

# System Design Document

---

POINT OF SALE SYSTEM FOR A SMALL GROCERY STORE

Aaron Banahene (E4053947)  
Anas Karoo (E4118091)  
Marcello Gold (E4116721)

# OVERVIEW

## TABLE OF CONTENTS

1.	OVERVIEW.....	1
1.1	PURPOSE.....	5
1.2	SYSTEM OVERVIEW.....	5
1.3	CONTEXT.....	5
2.	SYSTEM REQUIREMENTS.....	6
2.1	TABLE OF FUNCTIONAL REQUIREMENTS.....	6
2.2	TABLE OF NON-FUNCTIONAL REQUIREMENTS.....	8
2.3	HIGH PRIORITY REQUIREMENTS.....	8
2.3.1	REQUIREMENT 2.1.....	8
2.3.2	REQUIREMENT 3.2.....	9
2.3.3	REQUIREMENT 4.....	9
2.3.4	REQUIREMENT 7.....	9
2.4	REQUIREMENT ASSUMPTIONS AND CONSTRAINTS.....	10
2.4.1	ASSUMPTIONS.....	10
2.4.2	CONSTRAINTS.....	11
3.	SYSTEM DESIGN.....	12
3.1	GROCERY STORE CHECKOUT USE CASE DIAGRAM.....	12
3.1.1	UPDATE INVENTORY.....	13
3.1.2	CALCULATE TOTAL.....	13
3.1.3	VALIDATE LOYALTY CARD.....	14
3.1.4	APPLY DISCOUNTS.....	14
3.1.5	PROCESS PAYMENT.....	15
3.2	GROCERY STORE CHECKOUT CLASS DIAGRAM.....	16
3.2.1	CUSTOMER CLASS.....	17
3.2.1.1	FIELDS.....	17
3.2.2	CHECKOUT CLASS.....	17
3.2.2.1	FUNCTIONS.....	17
3.2.3	CUSTOMER DATABASE CLASS.....	18
3.2.3.1	FIELDS.....	18
3.2.4	LOYALTY CARD CLASS.....	18
3.2.4.1	FIELDS.....	18
3.2.4.2	FUNCTIONS.....	18
3.2.5	PAYMENT CLASS.....	19
3.2.5.1	FIELDS.....	19
3.2.5.2	FUNCTIONS.....	19
3.2.6	PRODUCT CLASS.....	19
3.2.6.1	FIELDS.....	20
3.2.6.2	FUNCTIONS.....	20
3.2.7	INVENTORY CLASS.....	20
3.2.7.1	FIELDS.....	21
3.2.7.2	FUNCTIONS.....	21
3.3	GROCERY STORE CHECKOUT SEQUENCE DIAGRAM.....	22
3.3.1	SCAN PRODUCT.....	23
3.3.2	SCAN LOYALTY CARD.....	23
3.3.3	MAKE PAYMENT.....	24
3.4	LOYALTY CARD SYSTEM CLASS DIAGRAM.....	25
3.4.1	CUSTOMER CLASS.....	26

# OVERVIEW

3.4.1.1	FIELDS.....	27
3.4.1.2	FUNCTIONS.....	27
3.4.2	REWARD CLASS.....	28
3.4.2.1	FIELDS.....	28
3.4.2.2	FUNCTIONS.....	28
3.4.3	REWARD REDEMPTION CLASS.....	29
3.4.3.1	FIELDS.....	29
3.4.3.2	FUNCTIONS.....	29
3.4.4	TRANSACTION CLASS.....	30
3.4.4.1	FIELDS.....	30
3.4.4.2	FUNCTIONS.....	30
4.	DATABASE DESIGN.....	31
4.1	REQUIRED DATABASE INFORMATION.....	31
4.2	DATABASE RELATIONSHIPS.....	32
4.2.1	CUSTOMER – TRANSACTION RELATIONSHIP.....	32
4.2.2	CUSTOMER – REWARD REDEMPTION RELATIONSHIP.....	33
4.2.3	REWARD REDEMPTION – REWARD RELATIONSHIP.....	33
4.2.4	TRANSACTION – INVENTORY RELATIONSHIP.....	34
4.2.5	INVENTORY – STOCK RELATIONSHIP.....	34
4.2.6	STAFF TABLE.....	35
5.	TESTING.....	36
5.1	TESTING OBJECTIVES.....	36
5.2	TESTING V-MODEL.....	37
5.3	WHITE BOX VS BLACK BOX TESTING.....	39
5.4	TEST CASES.....	40
5.5	PROTOTYPE TESTING.....	43
6.	SYSTEM ARCHITECTURE.....	47
6.1	SYSTEM ARCHITECTURE OVERVIEW.....	47
6.1.1	TIER 1 PRESENTATION TIER.....	48
6.1.2	TIER 2 BUSINESS LOGIC TIER.....	48
6.1.3	TIER 3 DATA ACCESS TIER.....	48
6.1.4	TIER 4 DATABASE TIER.....	48
6.2	ARCHITECTURE JUSTIFICATION.....	49
6.2.1	SCALABILITY.....	49
6.2.2	PERFORMANCE.....	49
6.2.3	MAINTAINABILITY.....	50
6.2.4	SECURITY.....	50
6.3	ARCHITECTURE IMPLEMENTATION CONSIDERATIONS.....	51
6.3.1	TIER 1 PRESENTATION TIER CONSIDERATIONS.....	51
6.3.2	TIER 2 BUSINESS LOGIC TIER CONSIDERATIONS.....	51
6.3.3	TIER 3 DATA ACCESS TIER CONSIDERATIONS.....	51
6.3.4	TIER 4 DATABASE TIER CONSIDERATIONS.....	52
6.3.5	COST CONSIDERATIONS.....	52
7.	SYSTEM IMPLEMENTATION.....	53
7.1	HARDWARE.....	53
7.2	SOFTWARE.....	53
7.3	DATABASES.....	53
7.4	ARCHITECTURE.....	53
8.	SYSTEM MAINTENANCE.....	54
8.1	MAINTENANCE PLAN.....	54
8.2	MAINTENANCE TASKS.....	54
8.3	SECURITY.....	55

# OVERVIEW

8.4	RECOVERY & BACKUP.....	55
8.5	USER SUPPORT.....	55
9.	FUTURE ENHANCEMENTS.....	56
9.1	SCALABILITY.....	56
9.2	SELF-CHECKOUT SYSTEM.....	57
9.2.1	SELF-CHECKOUT IMPLEMENTATION.....	57
9.2.1.1	REQUIREMENTS.....	57
9.2.1.2	ADVANTAGES.....	58
9.2.1.3	DISADVANTAGES.....	58
9.2.2	SELF-CHECKOUT UI.....	59
9.2.2.1	CURRENT CHECKOUT UI.....	59
9.2.2.2	UPDATED UI.....	60
9.2.2.3	KEY DIFFERENCES.....	60
9.2.3	SELF-CHECKOUT SUMMARY.....	61
9.3	ONLINE STORE.....	62
9.3.1	ONLINE STORE DESIGN.....	62
9.3.1.1	IMPLEMENTATION WITH CURRENT SYSTEM.....	63
9.3.2	STRIPE INTEGRATION.....	64
9.3.3	WEB GUI DESIGN.....	65
9.3.4	ONLINE STORE SUMMARY.....	66
10.	SUMMARY.....	67
11.	REFERENCES.....	68

# OVERVIEW

## TABLE OF FIGURES

Figure 1.	Table of Functional Requirements.....	6
Figure 2.	Table of Non-Functional Requirements.....	8
Figure 3.	Requirement 2.1.....	8
Figure 4.	Requirement 3.2.....	9
Figure 5.	Requirement 4.....	9
Figure 6.	Requirement 7.....	9
Figure 7.	Image of Grocery Store Checkout Use Case Diagram.....	12
Figure 8.	Image of Update Inventory Use Case Diagram.....	13
Figure 9.	Image of Calculate Total Use Case Diagram.....	13
Figure 10.	Image of Validate Loyalty Card Use Case Diagram.....	14
Figure 11.	Image of Apply Discount Use Case Diagram.....	14
Figure 12.	Image of Process Payment Use Case Diagram.....	15
Figure 13.	Image of Grocery Store Checkout Class Diagram.....	16
Figure 14.	Image of Customer Class Diagram.....	17
Figure 15.	Image of Checkout Class Diagram.....	17
Figure 16.	Image of Customer Database Class Diagram.....	18
Figure 17.	Image of Loyalty Card Class Diagram.....	18
Figure 18.	Image of Payment Class Diagram.....	19
Figure 19.	Image of Product Class Diagram.....	19
Figure 20.	Image of Inventory Class Diagram.....	20
Figure 21.	Image of Grocery Store Checkout Sequence Diagram.....	22
Figure 22.	Image of Scan Product Sequence Diagram.....	23
Figure 23.	Image of Scan Loyalty Card Sequence Diagram.....	23
Figure 24.	Image of Make Payment Sequence Diagram.....	24
Figure 25.	Image of Loyalty Card System Class Diagram.....	25
Figure 26.	Image of Customer Class Diagram.....	26
Figure 27.	Image of Reward Class Diagram.....	28
Figure 28.	Image of Reward Redemption Diagram.....	29
Figure 29.	Image of Transaction Class Diagram.....	30
Figure 30.	Image of Database Relationship Diagram.....	32
Figure 31.	Visualisation of Customer – Transaction Relationship.....	32
Figure 32.	Visualisation of Customer – Reward Redemption Relationship.....	33
Figure 33.	Visualisation of Reward Redemption - Reward Relationship.....	33
Figure 34.	Visualisation of Transaction - Inventory Relationship.....	34
Figure 35.	Visualisation of Inventory – Stock Relationship.....	34
Figure 36.	Image of Staff Table.....	35
Figure 37.	Image of Testing V-Model Diagram.....	37
Figure 38.	Table of Test Cases.....	40
Figure 39.	Table of Prototype Testing Log.....	43
Figure 40.	Image of Tiered Architecture Diagram.....	47
Figure 41.	Visualisation of Checkout UI.....	59
Figure 42.	Visualisation of Self-Checkout UI.....	60
Figure 43.	Visualisation of Online Store Process.....	62
Figure 44.	Visualisation of Online Store UI.....	65

## 1. OVERVIEW

### 1.1 PURPOSE

This System Design Document serves to provide information as how to design and implement a Point of Sale (PoS) system for a small sized grocery store. The following document will include the requirements for the PoS system as discussed with the stakeholder (owner of the store), the appropriate diagrams to demonstrate how the system will operate, and the databases required for the system as well as the relationships between the databases. The document will also include some information on how this system could potentially be enhanced in the future so that it can scale along with the stakeholder's business.

### 1.2 SYSTEM OVERVIEW

This PoS system is intended to operate as a checkout system in a small sized grocery store. A PoS system contains the software and hardware required to manage transactions. For the Grocery Store this will entail the scanning of items, the calculating of a total and the management of payment. In order to enhance the system beyond a simple PoS system, the Grocery Store checkout will include an automatic inventory system and a customer loyalty card system. Many PoS systems are also required to be scalable so, in order to support this, the system will include a tiered architecture as well as provisions for how the system could work for a company that operates on a larger scale.

### 1.3 CONTEXT

The stakeholder owns a small grocery store. Their current checkout system is outdated and inefficient. In addition to this, all of their inventory management is done manually. This has resulted in significant amounts of inefficiency and could potentially lead to a loss of customers due to the time it takes to checkout. The stakeholder has recently employed some new employees and has therefore decided to upgrade their current checkout system to try and make their business more efficient. This would require a new inventory system which automatically updates. The stakeholder would also like to create a loyalty card programme, intended for repeat customers. Furthermore, the stakeholder believes that they could potentially expand their business in the next 5 to 10 years and therefore would like the new system to be scalable.

# SYSTEM REQUIREMENTS

## 2. SYSTEM REQUIREMENTS

At the beginning of the planning phase for the design of this PoS system, conversations were held between the design team and the stakeholder in order to identify what was required of the system. These requirements have been organised into functional and non-functional requirements. The design team as well as the stakeholder have also identified some of these requirements which will be of a higher priority than others. All these requirements, in addition to any assumptions made in this process are detailed below.

### 2.1 TABLE OF FUNCTIONAL REQUIREMENTS

REQ. NUM.	REQUIREMENT
<b>R1</b>	<b>Hardware</b>
<b>R1.1</b>	The system requires a functioning touchscreen monitor or other type of visual display
<b>R1.2</b>	The system requires a functioning cash register and drawer with appropriate locking mechanism
<b>R1.3</b>	The system requires a functioning card reader
<b>R1.4</b>	The system requires a functioning barcode scanner
<b>R1.5</b>	The system requires a functioning receipt printer
<b>R1.6</b>	The system requires a functioning set of scales
<b>R1.7</b>	The system requires a functioning computer or laptop to run software
<b>R1.8</b>	The system requires a stable internet connection
<b>R2</b>	<b>Software</b>
<b>R2.1</b>	The system is required to be fully capable of processing a transaction
<b>R2.2</b>	The system is required to be able to access associated databases
<b>R2.3</b>	The system requires a functioning UI so that it can be operated by an employee

# SYSTEM REQUIREMENTS

<b>R2.4</b>	The system requires a sign in system for employees as well as a secure database to store employee details
<b>R2.5</b>	The system must keep track of purchases and store data so that the stakeholder can analyse how well products are selling
<b>R3</b>	<b>Inventory System</b>
<b>R3.1</b>	The inventory system must be on the cloud so if the business is expanded it can be used by multiple systems
<b>R3.2</b>	The inventory system must automatically update after a purchase is made so that the stakeholder can manage stock
<b>R3.3</b>	The inventory system requires a secure database which stores all product information
<b>R4</b>	<b>Loyalty Card System</b>
<b>R4.1</b>	The loyalty card system must be on the cloud so if the business is expanded it can be used by multiple systems
<b>R4.2</b>	The loyalty card system requires a method of signing up for a loyalty card
<b>R4.3</b>	The loyalty card system must automatically add points to a customer's account after a purchase is made
<b>R4.4</b>	The loyalty card system must automatically apply any discounts after a purchase is made
<b>R4.5</b>	The loyalty card system requires a secure database which holds customer information, point totals and any point-based rewards
<b>R5</b>	<b>Payment System</b>
<b>R5.1</b>	The system must be able to accept cash payment
<b>R5.2</b>	The system must be able to accept chip and pin card payment
<b>R5.3</b>	The system must be able to accept contactless payment
<b>R5.4</b>	The system must ensure that all non-cash payments can be made securely without a risk of private data being accessed

*Figure 1. Table of Functional Requirements*



# SYSTEM REQUIREMENTS

## 2.2 TABLE OF NON-FUNCTIONAL REQUIREMENTS

REQ. NUM.	REQUIREMENT
<b>R6</b>	The system must be simple to use so that new employees can be trained easily
<b>R7</b>	The system must be efficient and responsive in order to process checkouts quicker
<b>R8</b>	The system must be fully available to access when the store is open i.e. it should not crash often
<b>R9</b>	The system must reliably calculate totals and apply discounts
<b>R10</b>	The system and associated databases must be secure in order to protect customer and employee information
<b>R11</b>	The system must have the capacity to be scaled up without having to install new software

*Figure 2. Table of Non-Functional Requirements*

## 2.3 HIGH PRIORITY REQUIREMENTS

### 2.3.1 REQUIREMENT 2.1

<b>R2.1</b>	The system is required to be fully capable of processing a transaction
-------------	--

*Figure 3. Requirement 2.1*

Requirement 2.1 was selected as high priority as it is fundamental to the success of the stakeholder's business and therefore must be designed and implemented correctly to ensure that the entire system functions properly.

