

TROY UNIVERSITY
CS 3332 – Spring 2025
Software Engineering

Group Project

Cafeteria Management System

Student:
[STUDENT_1]
[STUDENT_2]
[STUDENT_3]
[STUDENT_4]
[STUDENT_5]
Lecturer:
Assoc Prof. Cao Tuan Dung

May 27, 2025

Troy University Resources

Table of Contents

1 Abstract **1**

2 Market Demands and Problem Analysis **2**

2.1 Project's Description . . . 2

2.2 Market Demands On the System . . . 2

2.3 Problem Analysis . . . 3

2.4 Developing Diagrams . . . 3

2.5 Project Planing and Milestones . . . 4

3 System Design Specification **5**

3.1 General Introduction . . . 5

3.2 System Specific Requirements . . . 6

3.2.1 Back-End Server Requirements . . . 6

3.2.2 Manager Interface Requirements . . . 6

3.2.3 Storage Manager Requirements . . . 7

3.2.4 Cashier (POS) Requirements . . . 7

3.2.5 Bartender Interface Requirements . . . 8

3.2.6 General System Requirements . . . 8

3.3 Use case Diagram . . . 9

3.3.1 System Administration . . . 9

3.3.2 Inventory Management . . . 10

3.3.3 Order Processing . . . 12

3.3.4 Beverage Preparation . . . 14

3.3.5 Use Case Dependencies . . . 14

3.4 Use case Specification . . . 15

3.4.1 System Administration - Manager . . . 15

3.4.2 Order Processing Use Case . . . 18

3.4.3 Inventory Management . . . 22

3.4.4 Beverage Preparation Use Case . . . 25

3.5 Activity Diagram . . . 25

3.5.1 System Administration – Manager . . . 25

3.5.2 Order Processing Use Case . . . 28

3.5.3 Inventory Management . . . 34

3.5.4 Beverage Preparation Use Case . . . 37

4 System Architucture **39**

4.1 Overall System Architecture . . . 39

4.2 System Design . . . 39
4.3 Sequential Diagrams . . . 39
4.4 Class Diagrams . . . 39
4.4.1 System Administration & Manager . . . 40
4.4.2 Order Processing/Sale . . . 41
4.4.3 Inventory Management . . . 43
4.4.4 Beverage Preparation . . . 45
4.5 Class Diagram . . . 46
4.5.1 Payload . . . 47
4.5.2 Response . . . 48
4.5.3 Handler . . . 49
4.5.4 Data Manager . . . 50
4.6 Testing . . . 51
4.6.1 Testing Strategy . . . 52
4.6.2 Test Case Partitioning . . . 52
4.6.3 Test Cases . . . 53
4.6.4 Test Implementation . . . 55
4.6.5 Test Results and Metrics . . . 56
4.7 Development and Architectural Models . . . 57
4.8 Architecture Specification . . . 58
4.8.1 Core Components . . . 58
4.8.2 System Flow . . . 58
4.8.3 Request-Response Flow . . . 58
4.8.4 Key Components . . . 59
4.8.5 Data Models . . . 60
4.8.6 Error Handling . . . 60
4.8.7 Security . . . 60
4.8.8 Database . . . 61
4.8.9 Implementation Details . . . 61

5 User Interface **63**
5.1 Manager UI . . . 63
5.1.1 Login . . . 63
5.1.2 Profile . . . 67
5.1.3 Staff Account . . . 70
5.1.4 Sale Data . . . 75
5.1.5 New Item . . . 78
5.1.6 Tab Translation . . . 82
5.2 Cashier UI . . . 83

Troy University Resources

5.2.1 Order . . . 84
5.2.2 Map . . . 87
5.2.3 History . . . 92
5.2.4 Customer Order . . . 92
5.3 Inventory UI . . . 93
5.3.1 Dashboard Menu . . . 94
5.3.2 Inventory Menu . . . 94
5.3.3 Import Menu . . . 95
5.3.4 History Menu . . . 99
5.3.5 Daily Checks Menu . . . 100
5.4 Bar UI . . . 101

6 Tools and Workflow **102**
6.1 System Architecture Overview . . . 102
6.2 Collaborative Development and API Testing . . . 102
6.3 CI/CD Workflow with GitHub Actions and Azure . . . 103
6.4 Backend Automation and Release Workflows . . . 104

7 Conclusion **107**

8 References **109**

9 Contribution **110**

1 Abstract

The cafeteria industry plays a crucial role in daily life, serving food and beverages to a diverse range of customers, including students, employees, and the general public. With the rapid advancement of technology, implementing a Cafeteria Management System (CMS) is essential to optimizing operations, improving service quality, and ensuring a seamless dining experience. To address the challenges in cafeteria management, our team has chosen to develop the project: "Building a Cafeteria Management System.

This report outlines the entire process of designing, developing, and implementing the system, focusing on enhancing efficiency and streamlining cafeteria operations. The Cafeteria Management System is designed to digitize and automate key processes, including order management, inventory tracking, and billing automation. Additionally, the system enables real-time monitoring of transactions, menu updates, and inventory levels, while ensuring secure payment processing and user-friendly access for both customers and staff. By integrating modern technology and data-driven decision-making, the CMS aims to enhance operational efficiency, reduce manual errors, and improve customer satisfaction, making it a robust solution for modern cafeteria management.

Keywords: Cafeteria Management System, Management System, Order Management System, Inventory Tracking, Point of Sale (POS)

Troy University Resources

2 Market Demands and Problem Analysis

2.1 Project's Description

The Cafeteria Management System (CMS) is designed to enhance the efficiency and accuracy of cafeteria operations by automating food ordering, inventory management, billing, and reporting processes. The primary objective of this project is to develop a digital platform that streamlines customer interactions, optimizes kitchen workflows, and improves overall service delivery. The system will feature an intuitive interface for customers to place orders, an admin panel for staff to manage inventory and track sales, and a secure payment gateway for cashless transactions. In addition, real-time analytics and reporting tools will be integrated to help cafeteria managers make data-driven decisions. By reducing manual errors, improving service speed, and ensuring smooth coordination between different cafeteria operations, the CMS will serve as a robust and scalable solution tailored to modern cafeteria needs.

2.2 Market Demands On the System

In the age of IoT (Internet of Things) and rapid technological advancements, Cafeteria Management Systems (CMS) have become indispensable in modern food service operations. The integration of smart technology, automation, and data-driven solutions has revolutionized the way cafeterias operate, improving both efficiency and user experience.

The application of IoT enabled devices allows real-time inventory tracking, automated order processing, and seamless communication between customers, staff, and kitchen operations. Features such as self-service kiosks, mobile app ordering, and contactless payments provide customers with a faster, more convenient, and personalized dining experience. Meanwhile, cloud-based management systems enable cafeteria administrators to monitor sales, track food consumption, and optimize workflows from anywhere.

In addition, AI-powered analytics and smart recommendation systems help cafeterias better understand customer behavior, forecast demand, and improve decision-making. The integration of advanced security protocols and encrypted transactions ensures data privacy, secure payments, and fraud prevention.

Using the latest technological advances, the CMS not only improves operational efficiency but also creates a more seamless and satisfying experience for both customers and staff, making it an essential tool for the future of cafeteria management.

2.3 Problem Analysis

Managing the cafeteria involves a number of crucial elements used to maximize operating procedures, enhance service standards, and guarantee efficient operations:

- **Order and transaction management:** helping clients place orders, monitoring their progress, handling payments, and keeping track of past transactions.
- **Inventory and materials management:** Monitoring stock levels, updating material quantities, forecasting consumption demand, and minimizing food waste.
- **Staff Management:** Support check-in and check-out for employees, records working hours, and tracks staff performance.
- **Client Services:** Adding promoting services like vouchers or references to ensure that the client has the best experiences and be able to contribute to the development of the store.
- **Data reporting and analysis:** Supports a variety of payment methods such as cash, bank cards, e-wallets, ensuring fast and secure transactions.

A Cafeteria Management System not only automates the process, but also improves operational efficiency, minimizes manual errors, and improve the customer experience.

2.4 Developing Diagrams

To visualize the Cafeteria Management System (CMS) operations and its functional structure, we have utilized various diagrams and functional breakdowns, including:

- **Functional Hierarchy Diagram:** Lists the core functions of the system and how they interact with one another. This helps in understanding the overall architecture of the CMS.
- **Use Case Diagrams and Specifications:** Illustrates how different users (such as customers, staff, and administrators) interact with the system in various real-world scenarios. This provides a clear overview of user roles and system responses.
- **Activity Diagrams:** Depicts the step-by-step workflow of the system, outlining processes like order placement, inventory management, staff check-in/check-out, and transaction processing.

Troy University Resources

These diagrams help provide a structured visualization of the CMS, ensuring a clear understanding of system functionalities and user interactions.

2.5 Project Planing and Milestones

Our team has created a general project plan to guide the development of the Cafeteria Management System (CMS), which includes the following key components:

- **Project Timeline:** Defining the major phases of the project and estimating the expected duration for each phase, including research, design, development, testing, and deployment.
- **Required Resources:** Identifying essential resources such as team members, their skill sets, and the necessary tools or technologies to successfully complete the project.
- **Risk Assessment:** Recognizing potential risks that may arise during the project, such as technical challenges, time constraints, or resource limitations, and establishing strategies to mitigate them.
- **Task Execution and Collaboration:** Ensuring that all team members work together efficiently, following the project plan to complete the assigned tasks within the set deadlines.

By following this structured plan, the team aims to develop a functional, efficient, and user-friendly Cafeteria Management System while minimizing risks and ensuring smooth project execution.

3 System Design Specification

3.1 General Introduction

This section outlines the overall requirements for a Cafeteria Manager System, detailing its functionalities, workflow, and non-functional requirements to ensure efficient operation. The system consists of multiple components, each catering to a specific role in the cafeteria's workflow.

- **Back-End Server:** responsible for handling all system logic, processing requests, managing the database, and performing data analysis. It ensures seamless communication between different parts of the system, processes customer orders, manages inventory, and handles payments. Additionally, it generates reports and analytics to assist in managerial decision-making.
- **Manager:** serves as the primary interface for overseeing the cafeteria's operations. It provides analytical reports on sales trends, popular menu items, and inventory levels. Through this interface, the manager can make informed decisions regarding stock replenishment, pricing adjustments, and employee role management. It also allows configuration of system settings and permissions for different staff members.
- **Storage Manager:** responsible for inventory tracking and stock management. It provides a clear overview of available ingredients, monitors stock levels, and alerts staff when supplies are running low. The system helps optimize restocking schedules and reduces waste by ensuring ingredients are used efficiently. It also integrates with the Back-End Server to provide real-time data on inventory consumption and replenishment needs.
- **Cashier:** functions as the cafeteria's Point of Sale (POS) system. It enables cashiers to place orders, calculate bills, and process payments efficiently. The system supports multiple payment methods, including cash, card, and digital wallets, ensuring a smooth checkout experience for customers.
- **Bartender:** designed to streamline beverage preparation. It displays a real-time list of drinks that need to be prepared and allows bartenders to confirm when a drink is ready. Once a drink is prepared, the system notifies waiters, ensuring timely and accurate delivery to customers.

Troy University Resources

3.2 System Specific Requirements

3.2.1 Back-End Server Requirements

Functional Requirements:

- Must process and validate all incoming requests from client applications
- Must maintain a centralized database for all system data
- Must implement secure authentication and authorization mechanisms
- Must handle real-time data synchronization between all system components
- Must process and validate payment transactions
- Must maintain audit logs of all system activities
- Must implement data backup and recovery mechanisms

Non-Functional Requirements:

- Must maintain 99.9% uptime
- Must process requests within 2 seconds
- Must support concurrent access from multiple users
- Must implement data encryption for sensitive information
- Must be scalable to handle increasing load
- Must comply with data protection regulations

3.2.2 Manager Interface Requirements

Functional Requirements:

- Must display real-time sales analytics and trends
- Must allow menu item management and pricing updates
- Must enable staff role and permission management
- Must generate customizable reports
- Must provide system configuration options
- Must display performance metrics and KPIs

Non-Functional Requirements:

- Must have an intuitive and user-friendly interface
- Must provide role-based access control
- Must support multiple data visualization formats
- Must ensure data accuracy in reports
- Must provide export functionality for reports

3.2.3 Storage Manager Requirements

Functional Requirements:

- Must track real-time inventory levels
- Must generate low-stock alerts
- Must track ingredient usage and consumption
- Must maintain supplier information
- Must track inventory movement history

Non-Functional Requirements:

- Must update inventory levels in real-time
- Must provide accurate stock level information
- Must maintain data integrity
- Must provide backup for inventory data

3.2.4 Cashier (POS) Requirements

Functional Requirements:

- Must process customer orders
- Must handle multiple payment methods
- Must generate receipts
- Must manage customer transactions
- Must process refunds and returns

Non-Functional Requirements:

- Must process transactions within 3 seconds
- Must be available 24/7 during operating hours

Troy University Resources

- Must ensure secure payment processing

3.2.5 Bartender Interface Requirements

Functional Requirements:

- Must display real-time order queue
- Must allow order status updates
- Must track preparation time
- Must notify when orders are ready
- Must track ingredient usage per order

Non-Functional Requirements:

- Must update order status in real-time
- Must have clear visual and audio notifications
- Must be easy to use during peak hours

3.2.6 General System Requirements

Functional Requirements:

- Must support multiple user roles and permissions
- Must provide data backup and recovery
- Must maintain system logs
- Must support system updates and maintenance
- Must provide help and documentation

Non-Functional Requirements:

- Must be accessible via web browsers or mobile applications
- Must maintain data security and privacy
- Must be scalable for future growth
- Must provide 24/7 technical support
- Must comply with industry standards and regulations
- Must have a response time of less than 2 seconds
- Must maintain data consistency across all components

3.3 Use case Diagram

This section provides a comprehensive overview of the system's use cases through various diagrams and specifications. It includes detailed descriptions of primary actors, their roles, and the interactions between different system components. The documentation encompasses:

- Use case diagrams illustrating the relationships between actors and system functionalities
- Activity diagrams showing the flow of operations and decision points
- Detailed specifications for each use case including preconditions, postconditions, and main scenarios
- General factors and constraints affecting the system's behavior
- Dependencies between different use cases and their impact on system operations

3.3.1 System Administration

Primary Actor:

- System Administrator Responsible for system maintenance, user management, and security configurations
- Manager Oversees system operations, approves system changes, and manages user access levels

Use case diagram:

Troy University Resources

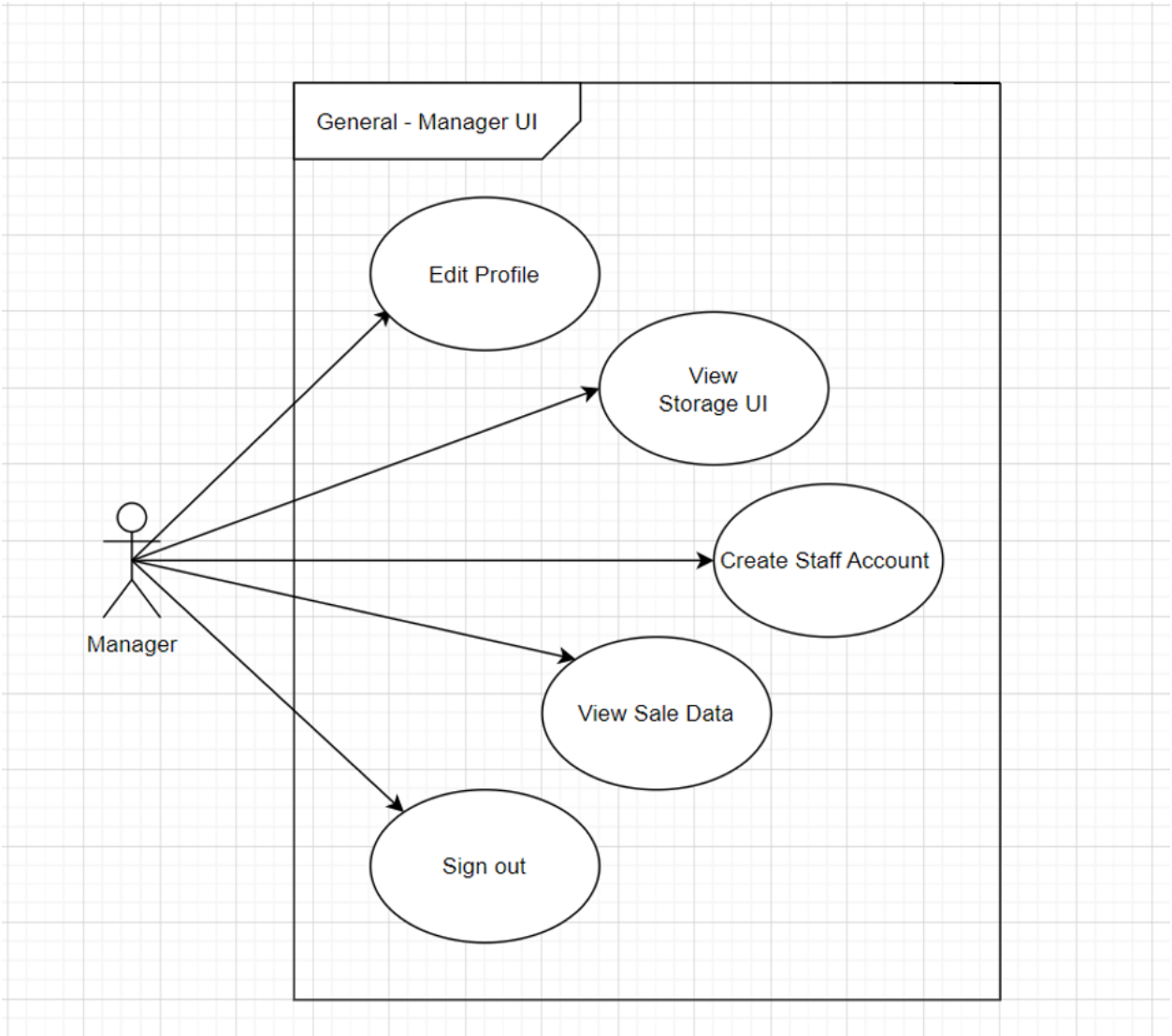


Figure 1: System Administration and Management General Use Case Diagram

3.3.2 Inventory Management

Primary Actors:

- Storage Manager Manages inventory levels, tracks stock movements, and handles supplier relationships

Use case diagram:

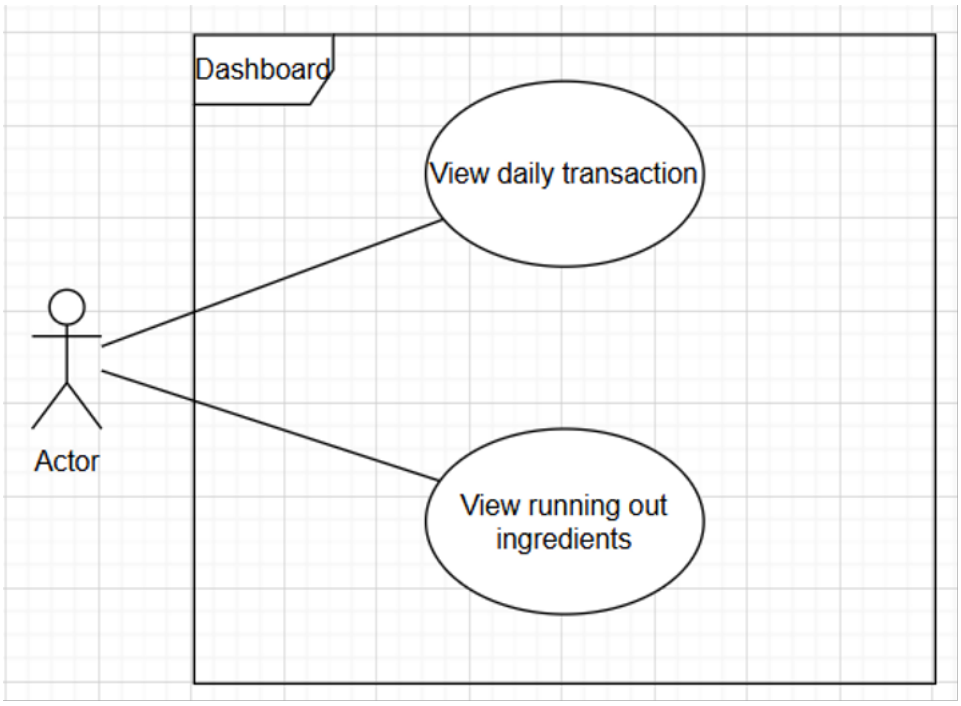


Figure 2: Inventory Management Overview Use Case Diagram

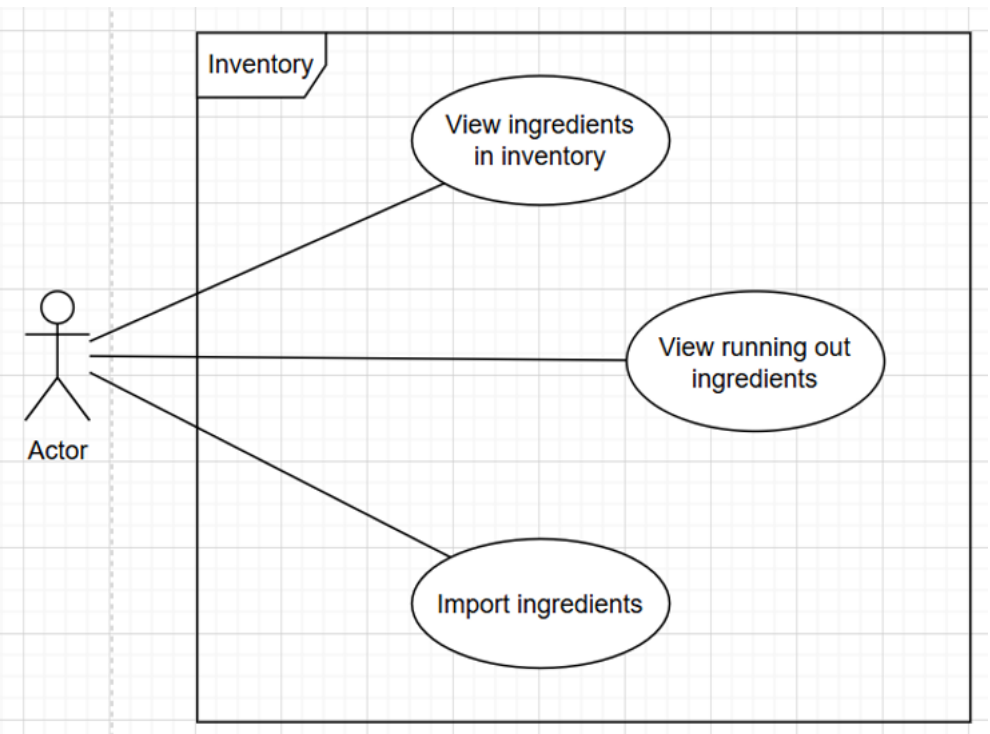


Figure 3: Inventory Supplies Import/Export Use Case Diagram

Troy University Resources

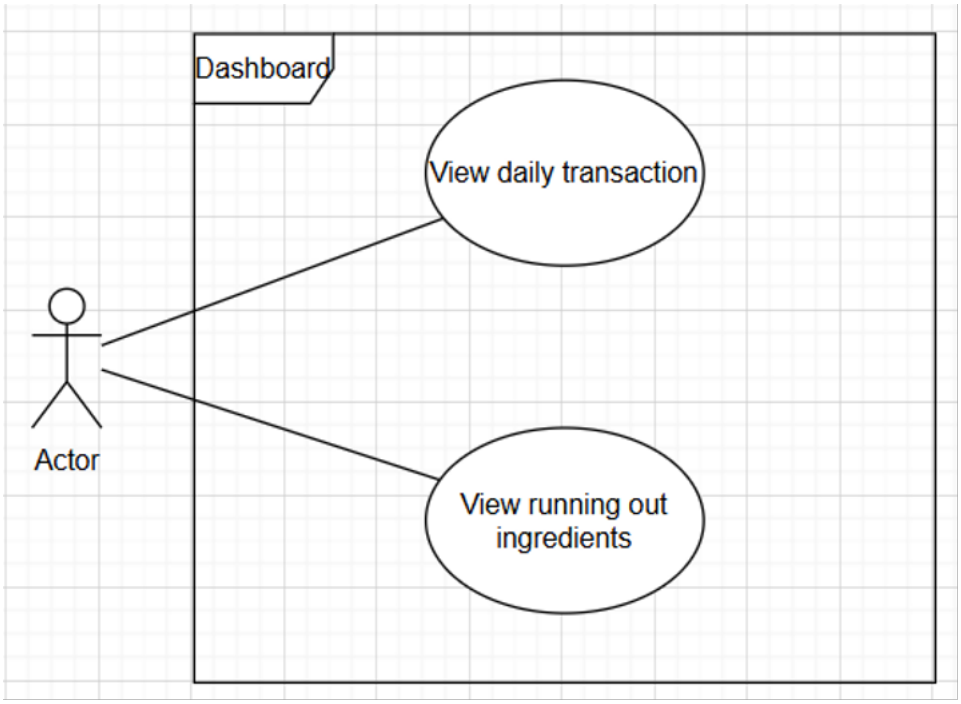


Figure 4: Inventory Supplies Management Use Case Diagram

3.3.3 Order Processing

Primary Actors:

- Cashier Processes customer orders, handles payments, and manages order queue
- Customer Places orders, makes payments, and receives prepared beverages

Use case diagram:

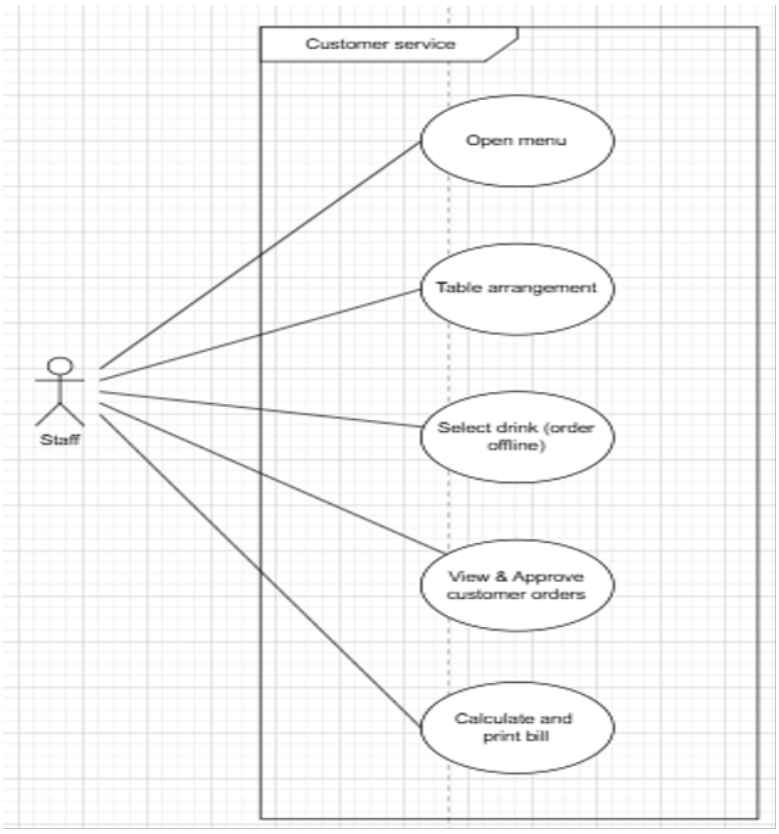


Figure 5: Order Processing General Use Case Diagram

Use case diagram:

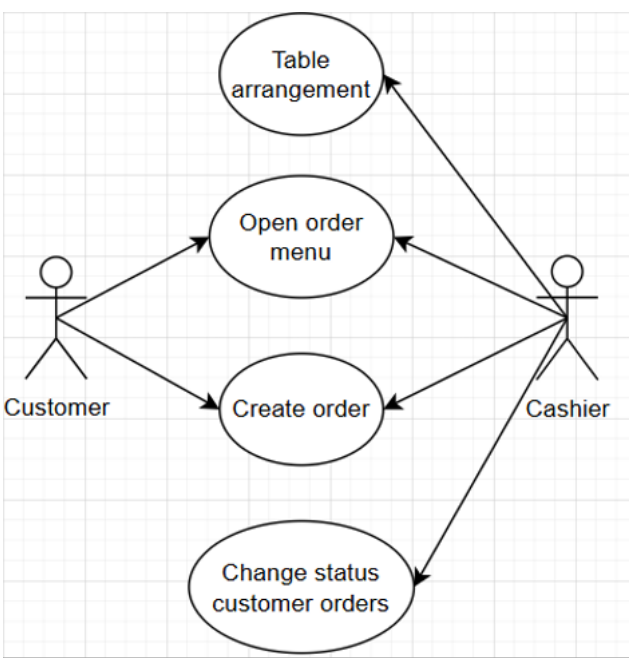


Figure 6: Customer Order Use Case Diagram

Troy University Resources

3.3.4 Beverage Preparation

Primary Actor:

- Bartender Prepares beverages according to recipes, manages preparation queue, and maintains quality standards

Use case diagram:

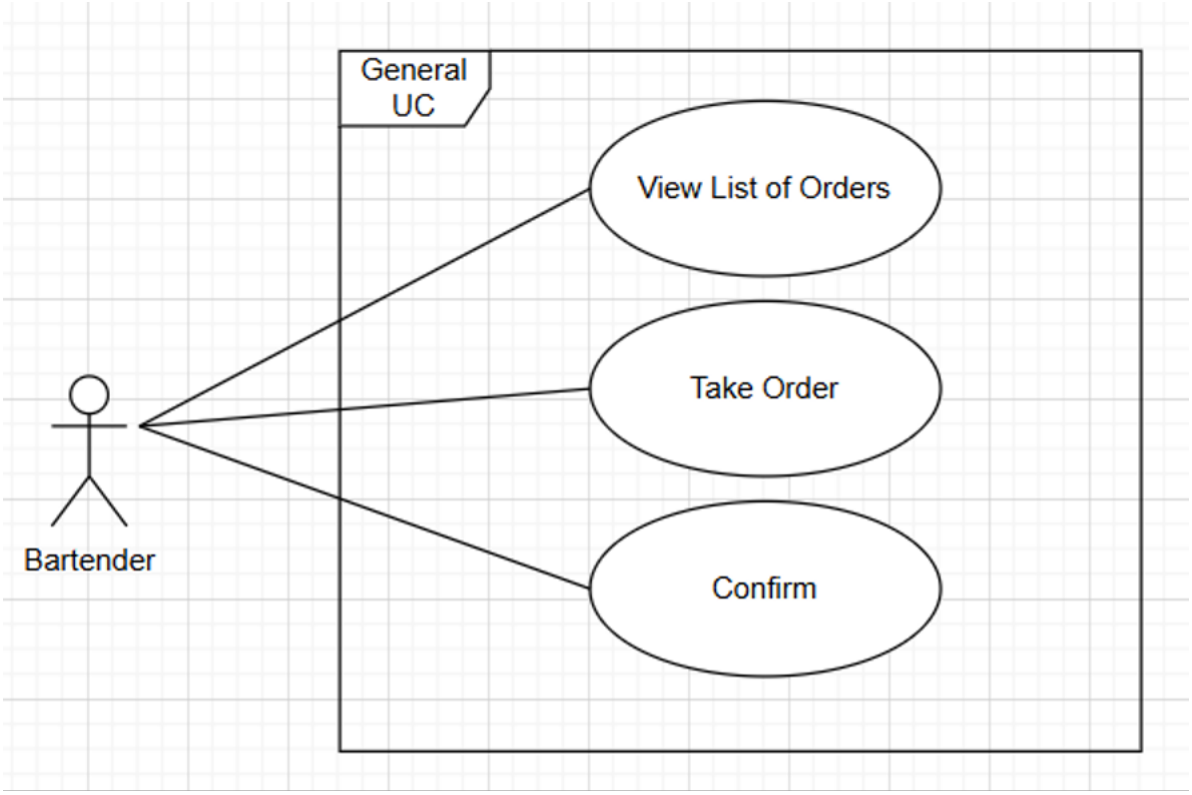


Figure 7: Beverage Preparation Use Case Diagram

3.3.5 Use Case Dependencies

- Order Processing depends on Inventory Management for stock availability
- Beverage Preparation depends on Order Processing for order details
- Inventory Management depends on Supplier Management for stock updates
- Manager Operations can access data from all other use cases
- System Administration provides the foundation for all other use cases

3.4 Use case Specification

This section provides detailed specifications for each use case identified in the system. Each specification includes comprehensive information about the actors involved, their interactions with the system, and the specific scenarios that can occur. The specifications are structured to provide a clear understanding of:

- Use case diagrams illustrating the relationships between actors and system functionalities
- Activity diagrams showing the flow of operations and decision points
- Detailed specifications for each use case including preconditions, postconditions, and main scenarios
- General factors and constraints affecting the system's behavior
- Dependencies between different use cases and their impact on system operations

These specifications serve as a foundation for system development, ensuring that all stakeholders have a clear understanding of the system's functionality and behavior. They also help in identifying potential issues and dependencies early in the development process.

This use case describes the process for managers to create new staff accounts, including input validation and error handling for invalid information.

3.4.1 System Administration - Manager

Use Case Name: Manager UI – Create Staff Account			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display a table with various input types (ex: check box, input field, password input field)
Main	2	Manager	Fill input field, tick check box
Main	3	Manager	Create account
Main	4	System	Get and verify information inside input fields
Main	5	System	Send data to server
Alternative	5a	System	If the information fails conditions then pop up error message

Table 1: Create Staff Account – Manager UI

Troy University Resources

This use case covers how managers view and update their profile information, with system feedback for invalid updates.

Use Case Name: Manager UI – Profile			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display private information of the current account
Main	2	Manager	Fill input fields
Main	3	Manager	Save information
Main	4	System	Retrieve input field data
Main	5	System	Verify input data
Main	6	System	Send new data to the server
Alternative	6a	System	If information is invalid, display error message: "Your new information doesn't meet the condition."

Table 2: Profile – Manager UI

Data requirements				
Request				
Field	Type	Sample	Constraint	Description
token	string	tokentokentoken		Token of the current account after login
name	string	[STUDENT_4], [STUDENT_3]	No number, No specical digit	client real name
username	string	[USER_4], [USER_3]	no space	login's name
password	string	123123, absdbasd, Tr@yPass	Have to be 8 digits at minimum and 32 digits at max, no unicode char	login's password
date of birth	long	1745918075	have to be long format in Unix time	Unix time of year/month/date (unixtimestamp.com)
Response				
Field	Type	Sample	Constraint	Description
message	string	your token is invalid		Error message

This use case details how managers view and analyze sales data over time, including changing time ranges and handling invalid input conditions.

Troy University Resources

Use Case Name: Manager UI – Sale Data			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display a chart summarizing the sale data in a period of time
Main	2	Manager	Change time range
Main	3	Manager	Apply change
Main	4	System	Get and verify information inside input fields
Main	5	System	Display a new chart in the given time range
Alternative	5a	System	If information doesn't meet the conditions then pop up error message

Table 3: Sale Data – Manager UI

This use case details how cashiers manage the physical layout of the cafeteria, including adding, removing, and selecting areas on the map. It also covers error handling for duplicate area names or attempts to delete areas containing tables.

3.4.2 Order Processing Use Case

Use Case Name: Cashier UI – Area Management			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display the map layout
Main	2	Cashier	Open map
Main	3	System	Show available areas on the map
Main	4	Cashier	Add a new area
Main	5	Cashier	Remove an existing area
Main	6	System	Update the map layout after area modification
Main	7	Cashier	Select an area
Main	8	System	Display area
Alternative	4a	System	Pop up error message if the area name already exists
Alternative	5a	System	Pop up error message if the area contains tables and cannot be deleted

Table 4: Area Management – Cashier UI

This use case outlines the process for managing tables within areas, such as adding, removing, editing, and updating table statuses. It includes alternative flows for handling errors when trying to delete tables in use.

Use Case Name: Cashier UI – Table Management			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display the map layout
Main	2	Cashier	Open map
Main	3	System	Show available areas on the map
Main	4	Cashier	Select an area
Main	5	System	Display area
Main	6	Cashier	Add a new table to the selected area
Main	7	Cashier	Remove a table from the selected area
Main	8	System	Update the table list after modification
Main	9	Cashier	Select a table within an area
Main	10	Cashier	Edit table status
Main	11	System	Update and display the new table status
Alternative	7a	System	Pop up error message if the area is in service and cannot be deleted

Table 5: Table Management – Cashier UI

This use case describes the process for cashiers to place orders offline, from selecting a table and drinks to sending the order. It includes error handling for out-of-stock items.

Troy University Resources

Use Case Name: Cashier UI – Order Offline			
Flow	**No.**	**Actor**	**Action**
Main	1	Cashier	Select table
Main	2	Cashier	Select add drink
Main	3	System	Open order web
Main	4	System	Display menu with available drinks
Main	5	Cashier	Open menu
Main	6	System	Display drink options
Main	7	Cashier	Select a drink
Main	8	Cashier	Send the order
Alternative	8a	System	Pop up error message if the drink is out of stock

Table 6: Order Offline – Cashier UI

This use case covers the customer's experience ordering drinks online, including menu navigation, cart management, and order submission. It also details alternative flows for out-of-stock items and empty carts.

Use Case Name: Customer UI – Order Online			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display menu with available drinks
Main	2	Customer	Open menu
Main	3	System	Display drink options
Main	4	Customer	Select a drink
Main	5	Customer	Add the selected drink to the cart
Main	6	System	Update cart and display current items
Main	7	Customer	View cart
Main	8	System	Display cart
Main	9	Customer	Submit the cart
Main	10	System	Send cart data to the server
Main	11	Customer	View ordered
Main	12	System	Display ordered
Alternative	5a	System	Pop up error message if the drink is out of stock
Alternative	8a	System	Pop up error message if the cart is empty
Alternative	10a	System	Pop up error message if the cart is empty when submitting
Alternative	12a	System	Display message if no orders have been submitted previously

Table 7: Order Online – Customer UI

These use cases describe how cashiers handle billing, manage orders, and review bill history, ensuring efficient and accurate transaction processing.

Use Case Name: Cashier UI – Bill			
Flow	**No.**	**Actor**	**Action**
Main	1	Cashier	Select table
Main	2	Cashier	Select bill
Main	3	System	Display bill
Main	4	Cashier	Select print

Table 8: Bill – Cashier UI

Troy University Resources

Use Case Name: Cashier UI – Order Management			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display the list of orders
Main	2	Cashier	Select an order
Main	3	Cashier	Confirm or delete the selected order
Main	4	System	Update order status

Table 9: Order Management – Cashier UI

Use Case Name: Cashier UI – Bill History			
Flow	**No.**	**Actor**	**Action**
Main	1	System	Display the list of bills
Main	2	Cashier	Select a bill
Main	3	System	Display the bill

Table 10: Bill History – Cashier UI

This use case describes how storage managers and the server interact to display inventory dashboards, including daily import/export reports and ingredient status.

3.4.3 Inventory Management

Use Case Name: Inventory – Dashboard			
Flow	**No.**	**Actor**	**Action**
Main	1	Storage Manager	Choose “Dashboard”
Main	2	Server	Display Report daily import-export & Running out ingredients
Main	3	Storage Manager	Choose sub-menu
Main	4	Server	Display corresponding submenu

Table 11: Dashboard – Inventory Management

This use case covers the process of viewing and managing inventory, including running out ingredient alerts and submenu navigation.

Use Case Name: Inventory – Inventory Management			
Flow	**No.**	**Actor**	**Action**
Main	1	Storage Manager	Choose “Inventory”
Main	2	Server	Display Inventory, Running out ingredients
Main	3	Storage Manager	Choose sub-menu
Main	4	Server	Display corresponding submenu

Table 12: Inventory – Inventory Management

This use case details the workflow for importing new ingredients, including input validation and error handling for invalid quantities.

Use Case Name: Import – Inventory Management			
Flow	**No.**	**Actor**	**Action**
Main	1	Storage Manager	Choose “Import”
Main	2	Server	Display Import Interface
Main	3	Storage Manager	Input ingredient’s name, quantity (*) and suppliers
Main	4	Server	Check valid and display information
Alternative	4a	Server	Inform error if quantity is invalid

Table 13: Import – Inventory Management

This use case outlines the daily inventory check process, comparing stocked ingredients and handling discrepancies or errors.

Troy University Resources

Use Case Name: Daily-check – Inventory Management			
Flow	**No.**	**Actor**	**Action**
Main	1	Storage Manager	Choose "Daily-check"
Main	2	Server	Display stocked ingredients from the previous day and at the current time
Main	3	Storage Manager	Input actual stocked ingredients to compare
Main	4	Server	Check if the two amount are equal or not
Alternative	4a	Server	Display equal
Alternative	4b	Server	Warning: transaction not be recorded or loss or discrepancy in sales.

Table 14: Daily-check – Inventory Management

This use case describes how storage managers and the server interact to review import/export history, including error handling for invalid dates.

Use Case Name: History – Inventory Management			
Flow	**No.**	**Actor**	**Action**
Main	1	Storage Manager	Choose "History"
Main	2	Server	Display Import & Export ingredients
Main	3	Storage Manager	Choose transaction date (*)
Main	4	Server	Check valid and display transaction history
Alternative	4a	Server	Inform error if date is invalid

Table 15: History – Inventory Management

This use case describes the workflow for bartenders managing beverage orders. It covers the process from viewing available orders, selecting and confirming an order, to handling alternative flows such as order conflicts or cancellations. The system ensures that only one bartender can take an order at a time and provides feedback for order status changes.

3.4.4 Beverage Preparation Use Case

Use Case Name: Bartender UI – Beverage Preparation			
Flow	**No.**	**Agent**	**Action**
Main	1	System	Display List of Orders
Main	2	Bartender	Choose an Order
Main	3	System	Get and Verify Data
Main	4	System	If other Bartenders do not take the Order, choose to make that order
Alternative	4a	System	If the Order is Taken, Server pops up a message "The Order has Taken"
Main	5	System	Display the Corresponding Order
Main	6	Bartender	Choose Confirm Order, which is Finished
Alternative	6a	Bartender	Choose "Cancel"
Main	7	System	Get and Verify Data on Database
Alternative	7a	System	Update status "Untaken" to Server and Display to User
Main	8	System	Set Order's status to "Finished"

Table 16: Beverage Preparation – Bartender UI

3.5 Activity Diagram

3.5.1 System Administration – Manager

This section presents activity diagrams for the System Administration use case from the manager's perspective. The diagrams cover account management, system configuration, and administrative workflows, highlighting key decision points and exception handling.

