POS Software Design Specification

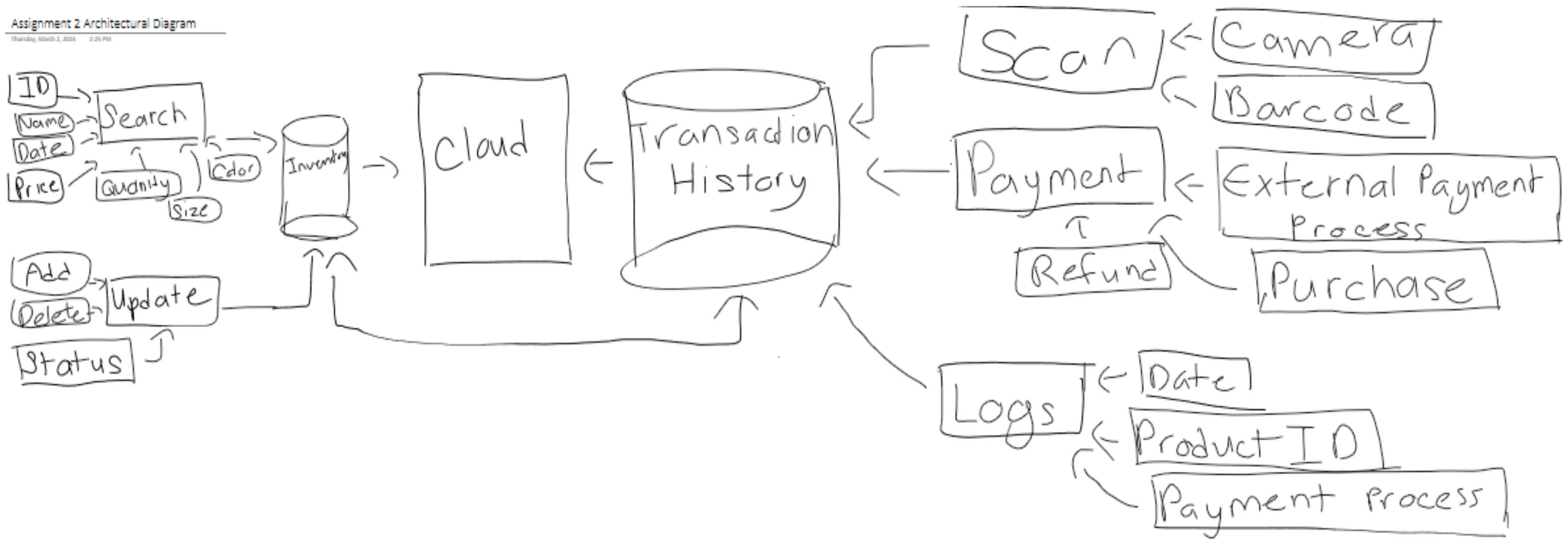
Completed by: Nick Russert, Christopher Ong, Hsa Moo

