The status of a Purchase Order has been updated.\n\n"
         . "PO Number:  #$poId\n"
         . "New Status: $status\n"
         . "Updated By: $actorEmail\n"
         . "Actor Role: $actorRole\n";


// 3. Build the attributes array
$attributes = [
  'event'=>'PO_STATUS_CHANGED',
  'status'=>$status,
  'poId'=>$poId,
  'actorRole'=>$actorRole
];

// 4. Define supplier-facing statuses
$supplierStatuses = [
  'PENDING_APPROVAL',
  'APPROVED_BY_SUPPLIER',
  'REJECTED_BY_SUPPLIER',
  'CONFIRMED',
  'SHIPPED'
];
```

Figure 5.2.4.6: SNS Message Construction with Status Metadata

This function standardizes the purchase order status, attaches metadata, and dispatches an email notification to internal teams and suppliers.

2. Item Management Events

```php
3 references
function item_notify_low_stock(string $itemId, string $itemName, string $itemCode, int $newStock): bool {
    $subject = "Low Stock Alert: $itemName ($itemCode)";
    $message = "An item is running low on stock and may require reordering.\n\n"
             . "Item ID:   #$itemId\n"
             . "Item Name: $itemName\n"
             . "Item Code: $itemCode\n"
             . "Current Stock: $newStock\n";

    return sns_notify(subject: $subject, message: $message, attrs: [
        'event'       => 'LOW_STOCK',
        'severity'    => 'warning',
        'itemId'      => $itemId
    ]);
}
```

Figure 5.2.4.7: Low Stock Item SNS Notification Function

Low-stock and out-of-stock alerts are sent externally to ensure timely replenishment decisions.

3. Supplier and User Account Events

```php
1 reference
function supplier_notify_created(string $supplierId, string $supplierName, string $actorEmail, string $actorRole): bool {
    $subject = "New Supplier Created: $supplierName";

    $message = "A new supplier has been added to the system by an administrator.\n\n"
             . "Supplier ID:   #$supplierId\n"
             . "Supplier Name: $supplierName\n"
             . "Added By:      $actorEmail\n";

    return sns_notify(subject: $subject, message: $message, attrs: [
        'event'      => 'SUPPLIER_CREATED',
        'supplierId' => $supplierId,
        'actorRole'  => strtoupper(string: $actorRole),
        'severity'   => 'info'
    ]);
}
```

Figure 5.2.4.8: Supplier Account Creation Notification Function

```php
1 reference
function user_notify_updated(string $userId, string $username, string $actorEmail, string $actorRole): bool {
    $subject = "User Profile Updated: $username";

    $message = "A user's profile details were updated by an administrator.\n\n"
             . "User ID:   #$userId\n"
             . "Username:  $username\n"
             . "Updated By: $actorEmail\n";

    return sns_notify(subject: $subject, message: $message, attrs: [
        'event'      => 'USER_UPDATED',
        'userId'     => $userId,
        'actorRole'  => strtoupper(string: $actorRole),
        'severity'   => 'info'
    ]);
}
```

Figure 5.2.4.9: User Account Update Notification Function

These notifications inform Admins or Managers whenever supplier or user records are created, updated, or deleted.

Through SNS topic-based messaging and structured payloads, the system ensures reliable, scalable, and real-time distribution of critical updates.

C. Integration Between Internal and External Notifications

Many system events trigger both internal web notifications and external SNS email notifications to ensure complete communication coverage. The purchase order update workflow provides a clear example of this integration. When a user changes the status of a purchase order in update_po.php, the system executes the following logic:

```php
if ($new_status !== $current_status) {
    $notif_msg = "PO #$po_id status was updated to '$new_status' by $currentUsername.";
    $notif_link = "/index.php?page=po_details&id=$po_id";
```

Figure 5.2.4.10: Internal Notification Trigger for Purchase Order Status Update

```php
if ($new_status !== $current_status) {
    $notif_msg = "PO #$po_id status was updated to '$new_status' by $currentUsername.";
    $notif_link = "/index.php?page=po_details&id=$po_id";

    // Log this action
    ActivityLogger::log(pdo: $pdo, action_type: 'Update', module: 'PurchaseOrder', description: "Updated PO #$po_id status
    from '$current_status' to '$new_status'");

    // Send to Admin
    InternalNotify::send(pdo: $pdo, title: "PO Status Updated", message: $notif_msg, link: $notif_link, userId: null, role: "Admin");

    // Send to Manager
    InternalNotify::send(pdo: $pdo, title: "PO Status Updated", message: $notif_msg, link: $notif_link, userId: null, role: "Manager");

    // Send to Staff (conditionally)
    $staff_statuses = ['Received', 'Issue', 'Completed', 'Delayed', 'Shipped'];
    if (in_array(needle: $new_status, haystack: $staff_statuses, strict: true)) {
        InternalNotify::send(pdo: $pdo, title: "PO Status Updated", message: $notif_msg, link: $notif_link, userId: null, role: "Staff");
    }
}
```

Figure 5.2.4.11: Role-Based Internal Notification Distribution Logic

These internal notifications appear on the dashboard of Admins and Managers, while Staff users receive the alert only when the new status is operationally relevant (e.g., Received, Issue, Completed, or Shipped).

Once the database transaction is committed, the system proceeds with external email notification:

```php
// Send notification, passing supplier email
po_notify_status_change(
    poId: (string)$po_id,
    rawStatus: $new_status,
    actorEmail: $currentUserEmail,
    actorRole: $currentUserRole,
    supplierEmail: $supplierEmail
);
```

Figure 5.2.4.12: External SNS Email Notification Trigger for Purchase Order Status Change

This SNS-based notification ensures that suppliers and off-platform stakeholders receive immediate updates even when they are not logged into the system.

By combining internal dashboard alerts with external SNS email delivery, the Notification Module provides a consistent, reliable, and multi-channel communication mechanism across the entire procurement workflow.

5.3 Testing Strategies and Approaches

Testing is essential to verify that the Inventory Management System meets functional requirements and operates reliably on the AWS cloud. This section details the unit, integration, system, and user acceptance testing strategies employed to ensure critical features, such as automated reordering and real-time notifications, perform accurately under realistic conditions.

5.3.1 Testing Methodology

The testing phase employed a structured approach to validate the system's functionality, reliability, and performance. Four distinct levels of testing were conducted to ensure that the individual components under my responsibility, specifically Reordering, Supplier Management, User Management, and Notifications, worked correctly in isolation and as an integrated system.

1. Unit Testing

Unit testing focused on verifying the smallest testable parts of the server-side logic and cloud functions.

- Backend Logic: Individual PHP functions were tested to ensure data integrity. The login logic in login_handler.php was tested to verify that it correctly hashes input passwords using SHA256 and compares them against the stored database hash, successfully routing Regular Users to the main dashboard and Suppliers to the supplier portal. In the Reordering Module, the add_po.php script was tested to ensure it validates inputs, such as preventing the submission of a Purchase Order without selecting items or entering a valid delivery date.
- Cloud Functions: The AWS Lambda function (AutoPO_Generator) was tested using configured test events. Specific logic within the lambda_handler function was validated, such as the filtering algorithm that checks if stock_quantity + ordered_qty is less than the threshold_quantity. Additionally, the calculation logic restock_qty = item['threshold_quantity'] * 2 was verified to ensure the system automatically orders the correct amount to replenish inventory.

2. Integration Testing

Once individual units were verified, integration testing was conducted to ensure that different modules and AWS services communicated correctly.

- Database Integration: The connection between the PHP application and the AWS RDS MySQL database was tested to confirm that critical records were correctly manipulated. For example, the update_supplier.php script was tested to verify that submitting the form correctly executed the UPDATE supplier SQL query and that changes were immediately reflected in the supplier table.

- Cloud Service Integration: A key focus was testing the interaction between the serverless backend and the database. The AutoPO_Generator Lambda function was executed to verify it could successfully establish a connection to the RDS instance using environment variables (DB_HOST, DB_USER). Tests confirmed that the function could query the item table, group low-stock items by supplier_id, and successfully insert new records into the purchase_order, purchase_order_details, and notification tables within a single execution cycle.

3. System Testing

System testing involved validating the complete, integrated workflows to ensure they met the functional requirements defined in Chapter 3.

- End-to-End Workflows: Complete procurement processes were executed to ensure system cohesion. The Manual Reordering Workflow was tested by logging in as a Staff member, navigating to the "New Purchase Order" page, submitting a request, and verifying that the system redirected to the dashboard with a success status. Simultaneously, the Automated Reordering Workflow was tested by triggering the Lambda function and confirming that an Admin user received an internal "Auto-Generated PO" notification with a direct link to the new order.
- Role-Based Access Control: The User Management Module was tested to ensure permissions were strictly enforced. Tests confirmed that unauthorized access attempts to restricted pages (such as a 'Staff' user trying to access 'Manager' functions) were successfully blocked, redirecting the user appropriately.

4. User Acceptance Testing (UAT)

The final phase, User Acceptance Testing, focused on usability and validating that the specific deliverables met user expectations.

- Requirement Validation: The system was evaluated against the project scope. This included verifying that the system correctly handled multiple items from the same supplier by grouping them into a single Purchase Order during automation, and that the activity_log correctly captured the IP address and Session ID for security auditing.
- Usability Check: The interfaces were tested to ensure navigation was intuitive and feedback was clear. For instance, the Login page was tested to confirm that entering incorrect credentials triggered an error flag (error=1) in the URL, prompting the interface to display a clear "Invalid username or password" message to the user.

5.3.2 Test Cases

The following test cases were executed to validate the system's functionality across all core modules. Each test case details the specific steps taken, the expected outcome based on the system design, and the actual result observed during testing.

1. Reordering Module

Table 5.3.2.1: Test Cases for Reordering Module

Test ID	Test Description	Test Steps	Expected Result	Actual Result	Status
TC-01	Create Manual Purchase Order (Success)	1. Login as Manager. 2. Open "New Purchase Order" modal. 3. Select Supplier "Alpha Supplies" and Date. 4. Add Item "Maggi" (Qty: 50). 5. Click "Create Purchase Order".	System validates inputs via add_po.php, creates PO, and redirects to a list with the message: "Purchase Order created successfully." [cite: index.php].	PO created and a success toast appeared.	Pass
TC-02	Create PO Validation (No Item)	1. Open "New Purchase Order" modal. 2. Select Supplier and Date. 3. Do not add any items. 4. Click "Create Purchase Order".	Frontend JS validation prevents submission and displays an alert: "Please add at least one item to the order." [cite: index.php JS].	Alert "Please add at least one item to the order." displayed.	Pass