Troy University Resources

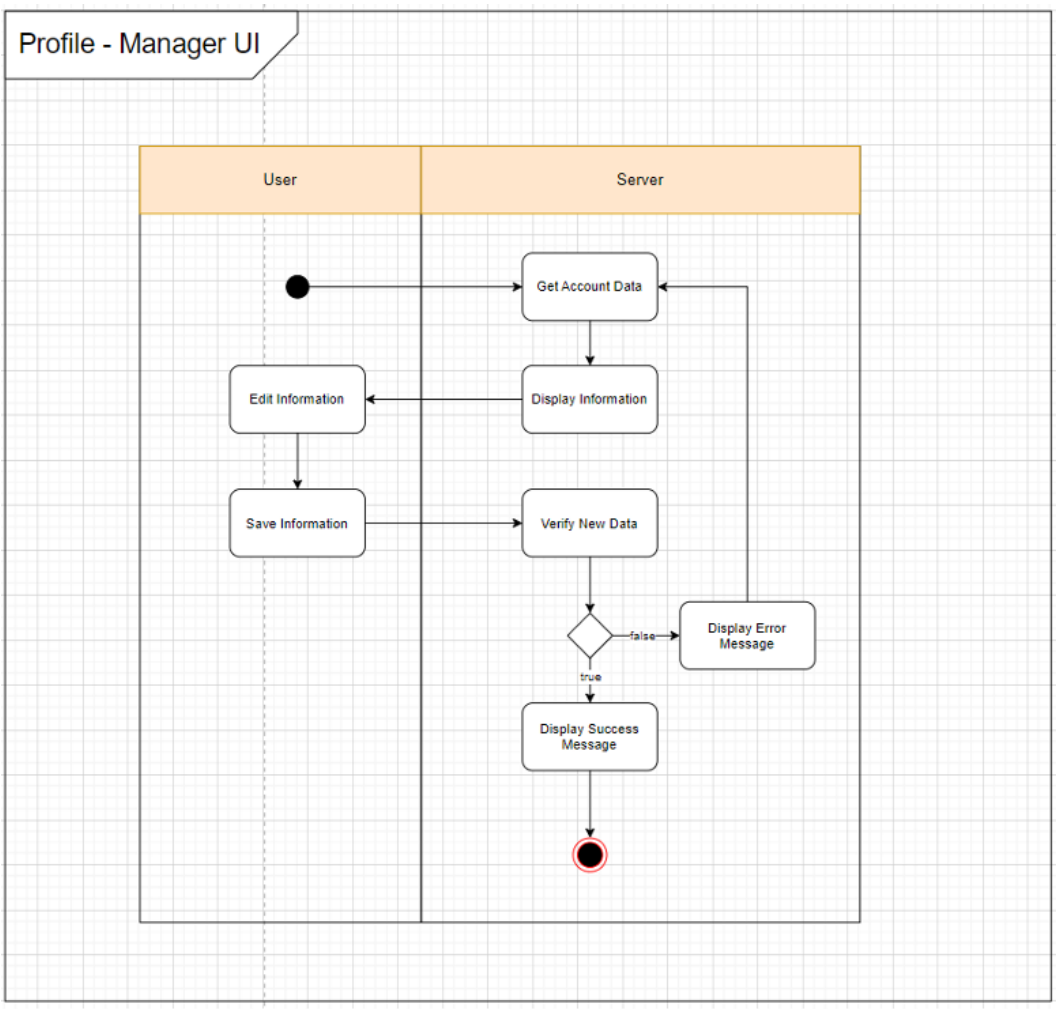


Figure 8: System Administration: Account Management

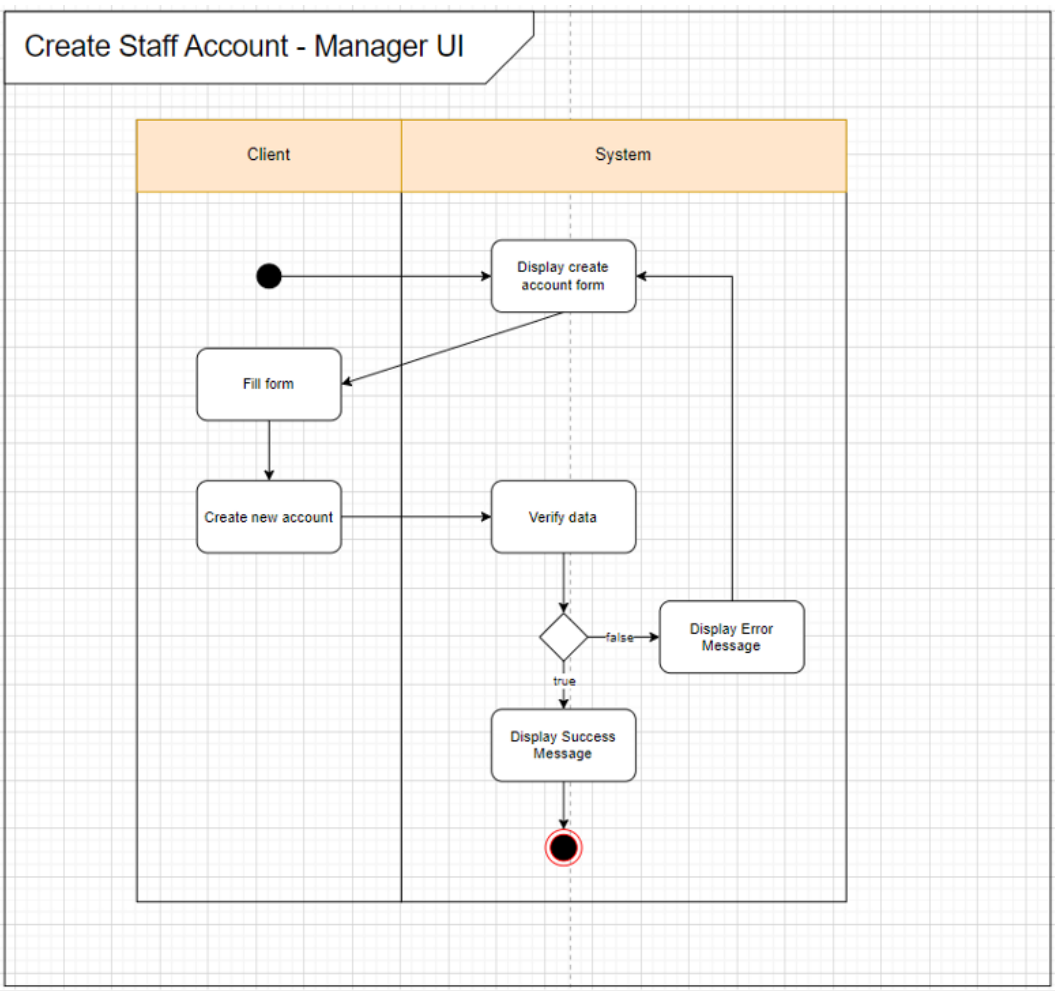


Figure 9: System Administration: System Configuration

Troy University Resources

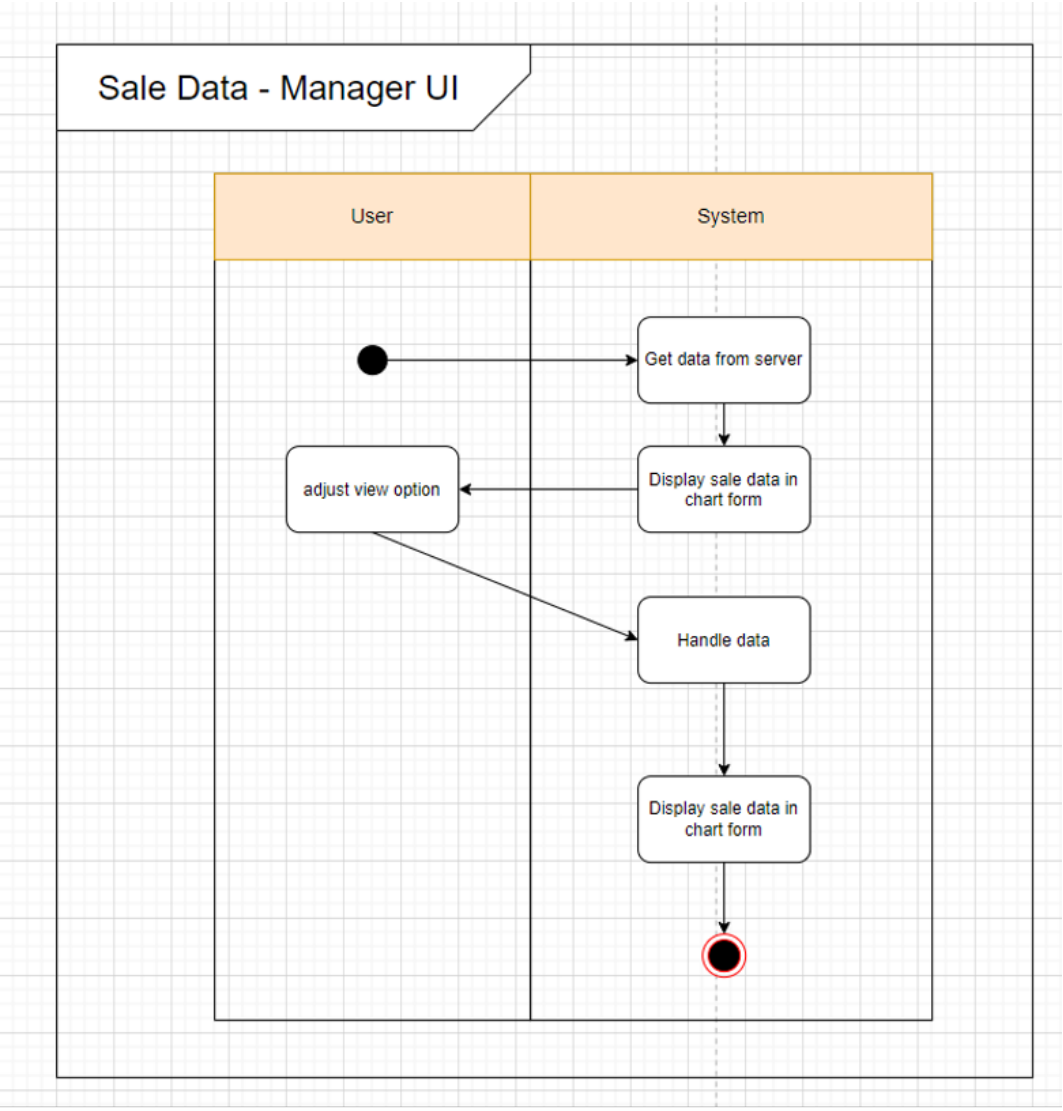
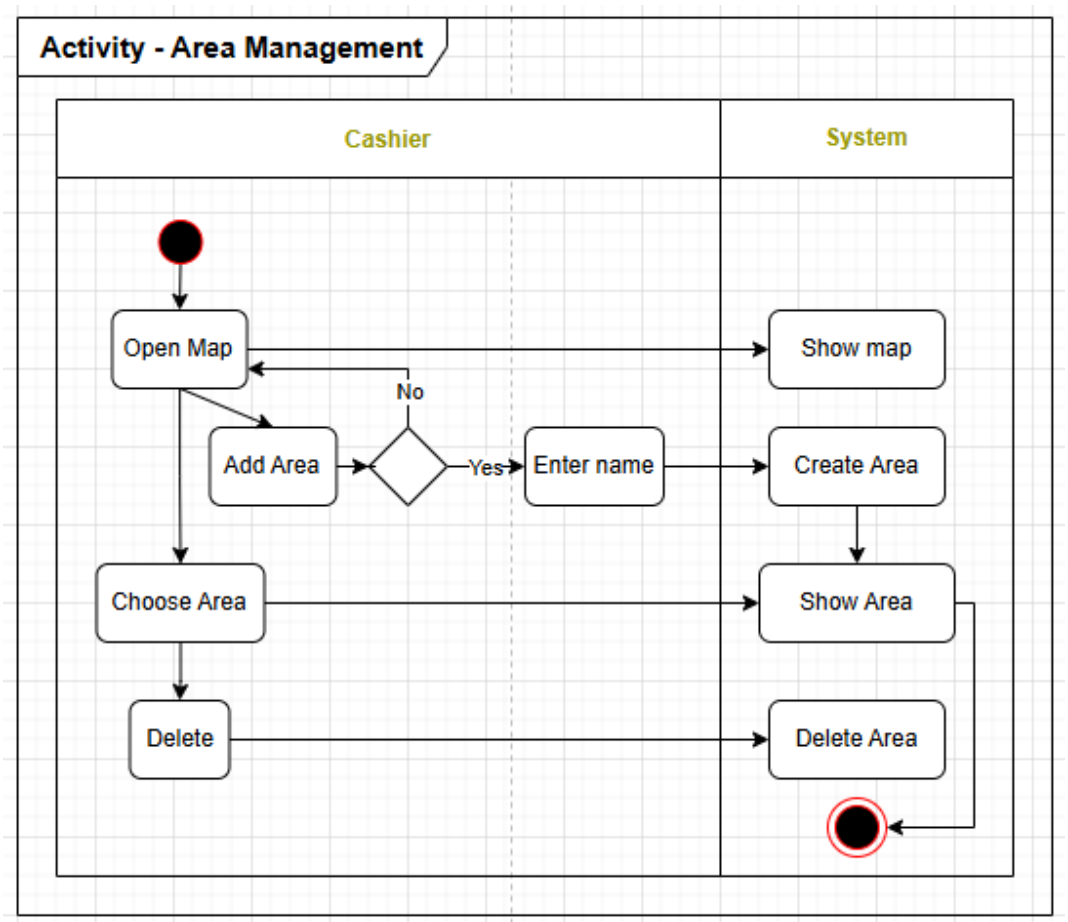


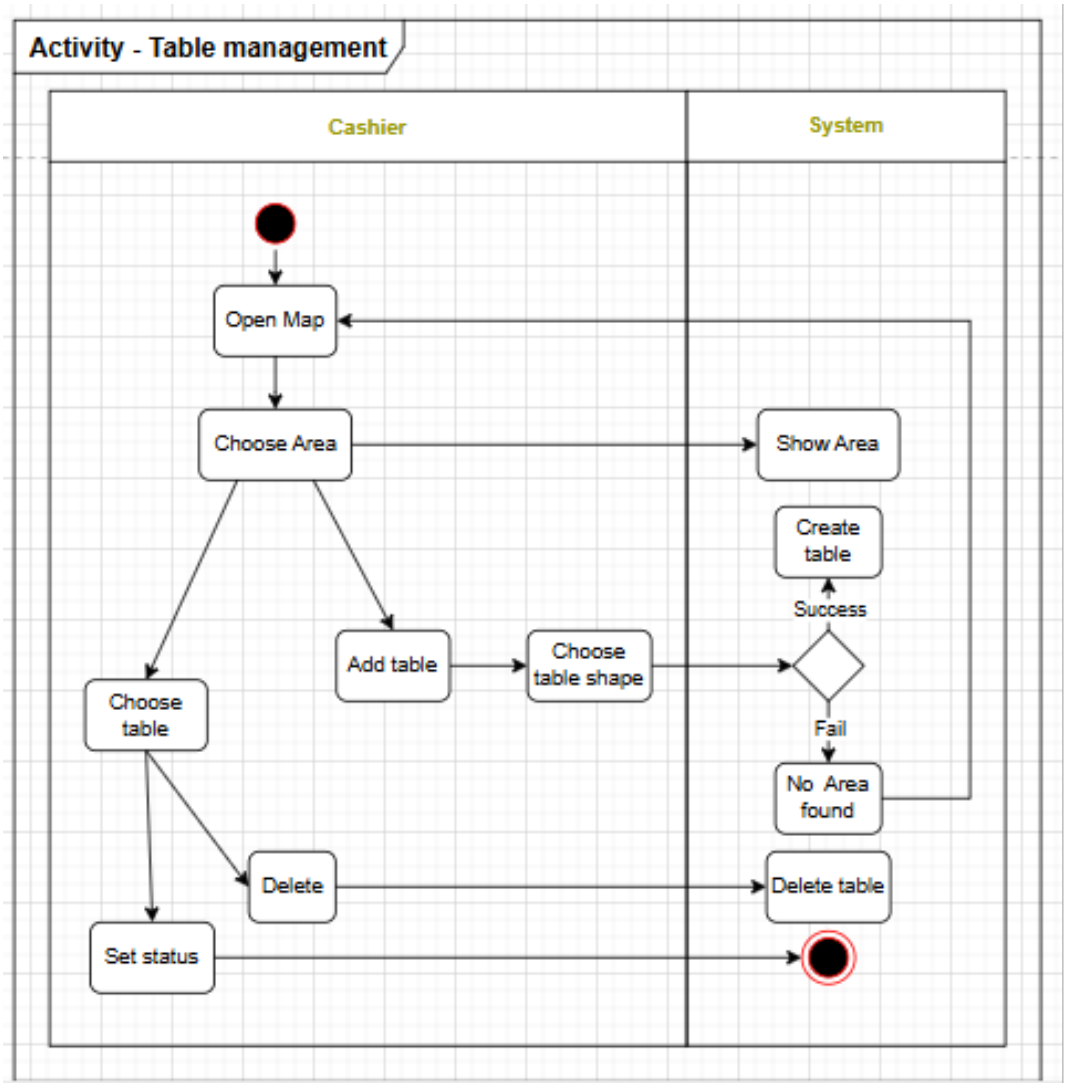
Figure 10: System Administration: Administrative Workflow

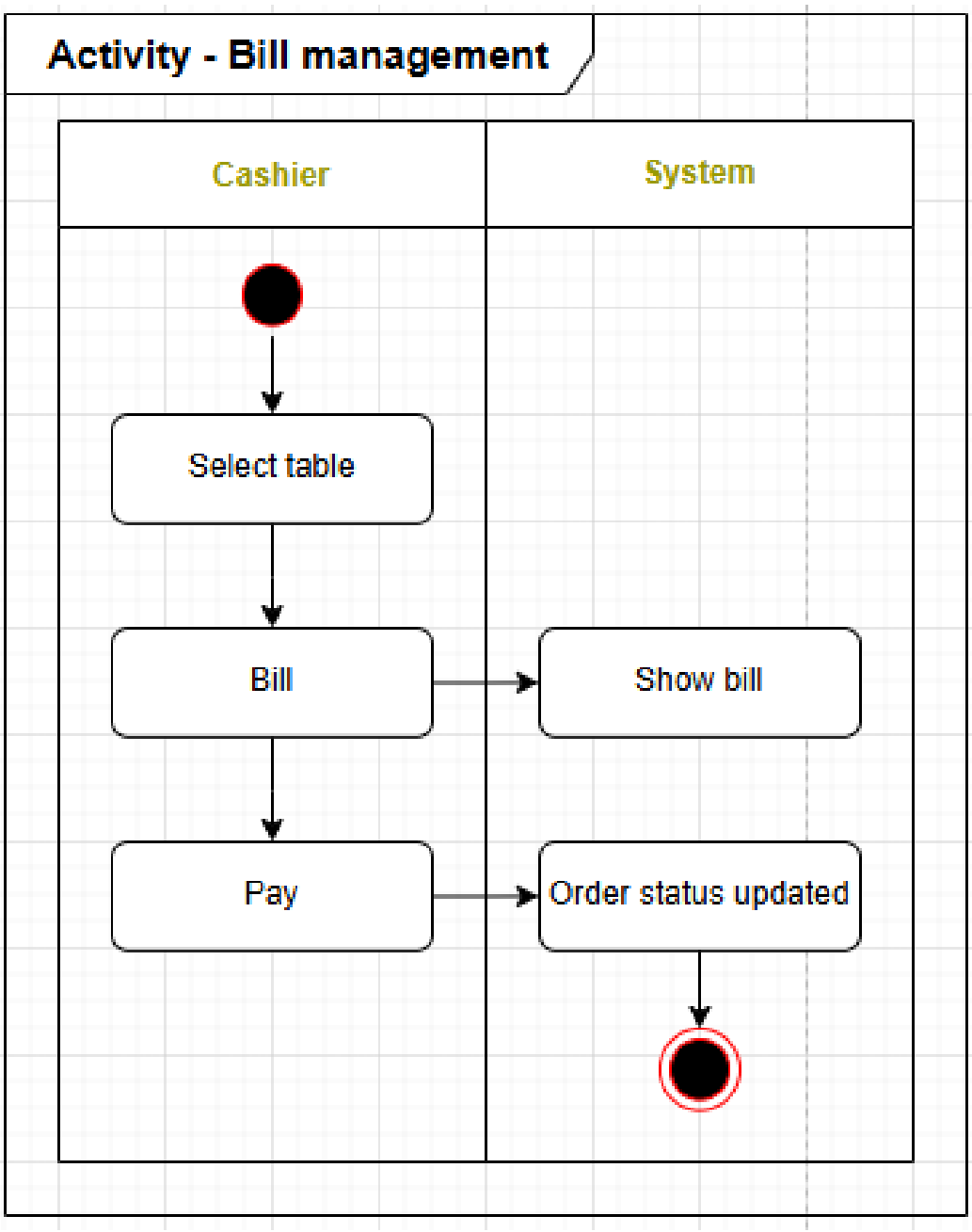
3.5.2 Order Processing Use Case

This section provides a series of activity diagrams for the Order Processing use case. These diagrams detail the steps involved in handling customer orders, from order placement to payment and fulfillment, including various scenarios and exception handling.

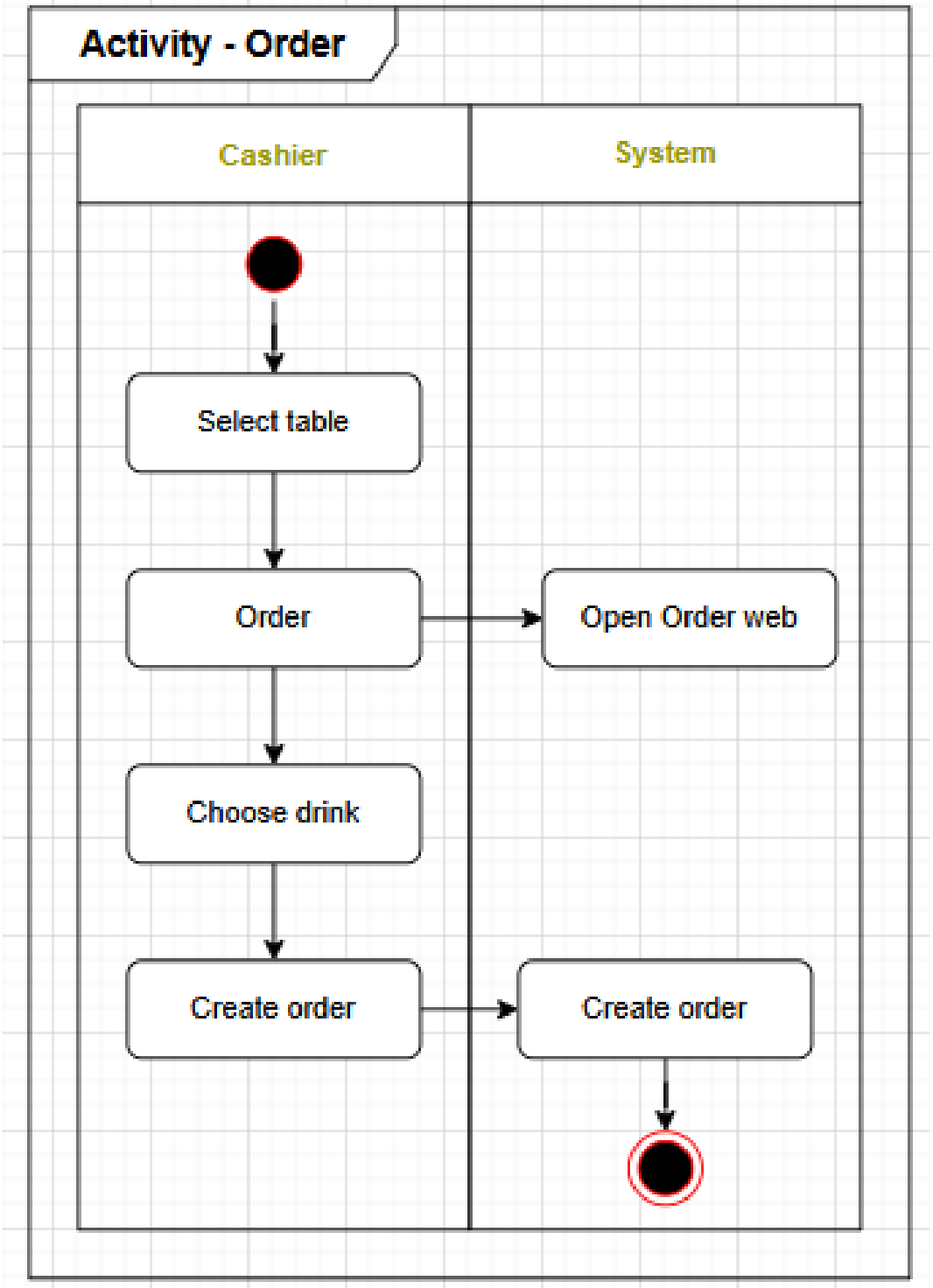


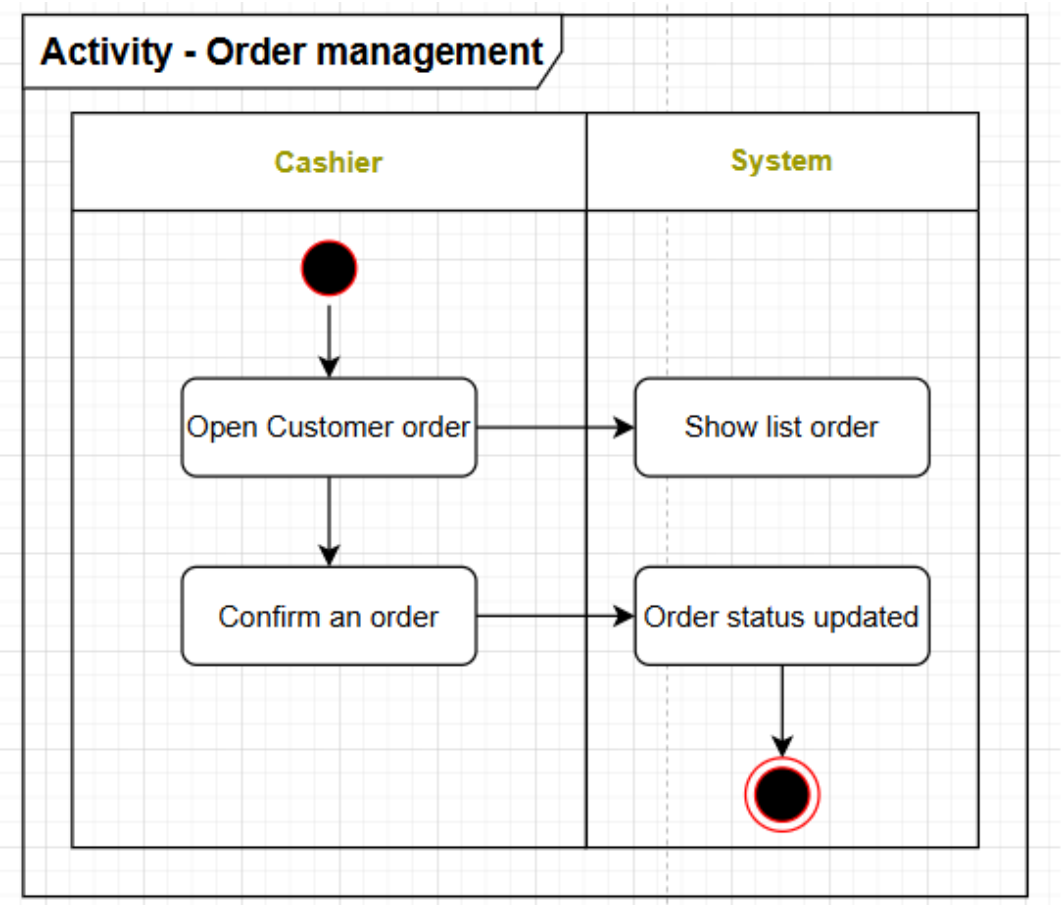
Troy University Resources

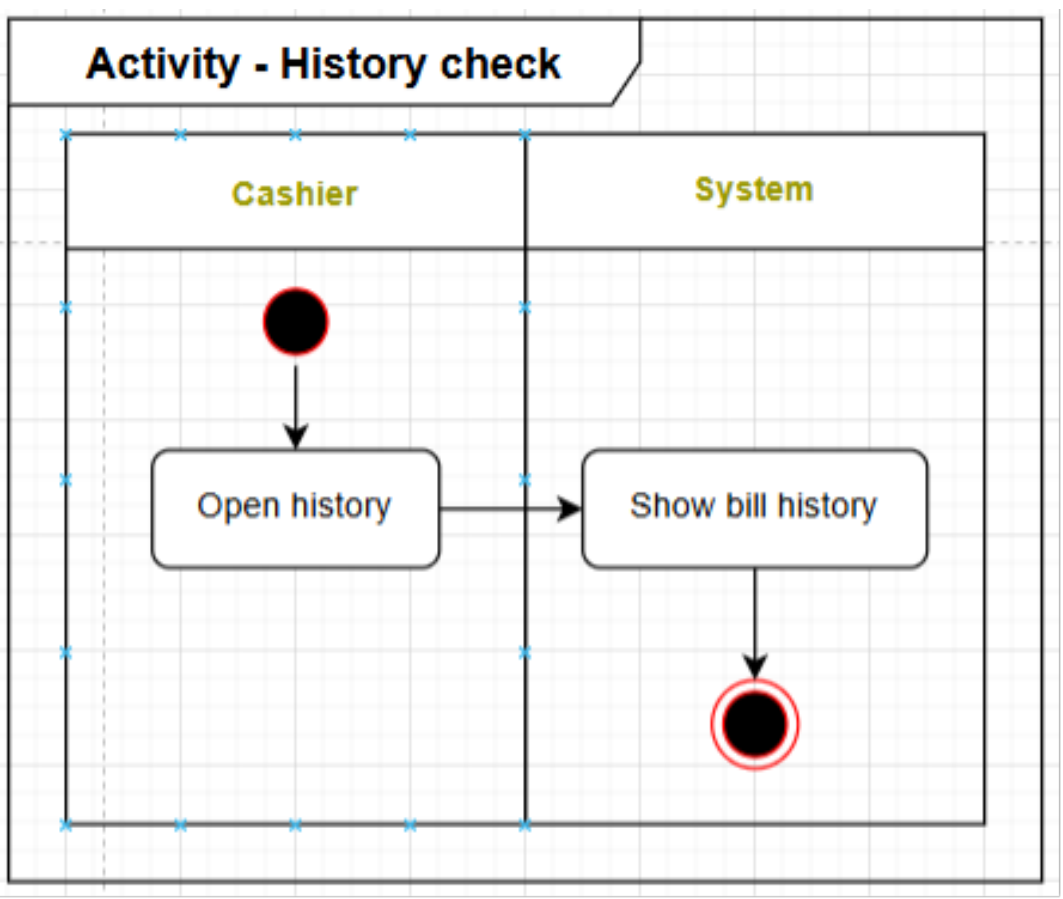

Activity - Table management
Cashier
System
Open Map
Choose Area
Show Area
Create table
Success
Add table
Choose table shape
Choose table
Fail
No Area found
Delete
Delete table
Set status



Troy University Resources


Activity - Order
Cashier
System
Select table
Order
Open Order web
Choose drink
Create order
Create order

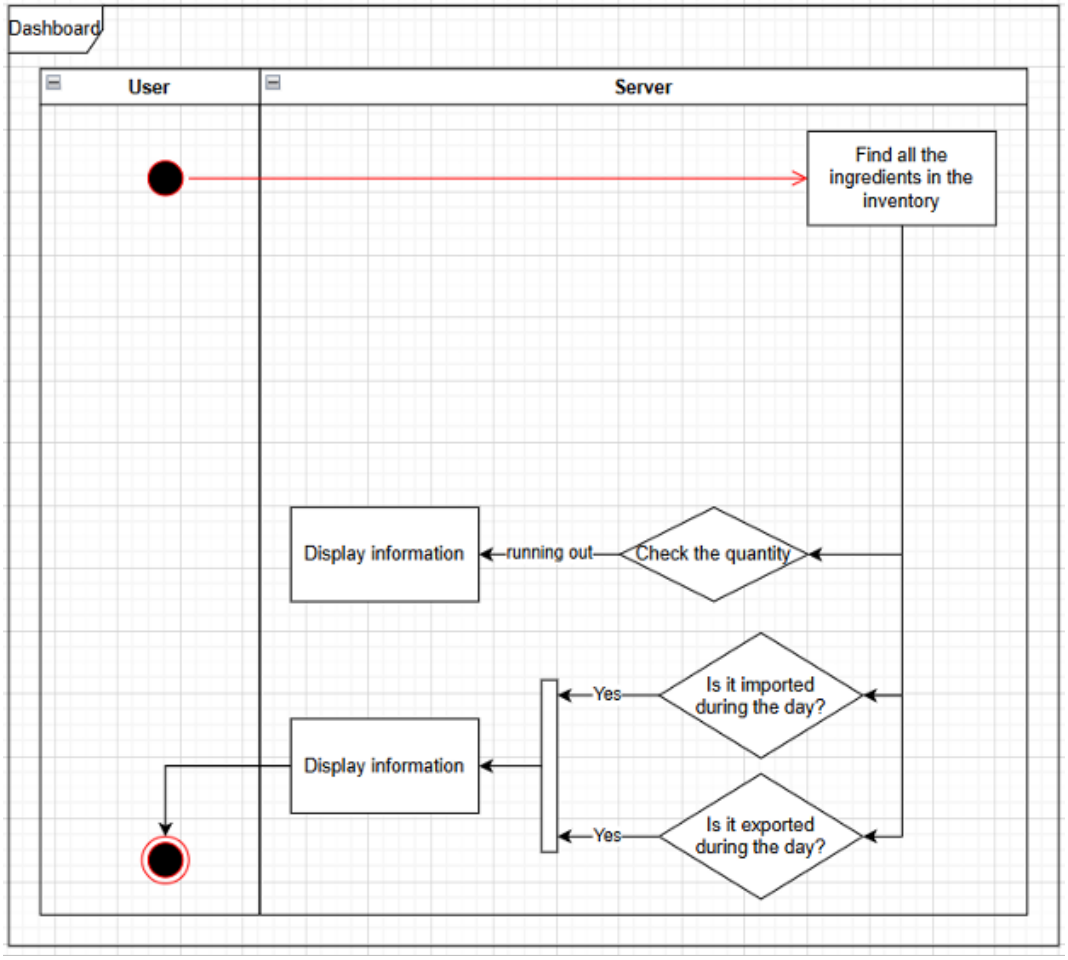


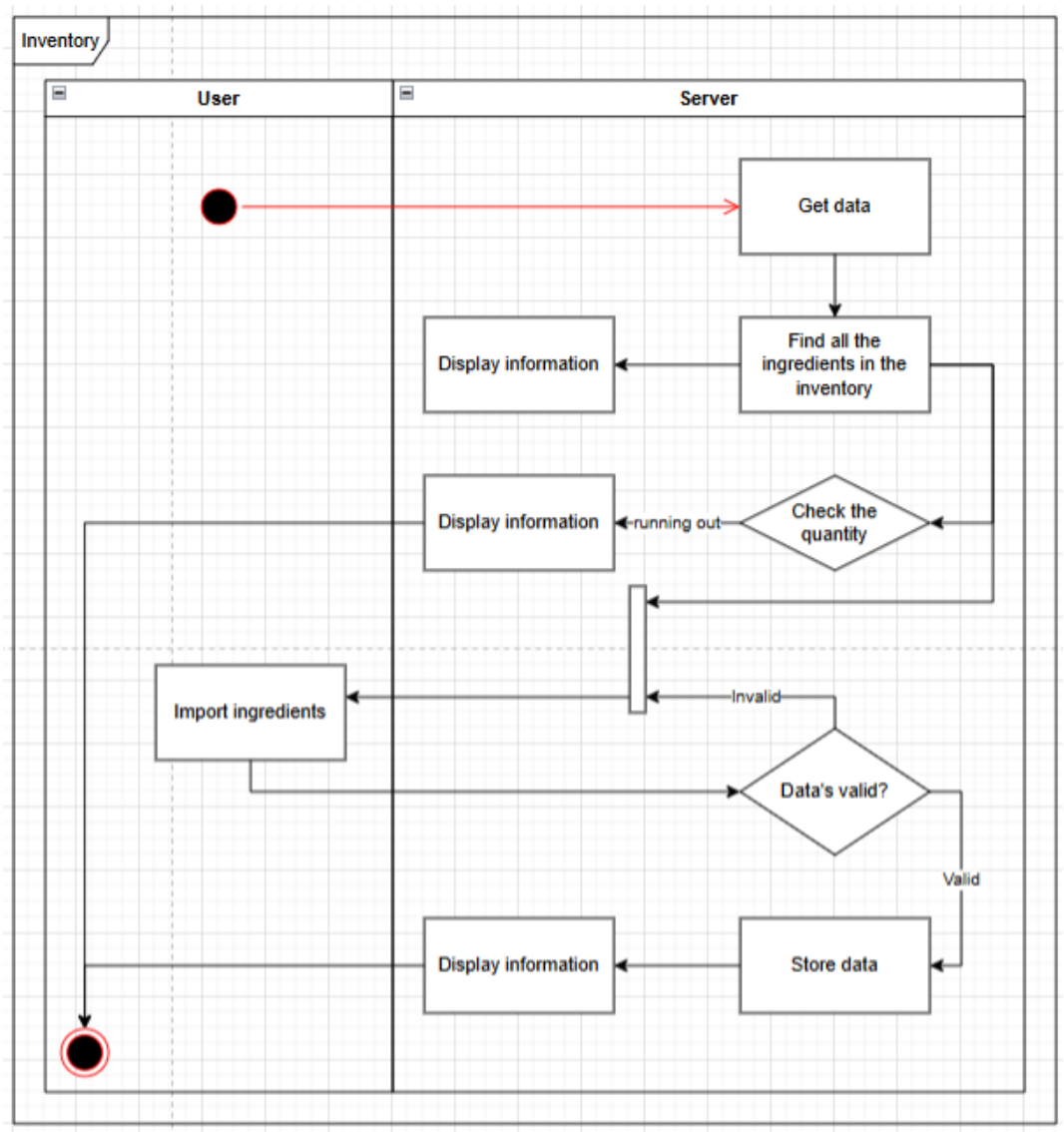


Troy University Resources

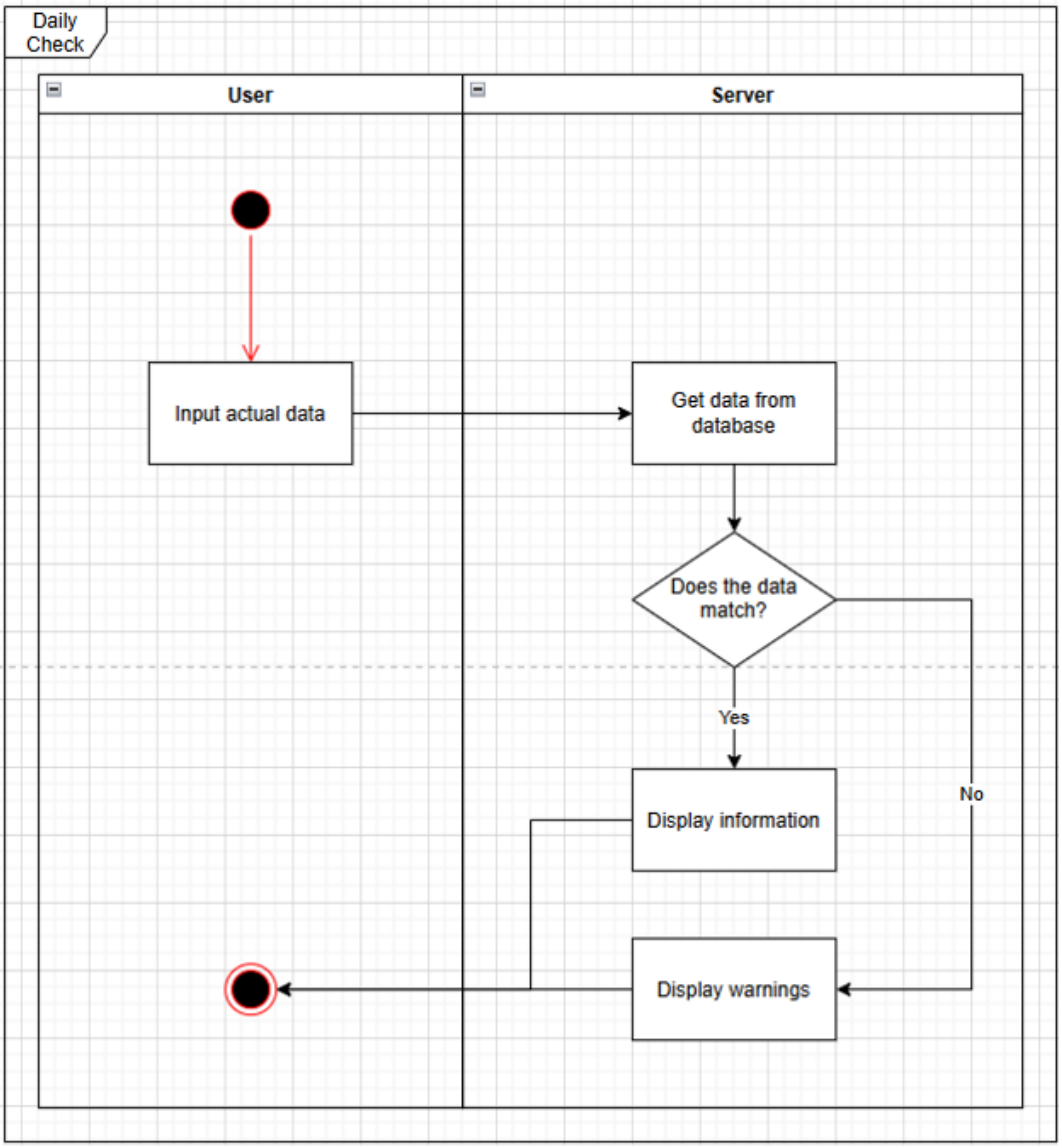
3.5.3 Inventory Management

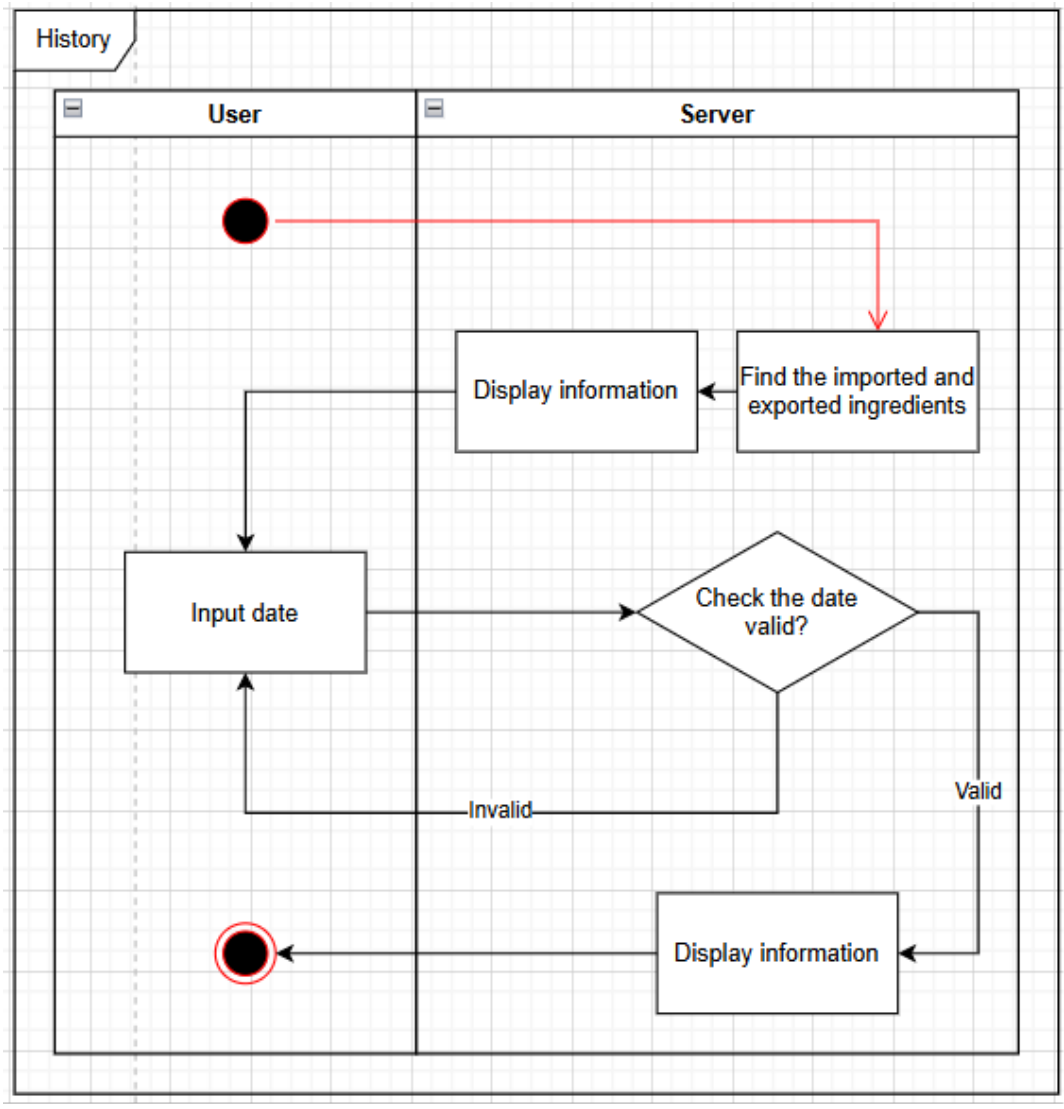
This section includes activity diagrams for the Inventory Management use case. The diagrams illustrate the processes of tracking inventory, handling imports and exports, managing stock levels, and responding to inventory-related events.





Troy University Resources

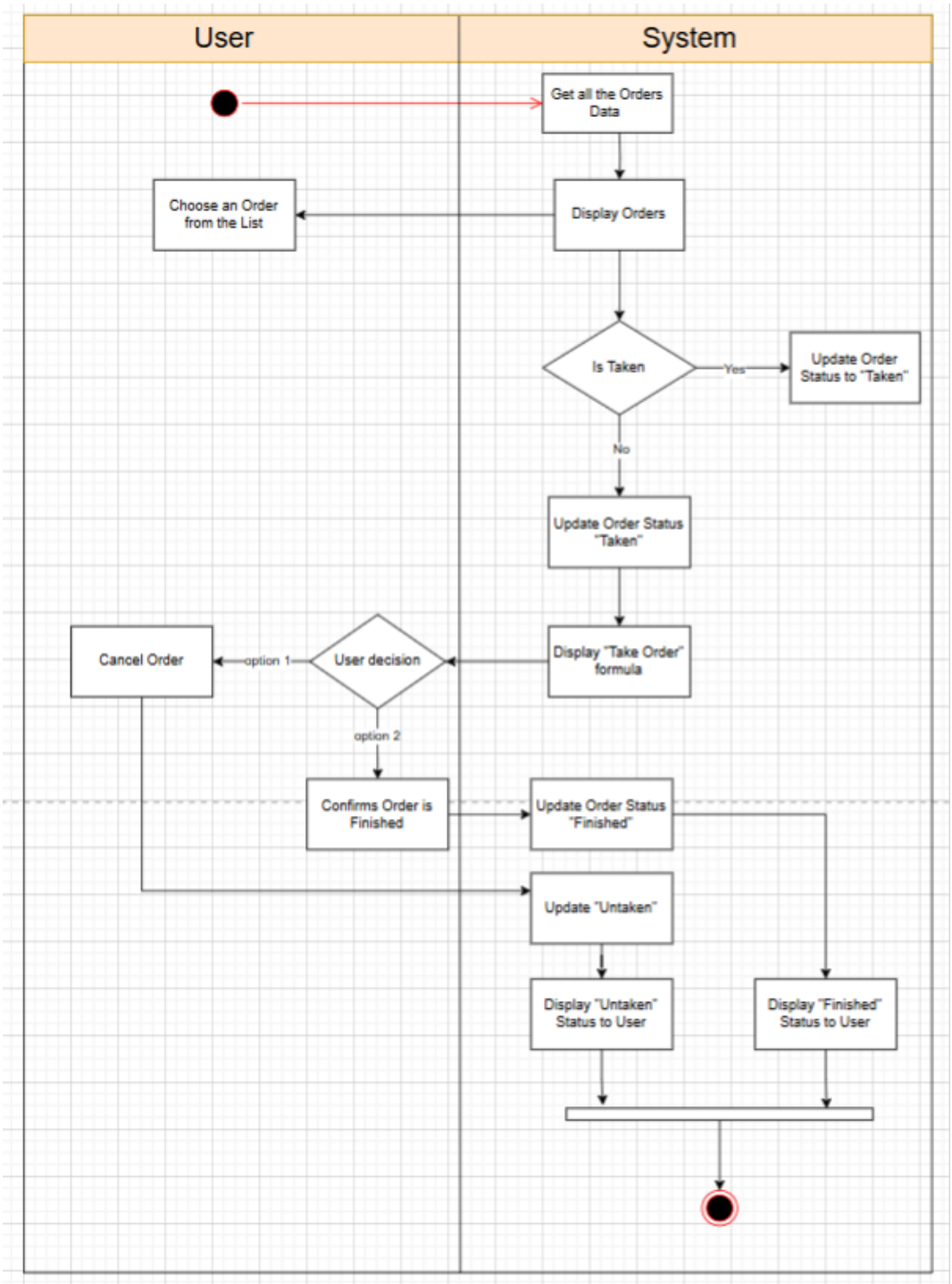

Daily Check
User
Server
Input actual data
Get data from database
Does the data match?
Yes
No
Display information
Display warnings



3.5.4 Beverage Preparation Use Case

This section presents the activity diagram for the Beverage Preparation use case. It illustrates the workflow followed by bartenders, from receiving an order to preparing and serving beverages, including decision points and possible exceptions.

Troy University Resources


User
System
Get all the Orders Data
Choose an Order from the List
Display Orders
Is Taken
Yes
Update Order Status to "Taken"
No
Update Order Status "Taken"
Cancel Order
option 1
User decision
Display "Take Order" formula
option 2
Confirms Order is Finished
Update Order Status "Finished"
Update "Untaken"
Display "Untaken" Status to User
Display "Finished" Status to User

4 System Architucture

4.1 Overall System Architecture

This section provides a comprehensive overview of the system architecture, outlining the fundamental design principles, major components, and their interactions. The architecture is designed to ensure scalability, maintainability, and efficient operation of the Cafeteria Management System.

4.2 System Design

The system is structured into several key subsystems, each responsible for a specific set of functionalities such as order processing, inventory management, beverage preparation, and system administration. These subsystems interact through well-defined interfaces, ensuring modularity and ease of maintenance. The design leverages both client-server and layered architectural patterns to separate concerns and promote reusability.

4.3 Sequential Diagrams

Sequential diagrams are used to illustrate the dynamic behavior of the system, showing how objects interact in a particular scenario of a use case. These diagrams detail the sequence of messages exchanged between system components, providing insight into the flow of control and data. They are essential for understanding the runtime interactions and for identifying potential bottlenecks or integration points.

4.4 Class Diagrams

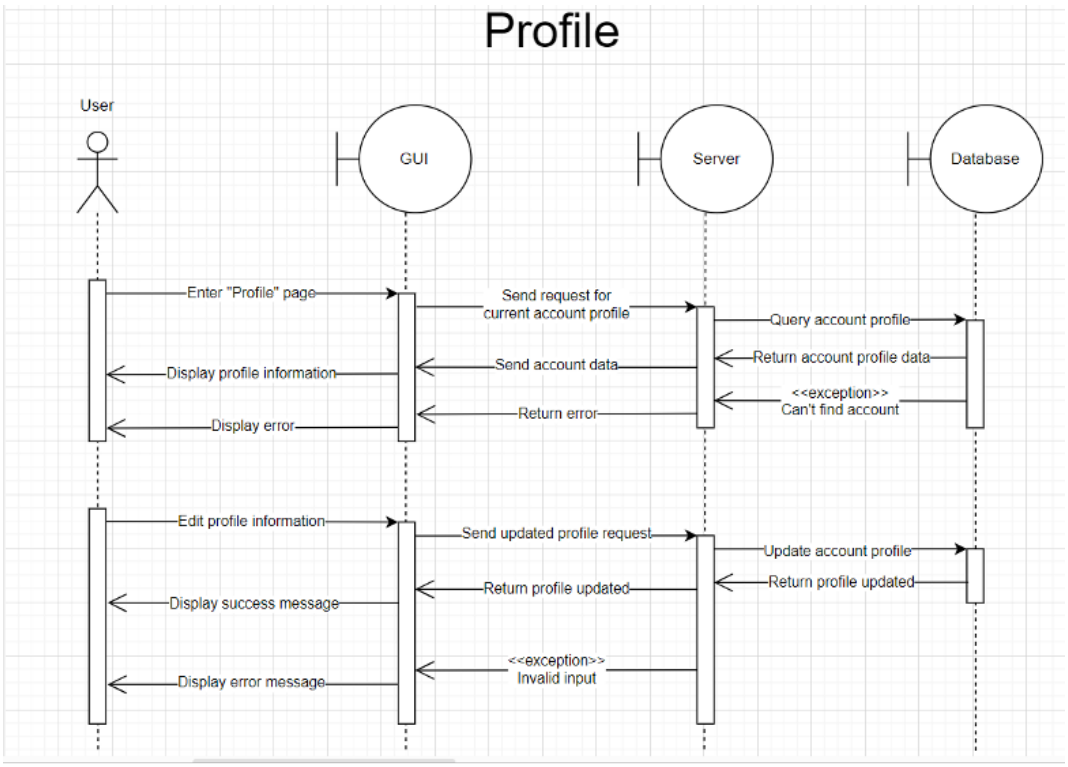
Class diagrams present the static structure of the system by depicting the system's classes, their attributes, methods, and the relationships among them. These diagrams serve as a blueprint for the system's implementation, ensuring that all necessary entities and their interactions are clearly defined. Class diagrams help in visualizing the organization of code and in maintaining consistency throughout the development process.