# SYSTEM REQUIREMENTS

## 2.3.2 REQUIREMENT 3.2

**R3.2** The inventory system must automatically update after a purchase is made so that the stakeholder can manage stock

*Figure 4. Requirement 3.2*

Requirement 3.2 was selected as high priority as, as outlined in Section 1.3, Context, the current inventory system that the stakeholder has must be updated manually which has led to lots of inefficiency and takes up a lot of time. With the proper design and implementation of a new inventory system and corresponding database, this problem should be solved.

## 2.3.3 REQUIREMENT 4

**R4** Loyalty Card System

*Figure 5. Requirement 4*

Requirement 4, and the associated sub requirements, was selected as high priority as the stakeholder wanted to implement a loyalty card system in order to reward frequent customers and therefore improve customer satisfaction as previously mentioned in Section 1.3, Context.

## 2.3.4 REQUIREMENT 7

**R7** The system must be efficient and responsive in order process checkouts quicker

*Figure 6. Requirement 7*

Requirement 7 was selected as high priority as the stakeholder recognised that the main issue with their current system is that it is inefficient so therefore, they want their new system to be as efficient as possible as outlined in Section 1.3, Context.

# SYSTEM REQUIREMENTS

## 2.4 REQUIREMENT ASSUMPTIONS AND CONSTRAINTS

### 2.4.1 ASSUMPTIONS

In order to create an effective list of requirements a few assumptions were made to ensure that the list of requirements outlined in Figure 1 and Figure 2 were as comprehensive as possible. The assumptions are as follows:

- The stakeholder's place of business is in an area where a reliable internet connection is achievable.
- The stakeholder will be able to get hold of the necessary hardware in order to operate the system.
- The stakeholder will maintain their hardware so that the system can run without fault.
- The cost of operating the new software will not lead to a significant decrease in profits as the increased cost due to subscriptions to online services should balance out with an increase in customers and therefore an increase in sales.
- The stakeholder, after having received guidance on operating the system, will be able to train their employees to use the system.
- All products currently sold in the store have barcodes or some form of product number/code, so that they can be added to the inventory system and can be detected by the checkout system when they are purchased.

# SYSTEM REQUIREMENTS

## 2.4.2 CONSTRAINTS

In addition to the assumptions made, there are also a series of constraints on the project as a whole, which must be monitored closely to ensure the project stays on track. They are as follows:

- The cost of implementation and continued running of the system must not exceed the budget that the stakeholder has, as if the stakeholder makes less profit than before, it defeats the purpose of installing the new system.
- The stakeholder will want the system to be implemented sooner, rather than later, therefore the design cannot be overly complicated as this could delay implementation.
- All databases that are associated with the system must follow all local laws regarding data protection while still allowing the stakeholder to access them if required.
- In order to ensure that the system is efficient as possible all databases have to be optimally designed.

## 3. SYSTEM DESIGN

The following section is intended to demonstrate the design of the PoS system as well as clarify the relationships between the different parts of the system. This includes figures of the relevant UML use case, class, and sequence diagrams to show the many processes involved within this system.

### 3.1 GROCERY STORE CHECKOUT USE CASE DIAGRAM

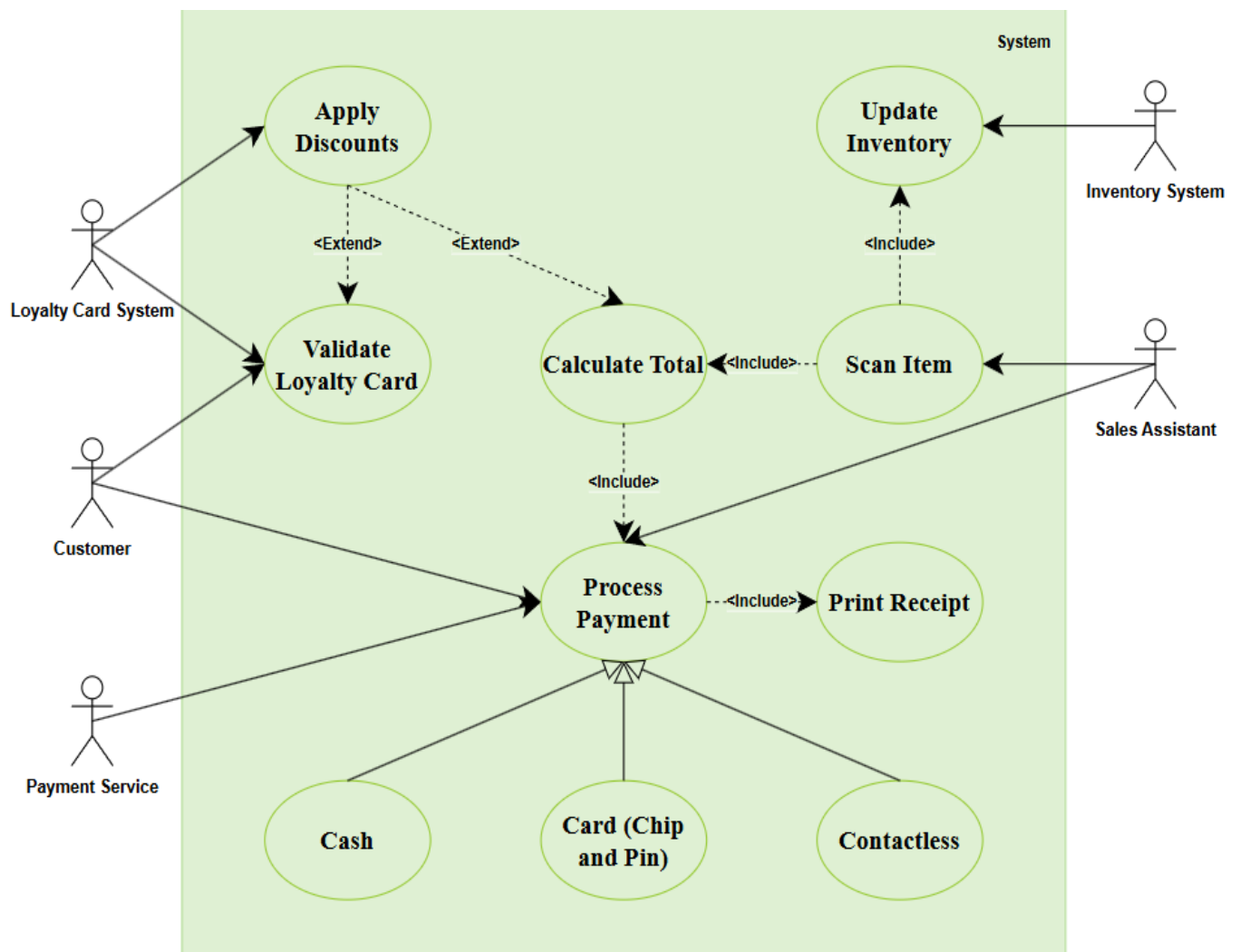


Figure 7. Image of Grocery Store Checkout Use Case Diagram

# SYSTEM DESIGN

## 3.1.1 UPDATE INVENTORY

The inventory system needs to automatically update after a product is scanned so that stock can be managed effectively. This should simply remove the number of items bought from the quantity or stock value in the inventory database.

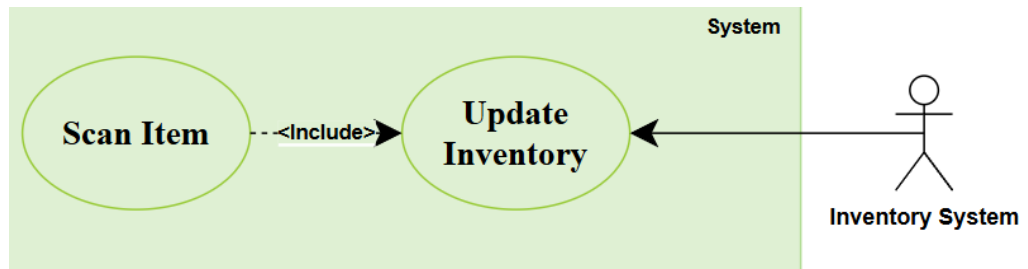


Figure 8. Image of Update Inventory Use Case Diagram

## 3.1.2 CALCULATE TOTAL

Once an item has been scanned by the sales assistant, the price of that item should automatically be added to the total of all products that are scanned for that single purchase.

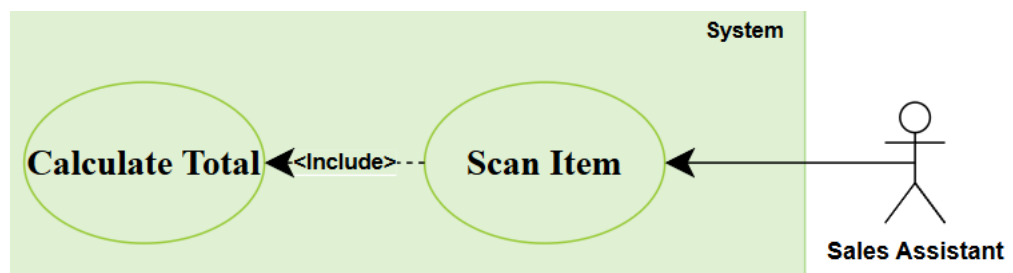


Figure 9. Image of Calculate Total Use Case Diagram

# SYSTEM DESIGN

## 3.1.3 VALIDATE LOYALTY CARD

Once a customer presents a loyalty card, this card should be checked and validated by the loyalty card system. After the card is approved, the customer will then become eligible to earn points or for any discounts.

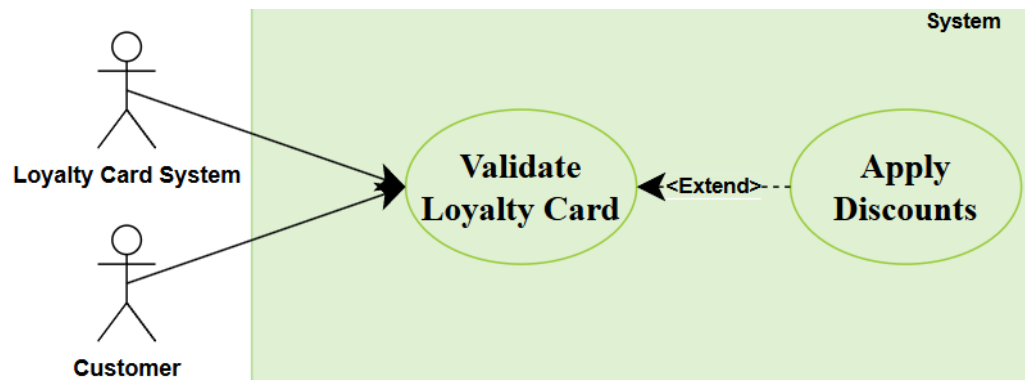


Figure 10. Image of Validate Loyalty Card Use Case Diagram

## 3.1.4 APPLY DISCOUNTS

After a loyalty card has been approved, any discounts earned will need to be applied to the total. Points will also be earned but as these are not directly involved in the transaction, assume they happen internally within the loyalty card system.



Figure 11. Image of Apply Discount Use Case Diagram

# SYSTEM DESIGN

## 3.1.5 PROCESS PAYMENT

After the total is calculated and all discounts have been applied, the sales assistant should begin to process a payment. This can be done by cash, card, or contactless payment which the customer will provide. If card or contactless are chosen the payment will have to be approved by a payment system. After the payment has been accepted a receipt displaying a summary of the purchase should be generated and printed automatically.