TC-03	Approve Purchase Order	1. Login as Manager. 2. View details of a 'Pending' PO. 3. Change status to 'Approved' and click "Save".	update_po.php validates manage_po_status_all permission, updates status, and triggers SNS email.	Status updated to 'Approved'; email sent.	Pass

TC-04	Delete PO Constraint	1. Login as Admin. 2. Attempt to delete a PO with status 'Approved' or 'Received'.	delete_po.php blocks deletion (only 'Created'/'Pending' allowed) and redirects with error: "That order cannot be deleted, it may already be processed." [cite: index.php].	Deletion failed; correct error message shown.	Pass
TC-05	Export PO to PDF	1. Open PO Details. 2. Click "Export to PDF".	export_po.php generates a PDF file named PO-[ID].pdf using the FPDF library.	PDF file downloaded successfully.	Pass
TC-06	Auto-Receive Logic	1. Login as Staff. 2. Open a 'Shipped' PO. 3. Update status to 'Received'.	update_po.php automatically creates a corresponding Goods Receipt record and updates the database.	Status changed; Goods Receipt created automatically.	Pass

2. Supplier Management Module

Table 5.3.2.2: Test Cases for Supplier Management Module

Test ID	Test Description	Test Steps	Expected Result	Actual Result	Status
TC-07	Add New Supplier (API Success)	1. Login as Admin. 2. Send POST request to /api/add_supplier.php with valid company_name, email, etc.	System validates input, inserts record, and returns JSON: {"status": "success", "message": "Supplier created."}.	JSON success response received.	Pass
TC-08	Add Supplier Validation (Duplicate)	1. Send a POST request with an email that already exists in the supplier table.	System detects duplicate via SQL check and returns JSON 409: {"status": "error", "message": "Email is already used by another supplier."} [cite: add_supplier.php].	Error 409 received with correct message.	Pass

TC-09	Delete Supplier Constraint	1. Login as Admin. 2. Attempt to delete a supplier linked to existing POs via /api/delete_supplier.php.	Database throws foreign key constraint error; PHP catches exception and redirects to: /index.php?page=suppliers&status=delete_error [cite: delete_supplier.php].	Redirected with status=delete_error.	Pass
TC-10	Supplier Portal Profile Update	1. Login to Supplier Portal. 2. Submit form to profile_handler.php with new phone number.	System validates session, updates record, and redirects to: /index.php?page=supplier_profile&status=updated.	Profile updated and user redirected.	Pass
TC-11	Supplier Update PO (Valid Flow)	1. Login as Supplier. 2. Submit POST to supplier_update_po.php changing PO status from 'Pending' to 'Approved'.	System validates flow map (Pending → Approved is allowed), updates DB, and redirects with status=updated [cite: supplier_update_po.php].	Status updated successfully.	Pass
TC-12	Supplier Update PO (Invalid Flow)	1. Login as Supplier. 2. Attempt to change a 'Pending' PO directly to 'Shipped'.	System checks $supplier_flow_map, detects invalid transition, and redirects with status=error_invalid_status_change.	Update blocked; error status returned.	Pass

3. User Management Module

Table 5.3.2.3: Test Cases for User Management Module

Test ID	Test Description	Test Steps	Expected Result	Actual Result	Status
TC-13	Add New User (Success)	1. Login as Admin. 2. Send POST request to /api/add_user.php with valid username, email, role, etc.	System inserts a new user record, logs activity via ActivityLogger, and returns JSON: {"status": "success", "message": "User created successfully."}.	JSON success response received.	Pass
TC-14	Add User Validation (Duplicate)	1. Send a POST request with a username or email that already exists.	System detects duplicates via SQL check (SELECT user_id ... LIMIT 1) and returns JSON 409: {"status": "error", "message": "Username or email is already in use."} [cite: add_user.php].	Error 409 received with correct message.	Pass
TC-15	Update User Profile	1. Login as Admin. 2. Submit form to /api/update_user.php changing the user's role from 'Staff' to 'Manager'.	System updates the record and logs the action: "Updated user [username] (ID: [id])" [cite: update_user.php].	Role updated and activity logged.	Pass
TC-16	Delete User	1. Login as Admin. 2. Send GET request to /api/delete_user.php?id=[ID].	System fetches username for logging, deletes the record, logs the action "Deleted user [username]", and redirects to user list [cite: delete_user.php].	User deleted and action logged.	Pass
TC-17	Activity Log Access Control	1. Login as 'Staff'. 2. Attempt to view the Activity Log page.	Auth::check_staff(['view_logs']) denies access because the 'Staff' role lacks this permission (only Admin/Manager allowed).	Access denied; user redirected.	Pass

TC-18	Verify Activity Log Entry	1. Perform an action (e.g., Add User). 2. Login as Admin and check activity_log table.	The table contains a new row with the correct user_id, action_type='Add', module='User', and the current ip_address [cite: ActivityLogger.php].	Log entry found with correct details.	Pass

4. Notification Module

Table 5.3.2.4: Test Cases for Notification Module

Test ID	Test Description	Test Steps	Expected Result	Actual Result	Status
TC-19	Internal Notification (PO Created)	1. Create a new PO as Admin. 2. Check the notification table in the database.	InternalNotify::send() inserts a new record for Admin/Manager with title "New PO Created" and correct link (/index.php?page=po_details&id=...).	Notification record found in database.	Pass
TC-20	External SNS Notification (Low Stock)	1. Manually reduce Item ID 4 stock to 0. 2. Trigger ItemNotify::item_notify_out_of_stock() via script.	System sends an SNS email with the subject "Out of Stock Alert: Horlicks (ITM-1004)" and severity 'critical'.	Email received via AWS SNS.	Pass
TC-21	PO Status Change Alert	1. Manager updates PO #25 to 'Confirmed'. 2. Check Supplier email inbox.	po_notify_status_change() triggers SNS email to supplier with subject "PO #25 Status Updated: CONFIRMED".	Supplier received email notification.	Pass
TC-22	Mark Single Notification as Read	1. Send GET request to /api/mark_notifications.php?action=mark_read&id=[ID].	API updates is_read=1 for that notification ID and returns {"status": "success", "message": "Notification marked as read"}.	JSON success response; DB updated.	Pass

TC-23	Mark All Notifications as Read	1. Send GET request to /api/mark_notifications.php?action=mark_all_read.	API updates is_read=1 for all notifications belonging to the current user.	All user notifications marked as read.	Pass
TC-24	Supplier Created Alert	1. Add a new Supplier via Admin portal. 2. Check the Admin email inbox.	supplier_notify_created() triggers SNS email with subject "New Supplier Created: [Name]" and event type SUPPLIER_CREATED.	Admin received email notification.	Pass