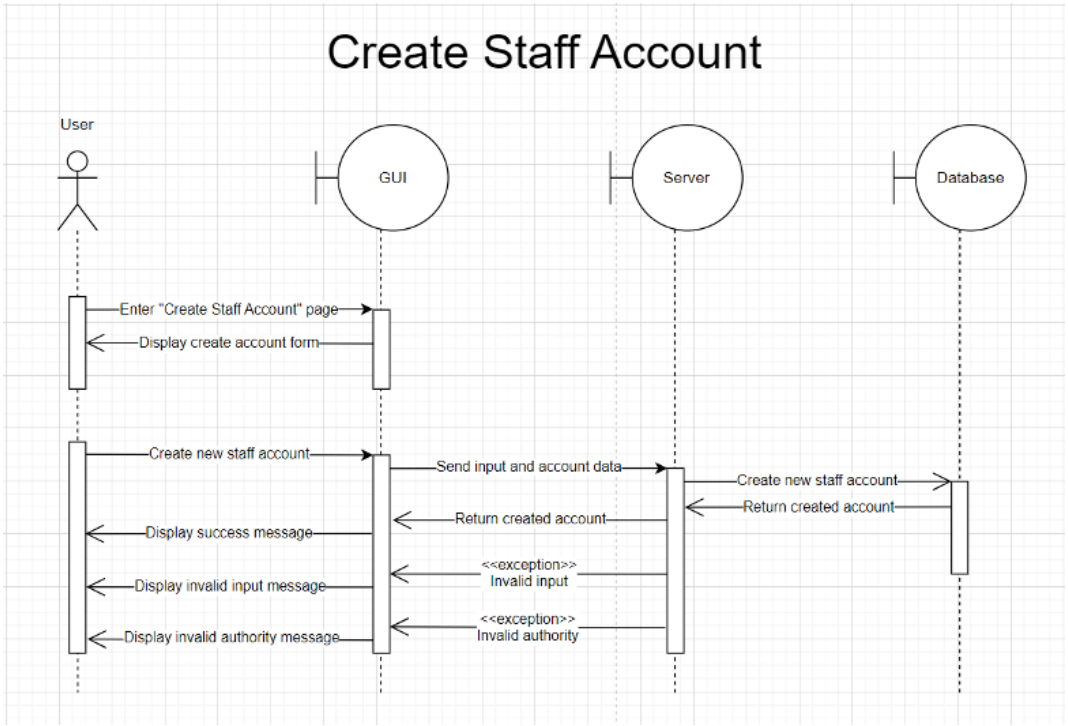
Together, these architectural views provide a holistic understanding of the system, supporting both high-level planning and detailed design activities.

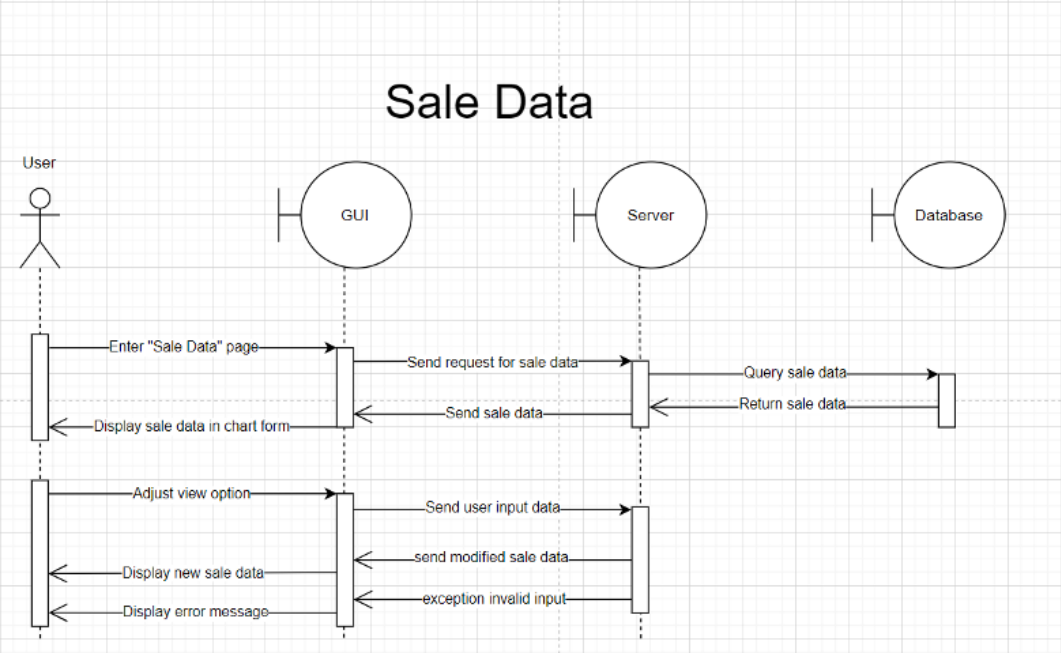
Troy University Resources

4.4.1 System Administration & Manager

This section presents the architecture diagrams for the System Administration and Manager subsystem. The diagrams illustrate the structure, main modules, and the flow of information for administrative and managerial operations.

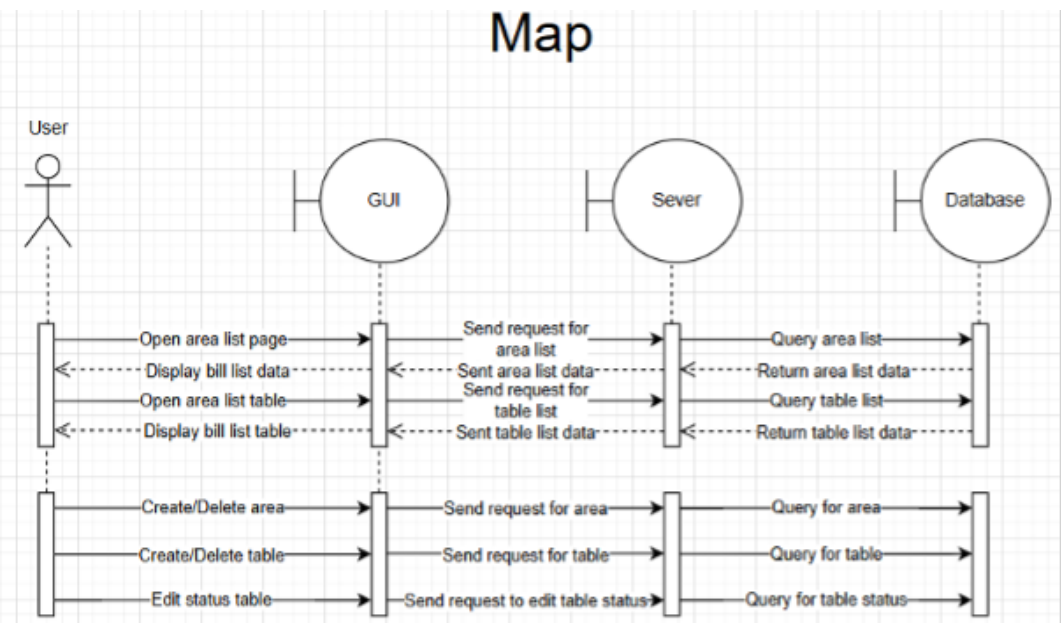




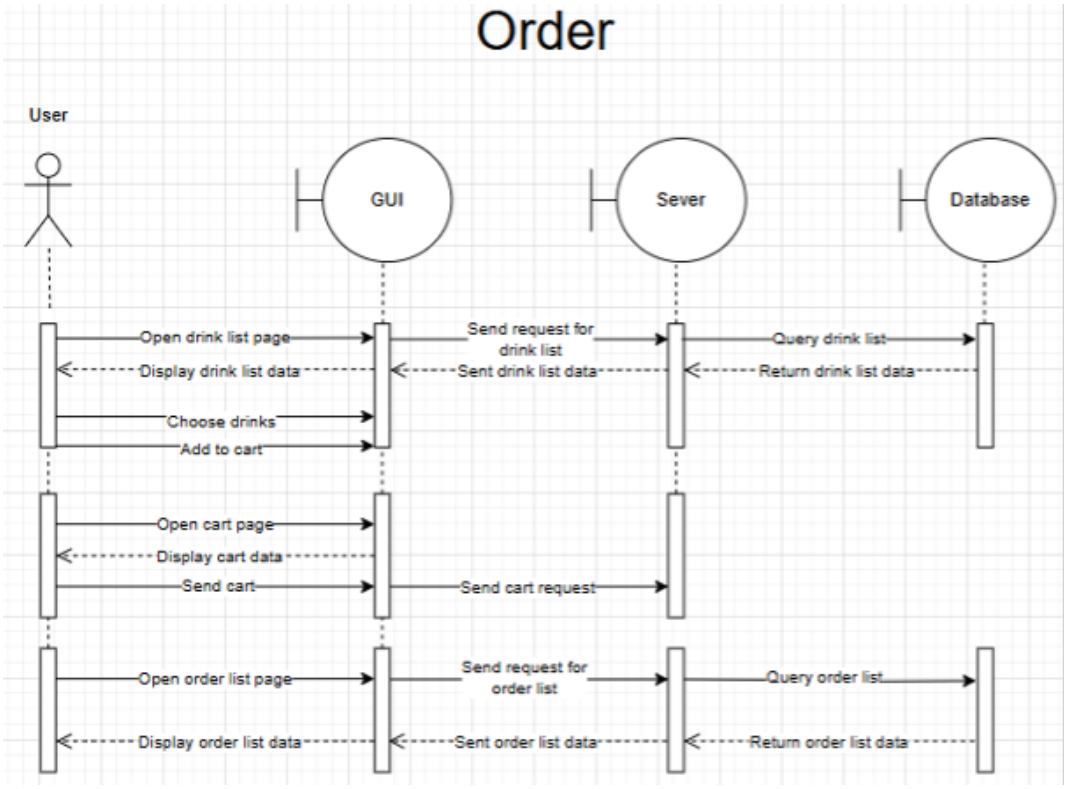


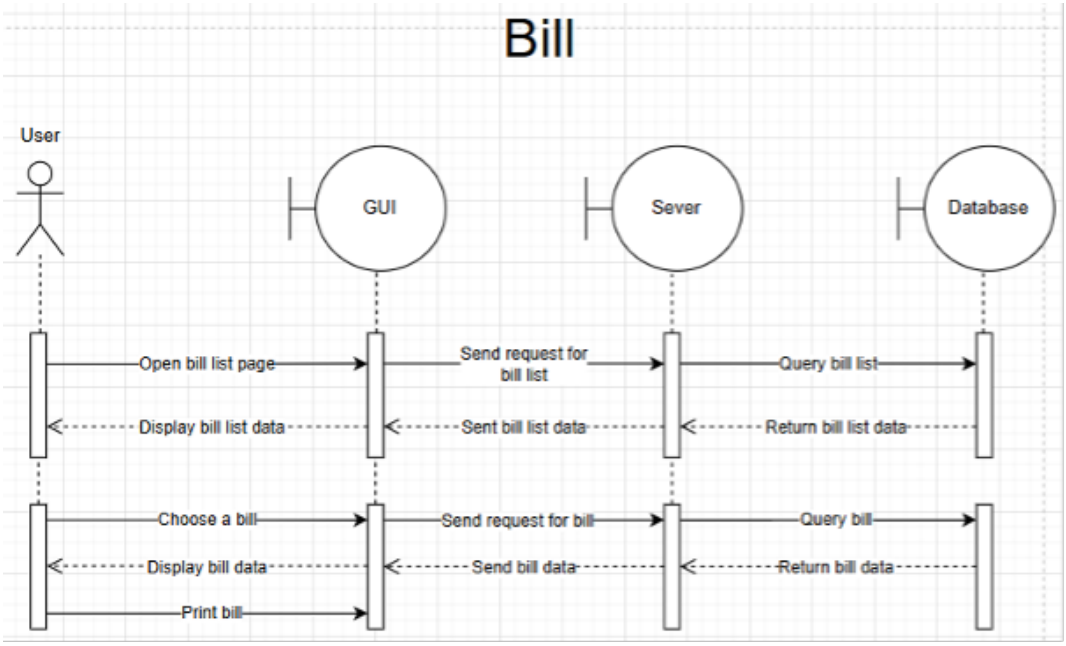
4.4.2 Order Processing/Sale

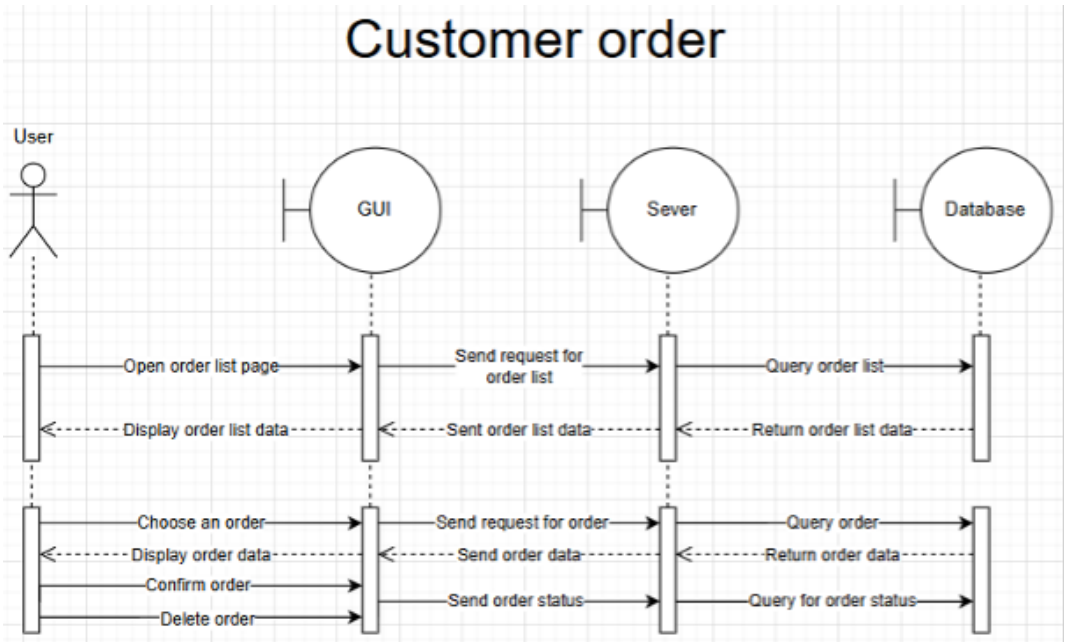
This section presents the architecture diagrams for the Order Processing and Sale subsystem. The diagrams illustrate the structure, key modules, and the flow of information between components involved in processing and completing sales transactions.

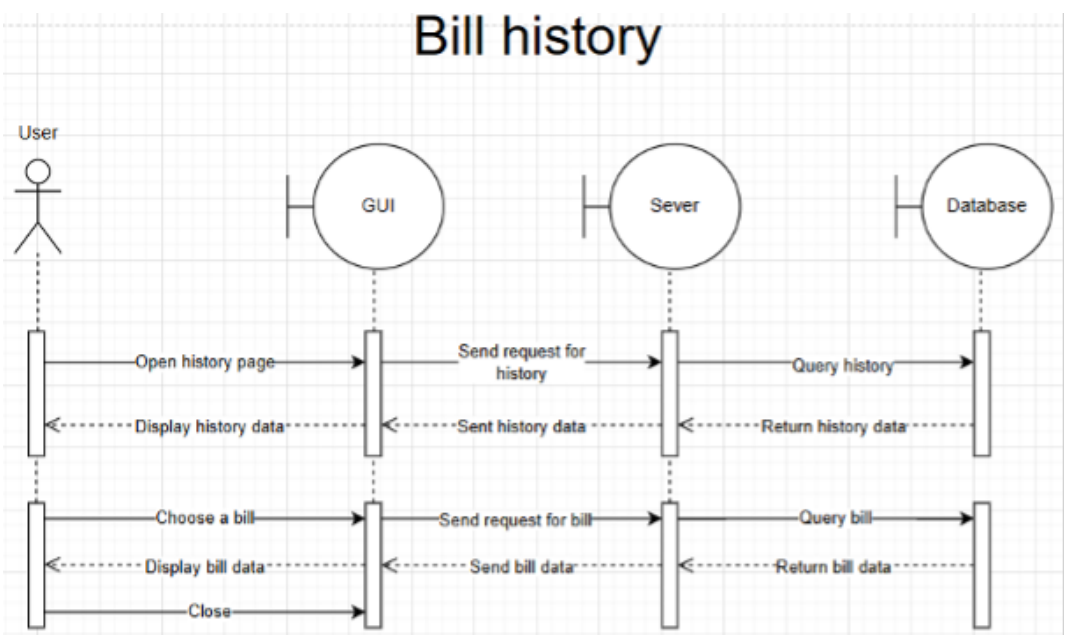


Troy University Resources


Order
User
GUI
Sever
Database
Open drink list page
Send request for drink list
Query drink list
Display drink list data
Sent drink list data
Return drink list data
Choose drinks
Add to cart
Open cart page
Display cart data
Send cart
Send cart request
Open order list page
Send request for order list
Query order list
Display order list data
Sent order list data
Return order list data


Bill
User
GUI
Sever
Database
Open bill list page
Send request for bill list
Query bill list
Display bill list data
Sent bill list data
Return bill list data
Choose a bill
Send request for bill
Query bill
Display bill data
Send bill data
Return bill data
Print bill

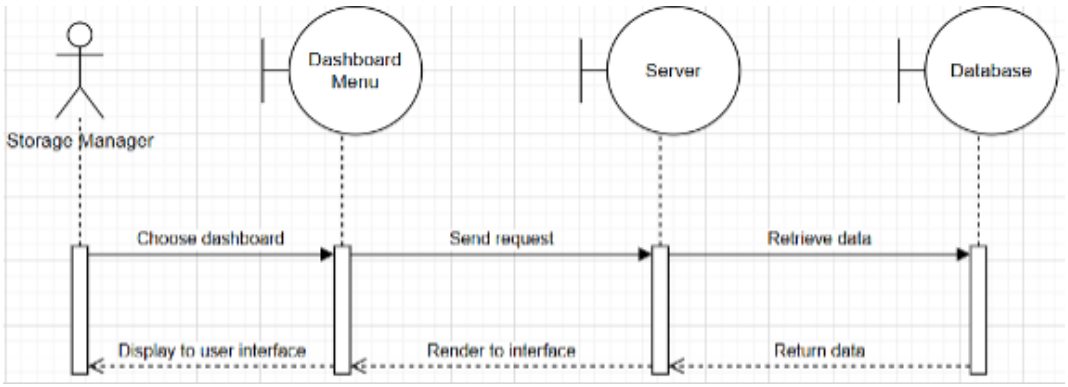


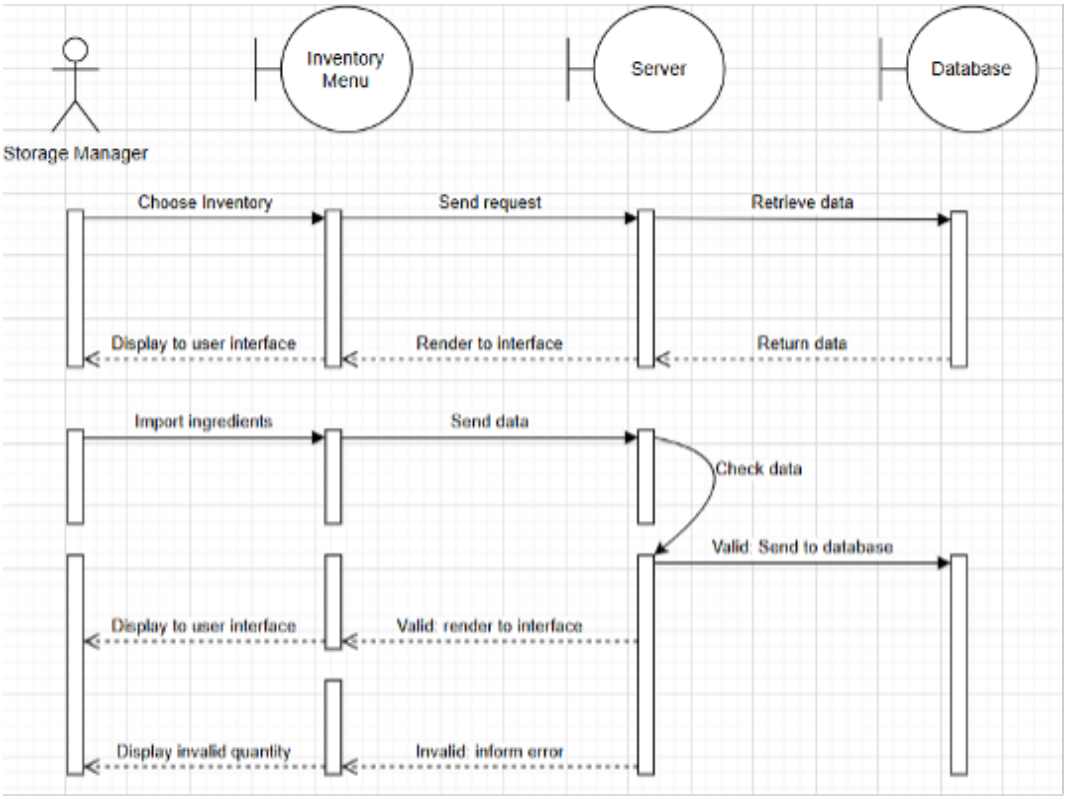


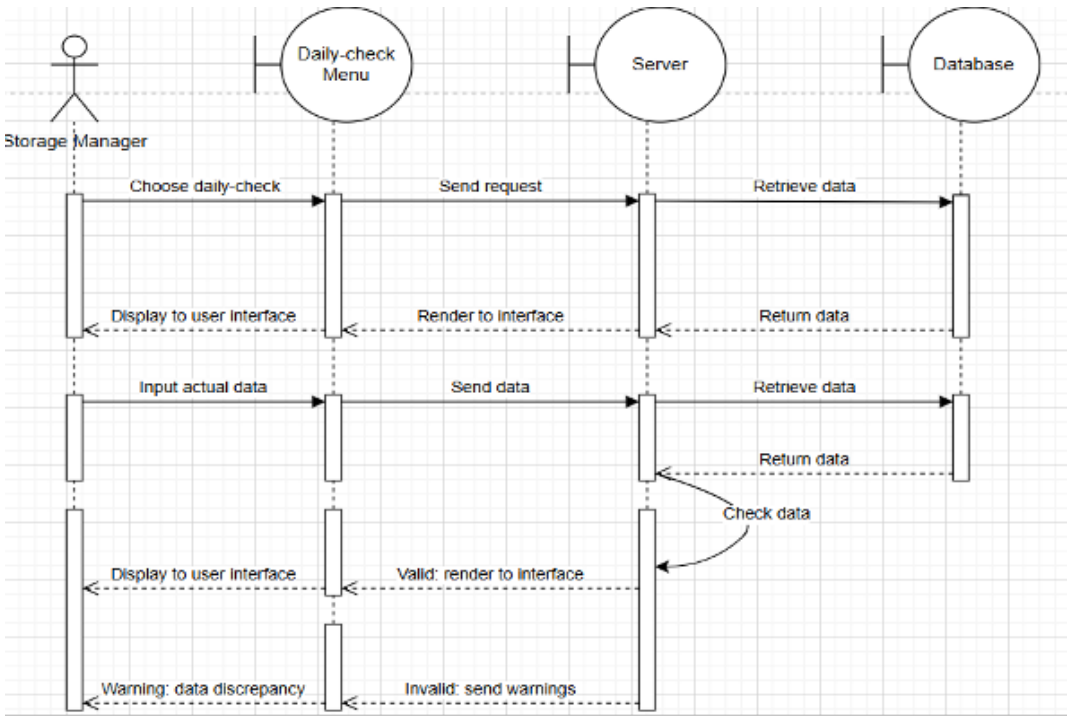
4.4.3 Inventory Management

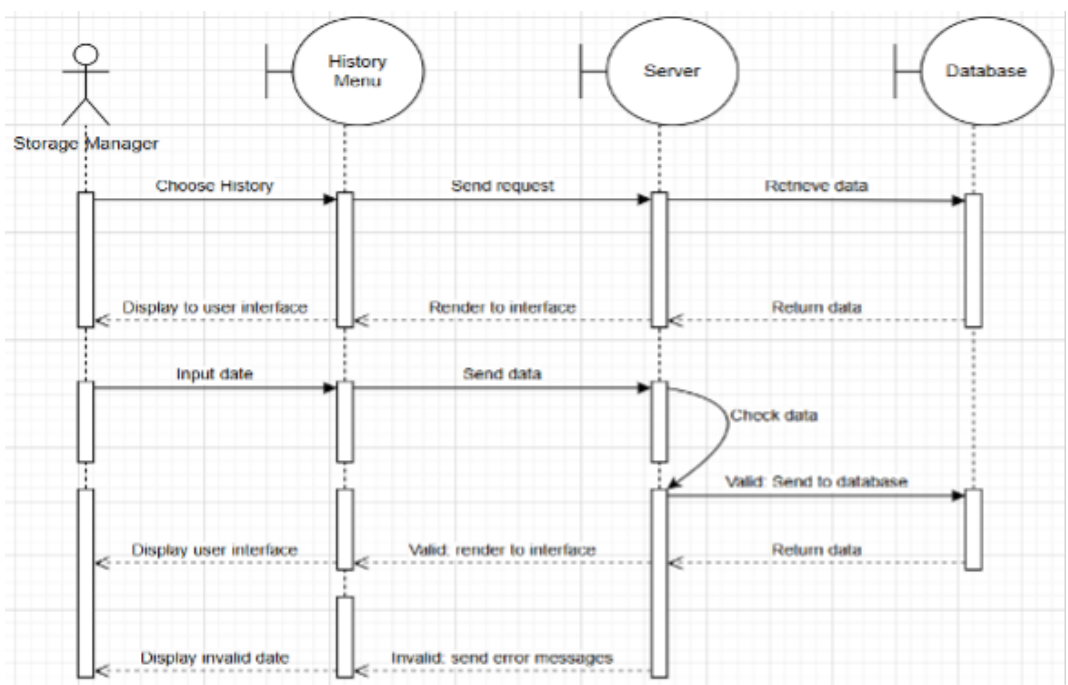
This section presents the architecture diagrams for the Inventory Management subsystem. The diagrams show the main components, their relationships, and the flow of data for inventory tracking, import/export, and stock adjustments.

Troy University Resources


Storage Manager
Dashboard Menu
Server
Database
Choose dashboard
Send request
Retrieve data
Display to user interface
Render to interface
Return data


Storage Manager
Inventory Menu
Server
Database
Choose Inventory
Send request
Retrieve data
Display to user interface
Render to interface
Return data
Import ingredients
Send data
Check data
Valid: Send to database
Display to user interface
Valid: render to interface
Display invalid quantity
Invalid: inform error

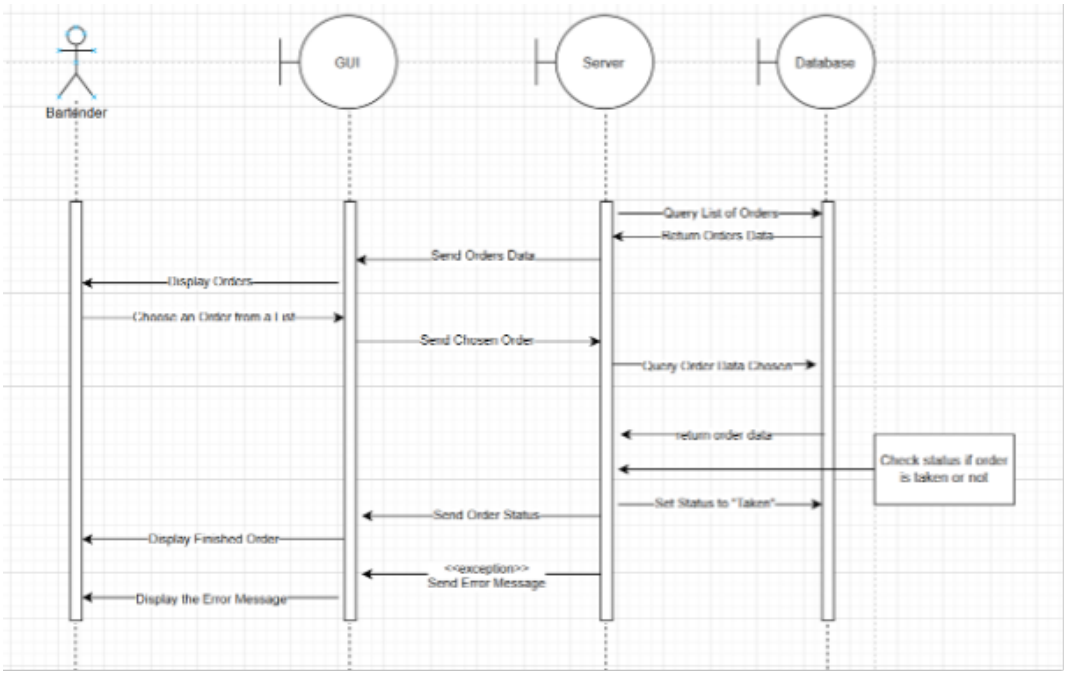


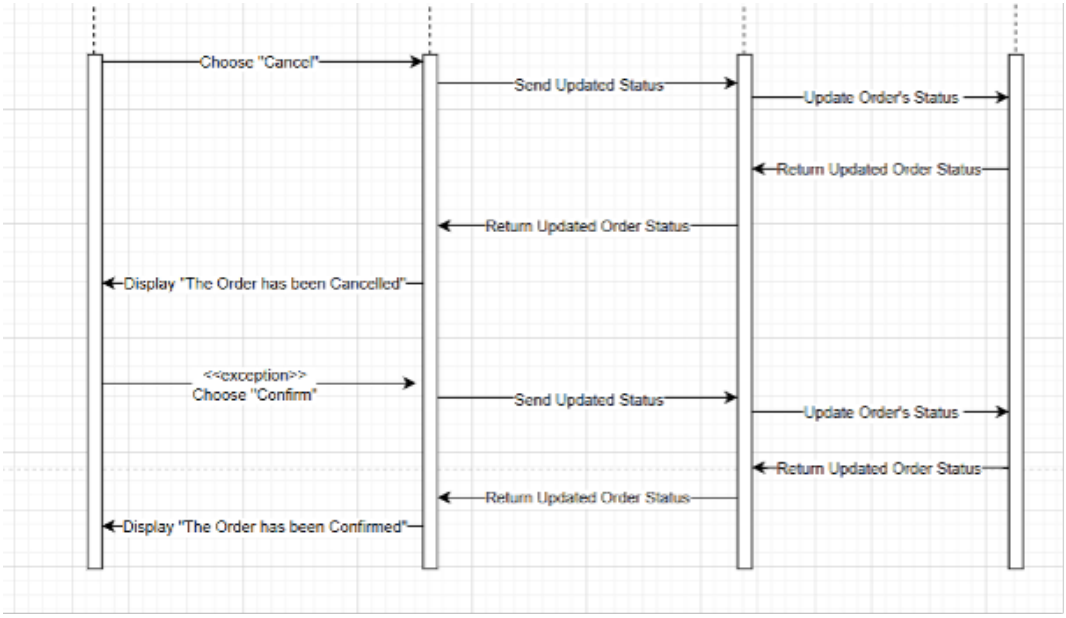


4.4.4 Beverage Preparation

This section presents the architecture diagrams for the Beverage Preparation subsystem. The diagrams illustrate the main components, their interactions, and the flow of data and control within the beverage preparation process.

Troy University Resources

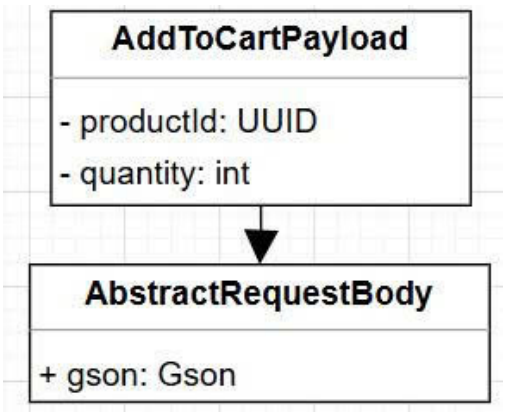


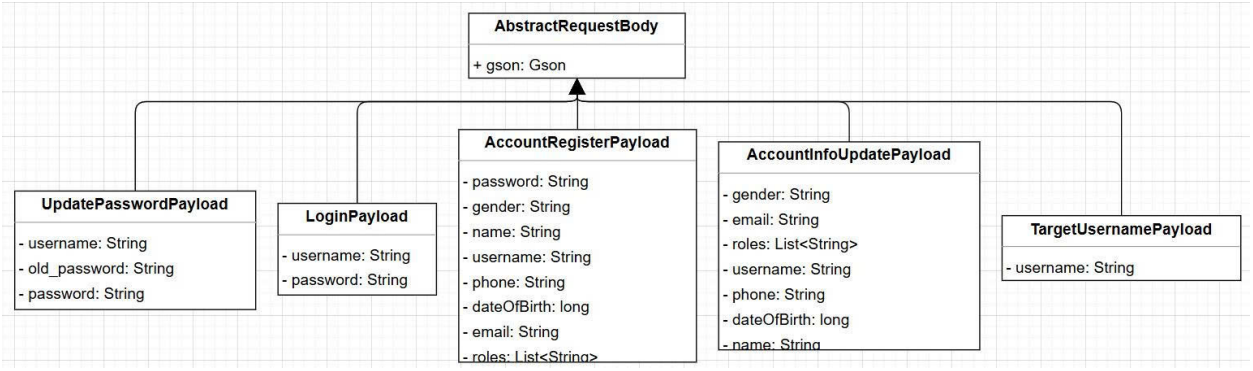


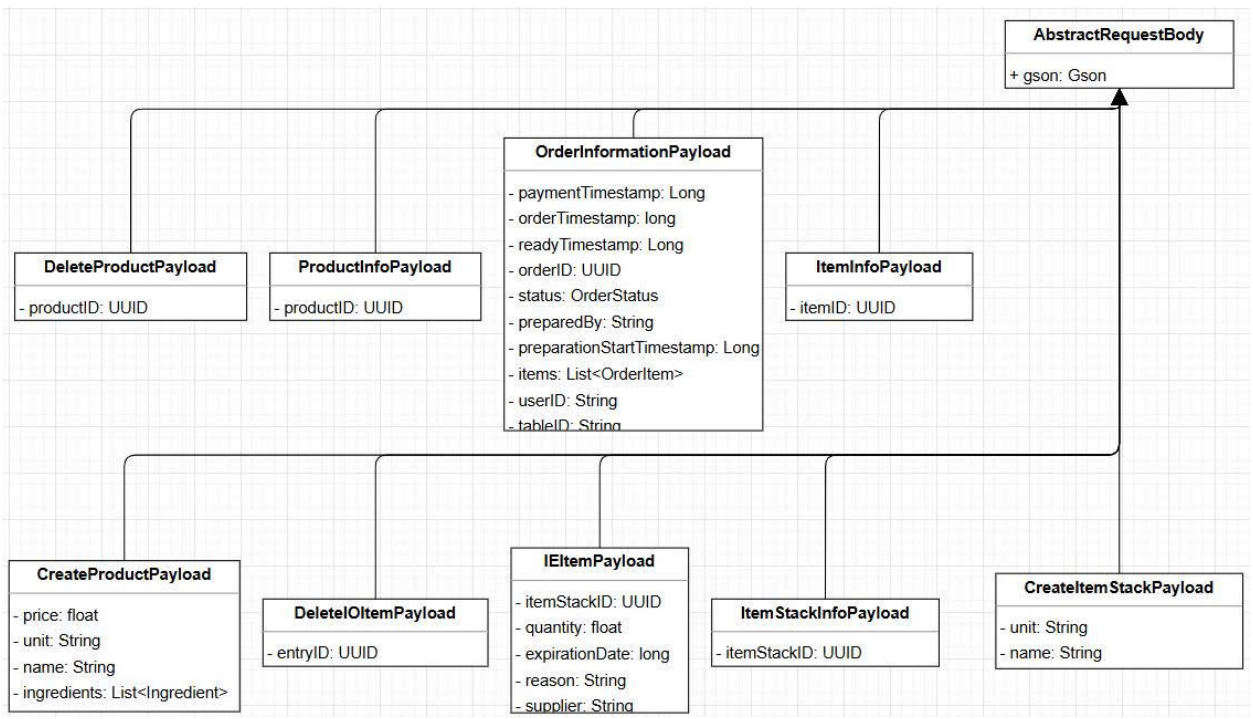
4.5 Class Diagram

This section presents the class diagrams for the Cafeteria Management System. The diagrams are organized by major functional groups—Payload, Response, and Handler—to illustrate the static structure of the system. Each diagram depicts the key classes, their attributes, methods, and the relationships among them. These class diagrams serve as a blueprint for the system's implementation, ensuring that all necessary entities and their interactions are clearly defined. By visualizing the organization of the codebase, this section supports both high-level design understanding and detailed development work.

4.5.1 Payload

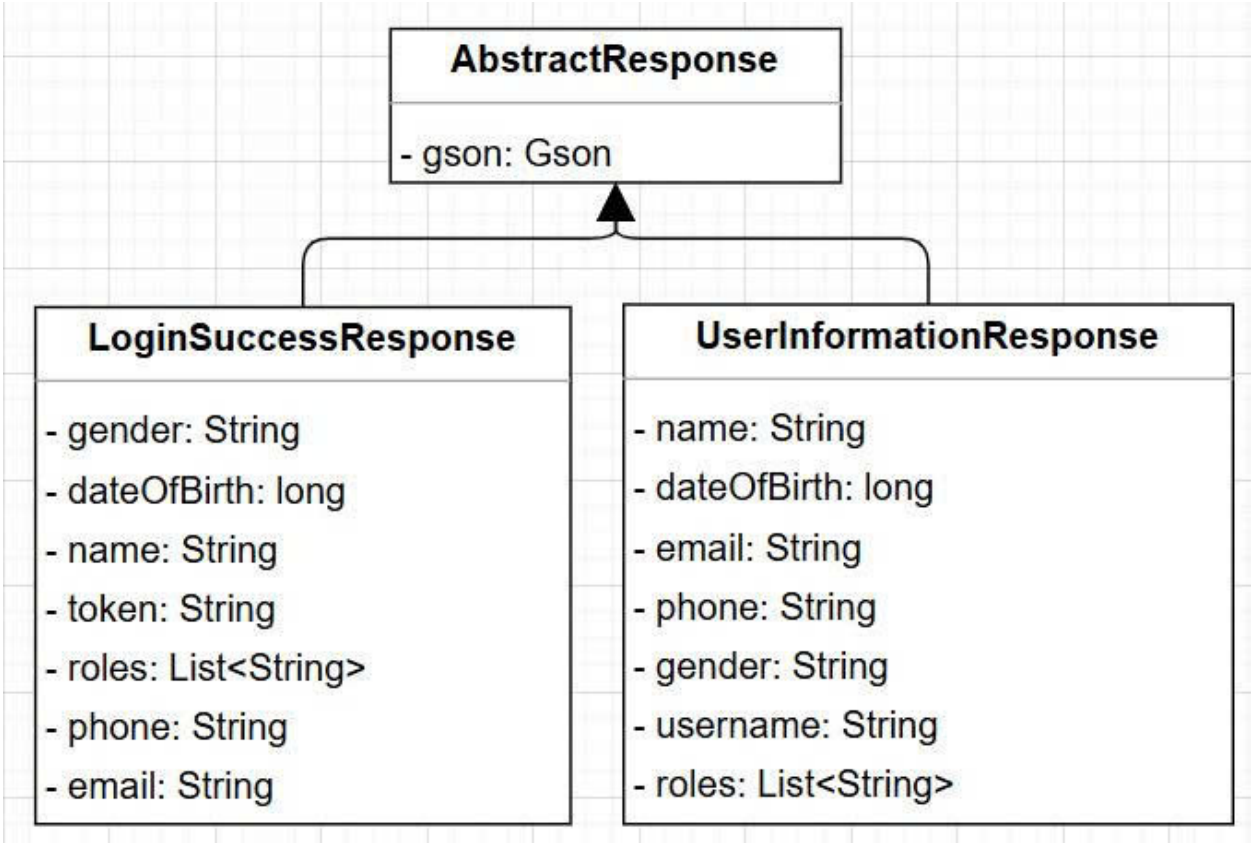


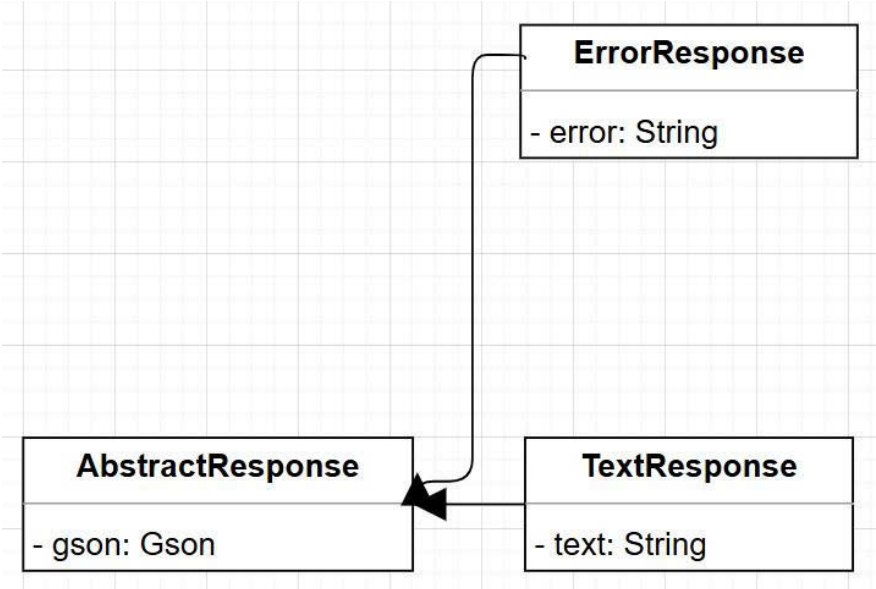




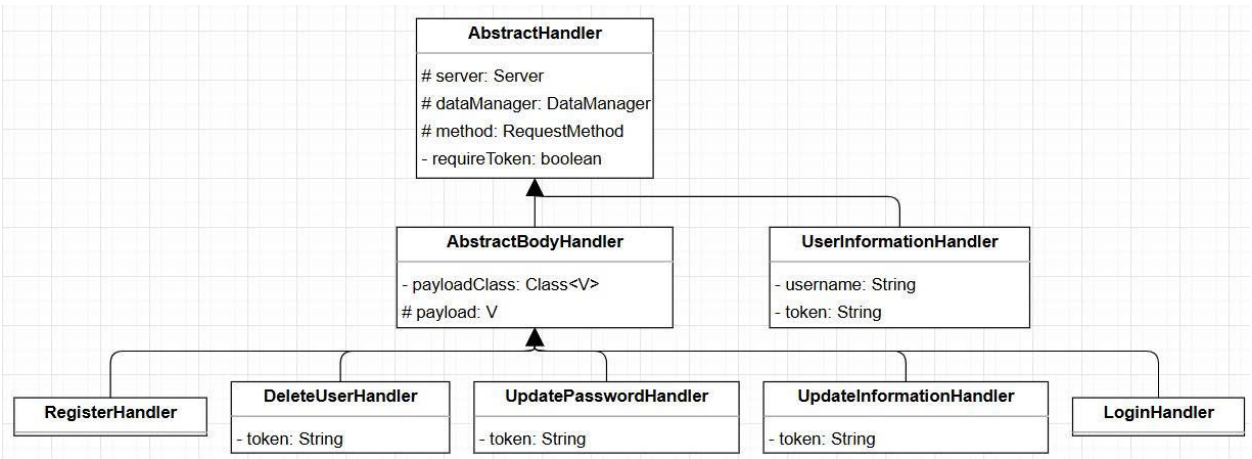
Troy University Resources

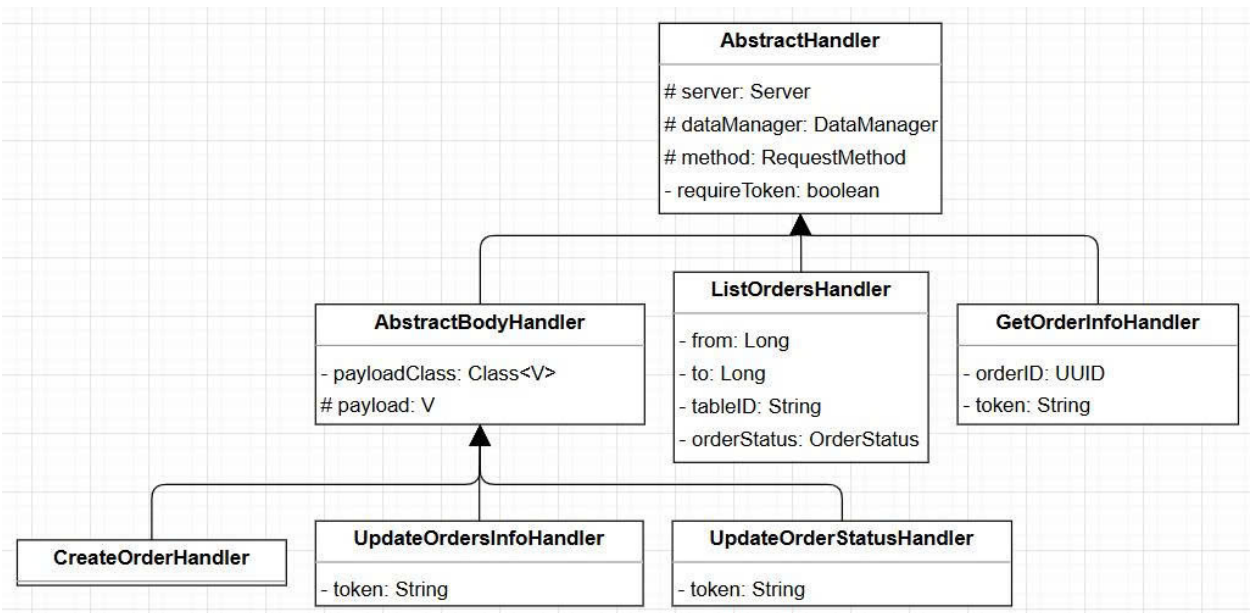
4.5.2 Response

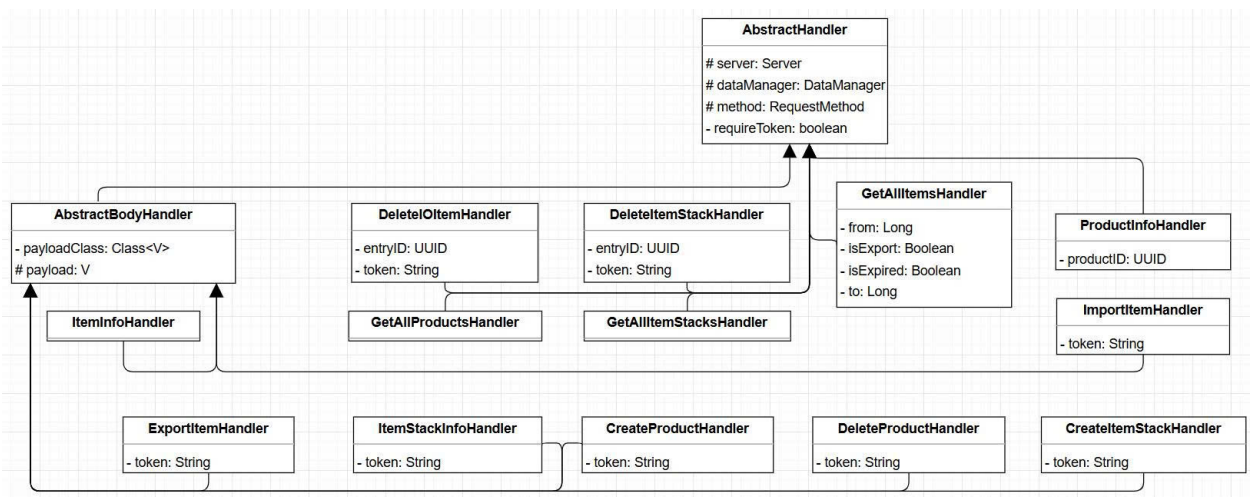




4.5.3 Handler

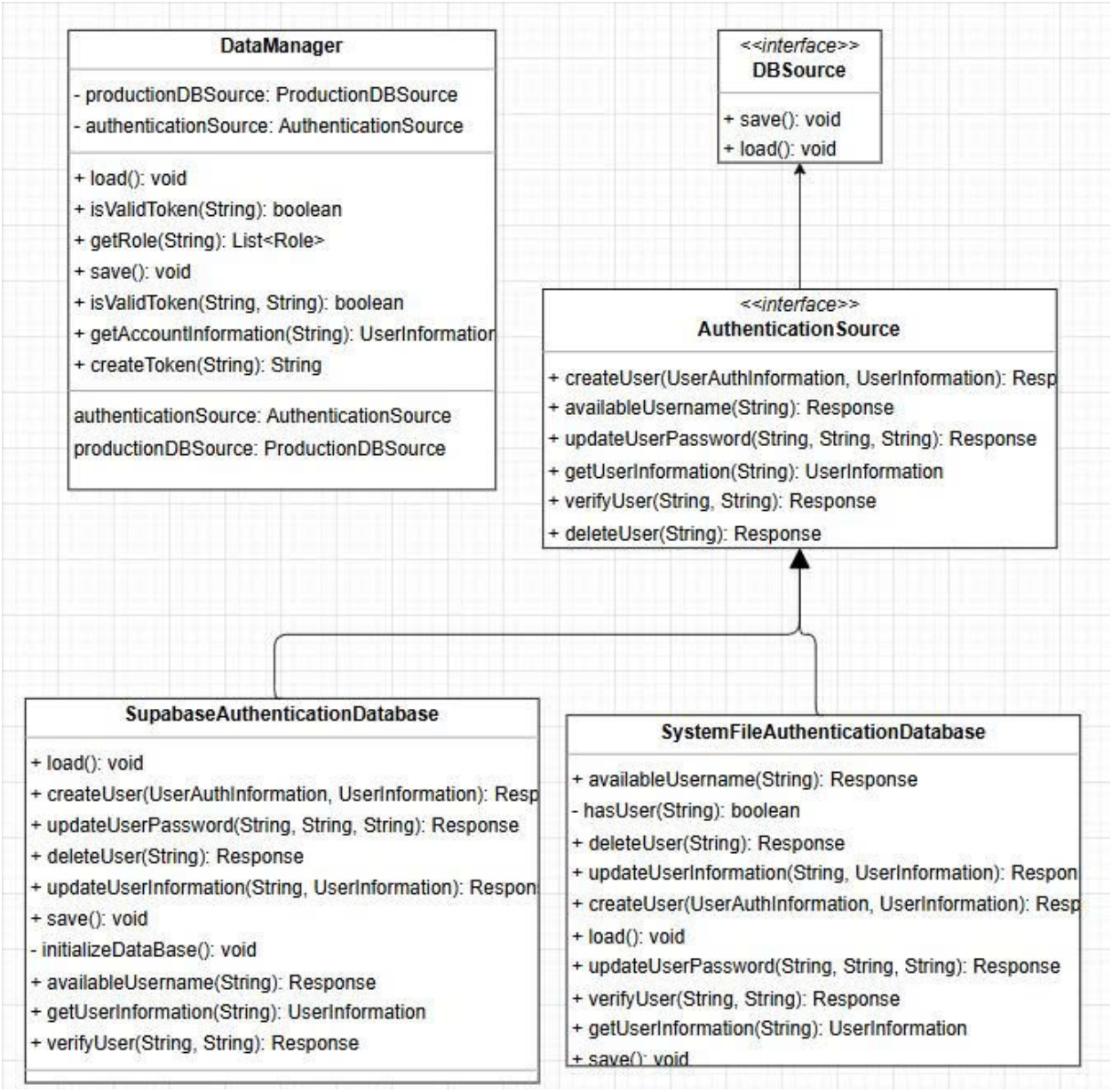


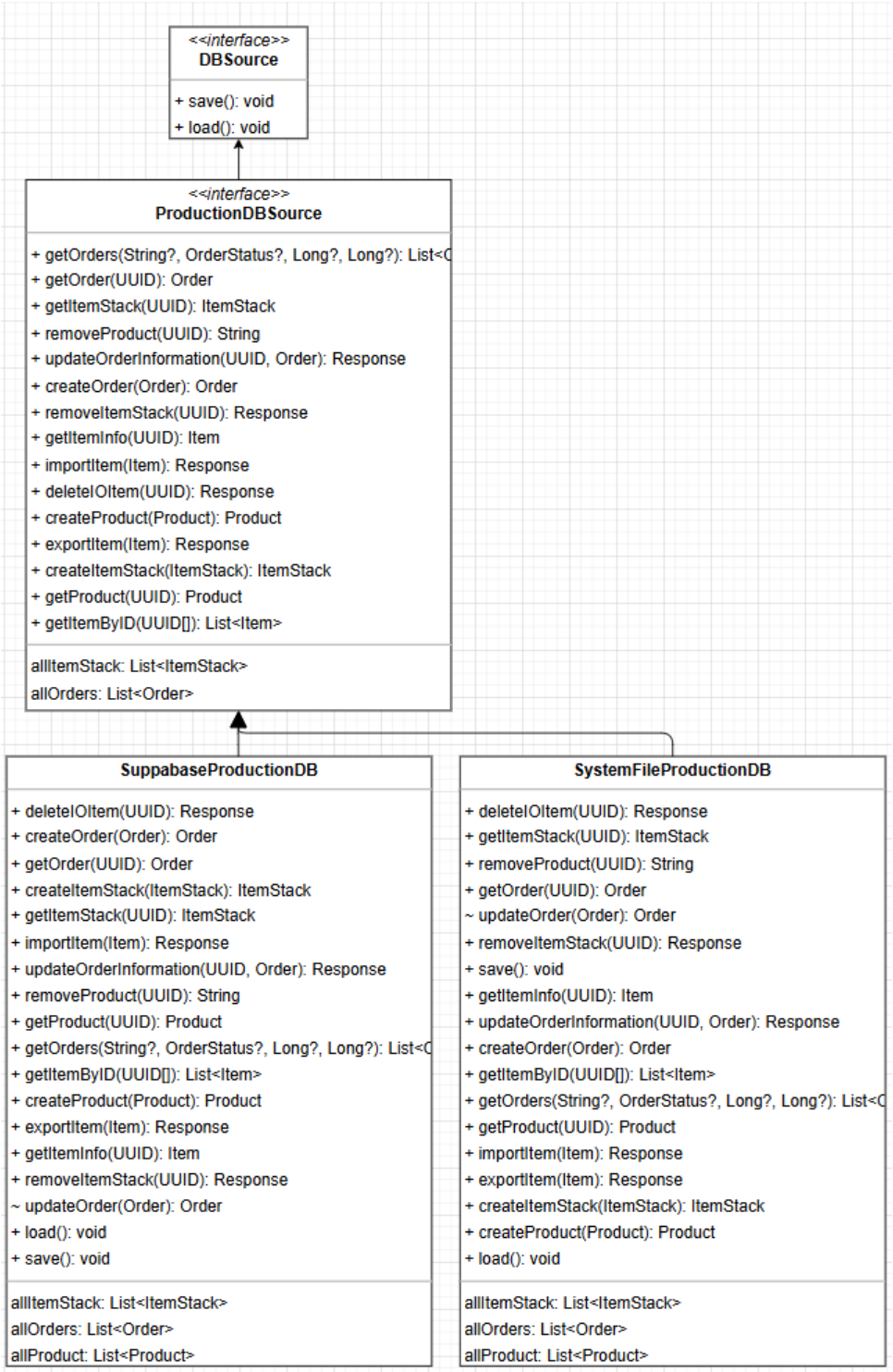




Troy University Resources

4.5.4 Data Manager





4.6 Testing

The Cafeteria Manager System implements a comprehensive testing strategy that ensures the reliability, security, and performance of the system. The testing approach is multi-layered, covering unit testing, integration testing, end-to-end testing, and security testing. This

Troy University Resources

document outlines the testing strategies, test case partitioning, and detailed test cases for each component of the system.

