Table of Contents:

Introduction……………………………………………………………………. **1**

Team Information ………………………………………………………………**2-3**

Use Case Diagram………………………………………………………………**4**

Use Case Description……………………………………………………………**5-13**

Transaction……………………………………………………………………

Manager………………………………………………………………………

Inventory………………………………………………………………………

Item……………………………………………………………………………

Costumer………………………………………………………………………

Cashier…………………………………………………………………………

Class Diagram……………………………………………………………...**14-16**

Class Description……………………………………………………………**17-29**

Communication Diagram……………………………………………………**30-44**

Transaction……………………………………………………………………

Manager………………………………………………………………………

Inventory………………………………………………………………………

Item……………………………………………………………………………

Costumer………………………………………………………………………

Cashier…………………………………………………………………………

C++ Implementation…………………………………………………………**45-49**

Conclusion……………………………………………………………………**50**

Introduction:

For the class, CS 3500, Software Engineering class, we did a system design for a POS system.
We designed our POS based on the criteria of the given blueprint. We tried our best to build a robust and user-friendly POS system for a grocery store where there is a 5 pos hardware and all of the transactions are done by either a cashier or a manager, and there is no provision for self-checkout. All of the functional tasks, such as handling sales, scanning items, applying discounts and applying coupons, accepting 4 types of payment and printing the receipts, along with the inventory management functions such as tracking the sales and the employee performance. The whole project is about designing a POS system for a grocery store to replace their manual cash register. Overall, the project stimulates a real-world software development experience by creating a complete and functional system.

Our aspect was to build a fully functional POS software, meeting all of the criteria set by Professor Dr. Cyril Ku. To simplify the design of POS, we divided it into six components: Transaction, Cashier, Customer, Manager, Inventory and Item. The system interacts with a central database, supports authentication and covers a wide range of scenario sequences such as normal transaction, cancellation, error, payment processing, etc. There are 2 roles, cashier and manager. Cashiers have limited access where whereas managers can do administrative tasks such as overriding price, generating reports and managing inventory. Our system also supports the loyalty program by recognizing preferred customers through scanning loyalty cards or by entering the phone number. Each of the components is designed using the OOP concept to ensure modularity and clarity. We used UML diagrams to visually represent use cases. To get into the operation efficiently, we created a use case diagram and its description so engineers can understand easily, a communication diagram for the operation sequence and the sequence diagram to understand the step-by-step process. At last, all of the designs are implemented using C++, so they could be easily understood by the schemas.

In conclusion: This project allowed us to apply Software Engineering Principles in real real-life setting and demonstrated our ability to design and implement a software program.

Team Information:

We were a group of three: Rojan Upreti, Anjana Pun, and Nishan Joti. To build mutual understanding, we discussed project ideas in class and each worked on the assignment individually at home. We then combined our ideas and implemented the best one. Below are our individual experiences.

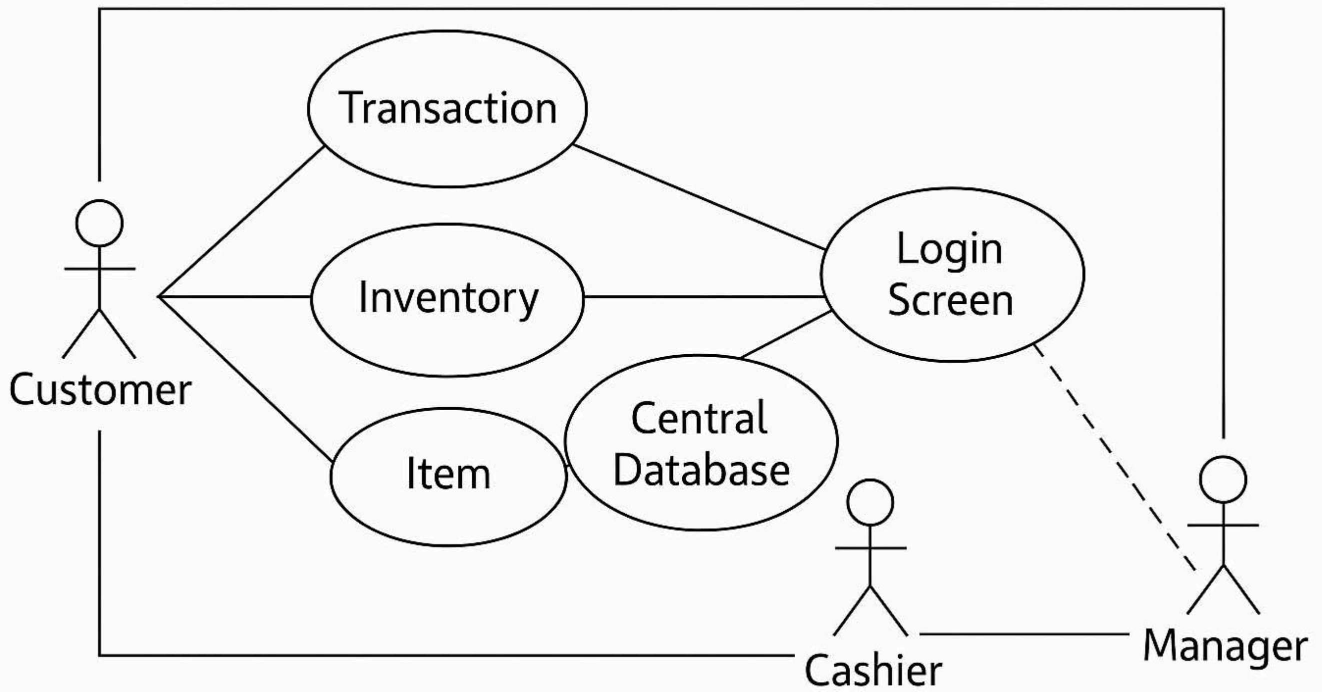
Nishan Joti's Experience: Pos System project proved to be a very fulfilling endeavor for me as a student in Computer Science study. I designed UML diagrams, transformed them into C++ classes, and made sure our implementation did reflect our anticipated architecture. This made me have a deeper understanding of object oriented programming including the part of the use of inheritance and encapsulation in the real world systems. While working on the project, I also developed my critical thinking of system design and system entity interaction. Working with my teammates was not a walk in the park as there were different challenges like conflict of schedules and differences in work style, but we overcame this because we were often communicating and we did it to the point of deciding that each of us does the things he or she is good at. Apart from improving my technical skills, this project taught me how to manage responsibilities in a team work environment. It provided me with a better understanding of what it is like to work on an actual software development team in the real world and has put me in a better position to undertake future academic and related initiatives.

Rojan Upreti's Experience: For me, it was a really good experience to build a POS system. From a long time, we have been brainwashed that software engineering is all about coding only. But after taking this class, learning about the software engineering principles, it broadened my skills, knowledge and way of thinking and understanding about software engineering. It was definitely a good experience. We started by creating a UML diagram, creating a use case, and implementing it. In my view, the whole project made me able to solve problems more quickly than before and helped me to sharpen my problem-solving skills for problem solving.

Anjana Pun's Experience:

Working on the Point-of-Sale (POS) system for the CS 3500 Software Engineering course was a meaningful and collaborative experience. Our team of three worked hard together and contributed in different ways based on our individual strengths. I was primarily focused on the design aspects of the project, including the Use Case diagrams, Use Case descriptions, and Class Diagrams. These tasks helped me understand how important a clear and well-thought-out structure is before beginning the coding phase. Nishan also contributed to the Use Case Diagram by helping identify key flows and system interactions. Rojan played a major role in the implementation phase by working on class attributes and writing functions in C++. We maintained good communication throughout the project and made sure to review each other's work to keep everything consistent and aligned with the initial requirements. This team effort allowed us to catch errors early and improve the quality of our final product. Through this project, I gained valuable experience in software design and development. I also learned the importance of teamwork, clear planning, and supporting one another. It was a great opportunity to apply what we've learned in class to a real-world scenario, and I look forward to building on these skills in future projects.

Use Case Diagram:



Use Case Description:

1. Transaction Use Case Diagram

Use Case Name: Transaction

Author: Rojan Upreti, Nishan Joti, Anjana Pun

Last Updated: May 7, 2025

Preconditions:

- A cashier should be logged into the system
- The customer should be present and ready to check out

Dialog:

– The cashier begins by selecting "Start Transaction" on the screen:- startTransaction()

– The transaction initializes with empty itemList, totalAmount = 0.0, and paymentMethod = "none".

– The cashier scans items one by one:- addItem(item)

– If the item is weight-based, the cashier enters the weight:- enterWeight(weight: float)

– If a wrong item is scanned, the cashier removes it:- removeItem(itemID: int)

– The cashier calculates the total:- calculateTotal()

– If the customer provides a coupon, it is entered and validated:- applyCoupon(code)

– The cashier asks for a payment method, and the system captures the choice:- requestPaymentMethod()

– Payment is processed:- finalizeTransaction(). If successful, the system continues. If not, the cashier may retry or cancel.