5.4 Chapter Summary and Evaluation

This chapter detailed the complete implementation and testing of the Business Process Automation System for Inventory Management. The system infrastructure was successfully established on the AWS Cloud, utilizing a serverless architecture powered by AWS Lambda, Amazon RDS, and AWS SNS. CloudWatch was configured for continuous monitoring, ensuring high availability.

The implementation phase focused on the four core modules: Reordering, Supplier Management, User Management, and Notification Services. Key achievements include the automated stock replenishment via the AutoPO_Generator Lambda function, a secure Supplier Portal for external updates, and strict role-based access control (RBAC) for user management. Additionally, a dual-channel notification system was established to deliver real-time alerts via both dashboard and email.

The testing phase validated the system through unit, integration, system, and user acceptance testing. A comprehensive suite of test cases confirmed compliance with functional requirements, including input validation and seamless procurement workflows. The system successfully demonstrated capabilities like preventing unauthorized deletions and automating goods receipt generation.

Overall, the successful implementation confirms that the system effectively automates inventory tasks and enhances data visibility for SMEs. The serverless design ensures cost-efficiency and scalability, while rigorous testing guarantees a stable platform. The next chapter will discuss the project's overall achievements, contributions, and potential future enhancements.

Chapter 6

Discussions and Conclusion

6 Discussions and Conclusion

6.1 Summary

The Business Process Automation System for Inventory Management was developed to address the critical operational inefficiencies faced by small and medium-sized enterprises (SMEs), specifically the reliance on manual stock tracking, the high risk of human error in procurement, and the lack of real-time data visibility. The proposed solution successfully digitized these processes through a cloud-native web application that automates stock monitoring, purchase order generation, and supplier communication. By centralizing these functions on a secure cloud platform, the project has transformed disjointed manual tasks into a cohesive, automated workflow that enhances operational speed and accuracy.

The choice of tools and methodologies was pivotal to the project's success. The Waterfall Model was selected as the development methodology because the functional requirements such as threshold-based reordering and supplier management were clearly defined during the planning phase, allowing for a structured and systematic development process from design to deployment. Technologically, the decision to utilize Amazon Web Services (AWS) provided a robust and scalable infrastructure without the high upfront costs of physical servers. Specifically, the use of AWS Lambda for serverless computing allowed for cost-effective automation of backend logic, while Amazon RDS ensured reliable relational data storage. PHP and MySQL were chosen for the application layer due to their compatibility with web hosting environments and ease of integration with AWS SDKs, proving to be an effective combination for building a responsive and accessible solution for SMEs.

6.2 Achievements

The project has successfully achieved its primary objectives of automating core inventory operations and enhancing visibility for business owners. A major achievement is the successful implementation of the automated reordering mechanism, where the system is capable of detecting low stock levels and auto-generating purchase orders without manual intervention. The development of the Supplier Management and User Management modules has also streamlined the administrative workflow, ensuring that supplier data is centralized and user access is securely controlled through role-based permissions. Furthermore, the integration of the Notification Module ensures that stakeholders are kept informed of critical system events, significantly reducing the reaction time to potential stockouts.

In terms of strengths, the system demonstrates high reliability in its core automation logic. The logic defined within the AWS Lambda functions operates accurately to process inventory data and trigger necessary actions. The system also excels in data accessibility, as the cloud-based deployment allows

users to access the dashboard and reports from any location with internet connectivity. However, a weakness identified during the development is the complexity of managing cloud credentials within a temporary educational environment, which required frequent manual updates to maintain connectivity. Despite this, the project stands as a fully functional prototype that demonstrates the power of cloud automation in solving real-world business problems.

6.3 Contributions

This project contributes significantly to the digital transformation of SMEs by offering a lightweight, cost-effective alternative to complex Enterprise Resource Planning (ERP) systems. The proposed system is necessary because many local businesses cannot afford the high licensing fees and infrastructure costs associated with traditional ERPs, yet they urgently need automation to remain competitive. By utilizing a serverless architecture, this project introduces an innovative approach that democratizes access to advanced inventory automation, allowing small businesses to pay only for the compute resources they use.

The creativity of the solution lies in its integration of disparate cloud services into a unified workflow. Unlike standalone inventory software that runs locally, this system leverages the power of the cloud to link stock monitoring directly with procurement actions. The marketability of the system is high, as it addresses a universal pain point inventory mismanagement with a solution that is scalable, secure, and accessible. It empowers business owners to make data-driven decisions through real-time reporting, transforming inventory management from a reactive burden into a proactive strategic advantage.

6.4 Limitations and Future Improvements