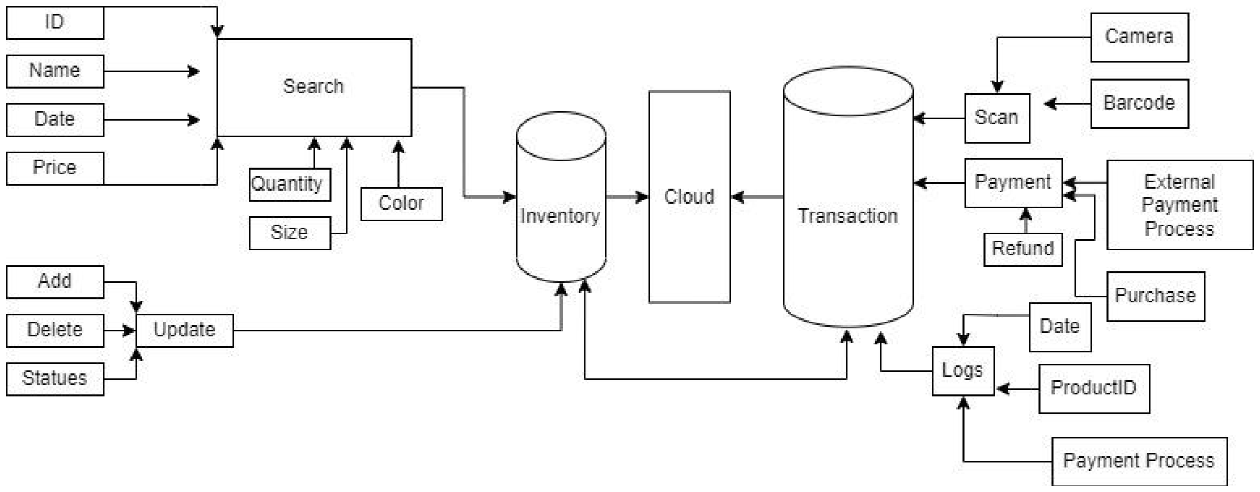
April 21, 2023

1 System Description

This point of sales system will be designed to make daily sales efficient and organized for employee and manager use. It will enable employees and managers to update and view store inventory to see what is available in stock, as well as handle transactions which include purchases and returns. The POS system will accept credit cards, debit cards, and cash and will calculate the total of the transaction including tax. The system will track inventory and after a sale is made, the store inventory will be automatically updated. This system is intended for daily use by both store employees, as well as managers. Further, this document will specify in depth the user requirements, the system requirements, including transactions, refunds, inventory management, and payment histories, and other features needed in the implementation of the software.

2 Software Architecture Overview



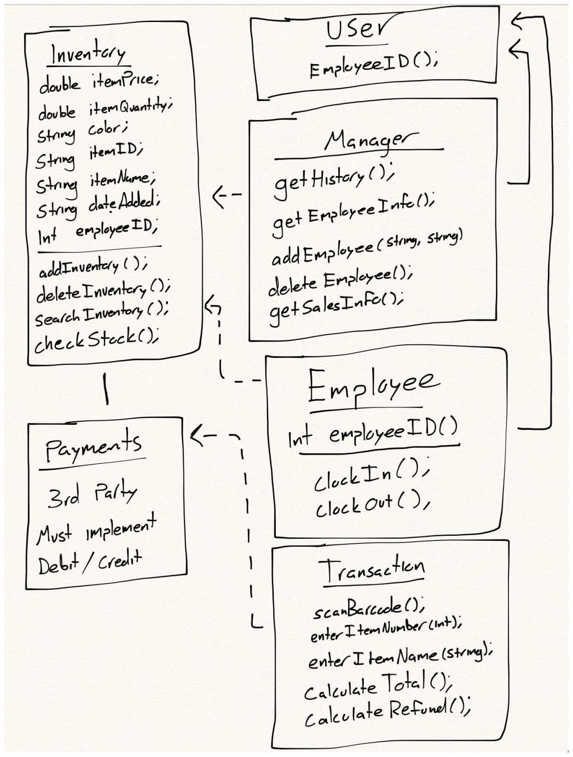


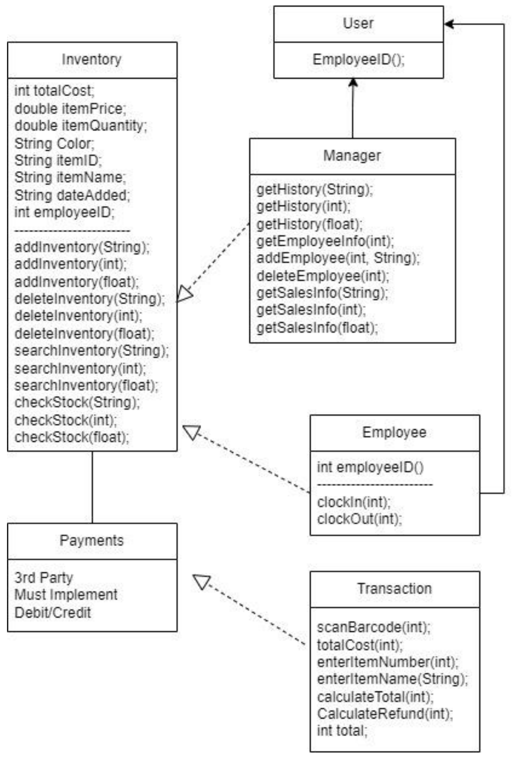
<u>Brief description of Architecture diagram:</u> This architecture diagram consists of two major databases, which are the inventory and transaction history database. These two databases are different from each other, but they are both supported and stored in the cloud. The inventory database has access to methods in relation to search and updating, and is able to access information from the transaction history database. The transaction history database has access to methods in relation to scanning, payment or recording data logs, and is able to access information from the inventory database. For more information about these databases, methods and classes within this architecture diagram, refer to the description of classes section of this document.