– A receipt is generated and printed:- generateReceipt()

– If the customer cancels the transaction at any point:- cancelTransaction()

Post Conditions:

-Whenever the complete transaction is done, then the whole process restarts from startTransaction()

-After each successful sale, the inventory gets updated automatically

-All the refunds for the purchased items are authorized only by the Manager.

2. Manager Use Case Diagram

Use Case Name: Manager

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Updated: May 8, 2025

Preconditions:

-The manager should be logged in with valid login credentials

-The POS system should be operational and should be connected to a central database

Dialog:

-To get started, the system uses the **logIn(username: string, password: string): boolean** method to check if the manager's username and password are correct. If they are, the system grants access to admin-level features.

-When a manager needs to find a specific product, they can use **searchItem(itemNameOrID: string)** to look it up by name or ID. This is useful for checking item info before changing prices or stock.

-If a customer returns asking for a refund, the manager can find the original transaction using **searchTransaction(transactionID: int)**. This allows them to review the details before processing any

return. Sometimes customers return items, and that's where **processRefund(transactionID: int)** comes in. It processes the return, updates the transaction log, and adjusts the inventory accordingly.

.

-To check on a cashier's activity, the manager uses **searchCashier(cashierID: int) and access trackCashierPerformance(cashierID: int)**, which brings up that cashier's profile and recent performance history.

-If a price needs to be changed, for example, during a promotion or error correction, the manager can call **approvePriceOverride(itemID: int, newPrice: float)** function. This updates the item's price in the system for future transactions.

-To see how the store is doing, the manager can generate a summary using **generateSalesReport(timePeriod: string)**. This report includes total sales and other useful business insights for any time range.

-The **generateInventoryReport()** method gives the manager an overview of all current stock levels. It's helpful for planning restocks and catching any missing or overstocked items.

Post Condition:

-Any changes made to item p[rice, refund or reports generated are daved in central database and reflect in all terminals

-Approved price overrides are immediately updated

-Any error during search or query will redirect them to the same page

3. Inventory Use Case

Use Case Name: Inventory

Author: Rojan Upreti, Nishan Joti, Anjana Pun

Last Updated: May 8, 2025

Precondition:

-The manager must be logged into the pos system

-The pos system must be operational and connected to the database

Dialog:

– The system begins by authenticating the manager with logIn(username: string, password: string). The user must be logged in using the manager's login credentials

– To find a specific item, the manager uses searchProduct(itemNameOrID: string), which searches the inventory by either product name or item ID.

– When new stock arrives, the manager adds it to the system using addProduct(product: Item). This creates a new entry in the inventory database.

– If a product is discontinued or entered by mistake, the manager removes it using removeProduct(itemID: int). This deletes the item from the active inventory list.

– When stock quantities change due to purchases, returns, or manual updates, the manager uses updateStockLevel(itemID: int, quantityChange: int) to adjust the available quantity in real time.

– To view how much of a product is available, the manager calls checkStockLevel(itemID: int), which returns the current quantity for that item.

– If any operation involves invalid input or an unavailable item ID, the system triggers error(), which will restart the program and prompt the user to the same screen again.

Post Condition:

-The inventory will be updated with any added, removed d or adjusted stock items

-All changes on the POS system are live and real-time.

-If the user fails the search query, then they will return to the same screen so they can retry

4. ITEM USE CASE

Name of Use Case: ITEM

Author: Rojan Upreti, Nishan Joti, Anjana Pun

Last Updated: May 08, 2025

Preconditions:

-The manager must log in to the POS system

-The item must already be registered/entered in the POS database

-The system should be connected to the central database.

Dialog:

– The manager logs into the POS system logIn(username: string, password: string). Once authenticated, they can access item-level operations.

– To check the current selling price of an item, the manager uses getPrice(), which retrieves the price from the central database.

– If the price seems incorrect or needs adjustment for a sale, the manager uses setPrice(price: float) to update it. This change is immediately reflected in future transactions.

– When an item is sold by weight (like produce or bakery items), the manager can use getWeight() to see the current weight assigned to that product.

– If needed, the manager can update the product's weight using setWeight(weight: float), ensuring accurate pricing based on actual weight.

– To determine whether tax should be applied, the manager uses isTaxable(), which returns a boolean indicating the tax status of the item.

– If any operation involves invalid input or an unavailable item ID, the system triggers error(), which will restart the program and prompt the user to the same screen again.

Post Condition:

-Any update made to an item's properties is saved and updated in real-time

-Tax status helps us to ensure an accurate total calculator

5. Customer USE CASE

Use case Name: Customer

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last updated: May 8, 2025

Precondition:

-The cashier or manager must be logged in to manage the customer

-The customer must be present at the checkout

-Optional: The customer might be pre-registered in the system for the loyalty member status.

Dialog:

– The system begins by authenticating the logged- cashier or manager logIn(username: string, password: string)

– To identify the customer, the vsdjoirt asks for a phone number.r

identifyCustomerByPhone(phoneNumber: string)

– Alternatively, the customer may provide a loyalty barcode, which is scanned :-

identifyCustomerByBarcode(barcode: string)

– Once identified, the system determines whether the customer is a regular or preferred member :-

getCustomerType()

– If the customer wishes to update their contact number, the manager can make changes :-

updatePhoneNumber(phone: string)

– For loyalty members, the system may apply a discount based on customer status and transaction total :-

applyLoyaltyDiscount(total: float)

– If the total purchase exceeds a certain amount (e.g., $500), the system checks eligibility for rewards such as a free turkey :- rewardTurkeyIfEligible(total: float)

– If invalid information is entered (e.g., incorrect phone number or fake barcode), or if the customer is not eligible for a loyalty benefit, the system will call:- error(), and the action will be rejected.

Post Condition:

- The customer status is identified, and the loyalty-based features are applied if eligible
- If the purchase is more than 500$, then the customer will get a coupon for a free turkey
- If the customer is not a preferred customer, the loyalty and coupon apply get skipped

6. Cashier Use Case Diagram

Use Case Name: Cashier

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Updated: May 8, 2025

Preconditions:

-The cashier must be logged into POS system with valid lpogin credentials

-The costumer must be present at checkout with items to be purchased

-The POS system must be connected to the central database

Dialog:

-The cashier starts by logging into the POS system using their username and password through the logIn() function. Once logged in, they begin the checkout process by clicking “Start Transaction” which calls startTransaction().

-As the customer gives their items, the cashier scans each one using addItem(itemID). The system checks the item information in the database. If the item is sold by weight (like fruits or meat), the cashier enters the weight using enterWeight(weight).

-The system then checks whether the item needs tax or not by using isTaxable(). If the customer is part of a loyalty program, the cashier either types in their phone number using enterPhoneNumber() or scans their loyalty card with scanLoyaltyCard(). The system finds out if the customer is preferred or regular using getCustomerType().

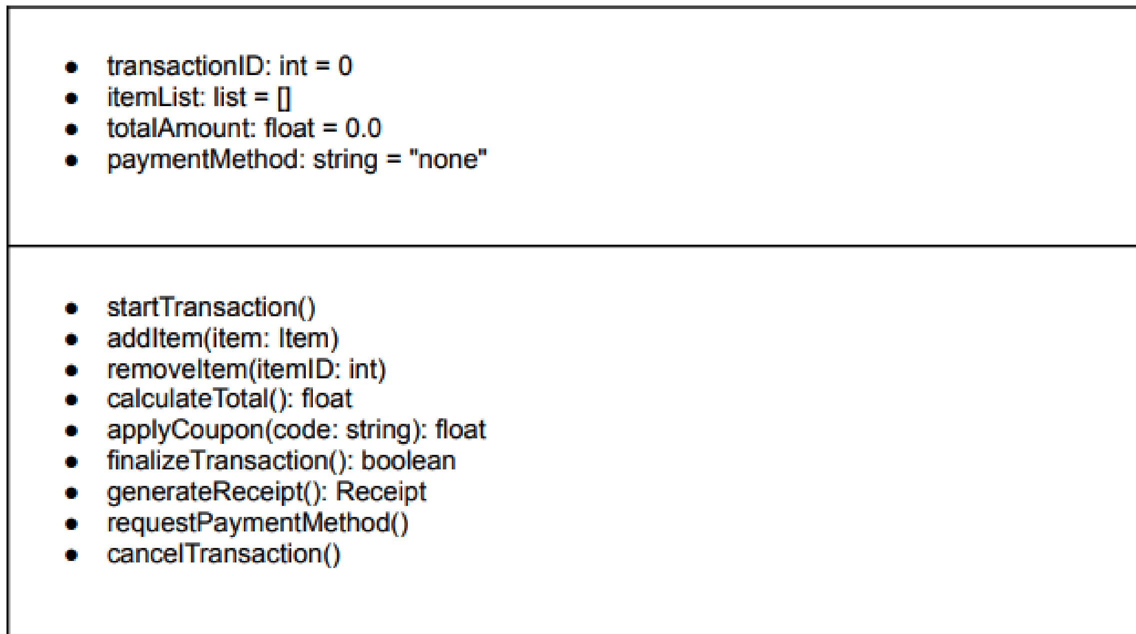
-If the customer has a coupon, the cashier enters it with applyCoupon(). After all items are scanned and any discounts or coupons are added, the cashier calculates the total using calculateTotal(). If the total amount is over $500, the system checks if the customer should get a free turkey or other reward using rewardTurkeyIfEligible().