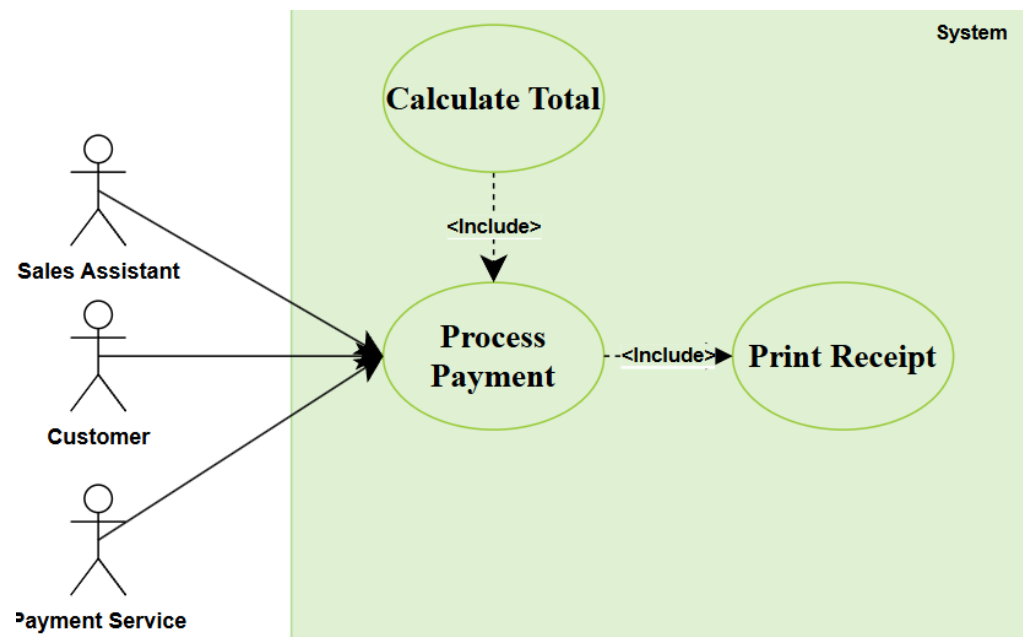


Figure 12. Image of Process Payment Use Case Diagram



# SYSTEM DESIGN

## 3.2 GROCERY STORE CHECKOUT CLASS DIAGRAM

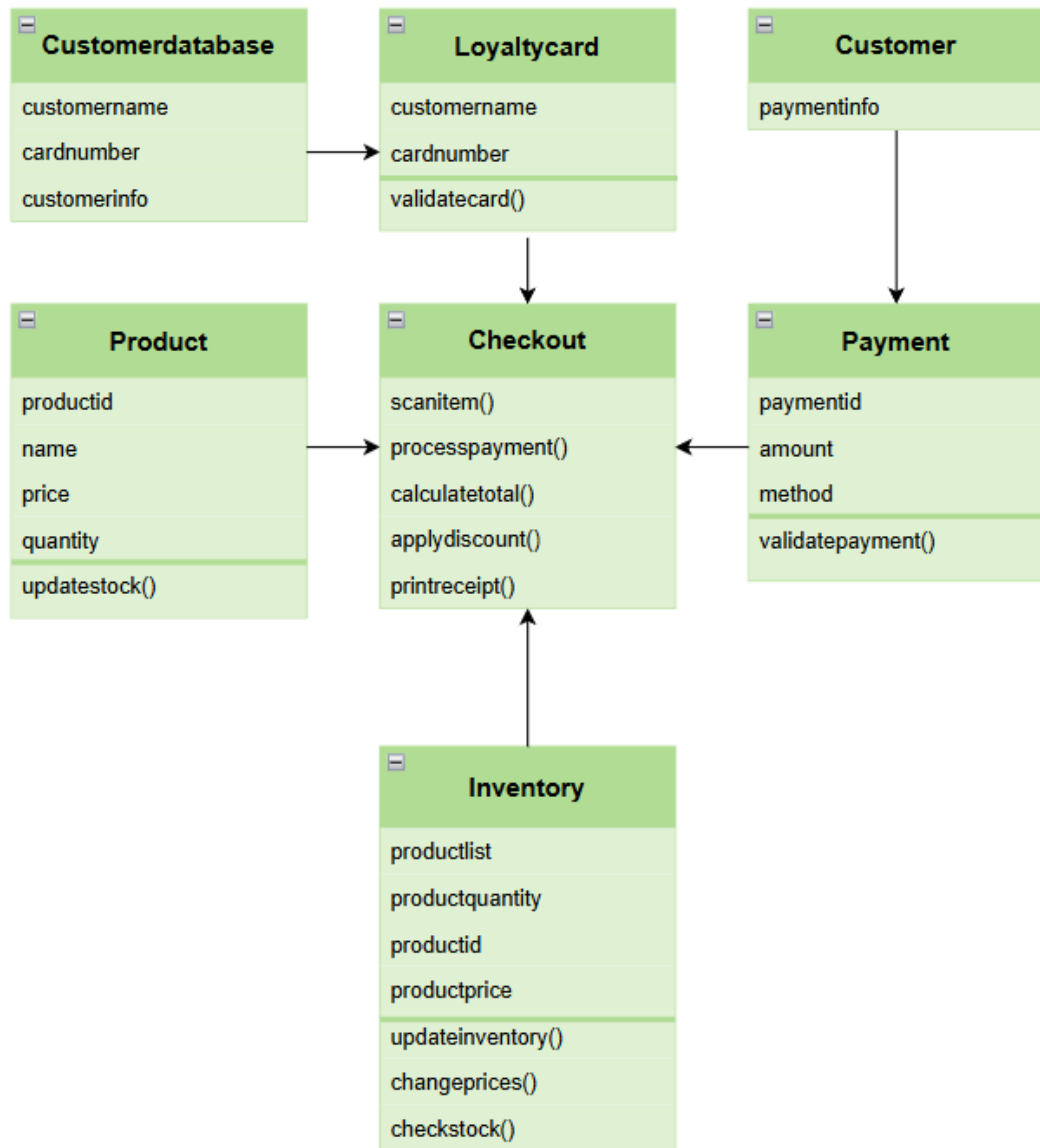


Figure 13. Image of Grocery Store Checkout Class Diagram

# SYSTEM DESIGN

## 3.2.1 CUSTOMER CLASS

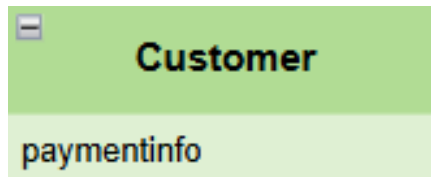


Figure 14. Image of Customer Class Diagram

### 3.2.1.1 FIELDS

- **paymentinfo** – This field relates to the payment information of the customer. Usually, this information will be cash, card, or contactless payment. This is the only information required for the customer when they go to checkout.

## 3.2.2 CHECKOUT CLASS

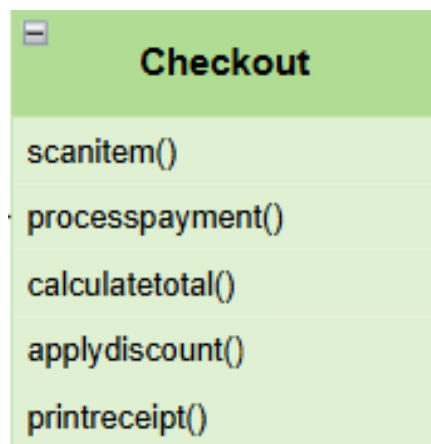


Figure 15. Image of Checkout Class Diagram

### 3.2.2.1 FUNCTIONS

- **scanitem()** – This function allows the sales assistant at the checkout to scan barcodes.
- **processpayment()** – This function allows the sales assistant to take payment from the customer either through cash, card or contactless.
- **calculatetotal()** – This function allows the checkout to automatically calculate the running total of products.
- **applydiscount()** – This function allows discounts to be applied after a total has been calculated.
- **printreceipt()** – This function allows the sales assistant to print a receipt with information about the last sale to give to the customer.

# SYSTEM DESIGN

## 3.2.3 CUSTOMER DATABASE CLASS

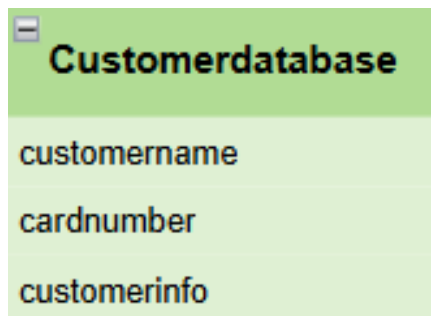


Figure 16. Image of Customer Database Class Diagram

### 3.2.3.1 FIELDS

- **customername** – This field contains the names of customers who have a loyalty card.
- **cardnumber** – This field contains a list of loyalty card numbers,
- **customerinfo** - This field contains various pieces of information about loyalty card customers. This is shown in more detail in Section 3.4, Loyalty Card System Class Diagram.

## 3.2.4 LOYALTY CARD CLASS

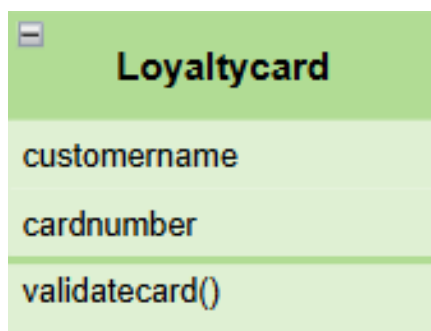


Figure 17. Image of Loyalty Card Class Diagram

### 3.2.4.1 FIELDS

- **customername** – This field refers to the customer's name that would be present on a customer's loyalty card.
- **cardnumber** - This field refers to the card number that would be present on a customer's loyalty card.

### 3.2.4.2 FUNCTIONS

- **validatecard()** – This function allows the sales assistant to validate a loyalty card so that points can be earned, and discounts can be applied.

# SYSTEM DESIGN

## 3.2.5 PAYMENT CLASS

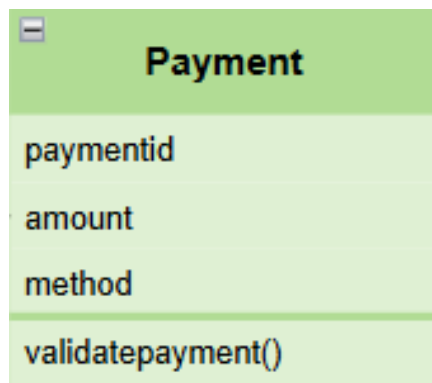


Figure 18. Image of Payment Class Diagram

### 3.2.5.1 FIELDS

- **paymentid** – This field refers to the transaction number assigned to a payment.
- **amount** – This field refers to the total cost of a purchase.
- **method** – This field refers to the chosen method of payment which is usually cash, card, or contactless payment.

### 3.2.5.2 FUNCTIONS

- **validatepayment()** – This function allows the system to approve or reject payments.

## 3.2.6 PRODUCT CLASS



Figure 19. Image of Product Class Diagram

## 3.2.6.1 FIELDS

- **productid** – This field refers to the unique product identification number which is given to each product as well as a barcode.
- **name** – This field refers to the name of the product.
- **price** – This field refers to the listed price of the product.
- **quantity** – This field refers to the current stock level of the product.

## 3.2.6.2 FUNCTIONS

- **updatestock()** – This function allows the system to automatically update the quantity of a product in the inventory system after it has been purchased.

## 3.2.7 INVENTORY CLASS

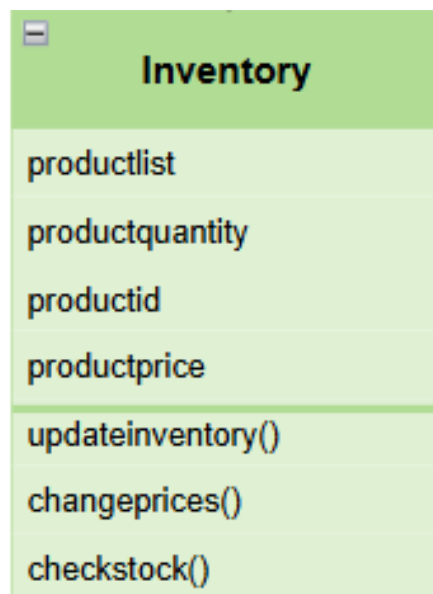


Figure 20. Image of Inventory Class Diagram

## 3.2.7.1 FIELDS

- **productlist** – This field refers to a list of every product that is currently sold in the grocery store.
- **productquantity** – This field refers to a list of the total stock of each product currently sold in the grocery store.
- **productid** – This field refers to a list of product identification numbers for every product currently sold in the grocery store.
- **productprice** – This field refers to a list of prices for every product currently sold in the grocery store.

## 3.2.7.2 FUNCTIONS

- **updateinventory()** – This function allows the stakeholder to change the stock of a product as well as add new products or alternatively remove products that they no longer wish to sell.
- **changeprices()** – This function allows the stakeholder to change the prices of products which will then be reflected in the price once the product is bought.
- **checkstock()** – This function allows the stakeholder to check the stock of products to see when it is running low, so they can then order more of the product if needed.

# SYSTEM DESIGN

## 3.3 GROCERY STORE CHECKOUT SEQUENCE DIAGRAM

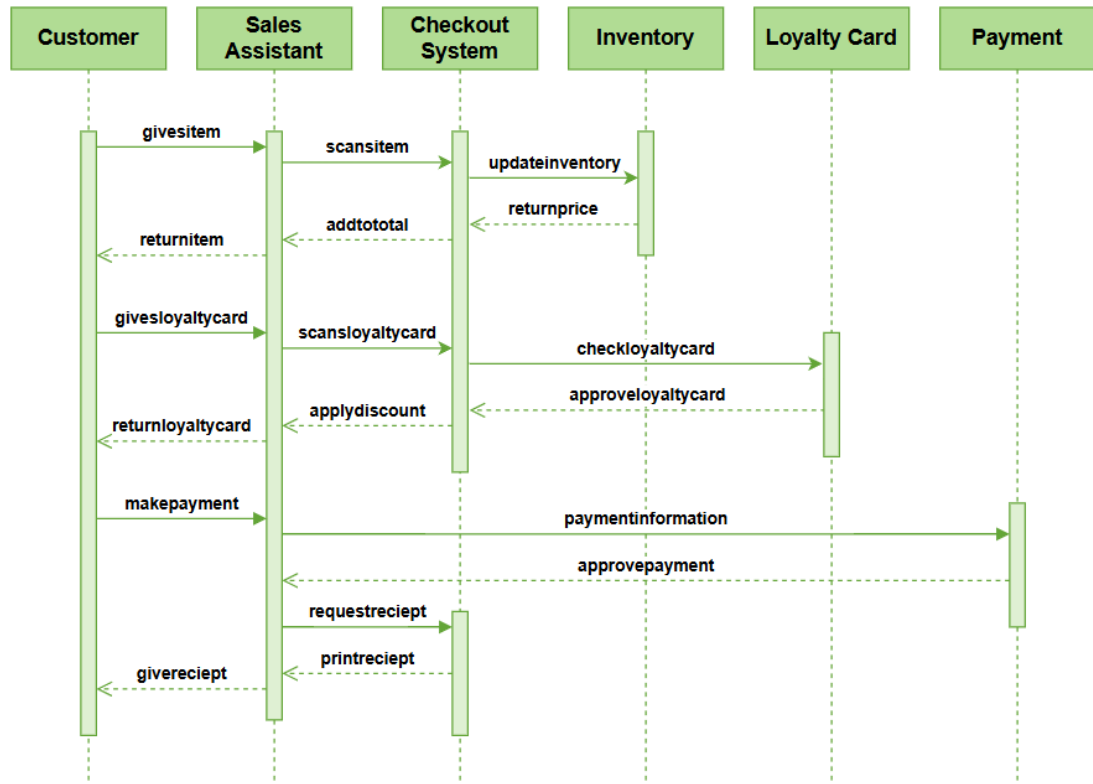


Figure 21. Image of Grocery Store Checkout Sequence Diagram

# SYSTEM DESIGN

## 3.3.1 SCAN PRODUCT

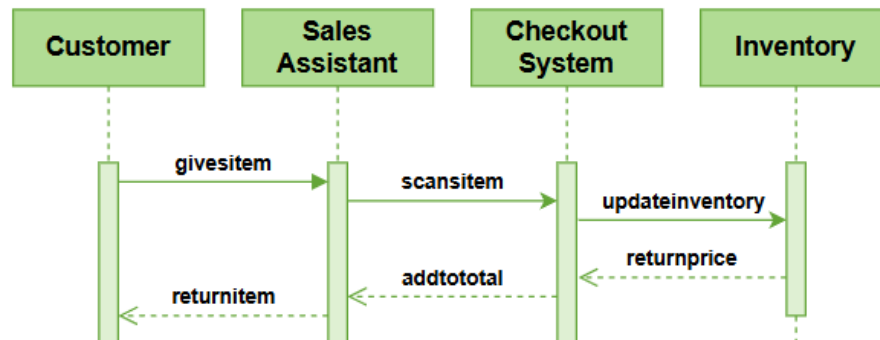


Figure 22. Image of Scan Product Sequence Diagram

Once the customer reaches the checkout, they hand their item to the sales assistant. The sales assistant then scans the barcode of the item using the barcode scanner at the checkout. The information from the product barcode is then sent from the checkout to the inventory system which updates stock and then returns a price to the checkout. The checkout then adds this price to the total before the sales assistant hands the product back to the customer. This process is repeated for every product that a customer wishes to purchase.