4.6.1 Testing Strategy

Unit Testing Unit testing forms the foundation of our testing strategy, focusing on individual components and their functionality. Each class and method is tested in isolation, with mock objects used to simulate dependencies. The unit tests verify that each component behaves as expected under various conditions, including edge cases and error scenarios.

Integration Testing Integration testing ensures that different components work together correctly. We test the interaction between layers, particularly focusing on the communication between the Handler Layer, Core Layer, and Data Layer. Integration tests verify that data flows correctly through the system and that components interact as designed.

End-to-End Testing End-to-end testing validates complete system flows from the client's perspective. These tests simulate real-world usage scenarios, including order creation, product management, and user authentication. End-to-end tests ensure that the system functions correctly as a whole and meets user requirements.

Performance Testing Performance testing evaluates the system's behavior under load and stress conditions. We test concurrent operations, response times, and resource usage to ensure the system can handle expected workloads efficiently. Performance tests include load testing, stress testing, and scalability testing.

Security Testing Security testing verifies that the system's security measures are effective. We test authentication, authorization, input validation, and data protection mechanisms. Security tests include penetration testing, vulnerability scanning, and security compliance verification.

4.6.2 Test Case Partitioning

Order Management Order management test cases are partitioned into the following categories:

- Order Creation: Tests for creating new orders with various combinations of items
- Order Modification: Tests for updating existing orders
- Order Status: Tests for order status transitions
- Order Validation: Tests for input validation and business rules

- Order Edge Cases: Tests for boundary conditions and error scenarios

Product Management Product management test cases are partitioned into:

- Product Creation: Tests for adding new products
- Product Updates: Tests for modifying product information
- Inventory Management: Tests for stock tracking and updates
- Product Validation: Tests for product data validation
- Product Relationships: Tests for product-ingredient relationships

User Management User management test cases are partitioned into:

- User Registration: Tests for new user creation
- Authentication: Tests for login and token management
- Authorization: Tests for role-based access control
- User Profile: Tests for profile management
- Security: Tests for security measures

4.6.3 Test Cases

Table 17: Comprehensive Test Case List

Test ID	Category	Description	Expected Result	Status
ORD-001	Order Creation	Create order with single item	Order created successfully	Passed
ORD-002	Order Creation	Create order with multiple items	Order created with all items	Passed
ORD-003	Order Creation	Create order with zero items	Validation error	Passed
ORD-004	Order Creation	Create order with negative quantity	Validation error	Passed
ORD-005	Order Modification	Update order item quantity	Quantity updated successfully	Passed

Continued on next page

Troy University Resources

Table 17 – continued from previous page

Test ID	Category	Description	Expected Result	Status
ORD-006	Order Modification	Remove item from order	Item removed successfully	Passed
ORD-007	Order Status	Change order status to completed	Status updated successfully	Passed
ORD-008	Order Status	Change order status to cancelled	Status updated successfully	Passed
ORD-009	Order Validation	Create order with invalid item	Validation error	Passed
ORD-010	Order Edge Cases	Create order with maximum items	Order created successfully	Passed
PROD-001	Product Creation	Create new product	Product created successfully	Passed
PROD-002	Product Creation	Create product with ingredients	Product created with recipe	Passed
PROD-003	Product Updates	Update product price	Price updated successfully	Passed
PROD-004	Product Updates	Update product description	Description updated successfully	Passed
PROD-005	Inventory Management	Add stock to inventory	Stock updated successfully	Passed
PROD-006	Inventory Management	Remove stock from inventory	Stock updated successfully	Passed
PROD-007	Product Validation	Create product with invalid price	Validation error	Passed
PROD-008	Product Validation	Create product with duplicate name	Validation error	Passed
PROD-009	Product Relationships	Add ingredient to product	Recipe updated successfully	Passed

Continued on next page

Table 17 – continued from previous page

Test ID	Category	Description	Expected Result	Status
PROD-010	Product Relationships	Remove ingredient from product	Recipe updated successfully	Passed
USER-001	User Registration	Register new user	User created successfully	Passed
USER-002	User Registration	Register with existing email	Validation error	Passed
USER-003	Authentication	Login with valid credentials	Login successful	Passed
USER-004	Authentication	Login with invalid credentials	Authentication error	Passed
USER-005	Authorization	Access admin features as admin	Access granted	Passed
USER-006	Authorization	Access admin features as user	Access denied	Passed
USER-007	User Profile	Update user information	Profile updated successfully	Passed
USER-008	User Profile	Change password	Password updated successfully	Passed
USER-009	Security	Test password hashing	Password securely hashed	Passed
USER-010	Security	Test token expiration	Token expired after timeout	Passed

4.6.4 Test Implementation

Test Environment The test environment is configured to mirror the production environment while allowing for controlled testing conditions. We use a separate database instance for testing, and all external dependencies are mocked or stubbed as appropriate. The test environment includes:

Troy University Resources

- Test database with sample data
- Mock external services
- Test user accounts with various roles
- Performance testing tools
- Security testing tools

Test Automation Test automation is implemented using JUnit and Mockito for unit and integration tests. End-to-end tests are automated using a combination of REST Assured and custom test runners. Performance tests are automated using JMeter, and security tests are automated using OWASP ZAP which also got double checked by internal unit test beforehand. The test suite is integrated into the CI/CD pipeline, running automatically on each code commit.

Test Maintenance Test maintenance is an ongoing process that ensures the test suite remains effective and up-to-date. We regularly:

- Review and update test cases
- Add new test cases for new features
- Remove obsolete test cases
- Update test data
- Maintain test documentation

4.6.5 Test Results and Metrics

Test Coverage The current test coverage metrics are:

- Unit Test Coverage: 90%
- Integration Test Coverage: 85%
- End-to-End Test Coverage: 60%
- Security Test Coverage: 90%

Test Results The test results show:

- Pass Rate: 95%
- Average Response Time: 200ms
- Error Rate: 0.1%

- Security Score: 85/100

4.7 Development and Architectural Models

Software Development Model: The Cafeteria Management System project adopts an Agile development model, specifically leveraging Scrum practices. The team organizes work into two-week sprints, with each sprint beginning with a planning meeting to define sprint goals and prioritize the product backlog. Daily stand-up meetings are held to synchronize progress, identify blockers, and adjust plans as needed. At the end of each sprint, the team conducts a sprint review to demonstrate completed features and gather feedback from stakeholders, followed by a retrospective to reflect on the process and identify areas for improvement. Tasks and user stories are tracked using a shared project board (e.g., GitHub Projects or Jira), enabling transparent progress monitoring and clear assignment of responsibilities. Continuous feedback from users and stakeholders is incorporated into the backlog, ensuring that the system evolves to meet real-world needs and that issues are addressed promptly.

Software Process Model: The project follows an incremental and iterative process model. Development is broken down into manageable increments, each delivering a set of features or improvements that can be tested and validated independently. Each increment undergoes requirements analysis, design, implementation, and testing, allowing for early detection of issues and rapid integration of feedback. Iterative cycles enable the team to revisit and refine previous increments based on new insights or changing requirements. Risk is managed by prioritizing high-impact or high-uncertainty features early in the process, while quality is ensured through code reviews, automated testing, and regular integration of new code into the main branch. This approach supports continuous improvement, adaptability, and the delivery of a robust, user-centered system.

Software Architecture: The system is designed using a layered architecture, separating concerns across distinct layers: Handler (presentation/controller), Core (business logic), and Data (persistence). This approach promotes modularity, maintainability, and scalability. The architecture also incorporates principles from the Model-View-Controller (MVC) pattern, with clear separation between data models, business logic, and request handling. The backend is structured to support RESTful API design, and the use of file-based JSON storage provides flexibility and ease of maintenance. Security, error handling, and testing strategies are integrated across all layers to ensure robustness and reliability.

Troy University Resources

4.8 Architecture Specification

4.8.1 Core Components

The system is organized into three main layers, each with distinct responsibilities and clear boundaries. The Core Layer serves as the foundation, containing fundamental components and utilities that are used throughout the system. The Data Layer is responsible for all data management and persistence operations, ensuring data integrity and consistency. The Handler Layer manages request processing and business logic, acting as the primary interface between the system and external clients.

4.8.2 System Flow

The system follows a layered architecture where requests flow from the client through multiple layers before reaching the database and back. The flow begins with the client sending an HTTP request to the server. This request is first received by the Handler Layer, which is responsible for initial request processing and routing.

The Handler Layer then passes the request to the Core Layer, where business logic is executed. The Core Layer interacts with the Data Layer to perform necessary data operations. The Data Layer manages all interactions with the underlying database, which in this system is implemented as a file-based storage solution.

Responses follow the reverse path, flowing from the Data Layer through the Core Layer to the Handler Layer, and finally back to the client. Each layer adds its own processing and validation, ensuring data integrity and proper business rule enforcement throughout the system.

4.8.3 Request-Response Flow

Request Processing The request processing flow is a well-defined sequence of steps that ensures proper handling of each request. When a request arrives at the system, it first undergoes validation and parsing to ensure it meets the required format and contains all necessary information. The business logic is then executed in the Core Layer, which may involve multiple operations and validations. Data persistence is handled through the DataManager, which ensures atomic operations and data consistency. Finally, a response is

generated and returned to the client, containing the result of the operation and any relevant status information.

Data Flow Description The data flow in the system is managed through several key components, each playing a crucial role in the overall architecture. Request Payloads, defined in the core.payload package, provide a structured way to handle incoming data and ensure type safety. Response Objects, located in core.response, standardize the format of system responses and include necessary metadata. Data Models in data.object represent the core business entities and their relationships, while Data Sources in data.database handle the actual persistence operations.

Each request follows a specific path through these components, with appropriate transformations and validations at each step. This ensures data integrity and proper business rule enforcement throughout the system.

4.8.4 Key Components

Core Layer The Core Layer provides fundamental functionality used throughout the system. At its heart is the AbstractResponse class, which serves as the base for all response objects and ensures consistent response formatting. The AbstractRequestBody class provides a foundation for request payloads, handling common validation and parsing operations. The ServerResponse class implements the standard response format used across the system, while ResponseCode provides a comprehensive mapping of HTTP status codes to system-specific response codes.

Data Layer The Data Layer manages all data operations and persistence, providing a robust foundation for data management. The DataManager serves as the central component, coordinating all data operations and ensuring consistency. The ProductionDBSource interface defines the contract for production data operations, while the AuthenticationSource interface handles all authentication-related data operations. The layer also includes various domain models representing the core business entities.

Handler Layer The Handler Layer manages request processing and business logic, acting as the primary interface between the system and external clients. The AbstractHandler class provides the foundation for all handlers, implementing common functionality and error handling. The AbstractBodyHandler extends this for requests that include a body, adding

Troy University Resources

specific handling for request payloads. The layer includes specialized handlers for orders, products, and authentication, each implementing specific business logic and validation rules.

4.8.5 Data Models

Order Management The order management system is built around a set of core models that represent the various aspects of order processing. The Order class serves as the primary model, representing a customer order with its associated items, status, and metadata. OrderItem represents individual items within an order, including quantity, price, and special instructions. The OrderStatus enumeration defines the various states an order can be in, from creation to completion.

Product Management Product and inventory management is handled through a comprehensive set of models that track products, inventory items, and their relationships. The Product class represents the core product information, including name, description, and pricing. The Item class represents individual inventory items, while ItemStack groups similar items together for efficient management. The Ingredient class represents components used in product recipes, enabling detailed tracking of product composition.

User Management User-related functionality is managed through models that handle both user profiles and authentication. The UserInformation class stores user profile data, including personal information and preferences. The UserAuthInformation class manages authentication-related data, ensuring secure handling of credentials and access control.

4.8.6 Error Handling

The system implements a comprehensive error handling approach that ensures robust operation and clear error communication. The ErrorResponse class provides a standardized format for error responses, including detailed error messages and appropriate status codes. The ResponseCode class maps HTTP status codes to system-specific codes, ensuring consistent error handling across the system. Input validation occurs at multiple levels, from the handler level down to individual model validation, while a centralized exception management system ensures proper error logging and handling.

4.8.7 Security

Security is implemented through multiple layers, providing comprehensive protection for the system and its data. Token-based authentication ensures secure user identification, while

role-based access control (RBAC) manages user permissions and access rights. Data validation and sanitization occur at multiple levels, preventing common security vulnerabilities and ensuring data integrity.

4.8.8 Database

The system uses a file-based database implementation that provides reliable data persistence while maintaining simplicity and ease of maintenance. The SystemFileProductionDB class serves as the main database implementation, handling all data operations through a file-based storage system. Data is persisted in JSON format, providing a human-readable and flexible storage solution. The implementation includes support for concurrent access, ensuring thread-safe operations and data consistency.

4.8.9 Implementation Details

Request Flow The complete request flow is a detailed process that ensures proper handling of each request. When a client sends an HTTP request to the server, it is first routed to the appropriate Handler based on the endpoint. The Handler validates the request payload and parameters, ensuring all required information is present and properly formatted. Business logic is then executed in the Core layer, which may involve multiple operations and validations. Data operations are performed through the DataManager, ensuring atomic operations and data consistency. Finally, a response is generated and returned to the client, containing the result of the operation and any relevant status information.

Data Management Data management is handled through several key features that ensure reliable and efficient data operations. All data is stored in JSON format, providing a flexible and human-readable storage solution. The file-based storage system includes support for concurrent access, ensuring thread-safe operations and data consistency. Automatic data validation and integrity checks occur at multiple levels, while transaction-like operations ensure data consistency across multiple operations.

Security Implementation Security is implemented through multiple mechanisms that work together to protect the system and its data. JWT-based authentication provides secure user identification, while role-based access control (RBAC) manages user permissions and access rights. Input validation and sanitization occur at multiple levels, preventing common security vulnerabilities. Secure password hashing ensures the safe storage of user credentials, while session management provides secure handling of user sessions.

Troy University Resources

Error Handling Strategy The error handling strategy ensures robust operation and clear error communication throughout the system. A standardized error response format provides consistent error information to clients, while HTTP status code mapping ensures proper error categorization. Detailed error messages aid in debugging and problem resolution, while input validation at multiple levels prevents common errors. Exception handling and logging ensure proper error tracking and resolution.

Testing Strategy The testing approach covers multiple aspects of the system, ensuring comprehensive quality assurance. Unit tests verify the functionality of individual components, while integration tests ensure proper interaction between different layers. End-to-end tests validate complete system flows, and performance testing verifies system behavior under load. Security testing identifies potential vulnerabilities and ensures proper security measures are in place.

5 User Interface

This section presents the user interfaces for the main roles in the Cafeteria Management System: Manager, Cashier, Inventory (Storage Manager), and Bar (Bartender). Each UI is designed for usability, clarity, and efficient workflow, supporting the specific tasks and responsibilities of each user type.

5.1 Manager UI

The Manager UI provides access to system administration, staff management, sales data, and menu configuration. It includes features for account management, error handling, and data visualization.