-Then the cashier asks how the customer wants to pay and selects it using requestPaymentMethod(). The customer chooses a method (like cash, card, or check) which is confirmed by selectPaymentMethod(). The cashier processes the payment using processPayment(). If the payment works, the system finishes the transaction with finalizeTransaction().

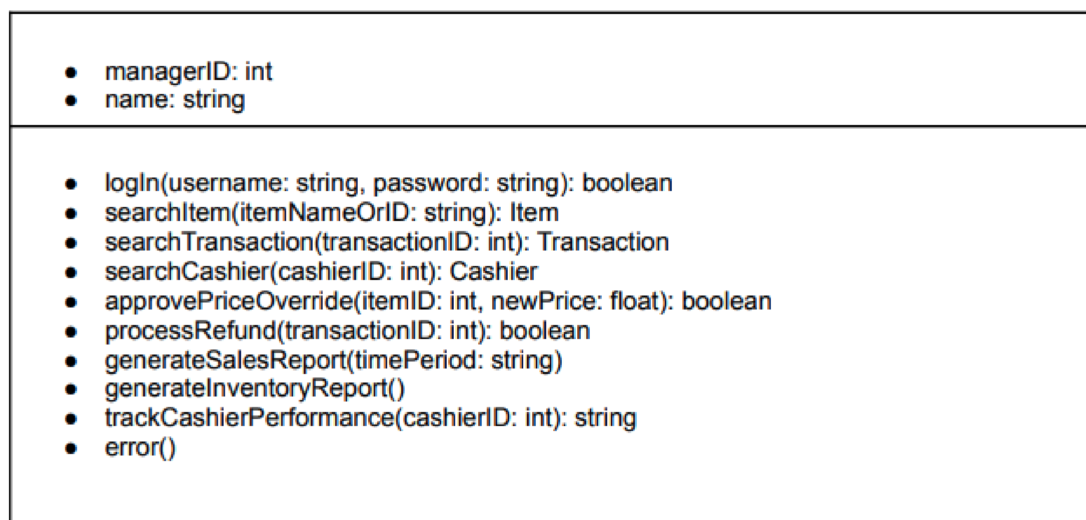
-After the payment, the system creates a receipt using generateReceipt() and prints it with printReceipt(). If something goes wrong, like a wrong item, failed payment, or invalid input, the system shows an error message using error(). If the customer changes their mind or wants to remove an item, the cashier can take it out with removeItem() or cancel everything with cancelTransaction().

Class Diagram

Transaction:



Manager:



Inventory

<table>
<tr><td></td></tr>
<tr><td>

- logIn(username: string, password: string): boolean
- searchProduct(itemNameOrID: string): Item
- addProduct(product: Item): boolean
- removeProduct(itemID: int): boolean
- updateStockLevel(itemID: int, quantityChange: int): boolean
- checkStockLevel(itemID: int): int
- error()

</td></tr>
</table>

Item

<table>
<tr><td>

- itemID: int = 0
- name: string = ""
- price: float = 0.0
- weight: float = 0.0
- taxable: boolean = true
- quantity: int = 1

</td></tr>
<tr><td>

- logIn(username: string, password: string): boolean
- getPrice(): float
- setPrice(price: float): void
- getWeight(): float
- setWeight(weight: float): void
- isTaxable(): boolean
- error()

</td></tr>
</table>

Costumer

• customerID: int • name: string • phoneNumber: string • loyaltyStatus: string = "non-preferred"
• logIn(username: string, password: string): boolean • identifyCustomerByPhone(phoneNumber: string): Customer • identifyCustomerByBarcode(barcode: string): Customer • getCustomerType(): string • updatePhoneNumber(phone: string): void • applyLoyaltyDiscount(total: float): float • rewardTurkeyIfEligible(total: float): boolean • error()

Cashier

• cashierID: int = 0 • usernamename: string = " "
• logIn(username: string, password: string): boolean • startTransaction(): void • addItem(itemID: int) • scanItem(itemID: int) • isTaxable(): boolean • enterWeight(weight: float) • enterPhoneNumber(phone: string) • scanLoyaltyCard(cardID: string) • getCustomerType(): string • applyCoupon(): float • calculateTotal(): float • rewardTurkeyIfEligible(total: float): boolean • requestPaymentMethod() • selectPaymentMethod(method: string) • processPayment(amount: float): boolean • finalizeTransaction() • generateReceipt() • printReceipt(transaction: Transaction) • error() • removeItem(itemID: int) • cancelTransaction()

Class Description:

Remarks:

Date Type:

INT: Numbers

Float: Decimal Numbers

Boolean: True or False

String: Alphabets & numbers

Transaction:

The Transaction class handles the POS purchase process. The steps start with adding or removing items, calculating the total amount, applying discounts or coupons, selecting the payment method, finalizing the transaction, and, at the end, generating the receipt. It is responsible for the entire checkout process and managing the items in the cart. In the end, it interacts with the receipt function/ class to generate a transaction summary.

Name: Transaction

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

Pre-Conditions:

– A transaction is initiated when a customer selects items for purchase.

– The system processes payments and generates receipts. There are 4 kinds of payment methods,

Dialog:

– A new transaction begins when the customer selects items –

[Transaction]:startTransaction()

– Customer adds an item to the cart –

[Transaction]:addItem(item: Item)

– Customer removes an item from the cart –

[Transaction]:removeItem(itemID: int)

– System calculates the total price of selected items –

[Transaction]:calculateTotal() : float

– Customer applies a discount using a coupon code –

[Transaction]:applyCoupon(code: string) : float

– Customer chooses a payment method and proceeds with the transaction

– System finalizes the transaction after confirming payment – [Transaction]:finalizeTransaction) : boolean

– A receipt is generated containing the details of the transaction – [Transaction]:generateReceipt(): Receipt

• If the payment method is not selected, the system requests the user to choose one – [Transaction]:requestPaymentMethod()

• If the payment is unsuccessful, the transaction is halted – [Transaction]:cancelTransaction()

Post-Conditions:

– If the transaction is completed successfully, a receipt is generated – [Transaction]:generateReceipt()

– If an error occurs, the transaction is either retried or cancelled

– The system updates the inventory after successful completion

Cashier:

Since there is no self-scan, all of the major tasks are performed by the cashier. The function starts by logging in. Cashier will be responsible for scanning items, entering item weight, applying loyalty discounts, asking the customer and selecting payment method and printing the receipt. The cashier is responsible for the interaction with the POS system to facilitate customer transactions.

Name: Cashier

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

Pre-Conditions

– The cashier must have a valid user PIN to log into the system.

– The POS system is operational and connected to the transaction database.

– A customer is present for checkout.

Dialog

– The cashier logs in to the system using

[Cashier]:logIn(username: string, password: string) : boolean.

Once logged in, the cashier begins processing the customer's items:

– The cashier scans each item using [Cashier]:scanItem(itemID: int), which retrieves item details from the database and updates the active transaction.

– If the item requires weight input, the cashier enters the weight through [Cashier]:enterWeight(weight: float), adjusting the price accordingly.

– If the customer has a loyalty card, the cashier scans it using [Cashier]:scanLoyaltyCard(cardID: string), applying any applicable discounts.

– Alternatively, if the customer provides a phone number linked to their loyalty account, the cashier enters it via [Cashier]:enterPhoneNumber(phone: string).

After all items have been scanned, the cashier selects the payment method using [Cashier]:selectPaymentMethod(method: string). The system then prompts the customer to complete the payment:

– If the payment is successful ([Cashier]:processPayment(amount: float) : boolean returns true), the transaction is finalized.

– If the payment fails, the cashier is prompted to retry or select another payment method.

Upon successful payment, the cashier generates and prints a receipt using [Cashier]:printReceipt(transaction: Transaction), completing the transaction and updating the system records.

Post-Conditions

– The transaction is completed, and the customer receives a printed receipt.

– The system updates inventory levels based on the purchased items.

Manager:

The manager is the main person who interacts with the POS and has administrative power. Along with all of the functionality that a cashier can perform, the manager can perform additional tasks, such as overriding the price of an item, processing customer refunds, and generating the inventry and sales report.

Name: Manager

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

Pre-Conditions

– The manager must have a valid user ID and password to log into the system.