## 3.3.2 SCAN LOYALTY CARD

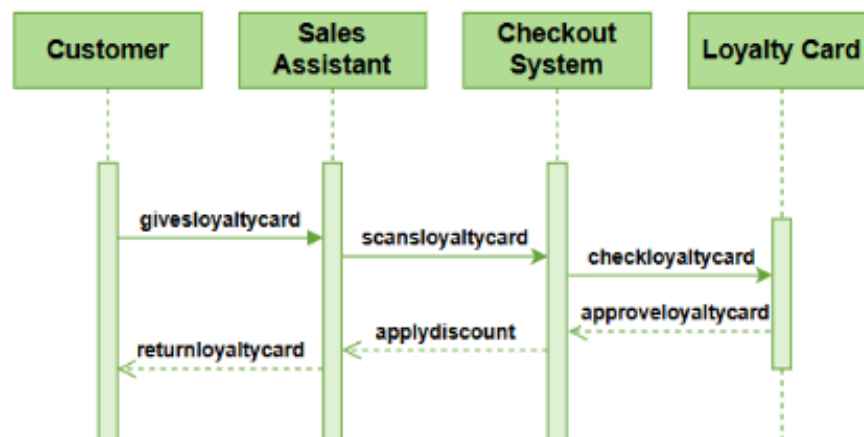


Figure 23. Image of Scan Loyalty Card Sequence Diagram

If the customer has a loyalty card, when making a purchase, they can give their loyalty card to the sales assistant, who will then scan their card using the checkout hardware. The checkout system then communicates the loyalty card number to the loyalty card system, which then either approves or rejects the card. Once approved the customer will earn points in the loyalty card system and the checkout will apply any loyalty card related discounts to the purchase total. The loyalty card is then returned to the customer.



# SYSTEM DESIGN

## 3.3.3 MAKE PAYMENT

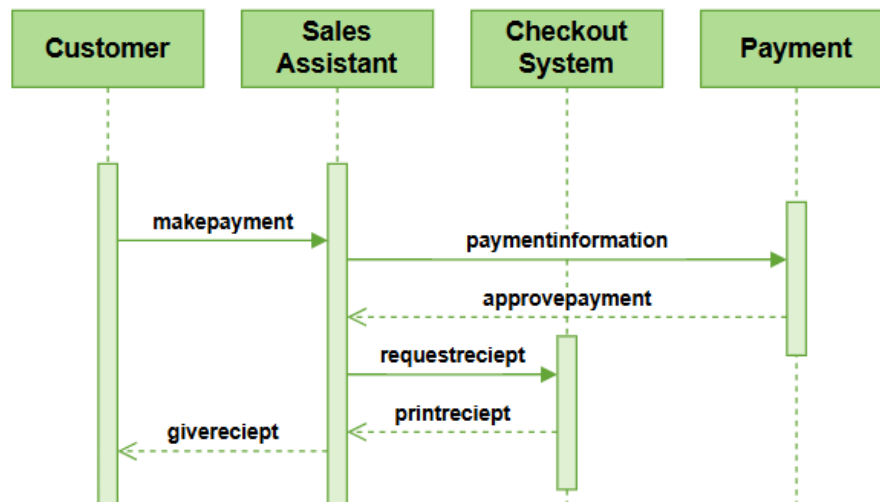


Figure 24. Image of Make Payment Sequence Diagram

After the total has been fully calculated a payment is required. The customer makes the payment and for cash the sales assistant must ensure it is the correct amount. For card and contactless payments, the customer's payment information is sent securely to the payment system which then either approves or rejects the payment. After a payment has been approved the sales assistant can request a receipt from the checkout system, which prints the receipt. This receipt is then given to the customer and the entire transactions is completed.

# SYSTEM DESIGN

## 3.4 LOYALTY CARD SYSTEM CLASS DIAGRAM

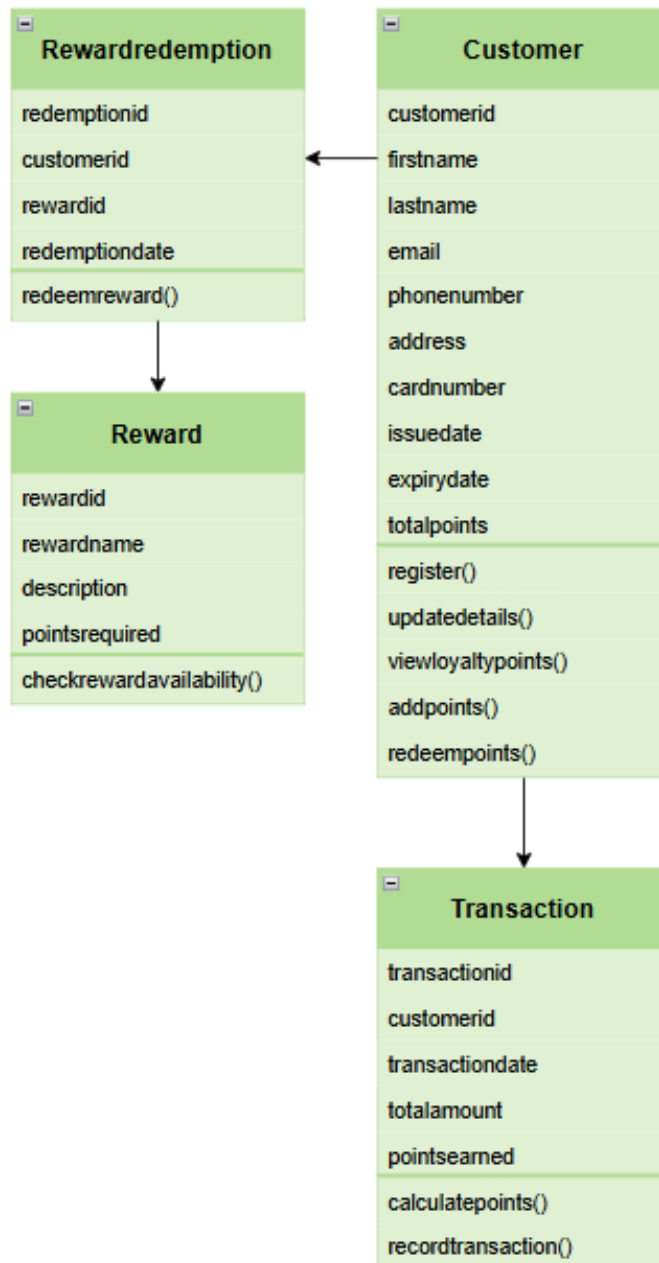


Figure 25. Image of Loyalty Card System Class Diagram

## 3.4.1 CUSTOMER CLASS

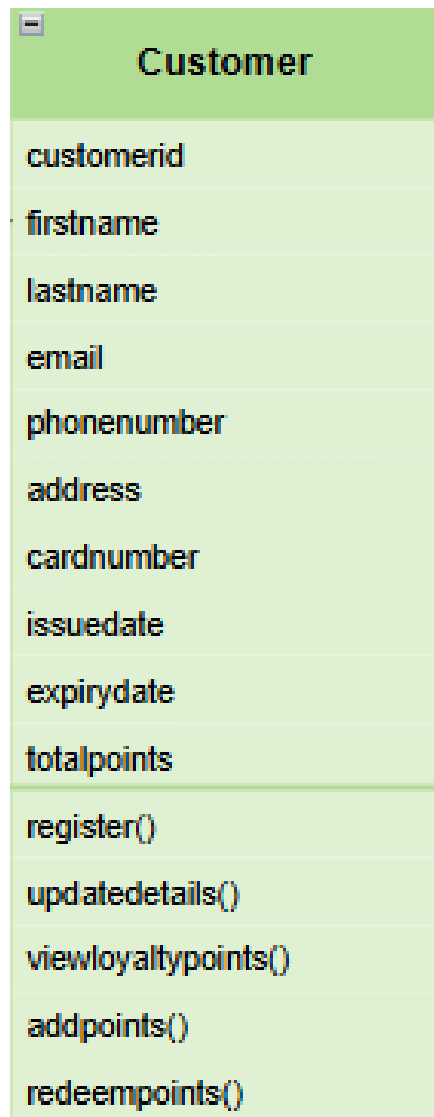


Figure 26. Image of Customer Class Diagram

## 3.4.1.1 FIELDS

- **customerid** – This field refers to a unique reference number given to each customer who signs up for a loyalty card.
- **firstname** – This field refers to the first name of the customer.
- **lastname** – This field refers to the last name of the customer.
- **email** – This field refers to the email address that the customer registers to their account when signing up for a loyalty card.
- **phonenumber** – This field refers to the phone number that the customer registers to their account when signing up for a loyalty card.
- **address** – This field refers to the address of the customer.
- **cardnumber** – This field refers to the unique reference number assigned to each loyalty card.
- **issuedate** – This field refers to the issue date of a loyalty card.
- **expirydate** – This field refers to the expiry date of a loyalty card.
- **totalpoints** – This field refers to the number of points associated with a single loyalty card.

## 3.4.1.2 FUNCTIONS

- **register()** – This function allows customers to register for a loyalty card.
- **updatedetails()** – This function allows customers to update their personal details such as phone number and email address once they have a loyalty card.
- **viewloyaltypoints()** – This function allows customers to view how many loyalty points they have accumulated.
- **addpoints()** – This function allows points to be added to a loyalty card after a transaction.
- **redeempoints()** – This function allows customers to redeem their points so that they can claim rewards.

## 3.4.2 REWARD CLASS

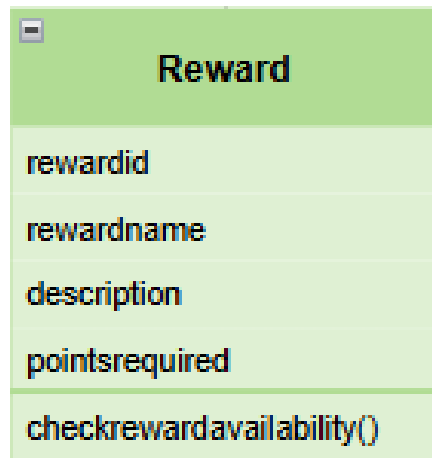


Figure 27. Image of Reward Class Diagram

### 3.4.2.1 FIELDS

- **rewardid** – This field refers to the unique reference number given to each reward.
- **rewardname** – This field refers to the name of the reward.
- **description** – This field refers to a brief description given to each reward which explains what the reward is.
- **pointsrequired** – This field refers to the points required to earn a specific reward.

### 3.4.2.2 FUNCTIONS

- **checkrewardavailability()** – This function allows customers to see if a reward is currently available, so that they do not try to redeem a reward that is not available.

## 3.4.3 REWARD REDEMPTION CLASS



Figure 28. Image of Reward Redemption Class

### 3.4.3.1 FIELDS

- **redemptionid** – This field refers to the unique identification number assigned to a redemption.
- **customerid** - This field refers to a unique reference number given to each customer who signs up for a loyalty card.
- **rewardid** – This field refers to the unique identification number given to a reward.
- **redemptiondate()** – This field refers to the date of a redemption.

### 3.4.3.2 FUNCTIONS

- **redeemreward()** – This function allows rewards to be redeemed.

# SYSTEM DESIGN

## 3.4.4 TRANSACTION CLASS

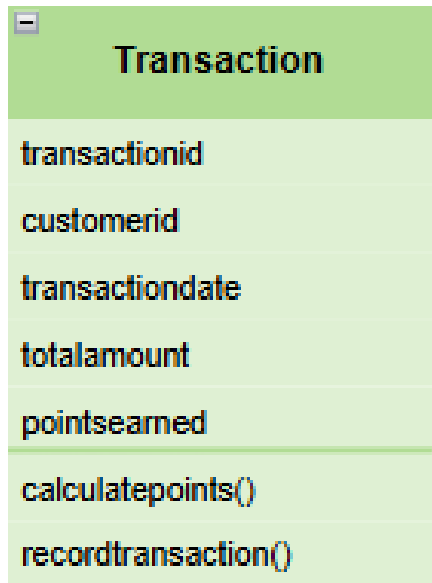


Figure 29. Image of Transaction Class Diagram

### 3.4.4.1 FIELDS

- **transactionid** – This field refers to the unique reference number given to each individual transaction.
- **customerid** - This field refers to a unique reference number given to each customer who signs up for a loyalty card.
- **transactiondate** – This field refers to the date of the transaction.
- **totalamount** - This field refers to the total cost of the transaction.
- **pointsearned** – This field refers to the point earned by the customer after a transaction has occurred.

### 3.4.4.2 FUNCTIONS

- **calculatepoints()** – This function allows the system to calculate the points earned during a transaction and add it to the customers total points on their loyalty card account.
- **recordtransaction()** – This function allows the transaction to be recorded, so the stakeholder can keep information for future use or data analytics.

## 4. DATABASE DESIGN

### 4.1 REQUIRED DATABASE INFORMATION

For the grocery store PoS system a database containing various tables is required. The database will also be contained in SQLite. The main tables within the database are shown in Section 3.4, Loyalty Card System Class Diagram. However, a few further tables have been included, such as Inventory, Stock and Staff, which were not previously shown in any Class Diagram. The tables are as follows:

- **Customer** – This table contains all the fields shown in the Customer class in Section 3.4, Loyalty Card System Class Diagram.
- **Transaction** - This table contains all the fields shown in the Transaction class in Section 3.4, Loyalty Card System Class Diagram.
- **Reward Redemption** - This table contains all the fields shown in the Reward Redemption class in Section 3.4, Loyalty Card System Class Diagram.
- **Reward** - This table contains all the fields shown in the Reward class in Section 3.4, Loyalty Card System Class Diagram.
- **Inventory** – This table contains all the fields related to an overall product inventory such as: Product ID, Product Name, Product Price, Purchase History and Points Earned
- **Stock** – This table contains all the fields related to an overall stock list such as Product ID and Product Quantity.
- **Staff** – This table contains all the fields related to staff information such as First Name, Last Name, Email, Username and Password.