Updated Software architecture overview:

The biggest alteration made to this diagram is the fact that it is now digitized. The main databases we use are the inventory and the transaction history, both backed up by the cloud.

UML Class Diagrams:





Brief description of UML class Diagram: One of the main components of the POS software is the inventory. The inventory class is used by almost every other class and holds a lot of data types. The main classes consist of the inventory, the user, manager, employee, transaction, and payments. For more information on these classes, data types, and methods see the description of classes section of this document.

There is also an updated version of the UML diagram. The main differences are the parameter types were added to the functions and this in turn also needed functions to be overloaded with multiple parameter types.

Description of classes:

1. Inventory: The inventory class will be the main database that will store all the objects and functions that will be used by other classes. The objects stored inside the inventory class are:
    a. itemPrice: The price of item
    b. itemQuantity: The number of item
    c. Color: The color of item
    d. itemID: The item's id
    e. itemName: The item's name
    f. dateAdded: The date the item's was added
    g. employeeID: The ID of employee

The function inside the inventory are:

a. addInventory(): Function to add items into inventory
b. deleteInventory(): Function to delete the item in inventory
c. searchInventory(): Function to search for specific item in the inventory

    d. checkStock(): Function to check if item are in inventory and how many are left

2. Manager: The manager class will allow users who are manager to have authorization to view and make changes to the inventory class. The function manager class contain are:
    a. getHistory(): Get the history of transactions and employee clock in and clock out time
    b. getEmployeeInfo(): Get employee information from the inventory
    c. addEmployee(): Add employee into inventory
    d. deleteEmployee(): Delete employee from the inventory
    e. getSalesInfo(): Get the sales information from inventory
3. Payments: The payments class allows 3rd parties to pay using debit/credit for items transferred from the transaction class and it will be stored into the inventory.
4. Transaction: The Transaction class will store items and functions that will be used to make payments:
    a. scanBarcode(): Scans the items barcode and matches it itemID
    b. enterItemNumber(): Allows the user to enter item number
    c. enterItemName(): Allows the user to enter item name
    d. calculateTotal(): Calculates the total cost of an item
    e. calculateRefund(): Calculate the refund total of an item

5. Employee: The Employee class allows users to clock in and clock out using the clockIn() and clockOut() function along with an employeeID() to identify which employee is working or not today and when they are done with work.
6. User: The User class contains the employeeID() which will be used to identify which class between the manager and employee is using the system.

Description of attributes:

There are multiple different types of attributes this system should contain in order to fully operate to the best of its ability. The first attribute is that the user should be able to login into the system, whether it be using their own employee or manager login. This attribute is important since this helps keep track of who's using the system at all times and this attribute provides access for the user to be able to access this system.

Another attribute this system is required to have is the ability to update inventory and search through the inventory. The user should be able to add or remove items within the system to update the inventory of the store and should be able to to search for certain products within the system, whether it be searching for item’s price, the quantity of an item, the color of the item, or based off the item’s ID or name. This is the most important attribute within the system since we want to ensure that we have enough products for the customers to purchase and order any more products we are low on. By having this attribute, this will help the users stay organized with their products and help keep customers satisfied.

The third attribute this system should have is the ability to be able to use a scanner to scan a product's designated barcode when a customer wants to purchase or refund an item and enter the product's name or number to purchase or refund an item. This attribute should be able to update the inventory of the system automatically if a customer decides to purchase or refund a product. This attribute should be also able to calculate the total cost of products purchased or refunded and be connected to an external third-party system to conduct payments. This is an important attribute since this will help the user keep an updated inventory and help conduct product transactions for the customers.