– The POS system is fully operational and connected to the transaction and inventory databases.

– The manager has administrative access rights to perform additional tasks beyond standard cashier functions.

Dialog

– The manager logs in using [Manager]:logIn(username: string, password: string) : boolean.

Once logged in, the manager can perform additional administrative tasks:

– The manager can approve price overrides for items. This is done using [Manager]:approvePriceOverride(itemID: int, newPrice: float) : boolean, which allows them to change the price of a product in the system. If the override is successful, the system updates the item price and the transaction details accordingly.

– The manager can process customer refunds by calling [Manager]:processRefund(transactionID: int) : boolean. This allows the manager to reverse a transaction, refund the customer, and update the inventory and financial records.

– The manager has the ability to generate sales reports for specific time periods. This is done using [Manager]:generateSalesReport(timePeriod: string), which generates a report detailing sales activity, revenue, and other key metrics.

– The manager can generate inventory reports to monitor stock levels and item availability using [Manager]:generateInventoryReport(). This helps the manager make decisions regarding restocking and

inventory management.

– The manager can track the performance of cashiers using [Manager]:trackCashierPerformance(cashierID: int) : string, which provides insights into each cashier's efficiency and sales figures.

Post-Conditions

– The manager successfully performs any of the tasks mentioned above, which are reflected in the system, whether it's updating item prices, processing refunds, generating reports, or monitoring cashier performance.

ITEM:

The Item class represents a product that can be scanned, sold, and processed at the POS system. Each item is uniquely identified and has predefined attributes stored in the central database. Once scanned, the item interacts with the POS system to execute various tasks, such as price calculations, tax application, and quantity updates.

Name: Item

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

Pre-Conditions

– Each item must be registered in the central database with a unique item ID.

– The POS system has access to the item database, and the item is available for sale and scanning.

– The item has predefined attributes like price, weight, tax status, and quantity in the database.

Dialog

– When a cashier scans an item using the POS system, the system identifies the item using its itemID (unique identifier). This triggers the item's details to be retrieved from the central database.

– The item's name, price, weight, taxable status, and quantity are all displayed and used for transaction processing.

The Item class provides the following functionalities:

– The getPrice() method retrieves the current price of the item, which is essential for price calculation and transaction processing. The item price is applied to the transaction during checkout.

– The setPrice(price: float) method allows the system to update the price of the item, which could be influenced by price overrides, discounts, or promotions.

– The getWeight() method retrieves the item's weight. This is important for items that are sold by weight, such as produce or deli items, allowing for price calculations based on weight.

– The setWeight(weight: float) method allows the system to update the weight of an item. This might be used when weighing items manually during the transaction.

– The taxable attribute indicates whether the item is subject to tax. If taxable is set to true, the system applies the relevant sales tax when the item is processed during checkout.

Post-Conditions

– The item's details, including price, weight, and quantity, are updated in the POS system as needed during the transaction.

– After the transaction, the item's quantity is reduced based on the amount sold, and any changes to the

price or weight are reflected in the database.

– If the item is not available in the required quantity or the price has been manually adjusted, the system updates the transaction accordingly and provides appropriate messages to the cashier or manager.

Costumer:

Name: Customer

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

The customer to whom all of the interaction is done. The customer data is distinguished by preferred or non-preferred. Preferred customer means a customer with a loyalty card. For instance, whenever shopping at the store, they can give their loyalty card barcode to the customer or may enter their phone number to include the purchase in order to get reward points.

Pre-Conditions

– The customer must have a registered account in the system, identified by a unique customerID.

– The customer may or may not have a loyalty card, which determines their loyaltyStatus ("loyal" or "non-loyal").

– The customer's details, including name, phoneNumber, and loyaltyStatus, are stored in the central database for use during transactions.

Dialog

The **Customer** class is responsible for managing customer interactions with the POS system, with key functionalities based on the customer's loyalty status:

– If the customer provides their loyalty card barcode or enters their phone number, the system identifies

them as a **loyal customer** and applies loyalty-based rewards.

– If the customer has no loyalty card, they are considered a **non-loyal customer**, and no loyalty discounts are applied.

The **Customer** class provides the following methods:

– The getCustomerType() method determines whether the customer is **loyal** or **non-loyal** based on their loyalty status. If the customer is a loyal customer (i.e., has a loyalty card), this method returns "loyal". Otherwise, it returns "non-loyal".

– The applyLoyaltyDiscount(total: float) method applies a discount to the total amount of the transaction if the customer is a loyal customer. The discount logic could vary based on the store's loyalty program, providing a percentage discount or reward points.

– The updatePhoneNumber(phone: string) method updates the customer's phone number in the system. This ensures the contact information is always up to date, which is important for both loyalty rewards and customer communication.

Post-Conditions

– If the customer is **loyal**, the loyalty discount is applied to the total amount of the transaction before payment is processed. Right now, there is a provision of a free turkey for a customer when the purchase is greater than 500$ and the free turkey is given via coupon.

– The customer's phoneNumber is updated in the system if the customer provides new information.

– The customer’s loyalty status remains unchanged unless they switch from non-loyal to loyal by signing up for the loyalty program or by other means.

Inventory:

The inventory class manages the store's stock and inventory. IT maintains the list of the items, tracks the items and provides the methods to add or remove products, update quantities and check. The inventory class captures the inventory and supports the manager's reporting function. Similarly, all of the interactions that are held at the POS, it's get updated in real time. Suppose if the customer buys a 20oz Diet Coke, then it gets updated in real time at the inventory class.

Name: Inventory

Author: Anjana Pun, Nishan Joti, Rojan Upreti

Last Update: Tuesday, April 1, 2025

Pre-Conditions

– The inventory system is initialized with a list of products (products) and their corresponding stock levels (stockLevels).

-Only the Manager has access to the inventory management

– The inventory system is integrated with the POS system, ensuring real-time updates during transactions.

– Each product in the inventory is represented by an Item object, which contains details such as itemID, name, price, weight, taxable, and quantity.

Dialog

The **Inventory** class is responsible for managing and tracking the stock of all items available in the store. This class facilitates the following operations:

- **addProduct(product: Item)**: This method adds a new product to the inventory. When a new item is added, it is appended to the products list and the corresponding stock level is initialized or

updated in the stockLevels dictionary.

- **removeProduct(itemID: int)**: This method removes a product from the inventory based on its itemID. Once removed, the item is no longer tracked in the inventory, and its stock level is deleted from stockLevels.

- **updateStockLevel(itemID: int, quantity: int)**: This method updates the quantity of a product in the inventory. When a transaction occurs (such as a purchase), the stock level for the purchased item is updated in real-time to reflect the change in quantity. For example, if a customer buys a 20 oz Diet Coke, the stock level for that item is reduced accordingly in the stockLevels dictionary.

- **checkStockLevel(itemID: int): int**: This method checks and returns the current stock level for a specific item based on its itemID. This allows managers or cashiers to monitor inventory levels and ensure stock availability.

Post-Conditions

– After a product is added, removed, or updated, the changes are reflected in the products list and stockLevels dictionary.

– If a product is purchased, the inventory is updated in real-time, ensuring that stock levels are always accurate.

– The inventory data supports reporting functions for managers, allowing them to generate sales and inventory reports based on the current state of the stock.

Explanation:

The POS system operates through a **central database** that connects five POS hardware units, ensuring seamless data synchronization and efficient transaction processing. The **Transaction** component manages the entire checkout process, from adding and removing items, calculating the total cost, applying discounts or coupons, selecting payment methods, and finalizing the purchase. At the end of each transaction, a **receipt** is generated, summarizing the purchase details.

The **Cashier** plays a crucial role in the system, as there is no self-checkout functionality. They begin by logging in using a unique ID and handle tasks such as scanning items, entering item weights for weighted products, applying loyalty discounts, asking for payment methods, and printing receipts. The cashier directly interacts with the **inventory system** to look up item details, ensure stock availability, and update quantities in real time. Additionally, they can scan coupons to apply discounts and manually enter prices when needed.

The **Manager** oversees the entire POS system and has administrative access beyond a cashier's capabilities. Managers can override item prices for price matching, process customer refunds, and generate detailed sales and inventory reports for specific time periods, such as daily, monthly, quarterly, or yearly. They also handle inventory management tasks, including adding, removing, and updating product details in the system. Managers are responsible for customer interactions related to loyalty programs, where preferred customers can scan a loyalty card or enter their phone number to earn reward points.

The **Item** component represents products available for purchase, with attributes such as UPC codes, prices, and weight-based pricing for certain goods. Items interact with the POS system to facilitate price calculations, tax application, and inventory updates.

The **Inventory** system is responsible for maintaining stock levels, tracking item availability, and updating quantities whenever a purchase is made. Since all transactions reflect in real-time, any item purchased immediately updates the stock count. For example, if a customer buys a 20oz Diet Coke, the system deducts it from the inventory instantly.

