IOT RETAIL STORE APP

OOAD PROJECT

Team Name:

SASS

Team Members:

Avantika Sivakumar – 2019103007
Shruthi Gopalakrishnan – 2019103061
Subhiksha Sai – 2019103067
Sweta Chakravarthi – 2019103070

TABLE OF CONTENTS

INTRODUCTION

Identification:

This Document shall be identified SRS/sass.

Purpose:

This document, referred to as the 'Software Requirement Specifications', outlines the requirements for the proposed MINI CASE STUDY MODULES.

Intended Audience:

The following are stakeholders for the modules. The stakeholder specific to each process in this module is listed under each process described in this document.

- Retail Stores
    - Customers

Proposed Mini Case Study statement:

An application which allows customers to shop at an IoT retail store that works on "Just walk out" technology. Customers scan the QR code upon entering, provided their wallet has a minimum amount. Customers can select and return products from shelves. Cameras are used to identify who picks up which product. The virtual cart is updated in real-time. On leaving the store, their wallet on the app is charged. The receipt is emailed to them. Points are added to their account. Customers can recharge their wallet at any time from the app. The users can also place orders online that can be delivered to them or picked up by them.

Benefits of the system:

- Time efficient
- Social distancing
- No checkout required
- Easy payments – cashless money

Proposed Mini Case Study Overview

This section provides a bird's eye view of the key functionalities of the Mini Case Study Modules.

Key Modules

CODE	KEY MODULES	PURPOSE	USE CASES	ACTORS
A	IoT Retail Store	Facilitates "just walk out" technology of shopping	Scan QR Code, Add to wallet, Update virtual cart, Update product stock, Get Receipt	Customer, Store, Employee, Bank
B	Automation	Identifies products and customers, keeps track of products purchased and returned	Identify product, select product, return product, leave store, Detect Person, Identify person, Scan Barcode	Customer, Camera, Employee, Customer Recognition, Product Recognition, Barcode Scanner, Proximity sensor
C	Delivery Order	Delivers orders to the customer who have placed an online order with automatic reduction of money from the app's wallet	Choose store, Place order, select delivery, Update cart, Pack up order, Deliver order	Customer, In-store employee, Delivery Executive
D	Pick-up Order	Allows customers to pick up order in a "drive thru" fashion after having chosen their products online and placed the order	Choose store, Place order, select pick-up, Pack up order, Update cart, Pick up order	Customer, In-store Employee
E	Contact customer support	Provides customer support service to the customers in case of any enquiry	Customer Support, chat, call, email	Customer, Customer service employee

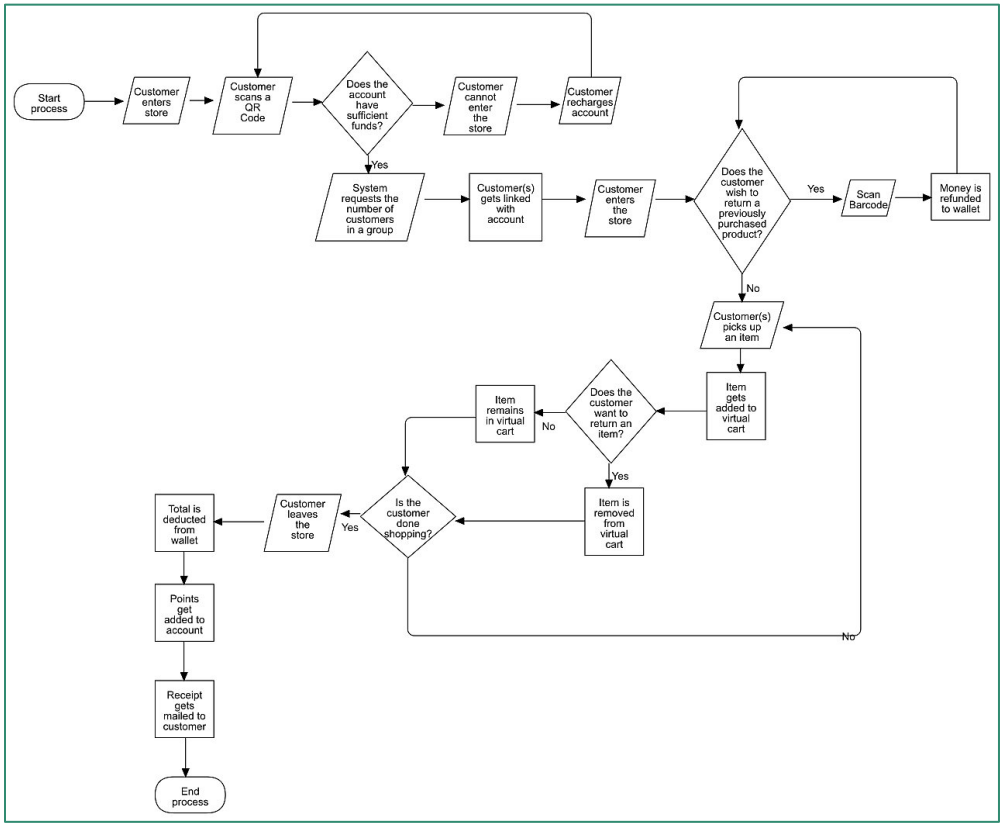
Dependencies with Other Modules

The following are the dependencies with other modules

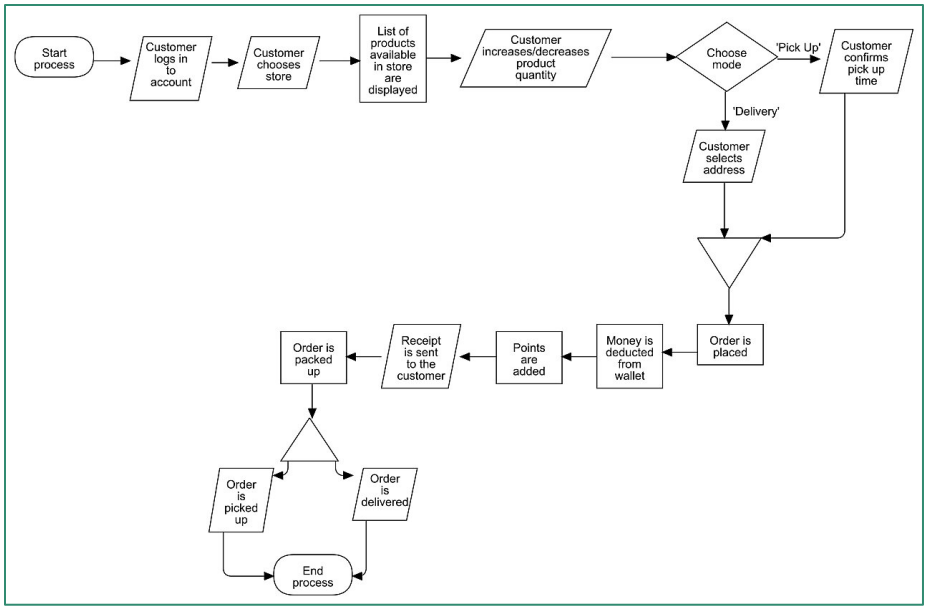
- Interface with Module B - Purchase Automation
  - Identifies who picks which product
  - Facilitates updation of virtual cart
- Interface with module C – Delivery order
  - Identifies who chose delivery order through online order
  - Delivers order to those customers
- Interface with module D – pick up order
  - Identifies who wished to pick up the order through online order
  - Allows to choose pick up time for their order
- Interface with module E – Contact customer support
  - Provides customer support service in case of any enquiries
  - Facilitates three modes namely, chat , call and email

Process Flow Chart

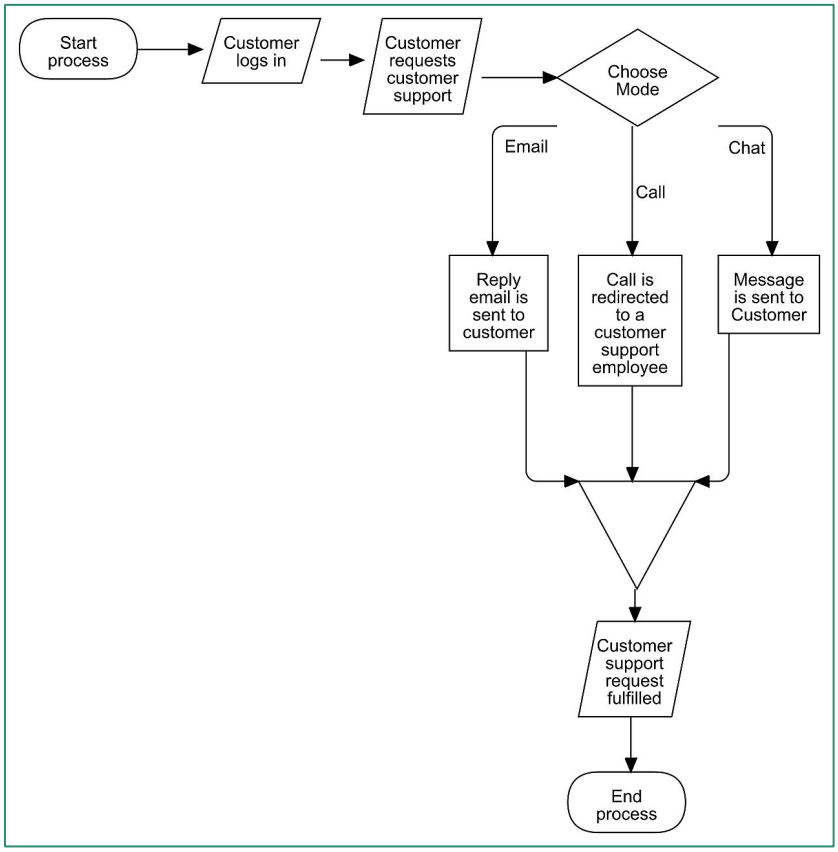
IOT RETAIL STORE:



ONLINE STORE:



CUSTOMER SUPPORT SERVICE:



A. IoT Retail Store App

Scope

This process involves scanning the QR code. Data such as wallet balance, cart details, products in stock, and account points will be populated automatically based on the customer's purchase.

Process Flow

Process Description

1.1.1.1 *Key Fields*

The following are the Key Fields in this process:

- Account ID
- QR Code
- Number of customers using the account
- Wallet balance
- Items in cart
- Cart total
- Account points
- Product ID
- Product stock
- Receipt ID

1.1.1.2 *Functional Specifications*

The process consists of six screens:

- Login/Register: Used by customers to use the app
- QR Code: Used by customers to enter the shop
- Shopping as a group: Allows multiple customers to shop with the same account
- Cart: Displays customer's cart
- Wallet: Customers can view their balance and add money to their wallet
- Receipt: Customers can view their receipt through email

1.1.1.3 *Process Behaviors*

IoT Retail Store App	
Action	Behavior
On entering the store	Data Validation: The app shall auto verify the QR code and link the customers entering the store to the account, if the wallet balance is sufficient.
On selecting/returning products	Data Validation: The systems shall update the virtual cart and the product stock.
On scanning the barcode	Data Validation: The system reads the barcode on the product and refund the money to the account's wallet
On leaving the store	Data Validation: The systems shall auto generate the receipt and deduct the amount from the wallet. Points are calculated and added to the account based on the purchase.

B. Automation

Scope

This process involves tracking customers, identifying products and recognition of customers and products . Data such as cart details, products in stock, and the account's points will be populated automatically based on the customer's purchase.

Process Flow

Process Description

1.1.1.4 *Key Fields*

The following are the Key Fields in this process:

- Account ID
- Customers using the account
- Product ID
- Product stock

1.1.1.5 *Functional Specifications*

The process consists of:

- Detecting customer upon entering the store
- Tracking customers as they walk around the store
- Identifying products picked up or returned by the customer
- Identifying when a customer leaves the store

1.1.1.6 Process Behaviors

Purchase Automation	
Action	Behavior
On entering the store	Data Validation: The app shall track customers around the store and link them to their account.
On selecting/returning products	Data Validation: The systems shall send the product and customer recognition data to the recognition software and update the virtual cart and the product stock.
On scanning the barcode	Data Validation: The system reads the barcode on the product and refund the money to the account's wallet
On leaving the store	Data Validation: The systems shall auto generate the receipt and deduct the amount from the wallet. Points are calculated and added to the account based on the purchase.

C. Delivery Order

Scope

This process involves allowing customers to place online orders and choose for delivering the order to thrm. Data such as cart details, products in stock, and the account's points will be populated automatically based on the customer's order.

Process Flow

Process Description

1.1.1.7 Key Fields

The following are the Key Fields in this process:

- Account ID
- Store ID
- Product ID
- Product stock
- Order ID
- Address
- Items in cart
- Cart total
- Account points

- Receipt ID

1.1.1.8 *Functional Specifications*

The process consists of seven screens:

- Address: allows customers to add address for delivery of order
- Choose store: to choose the store to place order
- Products: view available products to add to cart
- Cart: Displays customer's cart
- Place order: places the order by the customer upon confirmation
- Select delivery: Allows customer to opt for delivery and choose address
- Wallet: Customers can view their balance and add money to their wallet

1.1.1.9 *Process Behaviors*

Delivery order	
Action	Behavior
On choosing the store	Data Validation: The app shall display the products available in the chosen store and check for delivery to the chosen address.
On increasing/decreasing quantity	Data Validation: The app will check if quantity is greater than zero before decreasing, if so the product quantity is increased/decreased in the customer's cart.
On checking out	Data Validation: The app allows the customer to choose for delivery and add address. The system shall auto generate the receipt and deduct the amount from the wallet. Points are calculated and added to the account based on the purchase.

D. Pick up Order

Scope

This process involves allowing customers to place online orders and choose to pick up their order from the store. Data such as cart details, products in stock, and the account's points will be populated automatically based on the customer's order.

Process Flow

Process Description

1.1.1.10 *Key Fields*

The following are the Key Fields in this process:

- Account ID

- Store ID
- Product ID
- Product stock
- Order ID
- Pick up time
- Items in cart
- Cart total
- Account points
- Receipt ID

1.1.1.11 *Functional Specifications*

The process consists of six screens:

- Choose store: to choose the store to place order
- Products: view available products to add to cart
- Cart: Displays customer's cart
- Place order: places the order by the customer upon confirmation
- Select pick up: Allows customer to opt for self-pick up and choose pick up time
- Wallet: Customers can view their balance and add money to their wallet

1.1.1.12 *Process Behaviors*

Pick-up order	
Action	Behavior
On choosing the store	Data Validation: The app shall display the products available in the chosen store and check for pick up slots to the chosen address.
On increasing/decreasing quantity	Data Validation: The app will check if quantity is greater than zero before decreasing, if so, the product quantity is increased/decreased in the customer's cart.
On checking out	Data Validation: The app allows the customer to choose for a pick up time.The system shall auto generate the receipt and deduct the amount from the wallet. Points are calculated and added to the account based on the purchase.

E. Contact Customer Support

Scope

This process involves allowing customers to avail customer support incase of any enquiry. Data such as call, chat, email will be populated based on the customer's request

Process Flow

Process Description

1.1.1.13 Key Fields

The following are the Key Fields in this process:

- Account ID
- Request ID
- Store ID
- Order ID
- Employee ID

1.1.1.14 Functional Specifications

The process consists:

- Choose mode: to choose between call, chat and email
- Call: Lets customers make a call and request
- Chat: Opens a chat screen to allow customers to chat their queries
- Email: Opens the logged in mail of the account to send an email to the support desk
- Finish: Ends the support help once the queries are attended to

1.17.1.3Process Behaviors

Contact Customer Support	
Action	Behavior
On choosing the mode	Data Validation: The app shall display the various modes available and proceeds based on chosen mode.
On choosing call	Data Validation: The app will make a call to the respective account to provide support and take a note on the queries.
On choosing chat	Data Validation: The app will open a chat bot and assign an employee to the account to provide required support to the customer.

On choosing email	Data Validation: The app will log into the registered email on the app and lets customers draft their queries into a mail

ACTORS

1. Primary actor:
    a. Customer
    b. Store

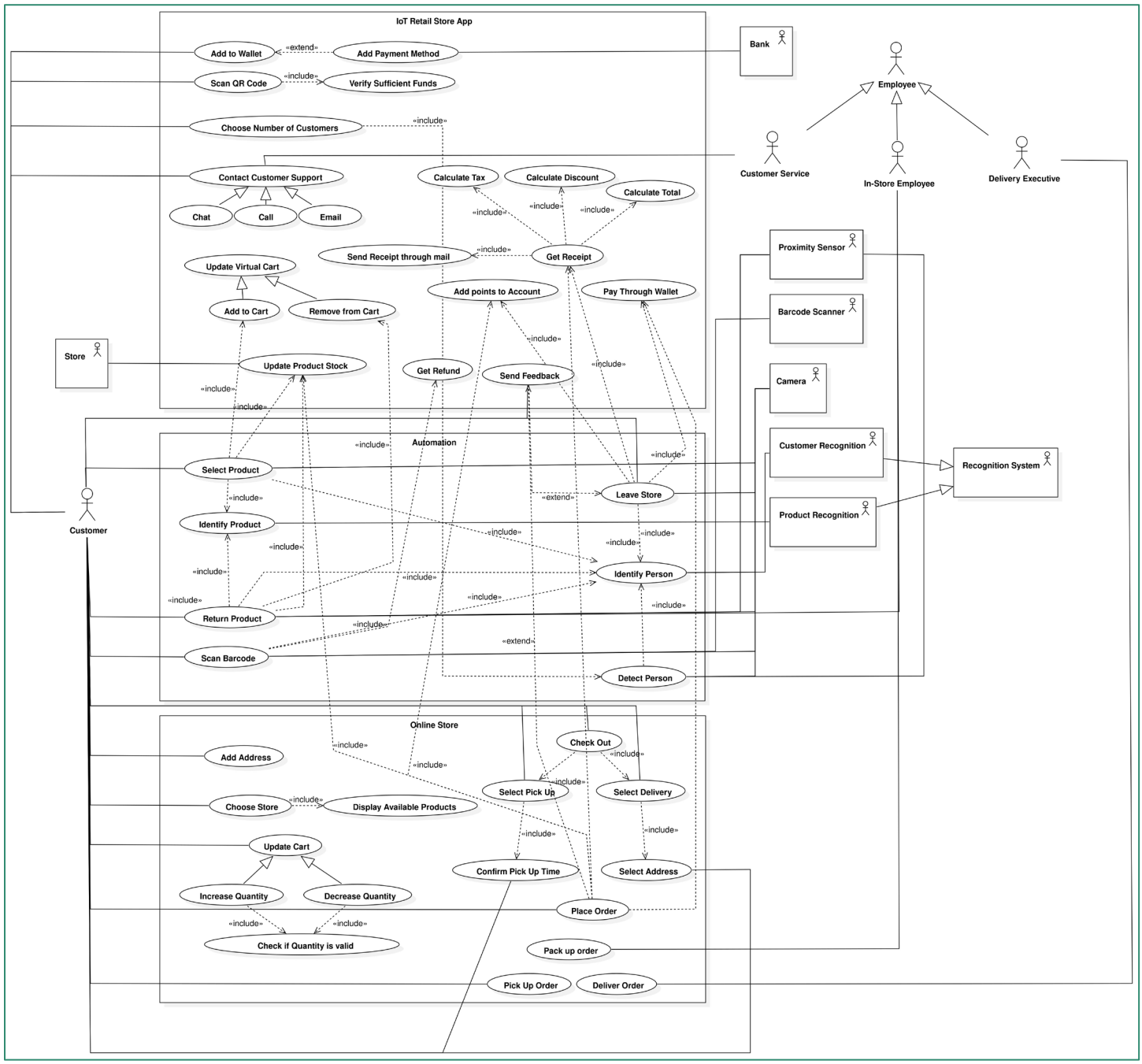
2. Secondary Actors:
    a. Bank
    b. Proximity Sensor
    c. Recognition system
        i. Customer Recognition
        ii. Product Recognition
    d. Barcode scanner
    e. Camera
    f. Employee
        i. In-store Employee
        ii. Customer Service
        iii. Delivery Executive

USE CASES

1. Add to wallet
2. Add payment method
3. Scan QR code
4. Verify sufficient funds
5. Choose number of customers for account
6. Select product
7. Return product
8. Scan barcode
9. Get refund
10. Identify product

11. Detect person
12. Identify person
13. Update virtual cart
    13.1. Add to cart
    13.2. Remove from cart
14. Update product stock
15. Leave store
16. Pay through online wallet
17. Get receipt
18. Calculate tax
19. Calculate discount
20. Calculate total
21. Send receipt through email
22. Add points to account
23. Send feedback
24. Contact customer support
    24.1. Chat
    24.2. Call
    24.3. Email
25. Add address
26. Choose store
27. Display available products
28. Update cart
    28.1. Increase quantity
    28.2. Decrease quantity
29. Check if quantity is valid
30. Check out
31. Select pick up
32. Confirm pick up time
33. Select delivery
34. Select address
35. Place order
36. Pack up order
37. Pick up order
38. Deliver order

USE CASE DIAGRAM



USE CASE DESCRIPTIONS

1 Add to wallet

1.1.1. Scope:

IoT Retail Store App System

1.1.2. Level:

User goal level

1.1.3. Primary Actor:

- Customer

1.1.4. Stakeholders and Interests:

- Customer: Wants to add money to their account's wallet

1.1.5. Preconditions:

- Customer has signed into their account on the app.