# DATABASE DESIGN

## 4.2 DATABASE RELATIONSHIPS

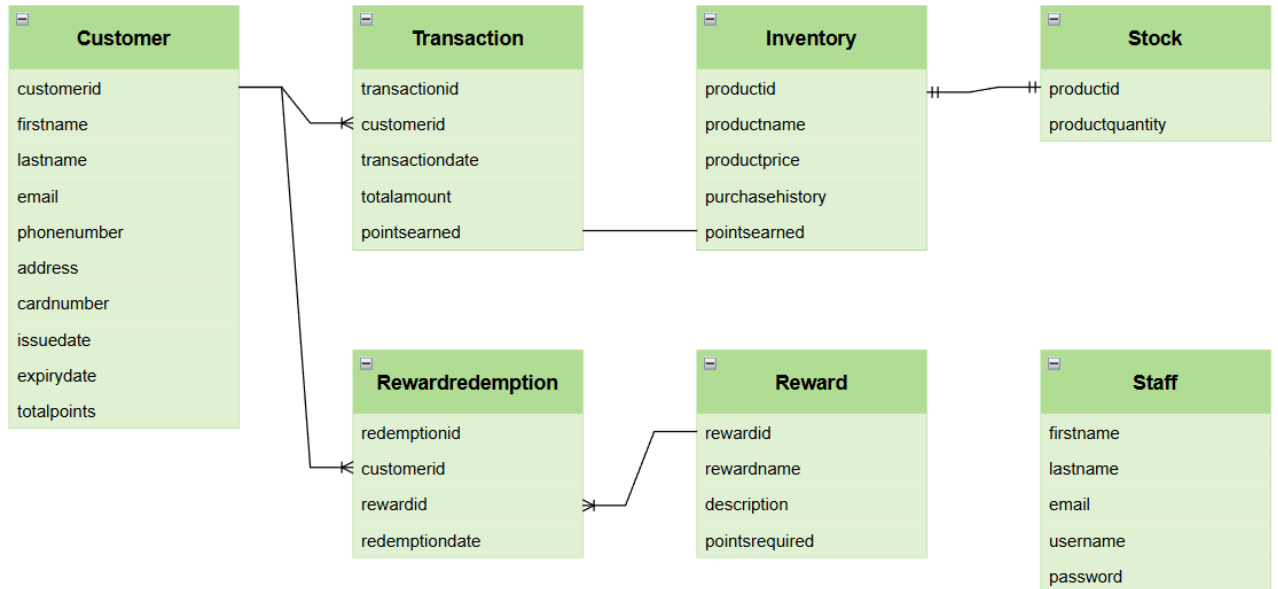


Figure 30. Image of Database Relationship Diagram

### 4.2.1 CUSTOMER – TRANSACTION RELATIONSHIP

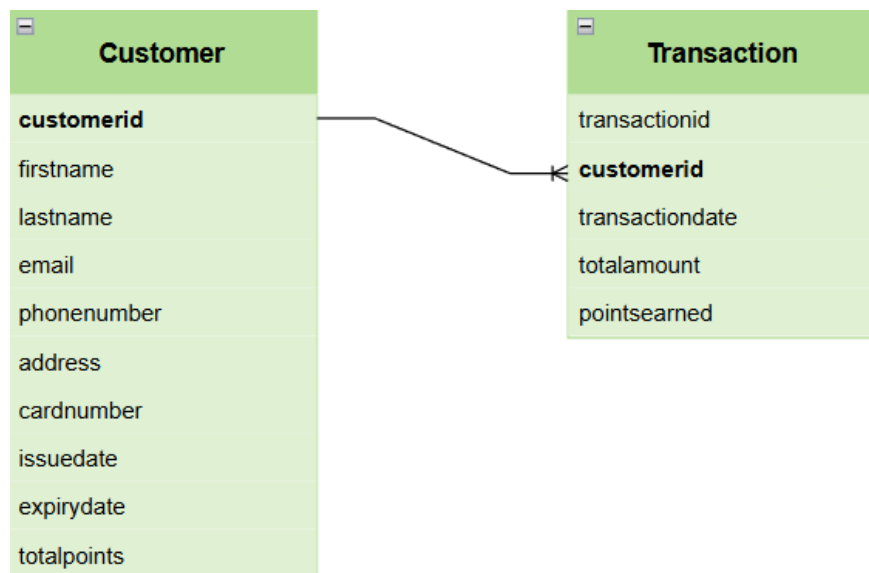


Figure 31. Visualisation of Customer – Transaction Relationship

The Customer – Transaction Relationship represents a one-to-many relationship with the Customer ID acting as a primary key in the Customer table and as a foreign key in the Transaction table. Each customer has a unique Customer ID in the Customer table but in the Transaction table a single customer can have multiple purchases so therefore there is a one-to-many relationship.

# DATABASE DESIGN

## 4.2.2 CUSTOMER – REWARD REDEMPTION RELATIONSHIP

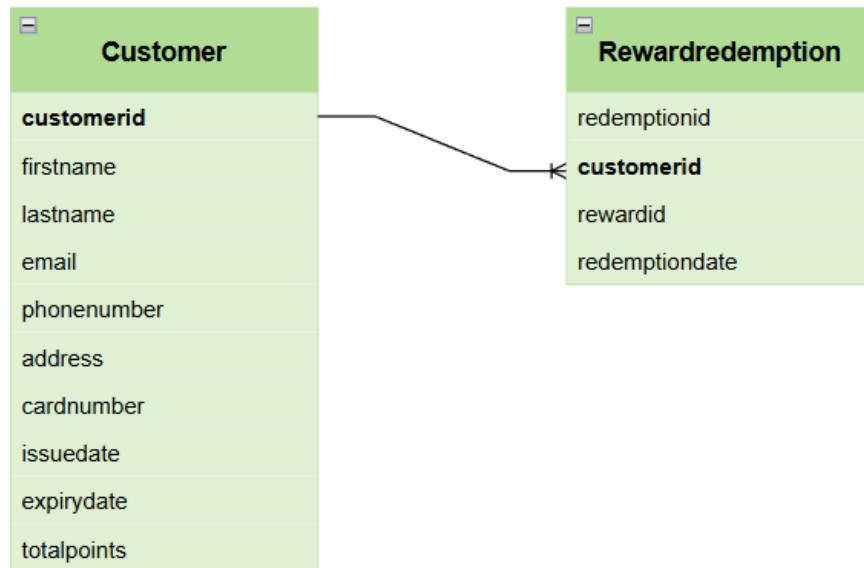


Figure 32. Visualisation of Customer – Reward Redemption Relationship

The Customer – Reward Redemption Relationship represents a one-to-many relationship with the Customer ID acting as a primary key in the Customer table and as a foreign key in the Reward Redemption table. Each customer has a unique Customer ID in the Customer table but in the Transaction table a single customer can redeem multiple rewards so therefore there is a one-to-many relationship.

## 4.2.3 REWARD REDEMPTION – REWARD RELATIONSHIP

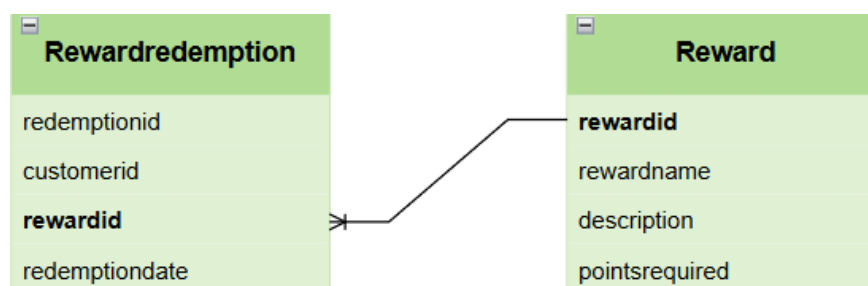


Figure 33. Visualisation of Reward Redemption - Reward Relationship

The Reward Redemption - Reward Relationship represents a many-to-one relationship with the Reward ID acting as a foreign key in the Reward Redemption table and as a primary key in the Reward table. Each Reward has a unique Reward ID in the Reward table, but each Reward can be redeemed multiple times in the Reward Redemption table, therefore there is a many-to-one relationship.

# DATABASE DESIGN

## 4.2.4 TRANSACTION – INVENTORY RELATIONSHIP

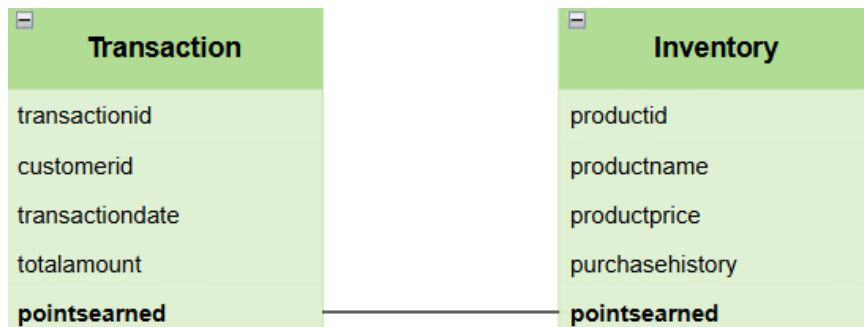


Figure 34. Visualisation of Transaction - Inventory Relationship

Points Earned acts as a relationship between the Transaction table and the Inventory table. It is not a primary key in either table, and therefore is also not a foreign key in either table. It is also not a one-to-one, one-to-many or many-to-many relationship. The relationship is due to the fact that in the Inventory Table each product will have a number of points earned for purchasing that product, and in the Transaction table there will be a Points Earned field which is a sum of the Points Earned from various products in the Inventory table, purchased in a single transaction.

## 4.2.5 INVENTORY – STOCK RELATIONSHIP

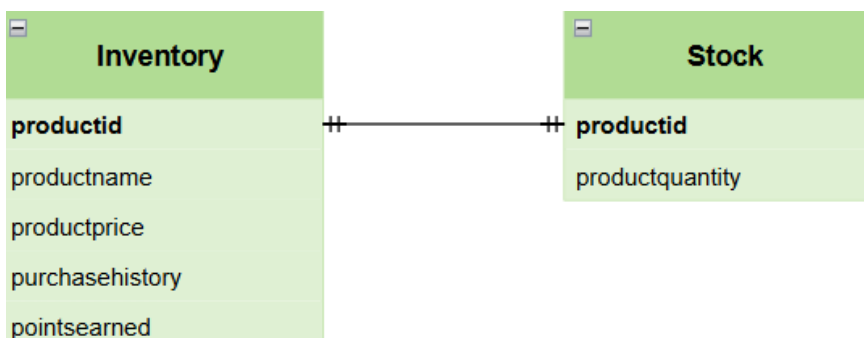
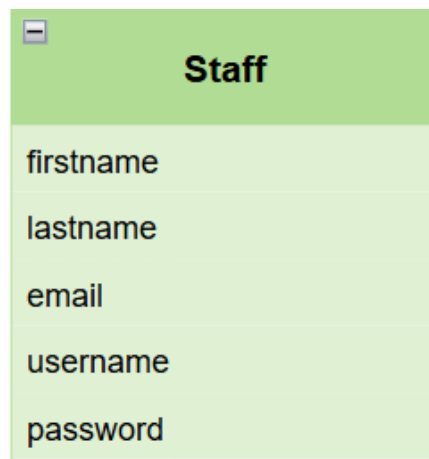


Figure 35. Visualisation of Transaction - Inventory Relationship

The Inventory - Stock Relationship represents a one-to-one relationship with the Product ID acting as a Primary key in both the Inventory table and the Stock table. Each product has a unique Product ID that is represented in both tables, therefore there is a one-to-one relationship. These two tables could be combined but as the stakeholder has expressed their wish to eventually expand their business, they have been separated so that if the stakeholder opens multiple stores, each store can have an individual stock list whilst having the same list of available products in the Inventory Table.

# DATABASE DESIGN

## 4.2.6 STAFF TABLE



Staff
firstname
lastname
email
username
password

*Figure 36. Image of Staff Table*

The Staff table has no relationships with other tables as it is completely independent of all other information stored in the database. This table will not interact with any other tables as its sole purpose is to be used to allow staff to login into the checkout system, so that only staff member can operate it. This table could potentially be excluded from the database and stored on some in-store hardware if the stakeholder wishes.

## 5. TESTING

The following section should outline a plan for the testing of the Grocery Store Checkout PoS system. Included will be, the objectives for the testing, the methods that will be used in testing, information on test cases and information on prototype testing. This should provide all the necessary information to complete thorough testing on the system and ensure that the end product that is released to the stakeholder is of high-quality.

### 5.1 TESTING OBJECTIVES

In order to ensure that the Grocery Store PoS system is released to the stakeholder in an acceptable state, testing is required. There are many objectives to testing, which are as follows:

- **Identifying Bugs** – The main objective of testing is to identify bugs in the software so that they can later be fixed. This is important as the system should only be handed over to the stakeholder for operational use once the software development team are confident that any major bugs and a majority of minor bugs have been fixed. This will also reduce maintenance costs for the stakeholder in the future.
- **Requirement Testing** – During testing, the system should be evaluated as to whether or not it meets the stakeholder's requirements outlined in Section 2, System Requirements. By the time the system is released, it should meet a majority of these requirements and meet all of the high-priority requirements listed in Section 2.3, High-Priority Requirements.
- **Usability Testing** – Testing should be used to find out if the system is user-friendly. If it is not, this must be fixed before release.

# TESTING

- **Transaction Reliability** – Testing should be used to ensure that the checkout system accurately and reliably processes transactions. This includes scanning items, calculating a total and applying discounts. It must be ensured that the system can do this over a significant amount of time.
- **Integration Testing** – Testing should be used to ensure that the system architecture functions as intended and that all systems are function cohesively.
- **Security** – Testing should be used to ensure that the system is secure. This is particularly important for the systems databases as they contain information about customers and staff. Furthermore, the system should be tested for compliance with local data protection regulations.

## 5.2 TESTING V-MODEL

In order to test the system effectively, the design team suggests the use of the V-Model as outlined below. This should allow for each area of the system to be thoroughly tested and ensure that the testing objectives are met.

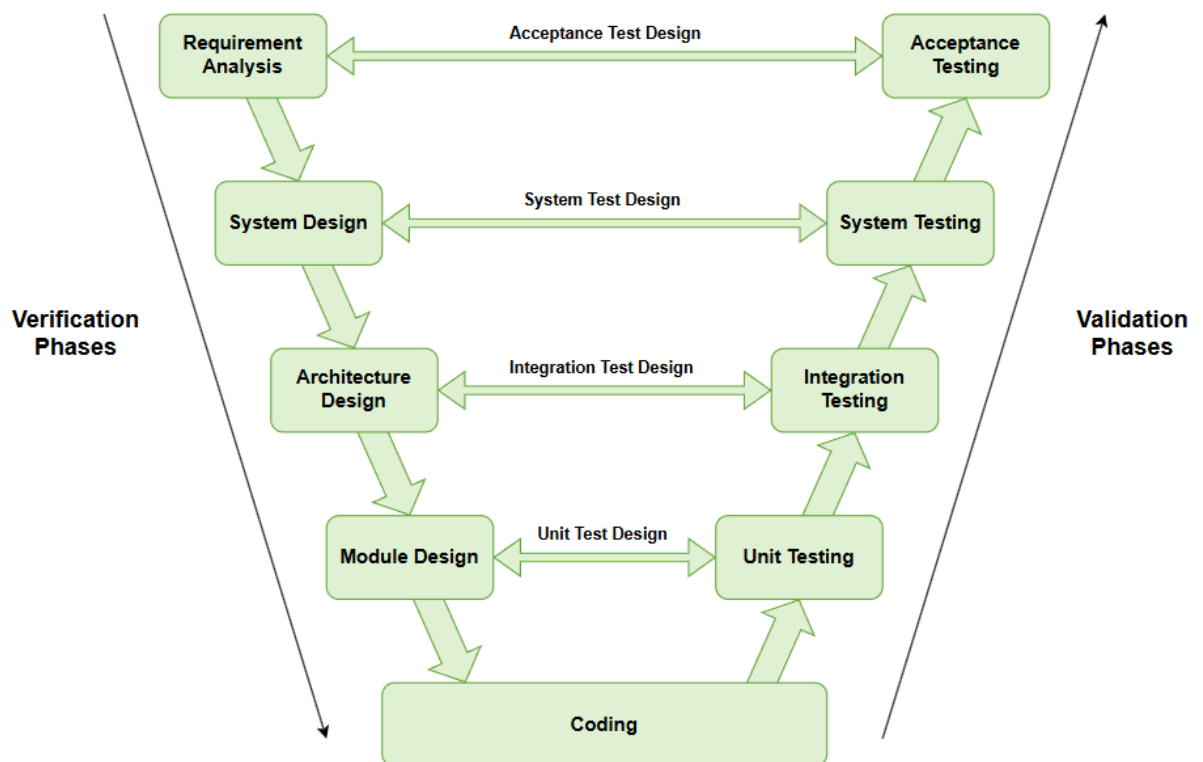


Figure 37. Image of Testing V-Model Diagram

## 5.2.1 VERIFICATION PHASES

- **Requirement Analysis** – This phase covers the identification and analysis of system requirements. This is shown in Section 2, System Requirements. The corresponding test phase is Acceptance Testing.
- **System Design** – This phase refers to the high-level design of the Grocery Store Checkout system. This is covered in Section 3, System Design. The corresponding test phase is System Testing.
- **Architecture Design** – This phase covers the design of the system architecture, which is covered in Section 6, System Architecture. The corresponding test phase is integration testing.
- **Module Design** – This phase refers to the low-level design of the system, mainly focusing on individual system functions like validating a loyalty card or scanning an item. The corresponding test phase is Unit Testing.