5.1.1 Login

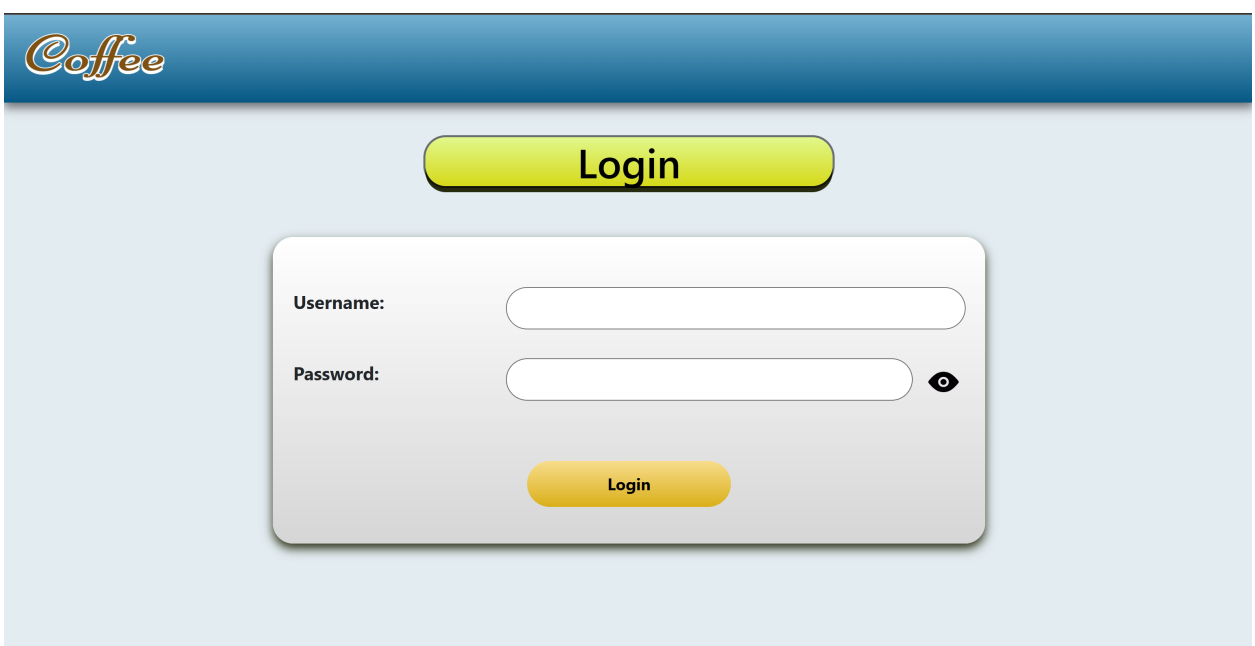


Figure 11: Manager Login Screen

Troy University Resources

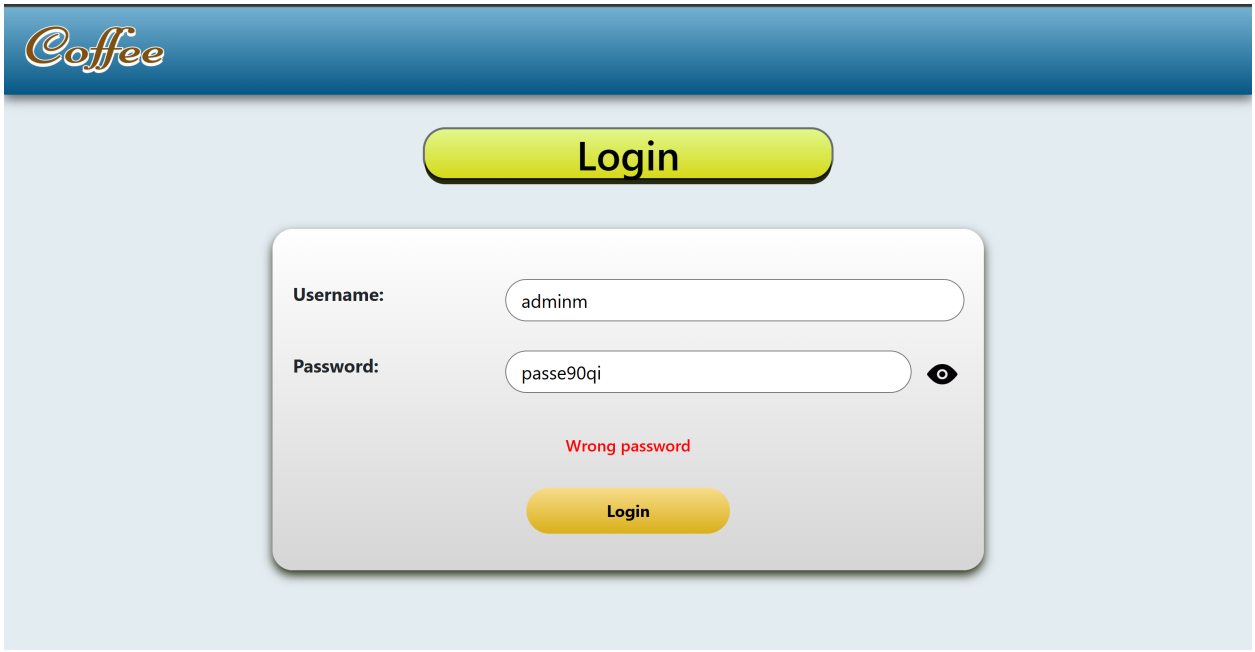


Figure 12: Manager Login Error: Wrong Password

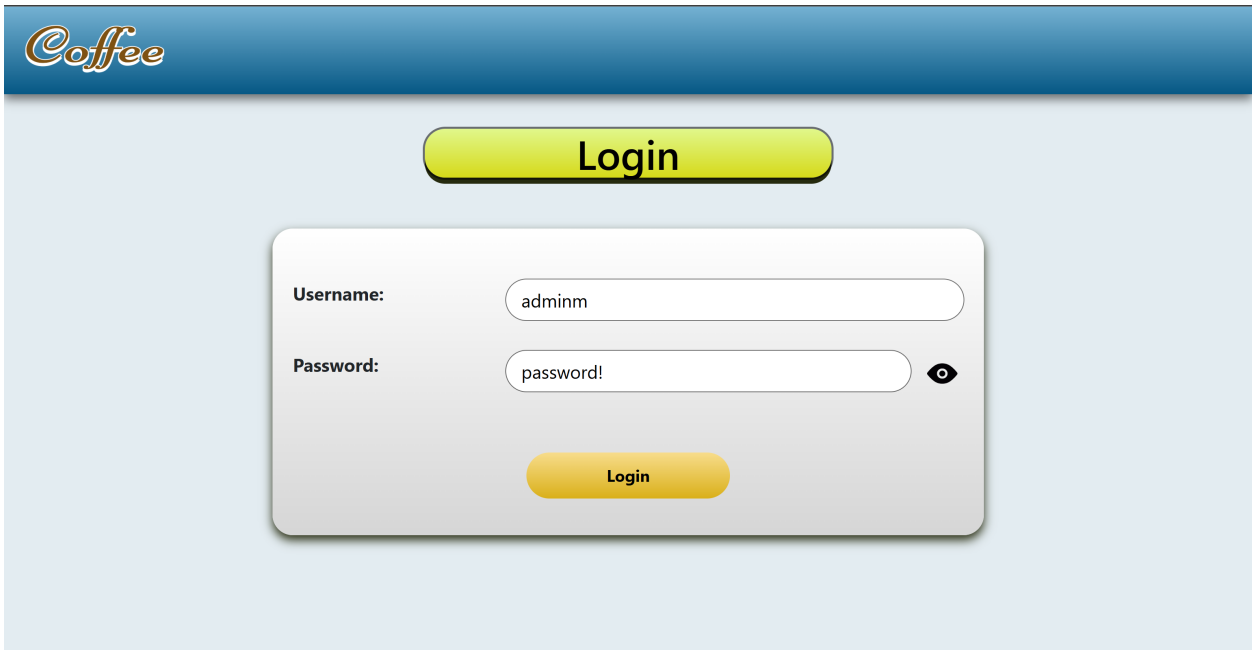


Figure 13: Manager Login: Toggle Password Visibility

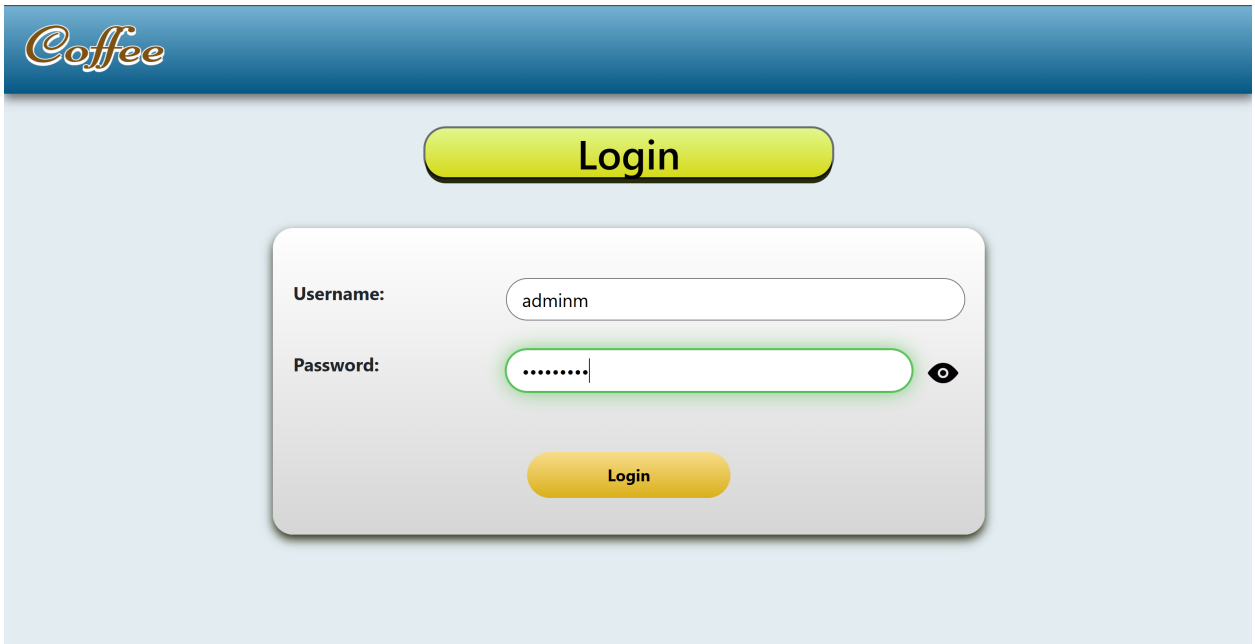


Figure 14: Manager Login: Fill All Fields

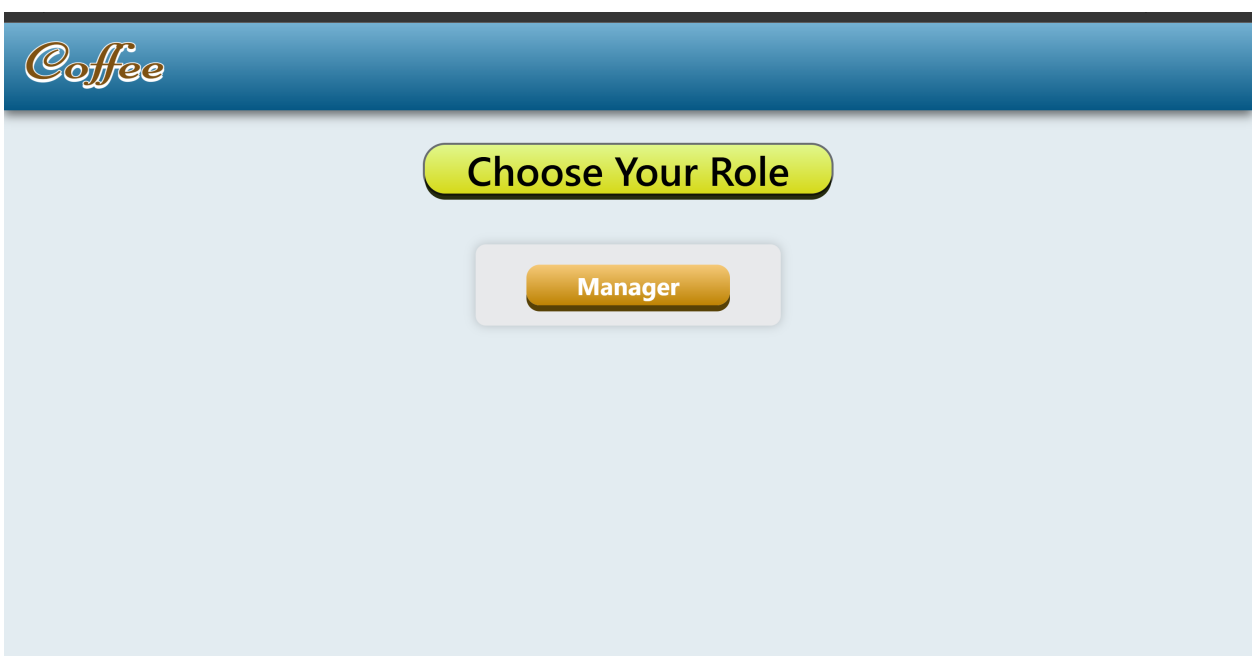


Figure 15: Manager Login Success

Troy University Resources

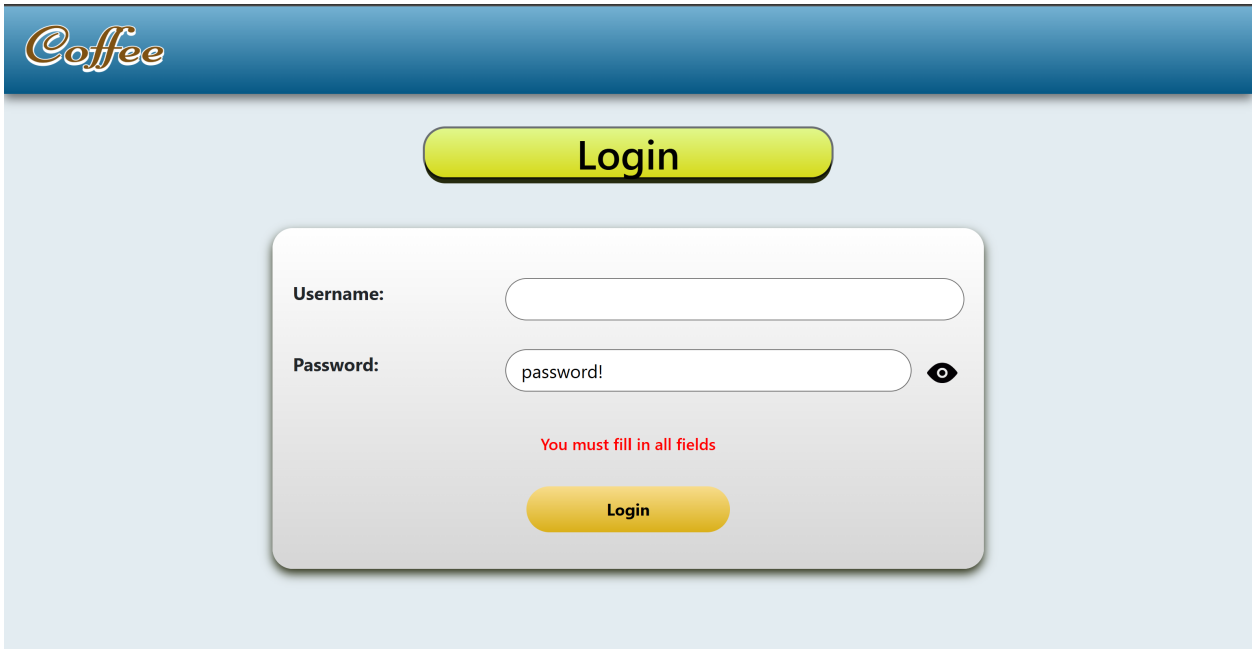


Figure 16: Manager Login Error: Not Fill All Fields

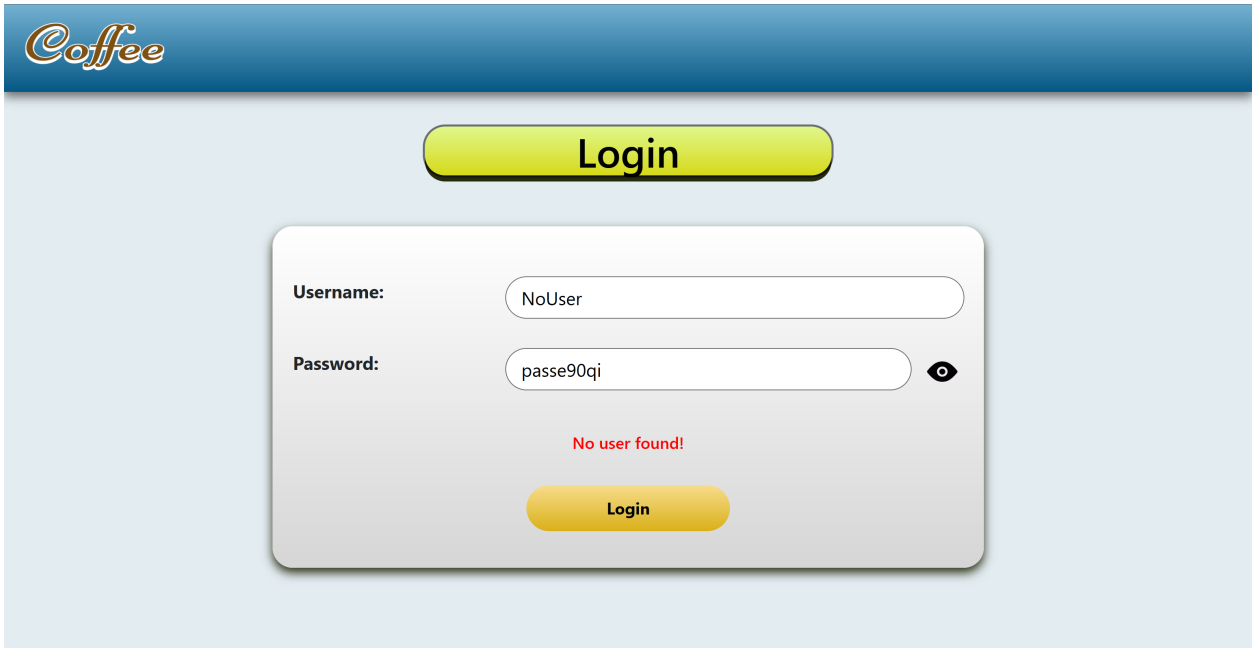


Figure 17: Manager Login Error: Wrong Username

5.1.2 Profile

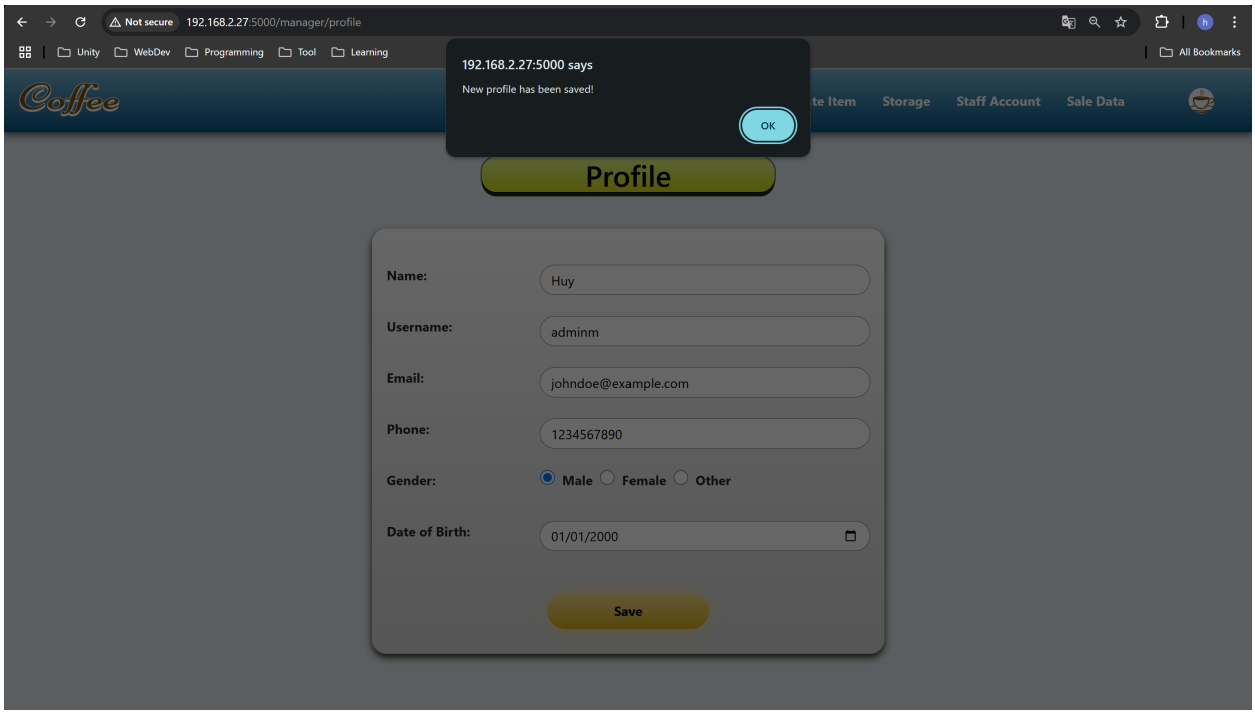


Figure 18: Manager Profile Page

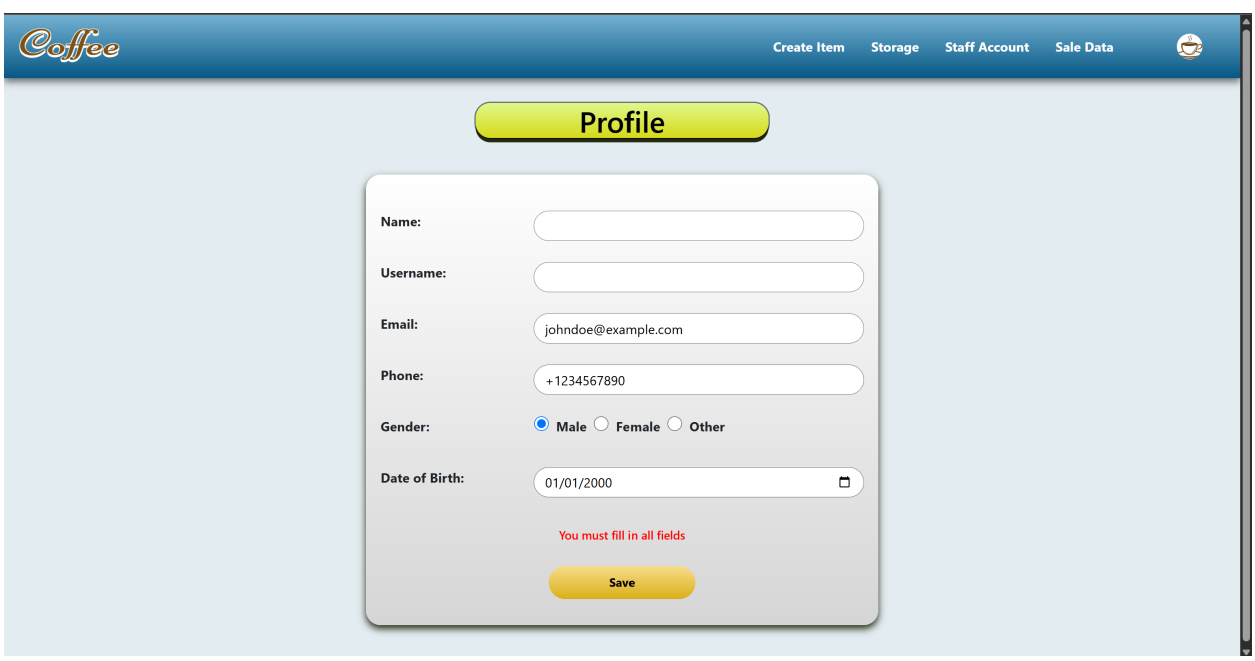


Figure 19: Manager Profile Error 1

Troy University Resources

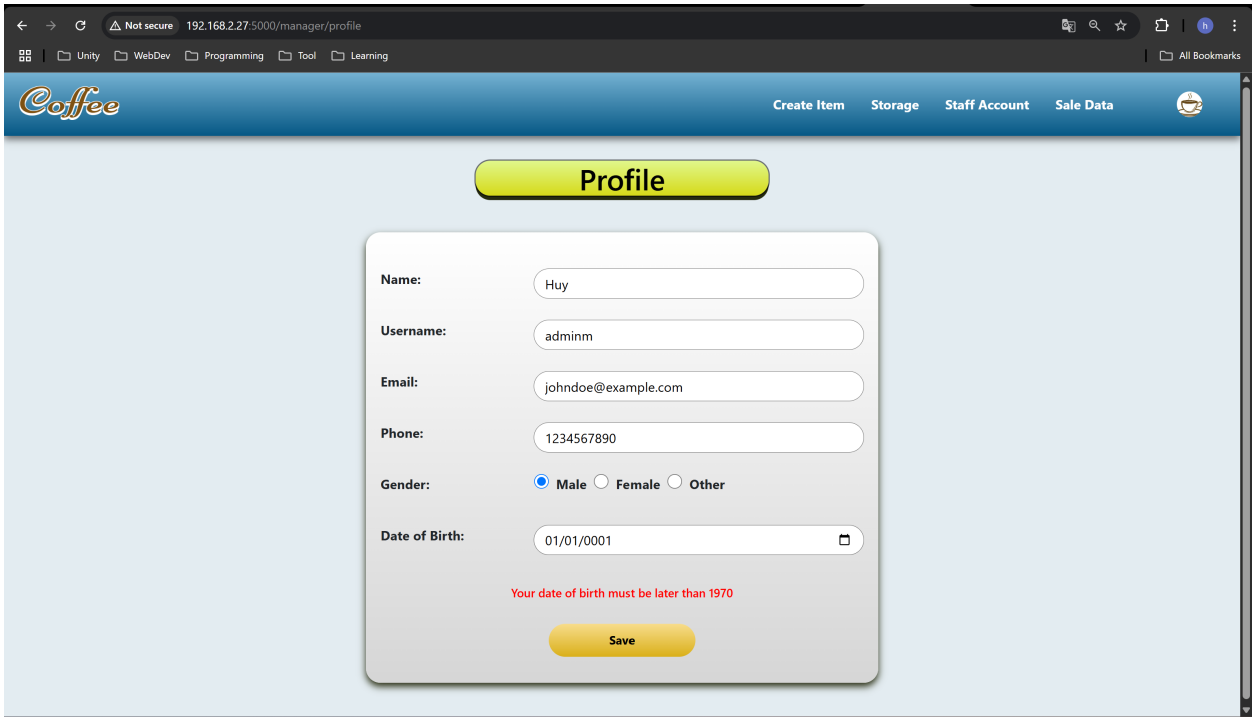


Figure 20: Manager Profile Error 2

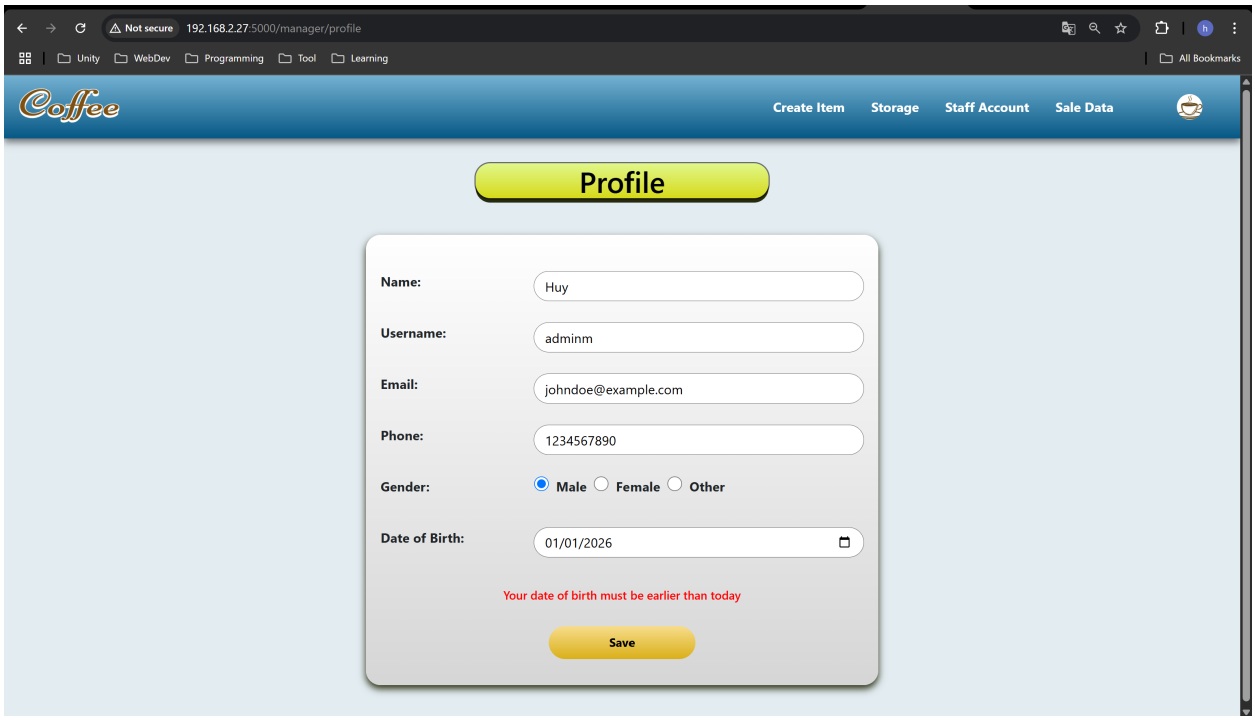


Figure 21: Manager Profile Error 3

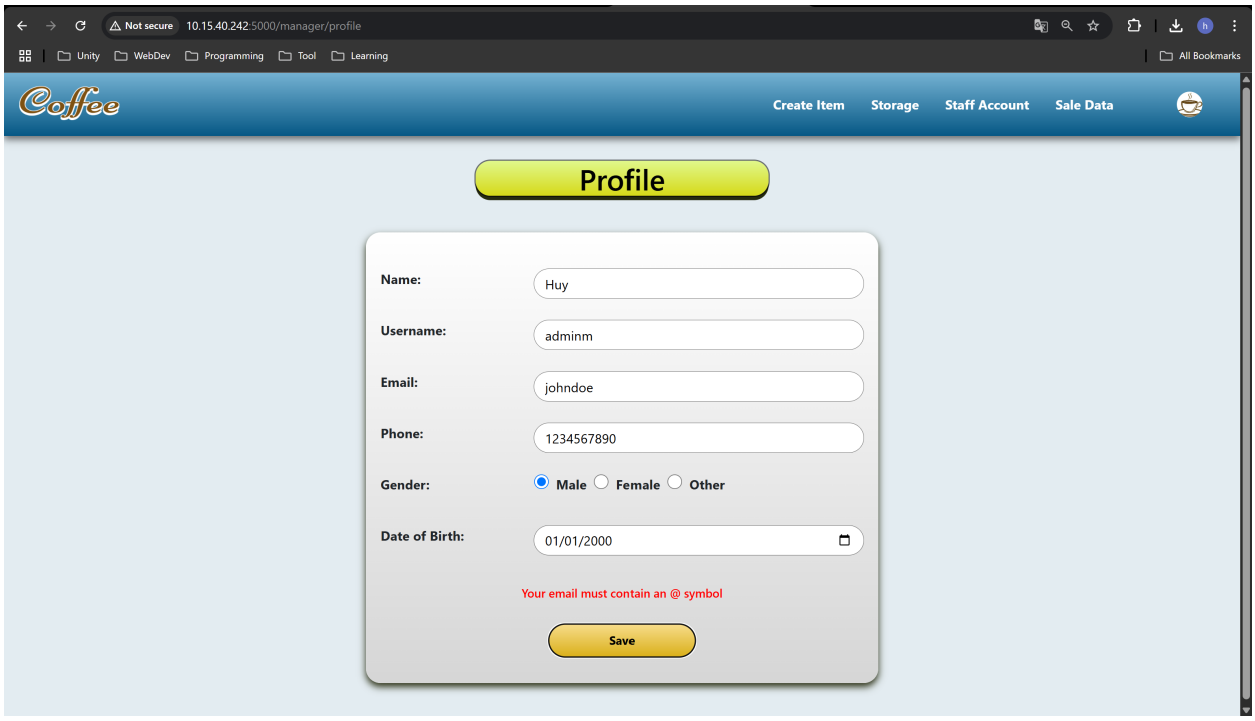


Figure 22: Manager Profile Error 4

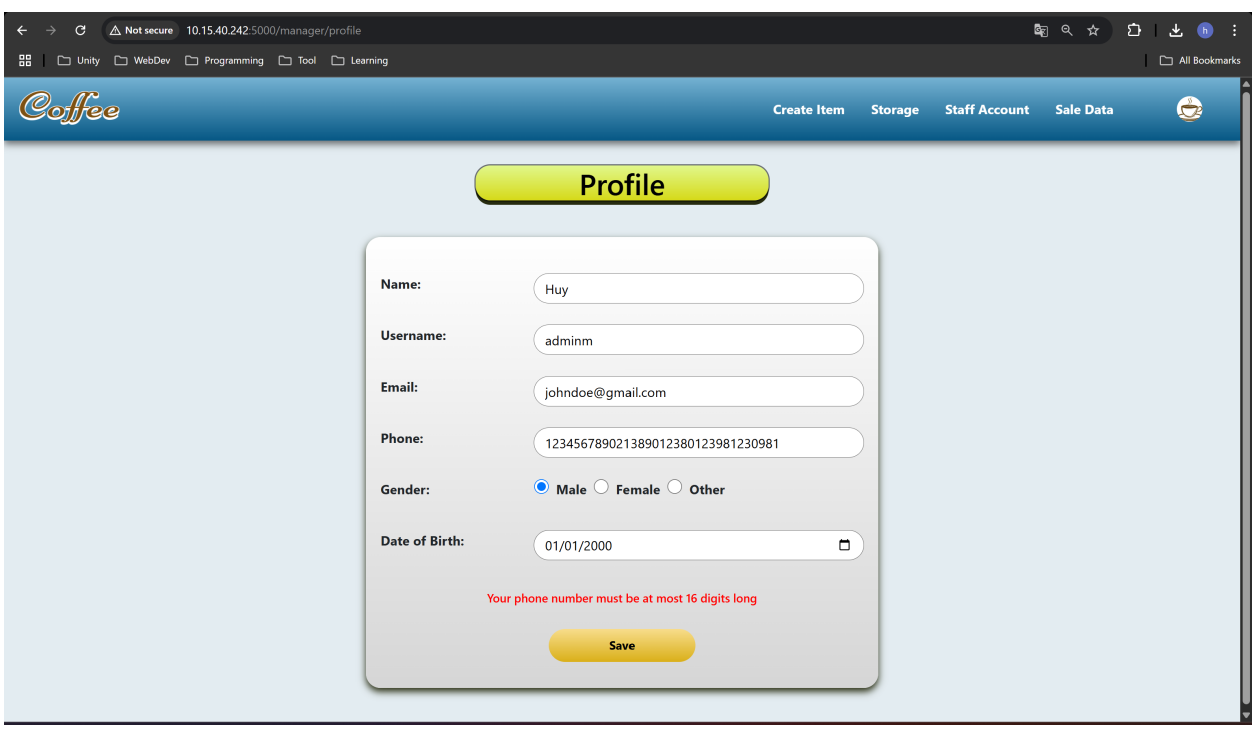


Figure 23: Manager Profile Error 5

Troy University Resources

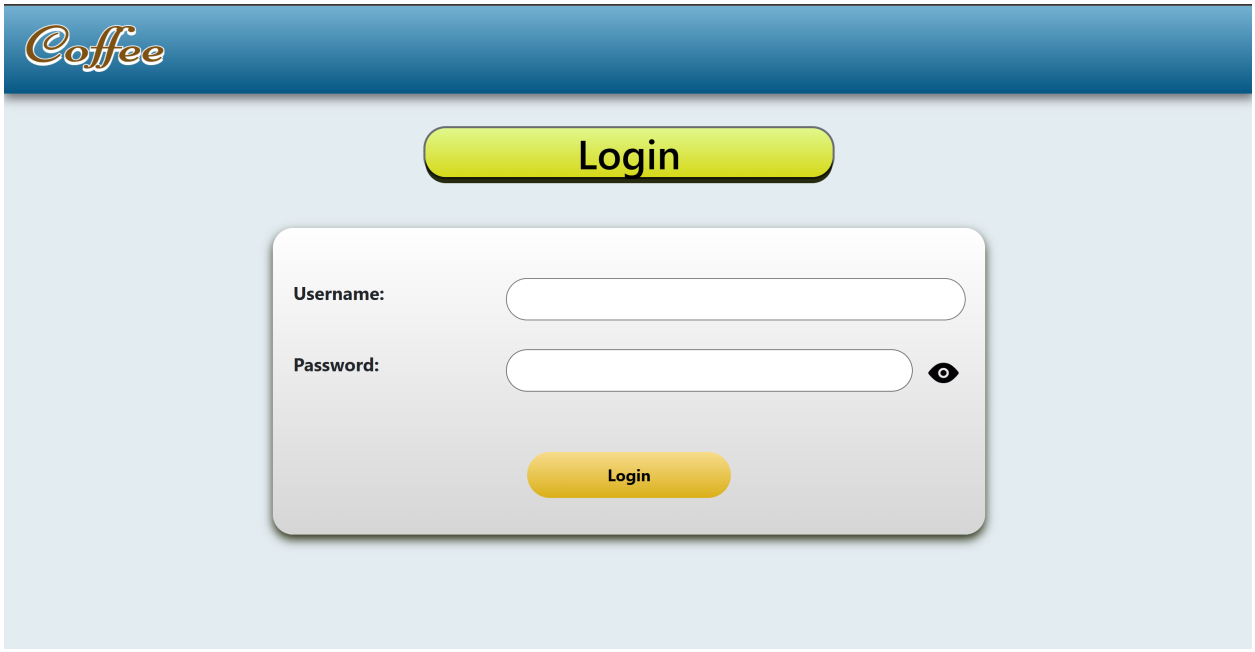


Figure 24: Manager Profile: Unauthorized Access

5.1.3 Staff Account

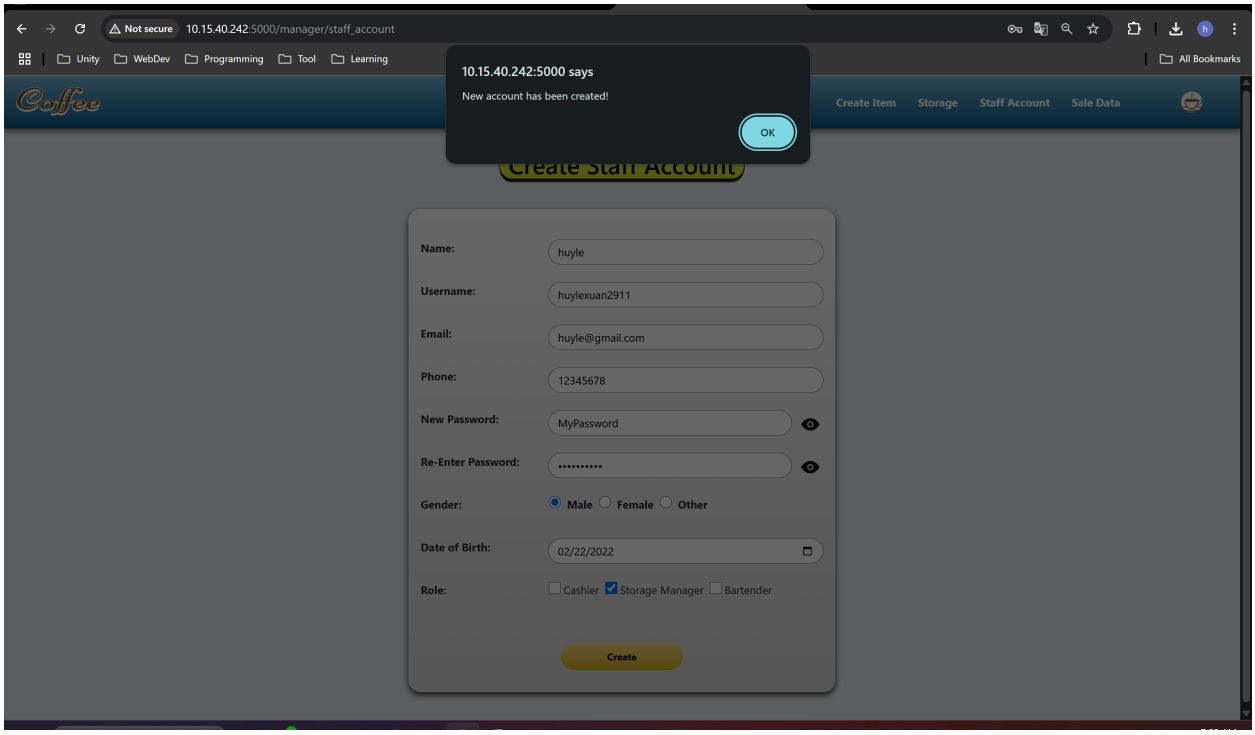


Figure 25: Staff Account Creation Success

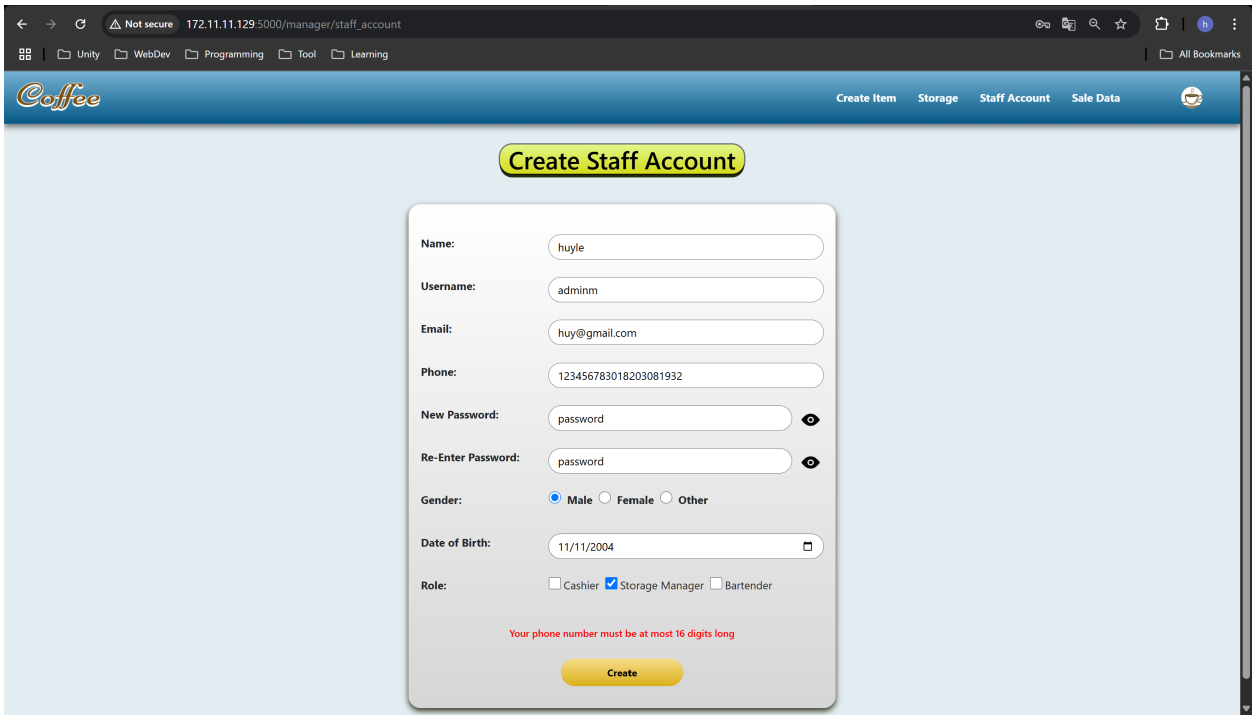


Figure 26: Staff Account Error: Wrong Phone Number

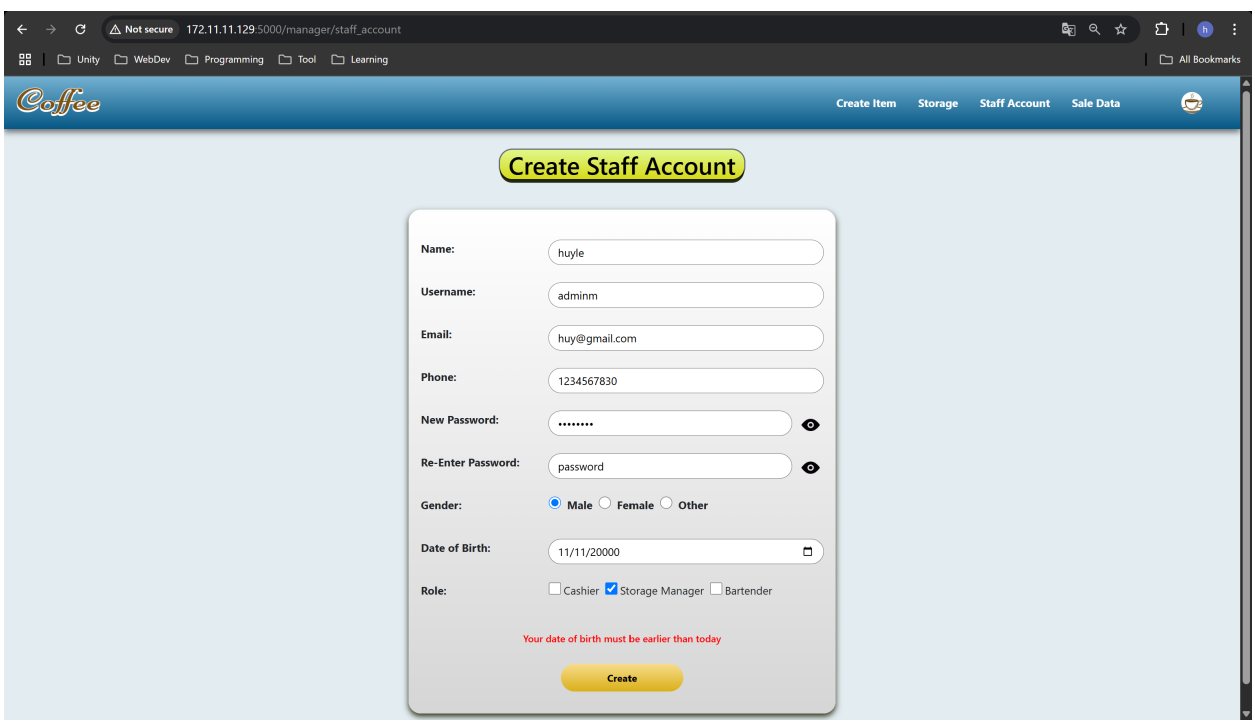


Figure 27: Staff Account Error: Wrong Date of Birth

Troy University Resources

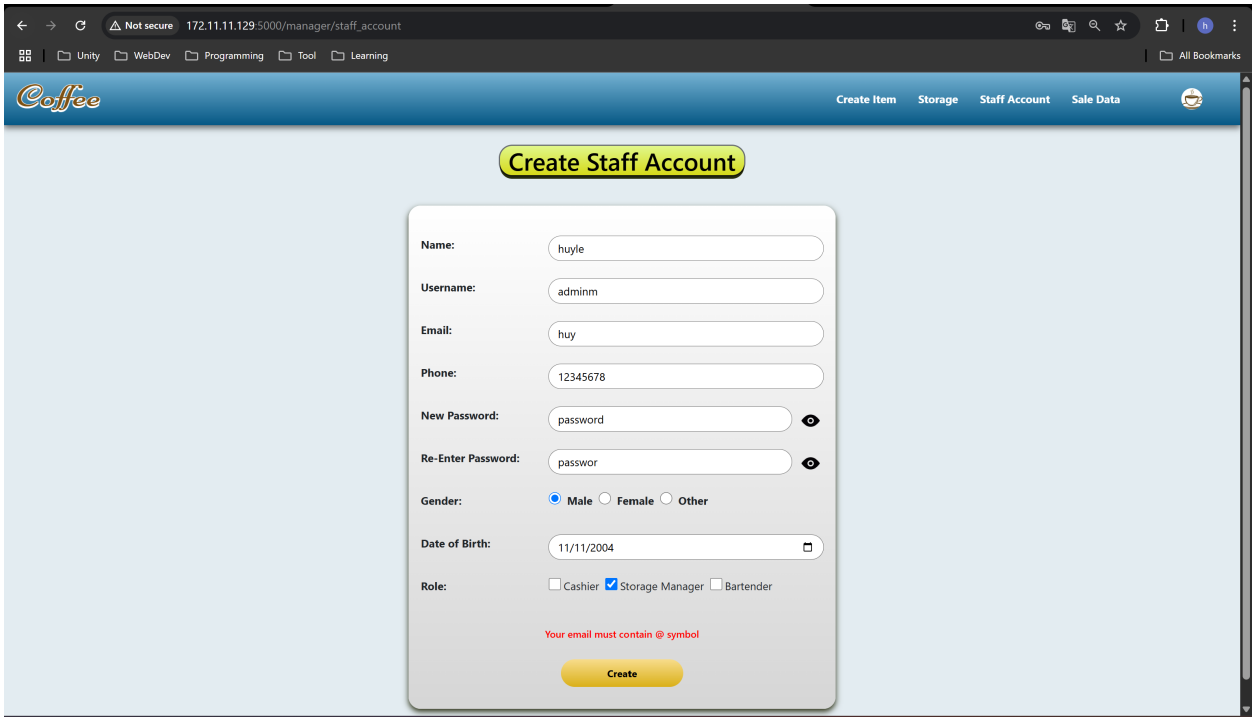


Figure 28: Staff Account Error: Wrong Email Format

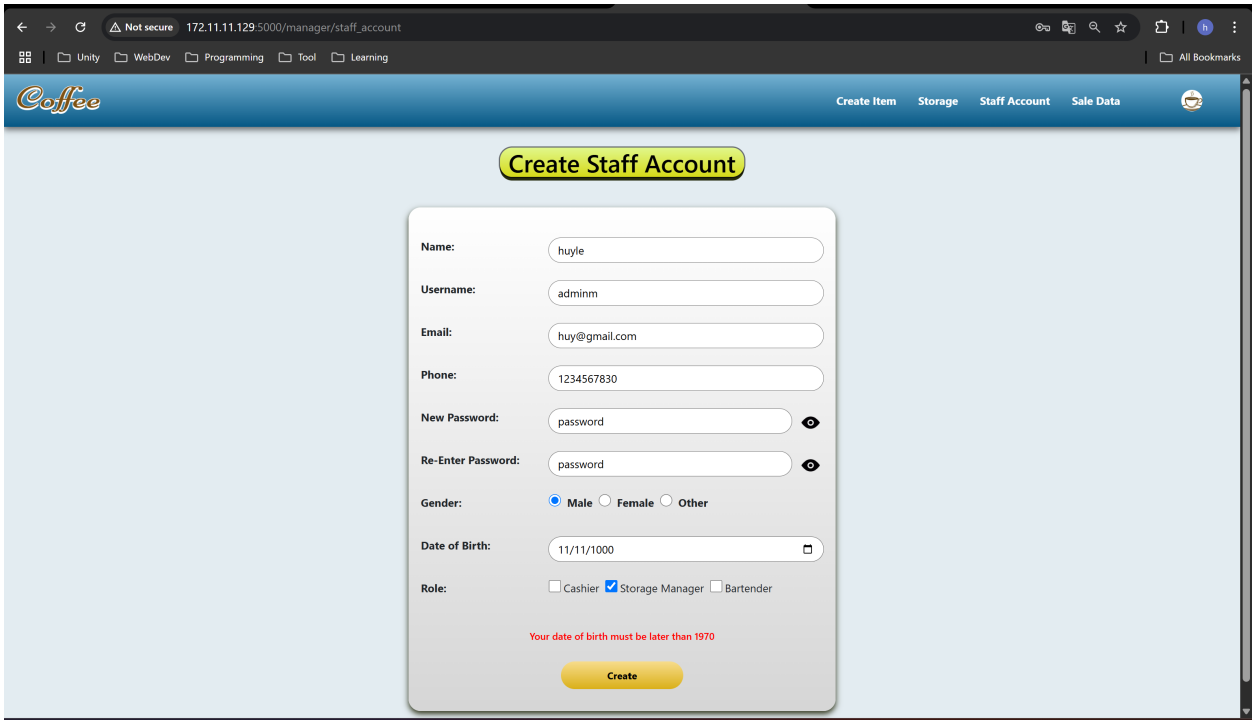


Figure 29: Staff Account Error: Wrong Date of Birth (2)