1.1.6. Success Guarantee/Post conditions:

- Wallet balance is updated.

1.1.7. Main Success Scenario/Basic Flow:

- Customer signs into their account.
- Customer chooses a payment method(from their saved options or a new payment method).
- Customer adds money to their wallet.
- Customer can enter stores to purchase products.

1.1.8. Extensions/Alternative Flows:

- Customer doesn't add to their wallet.
  Customer is not allowed inside the store.
  Customer can add money and then enter the store.
- Payment Method does not work.
  User can try again later or try a different payment method.
- Customer is not logged in.
  Customer can log in and then add to their wallet.

1.1.9. Special Requirements:

- Payment gateways to add money from their bank account to their wallet.

1.1.10. Technology and Data Variations List:

-

1.1.11. Frequency of Occurrence:

Can be very frequent

2 Add payment method

1.2.1. Scope:

IoT Retail Store App System

1.2.2. Level:

User goal level

1.2.3. Primary Actor:

- Customer
- Bank

1.2.4. Stakeholders and Interests:

- Customer: Wants to save their payment method for easy use every time they want to add to wallet.
- Bank: Verifies payment method.

1.2.5. Preconditions:

- Logged in to the app

1.2.6. Success Guarantee/Post conditions:

- Customer can use payment method to add money to their wallet.

1.2.7. Main Success Scenario/Basic Flow:

- Customer signs into app.
- Customer chooses a payment method.
- Customer adds their details and saves it.
- Customer can use this payment method to add their wallet without having to re-enter their details every time.

1.2.8. Extensions/Alternative Flows:

- Customer chooses not to save the payment method.

Customer can enter their details while adding to wallet or save a payment method at any time from the app.

- Customer is not logged in.
  Customer can log in and then add payment methods.
- Payment Method is invalid.
  Customer can re-enter the correct details and then save the payment method.
- Payment method doesn't get saved.
  Customer can retry after a while.

1.2.9. Special Requirements:

- Bank to verify the payment method.

1.2.10. Technology and Data Variations List:

- If the payment method doesn't get saved, the customer can use the payment method to add to their wallet without saving it.

1.2.11. Frequency of Occurrence:

Can be frequent

3 Scan QR Code

1.3.1. Scope:

IoT Retail Store App System

1.3.2. Level:

User goal level

1.3.3. Primary Actor:

- Customer

1.3.4. Stakeholders and Interests:

- Customer: Wants to enter store
- Camera: Detects customer entering and tracks customer around the store

1.3.5. Preconditions:

- Logged in to the app

1.3.6. Success Guarantee/Post conditions:

- Customer can enter store

1.3.7. Main Success Scenario/Basic Flow:

- Customer scans QR code.

- Customer has sufficient wallet balance.
- Customer can enter the store.

1.3.8. Extensions/Alternative Flows:

- Customer has insufficient wallet balance.

  Customer cannot enter the store.

  Customer can add to wallet and then scan the QR code again.
- Customer is not logged in.

  Customer can log in and then scan the QR code.

1.3.9. Special Requirements:

- Device capable of generating the QR code.

1.3.10. Technology and Data Variations List:

- If the QR code scanner can't scan the generated QR code, the employee can manually enter the account number and allow customers to enter.

1.3.11. Frequency of Occurrence:

Can be very frequent

4 Verify sufficient funds

1.4.1. Scope:

IoT Retail Store App System

1.4.2. Level:

Subfunction

1.4.3. Primary Actor:

- Customer
- Store

1.4.4. Stakeholders and Interests:

- Customer: Wants to enter store
- Store: Needs to check if customer is allowed to enter

1.4.5. Preconditions:

- Customer has scanned QR code.

1.4.6. Success Guarantee/Post conditions:

- Customer is allowed to enter the store.

1.4.7. Main Success Scenario/Basic Flow:

- Customer scans QR code.
- Customer has enough money in their wallet.
- Customer is allowed to enter the store.

1.4.8. Extensions/Alternative Flows:

- Customer has a negative balance in their account wallet and is not allowed to enter the store.
  Customer can add money to their wallet through the app and then scan the QR code again.

1.4.9. Special Requirements:

-

1.4.10. Technology and Data Variations List:

-

1.4.11. Frequency of Occurrence:

Can be very frequent

5 Choose number of customers

1.5.1. Scope:

IoT Retail Store App System

1.5.2. Level:

User goal level

1.5.3. Primary Actor:

- Customer

1.5.4. Stakeholders and Interests:

- Customer: Wants to choose number of people to link to account while entering store
- Camera: Detects chosen number of customer(s) entering and tracks customer(s) around the store

1.5.5. Preconditions:

- Customer has scanned QR code.
- Customer has entered the store.

1.5.6. Success Guarantee/Post conditions:

- Chosen number of customers enter store
- They are linked to the same account and are tracked by the camera.

1.5.7. Main Success Scenario/Basic Flow:

- Customer scans QR code.
- Customer has sufficient wallet balance.
- Customer chooses number of people to be linked to the account.
- Customer(s) can enter the store.
- Customer(s) are linked to the account.
- Customer(s) are tracked by the camera.

1.5.8. Extensions/Alternative Flows:

- Customer does not choose number.
  Customer cannot enter store till a number is chosen on the app.

1.5.9. Special Requirements:

- Device to enter number manually in case app doesn't work.

1.5.10. Technology and Data Variations List:

- If the app doesn't allow the user to choose the number, they can manually enter the number on the device available in the store.

1.5.11. Frequency of Occurrence:

Can be very frequent

6 Select product

1.6.1. Scope:

Automation System

1.6.2. Level:

User goal level

1.6.3. Primary Actor:

Customer

1.6.4. Stakeholders and Interests:

- Customer: Wants to purchase items and exit store
- Camera: Identifies which customer selects which product
- Proximity Sensor: Detects customer and helps identify product

- Store: Updates product stock accordingly

1.6.5. Preconditions:

- Customer has scanned QR code and entered store.

1.6.6. Success Guarantee/Post conditions:

- Product has been added to that customer's virtual cart.
- Product stock has been updated.

1.6.7. Main Success Scenario/Basic Flow:

- Customer has entered store.
- Customer picks up an item.
- Virtual cart and product stock is updated.

1.6.8. Extensions/Alternative Flows:

-

1.6.9. Special Requirements:

- Camera and proximity sensor that collect data about which customer is picking up which product.
- Recognition software to identify the customer and the product.

1.6.10. Technology and Data Variations List:

-

1.6.11. Frequency of Occurrence:

Can be very frequent

7 Return product

1.7.1. Scope:

Automation System

1.7.2. Level:

User goal level

1.7.3. Primary Actor:

Customer

1.7.4. Stakeholders and Interests:

- Customer: Wants to pick another item for purchase and exit store
- Camera: Identifies which customer returns which product

- Proximity Sensor: Detects customer and helps identify product
- Store: Updates product stock accordingly

1.7.5. Preconditions:

- Customer has entered the store.

1.7.6. Success Guarantee/Post conditions:

- Product has been removed from customer's virtual cart.
- Product stock has been updated.

1.7.7. Main Success Scenario/Basic Flow:

- Customer has entered the store
- Customer can return the product before purchase
- Virtual cart and product stock get updated accordingly.

1.7.8. Extensions/Alternative Flows:

- Customer has not returned the product to the correct shelf

Product will not be removed from the cart

In-store Employee returns product to the right place

1.7.9. Special Requirements:

- Camera and proximity sensor that collect data about which customer is picking up which product.
- Recognition software to identify the customer and the product.

1.7.10. Technology and Data Variations List:

- If the cart is not automatically updated, the employee can manually enter the account number and return the product.

1.7.11. Frequency of Occurrence:

Can be very frequent

8 Scan barcode

1.8.1. Scope:

Automation System

1.8.2. Level:

User goal level

1.8.3. Primary Actor:

- Customer

1.8.4. Stakeholders and Interests:

- Customer: Wants to return purchased item and get refund
- Barcode Scanner: Scans the barcode of the product to be returned

1.8.5. Preconditions:

- Customer has scanned QR code and entered store.

1.8.6. Success Guarantee/Post conditions:

- Refund to product is sent to wallet.
- The product stock is updated.

1.8.7. Main Success Scenario/Basic Flow:

- Customer has entered the store.
- Customer scans the barcode of the purchased product to be returned.
- Wallet is refunded and product stock gets updated.

1.8.8. Extensions/Alternative Flows:

- Customer wants a new product instead of refund.
  Customer asks employee for exchange of product.

1.8.9. Special Requirements:

- Barcode Scanner which scans the barcode of the product
- Recognition software to identify the customer

1.8.10. Technology and Data Variations List:

- If the barcode scanner can't scan the product, the employee can manually enter the account number and initiate refund.

1.8.11. Frequency of Occurrence:

Can be very frequent.

9 Get refund

1.9.1. Scope:

IoT Retail Store App System

1.9.2. Level:

Subfunction

1.9.3. Primary Actor:

- Customer

1.9.4. Stakeholders and Interests:

- Customer: Wants to return a purchased product.
- Camera: Detects customer entering and tracks customer around the store.
- Barcode Scanner: Scans the barcode of the product to be returned.

1.9.5. Preconditions:

- Customer scans QR code and enters store.
- Customer should scan the barcode of an already purchased product

1.9.6. Success Guarantee/Post conditions:

- Customer gets refund to their wallet.

1.9.7. Main Success Scenario/Basic Flow:

- Customer scans QR code and enter store.
- Customer scans barcode of the product to be returned.
- Customer's wallet is recharged with refund.

1.9.8. Extensions/Alternative Flows:

- Customer does not get refund to scanned product.

Customer can scan the same barcode again.

1.9.9. Special Requirements:

- Device capable of scanning the product barcode

1.9.10. Technology and Data Variations List:

- If the refund is not initiated automatically, the in-store employee can manually enter the account number and initiate refund.

1.9.11. Frequency of Occurrence:

Not very frequent

10 Identify product

1.10.1. Scope:

Automation System

1.10.2. Level:

Subfunction

1.10.3. Primary Actor:

- Product Recognition

1.10.4. Stakeholders and Interests:

- Camera: Identifies which product is picked up from/returned to the shelf
- Proximity sensor: identifies the product that the customer interacts with

1.10.5. Preconditions:

- Customer picks up or returns a product

1.10.6. Success Guarantee/Post conditions:

- Product has been identified and added to/removed from virtual cart
- Product stock is updated

1.10.7. Main Success Scenario/Basic Flow:

- Customer picks/returns a product
- Proximity sensor identifies picked up/returned product
- Camera identifies customer picking up/returning the product
- Virtual cart and product stock is updated.

1.10.8. Extensions/Alternative Flows:

- Product identified by camera is wrong
  Customer can ask in-store employee to scan barcode

1.10.9. Special Requirements:

- Camera and proximity sensor to identify chosen product.

1.10.10. Technology and Data Variations List:

-

1.10.11. Frequency of Occurrence:

Can be very frequent.

11 Detect person

1.11.1. Scope:

Automation System

1.11.2. Level:

Subfunction

1.11.3. Primary Actor:

- Customer
- Proximity Sensor

1.11.4. Stakeholders and Interests:

- Camera: Detects which customer enters and their account
- Proximity sensor: detects persons location and helps recognition software detect person

1.11.5. Preconditions:

- Customer scans QR code and enters the store.

1.11.6. Success Guarantee/Post conditions:

- Customer is tracked by proximity sensor and camera upon entering the store.

1.11.7. Main Success Scenario/Basic Flow:

- Customer scans QR code.
- Customer recognition software detects the person.
- Camera identifies person entering the store.
- Customer enters the store and proximity sensor keeps a track of their location

1.11.8. Extensions/Alternative Flows:

-

1.11.9. Special Requirements:

- Camera that detects a person entering.
- Proximity sensor to track the person and facilitate their shopping.

1.11.10. Technology and Data Variations List:

- If the recognition software is not able to recognise a person entering after scan of QR code, in-store employee can ask them to re-enter.

1.11.11. Frequency of Occurrence:

Can be very frequent

12 Identify person

1.12.1. Scope:

Automation System

1.12.2. Level:

Subfunction

1.12.3. Primary Actor:

Customer Recognition

1.12.4. Stakeholders and Interests:

- Customer Recognition software: identifies the person entering the store.

1.12.5. Preconditions:

- Customer has logged in to the app.
- Customer has scanned QR code to enter store.

1.12.6. Success Guarantee/Post conditions:

- Customer enters the store.
- Virtual cart of the respective customer is updated if customer selects/returns product(s).
- Product stock is updated if customer selects/returns product(s).

1.12.7. Main Success Scenario/Basic Flow:

- Customer scans QR code to enter store.
- Recognition software identifies the person entering the store.
- Customer can pick or return products.
- Corresponding virtual cart is populated and product stock is updated.

1.12.8. Extensions/Alternative Flows:

- Recognition software is unable to identify the person.
  In-store Employee asks customer to re-enter to store.

1.12.9. Special Requirements:

- Recognition software that identifies the person entering the store or picking up/returning a product.

1.12.10. Technology and Data Variations List:

-

1.12.11. Frequency of Occurrence:

Can be very frequent.

13 Update virtual cart

1.13.1. Scope:

IoT Retail Store App System

1.13.2. Level:

Summary

1.13.3. Primary Actor:

- Customer

1.13.4. Stakeholders and Interests:

- Customer: Wants to purchase product
- Product Recognition: Has to detect the specific product selected
- Customer Recognition: Has to identify the customer shopping

1.13.5. Preconditions:

- Customer has logged into the app and scanned the QR to enter the store.
- Customer has picked up/returned a product.

1.13.6. Success Guarantee/Post conditions:

- Virtual cart is updated.

1.13.7. Main Success Scenario/Basic Flow:

- Customer enters store.
- Customer picks up/returns product.
- Virtual cart is updated.

1.13.8. Extensions/Alternative Flows:

- Virtual cart isn't updated.

Customer can ask in-store employee to manually update it.

1.13.9. Special Requirements:

- Product Recognition that recognizes the product.

- Customer Recognition that identifies the customer.

1.13.10. Technology and Data Variations List:

-

1.13.11. Frequency of Occurrence:

Very frequent

14 Add to cart

1.14.1. Scope:

IOT Retail Store App System

1.14.2. Level:

Subfunction

1.14.3. Primary Actor:

- Product Recognition
- Camera

1.14.4. Stakeholders and Interests:

- Product Recognition: Identifies product selected by the customer
- Camera: tracks which customer selects what product.

1.14.5. Preconditions:

- Customer has scanned QR code and entered store.
- Customer has picked up a product.

1.14.6. Success Guarantee/Post conditions:

- Product is added to cart.
- Product stock is updated.

1.14.7. Main Success Scenario/Basic Flow:

- Customer picks up a product.
- Product Identifier recognizes product.
- Selected product is added to cart.
- Product stock is updated.

1.14.8. Extensions/Alternative Flows:

- Product isn't properly added to the cart.
  An in-store employee can manually add the item into the cart upon scanning the barcode.

1.14.9. Special Requirements:

- Product Recognition which recognizes the selected product.

1.14.10. Technology and Data Variations List:

- If the cart is not updated upon selection of product, the customer can scan the barcode to add it to their cart.

1.14.11. Frequency of Occurrence:

Can be very frequent

15 Remove from cart

1.15.1. Scope:

IOT Retail Store App System

1.15.2. Level:

Subfunction

1.15.3. Primary Actor:

- Product Recognition
- Camera

1.15.4. Stakeholders and Interests:

- Product Recognition: Identifies product removed by the customer.
- Camera: tracks which customer returns what product.

1.15.5. Preconditions:

- Customer has entered the store.
- Customer has selected one or more products.

1.15.6. Success Guarantee/Post conditions:

- Product is successfully removed from virtual cart.
- Product stock is updated

1.15.7. Main Success Scenario/Basic Flow:

- Customer scans QR code and enters store.
- Customer returns a chosen product.
- Product is removed from virtual cart.

- Product stock is updated

1.15.8. Extensions/Alternative Flows:

- Product is not removed from cart

Customer can request in-store employee to manually delete product from their virtual cart

1.15.9. Special Requirements:

- Product Recognition which identifies the specific product.

1.15.10. Technology and Data Variations List:

-

1.15.11. Frequency of Occurrence:

Can be very frequent

16 Update product stock

1.16.1. Scope:

IOT Retail Store App System

1.16.2. Level:

Subfunction

1.16.3. Primary Actor:

- Store
- Product Recognition

1.16.4. Stakeholders and Interests:

- Product Recognition: Identifies product selected by the customer
- Store: Product gets removed from store stock

1.16.5. Preconditions:

- Customer has picked up/returned a product OR Customer has placed an online order.

1.16.6. Success Guarantee/Post conditions:

- Product stock is increased or reduced accordingly.

1.16.7. Main Success Scenario/Basic Flow:

- Customer selects/returns product.
- Product gets added to/removed from cart.
- Product stock is increased or reduced.

1.16.8. Extensions/Alternative Flows:

- Product doesn't get reduced from stock.
  The store can periodically check for equivalence of products bought/returned and product stock.

1.16.9. Special Requirements:

- Product Recognition which identifies the specific product.

1.16.10. Technology and Data Variations List:

-

1.16.11. Frequency of Occurrence:

Can be very frequent

17 Leave Store

1.17.1. Scope:

Purchase Automation System

1.17.2. Level:

User goal level

1.17.3. Primary Actor:

- Customer

1.17.4. Stakeholders and Interests:

- Customer: Wants to purchase items and exit store
- Camera: Detects customer leaving and confirms purchase

1.17.5. Preconditions:

- Customer has scanned QR code and entered store.
- Customer is done shopping.

1.17.6. Success Guarantee/Post conditions:

- Wallet is charged.
- Receipt is sent.
- Points are added.

1.17.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store.

- Purchase is confirmed.

1.17.8. Extensions/Alternative Flows:

- Customer doesn't leave store by closing time.
  In-store employee asks customer to leave store.

1.17.9. Special Requirements:

- Camera at the exit that detects which customer is leaving.

1.17.10. Technology and Data Variations List:

-

1.17.11. Frequency of Occurrence:

Can be very frequent

18 Pay through online wallet

1.18.1. Scope:

IOT Retail Store App System

1.18.2. Level:

Subfunction

1.18.3. Primary Actor:

- Customer

1.18.4. Stakeholders and Interests:

- Customer: Leaves the store to confirm purchase
- Camera: Detects customer exiting the store

1.18.5. Preconditions:

- Customer has scanned QR code and entered store.
- Customer is done shopping.
- Customer has left the store OR placed an online order.
- Receipt is generated.

1.18.6. Success Guarantee/Post conditions:

- Wallet has been charged.

1.18.7. Main Success Scenario/Basic Flow:

- Customer leaves store OR places online order.

- The app calculates total amount to be paid and deducts it from the wallet.
- The receipt is sent to the customer through email.

1.18.8. Extensions/Alternative Flows:

- Customer has insufficient wallet balance.

Customer has to pay the amount of money owed to store before entering the store again.

1.18.9. Special Requirements:

- Camera that detects the customer leaving the store

1.18.10. Technology and Data Variations List:

-

1.18.11. Frequency of Occurrence:

Can be very frequent

19 Get receipt

1.19.1. Scope:

IOT Retail Store App System

1.19.2. Level:

Subfunction

1.19.3. Primary Actor:

- Customer

1.19.4. Stakeholders and Interests:

- Customer: Gets receipt after leaving store
- Camera: Detects customer leaving and confirms purchase

1.19.5. Preconditions:

- Customer is done shopping and has left the store OR has placed online order.
- Tax, total and discount are calculated

1.19.6. Success Guarantee/Post conditions:

- Total is calculated.
- Receipt is generated.

1.19.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store OR places online order.
- Purchase is confirmed.
- Receipt is generated.

1.19.8. Extensions/Alternative Flows:

- Customer doesn't buy anything from store
  Receipt not generated

1.19.9. Special Requirements:

- Camera at the exit that detects which customer is leaving.

1.19.10. Technology and Data Variations List:

-

1.19.11. Frequency of Occurrence:

Can be very frequent

20 Calculate tax

1.20.1. Scope:

IoT Retail Store App System

1.20.2. Level:

Subfunction

1.20.3. Primary Actor:

Customer

1.20.4. Stakeholders and Interests:

- Camera: Detects customer leaving and confirms purchase

1.20.5. Preconditions:

- Customer is done shopping and leaves the store OR places online order.

1.20.6. Success Guarantee/Post conditions:

- Tax is calculated on total

1.20.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store OR places online order.