Lastly, there should be an attribute where the user is able to clock in and clock out for their shifts and the manager should be able to search the employee's info and history, be able to add or delete employees within the system, and search the total sales of the store or of certain products. These attributes are intended to help run the work system by being able to clock in and out, and aids managers in the ability to update the work roster, see what employees are doing, and help them see the total sales of products.

Description of operation:

Summed up, the POS system should be designed with the ability of the user in mind to utilize the system in handling transactions including purchases and refunds through credit, debit, or cash. The system must operate on either apple or android operating devices, including phones or tablets, utilizing a camera for scanning barcodes. This section will go more in depth on the operations during payment processing, online inventory, security and storage, and user functions.

To begin, during a transaction, the system must have the ability to ring up items. Operations used to accomplish this must include the ability to enter a unique item number, scan an item's barcode, or manually enter an item ID, all of which is stored in the online inventory. The system must include an operation to automatically calculate the total price, including sales tax, and after a transaction, will automatically remove the item from the store's inventory. Credit and debit card operation will all be handled by an external payment processor but the two systems must operate in harmony with one another.

Inventory is one of the most important elements of the POS system and must operate efficiently and effectively. The inventory operates through an online system and is used as a tool for employees to see what is available in the store's stock. Functions and operations the inventory system includes are the ability to search for specific items and see whether or not the store has a particular item in stock, or if a neighboring store has the item. The search operation will require either an item ID, the item's name, or the sata it was added to the inventory. Employees will also operate the inventory by adding or removing items and assigning items with relative information such as the item's price, color, size, quantity, and ID number.

The system will also operate with the help of the cloud for memory and database storage and security. All transaction history and sales will be stored securely and separately from inventory. The system will include a special manager operation that will allow accessing the secure data and for creating new employee accounts.

3 Development plan and timeline

Partitioning of tasks:

In the first 2 month:

- Someone to supervise the project and make sure that the clothing store point of sale system is being created and is on time
- Someone to communicate with the client and update the team on what the client wants in the system

In the following months till end of project:

- Programmers that will write the code need for the system to look and function how the client wants it
- Someone to test if the system that is created is the same as what the client asked for

Team member responsibilities:

- Project Managers are team members responsible for overseeing the project, making sure that the team members are enough for the assigned tasks, and will be responsible for how the system will be created.
- Back-end developer are team member responsible for writing code to implement into the system that follows the UML class diagram

- Front-end developer are team members who are responsible for making the display of the website to what the clients want and so they are responsible for communicating with the clients and the back-end developers as well
- Quality tester will be responsible for following a series of test on the system and reporting the bugs to other team members

4 SDS: Test Plan

<u>This Section:</u>

In this section of the document, test cases will be laid out to make sure that units, functions, and the system all work in unison to make sure the system works as desired. In this section -> means the next method used in sequence. The notation of the method is methodName(parameter), desiredOutput

Test case to test Inventory:

<u>Unit test</u>- addInventory(shirt),

In order to verify that this method works correctly, we need to check the inventory before and after to make sure that the shirt was added properly. In pseudo code:

```
checkStock(shirt), returns 0 -> addInventory(shirt), added - > checkStock(shirt), returns 1
```

In this test case, the checkStock function returns 0 or false because there are no shirts in stock, then we add shift to the inventory, after we checkStock for the shirt again, it returns 1, confirming that the shirt was added to the inventory.

<u>Function test-</u> Testing that calculateTotal, Deletes from Inventory checkStock(23489134), returns 1, itemNumber(23489134) -> calculateTotal(23489134) → deleteInventory(23489134) → checkStock(23489134), returns 0

The above function test is an example of a user entering an item number to be purchased, calculating the total cost of the purchase, then updating the inventory database after each transaction.

In order to verify this intended behavior above works properly, we first need to see if each individual method works first. First, the function should check if the item is present within the inventory database through the checkStock(23489134) method. This method should output 1 if the item is present in the database, or 0 if the item is not present in the database. To progress to the next part of this function, the method should output 1 to prove the item is available. After this, when the user enters an item number for the item to be purchased through the method, itemNumber(23489134), it should first print out the information correlated with the product such as the description and the price. Afterwards, it will calculate the total of the transaction through this method, calculateTotal(23489134), where it'll take in the cost of the product based off the item's product number, and store it into aint variable called total, where it'll store the total cost of the total transaction. After this method executes, the method deleteInventory(23489134) should be executed where it's a void type method where it'll delete the product in the inventory database. Lastly, the method checkStock(23489134) should be executed where it'll return 0 to prove the item is not within the inventory database anymore.