The **Customer** is the end user benefiting from the POS interactions. The system distinguishes between preferred and non-preferred customers, where preferred customers have loyalty cards and can earn rewards on purchases. Customers also have the option to use store-generated coupons for discounts or promotional items if they meet specific spending thresholds, such as earning a free turkey or ham after spending over $500.

The **POS system workflow** is designed for seamless transaction handling, starting with a cashier logging in, scanning or manually entering items, weighing products when necessary, applying discounts, and finalizing payments. Managers intervene when needed for refunds, price overrides, and inventory updates. At the end of every transaction, a receipt is generated, providing a summary of the items purchased, applicable taxes, and total costs.

This integrated system ensures an efficient checkout experience, accurate inventory tracking, and streamlined store operations, improving overall customer service and business management.

Communication Diagram

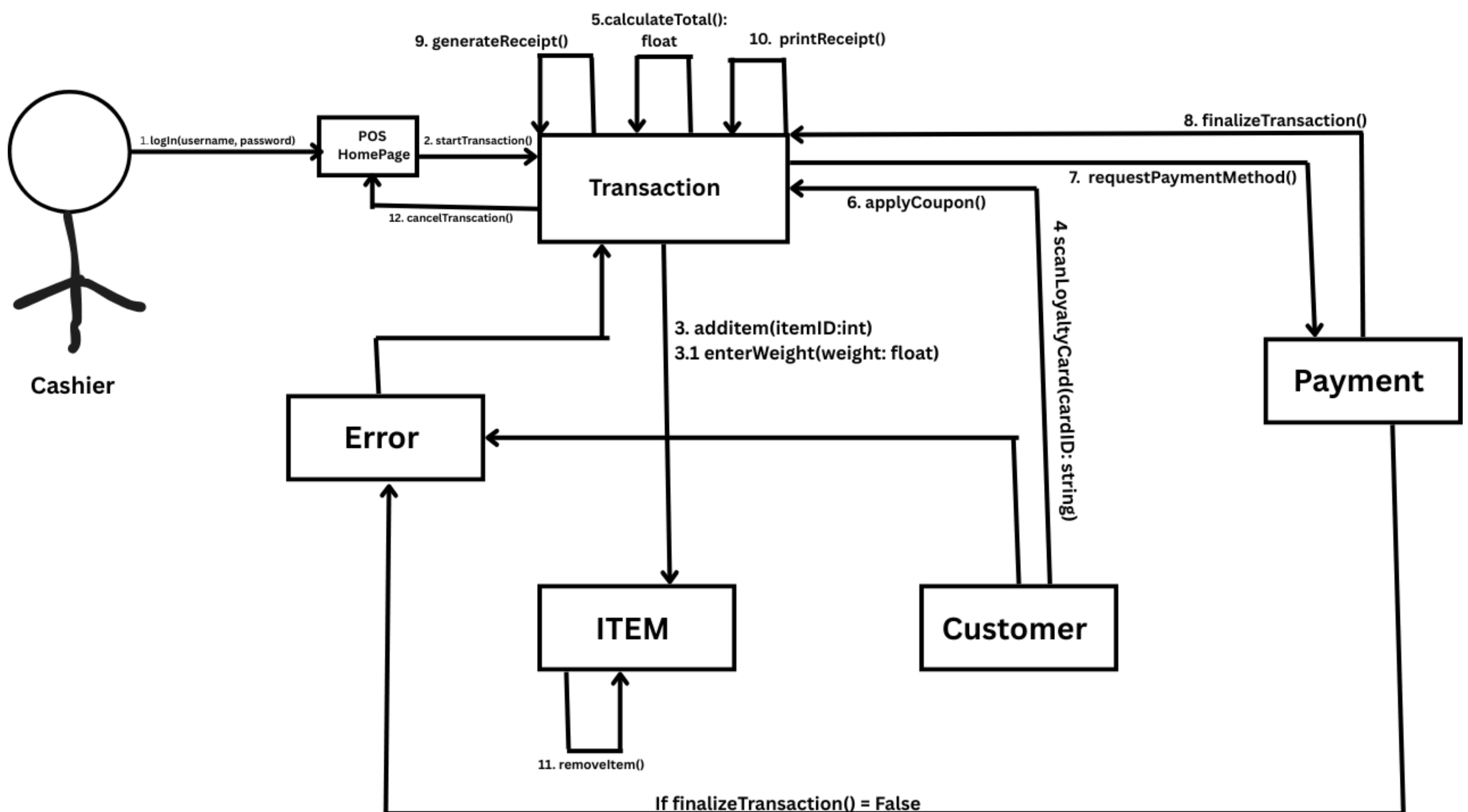
1. Transaction

The objects involved in the transaction process are Customer. In the transaction use case, all of the components from starting a transaction, adding an item card, removing items from the cart, calculating a total, applying a coupon code, finalizing the transaction, accepting the payment method, generating a receipt and cancelling the transaction.

Primary Actor: Cashier

Supporting Actor: Customer

i) Communication Diagram:



ii) Operation Sequence:

1 logIn(username, password)

2 startTransaction()

3 addItem(itemID: int)

3.1 enterWeight(weight: float)

4 scanLoyaltyCard(cardID: string)

5 calculateTotal(): float

6 applyCoupon()

7 requestPaymentMethod()

8 finalizeTransaction()

9 generateReceipt()

10 printReceipt()

11 removeItem()

12 cancelTransaction()

13 error()

iii) Scenario Sequence

1, 2, 3, (3.1), (4), 5, (6), 7, 8, 9, 10: **The complete transaction process without any error, where we kept the item with the right loyalty card and coupon code as an optional input.**

1, 2, 3, (3.1), 11, (4), 5, (6), 7, 8, 9, 10: **When we added an item mistakenly and need to remove that item from the cart and complete the transaction**

1, 2, 3, (3.1), (4), 5, 6, 7, 8, 9, 10: **When a customer wants to use/apply a coupon, the operation number is used. 6 became mandatory instead of optional**

1, 2, 3, (3.1), (4), 5, (6), 7, 13, 12: **When the costumer is paying and when teh requestPaymentethod() is passed to the FinalizeTransaction() and the value comes false, which means the payment fails, then there will be a transaction error and the transaction will be cancelled.**

1, 2, 3, 3.1, (4), 5, (6), 7, 8, 9, 10: **When the customer buys the items with the weight, then the operation no. 3.1 will no longer be optional, and the cashier will be required to enter the weight of the item to complete the transaction.**

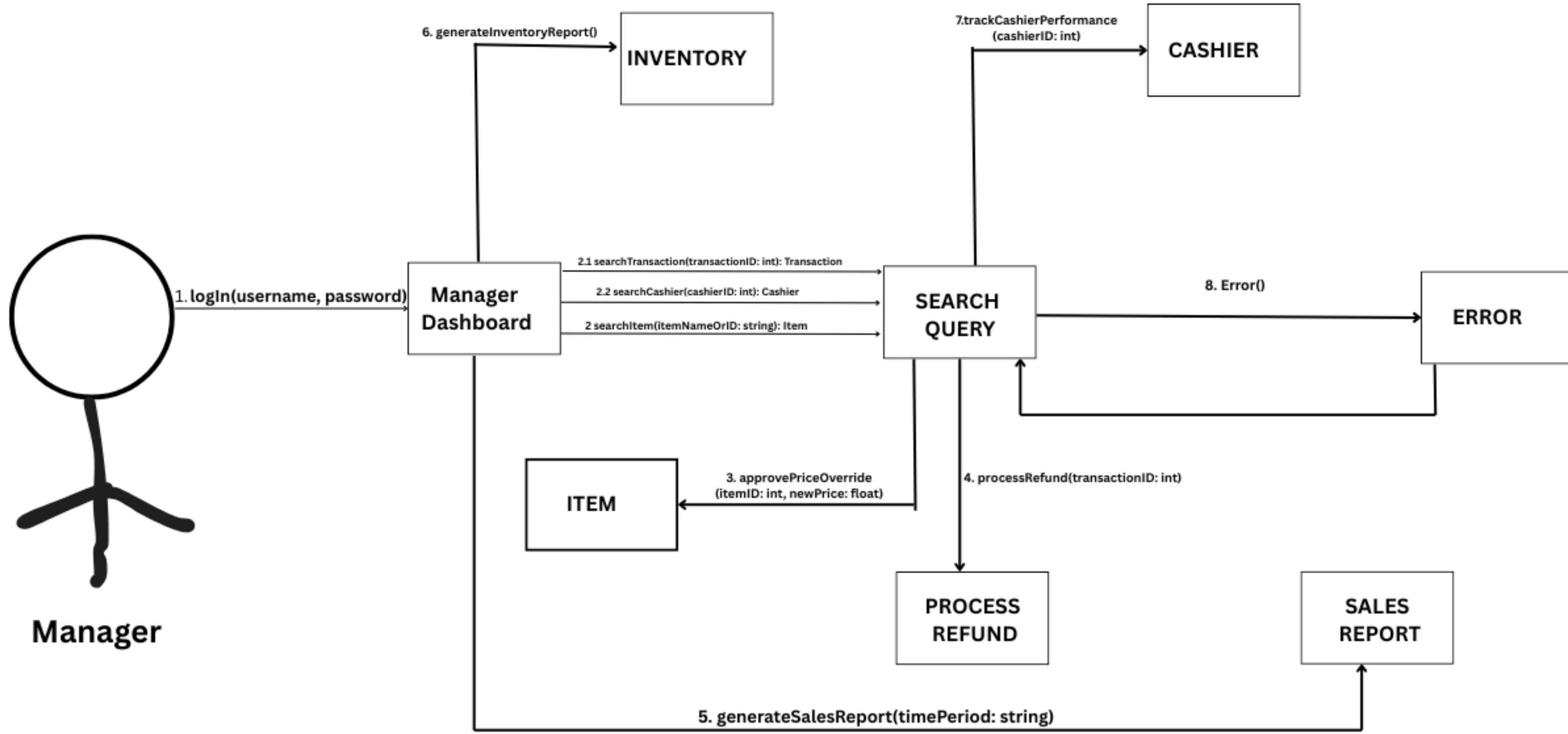
1, 2, 3, 3.1, 4, 5, 6, 7, 8, 9, 10: **The complete transaction where the customer is buying a weighted item, scanning a bar code for the loyalty and has a coupon code.**