## 5.2.2 VALIDATION PHASES

- **Unit Testing** – This involves testing individual functions within the system to see if they work as intended. It breaks down all the software into individual tasks allowing for quicker identification of bugs and areas which may be causing the system to fail.
- **Integration Testing** – This involves testing separate components of the system to ensure they work together. For example, testing whether the checkout system interacts correctly with the databases and whether the loyalty card system functions correctly with the checkout system.
- **System Testing** – This involves testing whether the complete system is able to carry out the entire process shown in Section 3.3, Grocery Store Checkout Sequence Diagram. This tests whether the system has been designed as intended.
- **Acceptance Testing** – This involves testing the system to see if it meets the requirements set out by the stakeholder in Section 2, System Requirements.

## 5.3 WHITE BOX VS BLACK BOX TESTING

For the testing phase of the project the design team has had to analyse whether White Box, Grey Box or Black Box testing would be optimal. The design team has evaluated all three options and decided that using a combination of all three options would be optimal, due to the benefits of each option. These benefits are shown below:

- **White Box Testing** – This method of testing is where the person performing the tests has an in-depth knowledge of the code and structure of the system's software. This allows the tester to identify problems within the software and rectify them. This is optimal when bug-testing as any bugs that are identified can immediately be fixed. This method would be the fastest method for identifying and fixing bugs. For the Grocery Store checkout system, it is recommended that White Box testing is conducted throughout the coding process.
- **Grey Box Testing** – This method of testing is where the person performing the tests has some knowledge of the code and structure of the system's software. This allows the tester to test individual functions of the system that they did not create. This is optimal as it allows the coder to get a second opinion from someone else within the team on a system or function. Within this system Grey Box testing would be performed after someone has already performed a White Box test.
- **Black Box Testing** – This method of testing is where the person performing the tests has no knowledge of the code and structure of the system's software. This is optimal for conducting user tests to ensure that the system can be used by the intended user. This is best done after the previous two methods of testing as someone who has no idea of how the system works may try to do something that the system designers did not anticipate which would then require the code to be rectified.



# TESTING

## 5.4 TEST CASES

As part of the testing process, the design team has identified a series of test cases which should be conducted to ensure the system functions. The majority of these test cases will fall under Unit Testing as they are testing individual functions of the system. Some of the test cases which involve databases will come under Integration Testing. The test cases and their expected outcomes have been identified in the table below.

NUM.	TEST CASE	EXPECTED OUTCOME
<b>T1.</b>	<b>Item Scanning</b>	
<b>T1.1</b>	Scan an individual item	System should display the correct price for that item.
<b>T1.2</b>	Scan an item with an incomplete barcode	System should respond with error message. E.g. Barcode could not be identified.
<b>T1.3</b>	Manually input product number	System should display the correct price for that item.
<b>T1.4</b>	Manually input invalid product number	System should respond with error message. E.g. Product could not be identified.
<b>T1.5</b>	Scan multiple of the same item	System should register multiple of the item and display the correct price.
<b>T1.6</b>	Remove an item	System should remove selected item from transaction and update prices accordingly.
<b>T1.7</b>	Apply discounts	System should register item and automatically apply any discounts to its price.
<b>T2.</b>	<b>Calculating Total</b>	
<b>T2.1</b>	Scan an individual item	System should provide correct total price for that item.
<b>T2.2</b>	Scan multiple items	System should provide correct total price for the sum of all the prices of the various items.
<b>T3.</b>	<b>Payment</b>	
<b>T3.1</b>	Pay by cash (correct amount)	System should accept cash and allow the sales assistant to access the cash drawer.
<b>T3.2</b>	Pay by cash (incorrect amount)	System should accept cash and allow the sales assistant to access the cash drawer. System should display correct amount of change to give to customer.
<b>T3.3</b>	Pay with valid card	System should accept and process card payment.
<b>T3.4</b>	Pay with invalid card	System should reject payment and display error message. E.g. Invalid Card.

# TESTING

<b>T3.5</b>	Pay by contactless (valid)	System should accept and process contactless payment.
<b>T3.6</b>	Pay by contactless (invalid)	System should reject payment and display error message. E.g. Invalid Contactless Payment.
<b>T4.</b>	<b>Loyalty Card Validity</b>	
<b>T4.1</b>	Scan valid loyalty card	System should accept loyalty card.
<b>T4.2</b>	Scan invalid loyalty card	System should reject loyalty card and display error message. E.g. Invalid Loyalty Card
<b>T4.3</b>	Scan expired loyalty card	System should reject loyalty card and display error message. E.g. Loyalty Card Expired
<b>T4.4</b>	Scan valid loyalty card at start of transaction	System should accept loyalty card.
<b>T4.5</b>	Scan valid loyalty card during transaction	System should accept loyalty card.
<b>T4.6</b>	Scan valid loyalty card at payment	System should accept loyalty card.
<b>T4.7</b>	Scan valid loyalty card after payment	System should reject loyalty card and display error message. E.g. No Transaction in Progress
<b>T5.</b>	<b>Loyalty Card Points</b>	
<b>T5.1</b>	Scan valid loyalty card and multiple products	System should add any loyalty points that are associated with products to the customer's account automatically.
<b>T5.2</b>	Check loyalty card balance	System should display the customer's loyalty card balance.
<b>T5.3</b>	Redeem reward with correct number of points in account	System should allow the selected reward to be redeemed. The points cost of the reward should be removed from the customer's account.
<b>T5.4</b>	Redeem reward with too few loyalty card points in account	System should not allow the reward to be redeemed and should display error message. E.g. Not Enough Points
<b>T6.</b>	<b>Loyalty Card Sign Up</b>	
<b>T6.1</b>	Sign up with valid email, phone number and address	The system should allow the customer to sign up for a loyalty card and a new card number should be issued.
<b>T6.2</b>	Sign up with invalid email	The system should not allow sign up and should display error message. E.g. Invalid Email
<b>T6.3</b>	Sign up with invalid phone number	The system should not allow sign up and should display error message. E.g. Invalid Phone Number
<b>T6.4</b>	Sign up with invalid address	The system should not allow sign up and should display error message. E.g. Invalid Address

# TESTING

<b>T7.</b>	<b>Inventory System</b>	
<b>T7.1</b>	Scan item by barcode	The system should automatically update the stock of that item.
<b>T7.2</b>	Manually input product number	The system should automatically update the stock of that item.
<b>T7.3</b>	Update stock	The system should allow the user to manually update the stock of a product.
<b>T7.4</b>	Update price	The system should allow the user to manually update the price of a product. The item should then appear at its new price when next scanned.
<b>T7.5</b>	Add product to inventory	The system should allow the user to add a new product to the inventory.
<b>T7.6</b>	Scan new item	The system should scan the product as it would any other product.
<b>T7.7</b>	Remove product from inventory	The system should allow the user to remove a product from the inventory.
<b>T7.8</b>	Scan removed item	The system should not allow a removed product to be scanned and should display an error message. E.g. Invalid Barcode
<b>T7.9</b>	Adjust loyalty points earned	The system should allow the user to adjust the number of loyalty points earned by purchasing a certain product. The item should then have its updated loyalty point amount when next scanned.
<b>T7.10</b>	Scan out of stock item	The system should display an error message. E.g. This Product is Out of Stock
<b>T8.</b>	<b>Receipt Generation</b>	
<b>T8.1</b>	Print Receipt	The system should print a receipt with a list of products purchased, prices, loyalty points earned, payment method and total amount paid.
<b>T8.2</b>	Print Receipt when out of paper	The system should display an error message. E.g. No Paper in Printer
<b>T9.</b>	<b>Staff Sign In</b>	
<b>T9.1</b>	Sign in with correct username and password	The system should allow the user to sign in and access the system.
<b>T9.2</b>	Sign in with incorrect username	The system should display an error message. E.g. Invalid Username
<b>T9.3</b>	Sign in with incorrect password	The system should display an error message. E.g. Incorrect Password.
<b>T10.</b>	<b>User Interface</b>	All buttons in the UI should function correctly when pressed.

Figure 38. Table of Test Cases

# TESTING

## 5.5 PROTOTYPE TESTING

Below is a testing log for a prototype of the system. It does not include every test case that is included in the overall system as it is only a prototype. However, it does include the actual outcomes, and any adjustments made during the prototype testing.

NUM.	TEST CASE	STEPS	EXPECTED OUTCOME	ACTUAL OUTCOME	RESULT (PASS/FAIL)
<b>T1.</b>	<b>Main Menu</b>				
<b>T1.1</b>	Check Main Menu Display	Run the program. Observe the output when the main menu is presented.	"WELCOME TO THE GROCERY STORE!" and menu options shown.	"WELCOME TO THE GROCERY STORE!" and menu options shown.	PASS
<b>T1.2</b>	Navigate to Inventory System	When Main Menu appears, select option '1'.	The program displays the Inventory System Menu.	INVENTORY SYSTEM MENU: 1. View Inventory 2. Add Product 3. Update Product Quantity 4. Export Inventory to JSON 5. Back to Main Menu Choose an option:	PASS
<b>T1.3</b>	Navigate to Checkout System	When Main Menu appears, select option '2'.	The program displays the Checkout System Menu.	Staff Login + Inventory/Products  CHECKOUT SYSTEM MENU: 1. Add an item to the cart 2. Remove/Edit an item in the cart 3. View cart 4. Proceed to checkout 5. Back to Main Menu Choose an option:	PASS
<b>T1.4</b>	Navigate to Loyalty Card System	When Main Menu appears, select option '3'.	The program displays the Loyalty Card System Menu.	LOYALTY CARD SYSTEM MENU: 1. Add Customer 2. Record Transaction 3. Redeem Reward 4. Add Reward	PASS

# TESTING

				5. Export Data to JSON 6. Back to Main Menu Choose an option:	
<b>T1.5</b>	Exit the Program	When Main Menu appears, select option '4'.	The program exits and shows a goodbye message.	“Exiting the Grocery Store System...”  “Thank you! Exit Successful! Signing Off!”  “See you again soon!”	PASS
<b>T1.6</b>	Invalid Option Handling	At the Main Menu, enter an invalid option (e.g., '5').	An error message is displayed indicating invalid option.	“Invalid option! Please try again!”	PASS
<b>T1.7</b>	Main Menu Loop	After interacting with any menu option, verify that the main menu reappears when choosing to return to the main menu.	Main Menu should reappear for further actions.	Main Menu should reappear for further actions.	PASS
<b>T2. Inventory</b>					
<b>T2.1</b>	View All Products	Open Inventory System and select option to view inventory.	A list of all products with their details is displayed.	A list of all products with their details is displayed.	PASS
<b>T2.2</b>	Add New Product	Select the option to add a product. Input valid details (Product ID, Name, Price, Quantity, Category ID).	Product is added successfully, and confirmation message is shown.	Product is added successfully, and confirmation message is shown.	PASS
<b>T2.3</b>	Update Product Quantity	Select option to update product quantity. Input a valid product ID and a positive quantity to subtract.	Product quantity is updated successfully, confirmation message displayed.	Product quantity is updated successfully, confirmation message displayed.	PASS
<b>T2.4</b>	Export Inventory to JSON	Select the option to export inventory to JSON. Provide a valid filename.	Inventory data is exported successfully to the specified JSON file.	Inventory data is exported successfully to the specified JSON file.	PASS
<b>T2.5</b>	Initialize Products on First Run	Run the application for the first time with the Inventory System.	Default products and categories are added to the database.	Default products and categories are added to the database.	PASS

# TESTING

<b>T2.6</b>	Fetch Product Details	Call the method to fetch details for an existing product ID.	Details of the product (ID, Name, Price, Quantity) are returned.	Details of the product (ID, Name, Price, Quantity) are returned.	PASS
<b>T3.</b>	<b>Checkout</b>				PASS
<b>T3.1</b>	Login to Checkout System	Start the Checkout System and input valid credentials (username and password).	Successful login message displayed.	Successful login message displayed.	PASS
<b>T3.2</b>	Display Products	After login, display available products for purchase.	A list of all available products is shown.	A list of all available products is shown.	PASS
<b>T3.3</b>	Add Valid Product to Cart	Select a valid product, input a valid quantity to add to the cart.	Product is added to the cart, and confirmation message is shown.	Product is added to the cart, and confirmation message is shown.	PASS
<b>T3.4</b>	View Cart	Select the option to view items in the cart.	Display all items in the cart with details and total amount.	Display all items in the cart with details and total amount.	PASS
<b>T3.5</b>	Edit Product Quantity in Cart	Select an item from the cart and attempt to update its quantity (valid quantity).	Quantity of the product is updated successfully, with confirmation.	Quantity of the product is updated successfully, with confirmation.	PASS
<b>T3.6</b>	Remove Product from Cart	Select a product from the cart to remove.	Product is removed from the cart, and confirmation message is shown.	Product is removed from the cart, and confirmation message is shown.	PASS
<b>T3.7</b>	Checkout with Sufficient Cash	Select the checkout option, choose cash payment, and provide an amount greater than or equal to the total.	Successful payment message with change is displayed.	Successful payment message with change is displayed.	PASS
<b>T3.8</b>	Checkout with Insufficient Cash	Attempt to check out with cash and provide an amount less than the total.	Error message indicating insufficient funds.	Error message indicating insufficient funds.	PASS
<b>T3.9</b>	Checkout with Credit Card	Select the checkout option, choose card payment.	Successful payment message is displayed.	Successful payment message is displayed.	PASS
<b>T4.</b>	<b>Loyalty Card</b>				
<b>T4.1</b>	Add Customer	1. Enter valid customer details (first name, last	The customer is successfully added to the system.	The customer is successfully added to the system.	PASS