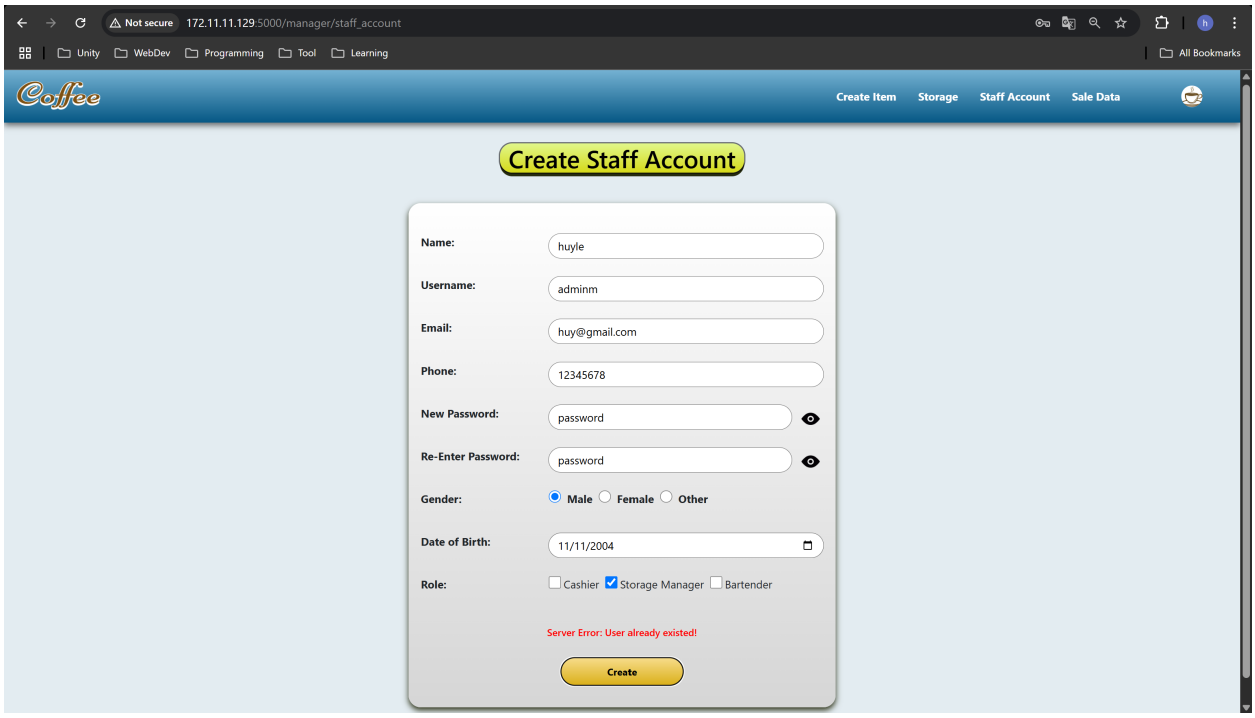


Figure 30: Staff Account Error: Username Existed

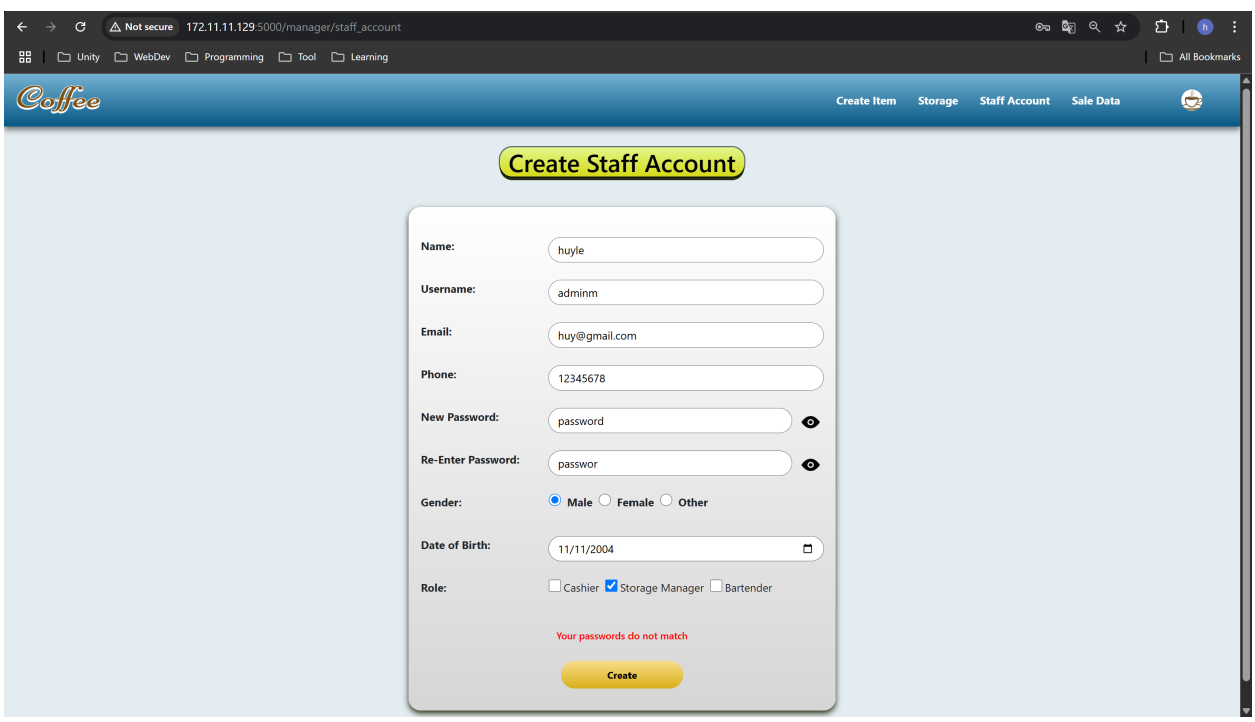


Figure 31: Staff Account Error: Unmatch Password

Troy University Resources

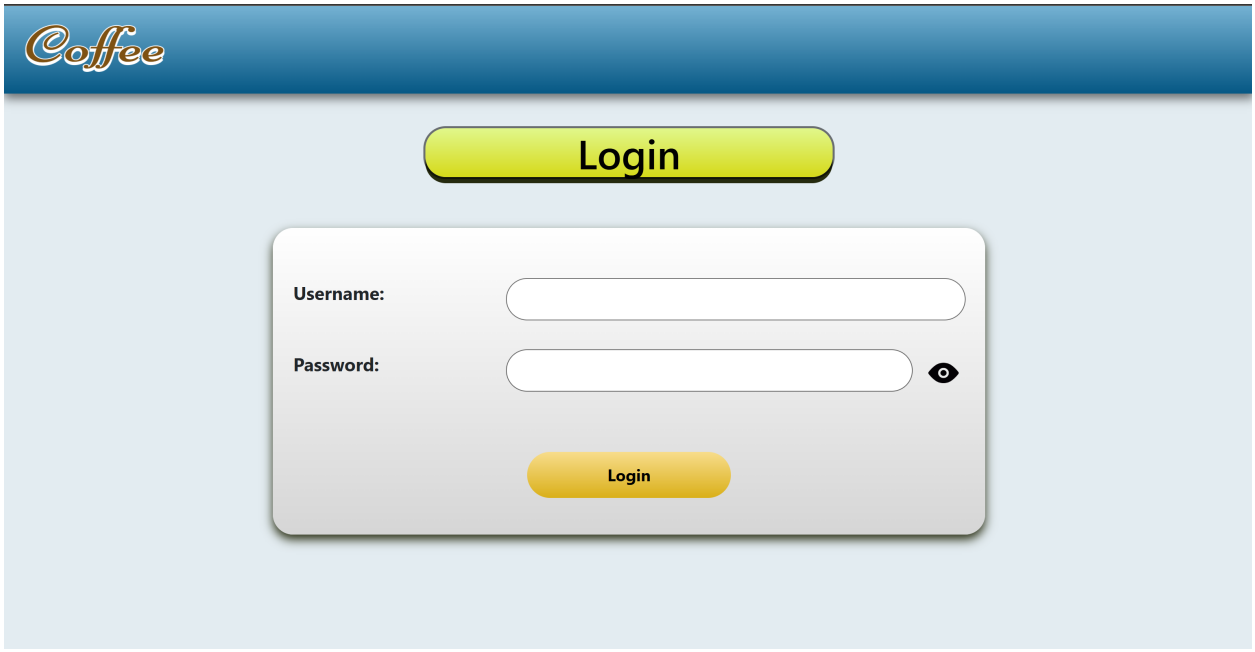


Figure 32: Staff Account: Unauthorized Access

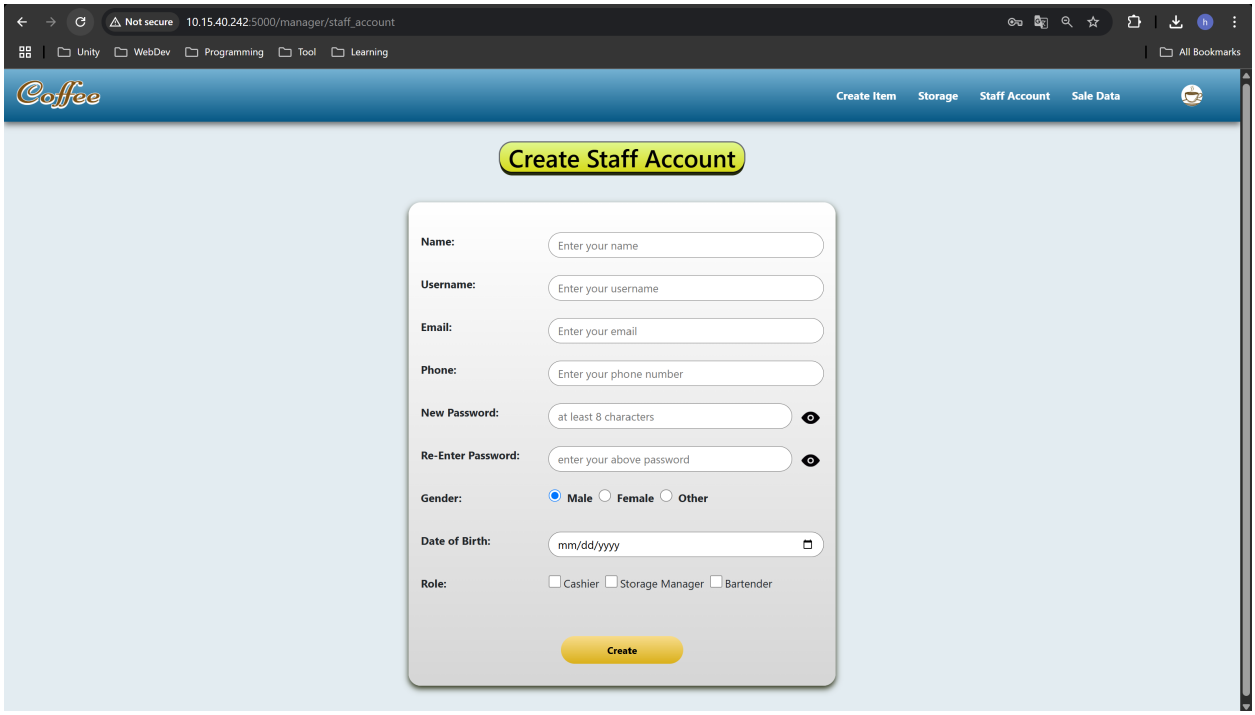


Figure 33: Staff Account Start Page

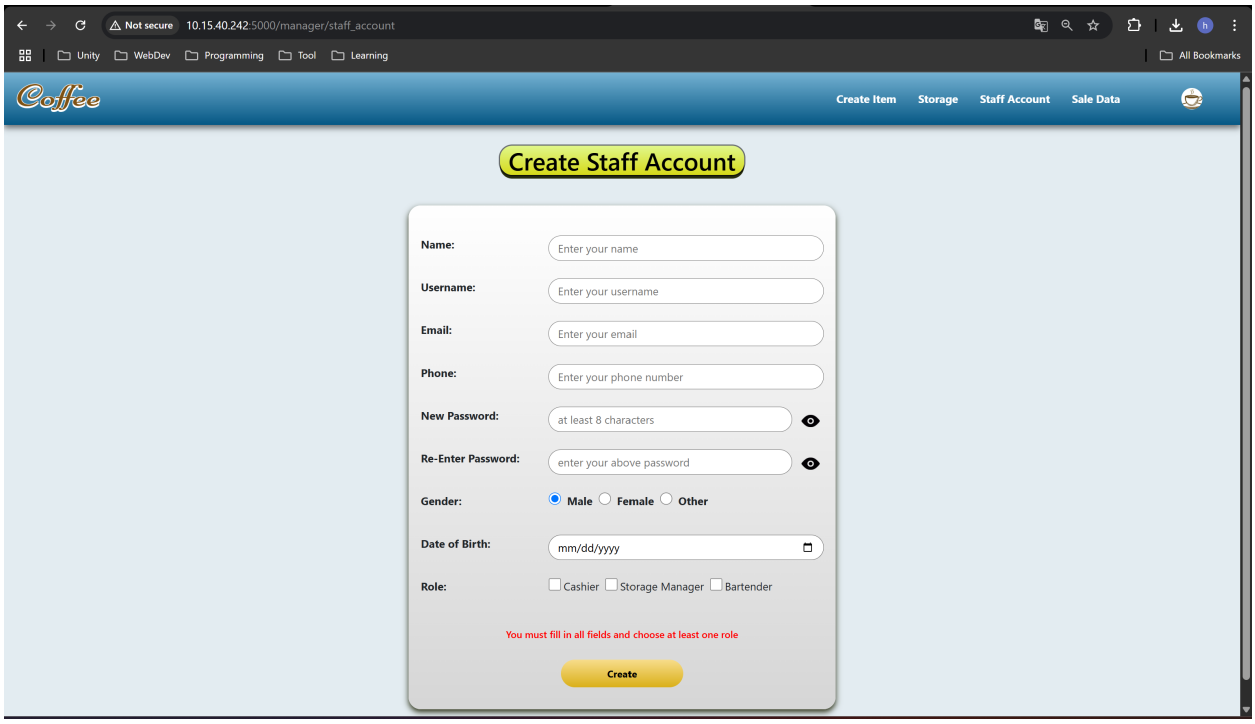


Figure 34: Staff Account Error: Not Choose Role

5.1.4 Sale Data

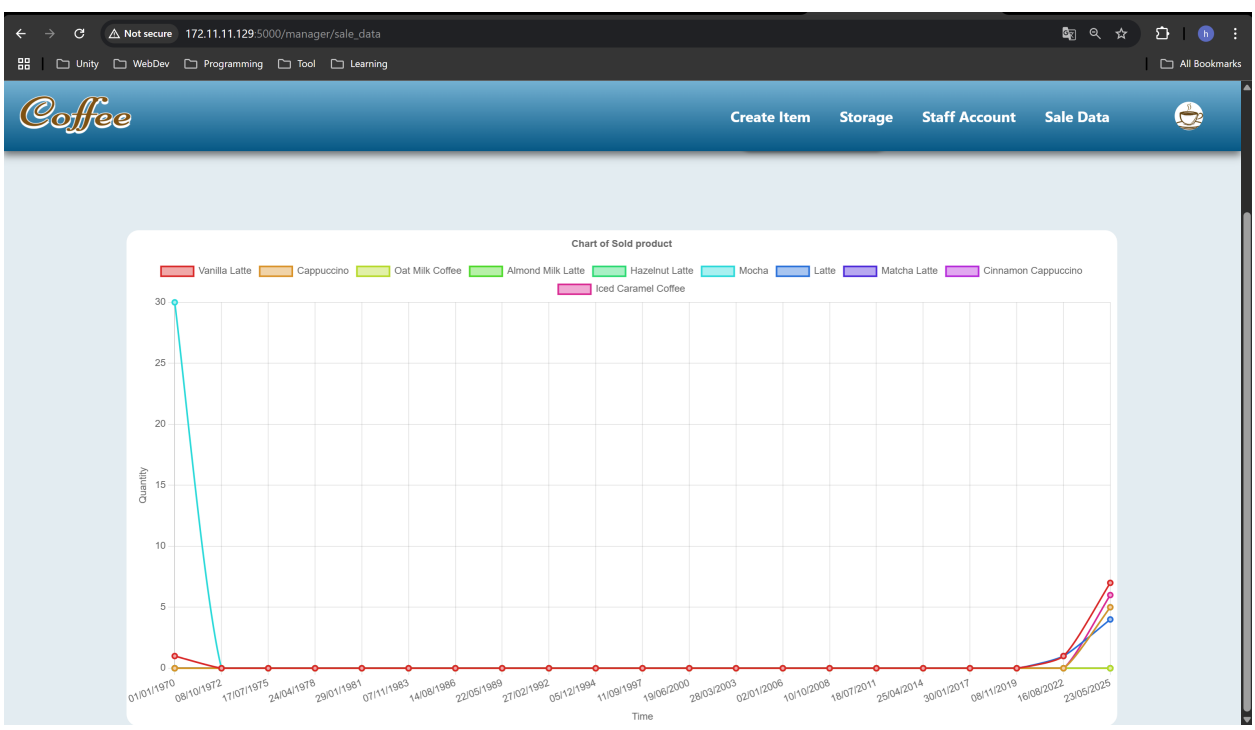


Figure 35: Sales Data Visualization

Troy University Resources

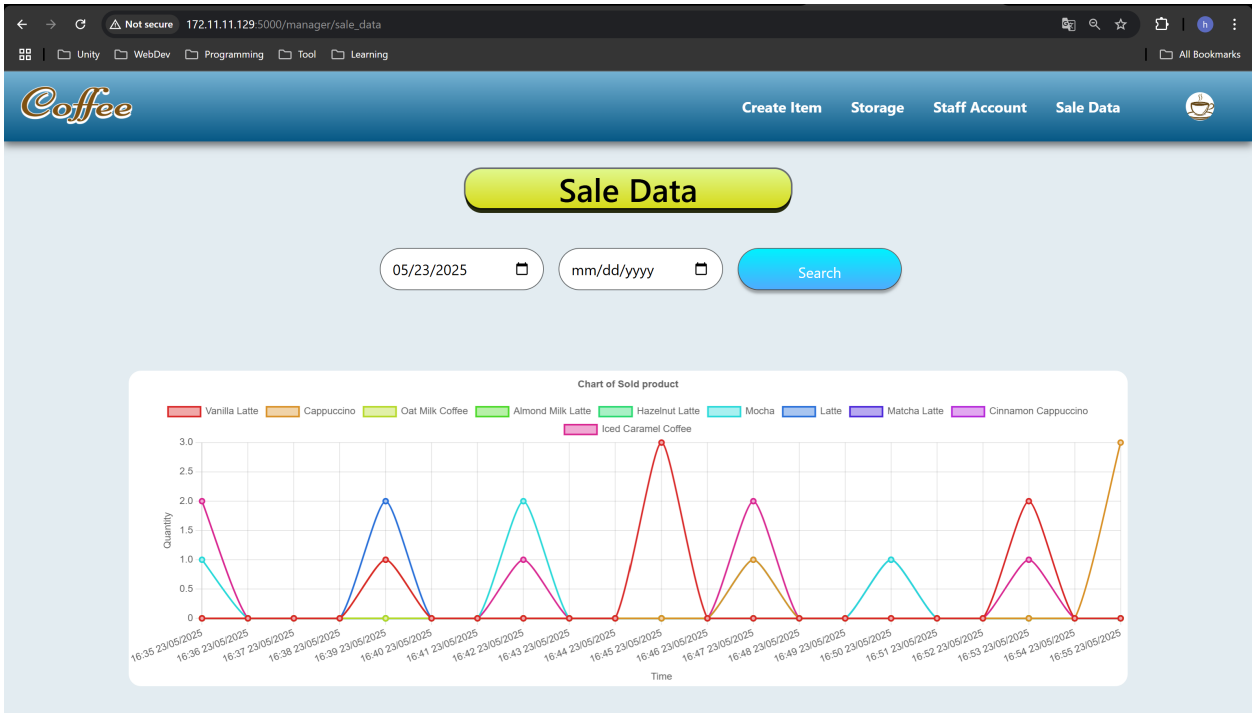


Figure 36: Sales Data: Choose Start Day

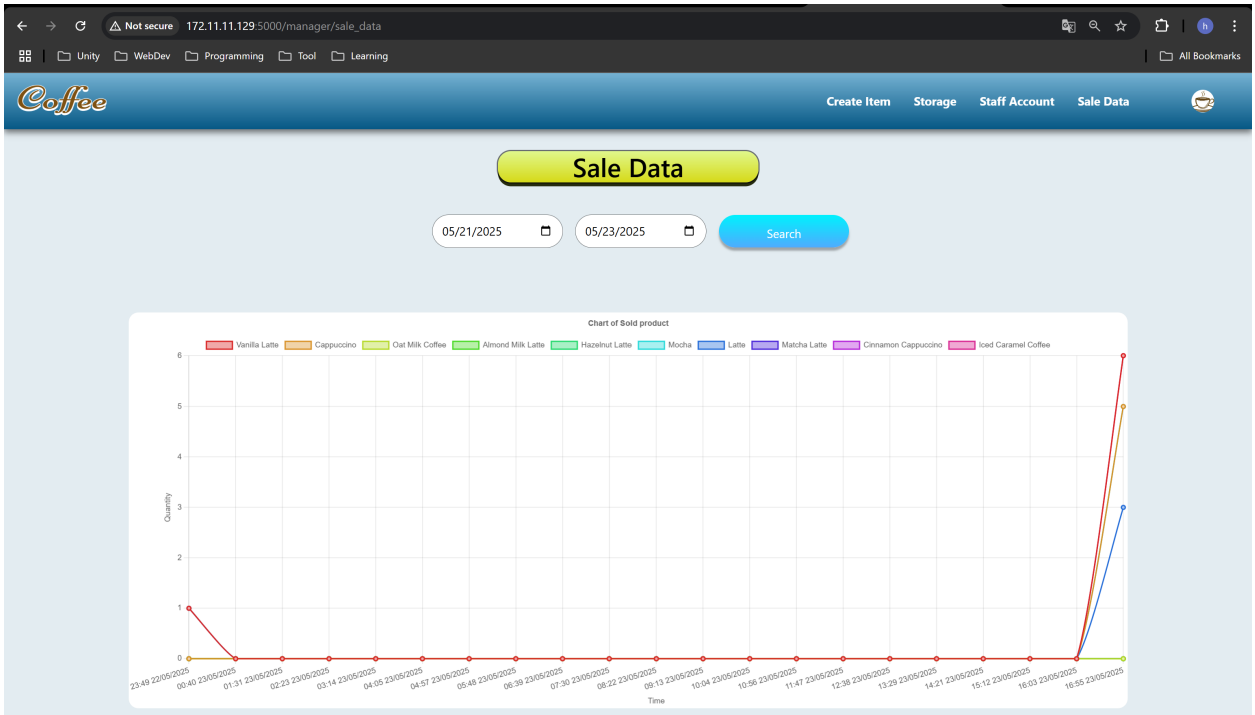


Figure 37: Sales Data: Choose In Range

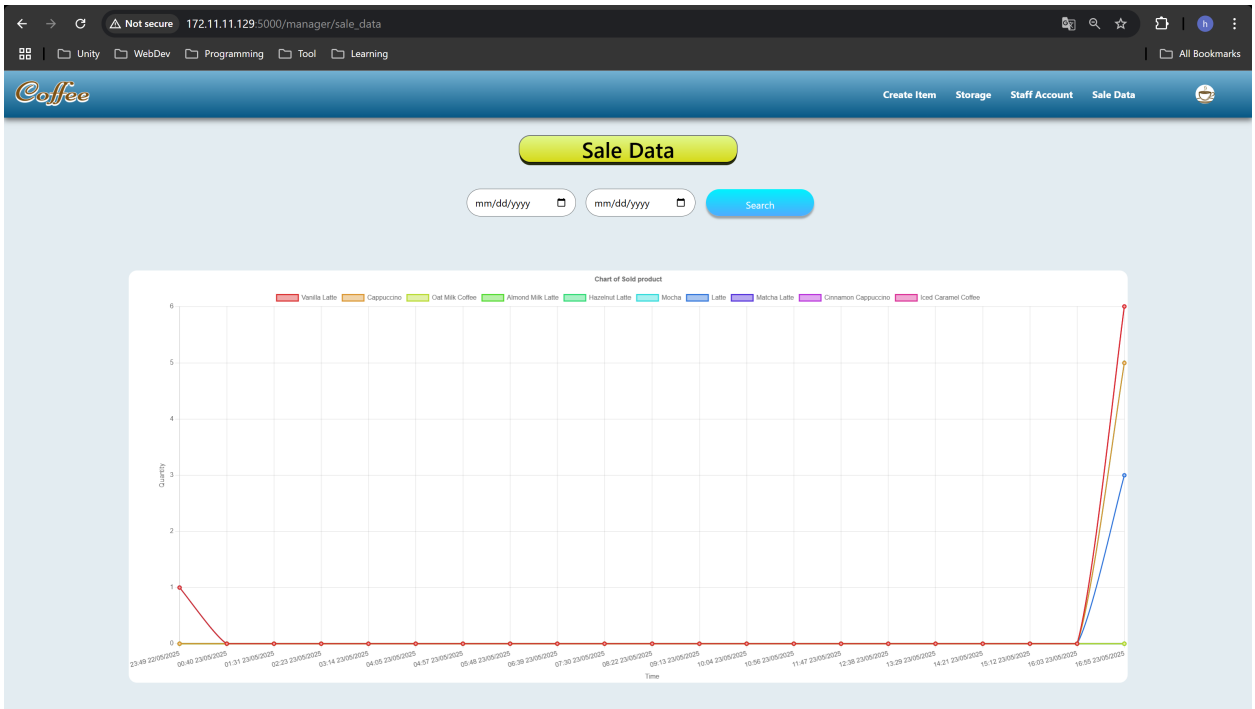


Figure 38: Sales Data Error: Not Choosing

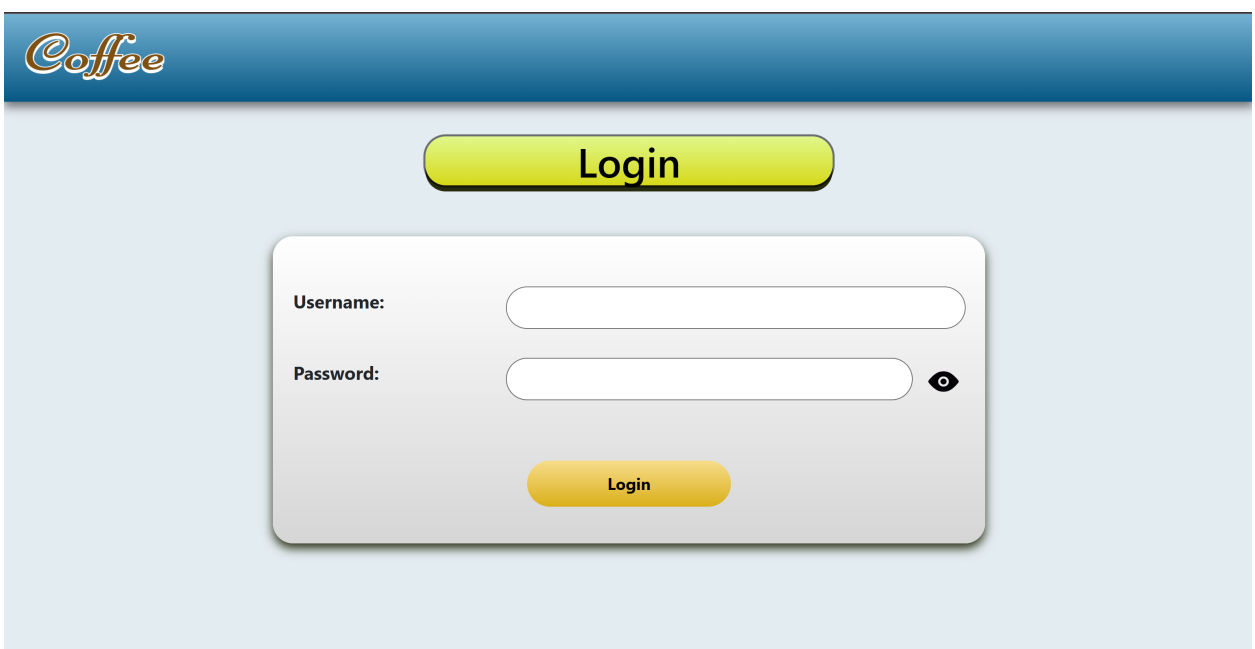


Figure 39: Sales Data: Unauthorized Access

Troy University Resources

5.1.5 New Item

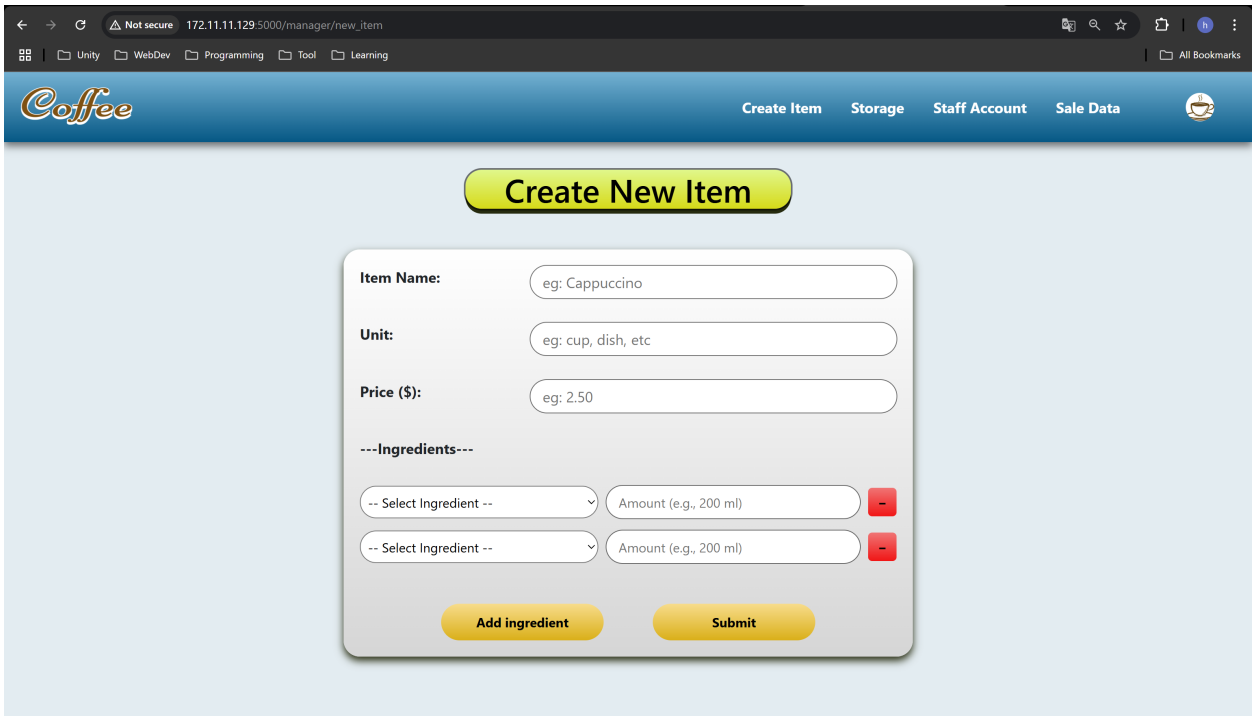


Figure 40: New Item Start Page

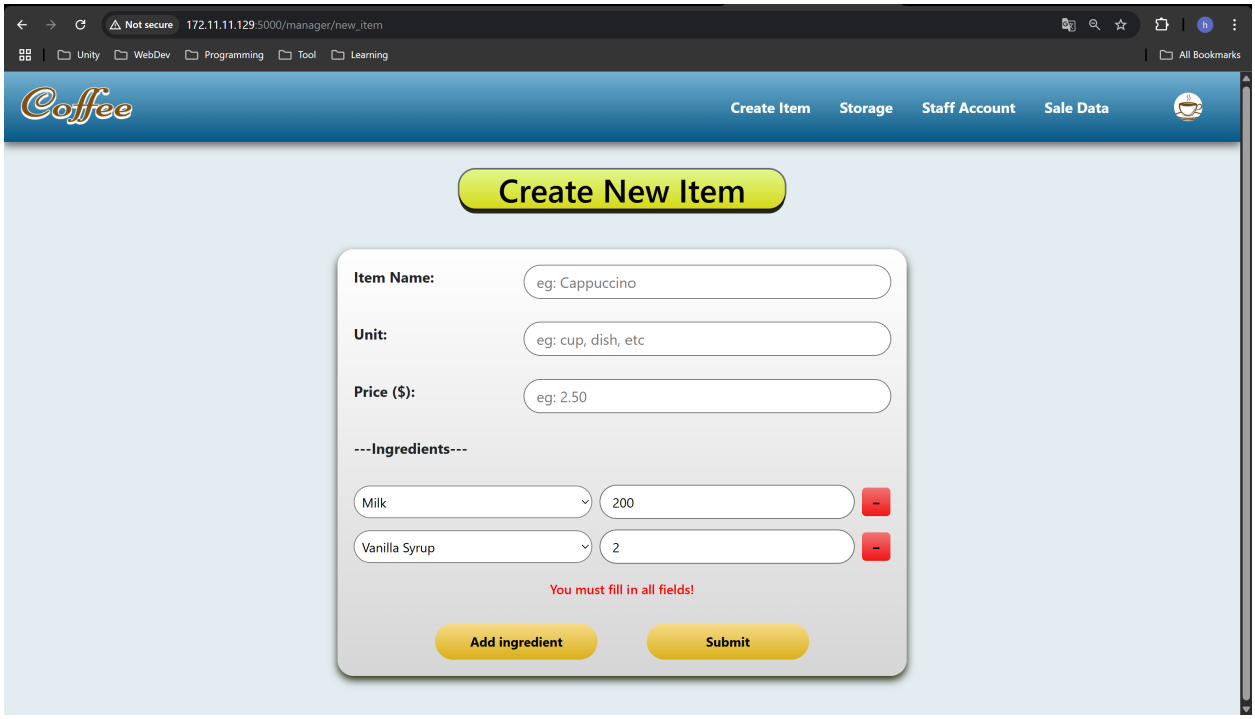


Figure 41: New Item Error: Not Fill All Field

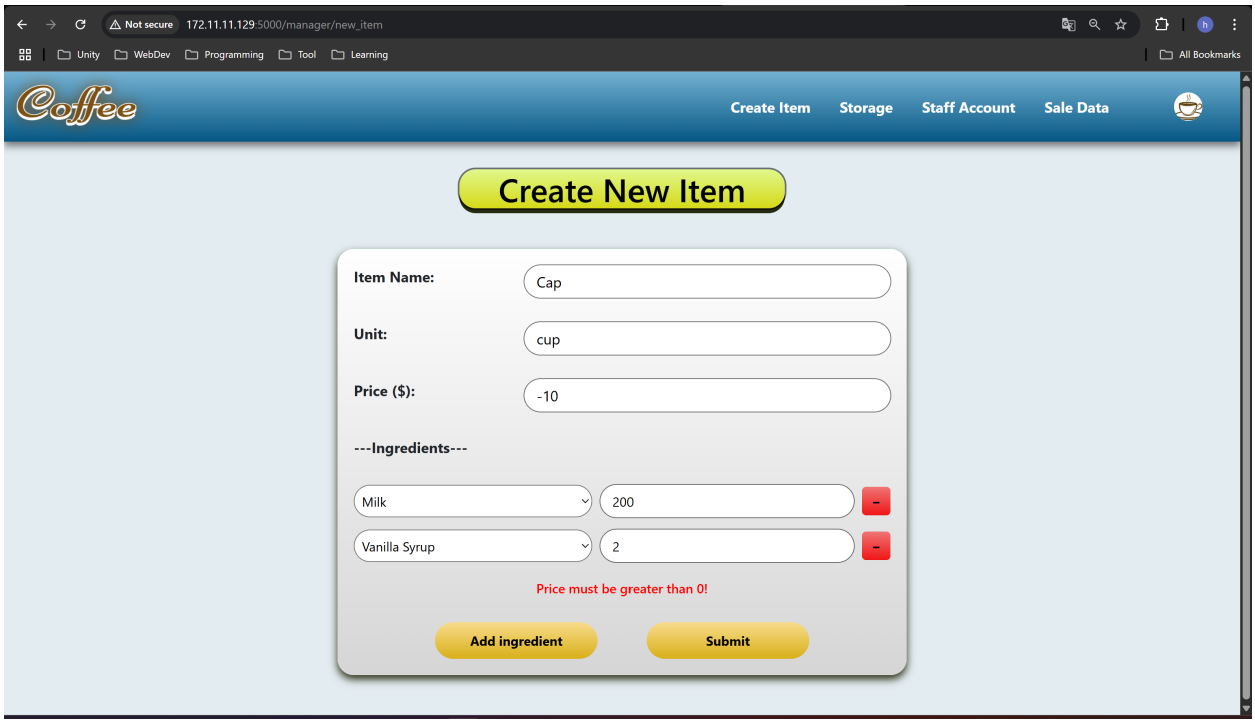


Figure 42: New Item Error: Invalid Price

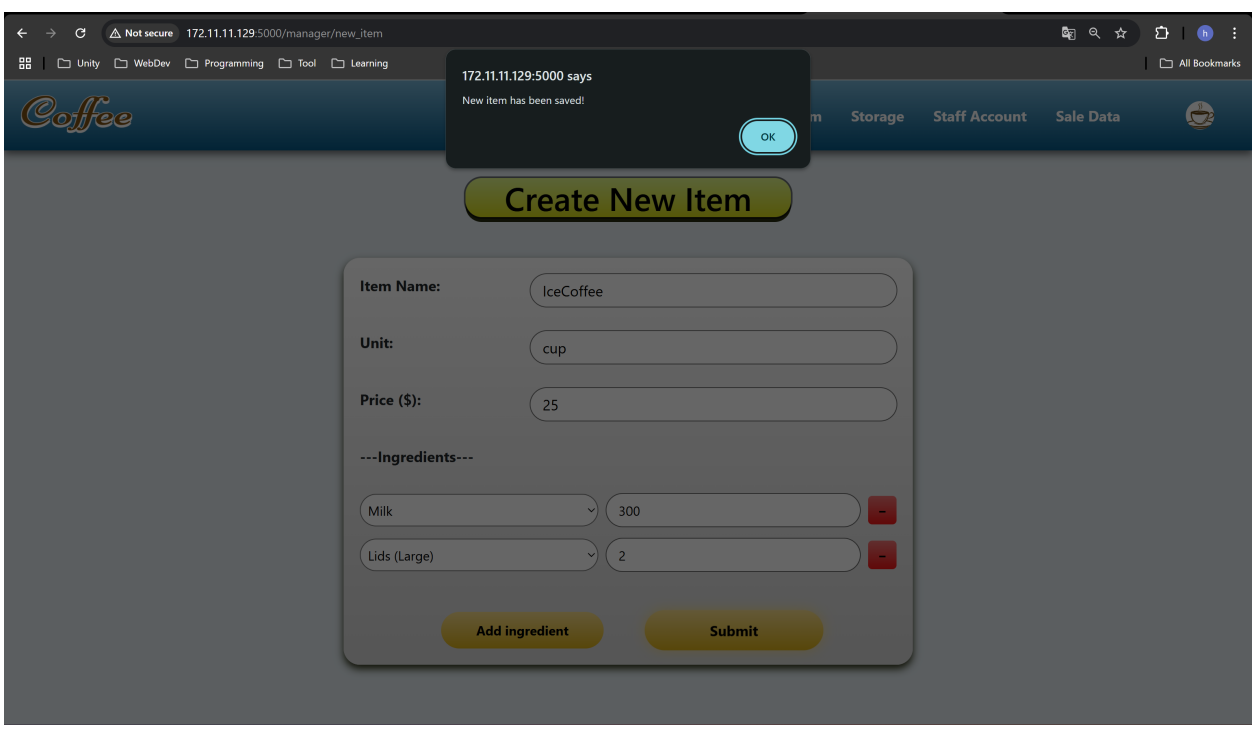


Figure 43: New Item Created Successfully

Troy University Resources

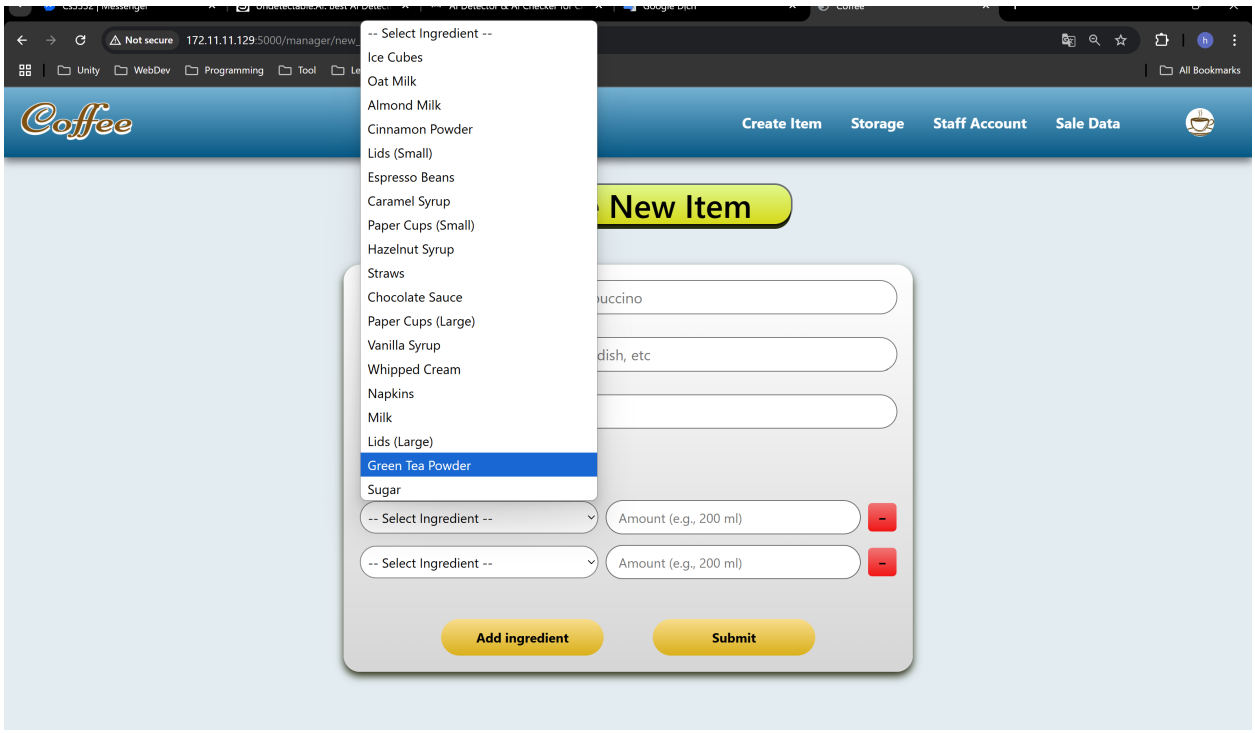


Figure 44: New Item: Choose Ingredient

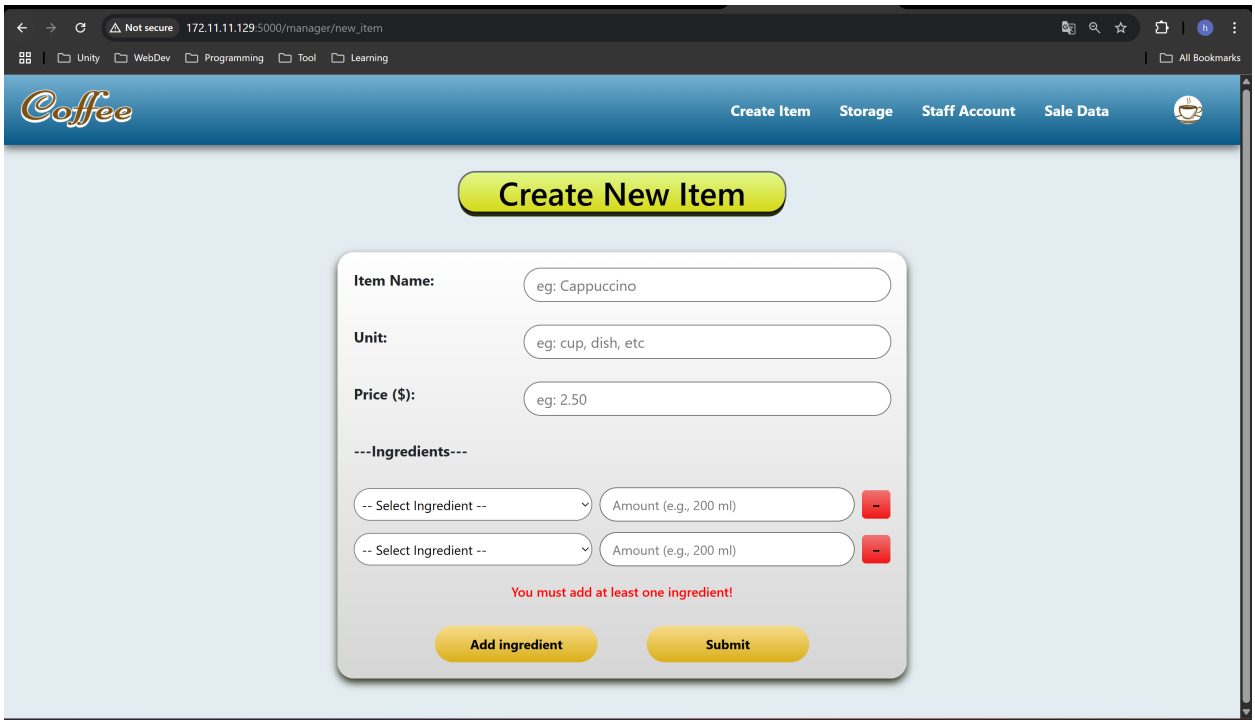


Figure 45: New Item Error: At Least One Ingredient

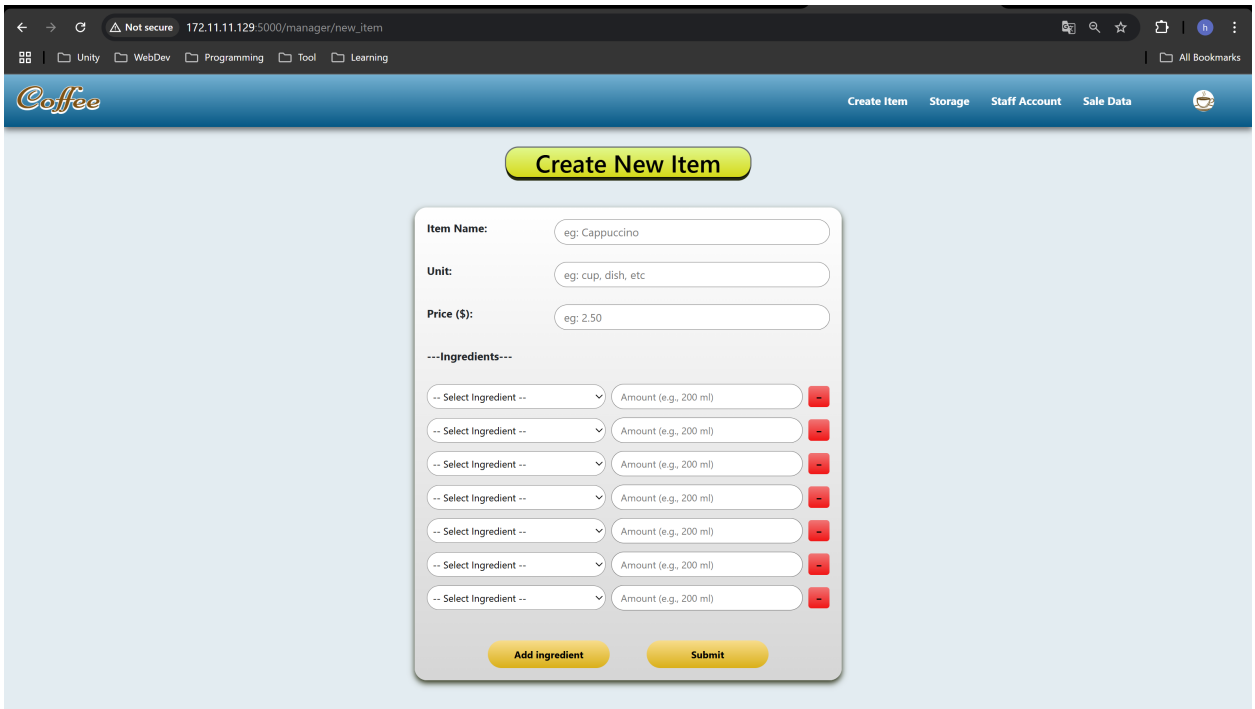


Figure 46: New Item: Add Ingredient

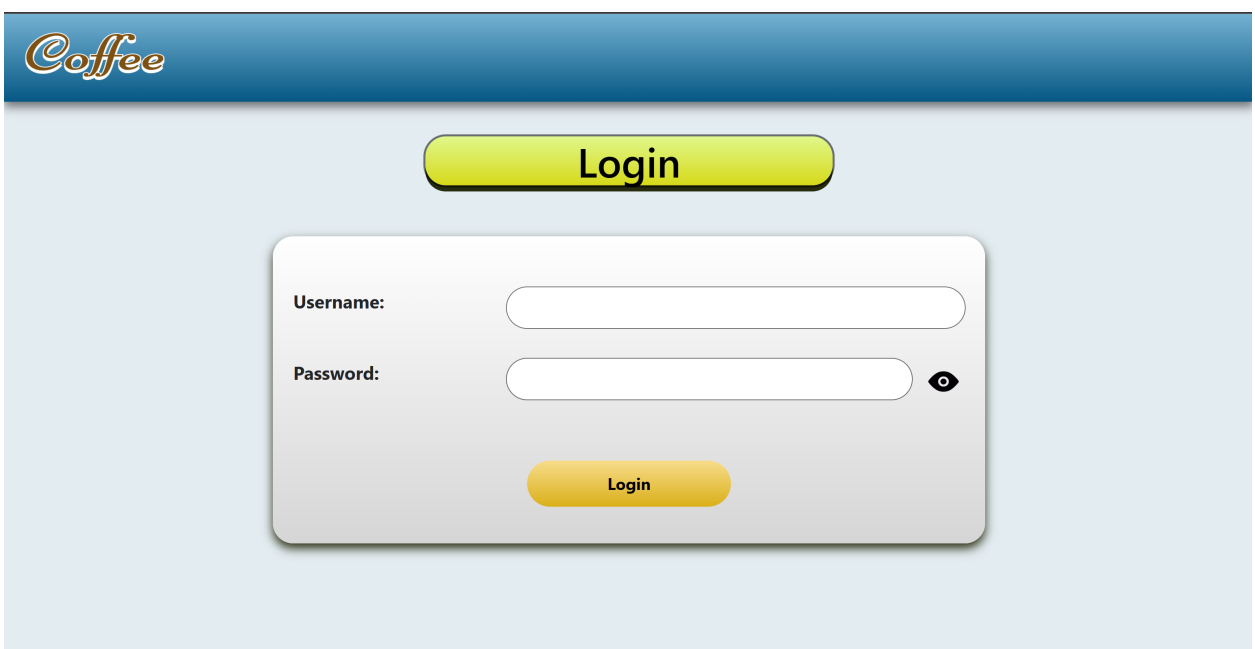


Figure 47: New Item: Unauthorized Access

Troy University Resources

5.1.6 Tab Translation

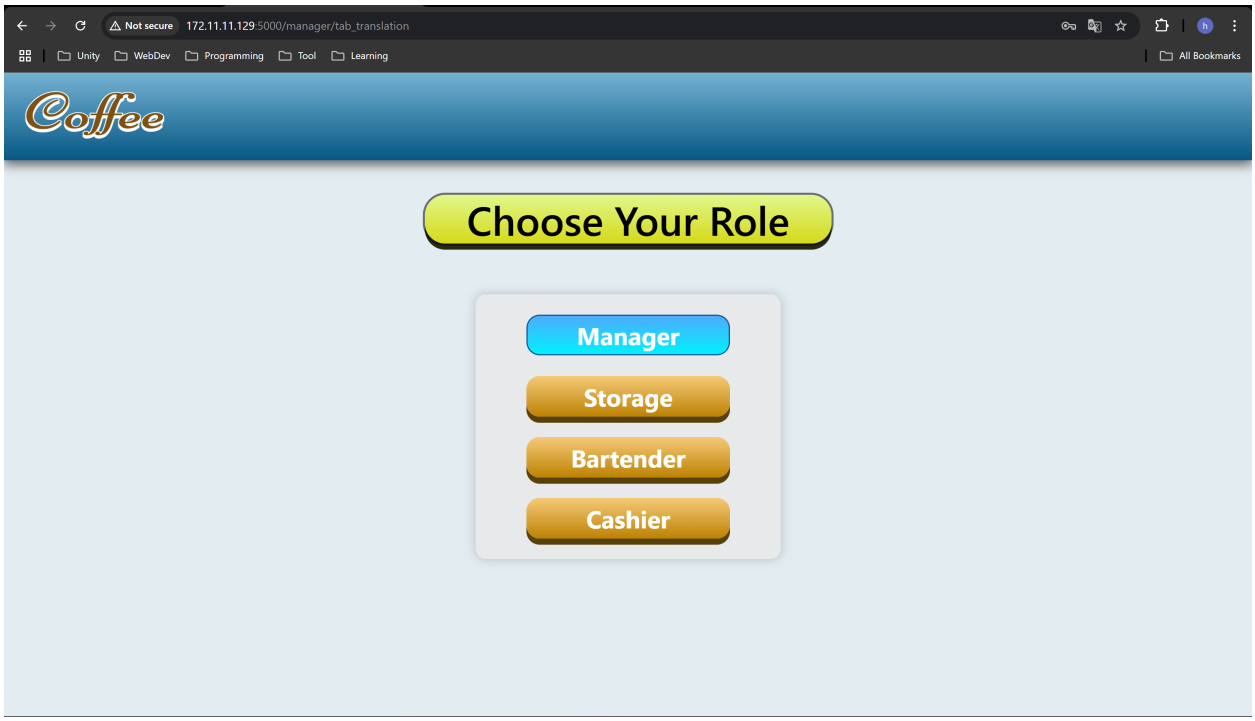


Figure 48: Tab Translation: Select

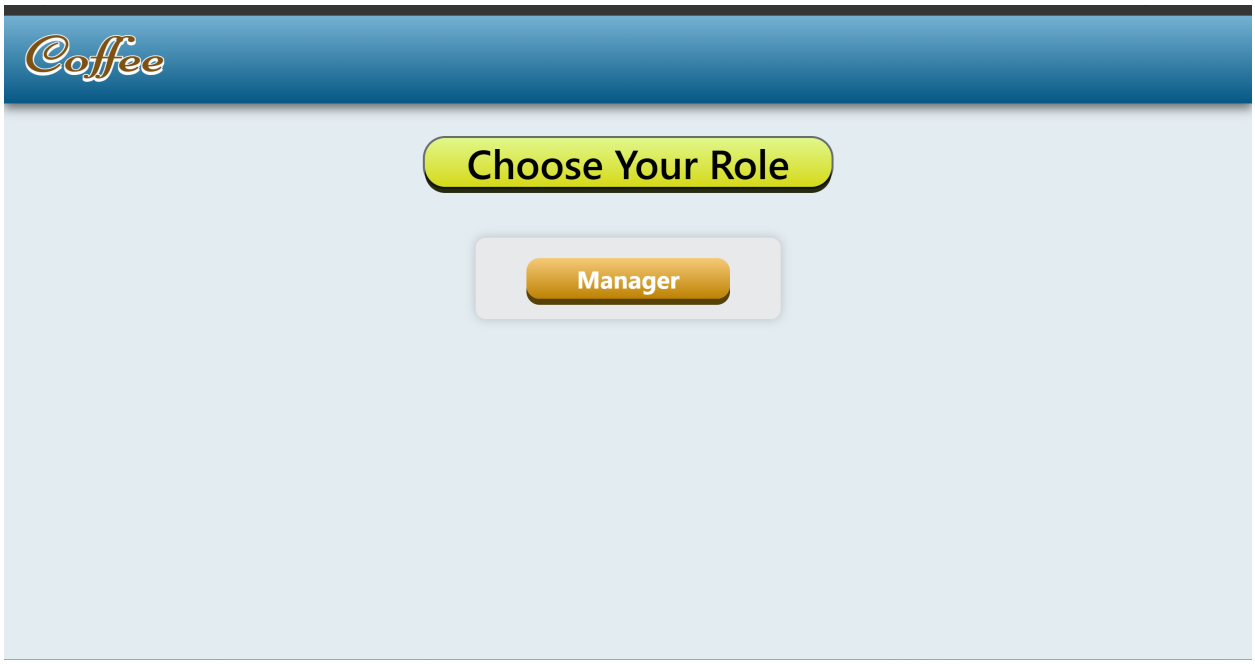


Figure 49: Tab Translation: Role

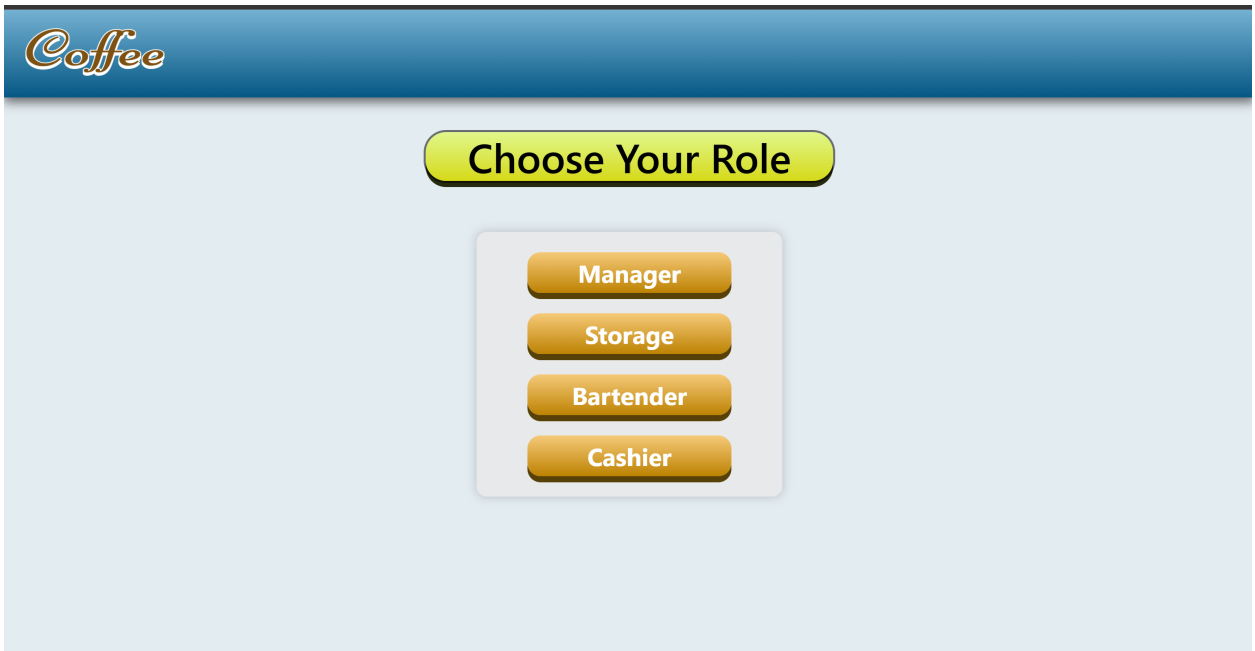


Figure 50: Tab Translation: All Role

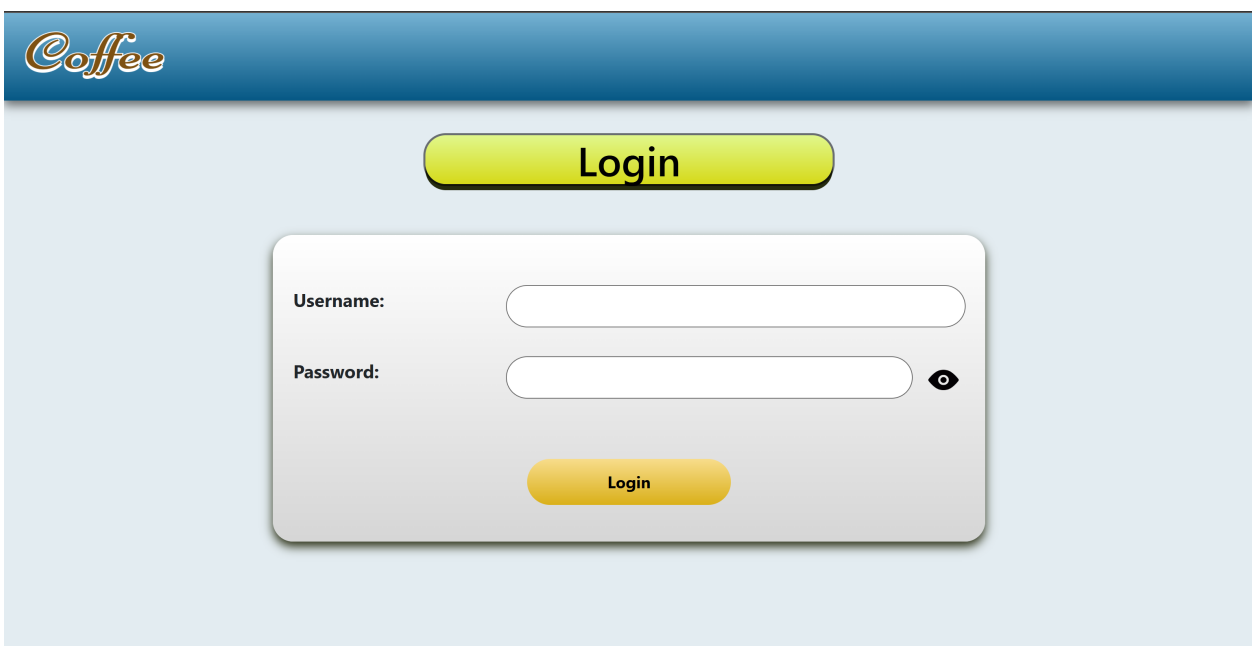


Figure 51: Tab Translation: Unauthorized Access

5.2 Cashier UI

The Cashier UI is optimized for fast order entry, table management, and billing. It supports both customer-facing and back-office operations, with clear feedback and error messages.

Troy University Resources

5.2.1 Order

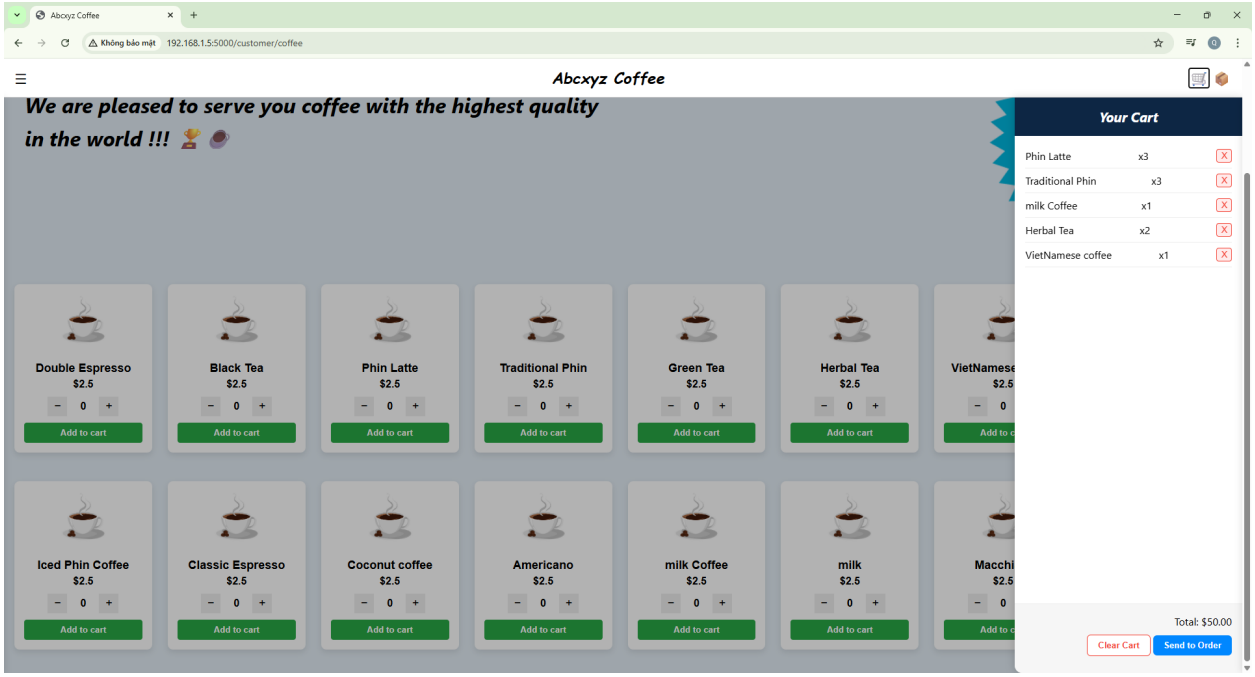


Figure 52: Cashier Order Entry

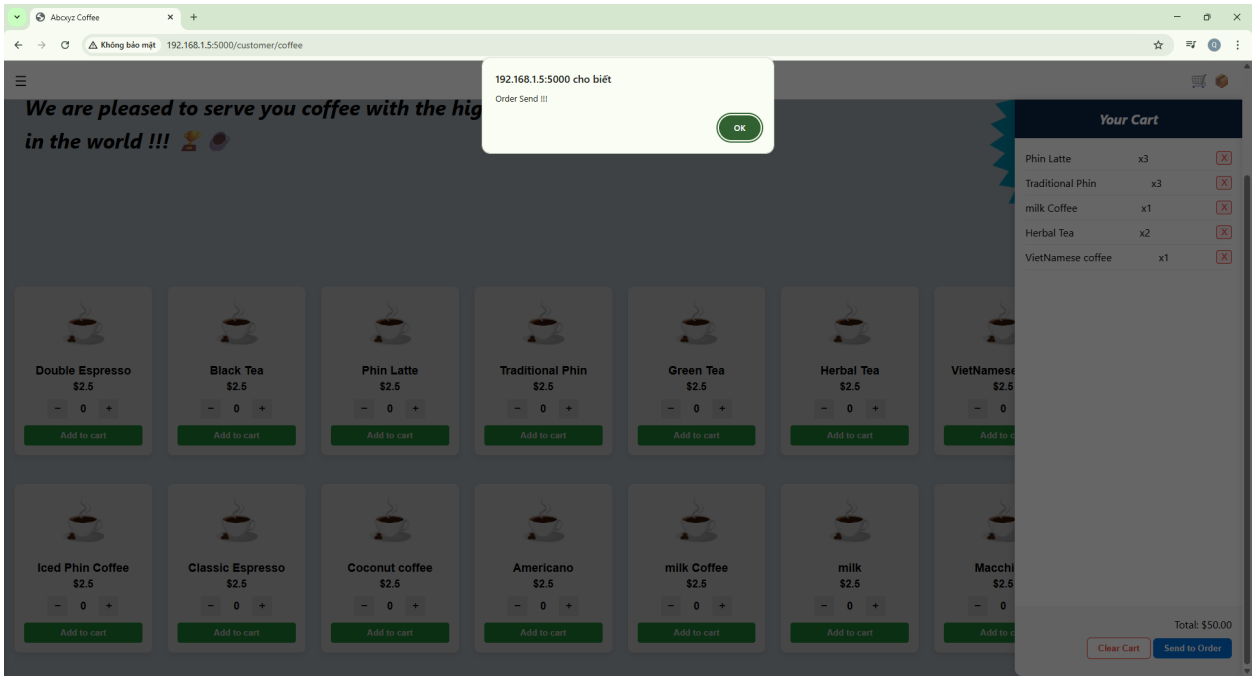


Figure 53: Cashier Order Entry (2)

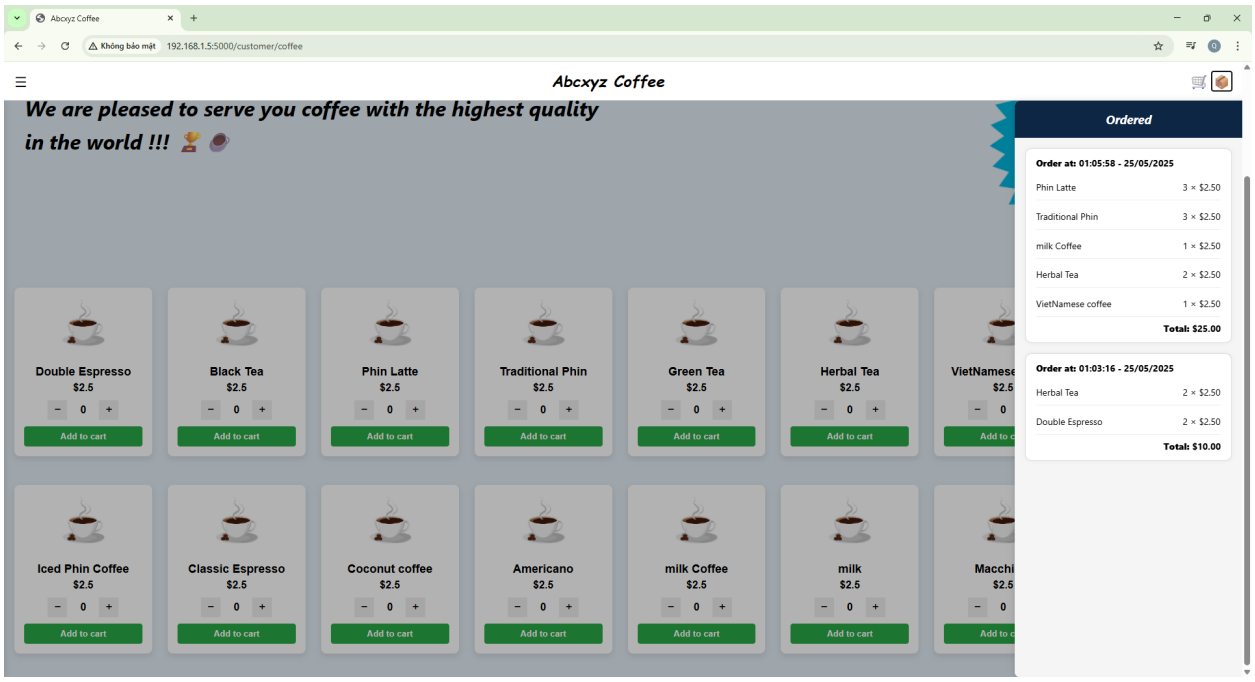


Figure 54: Cashier Order Entry (3)