2. Manager

Primary Actor: Manager

Supporting Actor: Cashier

i) Communication Diagram



ii) Operation Sequence

1 logIn(username: string, password: string)

2 searchItem(itemNameOrID: string): Item

2.1 searchTransaction(transactionID: int): Transaction

2.2 searchCashier(cashierID: int): Cashier

3 approvePriceOverride(itemID: int, newPrice: float)

4 processRefund(transactionID: int)

5 generateSalesReport(timePeriod: string)

6 generateInventoryReport()

7 trackCashierPerformance(cashierID: int)

8 Error()

iii) Scenario Sequence

1,2,3: **When the Manager needs to override the price of the item, then he will simply log in, search the item and update the item price.**

1,2.1,4: **Whenever the customer wants a refund, then the manager himself needs to login, search the transaction by transaction id and process the refund**

1, 5: **In order to generate the sales report, the manager can simply log in and generate the sales report within a time period**

1,2.2, 7: The **Manager can also track the performance of the cashier. Whenever the manager enters the cashier's username/ID, he can track all of the performance**

1, 6: **To generate the inventory report, the manager can simply log in and generate the inventory report on the basis of the item.**

1, (2 or 2.1 or 2.2) , 10: **When the manager wants to perform the tasks such as searching item to approve price override, searching the transaction to process the refund or searching the cashier id in order to view the cashier performance but he enters the wrong input in the search bar, then the error will be occurred and the operation fails.**

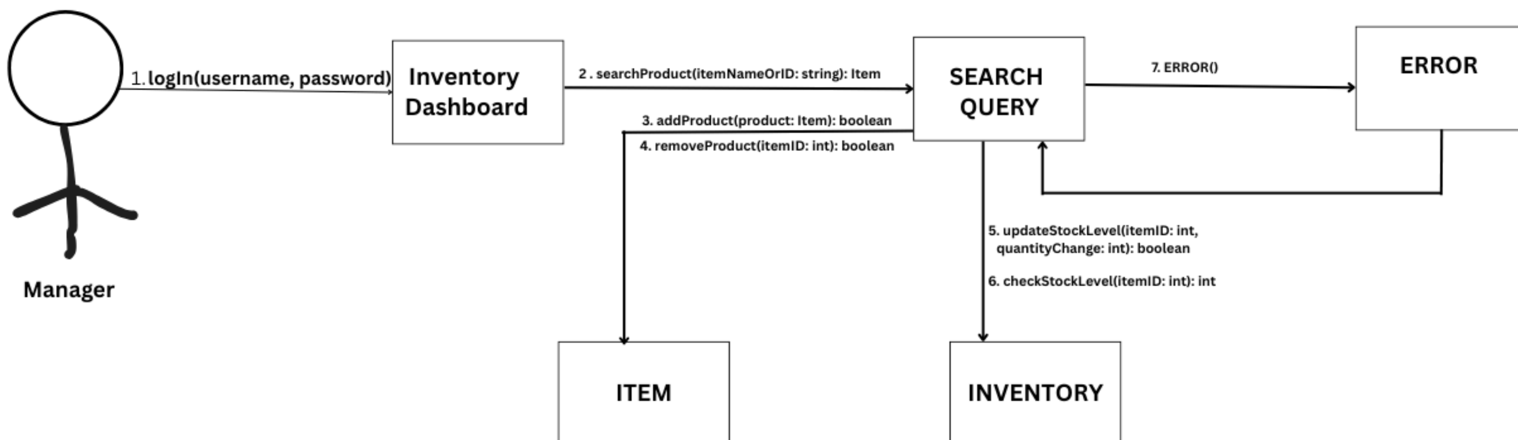
1, 8: **If the username and the password are not of the Manager**

3. Inventory

Primary Actor: Manager

Supporting Actor: System Database

i) Communication Diagram



ii) Operation Sequence

1. logIn(username: string, password: string): boolean
2. searchProduct(itemNameOrID: string): Item
3. addProduct(product: Item): boolean
4. removeProduct(itemID: int): boolean
5. updateStockLevel(itemID: int, quantityChange: int): boolean
6. checkStockLevel(itemID: int): int
7. Error()

iii) Scenario Sequence

1, 2, 3: **Adding the new product to the inventory by the Manager**

1, 10: **If the login username and password don't match with he manager**

1, 2, 4: **Removing the product from the inventory, by searching item with item name or ID by the Manager**

1,2,5: **When there is an issue with the products (backorders) and they can’t be sold, then the manager removes the item by searching for it with ID or name, after which the stock level will be updated**

1,5: Whenever the customer buys something, the respective item gets automatically deducted from the inventory, and the change in quantity is reflected.

1,2,6: In order to check the stock level of the specific item, the manager needs to search for the product with the specific ID and will be able to check the stock level

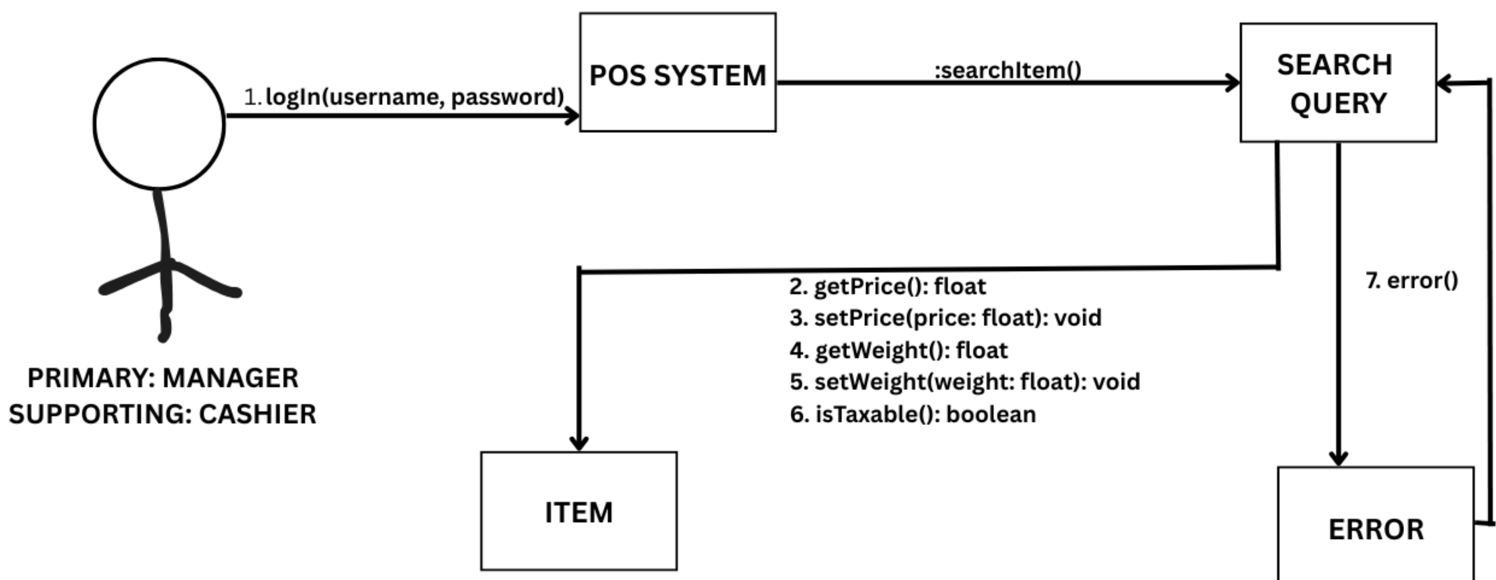
1,2,7: When the manager performs tasks such as removing product, updating or checking the stock, but he enters the wrong item ID/ name, then the error will occur

4 . ITEM

Primary Actor: Manager

Supporting Actor: Cashier

i) Communication Diagram



ii) Operation Sequence:

1. logIn(username: string, password: string): boolean
2. getPrice(): float
3. setPrice(price: float): void
4. getWeight(): float
5. setWeight(weight: float): void
6. isTaxable(): boolean
7. error()

iii) Scenario Sequence

1, 2: **The Manager logs into the POS system and can view the price of the item by using this operation**

1, 2, 3: **So, after checking the price, he might feel it is incorrect, then the manager will be able to update the new price.**

1, 4: **Suppose there is a specific item whose weight is dynamic under the same UPC, such as a cake or a bag of apples, which could have a different weight, so using this function, the manager can view the weight and make sure everything is correct.**

1, 4, 5: **As we mentioned, there might be a dynamic right, so to correct the weight of the item**.