- Purchase is confirmed
- Tax is calculated and added to the total

1.20.8. Extensions/Alternative Flows:

- Customer doesn't buy anything.
  No tax calculated

1.20.9. Special Requirements:

Camera at the exit that detects which customer is leaving.

1.20.10. Technology and Data Variations List:

-

1.20.11. Frequency of Occurrence:

Can be very frequent

21 Calculate discount

1.21.1. Scope:

IOT Retail Store App System

1.21.2. Level:

Subfunction

1.21.3. Primary Actor:

- Customer

1.21.4. Stakeholders and Interests:

- Camera: Detects customer leaving and confirms purchase

1.21.5. Preconditions:

- Customer is done shopping and exits the store OR places online order.

1.21.6. Success Guarantee/Post conditions:

- Discount is calculated on total

1.21.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store OR places online order.
- Purchase is confirmed.
- Discount is calculated and deducted from total.

1.21.8. Extensions/Alternative Flows:

- Customer doesn't buy anything.
  No discount calculated.
- No applicable offers for the purchase amount.
  No discount calculated.

1.21.9. Special Requirements:

- Camera at the exit that detects which customer is leaving.

1.21.10. Frequency of Occurrence:

Can be very frequent

22 Calculate total

1.22.1. Scope:

IoT Retail Store App System

1.22.2. Level:

Subfunction

1.22.3. Primary Actor:

- Customer

1.22.4. Stakeholders and Interests:

- Camera: Detects customer leaving and confirms purchase

1.22.5. Preconditions:

- Customer is done shopping and exits the store OR places online order.

1.22.6. Success Guarantee/Post conditions:

- Total is calculated with tax and discount.

1.22.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store OR places online order.
- Purchase is confirmed.
- Total is calculated.

1.22.8. Extensions/Alternative Flows:

- Customer doesn't buy anything.
  No total calculated.

1.22.9. Special Requirements:

- Camera at the exit that detects which customer is leaving.

1.22.10. Technology and Data Variations List:

-

1.22.11. Frequency of Occurrence:

Can be very frequent

23 Send receipt through email

1.23.1. Scope:

IoT Retail Store App System

1.23.2. Level:

Subfunction

1.23.3. Primary Actor:

- Customer

1.23.4. Stakeholders and Interests:

- Customer: Receives receipt through email

1.23.5. Preconditions:

- Customer exits the store OR places online order.
- Tax, total and discount are calculated
- Receipt is generated

1.23.6. Success Guarantee/Post conditions:

- Customer gets the receipt in their email

1.23.7. Main Success Scenario/Basic Flow:

- Customer leaves store OR places online order.
- Total is calculated
- Receipt is generated and sent

1.23.8. Extensions/Alternative Flows:

- Customer hasn’t bought anything.

Email is not sent.

1.23.9. Special Requirements:

-

1.23.10. Technology and Data Variations List:

- If the customer doesn't receive an email of their receipt, they can access it from the app.

1.23.11. Frequency of Occurrence:

Can be very frequent

24 Add points to account

1.24.1. Scope:

IOT Retail Store App System

1.24.2. Level:

User goal level

1.24.3. Primary Actor:

- Customer

1.24.4. Stakeholders and Interests:

- Customer: Points are added to customer’s account
- Camera: Detects customer leaving and confirms purchase

1.24.5. Preconditions:

- Customer buys products that have points value.
- Customer exits the store OR places online order.

1.24.6. Success Guarantee/Post conditions:

- Points are added to account

1.24.7. Main Success Scenario/Basic Flow:

- Customer is done shopping.
- Customer exits store OR places online order.
- Purchase is confirmed.
- Points are added to account.

1.24.8. Extensions/Alternative Flows:

- Customer doesn't buy products with points value.
  No points added to account.

1.24.9. Special Requirements:

-

1.24.10. Technology and Data Variations List:

- If points aren't added, the customer can select the order from the app to check for eligibility of points.

1.24.11. Frequency of Occurrence:

Averagely frequent

25 Send feedback

1.25.1. Scope:

IoT Retail Store App System

1.25.2. Level:

User goal level

1.25.3. Primary Actor:

- Customer

1.25.4. Stakeholders and Interests:

- Customer: Sends feedback to store app
- Store: Receives feedback

1.25.5. Preconditions:

- Customer has signed into their account.

1.25.6. Success Guarantee/Post conditions:

- Feedback is sent to the app

1.25.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer gives feedback

1.25.8. Extensions/Alternative Flows:

- Customer does not want to give feedback.

No feedback is given.

1.25.9. Special Requirements:

-

1.25.10. Technology and Data Variations List:

-

1.25.11. Frequency of Occurrence:

Averagely Frequent

26 Contact Customer Support

1.26.1. Scope:

System under Discussion

1.26.2. Level:

Summary

1.26.3. Primary Actor:

- Customer

1.26.4. Stakeholders and Interests:

- Customer: Contacts customer support
- Customer Service Employee: Helps customer

1.26.5. Preconditions:

- Customer has signed in to their account

1.26.6. Success Guarantee/Post conditions:

- Customer receives required assistance

1.26.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer contacts customer support
- Customer receives required assistance

1.26.8. Extensions/Alternative Flows:

- Customer does not want to contact customer support.
  No assistance is given.
- Customer isn't able to reach customer support.
  They can ask in-store employees for help.

1.26.9. Special Requirements:

-

1.26.10. Technology and Data Variations List:

-

1.26.11. Frequency of Occurrence:

Averagely Frequent

27 Chat

1.27.1. Scope:

System under Discussion

1.27.2. Level:

User goal level

1.27.3. Primary Actor:

- Customer

1.27.4. Stakeholders and Interests:

- Customer: Chats with customer service employee
- Customer Service Employee: Chats with customer

1.27.5. Preconditions:

- Customer has signed into their account.

1.27.6. Success Guarantee/Post conditions:

- Customer service employee provides assistance through chat

1.27.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer opens chat window with customer support
- Customer receives required assistance

1.27.8. Extensions/Alternative Flows:

- Customer does not want to contact customer support.
  No assistance is given
- Customer isn't able to chat with customer support.
  Customer can try other methods of contacting customer support.

1.27.9. Special Requirements:

-

1.27.10. Technology and Data Variations List:

-

1.27.11. Frequency of Occurrence:

Averagely Frequent

28 Call

1.28.1. Scope:

System under Discussion

1.28.2. Level:

User goal level

1.28.3. Primary Actor:

- Customer

1.28.4. Stakeholders and Interests:

- Customer: Calls customer support
- Customer Service Employee: Talks to customer

1.28.5. Preconditions:

- Customer has signed into their account.

1.28.6. Success Guarantee/Post conditions:

- Customer service employee provides assistance through call

1.28.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer calls customer support
- Customer receives required assistance

1.28.8. Extensions/Alternative Flows:

- Customer does not want to contact customer support.
  No assistance is given.
- Customer isn't able to call customer support.
  Customer can try other methods of contacting customer support.

1.28.9. Special Requirements:

-

1.28.10. Technology and Data Variations List:

-

1.28.11. Frequency of Occurrence:

Averagely Frequent

29 Email

1.29.1. Scope:

System under Discussion

1.29.2. Level:

User goal level

1.29.3. Primary Actor:

- Customer

1.29.4. Stakeholders and Interests:

- Customer: Sends email to customer support
- Customer Service Employee: Responds to customer email

1.29.5. Preconditions:

- Customer has signed into their account.

1.29.6. Success Guarantee/Post conditions:

- Customer Service Employee provides assistance through email

1.29.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer opens email window with customer support
- Customer Service Employee provides assistance through email

1.29.8. Extensions/Alternative Flows:

- Customer does not want to contact customer support.
  No assistance is given.
- Customer isn't able to email customer support.
  Customer can try other methods of contacting customer support.

1.29.9. Special Requirements:

-

1.29.10. Technology and Data Variations List:

-

1.29.11. Frequency of Occurrence:

Averagely Frequent

30 Add Address

1.30.1. Scope:

Online Store System

1.30.2. Level:

User goal level

1.30.3. Primary Actor:

- Customer

1.30.4. Stakeholders and Interests:

- Customer: Adds address

1.30.5. Preconditions:

- Customer has signed into their account.

1.30.6. Success Guarantee/Post conditions:

- Address is added

1.30.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer adds an address

1.30.8. Extensions/Alternative Flows:

- Customer does not want to add address.

No address is added.

1.30.9. Special Requirements:

-

1.30.10. Technology and Data Variations List:

-

1.30.11. Frequency of Occurrence:

Averagely Frequent

31 Choose Store

1.31.1. Scope:

Online Store System

1.31.2. Level:

User goal level

1.31.3. Primary Actor:

- Customer

1.31.4. Stakeholders and Interests:

- Customer: Chooses store to shop from

1.31.5. Preconditions:

- Customer has signed into their account.

1.31.6. Success Guarantee/Post conditions:

- Store is chosen
- Available products are displayed

1.31.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer chooses store to shop from
- Available products are displayed

1.31.8. Extensions/Alternative Flows:

-

1.31.9. Special Requirements:

-

1.31.10. Technology and Data Variations List:

-

1.31.11. Frequency of Occurrence:

Can be very frequent

32 Display Available Products

1.32.1. Scope:

Online Store System

1.32.2. Level:

Subfunction

1.32.3. Primary Actor:

- Customer

1.32.4. Stakeholders and Interests:

- Customer: Chooses store and views products

1.32.5. Preconditions:

- Customer has signed into their account.
- Customer has chosen a store to shop from

1.32.6. Success Guarantee/Post conditions:

- Available products are displayed

1.32.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer chooses store
- Available products at chosen store are displayed

1.32.8. Extensions/Alternative Flows:

-

1.32.9. Special Requirements:

-

1.32.10. Technology and Data Variations List:

-

1.32.11. Frequency of Occurrence:

Can be very frequent

33 Update Cart

1.33.1. Scope:

Online Store System

1.33.2. Level:

Summary

1.33.3. Primary Actor:

- Customer

1.33.4. Stakeholders and Interests:

- Customer: Increases or decreases quantity to cart
- Store: provides product stock

1.33.5. Preconditions:

- Customer has signed into their account.
- Customer has chosen a store

1.33.6. Success Guarantee/Post conditions:

- Cart is updated accordingly

1.33.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer has chosen a store
- Customer increases or decreases quantity

1.33.8. Extensions/Alternative Flows:

- Quantity exceeds product stock. Quantity is decreased
- Quantity is not decreased if value is 0

1.33.9. Special Requirements:

-

1.33.10. Technology and Data Variations List:

-

1.33.11. Frequency of Occurrence:

Can be very frequent

34 Increase Quantity

1.34.1. Scope:

Online Store System

1.34.2. Level:

User goal level

1.34.3. Primary Actor:

- Customer

1.34.4. Stakeholders and Interests:

- Customer: Increases quantity of product in cart
- Store: provides product stock

1.34.5. Preconditions:

- Customer has signed into their account.
- Customer has chosen a store
- Current quantity of product does not exceed product stock

1.34.6. Success Guarantee/Post conditions:

- Quantity of product in cart is increased

1.34.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer chooses store
- Customer increases quantity of item

1.34.8. Extensions/Alternative Flows:

- Quantity exceeds product stock.

1.34.9. Special Requirements:

-

1.34.10. Technology and Data Variations List:

-

1.34.11. Frequency of Occurrence:

Can be very frequent

35 Decrease Quantity

1.35.1. Scope:

Online Store System

1.35.2. Level:

User goal level

1.35.3. Primary Actor:

- Customer

1.35.4. Stakeholders and Interests:

- Customer: Decreases quantity of product in cart
- Store: Provides product stock

1.35.5. Preconditions:

- Customer has signed into their account.
- Customer has chosen a store
- Customer has increased item quantity at least once

1.35.6. Success Guarantee/Post conditions:

- Quantity of product in cart is decreased

1.35.7. Main Success Scenario/Basic Flow:

- Customer is logged in on the app
- Customer chooses store
- Customer increases quantity of item
- Customer decreases quantity of item

1.35.8. Extensions/Alternative Flows:

- Quantity of product in cart is 0. Quantity is not decreased

1.35.9. Special Requirements:

-

1.35.10. Technology and Data Variations List:

-

1.35.11. Frequency of Occurrence:

Can be very frequent

36 Check if quantity is valid

1.36.1. Scope:

Online Store

1.36.2. Level:

Subfunction

1.36.3. Primary Actor:

- Customer

1.36.4. Stakeholders and Interests:

- Customer: Increments/Decrements count of specific product
- Store: Provides stock of specific product

1.36.5. Preconditions:

- Customer wants to increase/decrease quantity.

1.36.6. Success Guarantee/Post conditions:

- If product stock is equal to zero, Customer is not able to increase quantity of product.
- If requested count of a product is checked to be greater than zero, it is added to cart at check out stage.

1.36.7. Main Success Scenario/Basic Flow:

- Customer increases/decreases quantity.
- System checks if there is enough stock of product
- If customer decreases count of product, System checks if decremented count is greater than 0
- If all above mentioned criteria are satisfied, the order is successfully placed at check out

1.36.8. Extensions/Alternative Flows:

- Customer tries to decrease count below 0.

  Error banner is shown
- Customer tried to increase count above available stock

  Out of stock banner is shown

1.36.9. Technology and Data Variations List:

-

1.36.10. Frequency of Occurrence:

Can be very frequent

37 Check out

1.37.1. Scope:

Online Store

1.37.2. Level:

User goal level

1.37.3. Primary Actor:

- Customer

1.37.4. Stakeholders and Interests:

- Customer: Finalises cart for purchase

1.37.5. Preconditions:

- Customer has signed in.
- Customer has chosen store.
- Customer has added at least one product to cart.

1.37.6. Success Guarantee/Post conditions:

- Cart is checked out and system proceeds to order options.

1.37.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer selects check out
- Customer is redirected to order options screen

1.37.8. Extensions/Alternative Flows:

Customer does not select check out

Items remain in cart until customer selects check out, even if customer logs out of account.

1.37.9. Special Requirements:

-

1.37.10. Technology and Data Variations List:

-

1.37.11. Frequency of Occurrence:

Can be very frequent

38 Select pick up

1.38.1. Scope:

Online Store

1.38.2. Level:

User goal level

1.38.3. Primary Actor:

- Customer

1.38.4. Stakeholders and Interests:

- Customer: Requests in person pick up of order.

1.38.5. Preconditions:

- Customer has signed into their account.
- Customer has checked out their cart.

1.38.6. Success Guarantee/Post conditions:

- Customer is redirected to confirm pick-up time screen.

1.38.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer selects check out.
- Customer selects pick up.
- Customer is redirected to confirm pick-up time screen.

1.38.8. Extensions/Alternative Flows:

- Customer does not select pick up.

Customer could select delivery.

1.38.9. Special Requirements:

-

1.38.10. Technology and Data Variations List:

-

1.38.11. Frequency of Occurrence:

Can be very frequent

39 Confirm pick up time

1.39.1. Scope:

Online Store

1.39.2. Level:

User goal level

1.39.3. Primary Actor:

- Customer

1.39.4. Stakeholders and Interests:

- Customer: Selects time to pick up order

1.39.5. Preconditions:

- Customer has checked out cart.
- Customer has selected pick up.

1.39.6. Success Guarantee/Post conditions:

- Customer can place order.

1.39.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer checks out cart.
- Customer selects pick up.
- Customer confirms pick up time.
- Customer places order.

1.39.8. Extensions/Alternative Flows:

- Customer does not specify time to pick up order

Customer cannot place order.

1.39.9. Special Requirements:

-

1.39.10. Technology and Data Variations List:

-

1.39.11. Frequency of Occurrence:

Can be very frequent

40 Select Delivery

1.40.1. Scope:

Online Store

1.40.2. Level:

User goal level

1.40.3. Primary Actor:

- Customer

1.40.4. Stakeholders and Interests:

- Customer: Requests delivery of order.

1.40.5. Preconditions:

- Customer has chosen store.
- Customer has selected product(s).
- Customer has checked out their cart.

1.40.6. Success Guarantee/Post conditions:

- Customer is redirected to choose delivery address screen.

1.40.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer selects check out.
- Customer selects delivery.
- Customer is redirected to choose delivery address screen.

1.40.8. Extensions/Alternative Flows:

- Customer does not select delivery

Customer could select pick up

1.40.9. Special Requirements:

-

1.40.10. Technology and Data Variations List:

-

1.40.11. Frequency of Occurrence:

Can be very frequent

41 Select Address

1.41.1. Scope:

Online Store

1.41.2. Level:

User goal level

1.41.3. Primary Actor:

- Customer

1.41.4. Stakeholders and Interests:

- Customer: Selects address to deliver order to.

1.41.5. Preconditions:

- Customer has checked out cart.
- Customer has selected delivery

1.41.6. Success Guarantee/Post conditions:

- Customer can place order.

1.41.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer checks out cart.
- Customer selects delivery.
- Customer confirms delivery address
- Customer can place order.

1.41.8. Extensions/Alternative Flows:

- Customer does not specify address.
  Customer cannot place order.
- Required address is not added yet.
  Customer can add address.

1.41.9. Special Requirements:

-

1.41.10. Technology and Data Variations List:

-

1.41.11. Frequency of Occurrence:

Can be very frequent

42 Place Order

1.42.1. Scope:

Online store

1.42.2. Level:

User goal level

1.42.3. Primary Actor:

- Customer

1.42.4. Stakeholders and Interests:

- Customer: Places order to be purchased

1.42.5. Preconditions:

- Customer has signed into their account.
- Customer has checked out their cart
- Customer has selected order options.

1.42.6. Success Guarantee/Post conditions:

- Order is received by store
- Cost of order is deducted from wallet
- Points are added.
- Receipt is sent to email.

1.42.7. Main Success Scenario/Basic Flow:

- Customer adds product(s) to cart.
- Customer checks out cart.
- Customer selects applicable order options
- Customer places order
- Order is received by store
- Cost of order is deducted from wallet
- Points are added.
- Receipt is sent to email.

1.42.8. Extensions/Alternative Flows:

- Customer does not place order

Order is not sent to store

Money is not deducted from wallet

1.42.9. Special Requirements:

-

1.42.10. Technology and Data Variations List:

-

1.42.11. Frequency of Occurrence:

Can be very frequent

43 Pack up order

1.43.1. Scope:

Online Store

1.43.2. Level:

Subfunction

1.43.3. Primary Actor:

- In Store Employee

1.43.4. Stakeholders and Interests:

- Customer: Places order via online store app
- Store: Receives order
- In Store Employee: Packs up order

1.43.5. Preconditions:

- Customer has placed order and money has been deducted from their wallet.

1.43.6. Success Guarantee/Post conditions:

- Order has been packed

1.43.7. Main Success Scenario/Basic Flow:

- Customer places an order on the online store app
- Store receives order
- In Store Employee packs up order

1.43.8. Extensions/Alternative Flows:

-

1.43.9. Special Requirements:

-

1.43.10. Technology and Data Variations List:

-

1.43.11. Frequency of Occurrence:

Can be very frequent

44 Pick up order

1.44.1. Scope:

Online Store

1.44.2. Level:

Subfunction

1.44.3. Primary Actor:

- Customer

1.44.4. Stakeholders and Interests:

- Customer: Collects order from store

1.44.5. Preconditions:

- Customer has placed order for pick up
- In Store Employee has packed the order

1.44.6. Success Guarantee/Post conditions:

- Customer is handed order at specified time.