Despite the successful implementation of core features, the project faced several limitations primarily due to the restrictions of the deployment environment. A significant limitation was the inability to fully implement AWS Identity and Access Management (IAM) roles and the AWS Backup service. This was because the system was developed using the AWS Academy Learner Lab, where the restricted LabRole policy did not grant the necessary permissions to configure custom IAM policies or initiate specific AWS Backup jobs. Consequently, the system relies on the default lab roles, which would not be sufficient for a commercial production environment requiring granular security controls.

Another critical limitation occurred within the notification workflow. While the internal system notifications function correctly and are successfully stored in the database, the AWS Lambda function responsible for auto-generating purchase orders failed to trigger external email notifications via AWS SNS. This failure was traced back to the LabRole lacking the specific "sns:Publish" permissions required for Lambda to interact with SNS directly in the learner environment. Furthermore, the

development team faced continuous challenges with the aws.php configuration file. Due to the temporary nature of the learner lab environment, the session tokens and secret keys expired frequently, necessitating manual updates to the configuration file to maintain the connection between the PHP application and AWS services.

Future improvements should focus on migrating the system to a commercial AWS account to overcome these policy restrictions. This would allow for the proper implementation of automated backups via AWS Backup and the configuration of specific IAM roles for tighter security. Additionally, the system could be enhanced by integrating it with external sales and procurement systems. Bridging these systems would create a more comprehensive and well-rounded solution, ensuring that sales data automatically updates inventory levels in real-time, thereby providing a holistic view of the business's operational health. The scope could also be expanded to include demand forecasting algorithms that use historical sales data to dynamically adjust reorder thresholds, further enhancing the automation capabilities of the system.

6.5 Issues and Solutions

During the development of the project, several technical challenges were encountered that required strategic problem-solving. One major issue was determining how to reliably automate the stock checking process without user intervention. The initial concept of running a continuous script was inefficient and costly. To solve this, the team implemented AWS CloudWatch to set up a cron job. This solution successfully triggers the Lambda function daily at a specific time (9:00 AM Malaysia Time), ensuring that stock levels are checked and purchase orders are generated consistently every day without requiring manual initiation.

Another technical challenge was generating professional and printable purchase orders directly from the web interface. The raw HTML output was not suitable for formal business documents. To address this, the team integrated the FPDF library into the PHP application. This library allowed for the dynamic generation of PDF documents, formatting the purchase order details into a professional layout that includes company branding and itemized tables, which can be easily downloaded and sent to suppliers.

Finally, ensuring data safety was a concern given the restrictions on the AWS Backup service within the Learner Lab. Although the team could not utilize the dedicated AWS Backup service due to permission errors, a solution was found by configuring Amazon RDS to perform regular automated backups and snapshots. This ensured that despite the limitations of the educational environment, the system maintained a reliable disaster recovery mechanism to prevent data loss in the event of a

breakdown. The integration of the AWS SDK also solved the general connectivity issues, providing a stable bridge between the application code and the various AWS services used.

6.6 Conclusion

In conclusion, the Business Process Automation System for Inventory Management successfully modernized SME operations by transitioning manual workflows into a scalable, cloud-native solution on AWS. Despite facing environmental constraints within the AWS Academy Learner Lab, particularly regarding IAM roles and automated backups, the project demonstrated technical resilience by implementing strategic alternatives like AWS CloudWatch for scheduling and the FPDF library for professional document generation. This development journey not only refined technical expertise in serverless architecture and system integration but also delivered a robust, cost-effective tool that empowers businesses with real-time visibility and operational efficiency, laying a strong foundation for future enhancements such as procurement and sales system integration.

References

AWS. (n.d.-a). *Amazon CloudWatch Pricing – Amazon Web Services (AWS)*. Amazon Web Services, Inc. Retrieved July 14, 2025, from https://aws.amazon.com/cloudwatch/pricing/

AWS. (2024a). *Amazon S3 Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/s3/pricing/

AWS. (2024b). *Amazon Simple Notification Service (SNS) Pricing | Messaging Service | AWS*. Amazon Web Services, Inc. https://aws.amazon.com/sns/pricing/

AWS. (2024c). *AWS Lambda – Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/lambda/pricing/

AWS. (2025). *Amazon Aurora Pricing | MySQL PostgreSQL Relational Database | Amazon Web Services*. Amazon Web Services, Inc. https://aws.amazon.com/rds/aurora/pricing/

Pricing. (n.d.-b). Amazon Web Services, Inc. https://aws.amazon.com/ec2/pricing/

Benedict, S., Rubiya Subair, Gupta, T., & Vedanta S.P. (2023). Penalty-Enabled Serverless Architecture for Cloud-based Startup Solutions. *Penalty-Enabled Serverless Architecture for Cloud-Based Startup Solutions*. https://doi.org/10.1109/icict57646.2023.10134026

Harvey Norman. (2025). *Asus Vivobook 15 i5-1335U/8GB/512GB SSD 15.6-inch FHD Laptop - Cool Silver (A1504V-ABQ353WS)*. Harveynorman.com.my. https://www.harveynorman.com.my/computing/computers-en/laptops-en/asus-vivobook-15-i5-1335u-8gb-512gb-ssd-15.6-inch-fhd-laptop-cool-silver-a1504v-abq353ws-en.html

Hypernix Sdn Bhd. (2025, June 18). *Oracle NetSuite Enterprise Resource Planning (ERP) System Malaysia*. Hypernix. https://www.hypernix.net/oracle-netsuite-erp/

Indeed. (2025). *Information technology intern salary in Malaysia*. Indeed.com; Indeed. https://malaysia.indeed.com/career/information-technology-intern/salaries

inFlow Inventory. (2025b, May 13). *Inventory Management Software - Made Easy - InFlow*. https://www.inflowinventory.com/

Inventory management software | Online inventory management for businesses - Zoho Inventory. (n.d.). [Video]. https://www.zoho.com/inventory/