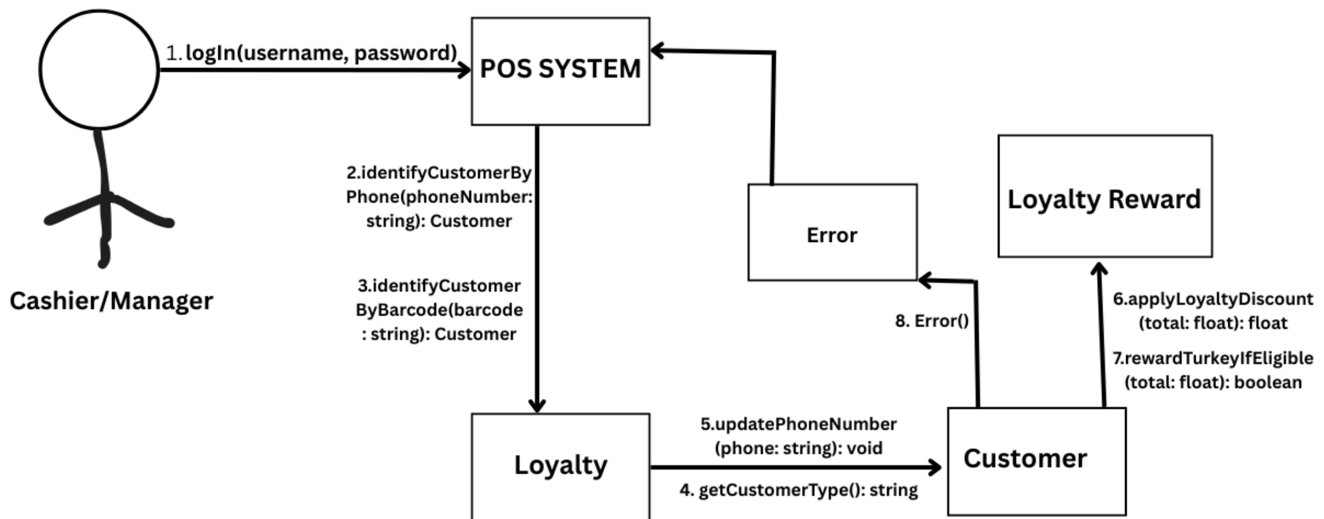
1, 6: **As we know, most of the food items are non-taxable. So, using this properly, the POS system will know if the item is taxable or not. The boolean is used for the logic, os if istaxable() is true, then it is taxable**

5. **Customer**

Primary Actor: Customer

Supporting Actor: Cashier/ Manager

i) Communication Diagram



ii) Operation Sequence

1. logIn(username: string, password: string): boolean
2. identifyCustomerByPhone(phoneNumber: string): Customer
3. identifyCustomerByBarcode(barcode: string): Customer
4. getCustomerType(): string

5. updatePhoneNumber(phone: string): void
6. applyLoyaltyDiscount(total: float): float
7. rewardTurkeyIfEligible(total: float): boolean
8. error()

ii) Scenario Sequence

1,2,(3),4: **When the cashier/ manager wants to know the type of customer, the cashier asks the customer. Since we are checking if the customer is not a loyal(preferred) customer, so they won't have a barcode, the operation will be done by asking them their phone number or barcode as an optional factor**

1, (2 or 3), 4, 5: **Let's suppose that the loyal(preferred) customer wants to update their phone number, then the manager can login, identify by phone number or barcode, get the customer type and update the phone number.**

1,(2 or 3), 4, 7: **Checking up if the customers are eligible for a free turkey if they did a shopping over 500$.**

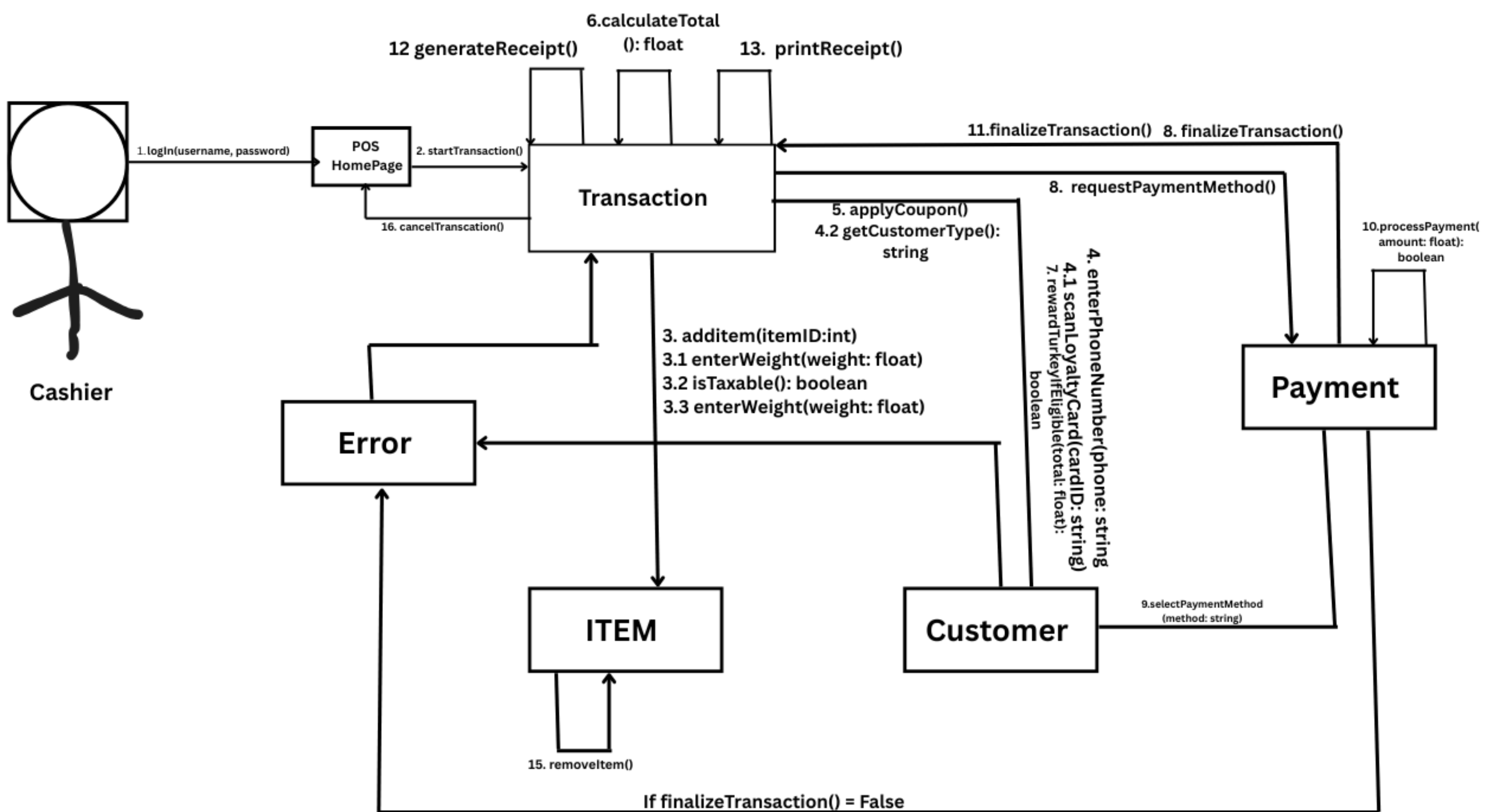
1, (2 or 3), 4, 8: **If the customer is not a loyal customer but wants to claim the loyalty discount, then the operation will be discarded and will be moved to error handling.**

6. Cashier

Primary Actor: Cashier

Supporting Actor: Manager

i) Communication Diagram



ii) Operation Sequence

1 logIn(username: string, password: string): boolean

2 startTransaction(): void

3 addItem(itemID: int)

