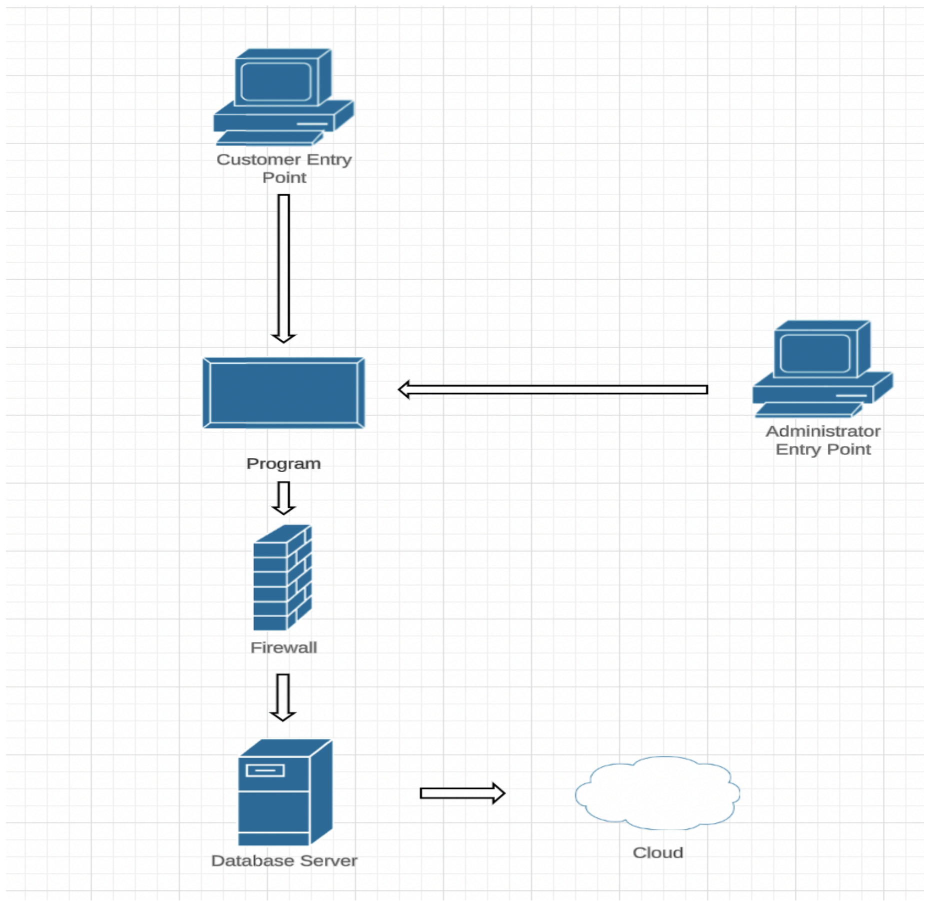
SDS: Test Plan for Clothing Store POS System

By: Shreyas Basu and Salvador De La Torre

1 Introduction

This document outlines the design and requirements for a storewide kiosk system that will initially be launched in one store and eventually spread store-wide. The purpose of this system is to make it easier for customers, employees, and the owners to shop/work in the store. It's needed because the current system is inefficient, not helpful/intuitive, and an upgrade is long overdue. This system-wide change will affect employees in that it will make their job easier. It will also affect customers by making the shopping experience next-level, more fun, and unique. Overall, it will be in the best interest for the company to get this system. This document is outlined into four sections, the introduction, user requirements, system requirements, and other information.