1.44.7. Main Success Scenario/Basic Flow:

- Customer places order and selects pick up
- Customer specifies pick up time
- Store receives order
- In-Store Employee packs up order
- Customer arrives at specified time and picks up order

1.44.8. Extensions/Alternative Flows:

- Customer arrives earlier than specified time to pick up order

Order is packed and handed to customer

- Customer arrives later than specified time to pick up order

Already prepared order is handed to customer

1.44.9. Special Requirements:

-

1.44.10. Frequency of Occurrence:

Can be very frequent

45 Deliver Order

1.45.1. Scope:

Online Store

1.45.2. Level:

Subfunction

1.45.3. Primary Actor:

- Delivery Executive

1.45.4. Stakeholders and Interests:

- Customer: Receives order from store
- Delivery Executive: Delivers order to customers

1.45.5. Preconditions:

- Customer has placed order for delivery
- In Store Employee has packed the order

1.45.6. Success Guarantee/Post conditions:

- Order is handed to customer at specified address

1.45.7. Main Success Scenario/Basic Flow:

- Customer places order and selects delivery
- Customer specifies delivery address
- Store receives order
- In Store Employee packs up order
- Delivery executive delivers order to specified address

1.45.8. Extensions/Alternative Flows:

- Customer entered the wrong address
  Delivery executive can contact customer or return order to store.
- Customer receives products damaged in transit
  Customer can contact customer support for refund/replacement

1.45.9. Special Requirements:

-

1.45.10. Technology and Data Variations List:

-

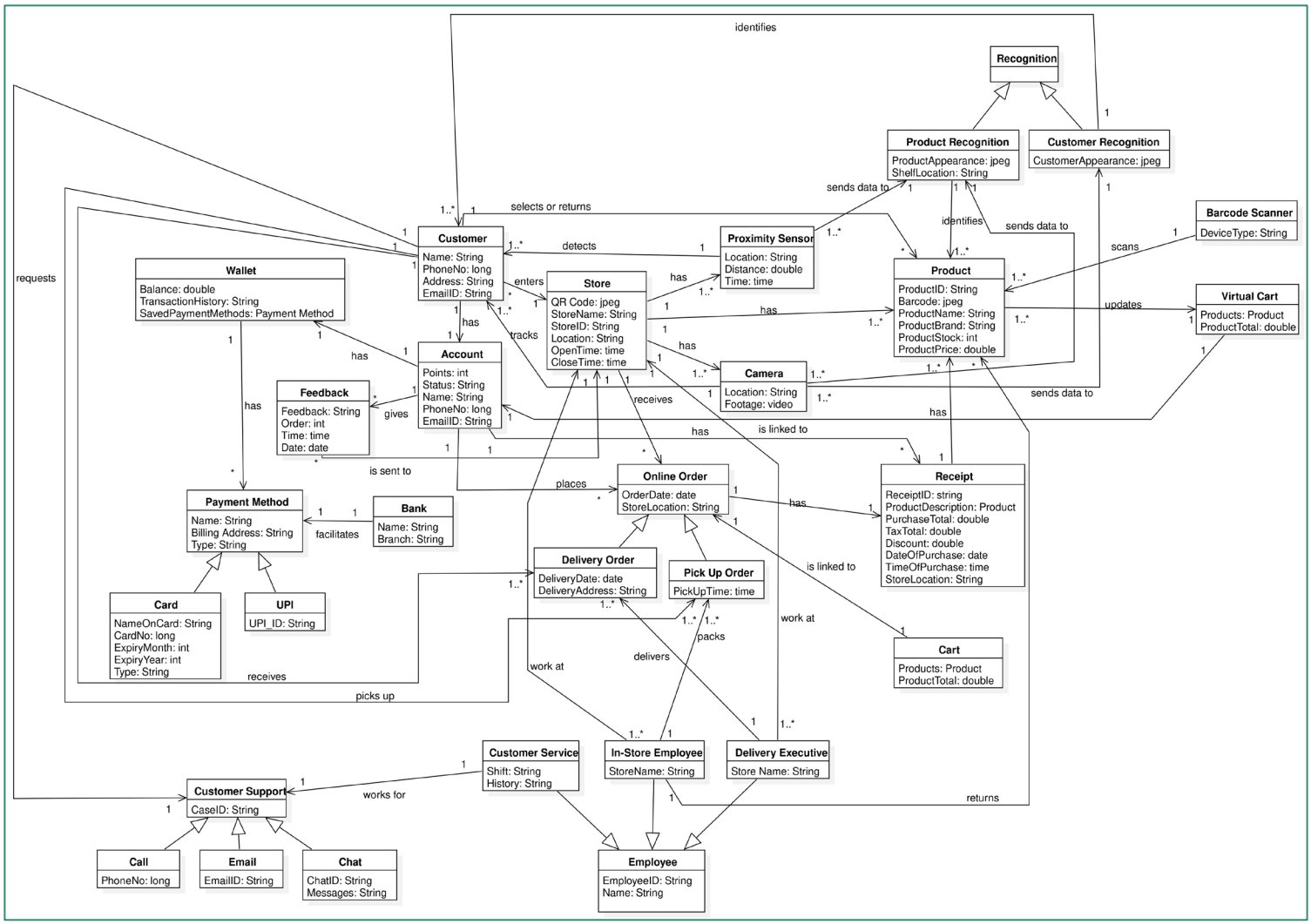
1.45.11. Frequency of Occurrence:

Can be very frequent

DOMAIN CLASSES

1. Customer
2. Employee
3. In-Store Employee
4. Delivery Executive
5. Customer Service
6. Store
7. Camera
8. Product
9. Virtual Cart
10. Cart
11. Account
12. Receipt
13. Feedback
14. Proximity Sensor
15. Wallet
16. Bank
17. Barcode Scanner
18. Recognition system
19. Customer Recognition
20. Product Recognition
21. Payment Method
22. UPI
23. Card
24. Customer Support
25. Call
26. Email
27. Chat
28. Online Order
29. Delivery Order
30. Pick Up Order

DOMAIN MODEL DIAGRAM



DATA DICTIONARY

CLASS NAME	DESCRIPTION
Customer	Customer is a person who uses the application and make their purchase. They can login to the app and use the facilities. They can also make online orders which can be picked up by them or delivered to them. Customers are the primary actors.
Employee	Employee is a person who works in the store and responds to customer enquiry. They arrange products in the store, pack up orders received and deliver the orders to customers
In-Store Employee	It is a subclass of Employee who works at the store to arrange products returned by the customer and pack up orders for delivery or pick up.
Delivery Executive	It is a subclass of Employee who works at the store to deliver the orders placed by customers online.
Customer Service	It is a subclass of Employee who works in customer support.
Store	Store is the primary actor which is the base for the whole system. Customers can go to the store to make their purchase or make online orders to a store and hence experience the benefits of the system.
Camera	Camera is a secondary actor of the system that is used to track people entering the store and link them to their account. It also tracks what product is picked up/returned and by whom hence facilitating “just walk out” technology of the system
Product	Product is in association to store class. It makes up the store. Customers can purchase or return these products.
Virtual Cart	Virtual cart is the important class of the system which is linked to the customer when they shop at the store and facilitates updating product stock. When the customer picks up/returns a product, it is automatically added/removed from the cart in association with camera and product recognition device. Customers do not access the virtual cart directly.

Cart	This is class which is accessed by the customers while placing online orders. The customer can increase or decrease the quantity of products in their cart.
Account	Account class is unique to each customer through which they use the system. Account class contains details about the owner and wallet details which facilitate automatic payment upon leaving the store after purchase.
Receipt	Receipt is generated after the customer has done purchase and left the store. The tax and discount are calculated, and the final total amount deducted from the wallet is sent as a receipt to the customer's registered email id.
Feedback	Feedback class is closely associated the customer class where customers can send their feedback of the store or the app through the system. Each account can send many feedbacks.
Proximity Sensor	Proximity sensor is used to detect the customer entering the store and keep track of their location with respect to the product picked up by them.
Wallet	Each customer's account has a wallet which allows the app to automatically deduct money from the customer upon leaving the store or placing an online order, thereby completing their purchase.
Bank	The bank class is in association with the wallet class. The customers need to recharge their wallet when their balance is insufficient. The bank facilitates this process.
Barcode Scanner	It is a secondary actor that can either be a smartphone, tablet, or any device capable of scanning the barcode, owned by the store. The barcode scanner is used by customers to return purchased products.
Recognition System	It is a software used to identify people and products and distinguish them. It facilitates shopping and has jpeg as attributes. It is owned by the store.
Customer Recognition	This secondary actor of the system is a software owned by the store that identifies person on entering the

	store and links them to their account. It is a subclass of Recognition
Product Recognition	This secondary actor of the system is a software owned by the store to identify product and also help in updating of virtual cart by identifying which product is picked up/returned by a customer. It is a subclass of recognition system
Payment Method	Payment method class is of importance to the wallet class that contains the different payment methods saved by the customer on their wallet to recharge their wallet
UPI	It is a subclass of Payment method class which is a type of payment mode used by customers to recharge their wallet
Card	It is a subclass of Payment method class which is a type of payment mode used by customers to recharge their wallet
Customer Support	This class works in association with customer class through which customers can request for help or enquiry.
Call	This is a mode to request customer support and a subclass of customer support. Customers can make a call to the support desk and get their enquiries resolved.
Email	This is a mode to request customer support and a subclass of customer support. Customers can send an email to the support desk and wait for their enquiry to be answered.
Chat	This is a mode to request customer support and a subclass of customer support. Customers can have an instant chat to the support desk and get their enquiry resolved in the conversation.
Online Order	This class facilitates customers to make online orders to a particular store and choose whether it should be delivered to them or picked up by them. Upon confirmation of order, the total is automatically deducted from their wallet.
Delivery Order	Delivery order is a subclass of online order. Customers can request the store to deliver their order to them.

Pick Up Order	Pick up order is a subclass of online order. Customers can pick up their order from the store as drive thru.
Total	Total is an interface that contains the operation to calculate total which the classes Receipt, Virtual cart and Cart implement
Updations	Updations is an interface which contains the operation to update product stock. Classes virtual cart and Cart implement it.

CLASS-RESPONSIBILITY-COLLABORATION

CLASS NAME	RESPONISIBILITY	COLLABORATION
Customer	Makes purchases at the shop using the IoT Retail Store app or places online orders for pick up/delivery using the app	Delivery Order, Pick up order, Customer Service, Account, Camera, Store, Proximity sensor, Product, Customer Recognition
Employee	Works in the store and responds to customer enquiry, arrange products in the store, pack up orders received and deliver the orders to customers	In-Store Employee, Delivery Executive, Customer Service
In-Store Employee	Works at the store to arrange products returned by the customer and pack up orders for delivery or pick up.	Delivery Order, Pick Up Order, Store, Product, Employee
Delivery Executive	Works at the store to deliver the orders placed by customers online	Store, Delivery Order, Employee
Customer Service	Works in customer support to assist enquiries.	Employee, Customer Support
Store	Has products which customers can shop at the store or make online orders using the IoT Retail Store App.	Customer, Feedback, Proximity Sensor, Product, Camera, Delivery Executive, Online order, In-Store Employee
Camera	Tracks people entering the store and link them to their account. It also tracks what product is picked up/returned by the customer.	Store, Customer, Product Recognition, Customer Recognition
Product	Makes up the store. Customers can purchase or return these products	Customer, Store, Receipt, Product Recognition, Barcode scanner, Virtual Cart, In-store Employee
Virtual Cart	Products picked up/returned by customers is automatically added/removed from the cart in association with camera and product recognition device.	Product, Account, Total, Updations

Cart	Accessed by the customers while placing online orders. The customer can increase or decrease the quantity of products in their cart	Updations, Total, Online order
Account	Unique to each customer with details about the owner and wallet details which facilitate automatic payment upon leaving the store after purchase	Customer, Wallet, Feedback, Receipt, Online order, Virtual Cart
Receipt	Generated after the customer has done purchase and left the store. The final total amount deducted from the wallet is sent as a receipt to the customer's registered email id	Product, Online order, Account, Total
Feedback	Customers can send their feedback of the store or the app through the system	Account, Store
Proximity Sensor	Detects customers entering the store and keeps track of their location with respect to the product picked up by them	Store, Customer, Product Recognition
Wallet	Allows the app to automatically deduct money from the customer upon leaving the store or placing an online order	Account, Payment Method
Bank	Facilitates customers to recharge their wallet	Payment Method
Barcode Scanner	A device used by customers to return purchased products.	Product
Recognition System	Software used to identify people and products and distinguish them	Product Recognition, Customer Recognition
Customer Recognition	Software owned by the store that identifies person on entering the store and links them to their account	Recognition, Camera, Customer

Product Recognition	Software owned by the store to identify product and help in updating of virtual cart	Recognition, Product, Camera, Proximity Sensor
Payment Method	Contains the different payment methods saved by the customer on their wallet	Bank, Wallet, Card, UPI
UPI	Type of payment mode used by customers to recharge their wallet	Payment Method
Card	Type of payment mode used by customers to recharge their wallet	Payment Method
Customer Support	Helps customers with their requests and enquiries	Customer service, Customer, Call, Chat, Email
Call	Mode to request customer support by making a call to the support desk	Customer Support
Email	Mode to request customer support by sending an email to support desk	Customer Support
Chat	Mode to request customer support by having a chat conversation with support desk	Customer Support
Online Order	Customers make online orders to a particular store and choose whether it should be delivered to them or picked up by them	Delivery Order, Pick up order, Cart, Receipt, Store, Account
Delivery Order	Customers can request the store to deliver their order to them.	Online order, Delivery Executive, In-store Employee, Customer
Pick Up Order	Customers can pick up their order from the store as drive thru.	Online order, Customer, In-store Employee
Total	Interface that contains the operation to calculate total	Receipt, virtual cart, Cart
Updations	Interface that contains the operation to update Product Stock	Virtual Cart, Cart

ATTRIBUTES

CLASS	ATTRIBUTE	DESCRIPTION
Customer	Name	Name of the customer
	PhoneNo	Mobile number of the customer
	Address	Address of the customer
	EmailID	Email ID of the customer
Employee	EmployeeID	Unique ID of the employee
	Name	Name of the employee
In-Store Employee	StoreName	Store at which the employee works
Delivery Executive	StoreName	Store at which the employee works
Customer Service	Shift	Timings which the employee works at
	History	Employee's previous customer service responses
Store	QR Code	Code that customers scan to enter the store
	StoreName	Name of the store
	StoreID	Unique ID of the store
	Location	Location of the store
	OpenTime	Opening time of the store
	CloseTime	Closing time of the store
Camera	Location	Part of the store where camera is located
	Footage	Video footage of the store taken by the camera
Product	ProductID	Unique ID of the product
	Barcode	Barcode of the product
	ProductName	Name of the product
	ProductBrand	Brand of the product
	ProductStock	Available stock of the product
	ProductPrice	Price of the product
Virtual Cart	Products	List of products in cart
	ProductTotal	Total price of cart items
Cart	Products	List of products in cart

	ProductTotal	Total price of cart items
Account	Points	Discount points of the account
	Status	Shopping at a store or not, and which store they're currently at
	Name	Name linked to account
	PhoneNo	Phone number linked to account
	EmailID	Email ID linked to account
Receipt	ReceiptID	Unique ID of the purchase
	ProductDescription	List of products purchased
	PurchaseTotal	Total price of purchase
	TaxTotal	Tax calculated
	Discount	Discount based on offers/points
	DateOfPurchase	Date purchased
	TimeOfPurchase	Time purchased
	StoreLocation	Store at which they shopped
Feedback	Feedback	Feedback message sent by customer
	Order	Order related to the feedback
	Time	Time of feedback
	Date	Date of feedbacl
Proximity Sensor	Location	Location of the sensor
	Distance	Distance from the motion
	Time	Time of motion
Wallet	Balance	Amount in wallet
	Transaction History	Previous payments
	SavedPaymentMethods	Payment methods of the user
Bank	Name	Name of bank
	Branch	Branch of bank
Barcode Scanner	DeviceType	Whether it is digital or not
Recognition System	-	-
Customer Recognition	CustomerAppearance	Customer's pictures

Product Recognition	ProductAppearance	Product's pictures
	ShelfLocation	Product's location on shelf
Payment Method	Name	Name of saved method
	Billing Address	Billing address of payment method
	Type	Card/UPI
UPI	UPI_ID	UPI ID
Card	NameOnCard	Name on card
	CardNo	Card number
	ExpiryMonth	Expiry month
	ExpiryYear	Expiry year
	Type	Credit/Debit
Customer Support	CaseID	Unique ID of support request
Call	PhoneNo	Phone number of customer support
Email	EmailID	Email ID of customer support
Chat	ChatID	Unique ID of chat thread
	Messages	Messages sent in chat thread
Online Order	OrderDate	Date of order
	StoreLocation	Store ordered from
Delivery Order	DeliveryDate	Expected delivery date
	DeliveryAddress	Delivery Address
Pick Up Order	PickUpTime	Pick up time

OPERATIONS

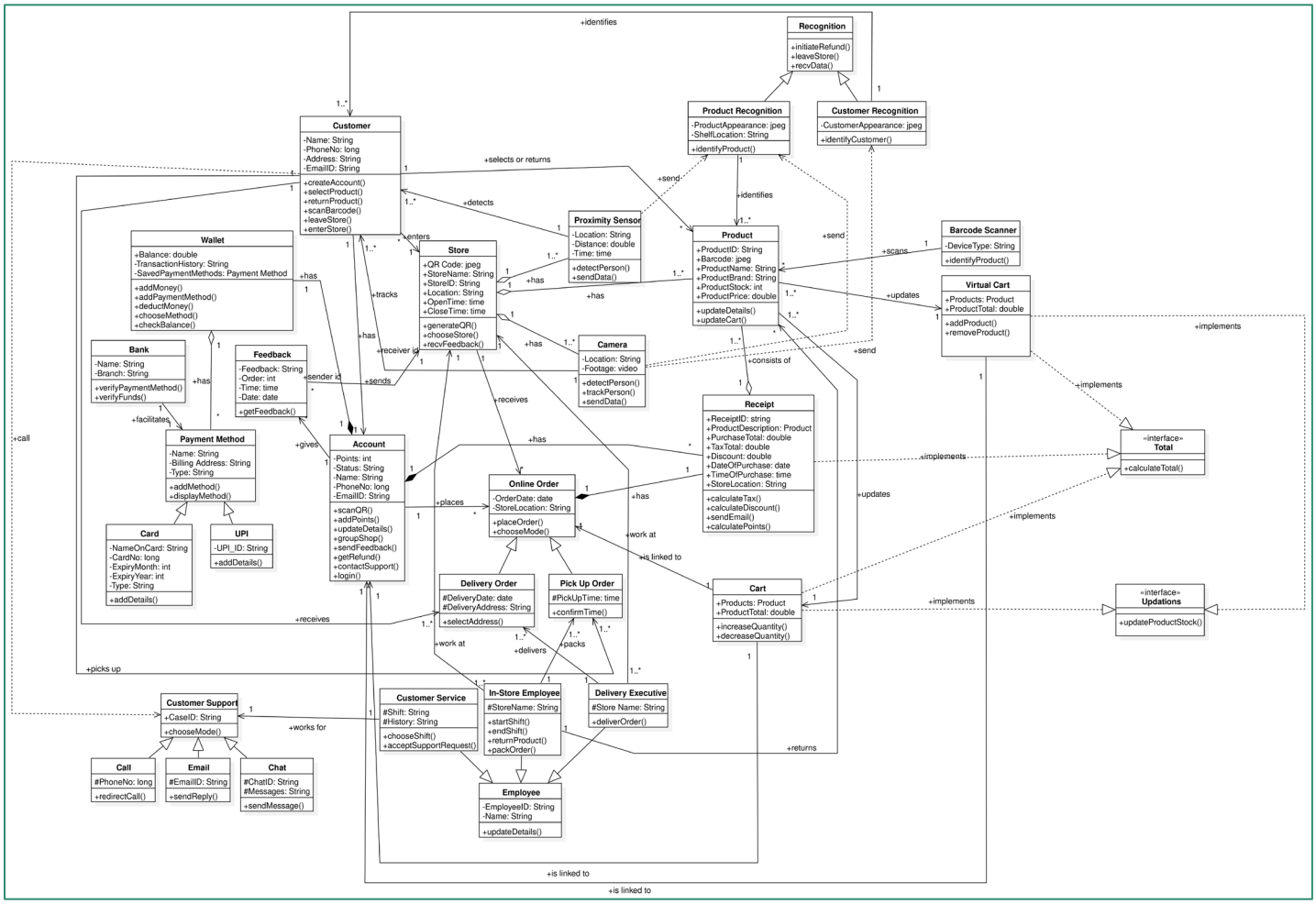
CLASS	OPERATION	DESCRIPTION
Customer	createAccount	To create an account for the customer
	selectProduct	Adds to virtual cart when the customers picks up a product
	returnProduct	Removes from virtual cart when the customers picks up a product
	scanBarcode	Calls necessary functions when customer scans a purchased product's barcode
Employee	updateDetails	Updates employee's details
In-Store Employee	startShift	Notes time of employee starting work
	endShift	Notes time of employee leaving work
	returnProduct	Identifies when employee returns product to shelf
	packOrder	Notes the online order to be packed
Delivery Executive	deliverOrder	Notes the online order requested for delivery
Customer Service	chooseShift	Allows customer service employee to choose shift
	acceptSupportRequest	Allows customer service employee to accept customer support request
Store	generateQR	Generates QR Code for customer to scan while entering the store
Camera	detectPerson	Detects person in view
	trackPerson	Tracks each customer around the store
	sendData	Sends footage data to the recognition system