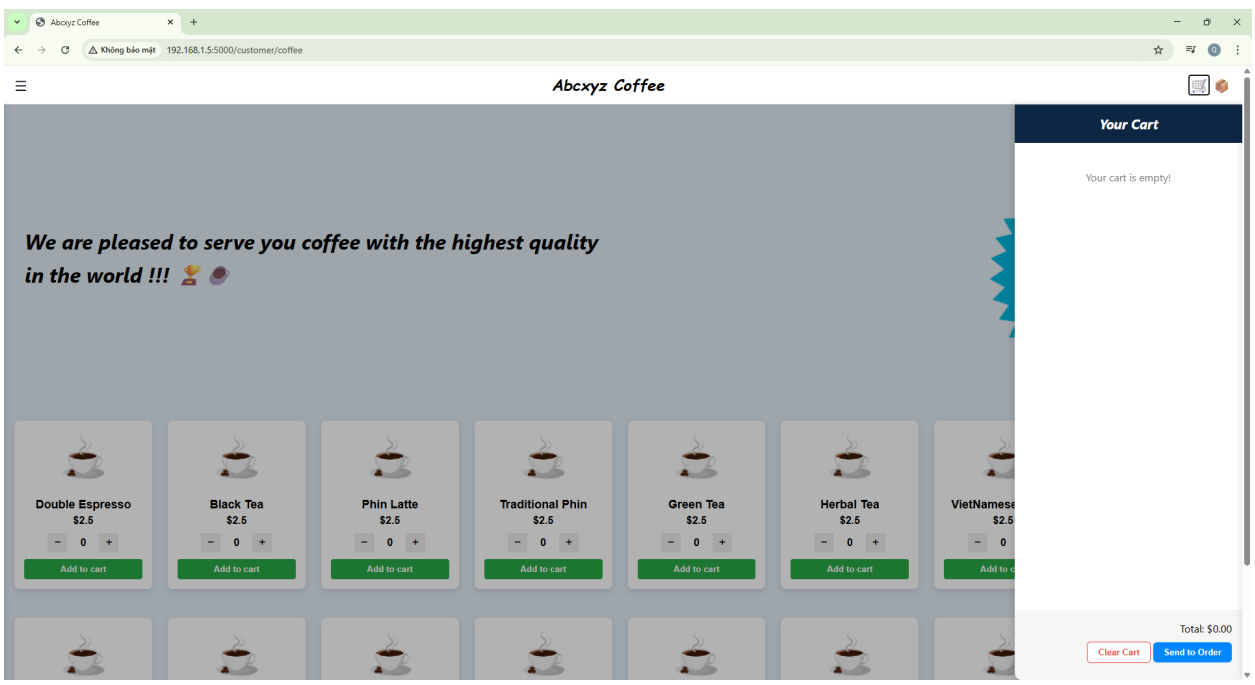


Figure 55: Cashier Order Entry (4)

Troy University Resources

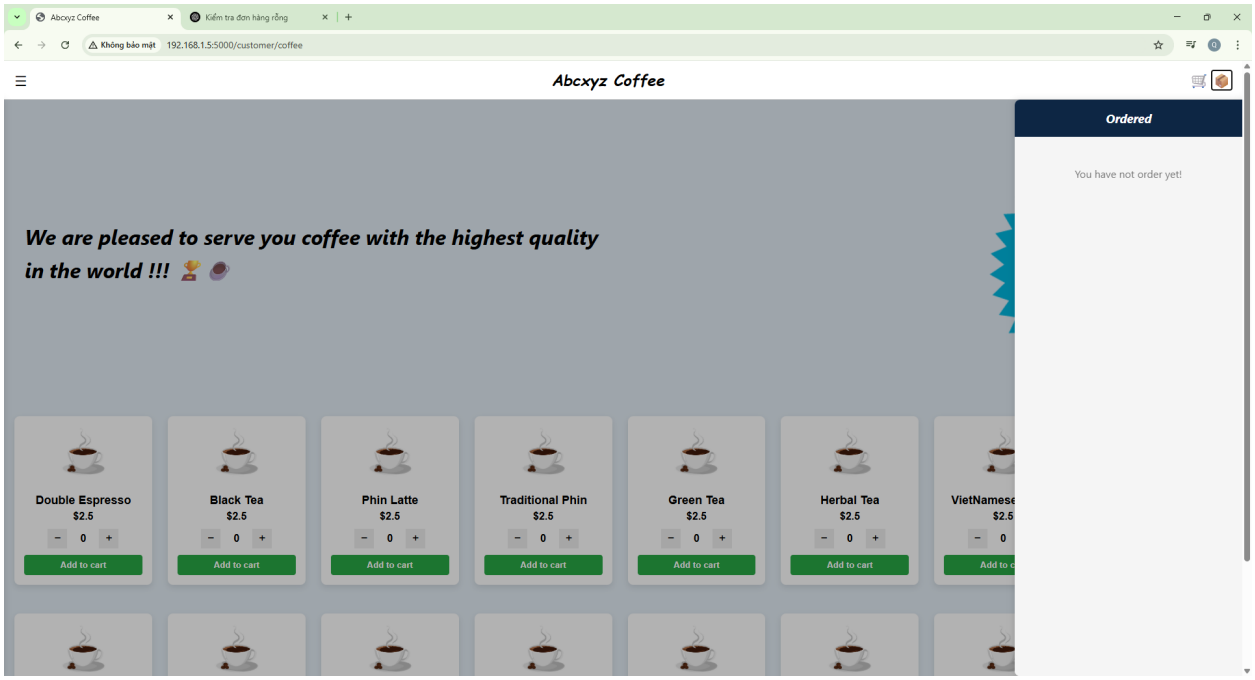


Figure 56: Cashier Order Entry (5)

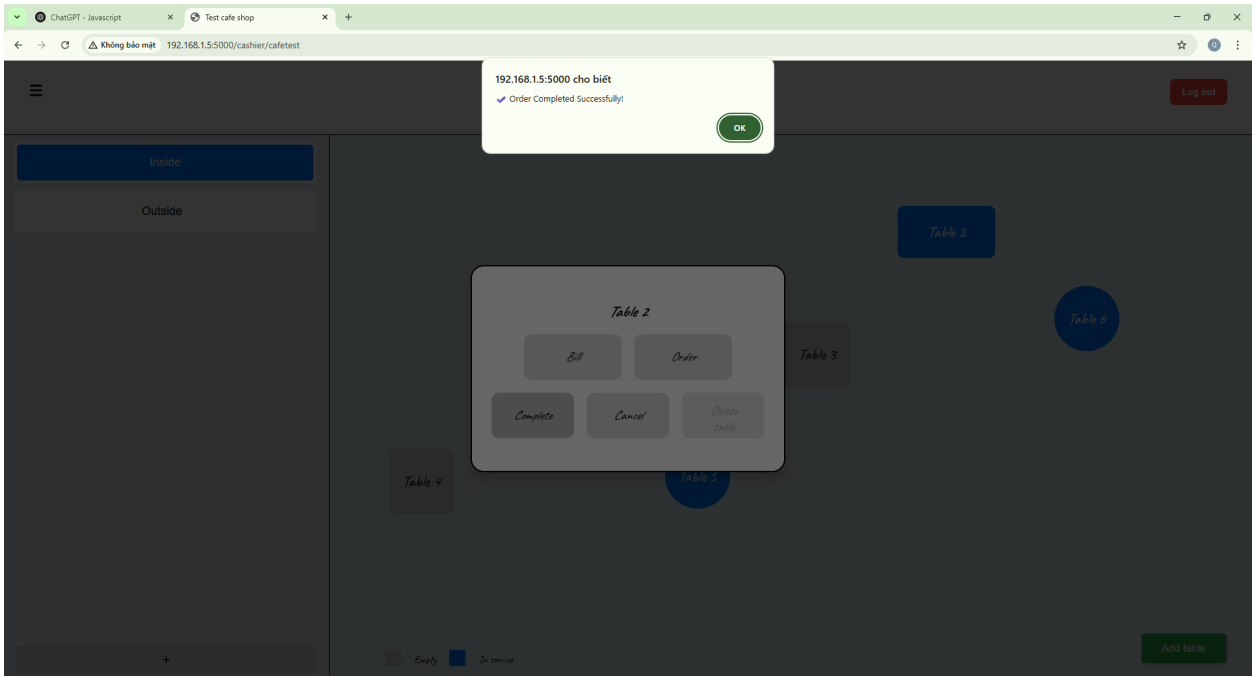


Figure 57: Cashier Order Entry (6)

5.2.2 Map

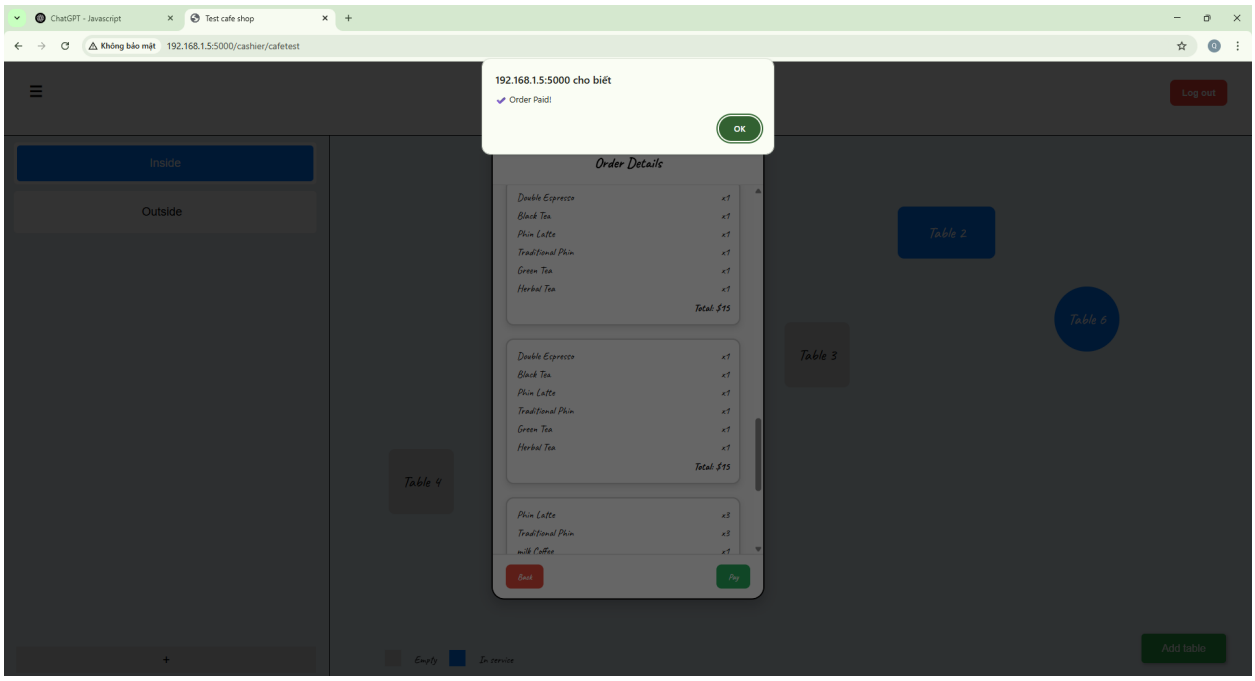


Figure 58: Table Map Management

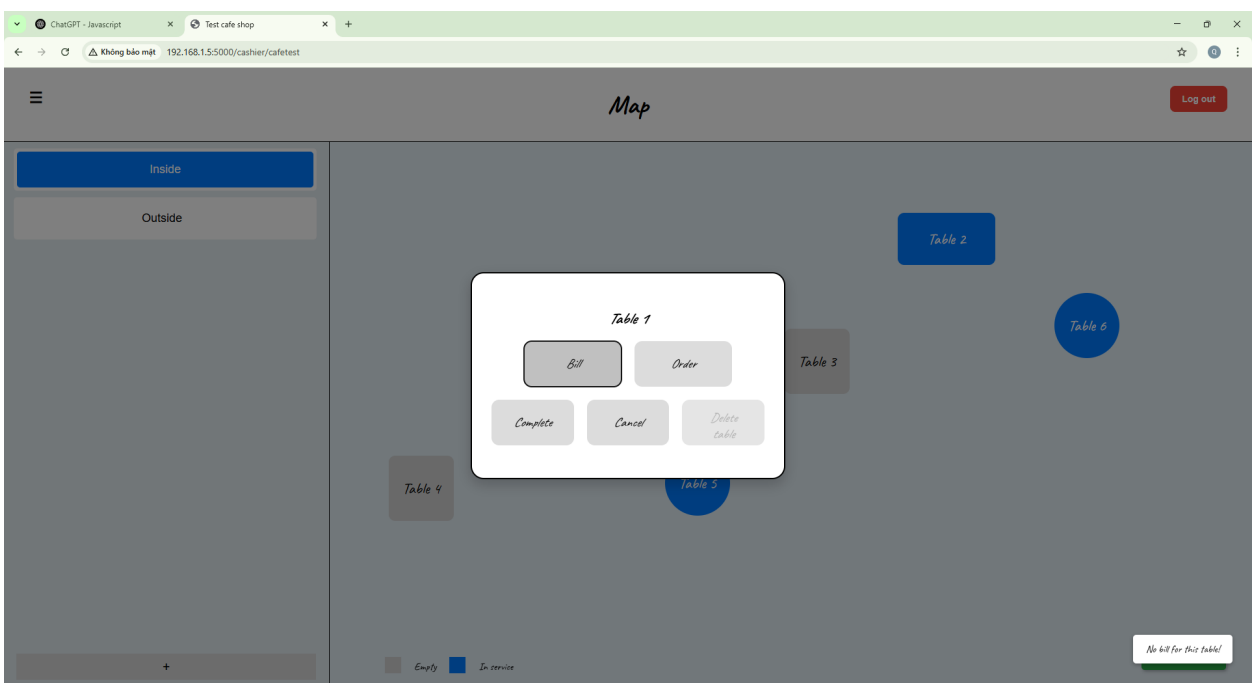


Figure 59: Table Map Management (2)

Troy University Resources

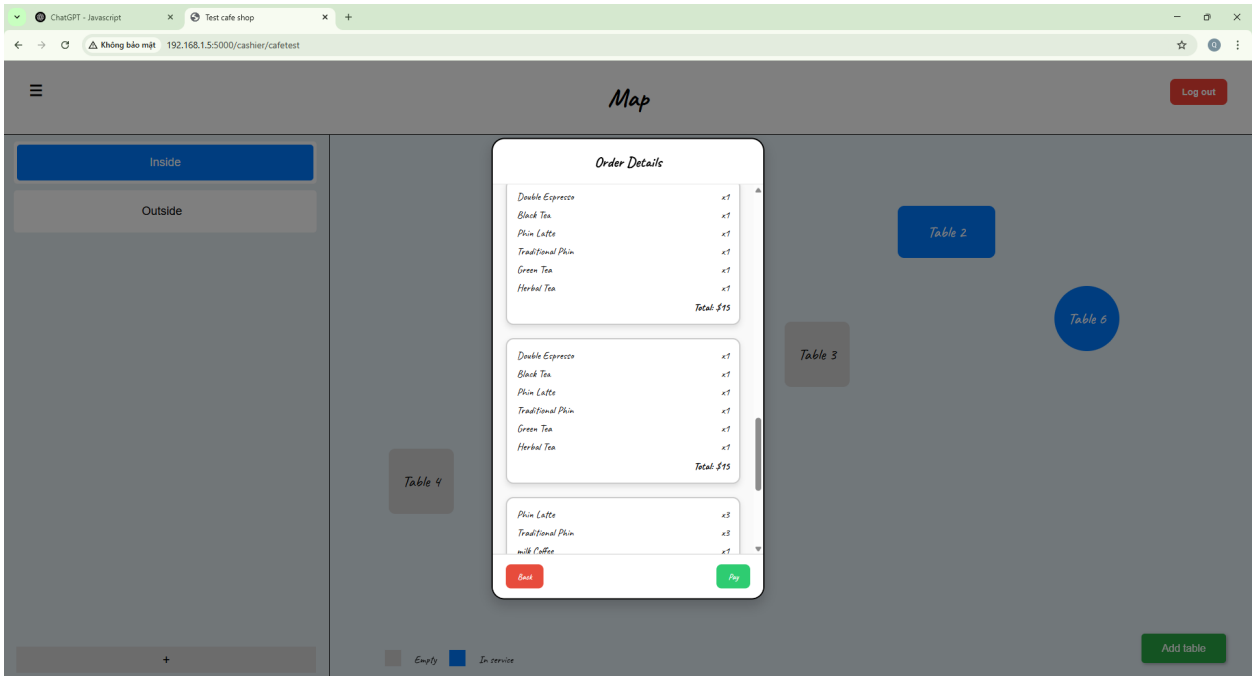


Figure 60: Table Map Management (3)

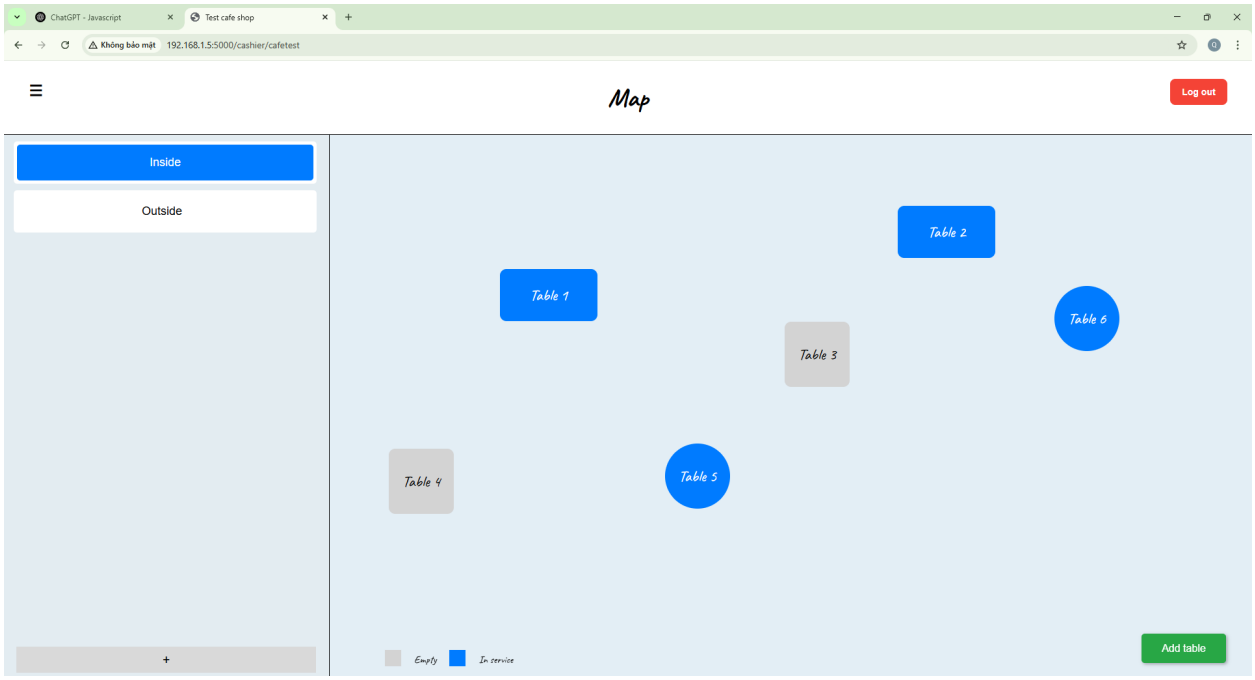


Figure 61: Table Map Management (4)

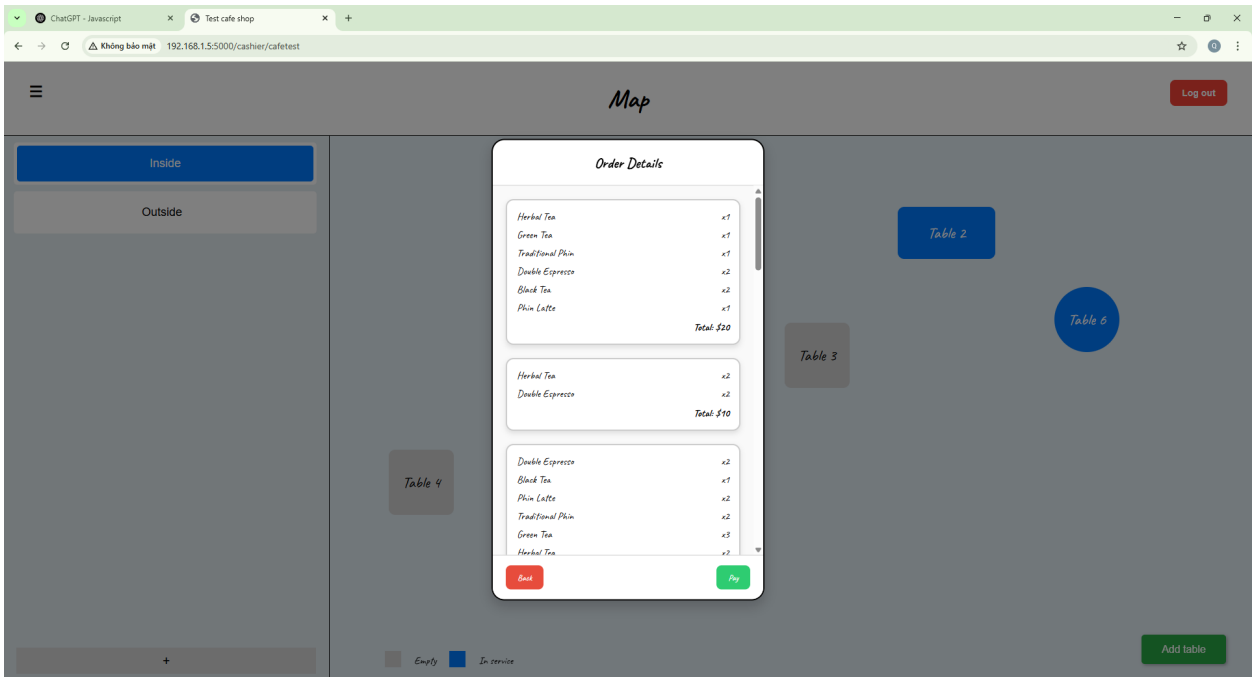


Figure 62: Table Map Management (5)

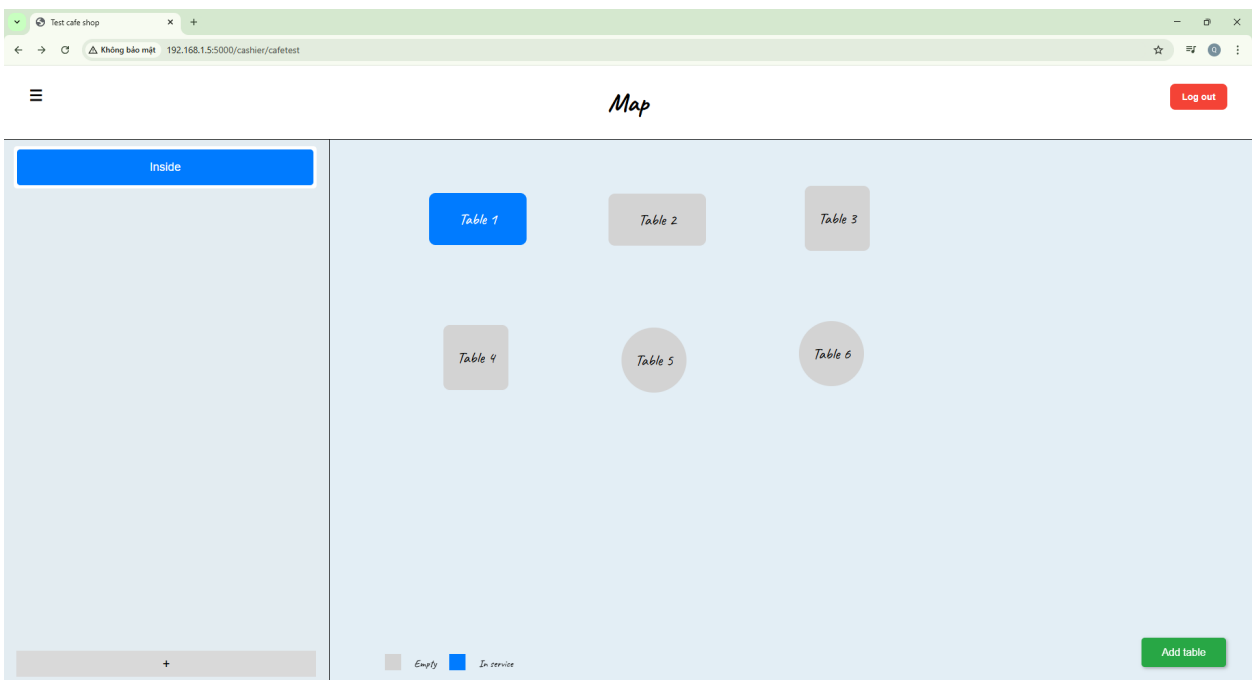


Figure 63: Table Map Management (6)

Troy University Resources

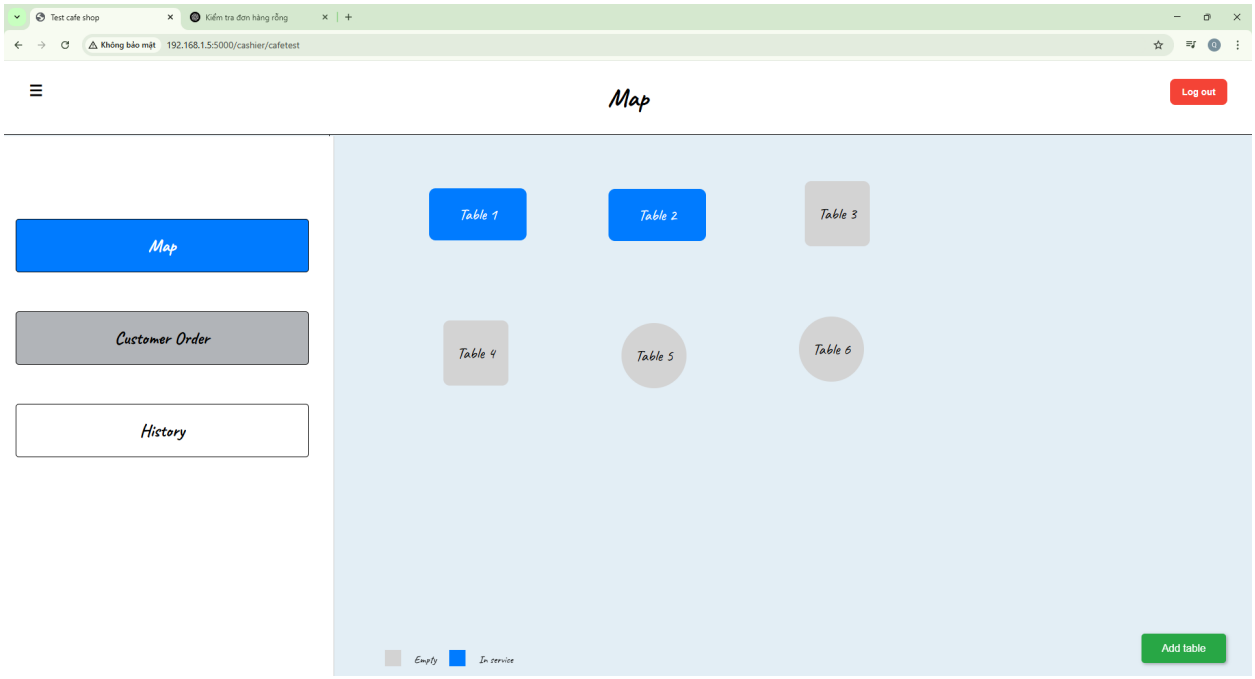


Figure 64: Table Map Management (7)

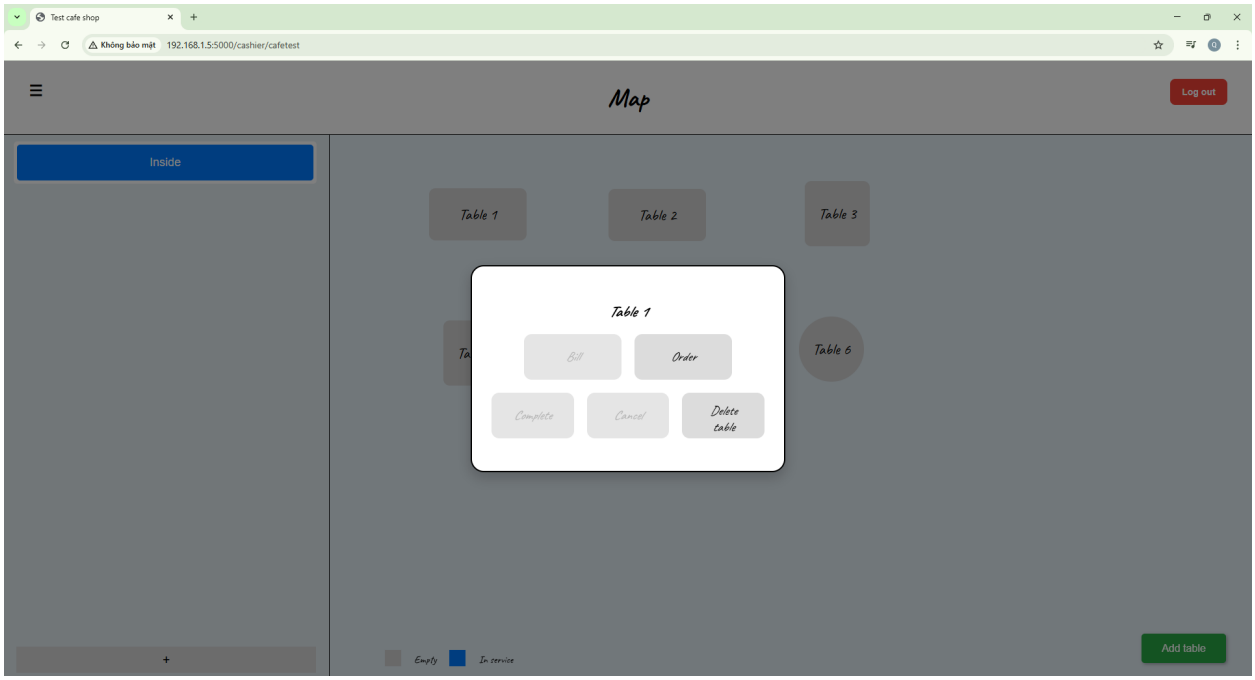


Figure 65: Table Map Management (8)

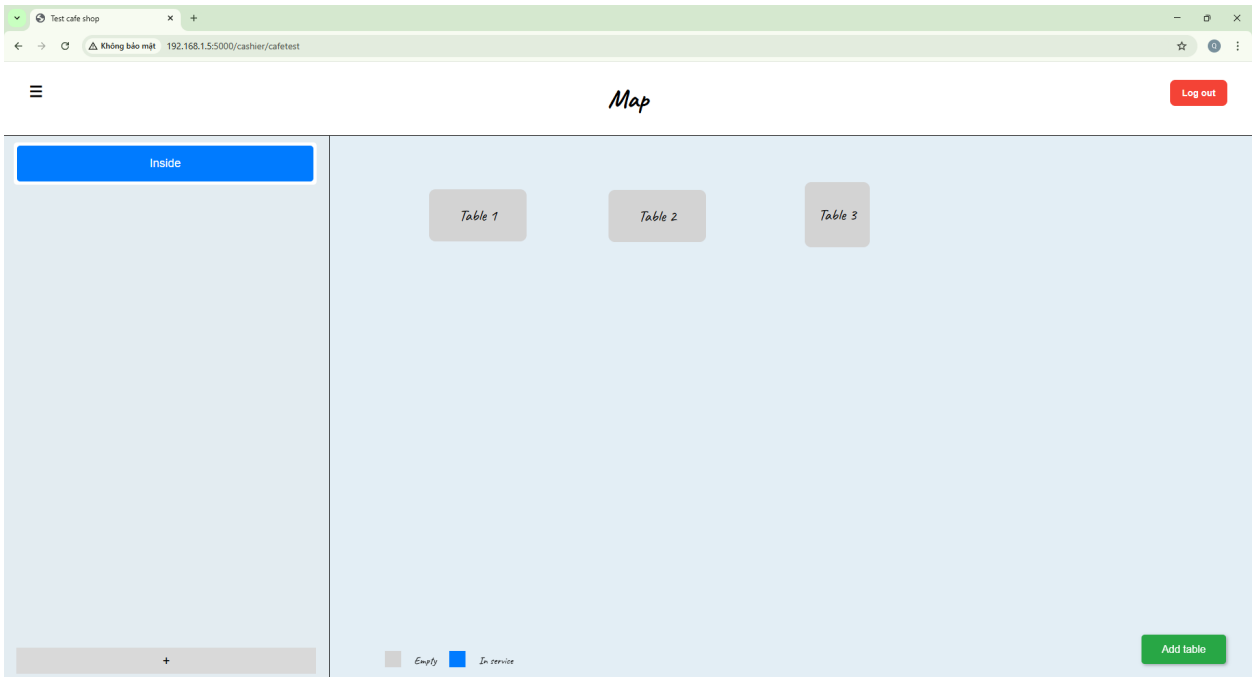


Figure 66: Table Map Management (9)

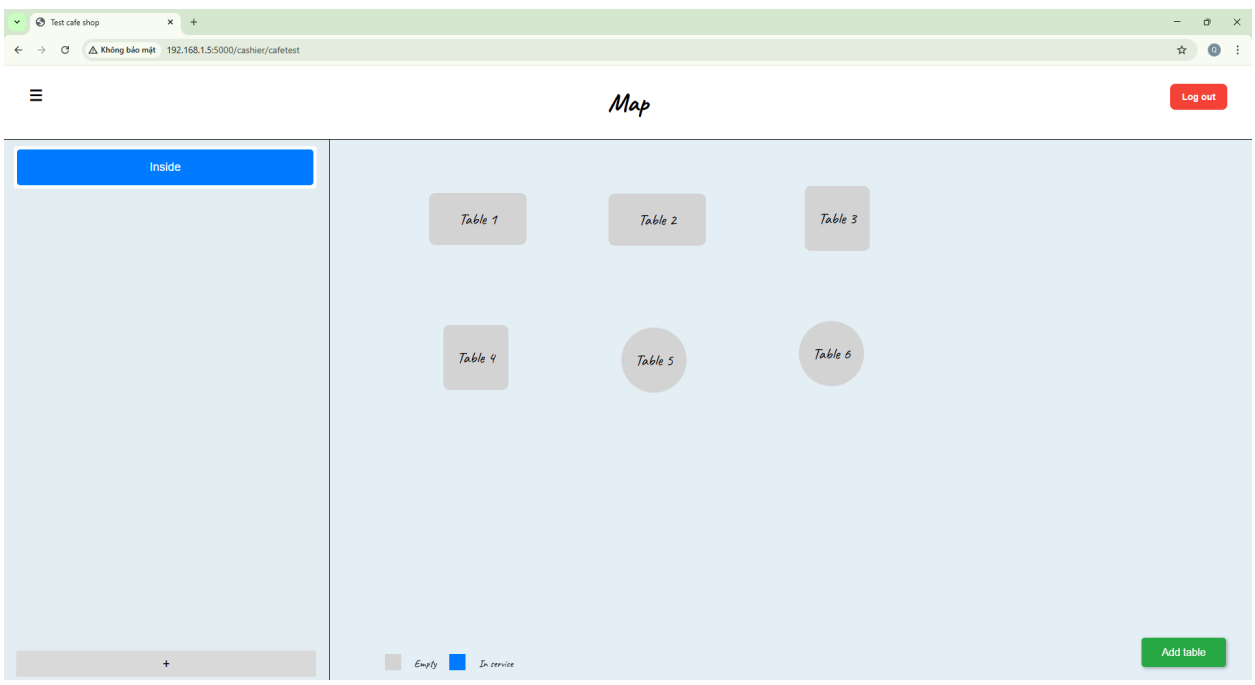


Figure 67: Table Map Management (10)

Troy University Resources

5.2.3 History

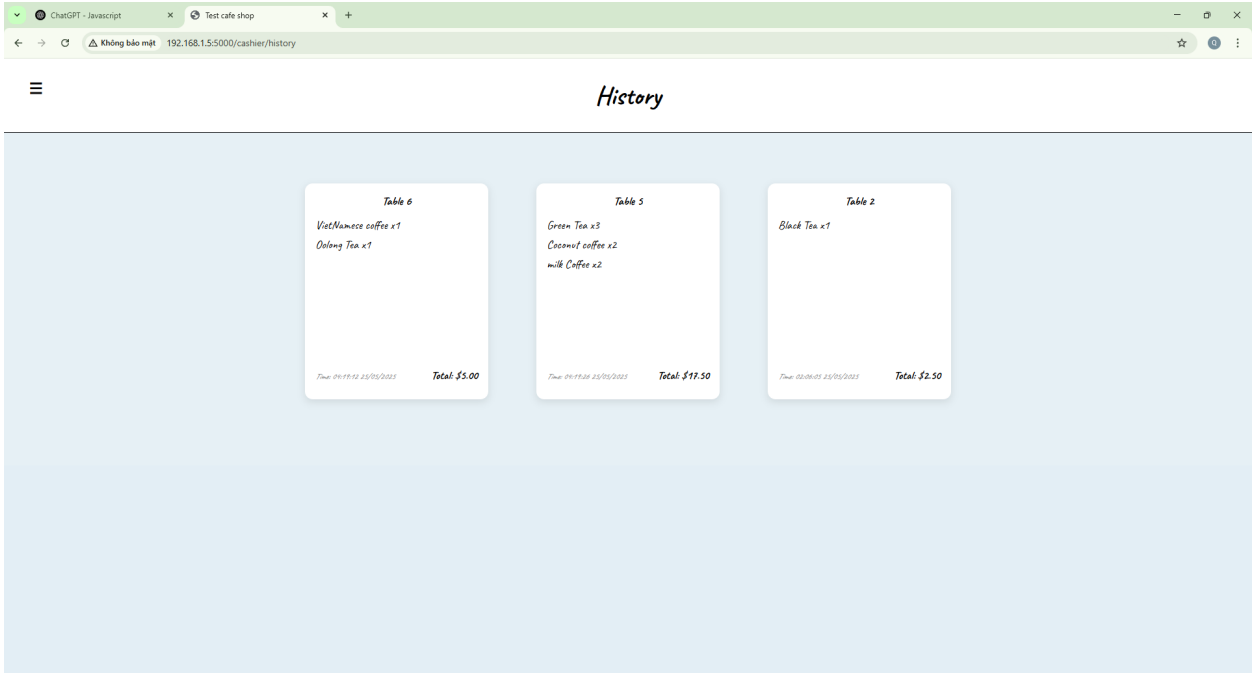


Figure 68: Order History

5.2.4 Customer Order

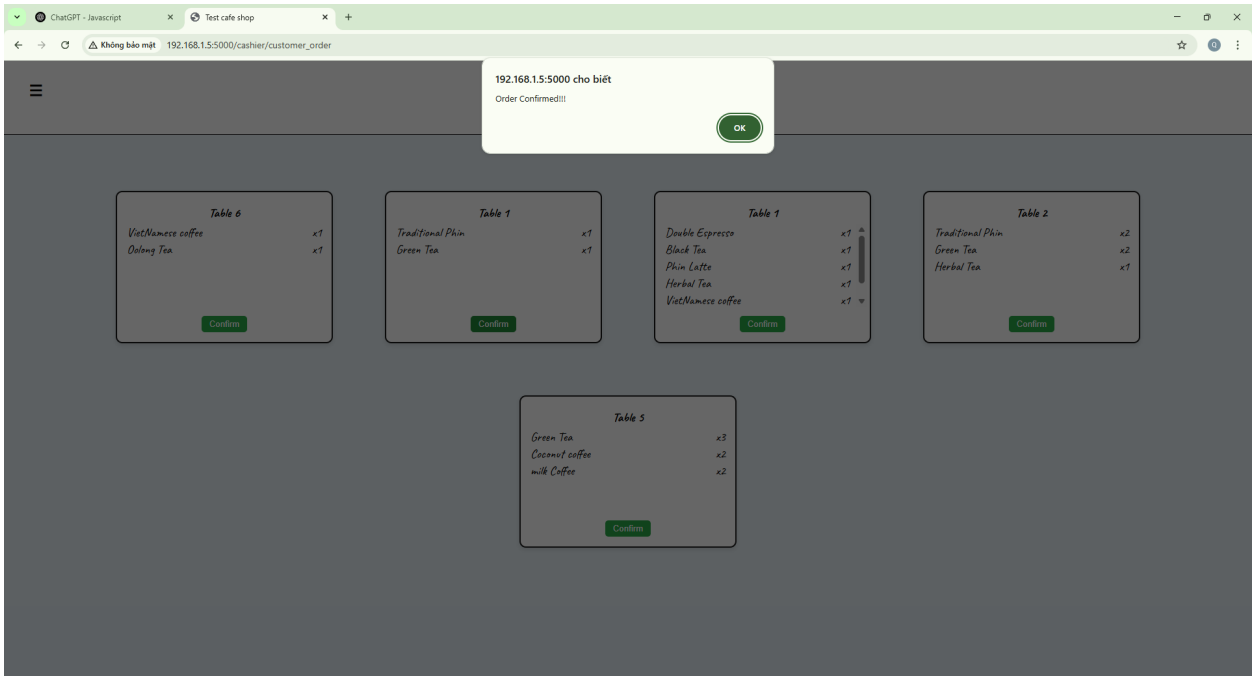


Figure 69: Customer Order

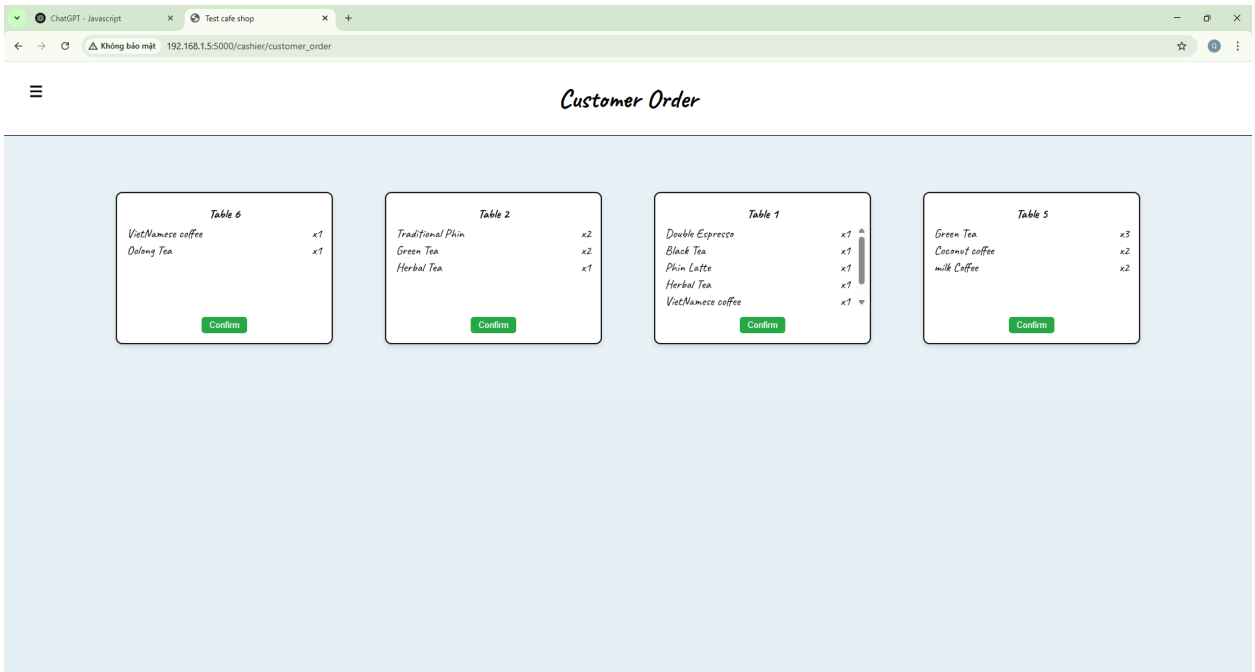


Figure 70: Customer Order (2)

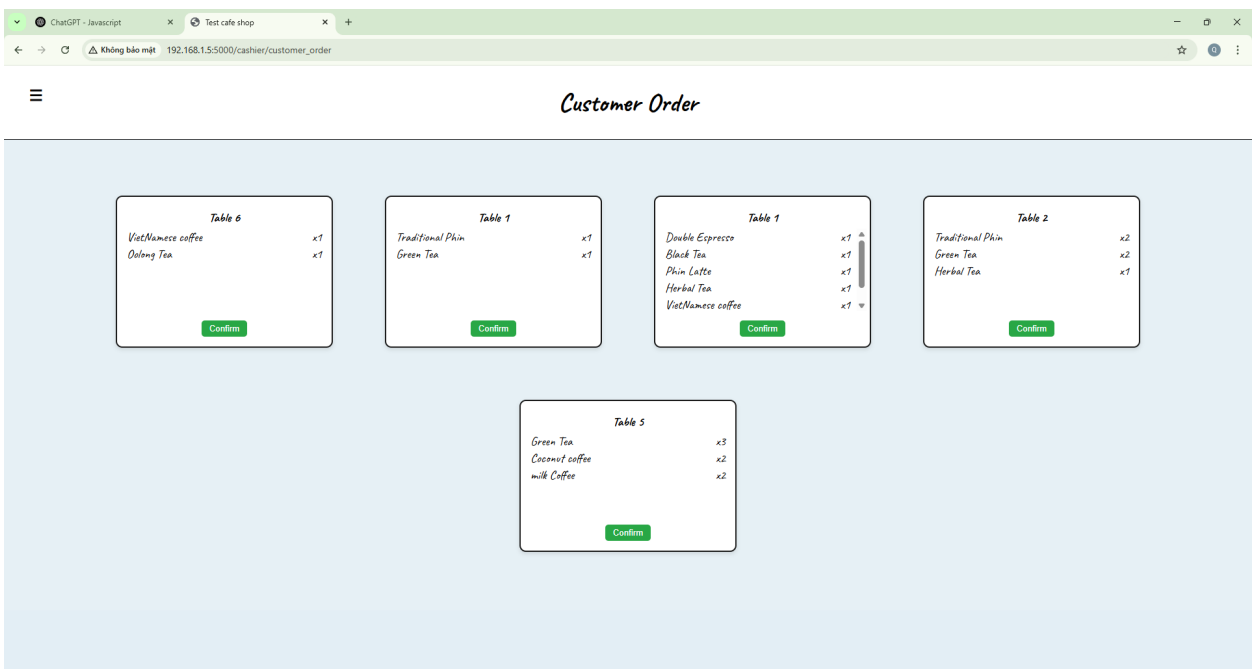


Figure 71: Customer Order (3)

5.3 Inventory UI

The Inventory UI (Storage Manager) allows for tracking stock, importing/exporting ingredients, and reviewing inventory history. It is designed for accuracy and efficiency in stock management.