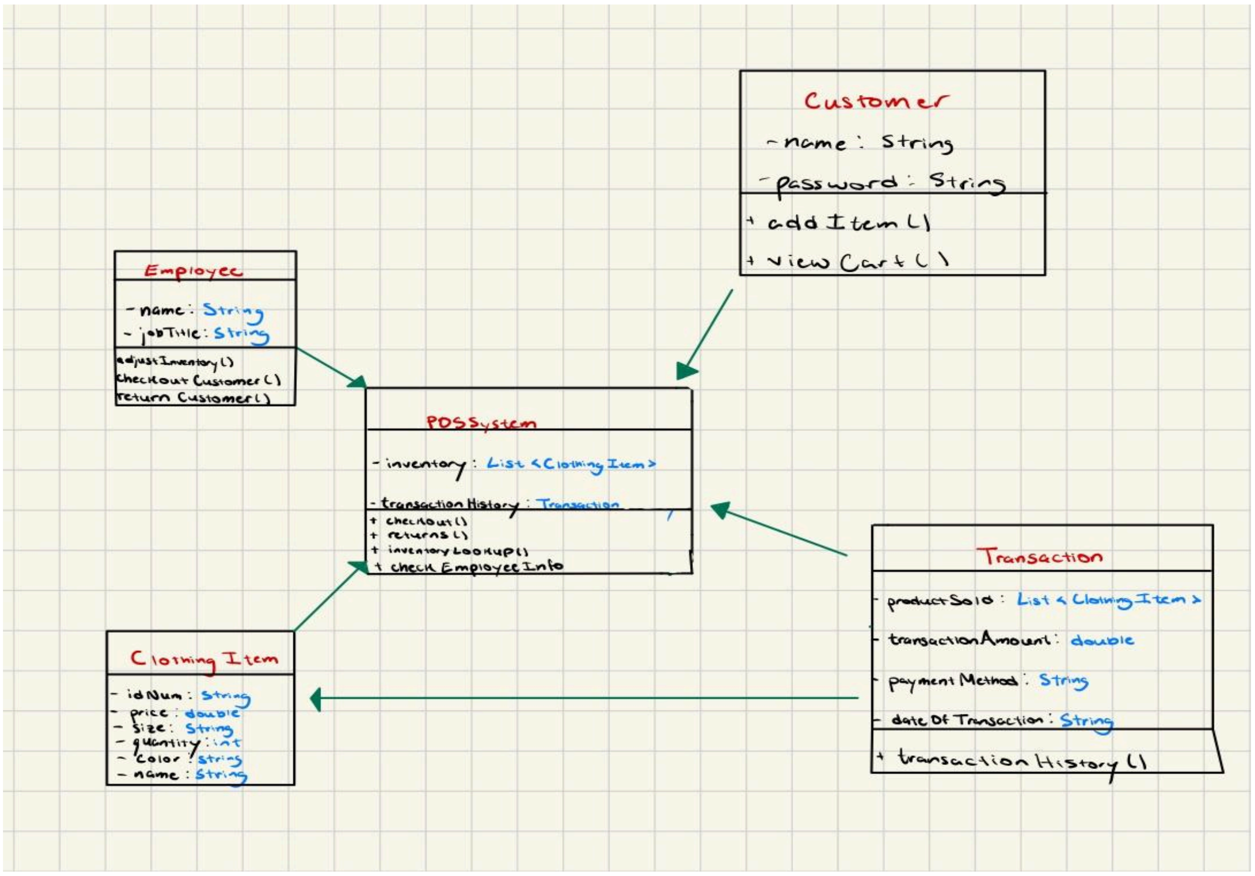
2 Architectural Diagram:



Description

This Software Architectural Diagram is precise and concise. There are two entry points, the user and administrator entries. Once correct credentials are entered, it goes to the program where they do what they need to do. For customers, it would be handing the items to the employee for them to be scanned and checked out. For administrators, it would be actions such as price matching, returns, account setup, seeing purchase history, etc. These programs store data in a database server which is then backed up by the cloud.

3 UML Diagram:



Description

Changes: The changes that we made from assignment 2 was getting rid of the TransactionHistory class, as we found it a little bit redundant and easier to just put the transactionHistory() method in our Transaction class. We also added a checkEmployeeInfo() method into the POSSystem main class, which allows us to check all the info of the Employees that are entered into the system.

We also added the **Customer** class, which holds the customers name, their password to their account, and methods addItem() which allows them to add items to their cart, and viewCart() which allows the customer to view their cart and what items are in there before they checkout.

The UML Class diagram for this system is quite simple and does not contain too many components.

The main class is the **POSSystem** class, which is meant to represent the main system and is the center of our diagram. This class contains two variables, inventory and transactionHistory. inventory is a variable that will hold in a list, all the available inventory at the store and the transactionHistory variable is an object of the TransactionHistory class which is meant to hold all the transaction history. The checkout() function is used to process any transactions where a customer is making a purchase. The return function is used to process any transactions where a customer is making a return. The inventoryLookup() function is meant to be able to lookup what that store or other stores using the same system have available in their inventory. Lastly, the checkEmployeeInfo() function is meant to look up all the employees in the system and show their names as well as their job titles.

Next would be the **Employee** class, which has variable name and jobTitle. Both variables in this class are string variables, and the name variable will hold the name of the employee and jobTitle will hold the jobTitle that that specific employee has. In this class are the adjustInventory(), checkoutCustomer(), and returnCustomer() functions. The adjustInventory() function will allow the employee to adjust the store's inventory after either a purchase or a return. The checkoutCustomer() function will allow the employee to ring up customers when they are ready to checkout, and the returnCustomer() function will allow the employee to process returns that a customer may have.

The **Transaction** class has the productSold variable which has a list of all clothingItems that have been sold. The transactionAmount variable is a double variable that represents the total cost of a particular transaction. The paymentMethod variable is a string variable that is meant to represent the type of payment that a customer is using for their transaction. Lastly the dateOfTransaction variable is a string variable that will hold the date that a particular transaction was done. There is the transactionHistory() function that is meant to be called in order to check the list of the transactions that have occurred through the system.