Product	updateDetails	Updates product's details
	updateCart	Updates customer's cart
Virtual Cart	addProduct	Product is added to virtual cart
	removeProduct	Product is removed from virtual cart
Cart	increaseQuantity	Product is added to cart
	decreaseQuantity	Product is removed from cart
Account	scanQR	Scans QR and identifies which store the customer wants to enter
	addPoints	Adds points to customer's account
	updateDetails	Updates account's details
	groupShop	Allows customer to select number of customer to link to account
	sendFeedback	Gets feedback from customer
	getRefund	Initiates refund to account wallet when product is returned or when customer service employee approves it
	contactSupport	Receives customer support request
Receipt	calculateTax	Calculates tax for the order
	calculateDiscount	Calculates discount for the order
	sendEmail	Sends receipt details to customer via email
	calculatePoints	Calculates points for the order
Feedback	sendToStore	Sends feedback to store
	getFeedback	Gets feedback from customer's account
Proximity Sensor	detectPerson	Detects person nearby

	sendData	Sends data to recognition system
Wallet	addMoney	Adds money to wallet
	addPaymentMethod	Adds payment method
	deductMoney	Deducts money from wallet
	chooseMethod	Allows customer to choose payment method
Bank	verifyPaymentMethod	Verifies if payment method is valid
Barcode Scanner	identifyProduct	Identifies product scanned
Recognition System	initiateRefund	Initiates refund when customer returns product
	leaveStore	Identifies customer leaving store and calls necessary function to confirm the order
	recvData	Receives data from the cameras and proximity sensors
Customer Recognition	identifyCustomer	Identifies customer from data received
Product Recognition	identifyProduct	Identifies product from data received
Payment Method	addMethod	Allows customer to add payment method
	displayMethod	Allows customer to view payment method
UPI	addDetails	Allows customer to add payment method details
Card	addDetails	Allows customer to add payment method details
Customer Support	chooseMode	Allows customer to choose whether to call, email, or chat with customer service employee
Call	redirectCall	Redirects customer's call to customer service employee

Email	sendReply	Sends email reply to customer's support request email
Chat	sendMessage	Sends message to customer's chat thread
Online Order	placeOrder	Confirms order and calls necessary functions
	chooseStore	Allows customer to choose which store they want to place an online order to
	chooseMode	Allows customer to choose between delivery and pick-up
Delivery Order	selectAddress	Allows customer to select their delivery address
Pick Up Order	confirmTime	Allows customer to select their pick-up time
Total (interface)	calculateTotal	Calculates order total
Updations (interface)	updateProductStock	Updates product stock

CLASS DIAGRAM



ASSOCIATION AND MULTIPLICITY

1. *Each* customer has *one* account
2. *Many* customers enter *one* store
3. *Each* customer selects/returns *many* products
4. *Each* customer recognition identifies *many* customers
5. *One* camera tracks *one or many* customers
6. *Each* product recognition identifies *one or many* products
7. *One* account gives *many* feedback
8. *Many* feedbacks are sent to *one* store
9. *One* virtual cart links to *one* account
10. *Each* in-store employee returns *many* products

11. *One or many* products update *one* virtual cart
12. *Each* barcode scanner scan *many* products
13. *Each* proximity sensor detects *one or many* customer
14. *One or many* in-store employee work at *one* store
15. One or many delivery executive work at one store
16. *One* bank facilitates *one* payment method
17. *One* account places *many* online orders
18. *One* store receives *many* online orders
19. *Each* in-store employee packs *one or many* pick-up orders
20. *Each* in-store employee packs *one or many* delivery orders
21. *Each* customer picks up *one or many* pick-up order
22. *Each* delivery executive delivers *one or many* delivery orders
23. *Each* customer receives *one or many* delivery orders
24. *One* cart is linked to *one* online order
25. *Each* customer service employee works for *one* customer support

DEPENDENCIES

1. *One or many* camera send data to *one* customer recognition
2. *One or many* cameras send data to *one* product recognition
3. *One or many* proximity sensors send data to *one* product recognition
4. *Each* customer requests *one* customer support

GENERALIZATIONS

1. Recognition has 2 derived classes: Product Recognition, Customer Recognition
2. Customer Support has 3 derived classes: Call, Email, Chat
3. Employee has 3 derived classes: Delivery Executive, In-Store Employee, Customer Service
4. Payment Method has 2 derived classes: Card, UPI
5. Online Order has 2 derived classes: Delivery Order, Pick Up Order

REALIZATIONS

Interfaces:

- Total
- Updations

1. Virtual cart implements Updations
2. Cart implements Updations
3. Virtual cart implements Total
4. Receipt implements Total
5. Cart implements Total

COMPOSITIONS

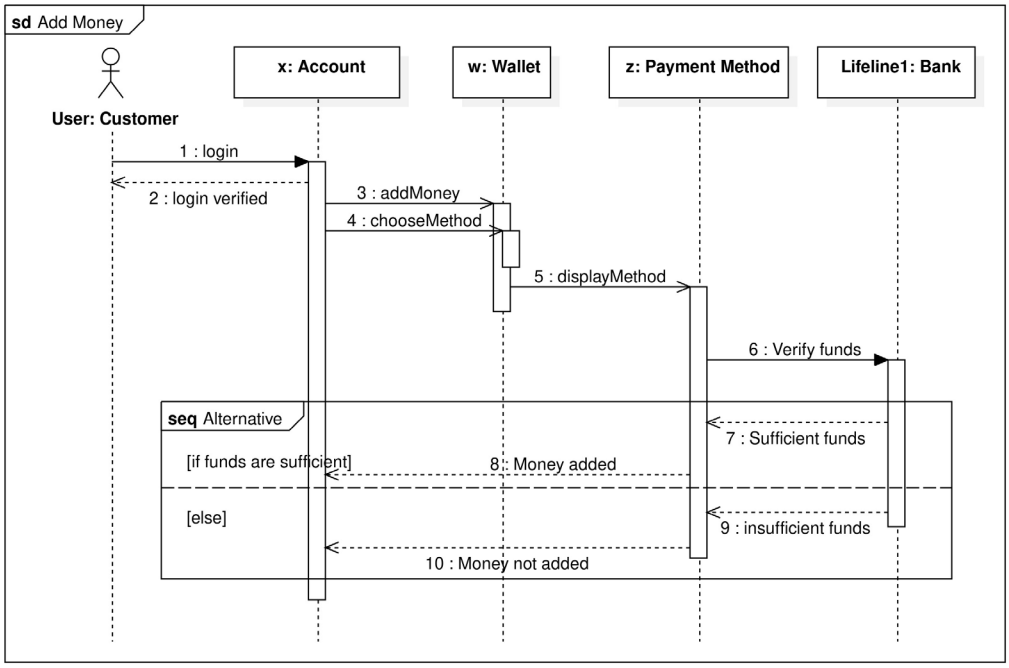
1. *One* account has *one* wallet
2. *One* account has *many* receipts
3. *Each* online order has *one* receipt

AGGREGATIONS

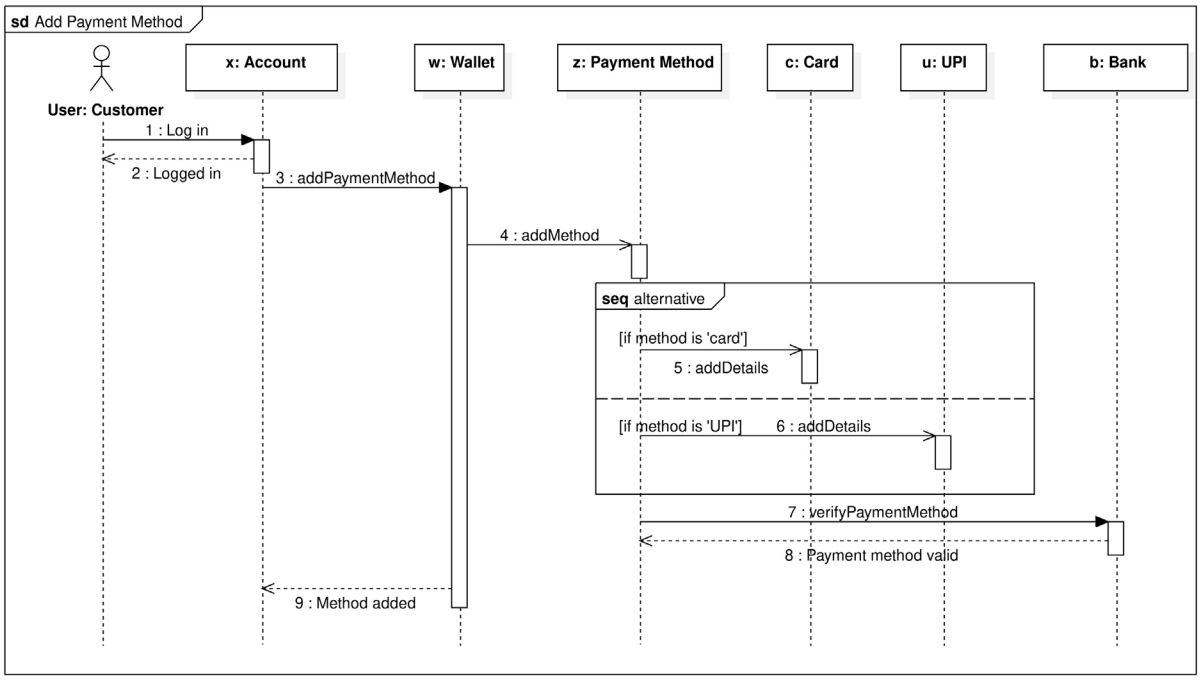
1. *Each* receipt has *one or many* products
2. *One* wallet has *many* payment methods
3. *Each* Store has *one or many* proximity sensors
4. *Each* store has *one or many* cameras
5. *Each* store has *one or many* products

SEQUENCE DIAGRAMS

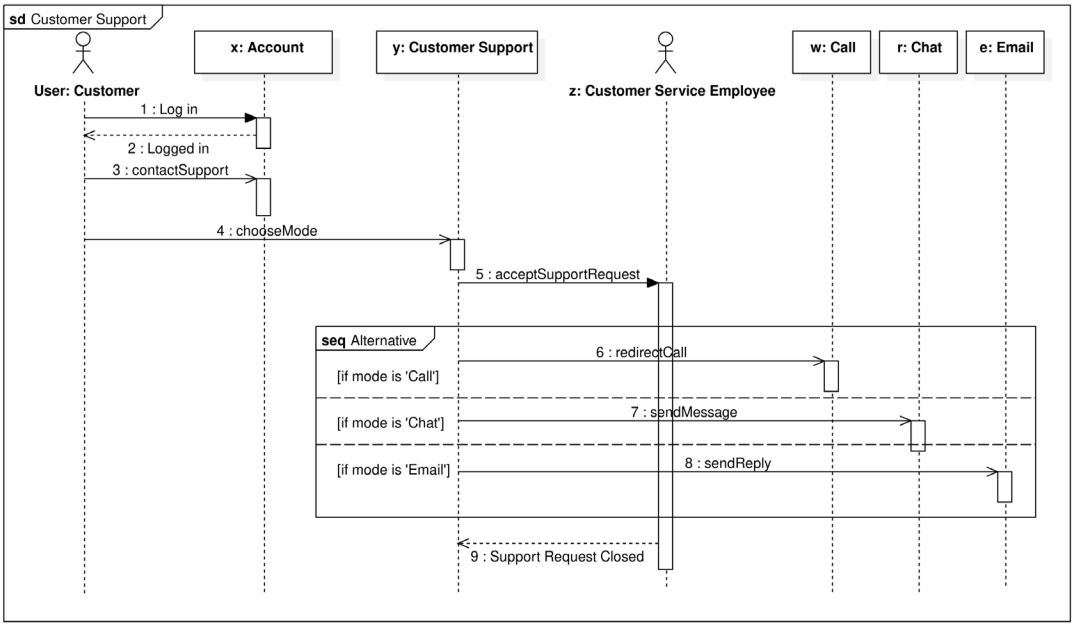
Add Money:



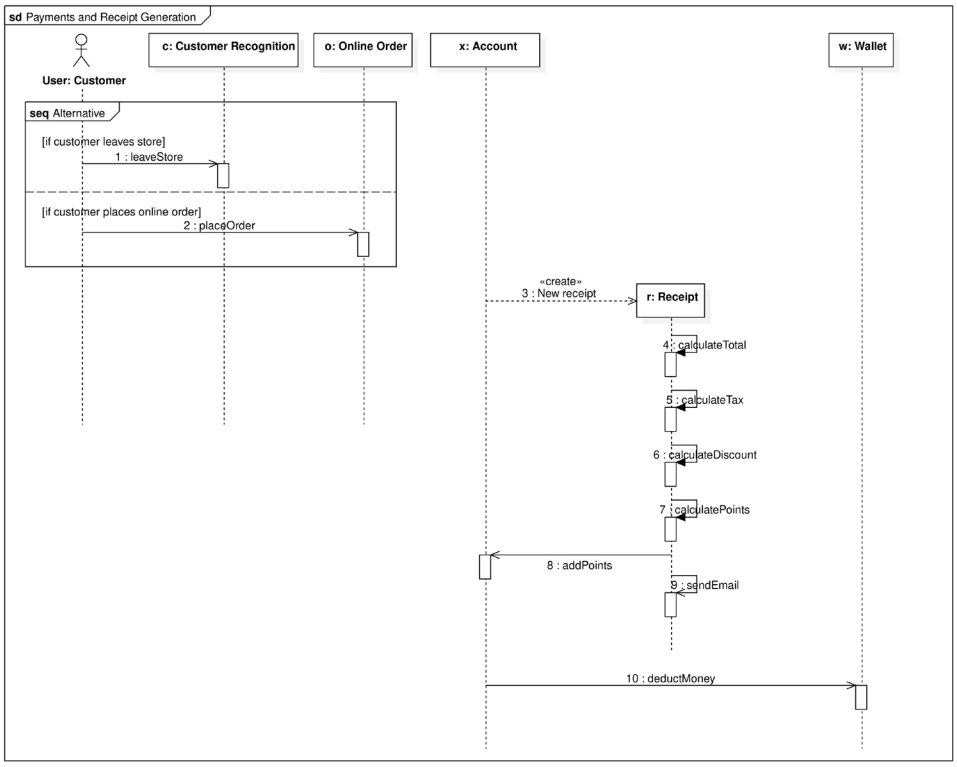
Add Payment Method:



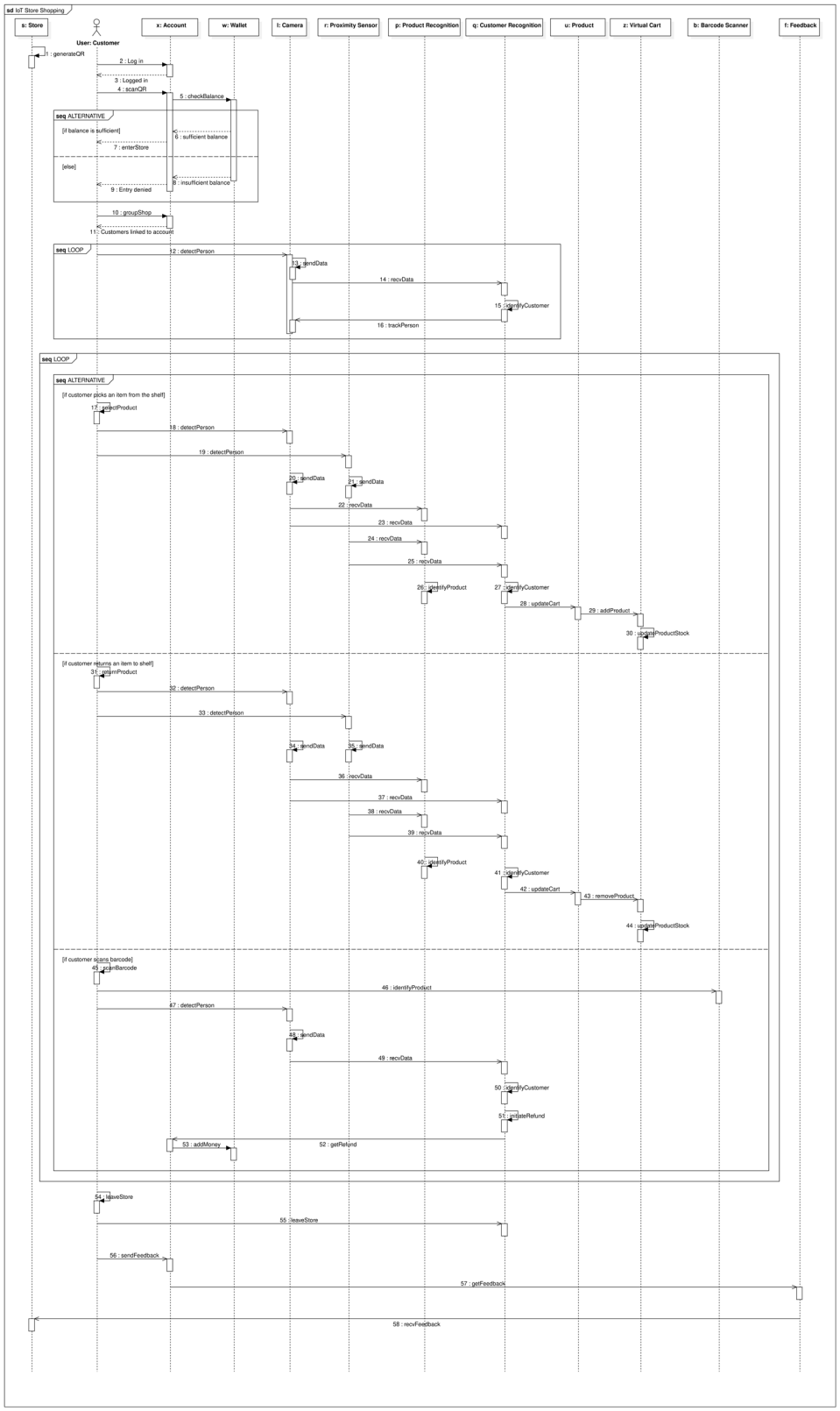
Customer Support:



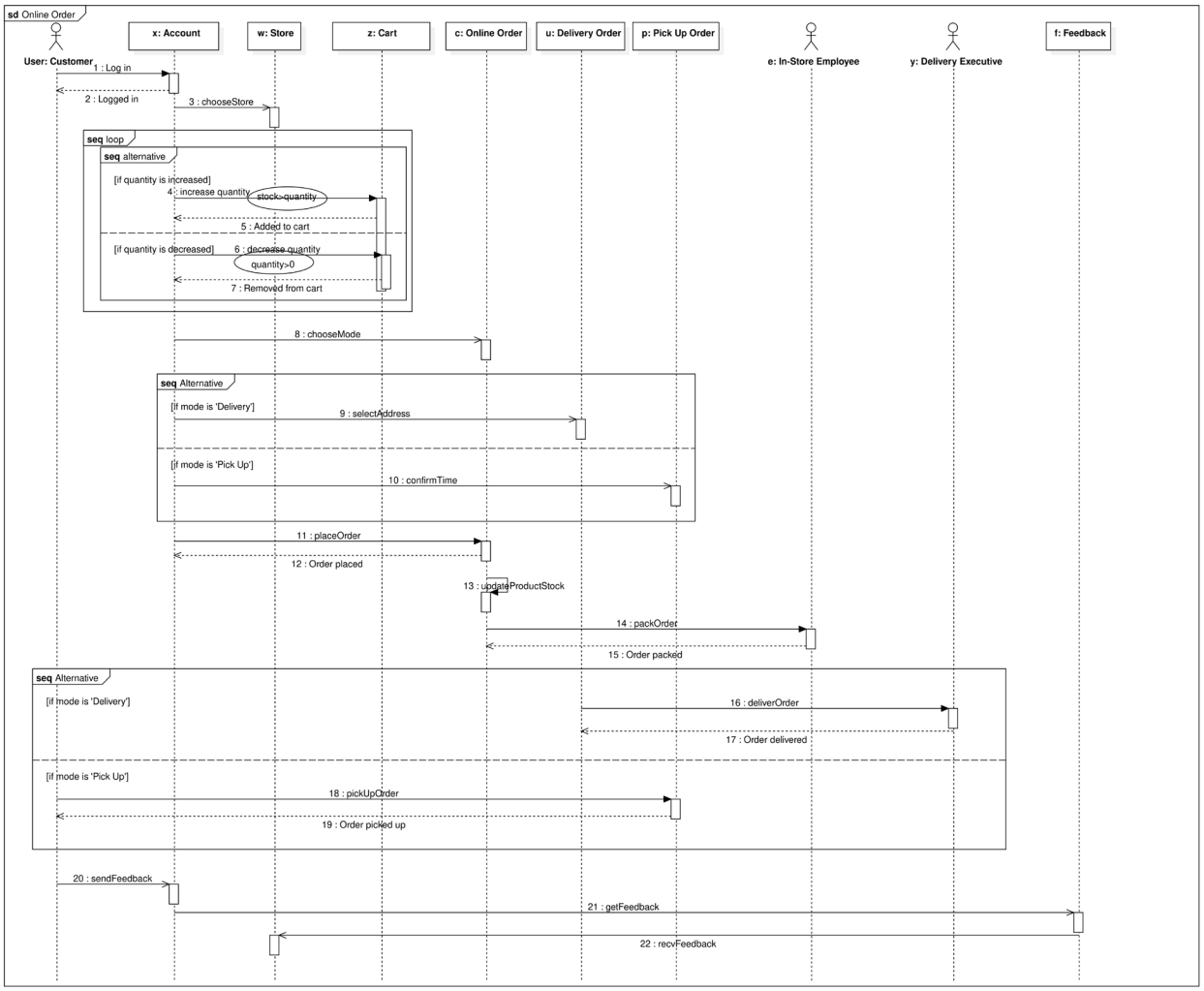
Payment and Receipt Generation:



IoT Store Shopping:

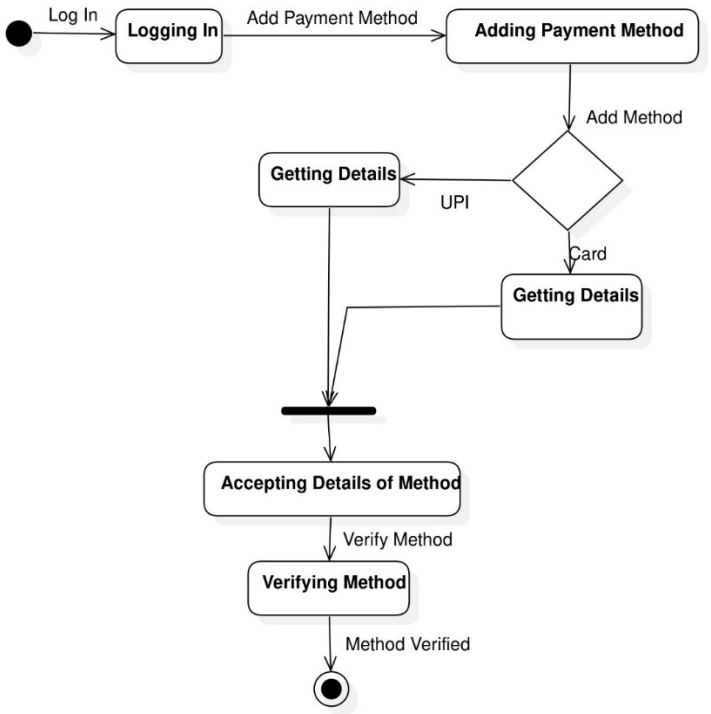


Online Order:

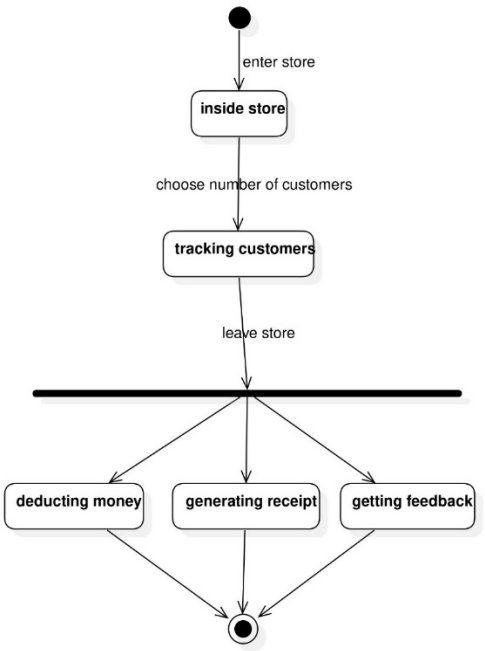


STATE MACHINE DIAGRAMS

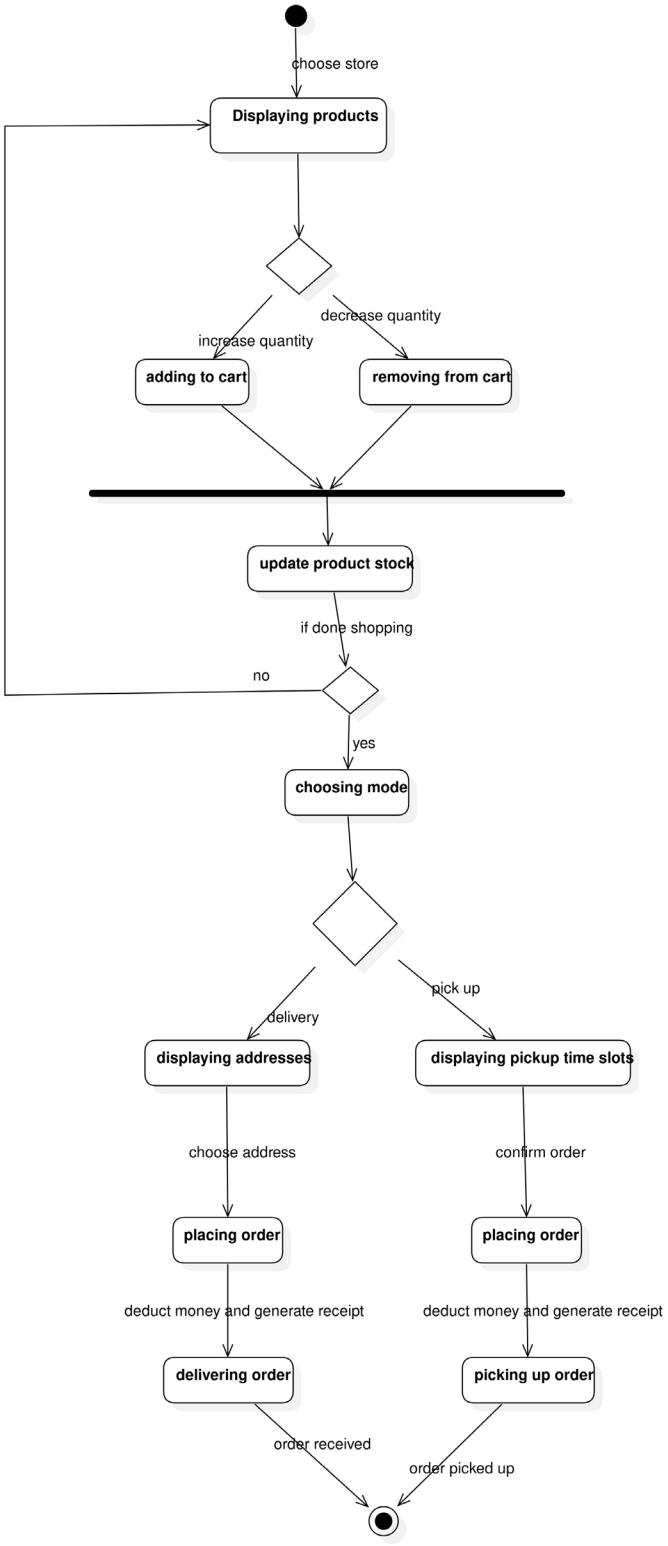
Add Payment Method:



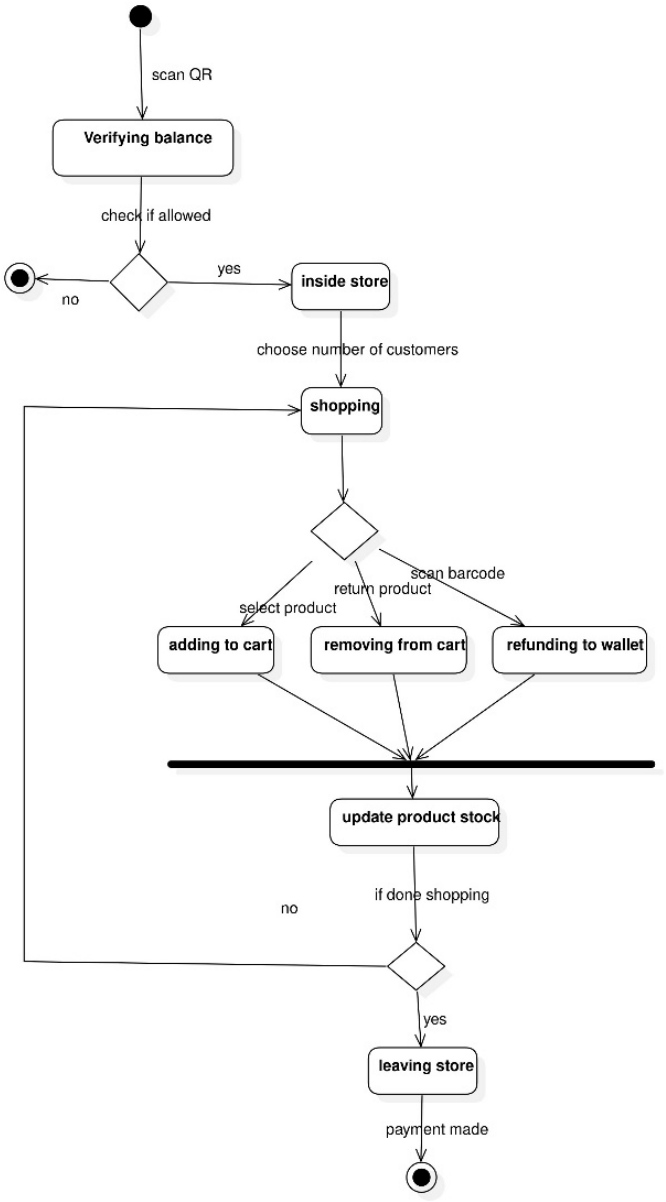
Recognition:



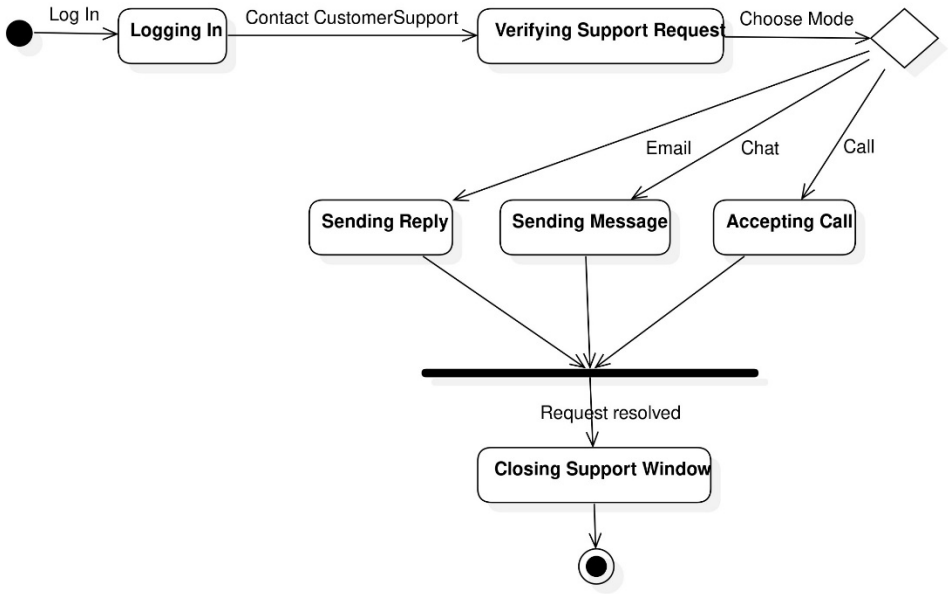
Online Store:



IOT Store:

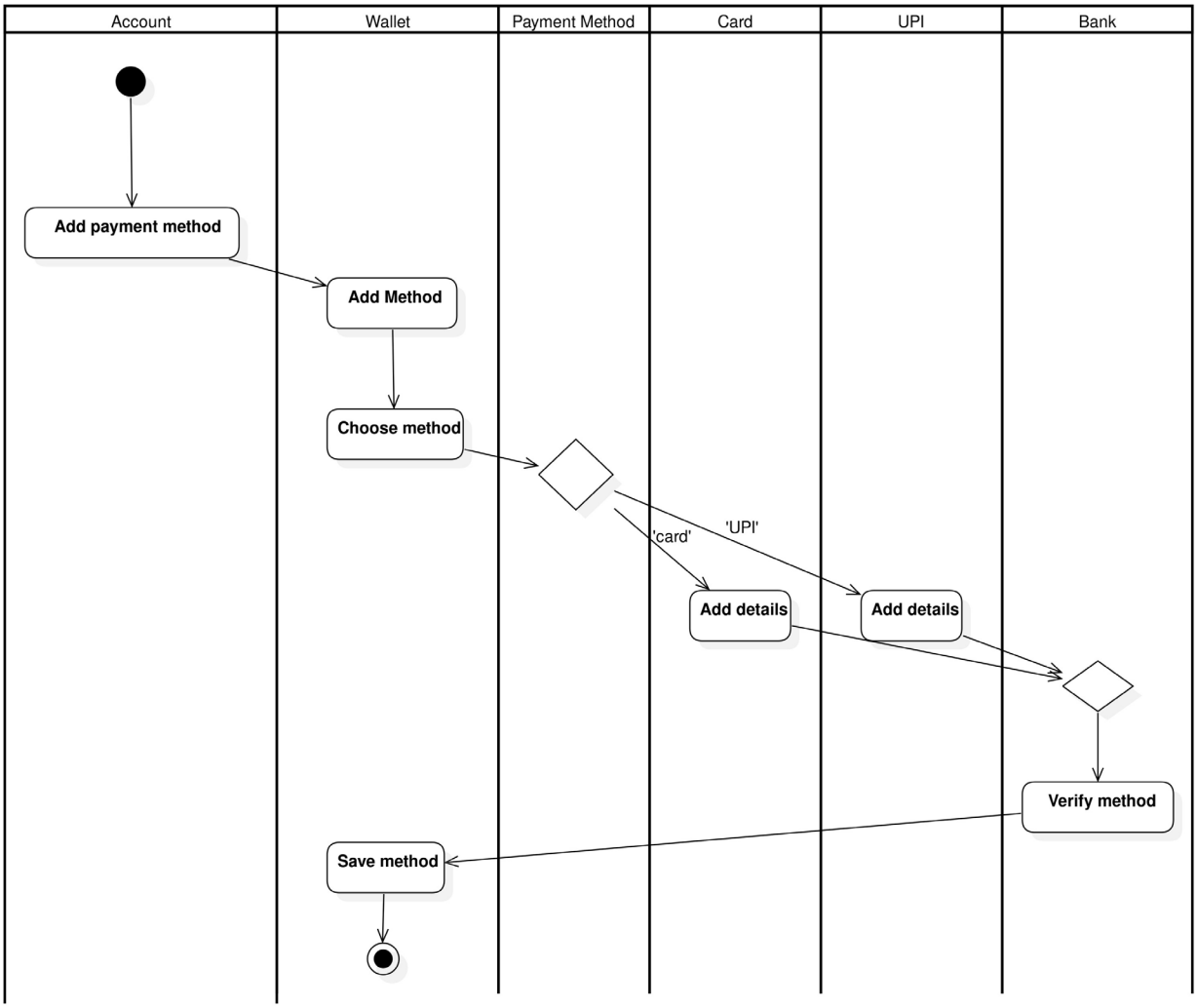

scan QR
Verifying balance
check if allowed
yes
no
inside store
choose number of customers
shopping
scan barcode
return product
select product
adding to cart
removing from cart
refunding to wallet
update product stock
if done shopping
no
yes
leaving store
payment made

Customer Support:

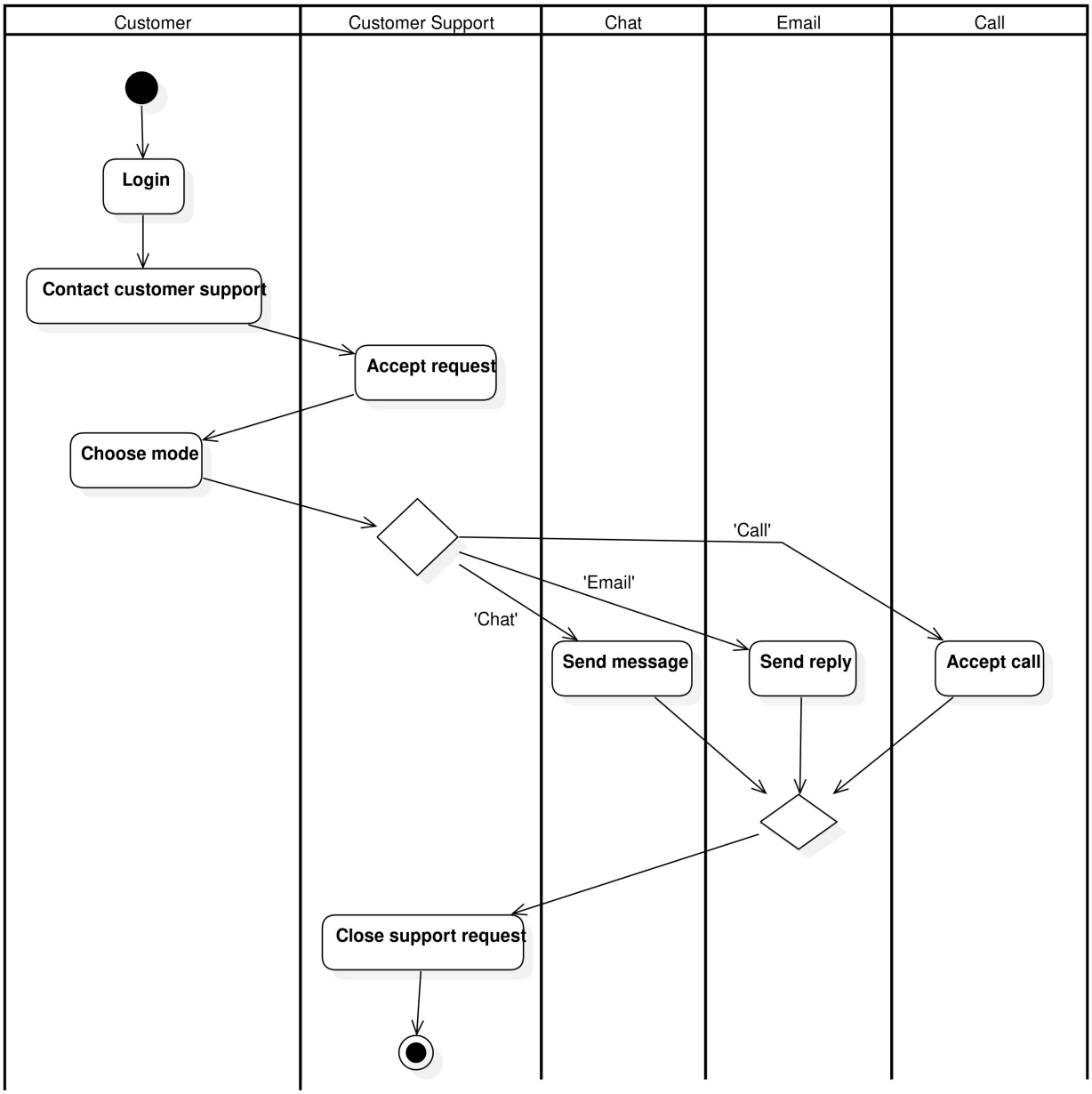


ACTIVITY DIAGRAMS

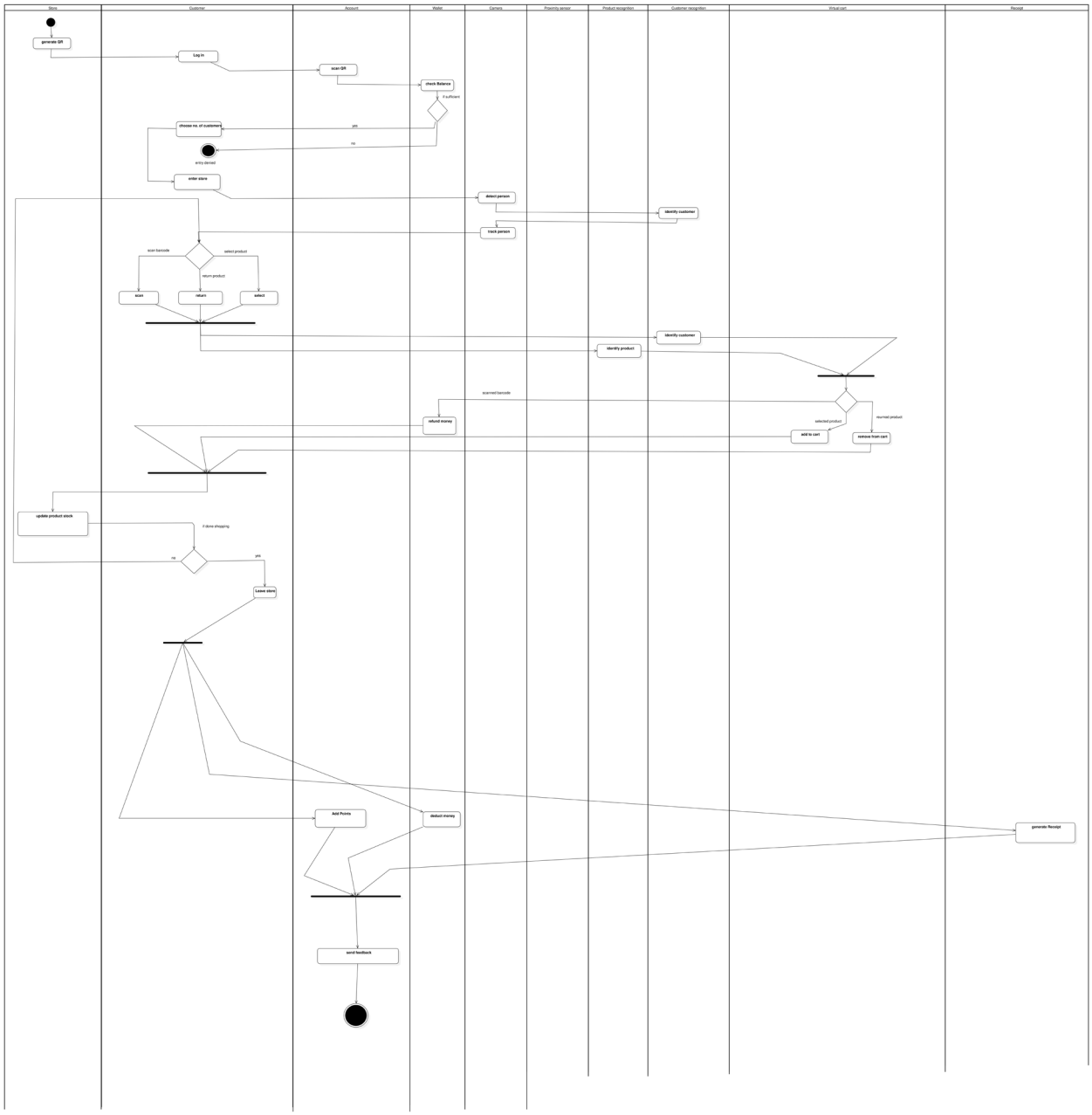
Add Payment Method:



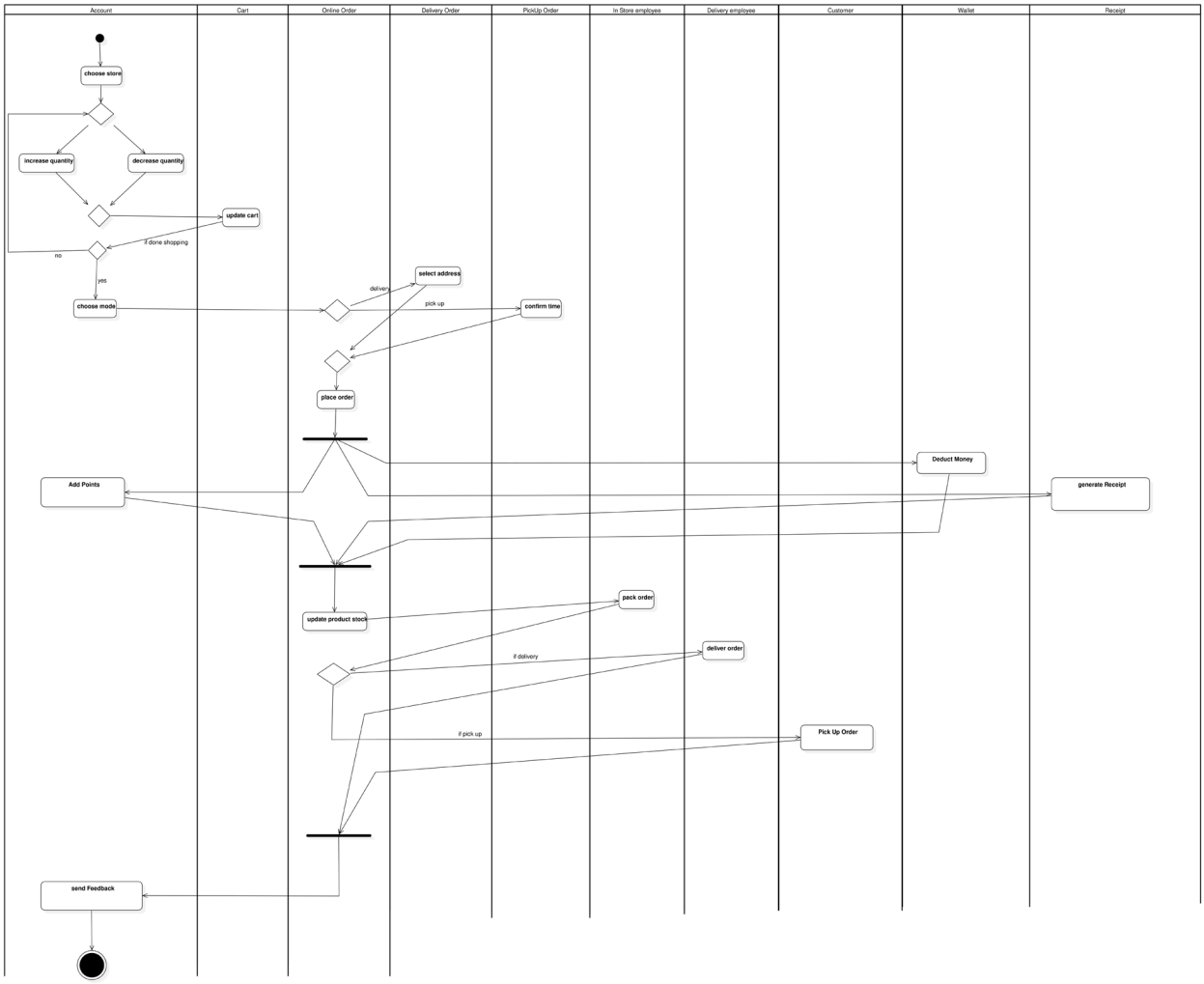
Customer Support:



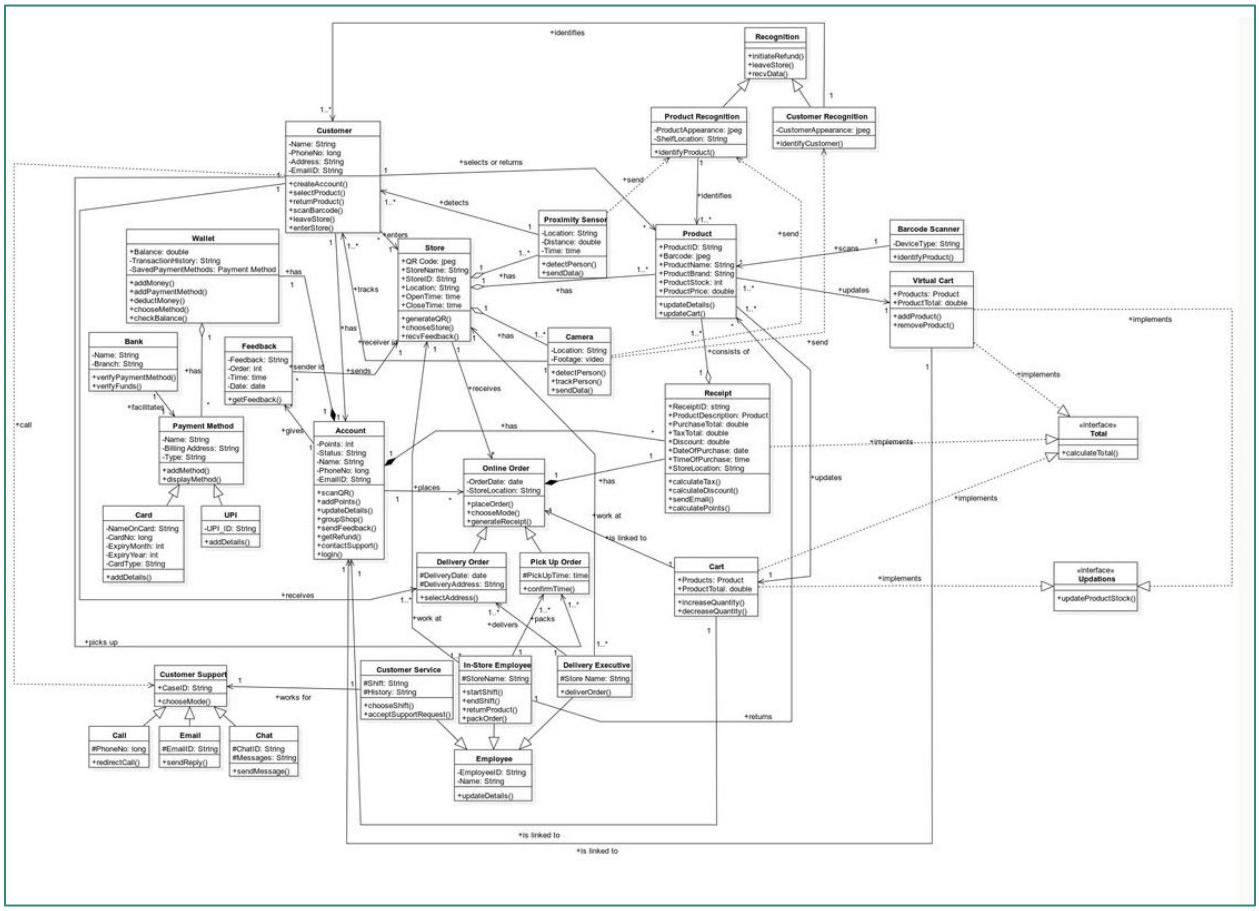
IOT Store:


Store
Customer
Account
Wallet
Camera
Proximity sensor
Product recognition
Customer recognition
Virtual cart
Receipt
generate QR
Log in
scan QR
check Balance
if sufficient
yes
no
choose no. of customers
entry denied
enter store
detect person
identify customer
track person
scan barcode
select product
return product
scan
return
select
identify customer
identify product
scanned barcode
selected product
returned product
add to cart
remove from cart
refund money
update product stock
if done shopping
no
yes
Leave store
Add Points
deduct money
generate Receipt
send feedback

Online Store:



CODE GENERATION



ACCOUNT CLASS:

```
import java.util.*;

/**
 *
 */
public class Account {

    /**
     * Default constructor
     */
    public Account() {
    }

    /**
```

```
 *
 */
private int Points;

/**
 *
 */
private String Status;

/**
 *
 */
private String Name;

/**
 *
 */
private long PhoneNo;

/**
 *
 */
private String EmailID;

/**
 *
 */
private Wallet has;

private Set<Receipt> receipts;




/**
 *
 */
public void scanQR() {
    // TODO implement here
}

/**
 *
 */
```

```
public void addPoints() {
    // TODO implement here
}

/**
 *
 */
public void updateDetails() {
    // TODO implement here
}

/**
 *
 */
public void groupShop() {
    // TODO implement here
}

/**
 *
 */
public void sendFeedback() {
    // TODO implement here
}

/**
 *
 */
public void getRefund() {
    // TODO implement here
}

/**
 *
 */
public void contactSupport() {
    // TODO implement here
}

/**
 *
 */
public void login() {
    // TODO implement here
```

```
    }

}
```

BANK CLASS:

```
import java.util.*;

/**
 *
 */
public class Bank {

    /**
     * Default constructor
     */
    public Bank() {
    }

    /**
     *
     */
    private String Name;

    /**
     *
     */
    private String Branch;


    /**
     *
     */
    public void verifyPaymentMethod() {
        // TODO implement here
    }

    /**
     *
     */
    public void verifyFunds() {
        // TODO implement here
    }
```

```
}
```

BARCODE SCANNER CLASS:

```
import java.util.*;

/**
 *
 */
public class BarcodeScanner {

    /**
     * Default constructor
     */
    public BarcodeScanner() {
    }

    /**
     *
     */
    private String DeviceType;


    /**
     *
     */
    public void identifyProduct() {
        // TODO implement here
    }

}
```

CALL CLASS:

```
import java.util.*;

/**
 *
 */
public class Call extends CustomerSupport {

    /**
     * Default constructor
     */
```

```
    public Call() {
    }

    /**
     *
     */
    protected long PhoneNo;

    /**
     *
     */
    public void redirectCall() {
        // TODO implement here
    }

}
```

CAMERA CLASS:

```
import java.util.*;

/**
 *
 */
public class Camera {

    /**
     * Default constructor
     */
    public Camera() {
    }

    /**
     *
     */
    private String Location;

    /**
     *
     */
    private video Footage;
```

```
    /**
     *
     */
    public void detectPerson() {
        // TODO implement here
    }

    /**
     *
     */
    public void trackPerson() {
        // TODO implement here
    }

    /**
     *
     */
    public void sendData() {
        // TODO implement here
    }

}
```

CARD CLASS:

```
import java.util.*;

/**
 *
 */
public class Card extends Payment Method {

    /**
     * Default constructor
     */
    public Card() {
    }

    /**
     *
     */
    private String NameOnCard;
```

```
    /**
     *
     */
    private long CardNo;

    /**
     *
     */
    private int ExpiryMonth;

    /**
     *
     */
    private int ExpiryYear;

    /**
     *
     */
    private String CardType;

    /**
     *
     */
    public void addDetails() {
        // TODO implement here
    }

}
```

CART CLASS:

```
import java.util.*;

/**
 *
 */
public class Cart implements Total, Updations {

    /**
     * Default constructor
     */
    public Cart() {
```

```
    }

    /**
     *
     */
    public Product Products;

    /**
     *
     */
    public double ProductTotal;




    /**
     *
     */
    public void increaseQuantity() {
        // TODO implement here
    }

    /**
     *
     */
    public void decreaseQuantity() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

}
```

CHAT CLASS:

```
import java.util.*;

/**
 *
 */
public class Chat extends CustomerSupport {

    /**
     * Default constructor
     */
    public Chat() {
    }

    /**
     *
     */
    protected String ChatID;

    /**
     *
     */
    protected String Messages;

    /**
     *
     */
    public void sendMessage() {
        // TODO implement here
    }

}
```

CUSTOMER CLASS:

```
import java.util.*;

/**
 *
 */
public class Customer {

    /**
```

```
 * Default constructor
 */
public Customer() {
}

/**
 *
 */
private String Name;

/**
 *
 */
private long PhoneNo;

/**
 *
 */
private String Address;

/**
 *
 */
private String EmailID;









/**
 *
 */
public void createAccount() {
    // TODO implement here
}

/**
 *
 */
public void selectProduct() {
    // TODO implement here
}
```

```
    /**
     *
     */
    public void returnProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void scanBarcode() {
        // TODO implement here
    }

    /**
     *
     */
    public void leaveStore() {
        // TODO implement here
    }

    /**
     *
     */
    public void enterStore() {
        // TODO implement here
    }

}
```

CUSTOMER RECOGNITION CLASS:

```
import java.util.*;

/**
 *
 */
public class CustomerRecognition extends Recognition {

    /**
     * Default constructor
     */
    public CustomerRecognition() {
    }
```

```
    /**
     *
     */
    private jpeg CustomerAppearance;



    /**
     *
     */
    public void identifyCustomer() {
        // TODO implement here
    }

}
```

CUSTOMER SERVICE CLASS:

```
import java.util.*;

/**
 *
 */
public class CustomerService extends Employee {

    /**
     * Default constructor
     */
    public Customer Service() {
    }

    /**
     *
     */
    protected String Shift;

    /**
     *
     */
    protected String History;


    /**
     *
```

```
     */
    public void chooseShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void acceptSupportRequest() {
        // TODO implement here
    }

}
```

CUSTOMER SUPPORT CLASS:

```
import java.util.*;

/**
 *
 */
public class CustomerSupport {

    /**
     * Default constructor
     */
    public CustomerSupport() {
    }

    /**
     *
     */
    public String CaseID;

    /**
     *
     */
    public void chooseMode() {
        // TODO implement here
    }

}
```

DELIVERY EXECUTIVE CLASS:

```
import java.util.*;

/**
 *
 */
public class DeliveryExecutive extends Employee {

    /**
     * Default constructor
     */
    public DeliveryExecutive() {
    }

    /**
     *
     */
    protected String Store Name;



    /**
     *
     */
    public void deliverOrder() {
        // TODO implement here
    }

}
```

DELIVERY ORDER CLASS:

```
import java.util.*;

/**
 *
 */
public class DeliveryOrder extends OnlineOrder {

    /**
     * Default constructor
     */
```

```
    public DeliveryOrder() {
    }

    /**
     *
     */
    protected date DeliveryDate;

    /**
     *
     */
    protected String DeliveryAddress;

    /**
     *
     */
    public void selectAddress() {
        // TODO implement here
    }

}
```

EMAIL CLASS:

```
import java.util.*;

/**
 *
 */
public class Email extends CustomerSupport {

    /**
     * Default constructor
     */
    public Email() {
    }

    /**
     *
     */
    protected String EmailID;

    /**
     *
```

```
     */
    public void sendReply() {
        // TODO implement here
    }

}
```

EMPLOYEE CLASS:

```
import java.util.*;

/**
 *
 */
public class Employee {

    /**
     * Default constructor
     */
    public Employee() {
    }

    /**
     *
     */
    private String EmployeeID;

    /**
     *
     */
    private String Name;

    /**
     *
     */
    public void updateDetails() {
        // TODO implement here
    }

}
```

FEEDBACK CLASS:

```
import java.util.*;
```

```
/**
 *
 */
public class Feedback {

    /**
     * Default constructor
     */
    public Feedback() {
    }

    /**
     *
     */
    private String Feedback;

    /**
     *
     */
    private int Order;

    /**
     *
     */
    private time Time;

    /**
     *
     */
    private date Date;

    /**
     *
     */
    public Store receiver id;

    /**
     *
     */
    public void getFeedback() {
        // TODO implement here
    }
```

```
}
```

IN-STORE EMPLOYEE CLASS:

```
import java.util.*;

/**
 *
 */
public class InStoreEmployee extends Employee {

    /**
     * Default constructor
     */
    public InStoreEmployee() {
    }

    /**
     *
     */
    protected String StoreName;



    /**
     *
     */
    public void startShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void endShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void returnProduct() {
        // TODO implement here
```

```
    }

    /**
     *
     */
    public void packOrder() {
        // TODO implement here
    }

}
```

ONLINE ORDER CLASS:

```
import java.util.*;

/**
 *
 */
public class OnlineOrder {

    /**
     * Default constructor
     */
    public OnlineOrder() {
    }

    /**
     *
     */
    private date OrderDate;

    /**
     *
     */
    private String StoreLocation;


    private Receipt recpt;
    /**
     *
     */
    public void placeOrder() {
        // TODO implement here
```

```
    }

    /**
     *
     */
    public void chooseMode() {
        // TODO implement here
    }

}
```

PAYMENT METHOD CLASS:

```
import java.util.*;

/**
 *
 */
public class PaymentMethod {

    /**
     * Default constructor
     */
    public PaymentMethod() {
    }

    /**
     *
     */
    private String Name;

    /**
     *
     */
    private String BillingAddress;

    /**
     *
     */
    private String Type;


    /**
     *
```

```
     */
    public void addMethod() {
        // TODO implement here
    }

    /**
     *
     */
    public void displayMethod() {
        // TODO implement here
    }

}
```

PICK UP CLASS:

```
import java.util.*;

/**
 *
 */
public class PickUpOrder extends OnlineOrder {

    /**
     * Default constructor
     */
    public PickUpOrder() {
    }

    /**
     *
     */
    protected time PickUpTime;

    /**
     *
     */
    public void confirmTime() {
        // TODO implement here
    }

}
```

PRODUCT CLASS:

```
import java.util.*;

/**
 *
 */
public class Product {

    /**
     * Default constructor
     */
    public Product() {
    }

    /**
     *
     */
    public String ProductID;

    /**
     *
     */
    public jpeg Barcode;

    /**
     *
     */
    public String ProductName;

    /**
     *
     */
    public String ProductBrand;

    /**
     *
     */
    public int ProductStock;

    /**
     *
     */
```

```
    public double ProductPrice;




    /**
     *
     */
    public void updateDetails() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateCart() {
        // TODO implement here
    }

}
```

PRODUCT RECOGNITION CLASS:

```
import java.util.*;

/**
 *
 */
public class ProductRecognition extends Recognition {

    /**
     * Default constructor
     */
    public ProductRecognition() {
    }

    /**
     *
     */
    private jpeg ProductAppearance;

    /**
     *
```

```
     */
    private String ShelfLocation;


    /**
     *
     */
    public void identifyProduct() {
        // TODO implement here
    }

}
```

PROXIMITY SENSOR CLASS:

```
import java.util.*;

/**
 *
 */
public class ProximitySensor {

    /**
     * Default constructor
     */
    public ProximitySensor() {
    }

    /**
     *
     */
    private String Location;

    /**
     *
     */
    private double Distance;

    /**
     *
     */
    private time Time;
```

```
    /**
     *
     */
    public void detectPerson() {
        // TODO implement here
    }

    /**
     *
     */
    public void sendData() {
        // TODO implement here
    }

}
```

RECEIPT CLASS:

```
import java.util.*;

/**
 *
 */
public class Receipt implements Total {

    /**
     * Default constructor
     */
    public Receipt() {
    }

    /**
     *
     */
    public string ReceiptID;

    /**
     *
     */
```

```
public Product ProductDescription;

/**
 *
 */
public double PurchaseTotal;

/**
 *
 */
public double TaxTotal;

/**
 *
 */
public double Discount;

/**
 *
 */
public date DateOfPurchase;

/**
 *
 */
public time TimeOfPurchase;

/**
 *
 */
public String StoreLocation;




/**
 *
 */
public void calculateTax() {
    // TODO implement here
}

/**
```

```
     *
     */
    public void calculateDiscount() {
        // TODO implement here
    }

    /**
     *
     */
    public void sendEmail() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculatePoints() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

}
```