Troy University Resources

5.3.1 Dashboard Menu

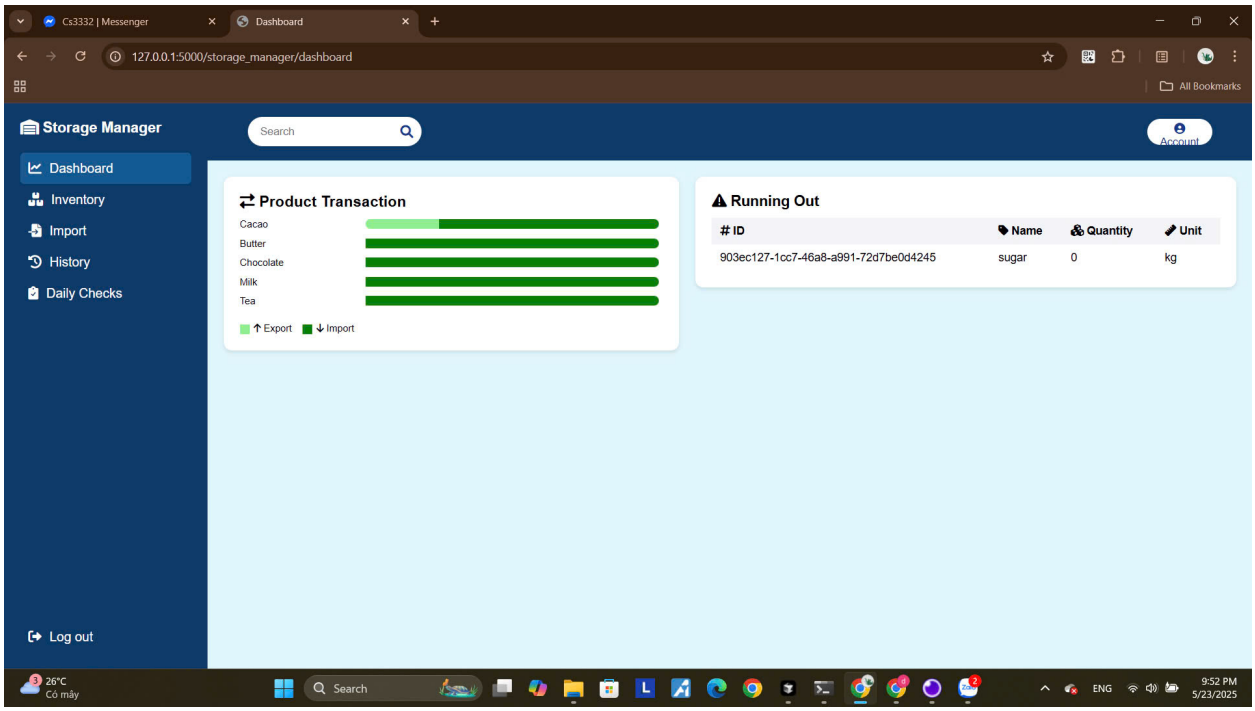


Figure 72: Inventory Dashboard

5.3.2 Inventory Menu

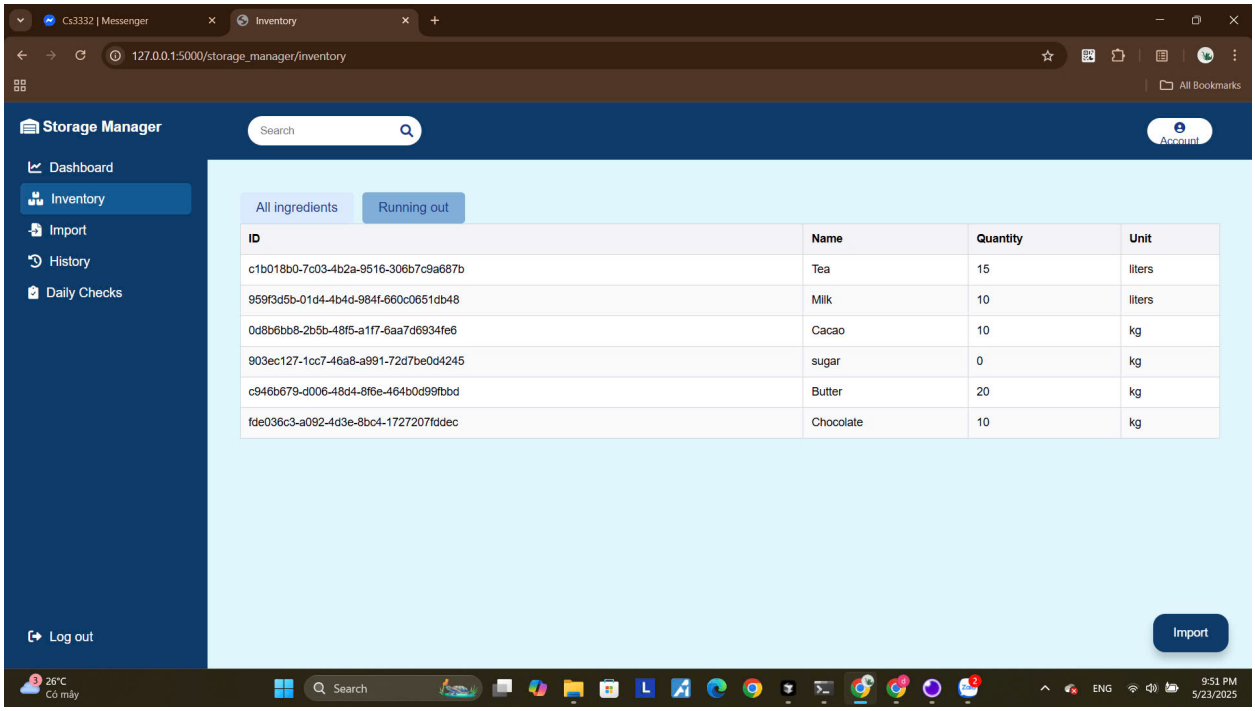


Figure 73: View Inventory

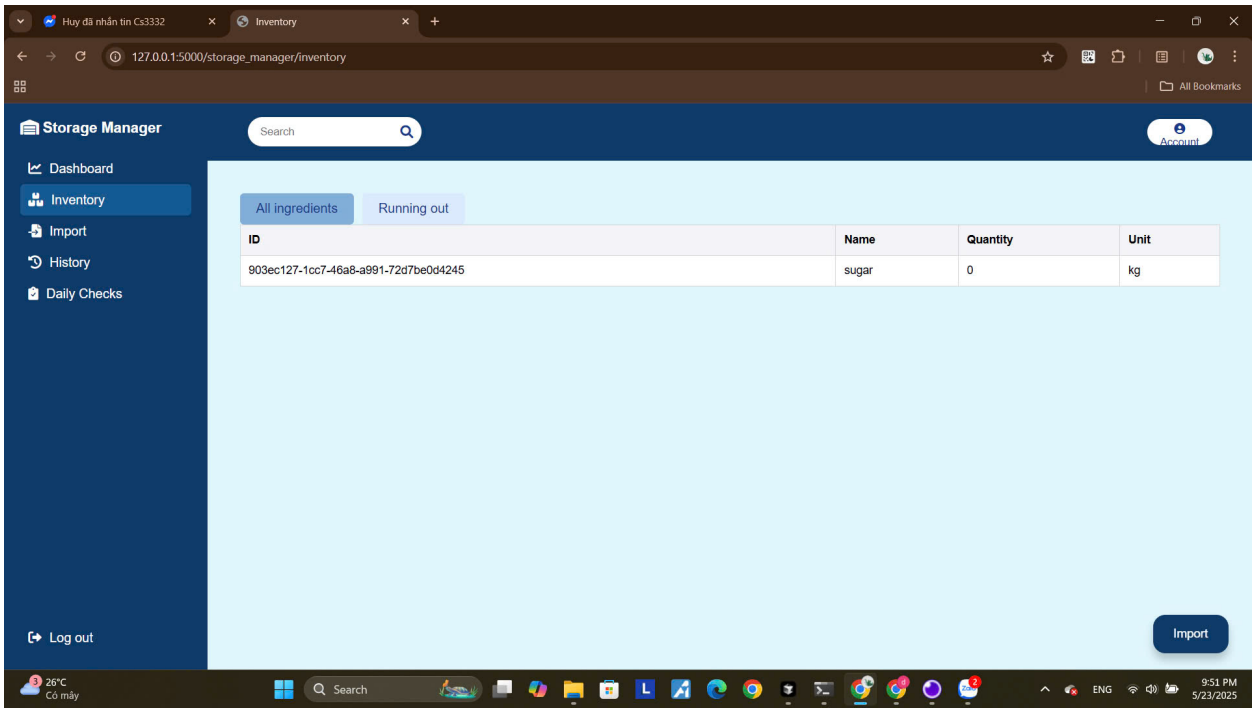


Figure 74: View Running Out Inventory

5.3.3 Import Menu

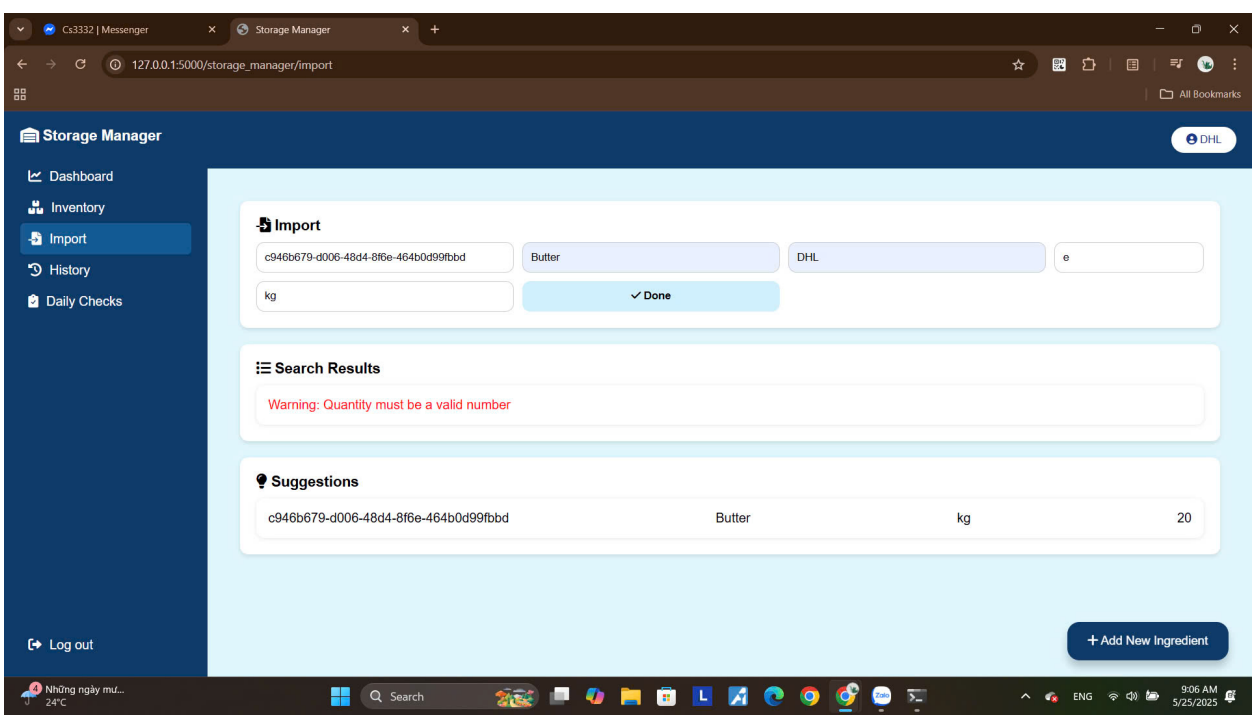


Figure 75: Import Error: Invalid Quantity

Troy University Resources

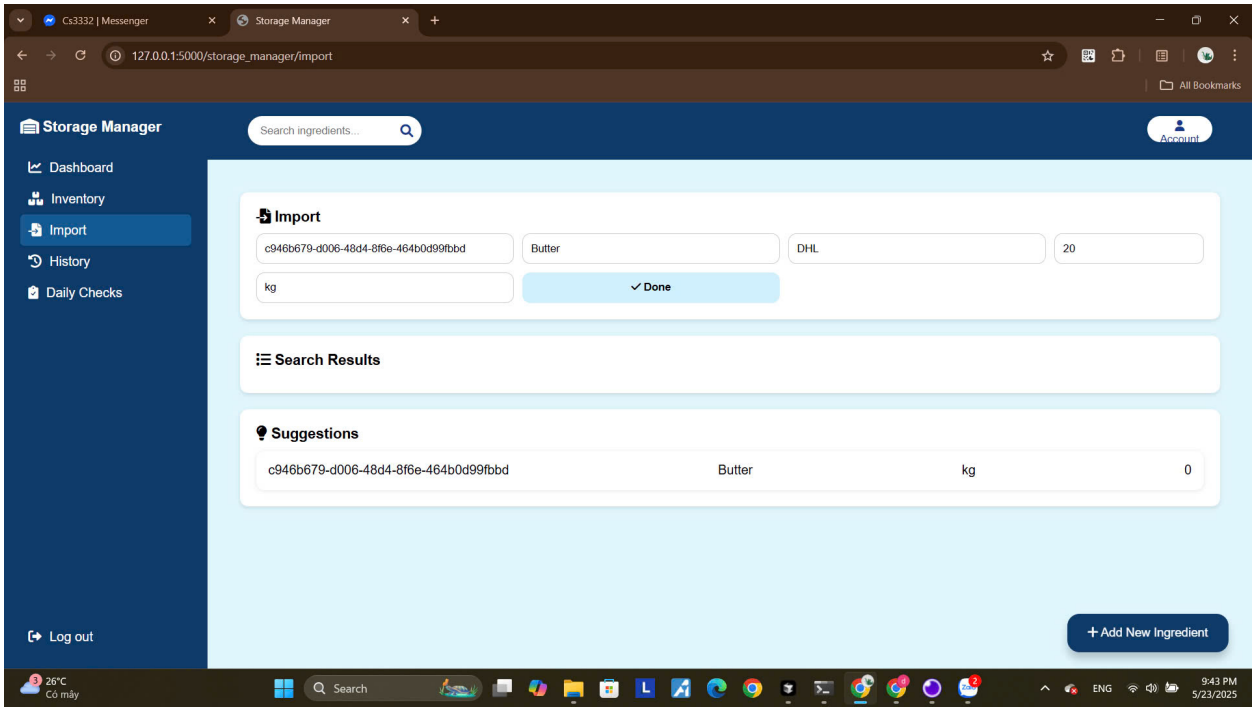


Figure 76: Import: Input Supplier and Quantity

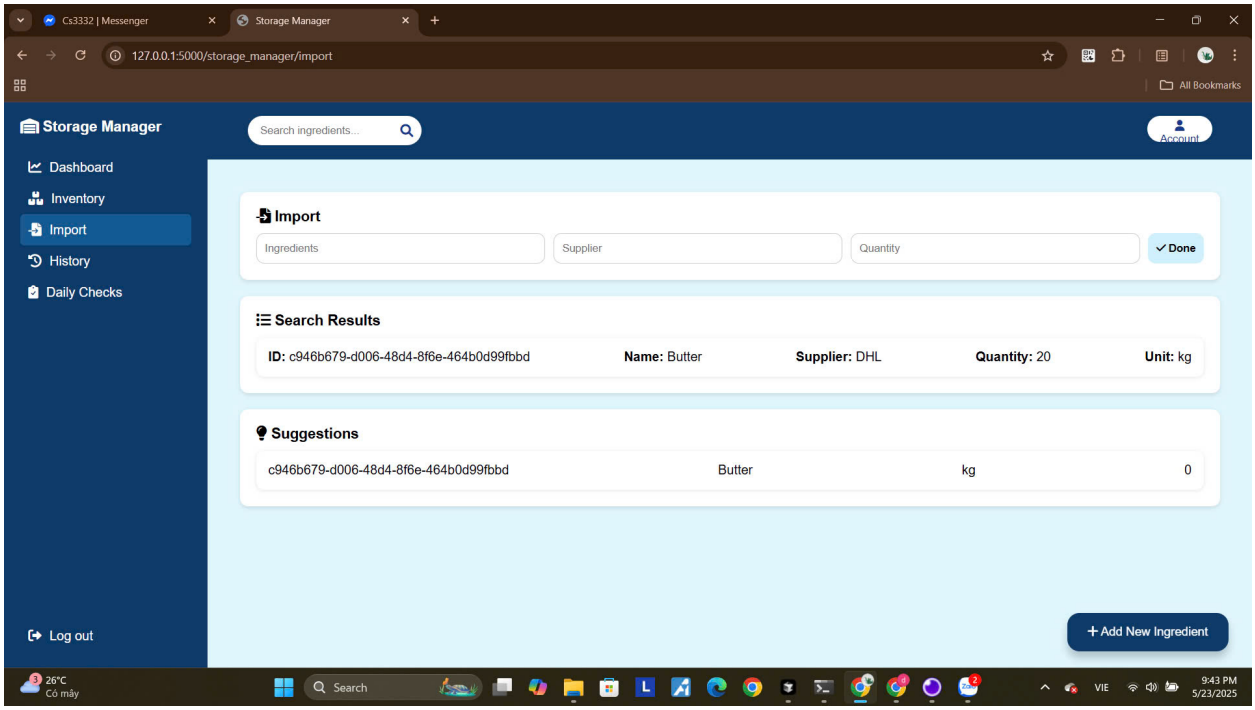


Figure 77: Import Ingredient Success

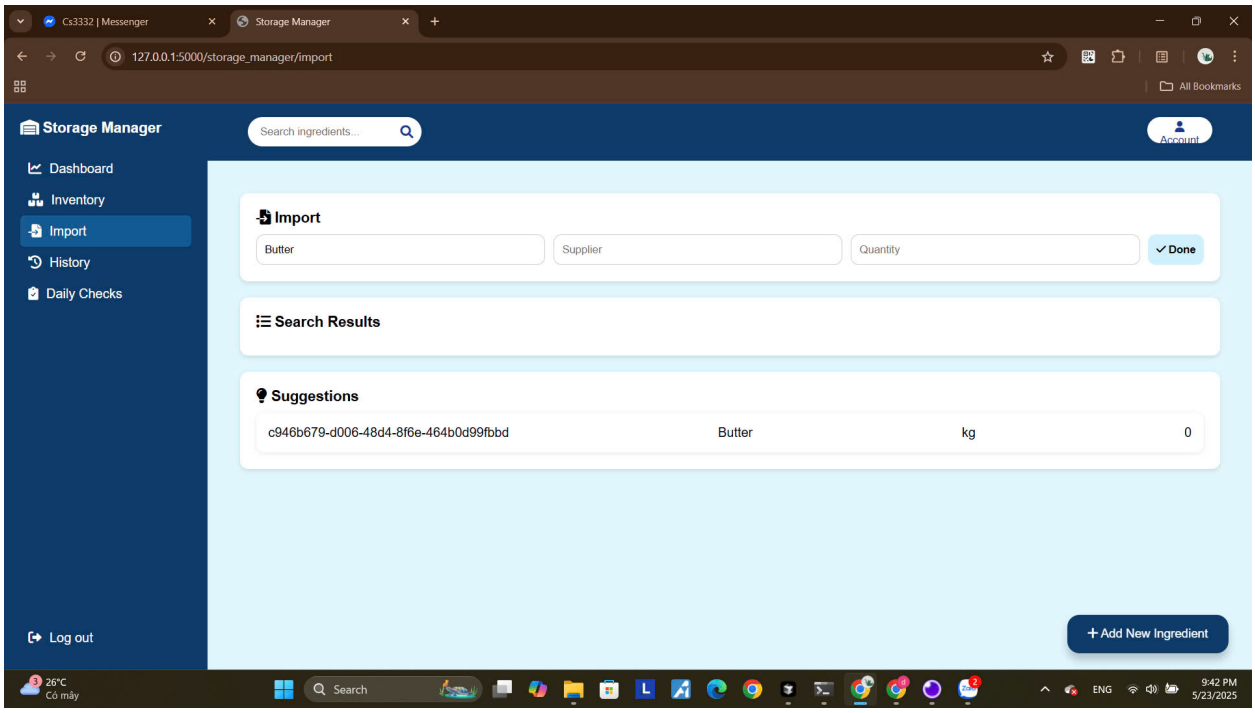


Figure 78: Import: Choose Suggestions to Import

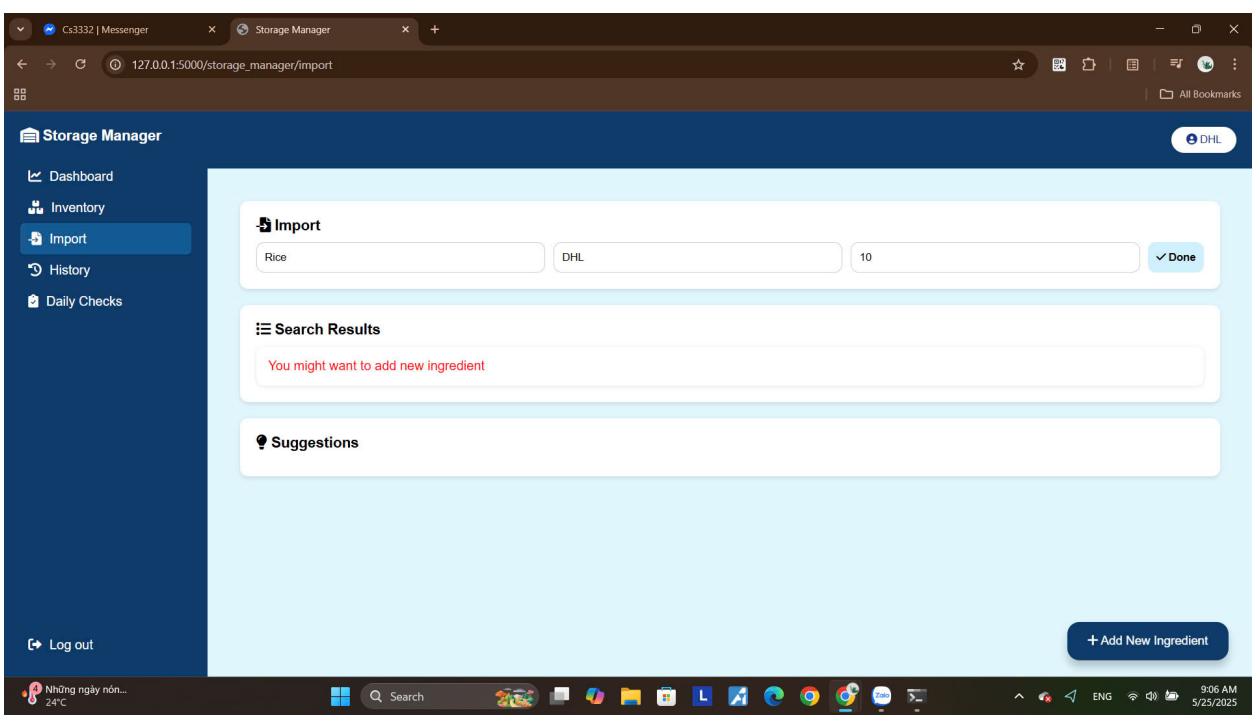


Figure 79: Import Error: Unavailable Ingredient

Troy University Resources

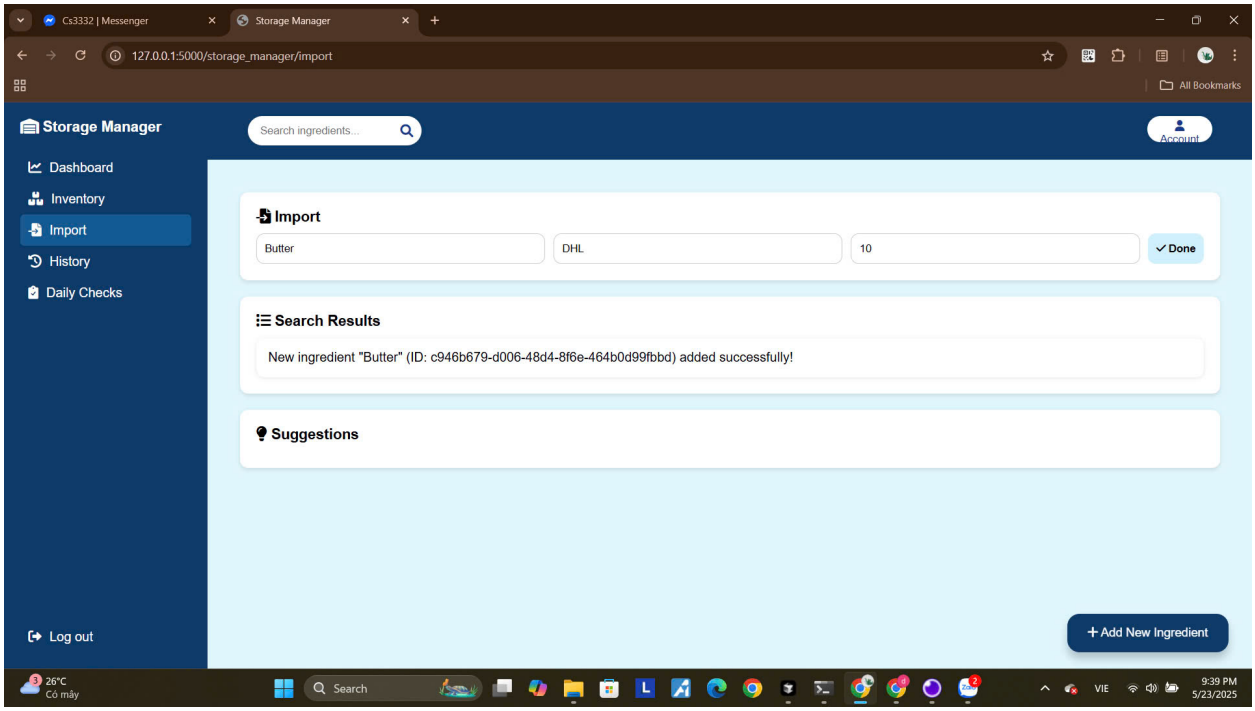


Figure 80: Import: Added Successfully

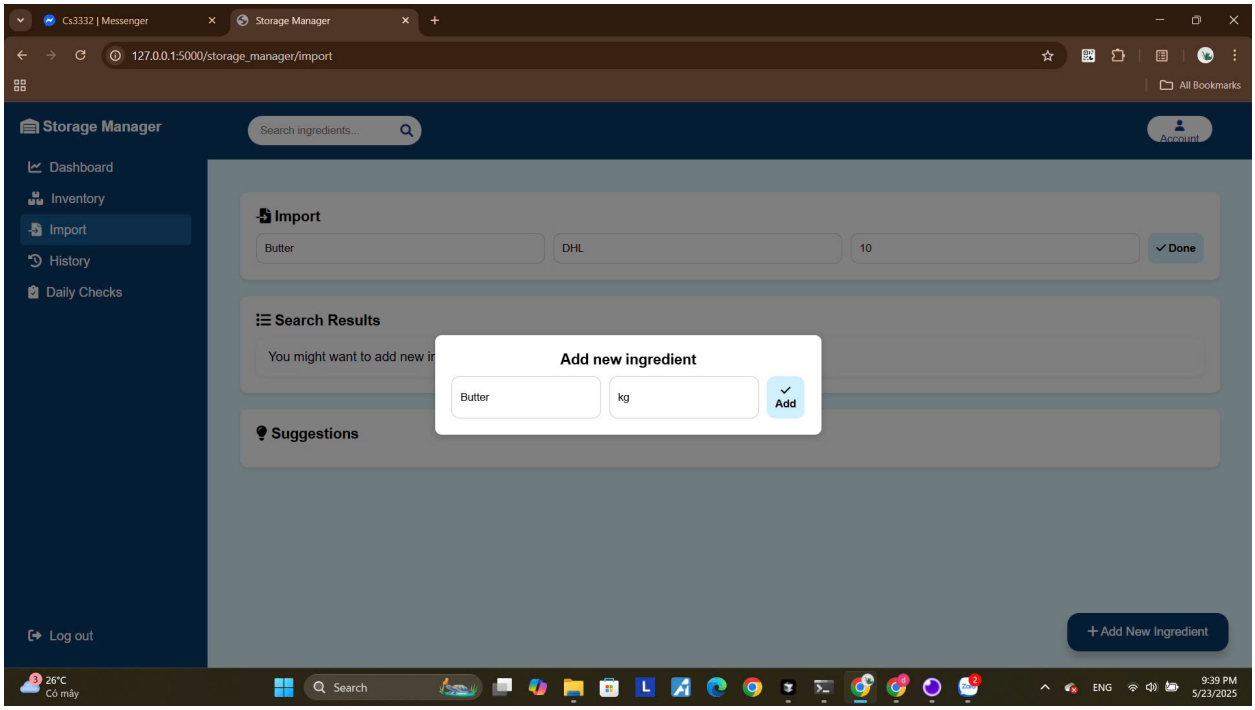


Figure 81: Import: Add New Ingredient

5.3.4 History Menu

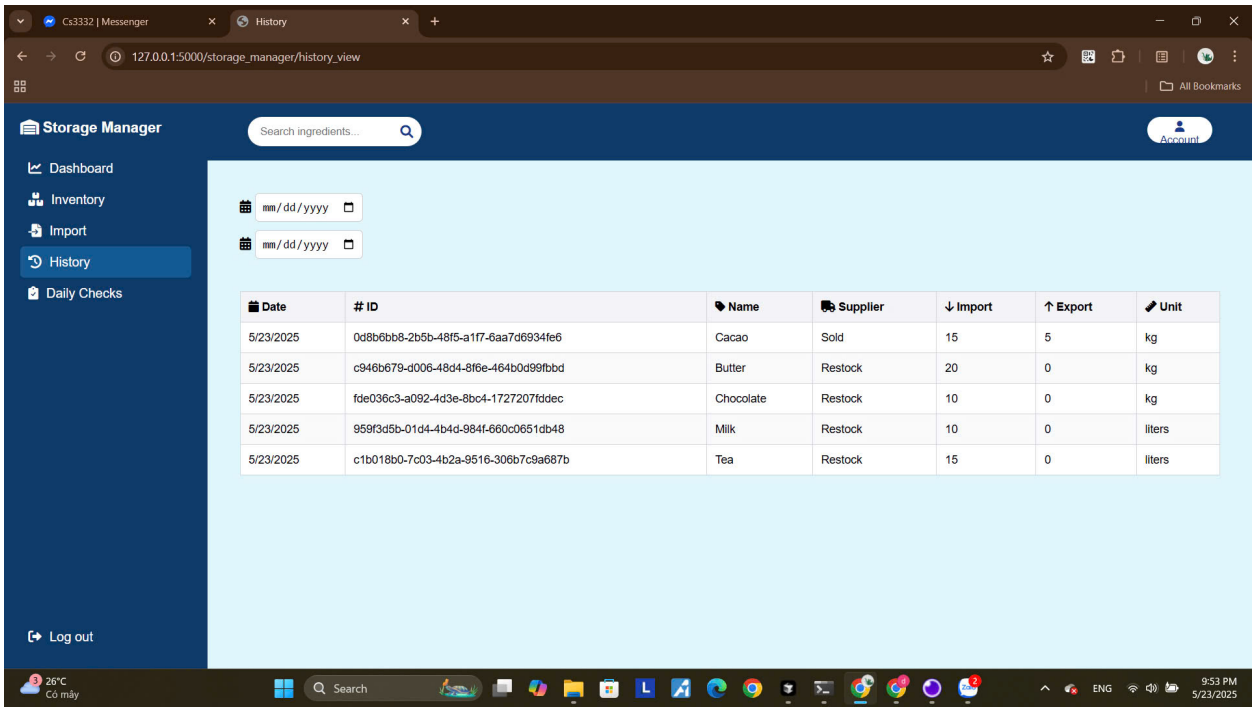


Figure 82: Inventory History

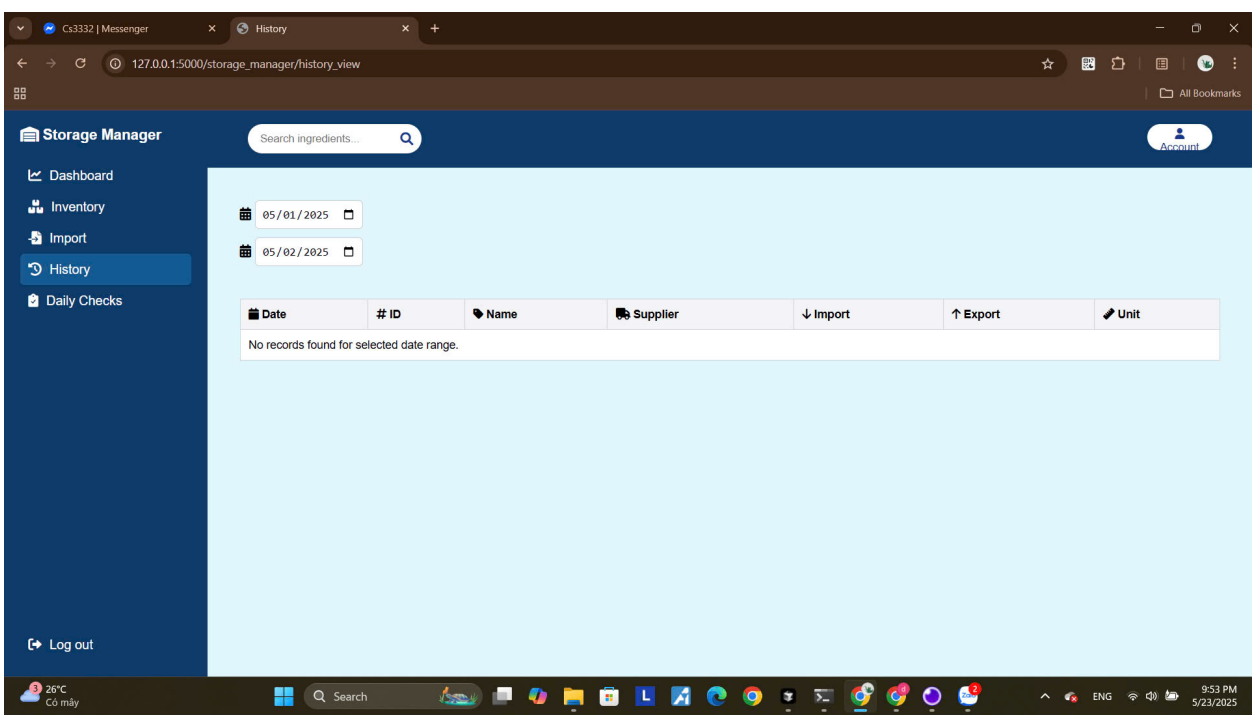


Figure 83: Inventory History Error: Date Unavailable

Troy University Resources

5.3.5 Daily Checks Menu

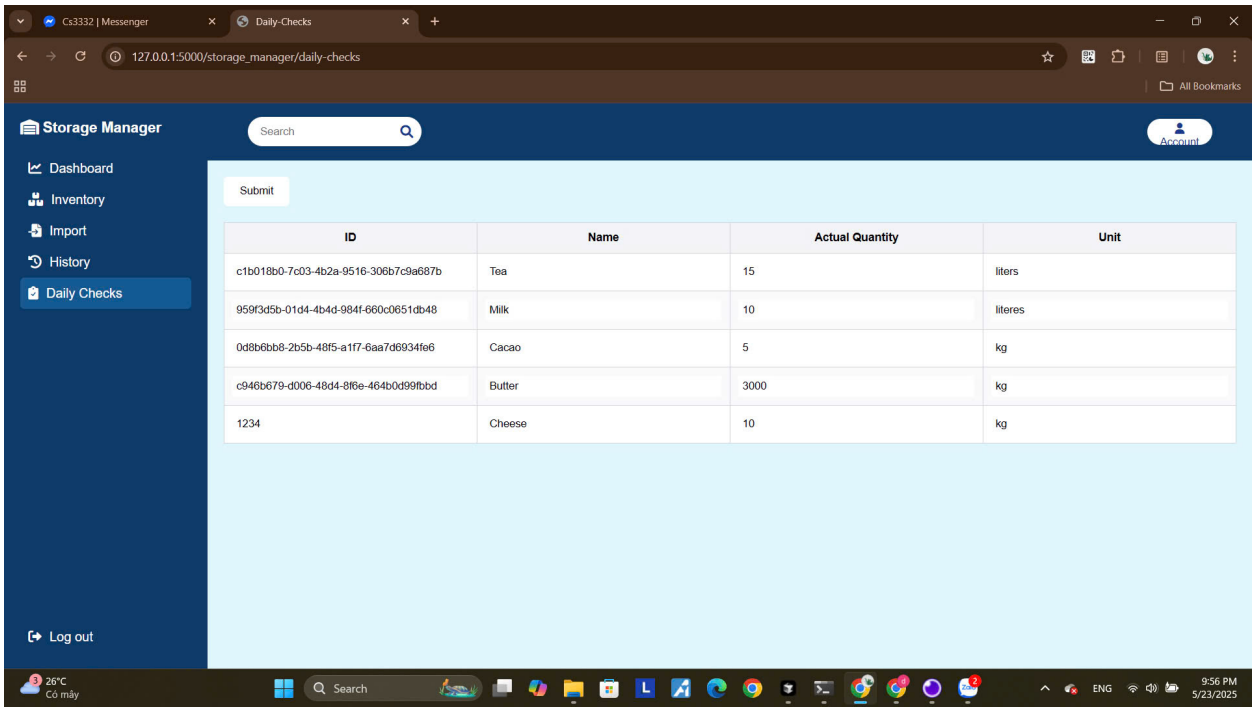


Figure 84: Daily Checks: Input

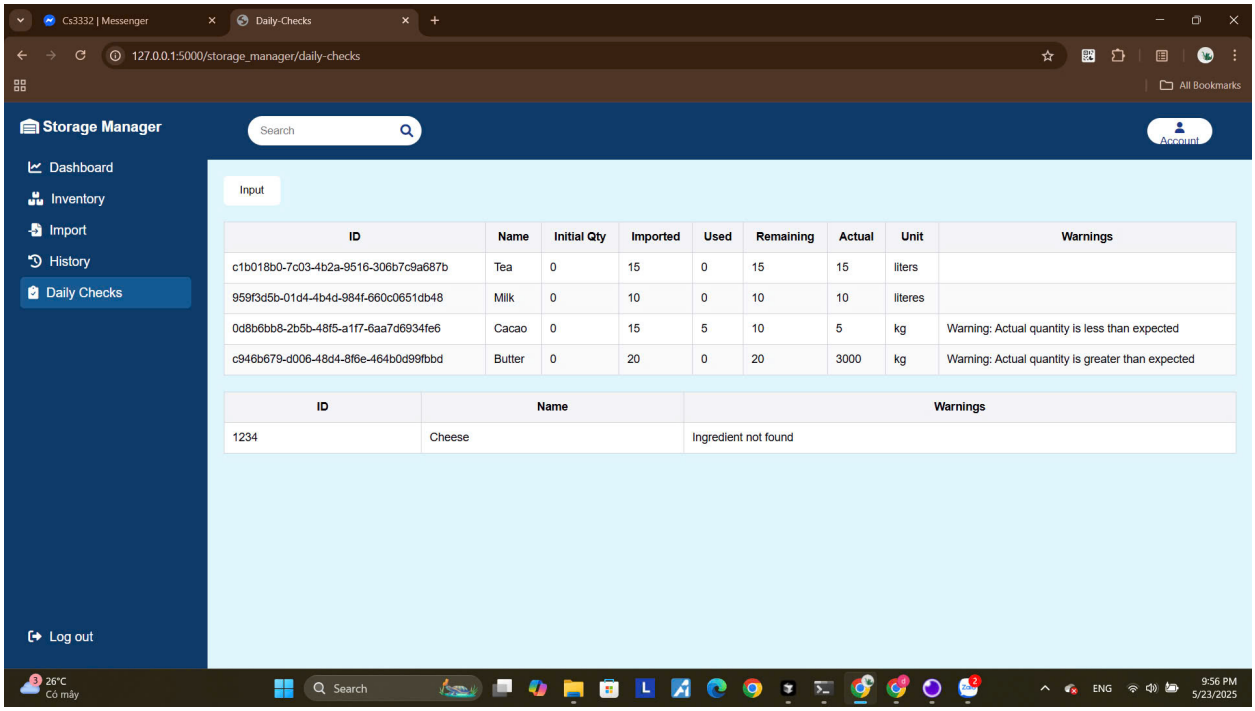


Figure 85: Daily Checks: After Checks

5.4 Bar UI

The Bar UI is tailored for bartenders to quickly view, prepare, and update beverage orders. It emphasizes speed and clarity for high-volume service environments.



Figure 86: Bartender Order Details



Figure 87: Bartender Order Completion

Troy University Resources

6 Tools and Workflow

This section describes the tools and workflow used in the Cafeteria Management System project, including the system architecture, CI/CD process, API testing, and collaborative development practices.

6.1 System Architecture Overview

The Cafeteria Management System is designed as a modular, scalable web application, leveraging microservices on Azure. The architecture consists of:

- **Frontend:** A web-based user interface for all user roles, maintained in the repository: `https://github.com/Chocon2911/Cs3332`
- **Backend:** A RESTful API server, maintained in the repository: `https://github.com/[STUDENT_1]/CafeteriaManagerSystem-Backend`
- **Database:** A various kind of database (e.g., Supabase, File System) for persistent storage.
- **Azure Microservices:** The backend is deployed as Azure microservices, allowing for independent scaling, robust logging, and modular development.
- **GitHub Actions and Runners:** Automated CI/CD workflows are managed using GitHub Actions, with runners executing build, test, and deployment jobs for both frontend and backend.
- **Shared Logging and Monitoring:** Azure provides centralized logging and monitoring which is bridged to the whole team by github actions, enabling the team to access logs and diagnostics collaboratively.

6.2 Collaborative Development and API Testing

- **Version Control:** All code is managed in GitHub repositories, with feature branches, pull requests, and code reviews to ensure quality and collaboration.
- **API Testing:** The team uses **Insomnia** as the primary API testing tool. Insomnia collections are shared among developers to test endpoints, validate responses, and automate regression testing.

- **Documentation:** API documentation and workflow guides are maintained in the repositories for easy onboarding and reference.
- Use branch protection rules and require status checks before merging.
- Store sensitive information (API keys, connection strings) in GitHub Secrets and Azure Key Vault.
- Automate tests and code quality checks to catch issues early.
- Use infrastructure-as-code (e.g., Azure Resource Manager templates) for reproducible deployments.
- Share Insomnia API collections and keep them up to date with backend changes.

6.3 CI/CD Workflow with GitHub Actions and Azure

The CI/CD pipeline is designed for automation, reliability, and transparency:

1. **Source Code Management:** Code is pushed to the respective GitHub repositories. Branch protection and required status checks are enforced.
2. **Continuous Integration (CI):**
    - On every push or pull request, GitHub Actions triggers workflows to build, lint, and test the application (both frontend and backend).
    - API endpoints are automatically tested using Insomnia or similar tools as part of the pipeline.
    - If all checks pass, the code is eligible for deployment.
3. **Continuous Deployment (CD):**
    - On merging to the main branch, GitHub Actions triggers deployment workflows.
    - The backend is built and deployed to Azure as microservices, while the frontend is deployed to Azure App Service or static hosting.
    - Docker images may be built and pushed to Azure Container Registry (ACR) for microservice deployment.
    - Deployment credentials and secrets are managed with GitHub Secrets and Azure Key Vault.
    - Azure's logging and monitoring tools provide real-time access to logs and diagnostics for the whole team.

Troy University Resources

4. **Auto-Build and Shared Logs:**
    - Every deployment triggers an auto-build and update of the running services.
    - Logs from all microservices are aggregated in Azure, allowing developers to collaboratively debug and monitor the system.

6.4 Backend Automation and Release Workflows

The following are the actual GitHub Actions workflow files used for backend automation, deployment, and release management. These workflows demonstrate how the backend is run with Ngrok for public access and how releases are built and published automatically.

```
name: Run CafeteriaManagerSystem Backend with Ngrok

on:
  workflow_dispatch:  # Manual trigger
  release:
    types: [published]  # Auto-run on release

jobs:
  run-backend:
    runs-on: self-hosted
    timeout-minutes: 86400  # Safety margin over 120 mins

    env:
      NGROK_AUTHTOKEN: ${{ secrets.NGROK_AUTHTOKEN }}
      NGROK_DOMAIN: ${{ secrets.NGROK_DOMAIN }}

    steps:
      - name: Create working directory
        run: mkdir -p app

      - name: Download JAR and JSON files
        working-directory: ./app
        run: |
          curl -L -o CafeteriaManagerSystem-Backend-1.0-SNAPSHOT.jar url
          curl -L -o data.json url
          download/Data/data.json
          curl -L -o production-data.json url

      - name: Start Backend JAR
        working-directory: ./app
        run: |
          nohup java -Xmx512M -Xms256M -jar .jar > backend.log 2>&1 &

      - name: Wait for server to start
        run: sleep 2

      - name: Install and configure ngrok
        run: |
          # curl -s   | sudo te
          # echo "deb  main" | sudo three
          # sudo apt update && sudo apt install ngrok -y
```

```
          # ngrok config add-authtoken "$NGROK_AUTHTOKEN"
          nohup ngrok http --domain="$NGROK_DOMAIN" 8080 > ngrok.log 2>&1 &

      - name: Wait for ngrok tunnel
        run: sleep 10

      - name: POST /test through ngrok public URL
        run: |
          echo "Sending POST request to https://${NGROK_DOMAIN}/test"
          RESPONSE=$(curl -s -o response.txt=
            -H "Content-Type: application/json" \
            -d '{"fileText":"your text"}')

          echo "Response Code: $RESPONSE"
          echo "Response Body:"
          cat response.txt

          if [ "$RESPONSE" -ne 200 ]; then
            echo "POST request failed with status $RESPONSE"
            exit 1
          fi

      - name: Stream backend logs live for 24 hours (86400 seconds)
        run: |
          echo "Streaming backend logs live for 24 hours. Press Ctrl+C to stop."
          timeout 86400 tail -f app/backend.log || true
```

```
name: Build and Publish Release

on:
  # push:
  #   branches: ["main"]
  pull_request:
    branches: ["main"]

permissions:
  contents: write
  security-events: write
  id-token: write

jobs:
  build:
    runs-on: ubuntu-latest
    # runs-on: self-hosted
    steps:
      - uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven

      - name: Build with Maven
        run: mvn -B package --file pom.xml
```


Troy University Resources

```
      - name: List target contents
        run: ls -la target

      # - name: Update dependency graph
      #   uses: advanced-security/maven-dependency-submission-action@v4

      - name: Create or Update Tag
        uses: jimeh/update-tags-action@v1
        with:
          tags: |
            v1.0.0
          when_exists: update
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

      - name: Create Release
        id: create_release
        uses: softprops/action-gh-release@v1
        with:
          tag_name: v1.0.0
          release_name: "Release 1.0.0"
          body: "Release of CafeteriaManagerSystem-Backend-1.0-SNAPSHOT"
          draft: false
          prerelease: false
          files: |
            target/*.jar
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

7 Conclusion

The Cafeteria Management System project represents a comprehensive effort to modernize and streamline cafeteria operations through the application of advanced software engineering principles. Over the course of the project, the team achieved several significant milestones. The system was designed with a clear separation of concerns, dividing functionality across frontend, backend, and database layers. This modularity has made the system easier to maintain, extend, and debug. By implementing automated build, test, and deployment workflows using GitHub Actions and Azure, the team ensured rapid delivery of new features and bug fixes, minimized manual errors, and maintained high code quality throughout development. Leveraging tools such as GitHub for version control and Insomnia for API testing fostered a collaborative environment where code reviews, shared API collections, and transparent logging improved both productivity and learning. The user interfaces for managers, cashiers, bartenders, and inventory staff were designed with usability in mind, resulting in intuitive workflows and clear feedback for all user roles. Automated and manual API testing ensured that endpoints were reliable, secure, and well-documented, reducing the risk of integration issues and improving system robustness. Hosting the system on Azure provided scalability, centralized logging, and real-time monitoring, which enabled the team to quickly identify and resolve issues in production. Comprehensive documentation of system design, workflows, and deployment processes has made onboarding new team members easier and supports future maintenance.

Despite these achievements, the project also encountered several limitations and areas for future improvement. While the current architecture supports moderate scaling, further work is needed to optimize performance under heavy load and to support distributed deployments across multiple regions. Some advanced features, such as predictive analytics for inventory management, real-time customer feedback, and integration with external payment gateways, remain as future enhancements. Although initial user testing was positive, broader deployment and more extensive feedback are required to refine the user experience and address edge cases. While best practices for security were followed, a formal security audit and compliance review (for example, regarding data privacy regulations) would further strengthen the system. The CI/CD pipeline could be enhanced with more sophisticated rollback and disaster recovery mechanisms to minimize downtime in case of deployment failures. Additionally, cloud resource

Troy University Resources

usage and cost optimization were not the primary focus during development and should be addressed as the system scales.

In summary, the Cafeteria Management System project has provided the team with invaluable experience in full-stack development, cloud computing, and DevOps practices. The achievements to date lay a strong foundation for future growth, while the identified limitations offer clear directions for ongoing improvement. By continuing to iterate on the system, incorporate user feedback, and adopt emerging technologies, the project can evolve to meet the changing needs of modern cafeteria management and serve as a model for similar digital transformation initiatives.

8 References

- Deitel, P. J., & Deitel, H. M. (2012). *Java: How to program* (9th ed.). Pearson.
- Pressman, R. S. (2010). *Software engineering: A practitioner's approach* (7th ed.). McGraw-Hill.
- Sommerville, I. (2011). *Software engineering* (8th ed.). Pearson.
- Sommerville, I. (2011). *Software engineering* (9th ed.). Pearson.
- Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: The hardware/software interface* (5th ed.). Elsevier.
- Stallings, W. (2015). *Computer organization and architecture: Designing for performance* (10th ed.). Pearson.

Troy University Resources

9 Contribution

Team Contributions			
Member	**ID**	**Main Responsibilities**	**Contributions**
[STUDENT_1]	[ID_1]	Designed and implemented backend architecture, APIs, and database. Managed backend integration, deployment, and project documentation.	35%
[STUDENT_2]	[ID_2]	Developed and tested the Storage Manager UI, focusing on inventory features and integration. Contributed to project documentation.	20%
[STUDENT_4]	[ID_4]	Designed and developed Manager and Bartender UIs, ensuring usability and integration with backend. Supported documentation.	20%
[STUDENT_3]	[ID_3]	Built and tested Cashier and Customer UIs, focusing on order and billing workflows. Assisted with user testing and documentation.	20%
[STUDENT_5]	[ID_5]	Wrote project reports and documentation for Bartender UI.	5%