# TESTING

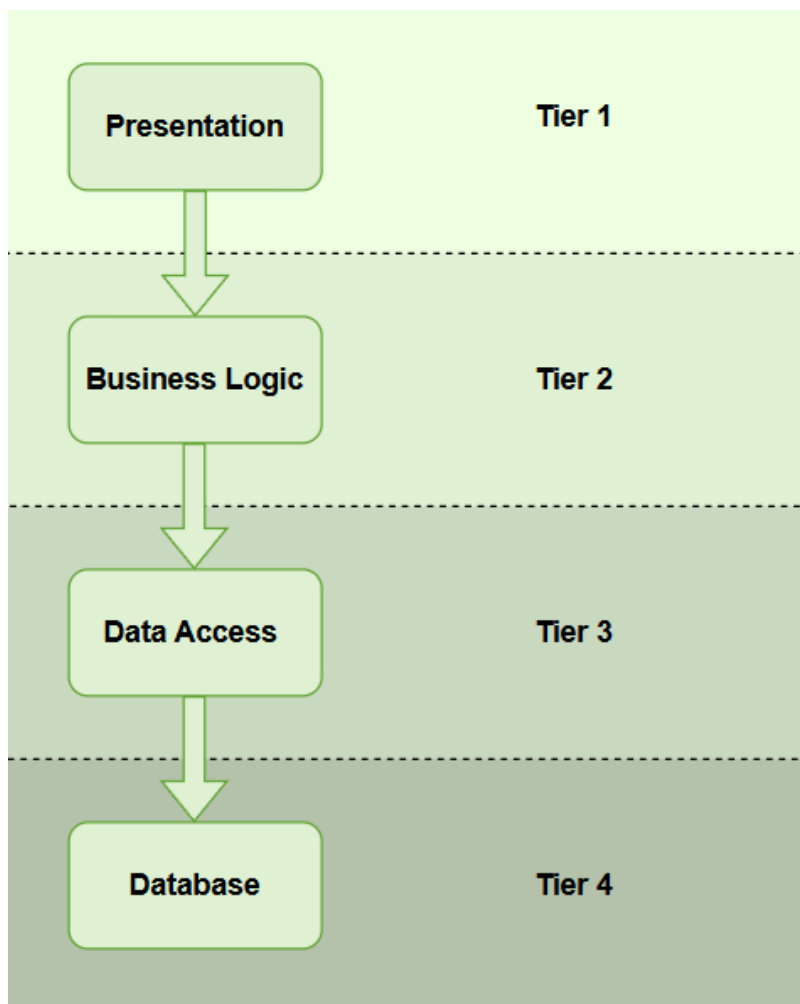
		name, email, phone number, address, card number, issue date, expiry date).			
<b>T4.2</b>	Record Transaction	1. Type customer ID.  2. Enter transaction date and total amount.	The transaction is successfully recorded in the system.	The transaction is successfully recorded in the system.	PASS
<b>T4.3</b>	Redeem Reward	1. Type customer ID.  2. Select a reward ID.  3. Enter redemption date.	The reward is successfully redeemed for the customer.	The reward is successfully redeemed for the customer.	PASS
<b>T4.4</b>	Add Reward	1. Enter reward name, description, and points required.	The reward is successfully added to the system.	The reward is successfully added to the system.	PASS
<b>T4.5</b>	Export Data to JSON	1. Select a table name (Customer, Transactions, Reward, Reward Redemption). 2. Enter a file name for the JSON export.	The data is successfully exported to a JSON file.	The data is successfully exported to a JSON file.	PASS

Figure 39. Table of Prototype Testing Log

## 6. SYSTEM ARCHITECTURE

### 6.1 SYSTEM ARCHITECTURE OVERVIEW

For this PoS system the design team opted to use a tiered architecture which is displayed in the diagram below. This style of architecture was chosen for many reasons which will be explained in Section 6.2, Architecture Justification. Overall, this architecture is what best meets the requirements of the stakeholder.



*Figure 40. Image of Tiered Architecture Diagram*



# SYSTEM ARCHITECTURE

## 6.1.1 TIER 1 PRESENTATION TIER

The Presentation Tier is the frontend of the entire system. For this grocery store checkout system, this is the display at the checkout that the sales assistant uses to process transactions. If the stakeholder were to expand their business, this tier would include things such as self-checkout screens and an online store which is covered in more detail in Section 9, Future Enhancements. This tier uses an API gateway to communicate with the Business Logic tier.

## 6.1.2 TIER 2 BUSINESS LOGIC TIER

The Business Logic Tier is the start of the backend of the system. For the grocery store checkout system, this is the tier which contains the fundamental code which allows the checkout to operate. This includes scanning items, calculating a total and payment processing. This tier is also where loyalty card scans happen, and inventory updates happen. These events are processed in the Business Logic Tier and sent as requests to the Data Access Tier. This tier will also perform functions such as asking the Data Access tier to validate a loyalty card.

## 6.1.3 TIER 3 DATA ACCESS TIER

The Data Access Tier acts as an intermediary between the Business Logic tier and the database tier. It takes requests from the Business Logic tier such as to update inventory after a product is purchased or to validate a loyalty card and translates them into operations to apply to the Database tier. After an operation has been performed on the Database tier, the Data Access can send results of any information request back to the Business Logic tier such as when validating a loyalty card or scanning a product. It also makes it easier to make changes to the databases without interfering with the Business Logic Tier. This tier controls all interactions with the Database tier.

## 6.1.4 TIER 4 DATABASE TIER

The Database Tier is where all the required databases for the system are stored. For the grocery store checkout system, this will include the inventory database and the loyalty card database. It serves as a place to store important data and can perform updates requested by the Data Access tier such as changing the stock level of a product.

# SYSTEM ARCHITECTURE

## 6.2 ARCHITECTURE JUSTIFICATION

### 6.2.1 SCALABILITY

As outlined in Requirement 11 in Section 2.2, Table of Non-Functional Requirements, it is important to the stakeholder that the system can be scaled up in case of future expansion. While a monolithic architecture would be suitable for the stakeholder's current situation with a single grocery store, this architecture would quickly become inefficient if further checkouts are opened or if the stakeholder wanted to open another store. In these scenarios the stakeholder would need multiple copies of the same system at each individual checkout and individual databases for each system. Not only is this inefficient but would also render the loyalty card system non-functional as a customer would have to sign up each time at different stores.

However, with the proposed 4 tier architecture only the first tier, the Presentation tier, would need to be installed on the checkouts, while the Business Logic tier, Data Access tier and the Database tier can be stored on the cloud. The checkout hardware would interact with the Business Logic tier through API gateways allowing the Business Logic tier to be stored in the cloud in addition to the Data Access tier and Database tier. This ensures that every checkout can use the same logic and databases as each other, no matter if they are from different grocery stores. This creates more potential for the business to be scaled as opening new checkouts or stores is simpler and multiple stores can act as a franchise rather than individual stores.

### 6.2.2 PERFORMANCE

As outlined in Section 1.3, Context, the stakeholder's current checkout system is inefficient and takes too long. By implementing a tiered architecture system this can be vastly improved. Firstly, with this tiered architecture the system can use caching across multiple tiers to improve performance. By using caching in the Presentation tier the checkout system will load quicker, allowing for a quicker transition between customers which is particularly useful, if the stakeholder wants to eventually implement self-checkout machines. In addition to this, the Business Logic tier can use in-memory caching for operations that are frequently being used such as scanning items. This improves speed of the entire system which also improve the scalability of the system.

# SYSTEM ARCHITECTURE

Furthermore, by implementing a tiered system architecture, workloads are spread evenly across the entire system instead of concentrating on a single system. This reduces the complexity of the system and also reduces the likelihood of any bottlenecks appearing within the system.

## 6.2.3 MAINTAINABILITY

By using a tiered architecture if part of a system fails, the system may be able to continue running which is highly important to a grocery store as the checkout needs to function for the entirety of store opening hours as outlined in Requirement 8, in Section 2.2, Table of Non-Functional Requirements. It also allows for faster bug-fixing as any bugs in the system become quicker to find and also easier to fix. In addition to this, any bug fixes that occur within a tier are unlikely to have unforeseen consequences in another tier, therefore ensuring that the system is able to keep running effectively.

## 6.2.4 SECURITY

By using a tiered architecture, the entire system becomes more secure which is highly important as customer data is stored on the databases. On the current system data can only be accessed by someone in the actual store. However, as explained previously if the stakeholder wants to expand having a monolithic architecture will no longer be possible. With a tiered architecture data can be made more secure as each tier can have its own security. Any information sent between tiers can be encrypted and information stored on databases can be more heavily encrypted.

Furthermore, instead of having all the information in a single system where if someone does access the system they can get to any data, a tiered architecture only allows a tier to have access to the information it needs. Therefore, if someone was to access a specific tier, they would not be able to find much information from other tiers. This is particularly important as there are databases in the system, so as long as the databases are encrypted to a point where they cannot be accessed by anyone other than those the stakeholder grants access, the information stored in the databases cannot be accessed through a different tier.

# SYSTEM ARCHITECTURE

## 6.3 ARCHITECTURE IMPLEMENTATION CONSIDERATIONS

### 6.3.1 TIER 1 PRESENTATION TIER CONSIDERATIONS

To implement the Presentation tier and create the UI some sort of frontend framework would need to be used to ensure it is functional. This applies to the checkout UI as well as if the stakeholder wants a UI for a self-checkout or an online store. The design would need to be simple and intuitive so new employees can learn to use the checkout system easily. There will need to be an authentication mechanism to ensure that only staff members can log in which only applies to the current checkout system. While it would not currently be required for this system, a CDN (Content Delivery Network) could be used in the future for the UI if multiple stores were opened. This would eliminate the need for on-site servers.

### 6.3.2 TIER 2 BUSINESS LOGIC TIER CONSIDERATIONS

To implement the Business Logic tier, the majority of the software would not require any external mechanisms as the checkout system would just use some code. However, a caching mechanism would be required to improve performance, and an API gateway would be needed in order to communicate with the Data Access tier. If the stakeholder wanted to create an online store, certain payment methods such as Stripe could be implemented which is covered more in Section 9.

### 6.3.3 TIER 3 DATA ACCESS TIER CONSIDERATIONS

To implement the Data Access tier a caching service may be used to ensure performance is fast and reliable. The tier would also need to be able to deal with errors in case a request fails and from there find an alternative way to process the request. Finally, all information that the data access layer uses would need to be encrypted as it accesses the databases where all the information related to the grocery store is held.

## 6.3.4 TIER 4 DATABASE TIER CONSIDERATIONS

To implement the Database tier a robust encryption service must be used to ensure that no one can access the databases unless they have been given permission by the stakeholder. The entire tier would also have to regularly perform backups so that if the system does happen to crash, it can restart with the most up to date information possible. The tier must also be capable of dealing with a high volume of requests at a single time which is likely to occur if the stakeholder decides to expand.

## 6.3.5 COST CONSIDERATIONS

By implementing a tiered architecture there are various effects on cost. There would be less of an upfront cost to the stakeholder as they would no longer need to purchase a computer or laptop to run the checkout software. They also would not need to purchase any physical servers, instead paying a subscription-based fee to use server space elsewhere. However, despite the reduced upfront cost the cost in the long term is increased as in order to effectively implement a tiered architecture the stakeholder would need to use some sort of subscription service such as AWS. This cost would be detrimental to the business if it continued to operate as a single store, however, if the stakeholder chose to expand the cost of using these services would be better than having to purchase computers and servers for every store. Furthermore, it allows the stakeholder to create an online store which could result in an increase in customers.

## 7. SYSTEM IMPLEMENTATION

### 7.1 HARDWARE

In order for the Grocery Store checkout system to be fully implemented all the hardware identified in Section 2.1, Table of Functional Requirements, would need to be purchased and installed in the store by the stakeholder before the implementation of the software. This includes a touchscreen monitor, a cash register, a card reader, a barcode scanner, a receipt printer, a set of scales, a laptop or computer and a stable internet connection.

### 7.2 SOFTWARE

The system will be coded using Python 3.13 as the coding language as it is a fairly universal language and is simple to understand so if any bugs are found they can be fixed even if no one from the design team is available. The checkout software will be able to access the system's databases and modify them. As a result, the software will need to be secure to ensure it cannot be accessed by anyone who is not an employee of the store. An automated system tester can be utilised to speed up the testing process, but this is not mandatory.

### 7.3 DATABASES

The database will be created using SQLite 3. It is essential that the databases are secure and all information that is sent from them to other parts of the system is properly encrypted as the databases contain personal information such as phone numbers, email addresses and home addresses.

### 7.4 ARCHITECTURE

The system architecture can be implemented using a subscription-based service such as AWS. The decision of which service to use should be left to the stakeholder as they will be the one paying for it. This decision must be made before the system is implemented.

## 8. SYSTEM MAINTENANCE

### 8.1 MAINTENANCE PLAN

The initial Maintenance Plan for the Grocery Store Checkout system would include a maintenance schedule. This schedule would give an outline for when maintenance will occur, this should be periodic, and it is recommended that after the initial release, the system should receive weekly maintenance. As part of the plan the maintenance team should monitor any diagnostic data sent from the system to identify problems early. Future updates can be designed towards any problems found. The system will store data on loading times and how fast certain actions are completed as well as identifying any errors that occur. This information can then be looked at by the maintenance team and future system updates can be planned. Any updates or maintenance should take place outside of store opening hours and be finished before the store opens.

### 8.2 MAINTENANCE TASKS

The main tasks of maintenance are as follows:

- **Corrective Maintenance** – This is where maintenance is used to fix bugs in the code and address any problems with the code.
- **Adaptive Maintenance** – This is where maintenance is used to make enhancements to the system and add new features to the system.
- **Perfective Maintenance** – This is where maintenance is used to improve efficiency within a system and make it more reliable. This is important to the stakeholder as efficiency is the main problem with their current system.
- **Preventive Maintenance** – This is where maintenance is performed periodically to identify problems early. After the initial release of the system this is the type of maintenance that will be performed.

# SYSTEM MAINTENANCE

## 8.3 SECURITY

As a significant portion of the system will be stored and run on the cloud, it is essential that there is effective security to protect employee and customer data and ensure local laws are followed. The system ensures this as each component of the system only has access to the data it requires, therefore ensuring that data cannot be accessed in different parts of the system. Furthermore, all databases and information sent to and from databases will be encrypted. As the system architecture will be implemented using a service like AWS, there will be in-built security features, which reduces the cost of implementing new security features.

## 8.4 RECOVERY & BACKUP

In order to ensure that the system can still operate even if parts of it critically fail, it is important that certain parts of the system are appropriately backed up. It is recommended that the primary system database is backed up constantly to a second inactive database. Therefore, if the primary database was to fail, the secondary database could quickly be used instead. Additionally, there should be backups of the software stored on drives kept in different locations. Therefore, if any new hardware is installed the software can quickly be installed onto it. It is also important to keep redundant hardware in the store so that if there is a failure the hardware can be quickly replaced, and the store will not lose out on sales. If the store was to expand it could be worthwhile having a redundant system that is operating to reduce down times.

## 8.5 USER SUPPORT

Initially, basic training on how to operate the system should be provided to the stakeholder, who should then be able to train their employees. The system should be fairly intuitive so very little training should be required. A user guide should also be provided on how to restart the system if it goes down. This guide should include information on how to install software onto new hardware and how to connect the system to the secondary database if the primary one was to fail. If the stakeholder or their employees are still unable to get the system running, there will be an email and phone number provided, that they can contact to talk to one of the members of the maintenance team.



## 9. FUTURE ENHANCEMENTS

### 9.1 SCALABILITY

The stakeholder has expressed interest in expanding their business in the near future. Therefore, the design team has included this section of the document detailing how the stakeholder's business could potentially expand and giving a roadmap for that expansion. This section will include detail of up scaling within an individual store and across a network of stores to increase the number of customers that make purchases from the stakeholder's business. In particular, the design team has looked at how a self-checkout system could potentially be implemented as well as how an online store that operates across multiple physical stores could work.

If the stakeholder chooses to expand and scale up their business in the future the necessary framework has been implemented into this system to allow them to expand without many system issues. By using the tiered architecture system outlined in Section 6, System Architecture, the stakeholder will be able to expand their business across multiple stores while having a combined customer loyalty card system. This architecture can also be used for different methods of purchasing products whether that be through an online store or through a self-checkout system.