Madamidola, O., Daramola, O., Akintola, K., & Adeboje, O. (2024). A Review of Existing Inventory Management Systems. *International Journal of Research in Engineering and Science*, *12*(9), 40–50. https://www.researchgate.net/publication/383947700_A_Review_of_Existing_Inventory_Management_Systems

Naveed S, Janani Sri R, & Prasad B. (2024). IoT-Enhanced Hostel Inventory Management System for Seamless Resource Monitoring and Control. *IoT-Enhanced Hostel Inventory Management System for Seamless Resource Monitoring and Control*. https://doi.org/10.1109/incos59338.2024.10527467

Pangaribuan, J. J., Margono, H., Barus, O. P., Pratama, Y. A., & Maulana, A. (2022, September 1). *Sales, Purchase, and Inventory Information System Design at SMEs*. IEEE Xplore. https://doi.org/10.1109/ICTIIA54654.2022.9935929

PC Image. (2025). *Microsoft Windows 10 Home 64 bit OEM pack (KW9-00139) | PC Image*. Pcimage.com.my; PC Image.

https://www.store.pcimage.com.my/microsoft-windows-10-home-64bit-oem-pack-kw9-00139?tag=Windows

Sharma, A., & Guleria, K. (2021, March 1). *A Framework for Hotel Inventory Control System for Online Travel Agency using Robotic Process Automation*. IEEE Xplore. https://doi.org/10.1109/ICACITE51222.2021.9404613

Time. (2019). *TIME 100% Fibre Optic Network | Data - Internet - Voice*. TIME 100% Fibre Optic Network | Data - Internet - Voice. https://www.time.com.my/

Appendices

Appendix A: User Guide

A.1 System Document

This section details the hardware and software environment required to access and operate the Business Process Automation System for Inventory Management effectively.

Hardware Requirements

- **Client Side:** Any standard laptop, desktop computer, or tablet with a stable internet connection. High processing power is not required as the system is cloud-hosted.
- **Server/Cloud Side:** An active Amazon Web Services (AWS) account subscribed to the following services:
    - **Amazon EC2:** For hosting the web application server.
    - **Amazon RDS:** For hosting the MySQL relational database.
    - **AWS Lambda:** For executing serverless background automation.
    - **Amazon S3:** For storing generated documents and static assets.
    - **Amazon SNS:** For delivering real-time email notifications.
    - **Amazon CloudWatch:** For monitoring system logs and scheduling cron jobs.

Software Requirements

- **Web Browser:** Google Chrome, Mozilla Firefox, or Microsoft Edge (latest versions recommended for optimal compatibility).
- **Development Tools (for Admin/Developers):**
    - **Visual Studio Code:** For editing source code (PHP, HTML, JavaScript, Python).
    - **MySQL Workbench:** For managing the RDS database connection.
    - **Composer:** Required for installing AWS SDK for PHP.
    - **Draw.io:** For viewing or editing system diagrams.

A.2 Installation / Deployment Guide

Since this system is a web-based application hosted on the cloud, traditional installation via executable files is not required. Instead, the following steps outline the deployment and access procedures.

Accessing the Application

1. Ensure the EC2 instance is running on the AWS console.
2. Open a web browser and navigate to the public IP address or domain URL provided for the system: http://54.163.82.194.

Uploading the System to EC2 Using FileZilla Because the system is hosted on an Amazon EC2 instance, FileZilla was used to securely transfer all PHP files, assets, and configuration files to the server.

1. Open FileZilla - Launch the FileZilla Client on your computer.
2. Connect to EC2 In the Quickconnect bar:
    - Host: ec2-user@54.163.82.194
    - Port: 22
    - Username: ec2-user
    - Import your EC2 .pem key under Settings → SFTP → Add key file
    - Click Quickconnect.
3. Open Web Directory On the EC2 panel (right side), navigate to:
    - /var/www/html ←This is where the website files must be uploaded.
4. Upload Project Files
    - On the left panel, open your project folder.
    - Drag all PHP, CSS, JS, and vendor files into /var/www/html.
5. View the System
    - Open a browser and visit:
        - http://54.163.82.194/
        - system should now be running on EC2.

Database Connection To manage the database backend, administrators must connect via MySQL Workbench using the following credentials:

- **Endpoint:** database-1.cxjo8lexejtp.us-east-1.rds.amazonaws.com
- **Port:** 3306
- **Username:** admin
- **Password:** 12345678

AWS Configuration Setup Due to the security policies of the AWS Academy Learner Lab, session tokens expire periodically. To ensure continuous connectivity between the PHP application and AWS services (SNS, S3), the configuration file must be updated whenever the lab session restarts.

1. Navigate to the config/ directory in the source code.
2. Open the aws.php file.
3. Update the access_key, secret_key, and session_token with the new credentials provided by the AWS Learner Lab console.

A.3 Operation Document (User Manual)

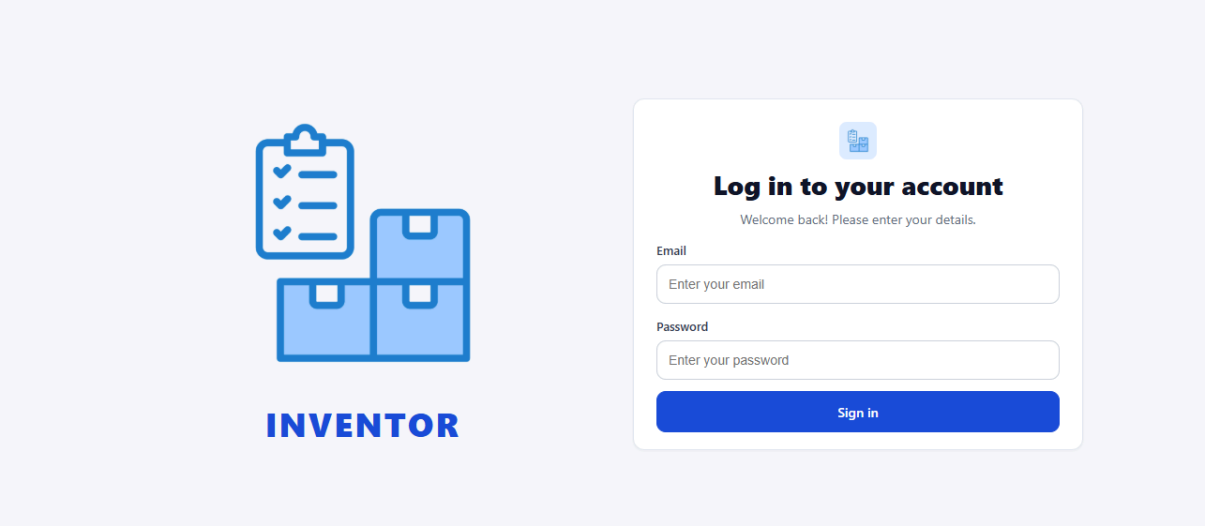
This section provides step-by-step instructions for operating the key modules of the system.

Login Credentials The following test accounts have been created for demonstration purposes:

- Admin: Email: leesj-wm22@student.tarc.edu.my / Password: 12345678
- Manager: Email: limyy-wm22@student.tarc.edu.my / Password: 12345678
- Staff: Email: lilylim0105@gmail.com / Password: 12345678
- Supplier: Email: siawjing.lee@gmail.com / Password: 12345678

1. Login Page

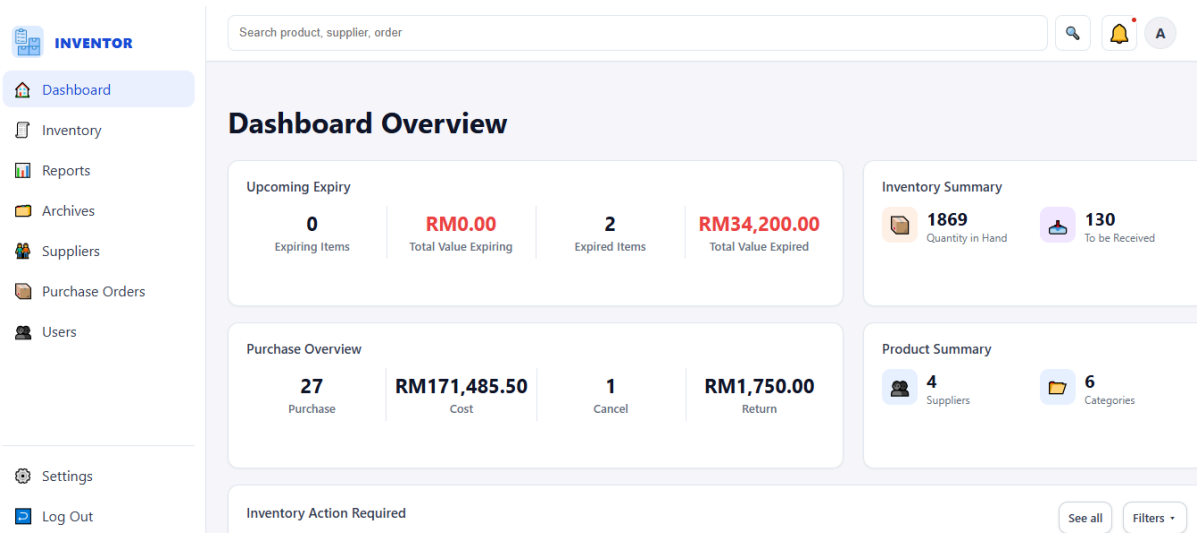
- **Description:** The secure entry point for all users.
- **Instructions:** Enter your username and password. Click the "Login" button. If the credentials are valid, you will be redirected to the dashboard corresponding to your role.



Appendix A1: Login Page

2. Inventory Dashboard

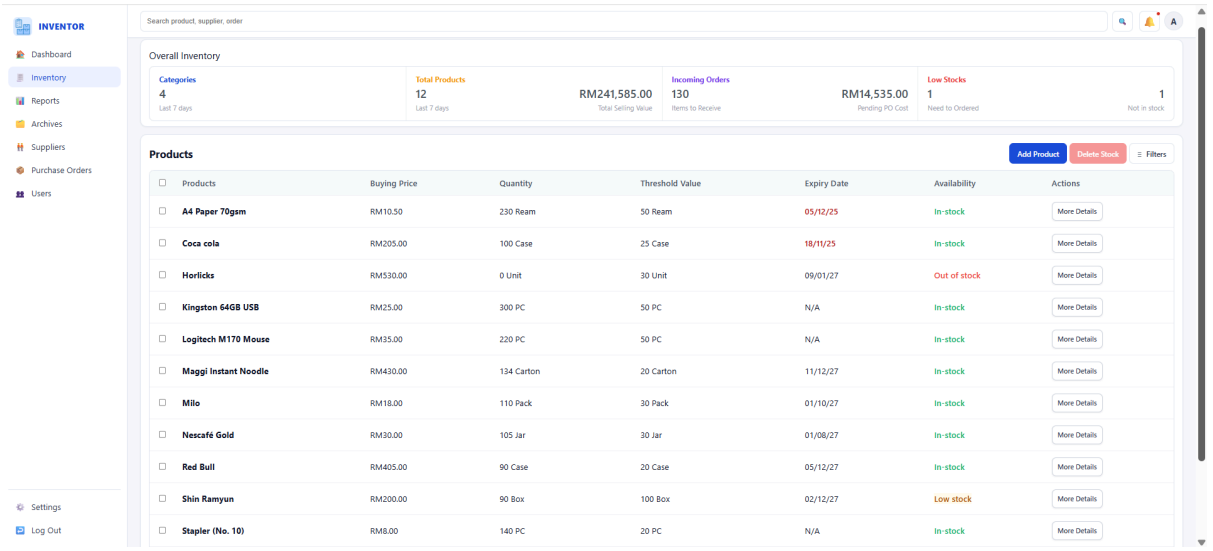
- **Description:** Displays a high-level overview of the inventory status.
- **Instructions:** View the "Low Stock Alerts" widget to identify items needing reordering. The "Recent POs" table shows the status of the latest purchase orders.



Appendix A2: Inventory Dashboard Page

3. Stock Monitoring Module

- **Description:** View and manage all stock items.
- **Instructions:**
    1. Navigate to Stock Monitoring → Stock List.
    2. View item quantity, expiry date, and availability badges.
    3. Click Details to view full item information.
    4. Use Add Item or More Details to update records.

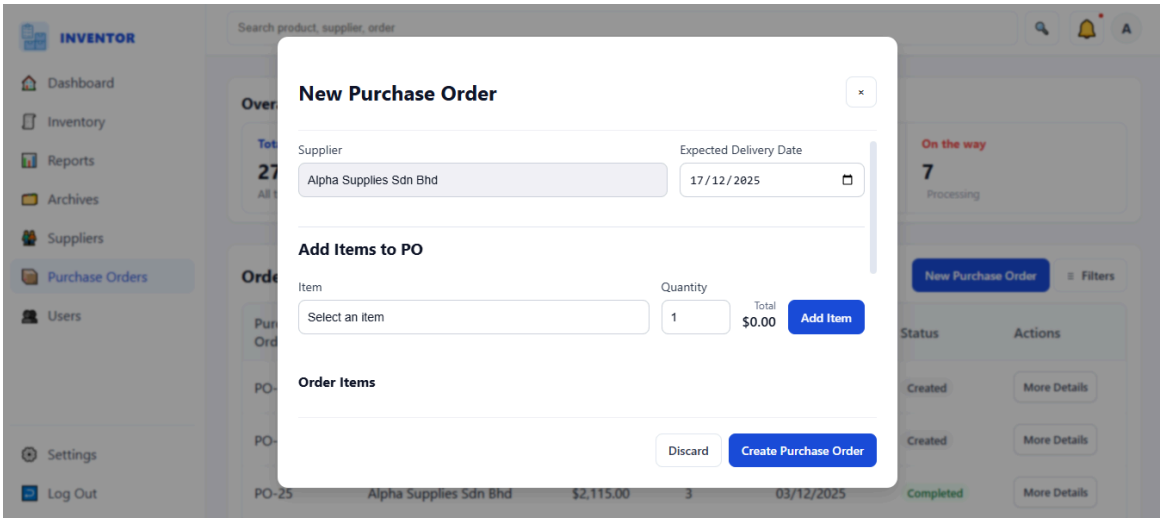


Appendix A3: Stock Monitoring Page

4 . Creating a Manual Purchase Order

- **Description:** Allows staff to manually raise a PO for specific items.

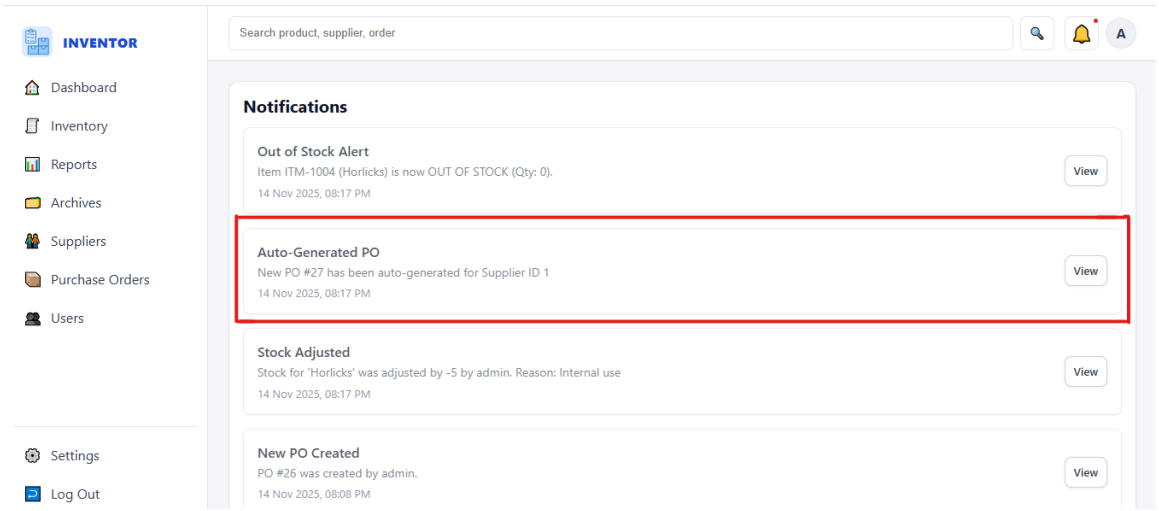
- **Instructions:**
    1. Navigate to the Reordering Module.
    2. Click "New Purchase Order".