3.1 scanItem(itemID: int)

3.2 isTaxable(): boolean

3.3 enterWeight(weight: float)

4 enterPhoneNumber(phone: string)

4.1 scanLoyaltyCard(cardID: string)

4.2 getCustomerType(): string

5 applyCoupon()

6 calculateTotal(): float

7 rewardTurkeyIfEligible(total: float): boolean

8 requestPaymentMethod()

9 selectPaymentMethod(method: string)

10 processPayment(amount: float): boolean

11 finalizeTransaction()

12 generateReceipt()

13 printReceipt(transaction: Transaction)

14 error()

15 removeItem()

16 cancelTransaction()

iii) Sequence Diagram

1, 2, 3, 3.1, 3.2, (3.3), (4 or 4.1), (4.2), (5), 6, 7, 8, 9, 10, 11, 12, 13: **This is a complete sequence for a successful program. The cashier logs in, and the transaction starts. Under the addItem function, the cashier scans the item. And the system will check if it's taxable or nontaxable, which is mandatory to calculate the total amount. Then, the weight of the item is optional. Then, The loyalty part is optional where the user can enter their phone no or can scan the barcode from which the cashier will know the customer type i.e if they are loyal or non preferred customer, then the total amount is calculated, if the purchase is above 500, they qualify for the free turkey. After that, the payment method is requested from the customer, they will select the payment method, the payment is processed, and if the transaction is finalized, then the receipt is generated and printed.**

1, 2, 3, 3.1, 3.2, (3.3), (4 or 4.1), (4.2), (5), 6, 7, 8, 9, 10 (false), 14, 16: **Whenever the payment fails, the POS get error and the transaction get cancelled. Up to before payment, it's a normal process. Then, when the cashier asks how to pay, the customer picks a payment method, amount 4, debit, credit, cash or cheque, and if any of them fail, the process doesn't go forward, and the transaction gets cancelled.**

1, 2, 3, 3.1, 3.2, (3.3), 15, 16: **Assume the customer changed their mind and wants to cancel shopping in the middle, then the transaction will be cancelled. For those scenarios, when the cashier scans the item but the customer abandons it, then the item will be removed from the cart and the transaction will be cancelled.**

1, 2, 3, 3.1, 3.2, (3.3), (4 or 4.1), (4.2), 5, 14, 6, 7, 8, 9, 10, 11, 12, 13: **When the customer wants to apply the coupon code but if it doesn't exists or expired then the error handling will take place and the operation applycoupon() will be skipped and the total will be calculated without any coupon.**

1, 2, 3, 3.1, 3.2, (3.3), (4 or 4.1), 14, 5, 6, 7, 8, 9, 10, 11, 12, 13: **When the customer thinks that they are loyal customers but they are not, then this kind of scenario happens. When they scan their barcode or enter their phone number, their loyalty status doesn't exist, so then the error handling occurs and the operation continues from the next step.**

C++ Implementation:

Use case: Transaction

1. Class Implementation

```
class Transaction {
private:
 int transactionID;
 float totalAmount;
 std::string paymentMethod;
 std::vector<Item> itemList;
 Customer customer;

public:
 void startTransaction();
 void addItem(Item item);
 void removeItem(int itemID);
 void calculateTotal();
 void applyCoupon(std::string code);
 bool finalizeTransaction();
 void generateReceipt();
 void requestPaymentMethod();
 void cancelTransaction();
};
```

2. Relationship Implementation

Transaction interacts with Item (stores multiple items in itemList) and Customer (who is making the purchase). It is used by the Cashier to process the checkout.

3. Superclass/Subclass

The transaction is not part of an inheritance hierarchy.

4. Explanation

This class captures all actions described in the Transaction use case: item scanning, coupon application, calculating totals, and handling payment.

Use Case: Manager

1. Class Implementation

```
class Manager: public User {
public:
 bool logIn(std::string username, std::string password) override;
 Item searchItem(std::string itemNameOrID);
 Transaction searchTransaction(int transactionID);
 Cashier searchCashier(int cashierID);
 void approvePriceOverride(int itemID, float newPrice);
 void processRefund(int transactionID);
 void generateSalesReport(std::string timePeriod);
 void generateInventoryReport();
 void trackCashierPerformance(int cashierID);
 void error();
};
```

2. Relationship Implementation

Manager accesses Inventory, modifies Item data, processes Transaction refunds, and tracks Cashier performance.

3. Superclass/Subclass

Manager is a subclass of User.

4. Explanation

This class reflects all tasks the Manager performs per the use case: overrides, refunds, tracking, and reporting.

Use Case: Inventory

1. Class Implementation

```
class Inventory {
private:
 std::vector<Item> items;

public:
 Item searchProduct(std::string itemNameOrID);
 bool addProduct(Item product);
```

```
 bool removeProduct(int itemID);
 bool updateStockLevel(int itemID, int quantityChange);
 int checkStockLevel(int itemID);
 void error();
};
```

2. Relationship Implementation

Inventory holds and manages multiple Item objects. It is accessed by Manager for product updates.

3. Superclass/Subclass

Inventory is a standalone class with no inheritance.

4. Explanation

This implementation handles product search, stock level checks, and real-time inventory updates as described in the use case.

Use Case: Item

1. Class Implementation

```
class Item {
private:
 int itemID;
 std::string name;
 float price;
 float weight;
 bool taxable;
 int quantity;

public:
 float getPrice();
 void setPrice(float price);
 float getWeight();
 void setWeight(float weight);
 bool isTaxable();
 void error();
};
```

2. Relationship Implementation

Item is used in Transaction, accessed in Inventory, and updated by Manager.

3. Superclass/Subclass

Item is a standalone data model class.

4. Explanation

This class represents the product being sold. It includes attributes and operations consistent with the Item use case.

Use Case: Customer

1. Class Implementation

```
class Customer {
private:
 int customerID;
 std::string name;
 std::string phoneNumber;
 std::string loyaltyStatus;

public:
 Customer identifyCustomerByPhone(std::string phoneNumber);
 Customer identifyCustomerByBarcode(std::string barcode);
 std::string getCustomerType();
 void updatePhoneNumber(std::string phone);
 float applyLoyaltyDiscount(float total);
 bool rewardTurkeyIfEligible(float total);
 void error();
};
```

2. Relationship Implementation

Customer is used in Transaction to apply loyalty rewards. Cashier interacts with Customer

during checkout. **3. Superclass/Subclass**

Customer is not a subclass of any class.

4. Explanation

Customer-related actions like loyalty handling and discount application are implemented here, matching the use case flow.

Use Case: Cashier

1. Class Implementation

```
class Cashier : public User {
public:
 bool logIn(std::string username, std::string password) override;
 void startTransaction();
 void addItem(int itemID);
 void enterWeight(float weight);
 void enterPhoneNumber(std::string phone);
 void scanLoyaltyCard(std::string cardID);
 void applyCoupon();
 float calculateTotal();
 bool rewardTurkeyIfEligible(float total);
 void requestPaymentMethod();
 void selectPaymentMethod(std::string method);
 bool processPayment(float amount);
 void finalizeTransaction();
 void generateReceipt();
 void printReceipt(Transaction transaction);
 void removeItem(int itemID);
 void cancelTransaction();
 void error();
};
```

2. Relationship Implementation

Cashier creates and controls a Transaction and interacts with the Customer and Item.

3. Superclass/Subclass

Cashier is a subclass of User.

4. Explanation

This class includes all cashier actions-starting transactions, applying coupons, handling payments-as detailed in the use case.

Conclusion:

Completely finishing this POS system project was not only a technical task; it ended up being one of the most rewarding learning experiences for our team. As computer science students, not only did we learn a lot but got to go through the whole software development process from requirements gathering and system design, to letting our ideas become messy, object-oriented C++ code, and clean. During the project, we constantly questioned how to improve the system's intuitiveness, realism and apply-ability in real world use in the grocery store.

The system we developed manages every necessary aspect of a check out process: scanning items, processing transactions, updating inventory, managing customer loyalty, and giving managers what they need to deal with overrides and produce reports. We ensured that the cashier and the manager roles were separated, and this real time connection to the main database will keep everything in sync on the terminals. All classes and functions were created to be modular, scalable, and to follow good object-oriented principles such as inheritance and encapsulation.

We had plenty of late nights, 3 a.m. bugs, schedule conflicts, everything under the sun but we saw it through, us being a team of 3. We learned how to work together better, to lean on each other’s strengths, how to communicate better, whether we were writing code, documenting it, or just trying to get on the same page.

This helped us integrate what we’ve been learning here in the class to something tangible. It forced us to think beyond writing code to think like real engineers of software. Looking back at how we began with rough whiteboard sketches and then seeing them all come together in a functioning C++ program is something we’re actually proud of. This has certainly made us ready for more large projects to come, in school and in the real world.