RECOGNITION CLASS:

```
import java.util.*;

/**
 *
 */
public class Recognition {

    /**
     * Default constructor
     */
    public Recognition() {
```

```
    }

    /**
     *
     */
    public void initiateRefund() {
        // TODO implement here
    }

    /**
     *
     */
    public void leaveStore() {
        // TODO implement here
    }

    /**
     *
     */
    public void recvData() {
        // TODO implement here
    }

}
```

STORE CLASS:

```
import java.util.*;

/**
 *
 */
public class Store {

    /**
     * Default constructor
     */
    public Store() {
    }

    /**
     *
     */
```

```
public jpeg QRCode;

/**
 *
 */
public String StoreName;

/**
 *
 */
public String StoreID;

/**
 *
 */
public String Location;

/**
 *
 */
public time OpenTime;

/**
 *
 */
public time CloseTime;









/**
 *
 */
public void generateQR() {
    // TODO implement here
}

/**
 *
 */
```

```
    public void chooseStore() {
        // TODO implement here
    }

    /**
     *
     */
    public void recvFeedback() {
        // TODO implement here
    }

}
```

TOTAL INTERFACE:

```
import java.util.*;

/**
 *
 */
public interface Total {

    /**
     *
     */
    public void calculateTotal();

}
```

UPDATIONS INTERFACE:

```
import java.util.*;

/**
 *
 */
public interface Updations {

    /**
     *
     */
    public void updateProductStock();

}
```

UPI CLASS:

```
import java.util.*;

/**
 *
 */
public class UPI extends PaymentMethod {

    /**
     * Default constructor
     */
    public UPI() {
    }

    /**
     *
     */
    private String UPI_ID;

    /**
     *
     */
    public void addDetails() {
        // TODO implement here
    }

}
```

VIRTUAL CART CLASS:

```
import java.util.*;

/**
 *
 */
public class VirtualCart implements Total, Updations {

    /**
     * Default constructor
     */
    public VirtualCart() {
    }
```

```
    /**
     *
     */
    public Product Products;

    /**
     *
     */
    public double ProductTotal;


    /**
     *
     */
    public void addProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void removeProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

}
```

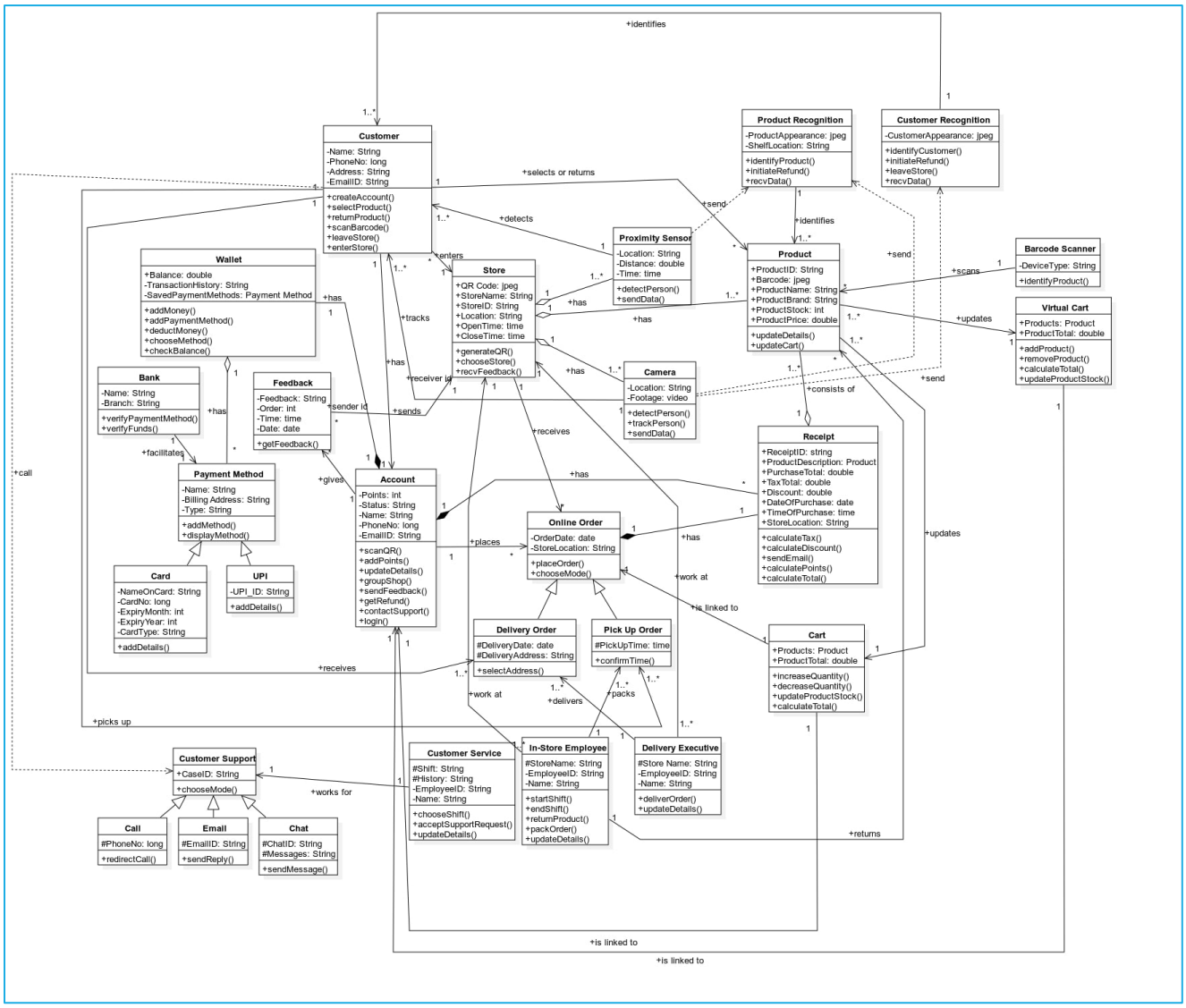
WALLET CLASS:

```
import java.util.*;

/**
 *
 */
public class Wallet {

    /**
     * Default constructor
     */
    public Wallet() {
    }

    /**
     *
     */
    public double Balance;

    /**
     *
     */
    private String TransactionHistory;

    /**
     *
     */
    private Payment Method SavedPaymentMethods;




    /**
     *
     */
    public void addMoney() {
        // TODO implement here
    }

    /**
     *
     */
    public void addPaymentMethod() {
```

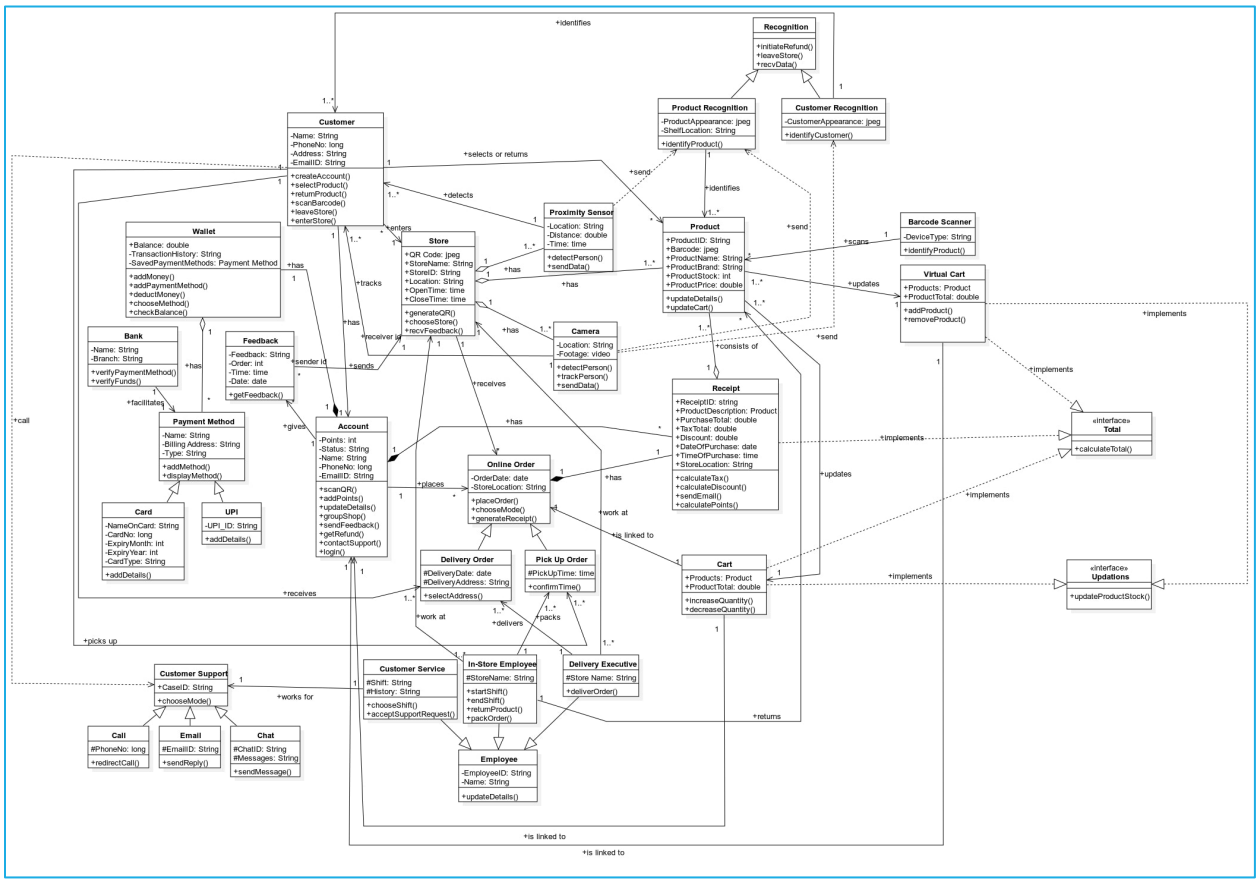
```
    // TODO implement here
}

/**
 *
 */
public void deductMoney() {
    // TODO implement here
}

/**
 *
 */
public void chooseMethod() {
    // TODO implement here
}

/**
 *
 */
public void checkBalance() {
    // TODO implement here
}
```

CODE OPTIMISATION USING DESIGN PATTERNS

Initial Diagram:



Optimized Diagram:



Improvements:

1 Online Order Class

As seen in the code below, online order class now has the responsibility of generating the receipt. It is a creator for class Receipt because it is an information expert for the initialization details needed to create a receipt.

Old Code:

```
import java.util.*;


/**
 *
 */
public class OnlineOrder {


    /**
     * Default constructor
     */
    public OnlineOrder() {
    }


    /**
     *
     */
    private date OrderDate;


    /**
     *
     */
    private String StoreLocation;
```

```
    /**
     *
     */
    public void placeOrder() {
        // TODO implement here
    }

    /**
     *
     */
    public void chooseMode() {
        // TODO implement here
    }

}
```

New Code:

```
import java.util.*;

/**
 *
 */
public class OnlineOrder {

    /**
     * Default constructor
     */
```

```
public OnlineOrder() {
}

/**
 *
 */
private date OrderDate;

/**
 *
 */
private String StoreLocation;




/**
 *
 */
public void placeOrder() {
    // TODO implement here
}

/**
 *
 */
public void chooseMode() {
    // TODO implement here
}
```

```
    /**
     *
     */
    public void generateReceipt() {
        // TODO implement here
    }

}
```

2 Polymorphism

We create 2 interfaces as when related alternatives or behaviors vary by type (class), we assign responsibility for the behavior—using polymorphic operations—to the types for which the behavior varies. Functions like updating product stock and calculating total differ slightly with online orders when compared to the IoT store as it may include some extra charges for delivery or packing up of orders.

Old Code:

Receipt.java:

```
import java.util.*;


/**
 *
 */
public class Receipt implements Total {


    /**
     * Default constructor
     */
    public Receipt() {
    }
```

```
    /**
     *
     */
    public string ReceiptID;

    /**
     *
     */
    public Product ProductDescription;

    /**
     *
     */
    public double PurchaseTotal;

    /**
     *
     */
    public double TaxTotal;

    /**
     *
     */
    public double Discount;

    /**
     *
     */
    public date DateOfPurchase;
```

```
/**
 *
 */
public time TimeOfPurchase;

/**
 *
 */
public String StoreLocation;
/**
 *
 */
public void calculateTax() {
    // TODO implement here
}

/**
 *
 */
public void calculateDiscount() {
    // TODO implement here
}

/**
 *
 */
public void sendEmail() {
    // TODO implement here
```

```
    }

    /**
     *
     */
    public void calculatePoints() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }
}
```

Cart.java:

```
import java.util.*;

/**
 *
 */
```

```
public class Cart{


    /**
     * Default constructor
     */
    public Cart() {
    }


    /**
     *
     */
    public Product Products;


    /**
     *
     */
    public double ProductTotal;









    /**
     *
     */
    public void increaseQuantity() {
        // TODO implement here
    }


    /**
```

```
     *
     */
    public void decreaseQuantity() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
 }
```

VirtualCart.java:

```
import java.util.*;

/**
```

```
 *
 */
public class VirtualCart{

    /**
     * Default constructor
     */
    public VirtualCart() {
    }

    /**
     *
     */
    public Product Products;

    /**
     *
     */
    public double ProductTotal;



    /**
     *
     */
    public void addProduct() {
        // TODO implement here
    }

    /**
```

```
     *
     */
    public void removeProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

}
```

New Code:

Receipt.java:

```
import java.util.*;

/**
 *
 */
```

```
public class Receipt implements Total {

    /**
     * Default constructor
     */
    public Receipt() {
    }

    /**
     *
     */
    public string ReceiptID;

    /**
     *
     */
    public Product ProductDescription;

    /**
     *
     */
    public double PurchaseTotal;

    /**
     *
     */
    public double TaxTotal;

    /**
```

```
     *
     */
    public double Discount;

    /**
     *
     */
    public date DateOfPurchase;

    /**
     *
     */
    public time TimeOfPurchase;

    /**
     *
     */
    public String StoreLocation;
    /**
     *
     */
    public void calculateTax() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateDiscount() {
```

```
        // TODO implement here
    }

    /**
     *
     */
    public void sendEmail() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculatePoints() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

}
```

Cart.java:

```
import java.util.*;
```

```
/**
 *
 */
public class Cart implements Total, Updations {

    /**
     * Default constructor
     */
    public Cart() {
    }

    /**
     *
     */
    public Product Products;

    /**
     *
     */
    public double ProductTotal;




    /**
     *
     */
    public void increaseQuantity() {
        // TODO implement here
```

```
    }

    /**
     *
     */
    public void decreaseQuantity() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

}
```

VirtualCart.java:

```
import java.util.*;

/**
```

```
 *
 */
public class VirtualCart implements Total, Updations {

    /**
     * Default constructor
     */
    public VirtualCart() {
    }

    /**
     *
     */
    public Product Products;

    /**
     *
     */
    public double ProductTotal;



    /**
     *
     */
    public void addProduct() {
        // TODO implement here
    }

    /**
```

```
     *
     */
    public void removeProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void calculateTotal() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateProductStock() {
        // TODO implement here
    }

}
```

Total.java:

```
import java.util.*;
public interface Total {
    public void calculateTotal();
}
```

Updations.java:

```
import java.util.*;
public interface Updations {
```

```
    public void updateProductStock();
}
```

3 Pure Fabrication

A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived. Class Employee and class Recognition are added so that classes with similar functionality or data members can inherit from a common class.

Old Code:

InStoreEmployee.java:

```
import java.util.*;

/**
 *
 */
public class InStoreEmployee{

    /**
     * Default constructor
     */
    public InStoreEmployee() {
    }

    /**
     *
     */
    protected String StoreName;

    /**
```

```
     *
     */
    private String EmployeeID;

    /**
     *
     */
    private String Name;









    /**
     *
     */
    public void startShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void endShift() {
        // TODO implement here
    }

    /**
     *
     */
```

```
    public void returnProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void packOrder() {
        // TODO implement here
    }

    /**
     *
     */
    public void updateDetails() {
        // TODO implement here
    }

}
```

DeliveryExecutive.java:

```
import java.util.*;

/**
 *
 */
public class DeliveryExecutive{

    /**
```

```
     * Default constructor
     */
    public DeliveryExecutive() {
    }

    /**
     *
     */
    protected String Store Name;

    /**
     *
     */
    private String EmployeeID;

    /**
     *
     */
    private String Name;



    /**
     *
     */
    public void deliverOrder() {
        // TODO implement here
    }
```

```
    /**
     *
     */
    public void updateDetails() {
        // TODO implement here
    }

}
```

CustomerService.java:

```
import java.util.*;

/**
 *
 */
public class CustomerService {

    /**
     * Default constructor
     */
    public CustomerService() {
    }

    /**
     *
     */
    protected String Shift;

    /**
```

```
     *
     */
    protected String History;

    /**
     *
     */
    private String EmployeeID;

    /**
     *
     */
    private String Name;



    /**
     *
     */
    public void chooseShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void acceptSupportRequest() {
        // TODO implement here
    }
```

```
    /**
     *
     */
    public void updateDetails() {
        // TODO implement here
    }

}
```

ProductRecognition.java:

```
import java.util.*;

/**
 *
 */
public class ProductRecognition{

    /**
     * Default constructor
     */
    public ProductRecognition() {
    }

    /**
     *
     */
    private jpeg ProductAppearance;

    /**
```

```
     *
     */
    private String ShelfLocation;



    /**
     *
     */
    public void identifyProduct() {
        // TODO implement here
    }

    /**
     *
     */
    public void initiateRefund() {
        // TODO implement here
    }

    /**
     *
     */
    public void recvData() {
        // TODO implement here
    }

}
```

CustomerRecognition.java:

```
import java.util.*;


/**
 *
 */
public class CustomerRecognition {


    /**
     * Default constructor
     */
    public CustomerRecognition() {
    }


    /**
     *
     */
    private jpeg CustomerAppearance;




    /**
     *
     */
    public void identifyCustomer() {
        // TODO implement here
    }


    /**
     *
```

```
     */
    public void initiateRefund() {
        // TODO implement here
    }

    /**
     *
     */
    public void leaveStore() {
        // TODO implement here
    }

    /**
     *
     */
    public void recvData() {
        // TODO implement here
    }

}
```

New Code:
Employee.java:

```
import java.util.*;

/**
 *
 */
public class Employee {
```

```
    /**
     * Default constructor
     */
    public Employee() {
    }

    private String EmployeeID;

    private String Name;

    public void updateDetails() {
        // TODO implement here
    }

}
```

InStoreEmployee.java:

```
import java.util.*;

/**
 *
 */
public class InStoreEmployee extends Employee {

    /**
     * Default constructor
     */
    public InStoreEmployee() {
```

```
    }

    /**
     *
     */
    protected String StoreName;




    /**
     *
     */
    public void startShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void endShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void returnProduct() {
        // TODO implement here
```

```
    }

    /**
     *
     */
    public void packOrder() {
        // TODO implement here
    }

}
```

DeliveryExecutive.java:

```
import java.util.*;

/**
 *
 */
public class DeliveryExecutive extends Employee {

    /**
     * Default constructor
     */
    public DeliveryExecutive() {
    }

    /**
     *
     */
    protected String StoreName;
```

```
    /**
     *
     */
    public void deliverOrder() {
        // TODO implement here
    }

}
```

CustomerService.java:

```
import java.util.*;

/**
 *
 */
public class CustomerService extends Employee {

    /**
     * Default constructor
     */
    public CustomerService() {
    }

    /**
     *
```

```
     */
    protected String Shift;

    /**
     *
     */
    protected String History;



    /**
     *
     */
    public void chooseShift() {
        // TODO implement here
    }

    /**
     *
     */
    public void acceptSupportRequest() {
        // TODO implement here
    }

}
```

Recognition.java:

```
import java.util.*;

/**
```

```
 *
 */
public class Recognition {

    /**
     * Default constructor
     */
    public Recognition() {
    }

    /**
     *
     */
    public void initiateRefund() {
        // TODO implement here
    }

    /**
     *
     */
    public void leaveStore() {
        // TODO implement here
    }

    /**
     *
     */
    public void recvData() {
        // TODO implement here
```

```
    }


}
```

ProductRecognition.java:

```
import java.util.*;

/**
 *
 */
public class ProductRecognition extends Recognition {

    /**
     * Default constructor
     */
    public ProductRecognition() {
    }

    /**
     *
     */
    private jpeg ProductAppearance;

    /**
     *
     */
    private String ShelfLocation;
```

```
    /**
     *
     */
    public void identifyProduct() {
        // TODO implement here
    }

}
```

CustomerRecognition.java:

```
import java.util.*;

/**
 *
 */
public class CustomerRecognition extends Recognition {

    /**
     * Default constructor
     */
    public CustomerRecognition() {
    }

    /**
     *
     */
    private jpeg CustomerAppearance;
```

```
    /**
     *
     */
    public void identifyCustomer() {
        // TODO implement here
    }

}
```