    3. Select a "Supplier" from the dropdown list.
    4. Add items to the order and specify quantities.
    5. Click "Submit" to send the PO for approval.



Appendix A4: New Purchase Order Page

5. Auto-PO Generation Logs

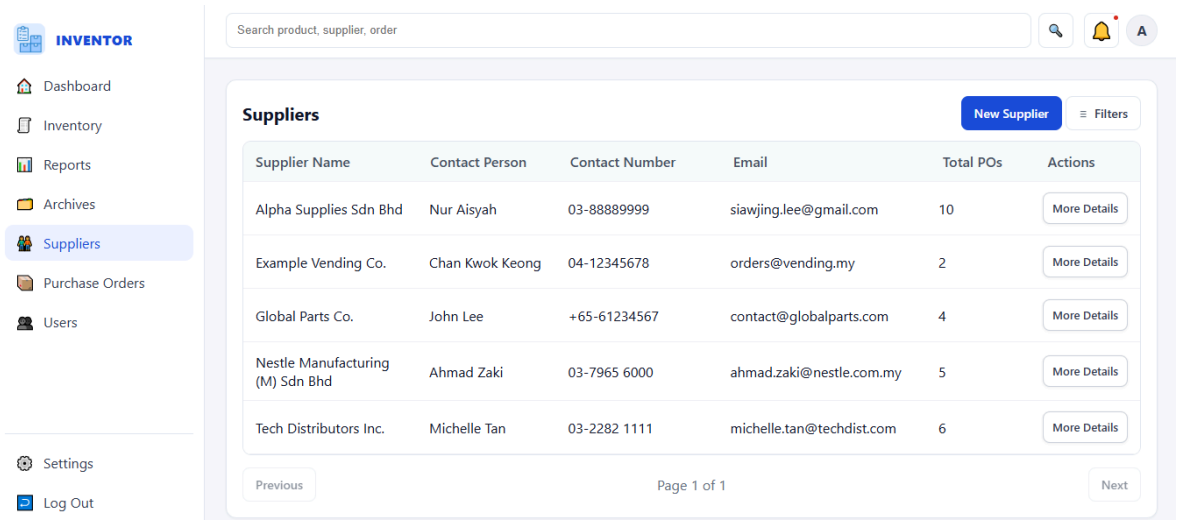
- **Description:** Verifying the automated generation of POs by the AWS Lambda function.
- **Instructions:** This process runs automatically at 9:00 AM daily via a cron job. To verify execution, Admins can view the Activity Log or check the Purchase Order List for new entries marked as "Auto-Generated."



Appendix A5: Auto-PO Generation Activity Logs

6. Supplier Management

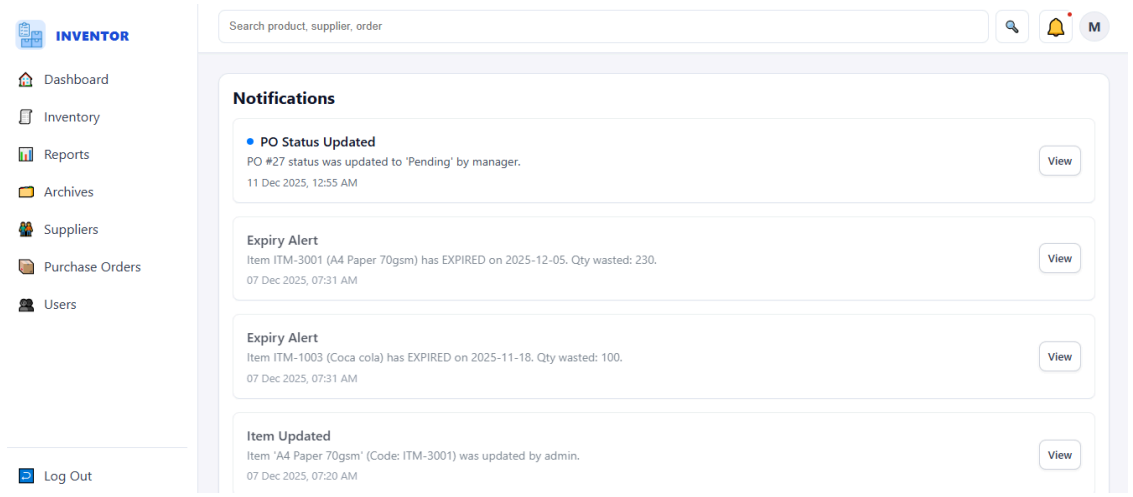
- **Description:** Managing supplier profiles and contact details.
- **Instructions:**
    1. Go to the Supplier Management tab.
    2. Click "Add Supplier" to create a new profile or "Edit" to modify an existing one.
    3. Fill in the company name, contact person, and email address.
    4. Save the changes.



Appendix A6: Supplier Management Page

7. Notifications

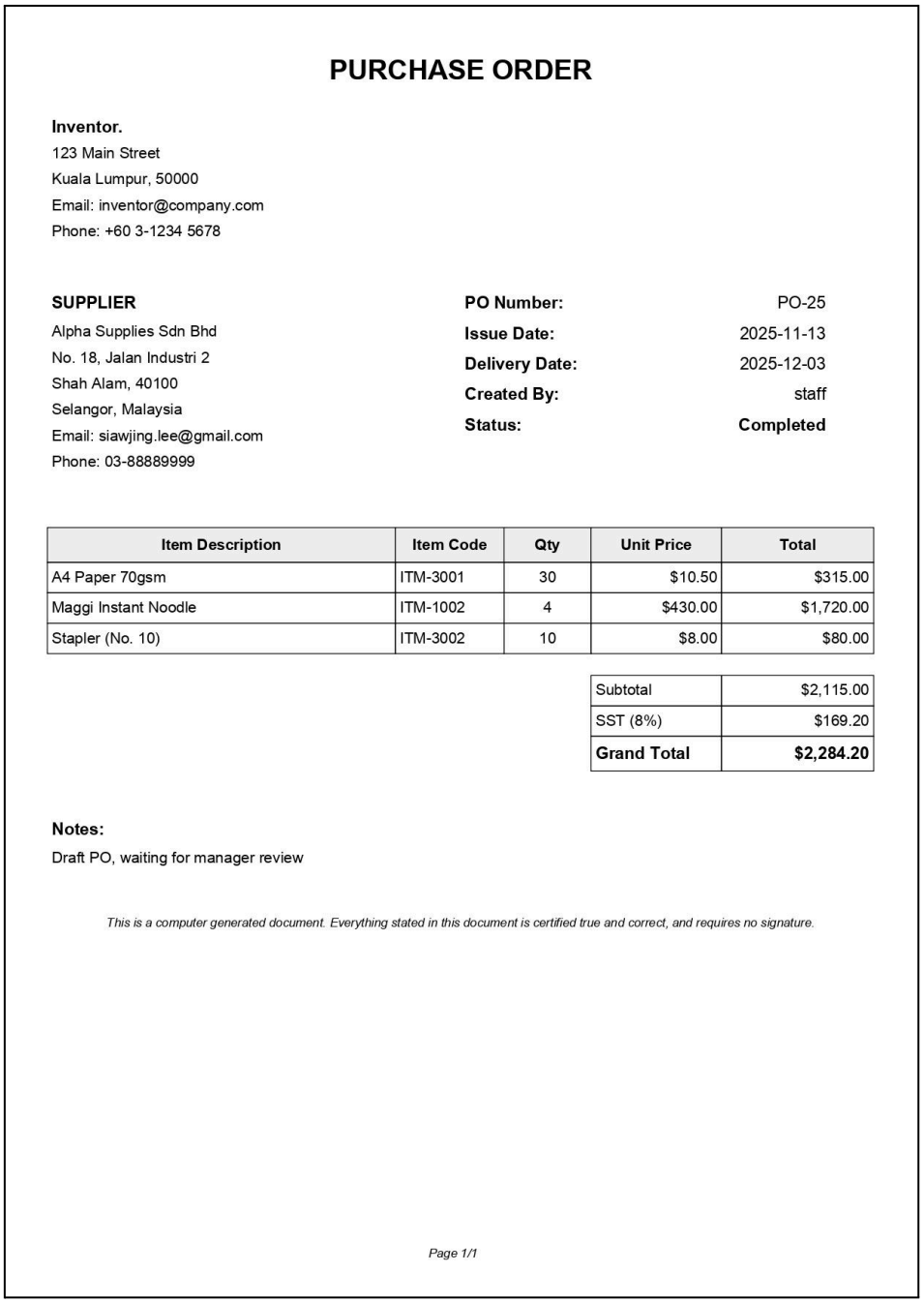
- **Description:** Viewing real-time system alerts.
- **Instructions:** Click the "Bell" icon in the top navigation bar to view internal alerts. For external notifications, check the registered email inbox for alerts sent via AWS SNS regarding low stock or PO status changes.



Appendix A7: Notifications Page

8. Generating Reports

- **Description:** Exporting system data for offline use.
- **Instructions:**
    1. Navigate to the Reports section or a specific Purchase Order.
    2. Click the "Export to PDF" button.
    3. The system will generate and download a formatted PDF document.
- Exp: http://54.163.82.194/api/export_po.php?id=25



PURCHASE ORDER

Inventor.
123 Main Street
Kuala Lumpur, 50000
Email: inventor@company.com
Phone: +60 3-1234 5678

SUPPLIER
Alpha Supplies Sdn Bhd
No. 18, Jalan Industri 2
Shah Alam, 40100
Selangor, Malaysia
Email: siawjing.lee@gmail.com
Phone: 03-88889999

PO Number: PO-25
Issue Date: 2025-11-13
Delivery Date: 2025-12-03
Created By: staff
Status: **Completed**

Item Description	Item Code	Qty	Unit Price	Total
A4 Paper 70gsm	ITM-3001	30	$10.50	$315.00
Maggi Instant Noodle	ITM-1002	4	$430.00	$1,720.00
Stapler (No. 10)	ITM-3002	10	$8.00	$80.00

Subtotal	$2,115.00
SST (8%)	$169.20
Grand Total	**$2,284.20**

Notes:
Draft PO, waiting for manager review

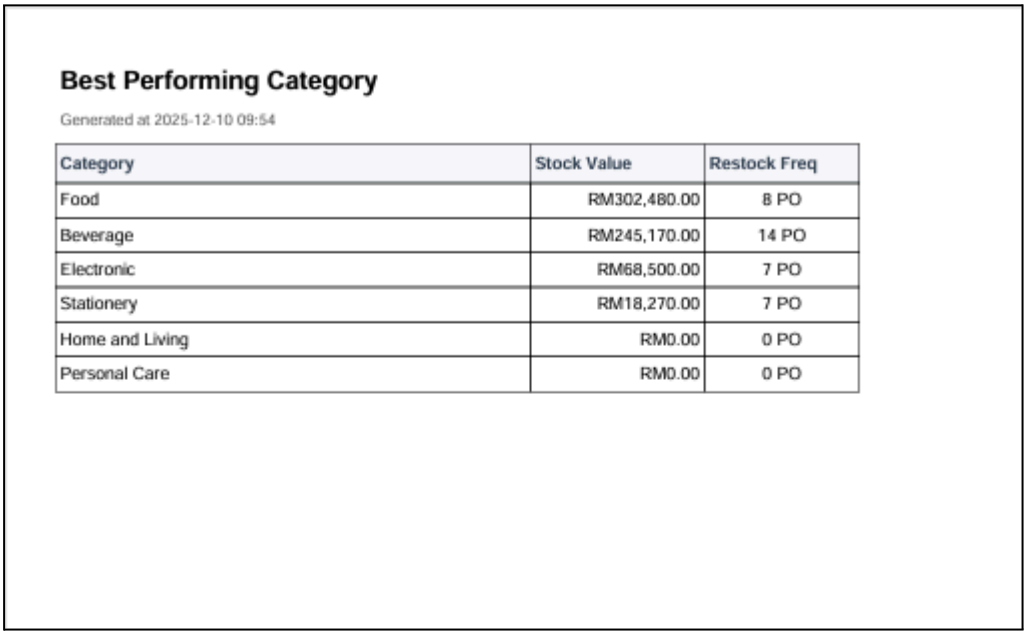
This is a computer generated document. Everything stated in this document is certified true and correct, and requires no signature.

Page 1/1

Appendix A8: Purchase Order PDF (PO-25)

9. Generating Reports for Cloud Reporting

- **Description:** This feature generates yearly inventory reports, where users select a year, filter the results, export them to PDF, and the system stores the file in S3.
- **Instructions:**
    1. Go to Reports → Cloud Reporting.
    2. Select Year → Filter.
    3. Click Export to PDF.
    4. The system generates PDF using FPDF, stores it in S3 and allows the user to Archive.



Best Performing Category

Generated at 2025-12-10 09:54

Category	Stock Value	Restock Freq
Food	RM302,480.00	8 PO
Beverage	RM245,170.00	14 PO
Electronic	RM68,500.00	7 PO
Stationery	RM18,270.00	7 PO
Home and Living	RM0.00	0 PO
Personal Care	RM0.00	0 PO

Appendix A9: Best Performing Category PDF

Appendix B: Developer Guide

This section outlines the technical setup for future developers or examiners maintaining the system.

B.1 Software & Libraries

- **Backend:** PHP (Native), AWS SDK for PHP (installed via Composer).
- **Database:** MySQL 8.0 (Hosted on Amazon RDS).
- **Serverless Functions:** Python 3.13 (deployed on AWS Lambda).
- **Libraries Used:**
    - **FPDF:** For dynamic PDF generation of purchase orders and reports.
    - **PyMySQL:** Used within the Lambda layer to establish database connections.
    - **Chart.js:** For Dashboard Analytics Visualization.

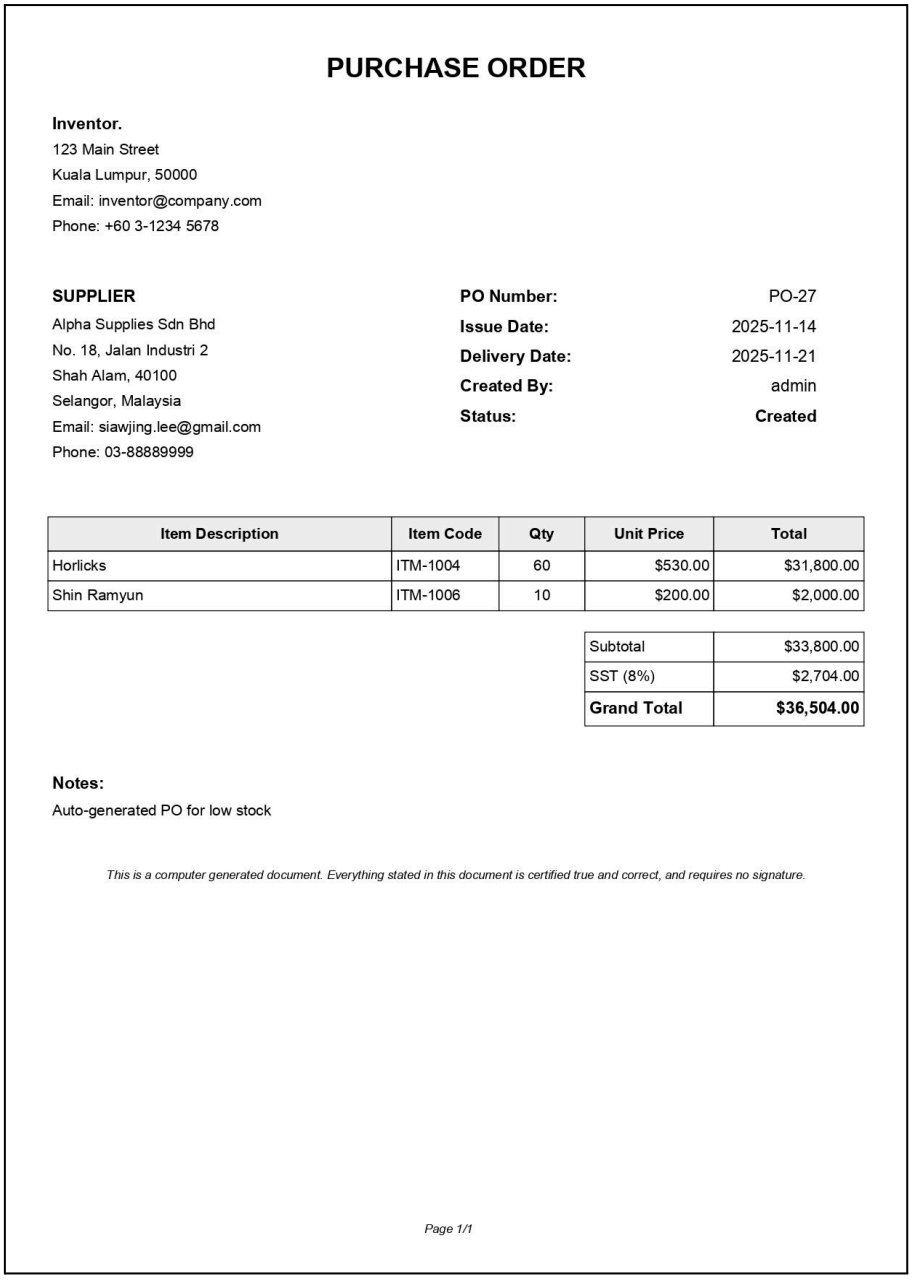
B.2 Authentication Details (Security)

- **Database Configuration:** The database connection settings are located in config/db.php. Note: Real passwords are masked in this documentation for security.
- **AWS Configuration:** The AWS credentials for SDK connectivity are located in config/aws.php. This file requires the following keys:
    - access_key
    - secret_key
    - session_token
    - region (e.g., us-east-1)

Appendix C: Supporting Documents

C.1 Sample Documents

- **Sample Purchase Order PDF:**



PURCHASE ORDER

Inventor.
123 Main Street
Kuala Lumpur, 50000
Email: inventor@company.com
Phone: +60 3-1234 5678

SUPPLIER
Alpha Supplies Sdn Bhd
No. 18, Jalan Industri 2
Shah Alam, 40100
Selangor, Malaysia
Email: siawjing.lee@gmail.com
Phone: 03-88889999

PO Number: PO-27
Issue Date: 2025-11-14
Delivery Date: 2025-11-21
Created By: admin
Status: **Created**

Item Description	Item Code	Qty	Unit Price	Total
Horlicks	ITM-1004	60	$530.00	$31,800.00
Shin Ramyun	ITM-1006	10	$200.00	$2,000.00

Subtotal	$33,800.00
SST (8%)	$2,704.00
Grand Total	**$36,504.00**

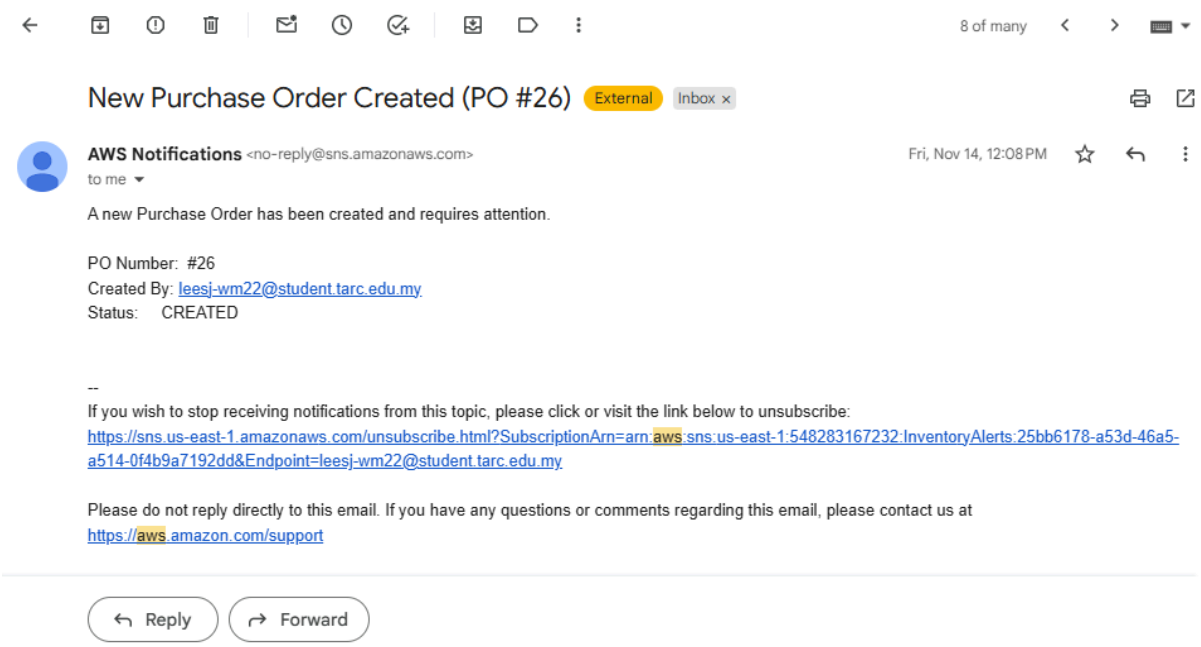
Notes:
Auto-generated PO for low stock

This is a computer generated document. Everything stated in this document is certified true and correct, and requires no signature.

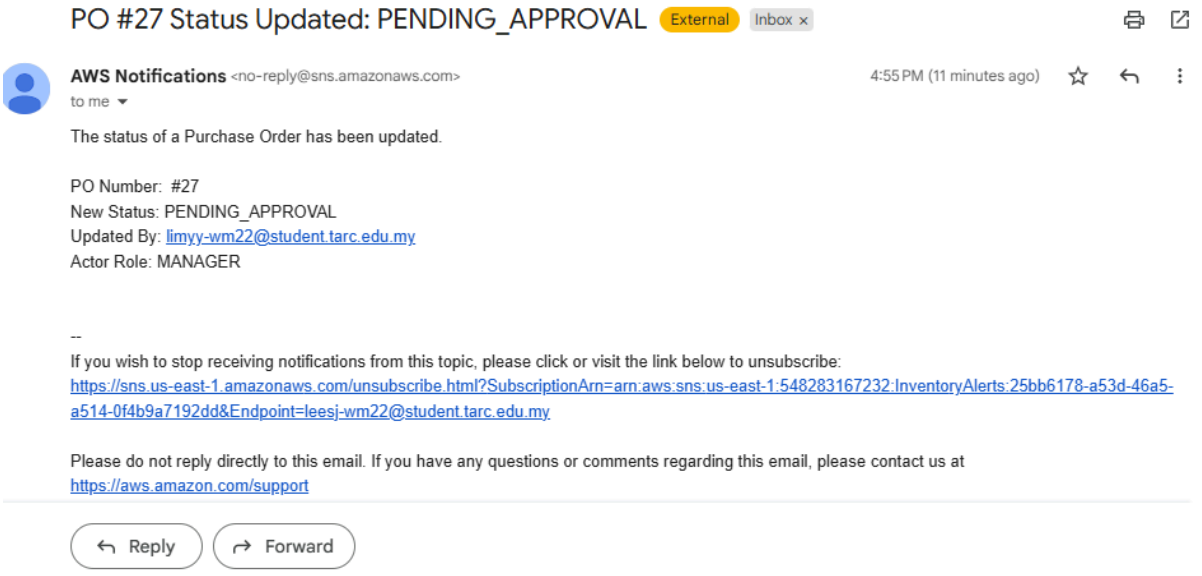
Page 1/1

Appendix C1: Sample Purchase Order PDF (PO-27)

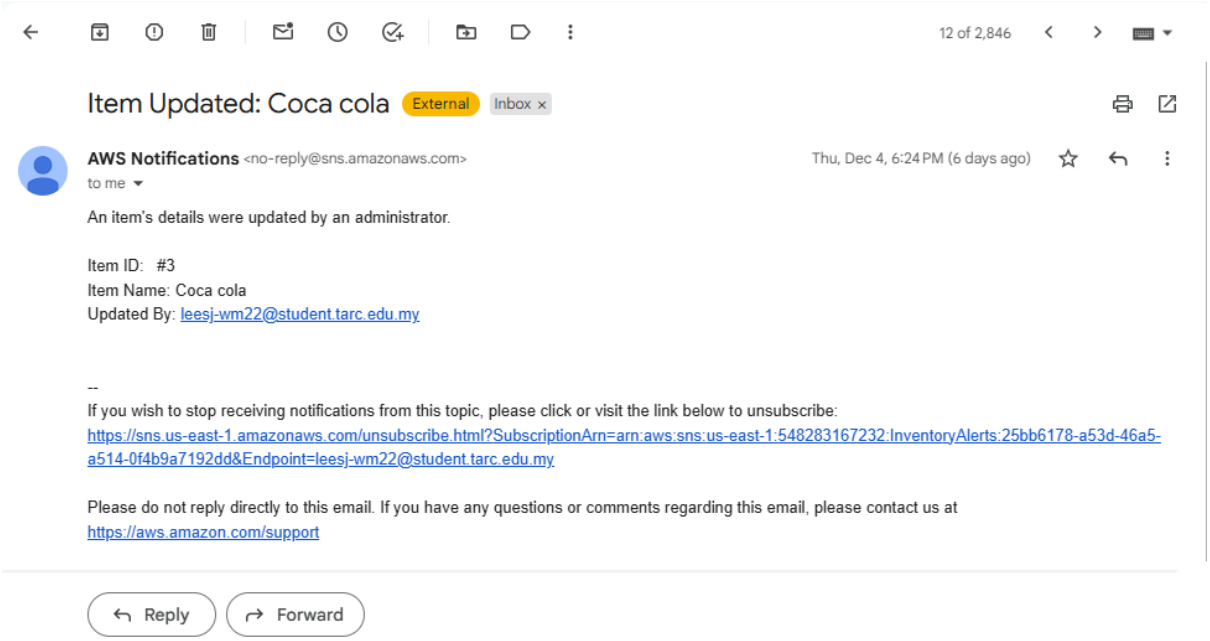
- **Sample Email Notification:**



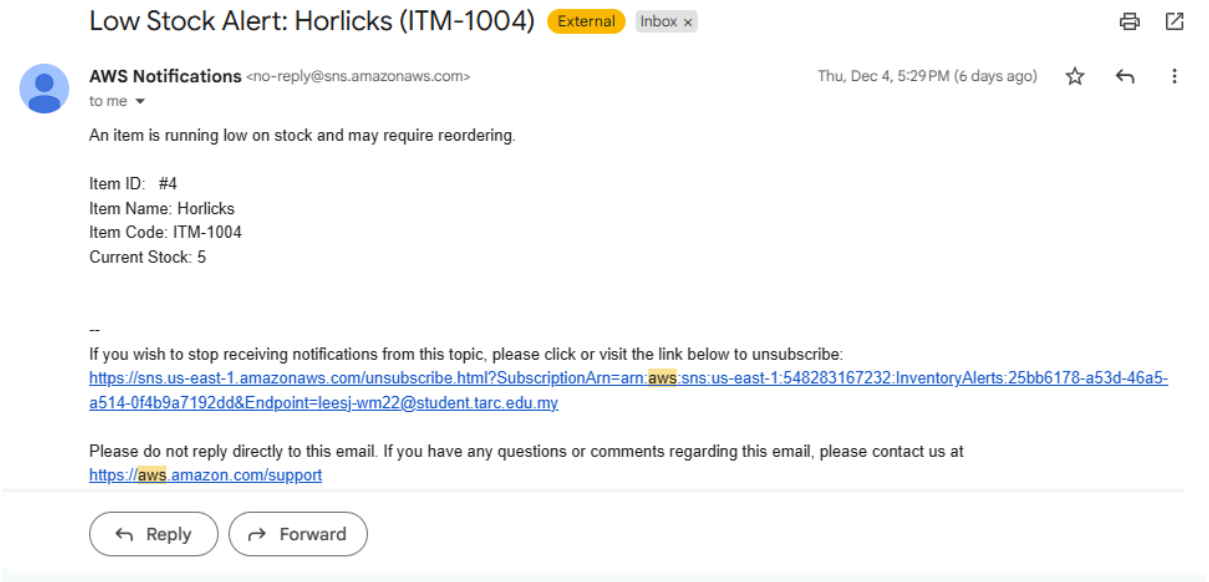
Appendix C2: Sample SNS Email Notification (New PO)



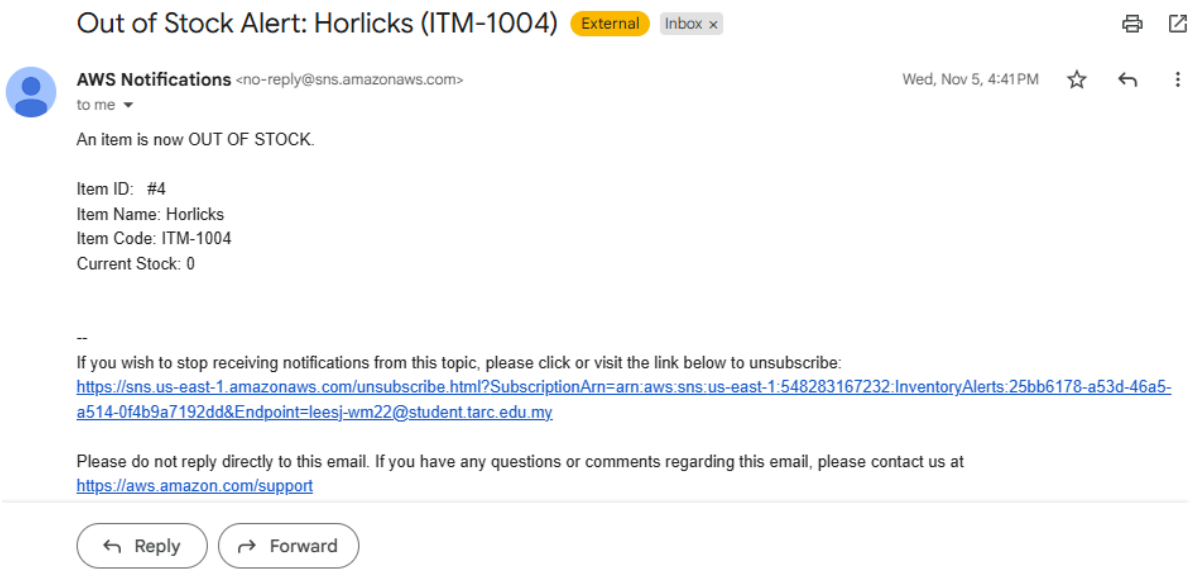
Appendix C3: Sample SNS Email Notification (Pending Approval)



Appendix C4: Sample SNS Email Notification (Item Updated)

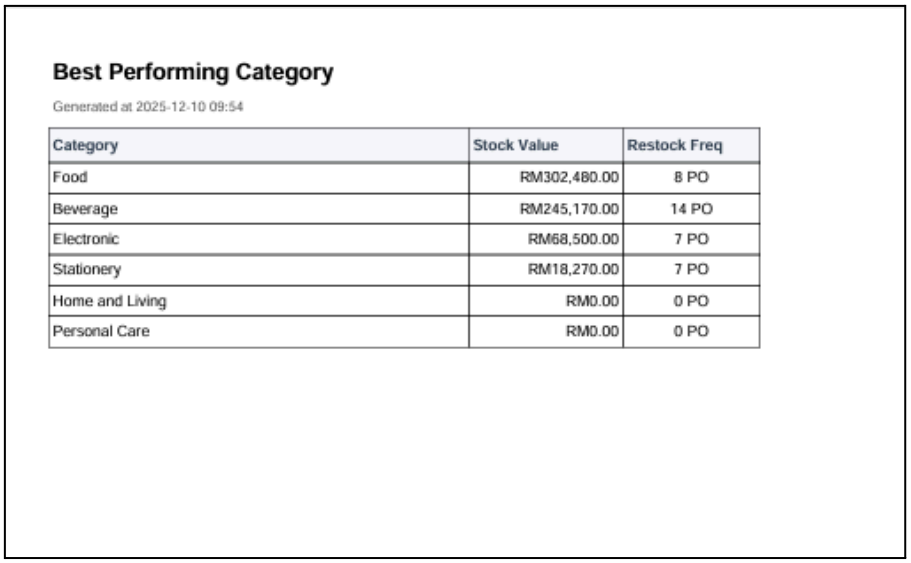


Appendix C5: Sample SNS Email Notification (Low Stock Alert)



Appendix C6: Sample SNS Email Notification (Out of Stock)

- **Sample Cloud Reporting PDF:**



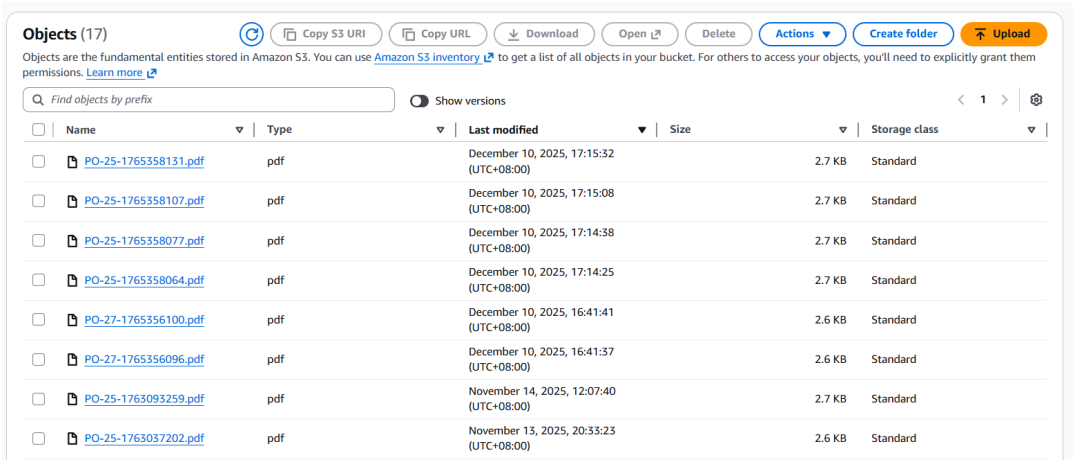
Best Performing Category

Generated at 2025-12-10 09:54

Category	Stock Value	Restock Freq
Food	RM302,480.00	8 PO
Beverage	RM245,170.00	14 PO
Electronic	RM68,500.00	7 PO
Stationery	RM18,270.00	7 PO
Home and Living	RM0.00	0 PO
Personal Care	RM0.00	0 PO

Appendix C7: Sample Cloud Reporting PDF

- **Sample file successfully stored in S3:**



Appendix C8: Sample Files in S3

C.2 Interview/Survey Results

- **Interview Summary:**
  - **Interviewee:** Mr. Nick Lee, Founder of Green Life Design Sdn Bhd.
  - **Date:** 26 July 2025
  - **Key Findings:** The interview highlighted the need for automating the manual stock checking process and centralizing supplier communication to prevent missed orders.

C.3 Code Snippets

- **Auto-PO Lambda Function (Python):**