# FUTURE ENHANCEMENTS

## 9.2 SELF-CHECKOUT SYSTEM

### 9.2.1 SELF-CHECKOUT IMPLEMENTATION

This section of the document will detail how a self-checkout system could potentially be implemented into the stakeholder's store. This would allow for a quicker checkout process as customers would be able to use the self-checkout system if there is a queue for the regular checkout. This could also reduce costs as less sales assistants would be needed to operate checkouts as opposed to installing more regular checkouts. The implementation of self-checkouts would also be more space effective as the space that is usually barred off from customers for the sales assistant would not be required to be expanded which is important as the stakeholder currently owns a small sized grocery store.

#### 9.2.1.1 REQUIREMENTS

In order to implement a self-checkout system there would be a few requirements that the stakeholder or design team would need to fulfil to ensure that the new system works just as well as the current checkout system. They are as follows:

- The stakeholder would need to purchase the self-checkout hardware which is usually a single machine.
- The checkout UI would need to be adapted from the one the sales assistant has, to one that is better for customers to use. This is because the sales assistant will have access to certain features that you would not want customers to have access to.
- The system would have to function with the checkout system outlined previously in this document. This includes compatibility with the inventory system and customer loyalty card system.
- The stakeholder would need to ensure that the self-checkout machines feature some sort of anti-theft measures such as automatic product weight checks.
- All members of staff would need to be retrained to understand how the self-checkout system works in case a customer needs help using the system.

# FUTURE ENHANCEMENTS

## 9.2.1.2 ADVANTAGES

As part of the process of laying out a roadmap for self-checkout implementation the design team analysed the potential advantages of implementing the system. They are as follows:

- The self-checkout system would be simple to integrate into the current checkout system. This ensures that the inventory system and loyalty card system will both be fully operational in the self-checkout system.
- The self-checkout system could help to reduce the amount of queuing inside the stakeholder's store.
- The self-checkout system takes up less space than a conventional checkout allowing for more customers to be served at the same time.
- Less staff would need to be in the store at the same time, therefore reducing the stakeholder's operating costs.

## 9.2.1.3 DISADVANTAGES

As part of the process of laying out a roadmap for self-checkout implementation the design team analysed the potential disadvantages of implementing the system. They are as follows:

- The initial cost of implementing the self-checkout system would be quite high as new hardware would have to be purchased.
- A new UI would have to be designed for the self-checkout system as the checkout UI would have features that should not be accessed by customers.
- Staff would have to be trained on how to use the self-checkout system in order to be able to help customers who may struggle to use the system.

# FUTURE ENHANCEMENTS

## 9.2.2 SELF-CHECKOUT UI

As the UI would need to be differentiated between the regular checkout system and the self-checkout system below are some mock ups of how the two UIs could potentially look as well as the key differences between them. It is important to note that these UIs are just an example and are not fully representative of the final system UI.

### 9.2.2.1 CURRENT CHECKOUT UI

Item	Price
Item A	£1.00
Item A x 2	£2.00
Item B	£4.00
Item C (Discounted)	£2.00
	- £1.00
Item D	£3.00

**Total: £11.00**

Item Search

Item Quantity

New Loyalty Card

Pay By Cash

Enter Product Number

Override Price

Enter Loyalty Card Number

Pay By Card

Remove Item

Add Discount or Reward

Check Loyalty Card Points

Duplicate Receipt

Sign Out

Cancel Transaction

Figure 41. Visualisation of Checkout UI

# FUTURE ENHANCEMENTS

## 9.2.2.2 UPDATED UI

Item	Price
Item A	£1.00
Item A	£1.00
Item A	£1.00
Item B	£4.00
Item C (Discounted)	£2.00 - £1.00
Item D	£3.00
<b>Total:</b>	<b>£11.00</b>

Enter Product Number

Item Search

Remove Item

Enter Discount or Reward Code

Enter Loyalty Card Number

Check Loyalty Card Points

Help

Cancel Transaction

Finish & Pay

Figure 42. Visualisation of Self-Checkout UI

## 9.2.2.3 KEY DIFFERENCES

- **Text Size** – In the self-checkout UI the text size has been greatly increased in order to account for someone who is untrained at operating the display. The increased text size allows for greater clarity and ease of use for a customer.
- **Item Buttons** – The self-checkout UI no longer allows the operator to adjust product quantity, override prices or to add discounts. This is because only a trained employee should be allowed discretion as to if a price should be manually changed and the general public should not be trusted with the ability to change prices. The self-checkout UI does allow the user to input discount or reward codes however, but these discounts and rewards will be preset, and the rate of discount cannot be adjusted.
- **Loyalty Card Buttons** – The self-checkout UI allows for a lot of the similar functionality as the checkout UI in regards to loyalty cards. However, it does not allow the option to issue a new loyalty card as only a trained employee should be able to do that.

# FUTURE ENHANCEMENTS

- **Admin Buttons** – The regular checkout UI has two admin buttons, sign out and cancel transaction. The self-checkout UI does not require a sign out button as no one has to sign in to the checkout to be able to operate it. However, it does have a help button, to allow customers to signal that they require assistance with the checkout. A member of staff should then be able to help the customer.
- **Payment Buttons** – For the regular checkout UI the sales assistant is able to choose between cash payment and card payment (chip and pin or contactless) which allows for either the cash register or the card reader to be activated. For the self-checkout UI this process has been simplified into a single finish & pay button which will allow the customer to pay. This may take them to a second screen where they can choose their payment method which is simpler and more streamlined than the regular checkout UI.

## 9.2.3 SELF-CHECKOUT SUMMARY

To summarise, the design team believes that if the stakeholder were to be in a position where their store is attracting too many customers, but the stakeholder is not in a position to open another store, a self-checkout system would be a viable solution. While the initial cost of the self-checkout system would be high, the fact that the regular checkout system, outlined previously in this document, could be implemented into the self-checkout system without much difficulty would be beneficial and in the long term the initial cost of the system hardware would be offset in the store's capability to deal with more customers.

# FUTURE ENHANCEMENTS

## 9.3 ONLINE STORE

If the stakeholder were to expand and in doing so open multiple stores all as part of the same franchise, it is possible that they would require an online store. This is due to the fact that the stakeholder would not be able to guarantee that each store would be the same size. Therefore, as some stores may be larger, some stores would be able to stock products that some of the other stores cannot. Alternatively, the stakeholder would not be able to control stock levels at each store and certain stores may be out of stock of a certain product. Assuming that the stores would vary significantly in location and not all be centralised, this would mean that certain customers would have to travel significant distances in order to purchase certain products. This could lead to customers looking to other places to purchase the product they want. In order to counteract this, the design team would like to propose the creation of an online store if a multiple store expansion were to take place.

### 9.3.1 ONLINE STORE DESIGN

The design team suggests that the online store, if it were to be implemented, would work as a Click & Collect system. A customer would be able to view the entire catalogue of products across all the stores in the stakeholder's franchise. The customer would then be able to purchase their desired product, then the customer would select a store to pick the item up from and pay for it online. This would allow the customer to purchase products that they would have to travel a significant distance to buy in person. The customer would receive a notification after the product is delivered to their local store, at which point they can collect the product. This is demonstrated in the diagram below.

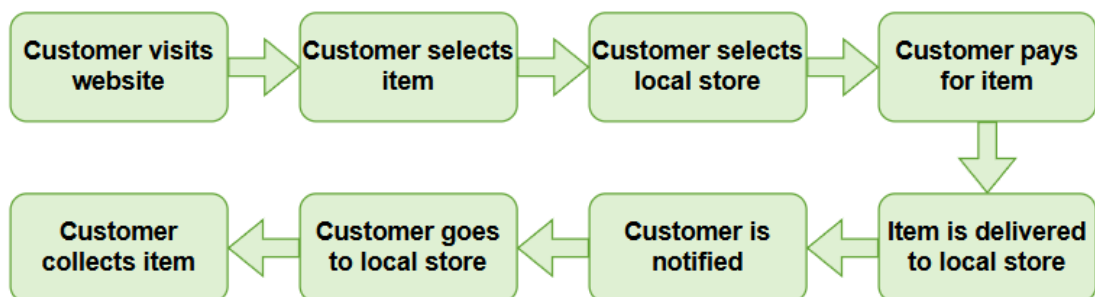


Figure 43. Visualisation of Online Store Process

# FUTURE ENHANCEMENTS

## 9.3.1.1 IMPLEMENTATION WITH CURRENT SYSTEM

The main issue with the implementation of the online store is the website itself. The online store should function with all other parts of the checkout system outlined in this document. It should offer the customer the opportunity to enter their loyalty card number to earn points and apply discounts.

The inventory system would need to be slightly adjusted as each individual store would have their own stock list. Therefore, the online store system would need to be able to access the stock lists of every store and update the stock lists of individual stores if an item is purchased from a certain store and sent to a different one. This can be implemented in the Business Logic tier of the system architecture.

One area that would need to change is the Presentation Tier of the system architecture. The Presentation tier would have a completely different UI to the checkout system. A CDN could be used to help the online store load quickly.

The Business Logic tier would need slight modification to allow customers to add products to a basket before paying. Adding a product to the basket would be the equivalent of scanning the item at the checkout as once the product is put in the basket, its product number is sent to the Data Access tier, and the item price is returned. The processes after that are similar but the payment system would need to be changed, and an e-receipt would have to be sent by email to the customer. This would mean that all customer's using the online store would have to submit their email which is different to the established checkout system. This email can be used to provide an order number to the customer and to notify the customer once their item is ready. There would also need to be option for the customer to select their local store.

The other tiers, Data Access and Database, would need little to no modification as long as they are operating quickly and efficiently. There may need to be a few modifications to the databases themselves, such as a database with store locations, but these would have been implemented when multiple stores were opened.



# FUTURE ENHANCEMENTS

## 9.3.2 STRIPE INTEGRATION

One area of the checkout system that would not be able to be implemented in an online store is the payment system. In the checkout system outlined in this document a physical card reader or cash drawer is required to accept payment. This is not possible with an online store, so a different form of payment needs to be integrated.

In order to implement a payment method into the online store the design team has recommended that Stripe should be integrated in the online store's design. Stripe would allow the online store to accept various forms of digital payment which is ideal for an online store where the method of payment that a customer will use is unknown. It is also capable of dealing with large amounts of traffic at the same time which allows the online store to be quick and avoids having to implement queues to get to checkout, if a large number of people were to try and purchase items at the same time. While this scenario is unlikely, stores tend to see an increase in traffic around events such as Black Friday and Christmas, so it is better for the online store to be capable of dealing with extra people.

Stripe also has in-built API gateways and security measures. This would help the stakeholder as they would not have to worry about implementing more security on their online store. They would just have to ensure that the checkout system outlined in this document has appropriate security. The integrated API gateway would allow the Stripe checkout system to work seamlessly with the rest of the checkout system.

The only issue as far as the stakeholder is concerned is that there would be an additional subscription-based fee that they would have to pay to be allowed to have Stripe integrated into their online store. However, as previously outlined, it is likely that an online store would not be created until after an expansion, therefore the business should have more money coming in order to cover the extra costs.

# FUTURE ENHANCEMENTS

## 9.3.3 WEB GUI DESIGN

In order to operate effectively an online store would need a simple and easy to use UI. Below is a visualisation of what a potential UI could look like for an online store.



Figure 44. Visualisation of Online Store UI

- **Simplicity** – The online store GUI has been designed to be as simple and easy to use as possible. This ensures that customers are able to navigate the online store and purchase the products that they want. The customer simply has to click on a product and select add to basket to send the product to their basket.
- **Store Location** – At the top of the GUI is a drop-down menu which allows the customer to select their local store. This is important as the entire point of the online store is to allow customers to purchase products that are not in their local store. This choice would be confirmed once in the checkout screen.

# FUTURE ENHANCEMENTS

- **Basket** – The basket contains information on which products the customer has selected. Once in the basket the customer can choose quantity of the product or if they want to remove the product from their basket. They can then choose to checkout which will allow them to enter their details and pay via Stripe.
- **Implementation** – In order to implement the online store UI, a website would have to be set up which may require a subscription service. The stakeholder may also wish to purchase a chosen domain name if they want the websites URL to contain certain words. Once the website is created, the UI itself can be implemented using Python's in-built GUI library, Tkinter. Alternatively, other frameworks such as PyQt or PySide could be used.

## 9.3.4 ONLINE STORE SUMMARY

In summary, the design team believes that if the stakeholder expands their business to the point that they own multiple stores across a variety of locations, that the implementation of an Online Click & Collect Style Store may be useful. The store itself should be able to use the inventory and loyalty card systems that have been outlined in this document. However, in order to implement an Online Store, it is important that a payment system and Web GUI are also correctly integrated. For example, using Stripe for the payments as outlined in Section 9.3.2, Stripe Integration, would be optimal to sort out the payment situation. The design team has also provided an example GUI, which could be used as a template for the Online Store UI.

## 10. SUMMARY

To summarise, this document has detailed the design of a checkout Point of Sale system for a small sized grocery store. This document has shown the key functional and non-functional requirements for this system and has identified a few high-priority requirements, in order to influence and explain any design choices.

Furthermore, through the use of UML Use Case, Class and Sequence diagrams, this document has shown the high-level design for the checkout system and how various components will work together. It has also shown how an inventory system and loyalty card system will function and how they will be integrated into the overall checkout system.

In addition, this document has explained how the primary database for the system will be designed using SQLite3, including demonstrating all relevant tables and fields as well as identifying primary and foreign keys. This document also includes a variety of information on how testing will be conducted for this system, including the identification of test cases and a prototype testing log. Most of this testing will be White Box testing and will follow the testing V-Model.

This document has also outlined the design of a 4-tier system architecture. This was chosen in order to aid security but also to allow the system to be scaled up if the stakeholder decides to expand. The software for the system will be coded in Python 3.13 as it is simple to use and change if bugs are found.

Additionally, the design team has included information on how the system could be scaled up if the stakeholder was to expand their business. This includes information on how a self-checkout system could potentially be incorporated into the system as well as how an online store could function within the system if the stakeholder were to open multiple stores.

## 11. REFERENCES

- Tan, Z (2024) *Acing the System Design Interview*. Published January 2024
- Sundaramoorthy, S (2022) *UML Diagramming a Case Study Approach*. Published May 2022
- Miles, R; Hamilton, K (2006) *Learning UML 2.0*. Published May 2006
- Halpin, T; Morgan, T (2024) *Informational Modelling and Relational Databases, 3<sup>rd</sup> Edition*. Published July 2024
- Heusser, M; Larsen, M (2023) *Software Testing Strategies*. Published December 2023
- Leloudas, P (2023) *Introduction to Software Testing: A Practical Guide to Testing, Design, Automation, and Execution*, Published May 2023
- Weilkiens, T; G.Lamm, J; Roth, S; Walker, M (2022) *Model-based system Architecture, 2<sup>nd</sup> Edition*. Published April 2022
- Matthes, E (2023) *Python Crash Course 3<sup>rd</sup> Edition*. Published January 2023