System test- We will test the systems use case of adding an item to the inventory,

ClockIn(012345) → employeeID(012345), → searchInventory(string shirt) → checkStock(shirt), return 0(out of stock) → addInventory(shirt) → checkStock(string shirt), returns 1(in stock)

In this test case, we are testing the systems operation and use case of an employee adding to the inventory.

In order to do this an employee must log in using their employee ID number. They will then access the system through that same ID number. In this case they search the inventory for a shirt, and checkStock(shirt) returns 0 because there are no shirts in stock. They then add a shirt to the inventory and then check stock again. The expected output for check stock is now 1 and returns true because the shirt was successfully added. This then tests the system for adding to inventory.

Test case to test Manager:

Unit test- test addEmployee(String)

getEmployee(Hsa), return 0 →addEmployee(Hsa) →
getEmployeeInfo(Hsa), return 1;

This unit test is to test adding an employee into the system.

The test uses the function getEmployee(Hsa) first to check if there is already an employee named Hsa in the system, the return value is 0 so there is not an employee Hsa. After return value is false (0) then the function addEmployee(Hsa) is called which would add that employee into the system and the getEmployeeInfo(Hsa) gets called to do a final check that employee hsa is in the system and return 1 if true.

<u>Function test</u>- test scanBarcode(int)

scanBarcode(234237) → calculateTotal(234237) → 3rd payment program → deleteInventory(234237) → checkStock(234237) → return 0 → getSalesInfo(234237) (for the manager to check)

This test scanBarcode(int) function.

The test uses the scanBarcode(234237) with an int value equal to 234237 where the int value is the corresponding numerical value to a product's barcode. Afterwards the calculateTotal(234237) is called next where it'll take in the cost of the product based on the item's product barcode, and store it into aint variable called total, where it'll store the total cost of the total transaction. Next the function would use an external 3rd party payment program to process the payment from the user, whether it be cash or card and after processing the payment, the item would be deleted from the inventory using the deleteInventory(234237) by removing the product from the inventory database. Then checkStock(234237) would check to see if that item is still in stock and should return 0 since the product has been removed from the inventory database. If a manager wants to check the sales of an item they would use getSalesInfo(234237), where the parameter is the item's product number, and this method should return the statistical values of the product, such as sales rate and product's popularity.

System test- Testing a manager Signing in to add and delete an employee.

clockIn(54321) → employeeID(54321) → getEmployeeInfo(012345), not found message → addEmployee(012345, Bob) → getEmployeeInfo(012345), 012345 Bob → deleteEmployee(012345) → getEmployeeInfo(012345)

This tests the use cases for the manager adding and deleting employees information.