```
import json
import pymysql
import os
import logging
from datetime import date, timedelta

# Initialize logger for CloudWatch
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    logger.info("Lambda triggered: AutoPO_Generator started")

    try:
        # Connect to RDS
```

```
        conn = pymysql.connect(
            host=os.environ['DB_HOST'],
            user=os.environ['DB_USER'],
            password=os.environ['DB_PASS'],
            db=os.environ['DB_NAME'],
            cursorclass=pymysql.cursors.DictCursor
        )
        logger.info("Database connection successful")

        with conn.cursor() as cursor:
            # STEP 1: Get all items below threshold
            logger.info("Fetching items below threshold...")
            cursor.execute("""
                SELECT i.item_id, i.item_name, i.supplier_id,
i.stock_quantity, i.threshold_quantity, i.unit_cost,
                    COALESCE((
                        SELECT SUM(pod.quantity)
                        FROM purchase_order_details pod
                        JOIN purchase_order po ON pod.po_id =
po.po_id
                        WHERE pod.item_id = i.item_id
                        AND po.status IN ('Created', 'Pending',
'Approved', 'Confirmed', 'Shipped', 'Delayed', 'Received')
                    ), 0) AS ordered_qty
                FROM item i
                WHERE i.stock_quantity < i.threshold_quantity
                ORDER BY i.supplier_id;
            """)
            raw_items = cursor.fetchall()
            logger.info(f"Fetched {len(raw_items)} low-stock
items")

            # STEP 2: Filter items where stock + ordered <
threshold
            items = []
            for item in raw_items:
                projected_stock = item['stock_quantity'] +
item['ordered_qty']
                if projected_stock < item['threshold_quantity']:
                    items.append(item)
            logger.info(f"{len(items)} items require
auto-reordering after filtering")
```

```
            if not items:
                logger.info("No items below threshold or already
covered by existing POs. Lambda execution completed.")
                return {
                    'statusCode': 200,
                    'body': json.dumps({'message': 'No items
below threshold or already covered by existing POs.'})
                }



            # STEP 3: Group items by supplier
            suppliers = {}
            for item in items:
                sid = item['supplier_id']
                if sid not in suppliers:
                    suppliers[sid] = []
                suppliers[sid].append(item)
            logger.info(f"Grouped items into {len(suppliers)}
supplier(s) for PO generation")

            po_results = []
            for supplier_id, item_list in suppliers.items():
                # STEP 4: Insert into purchase_order
                cursor.execute("""
                    INSERT INTO purchase_order
(created_by_user_id, supplier_id, issue_date, expected_date,
status, description)
                    VALUES (%s, %s, %s, %s, %s, %s)
                """, (
                    1,  # created_by_user_id (admin/system)
                    supplier_id,
                    date.today(),
                    date.today() + timedelta(days=7),
                    'Created',
                    'Auto-generated PO for low stock'
                ))
                po_id = conn.insert_id()
                logger.info(f"Created PO #{po_id} for Supplier ID
{supplier_id}")

                # STEP 5: Add items to purchase_order_details
                for item in item_list:
```

```
                    restock_qty = item['threshold_quantity'] * 2
                    unit_price = item['unit_cost']
                    total_cost = restock_qty * unit_price

                    cursor.execute("""
                        INSERT INTO purchase_order_details
(po_id, item_id, quantity, unit_price, purchase_cost)
                        VALUES (%s, %s, %s, %s, %s)
                    """, (
                        po_id,
                        item['item_id'],
                        restock_qty,
                        unit_price,
                        total_cost
                    ))
                logger.info(f"Added {len(item_list)} PO details
for PO #{po_id}")

                # STEP 6: Internal notifications
                cursor.execute("""
                    SELECT user_id FROM user
                    WHERE role IN ('Admin', 'Manager') AND status
= 'Active';
                """)
                for user in cursor.fetchall():
                    cursor.execute("""
                        INSERT INTO notification (user_id, title,
message, link)
                        VALUES (%s, %s, %s, %s)
                    """, (
                        user['user_id'],
                        'Auto-Generated PO',
                        f'New PO #{po_id} has been auto-generated
for Supplier ID {supplier_id}',
                        f'/index.php?page=po_details&id={po_id}'
                    ))
                logger.info(f"Notification sent for PO #{po_id}
(Supplier {supplier_id})")

                po_results.append({'supplier_id': supplier_id,
'po_id': po_id})

            conn.commit()
```

```
            logger.info("All auto-generated POs committed
successfully to database")

        logger.info("Lambda completed successfully")
        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': 'Auto-generated POs successfully.',
                'created': po_results
            })
        }

    except Exception as e:
        logger.error(f"Error occurred in AutoPO_Generator:
{str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

- **SNS Email Trigger (PHP):**

```
<?php
// app/Notify.php
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

function sns_notify(string $subject, string $message, array
$attrs = []): bool {
  // Load AWS configuration
  $cfg = require __DIR__ . '/../config/aws.php';

  $client_config = [
    'version'     => '2010-03-31',
    'region'      => $cfg['region'],
    'credentials' => [
        'key'    => $cfg['credentials']['key'],
        'secret' => $cfg['credentials']['secret'],
    ]
  ];

  // Add session token if required (for AWS Learner Lab)
  if (!empty($cfg['credentials']['token'])) {
```

```
        $client_config['credentials']['token'] =
$cfg['credentials']['token'];
    }

    $sns = new SnsClient($client_config);

    // Prepare message parameters
    $p = [
        'TopicArn' => $cfg['topic_arn'],
        'Subject'  => $subject,
        'Message'  => $message
    ];

    // Add message attributes for filtering
    if ($attrs) {
        foreach ($attrs as $k => $v) {
          $p['MessageAttributes'][$k] = [
              'DataType' => 'String',
              'StringValue' => (string)$v
          ];
        }
    }

    try {
      $sns->publish($p);
      return true;
    } catch (AwsException $e) {
      error_log('[SNS Error] ' . $e->getAwsErrorMessage());
      return false;
    }
}

// app/PoNotify.php
function po_notify_status_change(string $poId, string $rawStatus,
string $actorEmail, string $actorRole): bool {
    // Normalize status string
    $status = strtoupper(trim($rawStatus));
    $actorRole = strtoupper($actorRole);

    $subject = "PO #$poId Status Updated: $status";

    $message = "The status of a Purchase Order has been
updated.\n\n"
```

```
            . "PO Number:  #$poId\n"
            . "New Status: $status\n"
            . "Updated By: $actorEmail\n"
            . "Actor Role: $actorRole\n";

  // Send notification with attributes for filtering
  return sns_notify($subject, $message, [
    'event'     => 'PO_STATUS_CHANGED',
    'status'    => $status,
    'poId'      => $poId,
    'actorRole' => $actorRole
  ]);
}
?>
```

Appendix D: Originality Report

Originality report

COURSE NAME
FYP 202505

STUDENT NAME
SIAW JING LEE

FILE NAME
FYP Report _Lee Siaw Jing

REPORT CREATED
Dec 17, 2025

Summary

Flagged passages	2	0.3%
Cited/quoted passages	10	1%

Web matches

coursehero.com	2	0.5%
uonbi.ac.ke	1	0.1%
gulf.edu.sa	1	0.1%
inventorysystemsolutions.com	1	0.1%
muxlet.com	1	0.1%
testdevlab.com	1	0.1%
spendflo.com	1	0.1%
eurostop.com.sg	1	0.1%
getgenie.io	1	0.1%
docuwriter.ai	1	0.1%
joinstored.com	1	0%

1 of 12 passages

Student passage FLAGGED

All rights reserved. No part of this project documentation may be reproduced, stored in a **retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman**...

Top web match

All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman...

Comprehensive Guide to Project Documentation and Abstracts
... https://www.coursehero.com/file/101625685/FYP-Report-Templatedocx/

Appendix D1: Originality Report