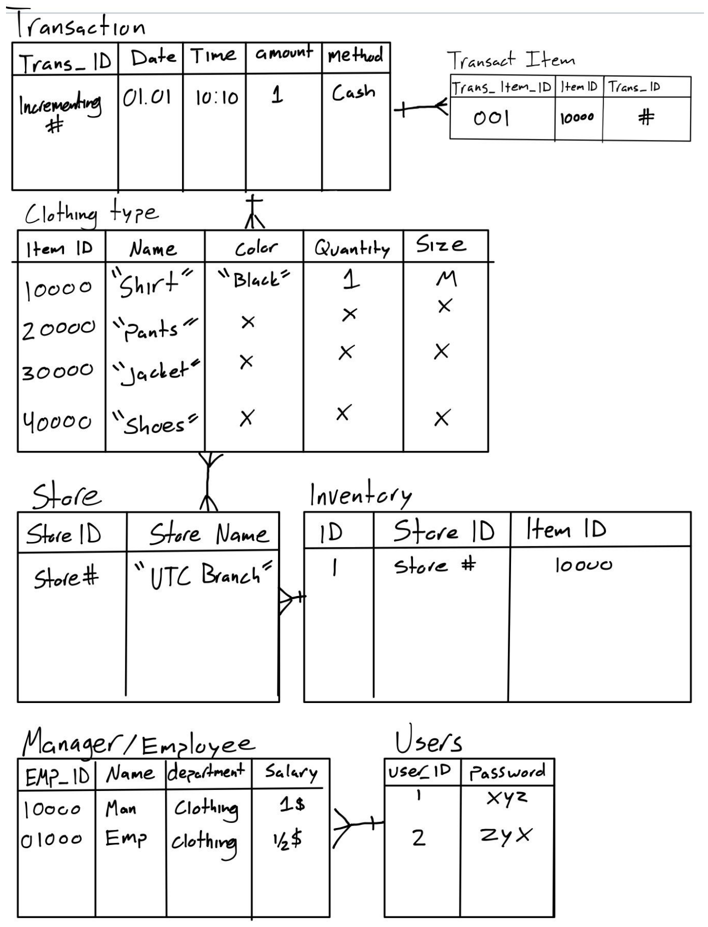
In this test case the manager clocks in with their employee ID and access the system through that same ID. They then check for an employee with the employee ID of 012345, this returns with a no employee found message because there is no employee in the system with that ID. They then add the employee with an employee number and corresponding name. This is followed by getting the employee info with that employee ID of 012345 and the expected output is the name and ID that was just added. The manager now deletes that employee's information and checks to get employee Info again, this then returns with the same error message that no employee was found because that employee's information was deleted.

5 Data Management Strategy

<u>This Section:</u>

This section will specify the SQL data management strategy for the POS System. Diagrams will be provided as well as descriptions of the diagrams for ease of use. Discussion of design decisions, databases, how the data is logically split, and possible alternatives that could have been made.

<u>SQL Diagram:</u>



Transaction

Trans_ ID	Date	Time	amount	method
Incrementing #	01.01	10:10	1	Cash

Transact Item

Trans_ Item_ID	Item ID	Trans_ ID
001	10000	#

Clothing type

Item ID	Name	Color	Quantity	Size
10000	"Shirt"	"Black"	1	M
20000	"Pants"	x	x	x
30000	"Jacket"	x	x	x
40000	"Shoes"	x	x	x

Store

Store ID	Store Name
Store#	"UTC Branch"

Inventory

ID	Store ID	Item ID
1	Store #	10000

Manager/Employee

EMP_ID	Name	department	Salary
10000	Man	Clothing	1$
01000	Emp	clothing	½$

Users

user_ID	Password
1	xyz
2	zyx

<u>Description of SQL diagram:</u>

In the SQL diagram one of the main databases is Inventory. The inventory table keeps track of the inventory for all stores and includes a unique ID number, Store ID and Item ID. We decided to have the inventory only have three elements because this is most likely the biggest database and it will store a bulk of the information. Another reason we decided on only having the three elements is because when searching for an item in the inventory, the system will only look at the three keys to let an employee know whether or not the item is in stock or not at any particular location. In a trade off for having less fields in the database, it created a couple additional databases that connect to the inventory.

One connection is the store database. The relationship between store and inventory is many to one. Many stores use one shared inventory, that way employees are able to look at stocks for other stores. The store database uses a storeID and a store Name. Many stores also have many types of clothing. Inside the clothing database, itemID, Name, Color, quantity and size, are used to uniquely identify items. This allows for stores to have multiple clothing types in its inventory, and clothing types can be present in multiple stores inventory.

The transaction table has a one to many relationship with clothing type. For every one transaction, you can purchase multiple types of clothing. Also one store can have many transactions. This is connected through the Item ID and the TransID keys in the Transaction Item table. This connects individual clothing items, to stores, and to each individual purchase. We decided to include the transaction database as its own individual table because if you need to go through transaction histories, you have its own individual database that is only full of the information you need and not all of the keys if we had only one database for all of the information.

Finally we have a disconnected database of users, manager, and employee. We disconnect this because the data storage does not have to exist within any sequence of the data storage for the Inventory or transactions. We have a user table that stores the userID and password. This has a one to many relationship with manager/employee because we have many employees and managers who all branch off one user database. Inside the employee table we have the keys, empID, name, department and salary. By disconnecting this database, when searching for an employee, you only need to search through the users database rather than the whole connection between transactions and inventory making it fast to look up employees.