The last class being the **ClothingItem** class is a class that holds all the descriptive properties of the clothing items. The idNum variable is a string that holds the unique ID number of a certain product. The size variable is a string variable that represents what size that particular product is. The price variable is a double variable that represents the cost of that particular product. The quantity variable is an int variable that holds the total number of that particular product in the store's inventory. The color variable is a string variable that holds the color of that particular product. And lastly, the name variable is a string variable that holds the name of that particular product.

4 Test Cases:

<u>Unit Test</u>

Test Set 1: adjustInventory()

The targeted features for this test set is testing how the inventory changes when certain inputs are inserted into the adjustInventory() method

Input:
idNum: 1234
size: small
color: black
quantity: 10
Expected Output:
The quantity of product number 1234 in a black size small will increase by 10 units in the inventory

Input:
idNum: AB45
size: medium
color: black

quantity: 0
Expected Output:
The quantity of product number AB45in a black size medium will remain the same in the inventory

Input:
idNum: MU01
size: small
color: orange
quantity: -4
Expected Output:
The quantity of product number MU01 in a orange size small will decrease by 4 units (only if quantity is at least 4 prior to adjustment)

Input:
idNum: MU01
size: small
color: orange
quantity: -4
Expected Output:
Error will occur (assuming product MU01 has a quantity of 3 or less prior to adjustment)

Input:
idNum:
size: small
color: black
quantity: 10
Expected Output:
An error will occur since the idNum section was left blank

Input:
idNum: PL45
size: small
color: black
quantity:
Expected Output:
An error will occur since the quantity section was left blank

Input:

idNum: PL45

Price: $49.99

Expected Output:

The price for all product PL45 will change to $49.99 from its previous marked price

Input:

idNum: PL45

name: “hoodie”

Expected Output:

Product PL45 will be named “hoodie” in the inventory

Test Set 2: transactionHistory()

The targeted features for this test set is testing the output of transactionHistory in different scenarios.

Input:

Nothing

Expected Output:

Null

Input:

product Sold: [“hoodie”]

transactionAmount: 35.68

paymentMethod: “cash”

dateOfTransaction: “3/22/2024”

Expected Output:

[[“hoodie”], 35.68, “cash”, “3/22/2024”]

Input:

product Sold: [“pants”]

transactionAmount: 25.68

paymentMethod: “credit”

dateOfTransaction: “3/21/2024”

Expected Output:
[[“pants”], 25.68, “credit”, “3/21/2024”]

Input:
product Sold: [“pants”, “hoodie”]
transactionAmount: 61.36
paymentMethod: “debit”
dateOfTransaction: “3/25/2024”
Expected Output:
[[“pants”, “hoodie”], 61.36, “debit”, “3/25/2024”]

Input:
product Sold: [“pants”, “hoodie”, “belt”]
transactionAmount: 71.36
paymentMethod: “apple pay”
dateOfTransaction: “3/19/2024”
Expected Output:
[[“pants”, “hoodie”, “belt”], 71.36, “apple pay”, “3/19/2024”]

Input:
product Sold: [“”]
transactionAmount: 12.12
paymentMethod: “samsung pay”
dateOfTransaction: “3/21/2024”
Expected Output:
Error: no field can be null

Integration Test

Test Set 1: Adding Employees into the System

The feature that is being tested in this, is seeing how the checkEmployeeInfo() class is adjusted when a new Employee object is created. The employee object must have a name and jobTitle to be a valid object, and we are testing if the Employee object is present and visible when the target method is called.

Admin attempts to add employee into the system
Input:
Employee Class

name: “John Smith”
jobTitle: “Team Member”
checkEmployeeInfo()
Expected Output:
John Smith is successfully entered into the system as a Team Member
Verify using checkEmployeeInfo() that John Smith is found in the list of employees as a Team Member

Input:
<u>Employee Class</u>
name: “John Smith”
jobTitle: “Cashier”
checkEmployeeInfo()
Expected Output:
An error occurs as there is already a John Smith in the system.
Verify using checkEmployeeInfo() that John Smith is found in the list of employees as a Team Member (previous example) and there is not a duplicate John Smith in the system as a cashier

Input:
<u>Employee Class</u>
name: “”
jobTitle: “Team Member”
checkEmployeeInfo()
Expected Output:
Error occurs as the name of an employee cannot be blank
Input:
name: “Alex Jones”
jobTitle: “”
Expected Output:
Error occurs as Alex Jones must hold a job title and it cannot be left blank

Input:
<u>Employee Class</u>
name: “Alex Jones”
jobTitle: “Cook”
checkEmployeeInfo()
Expected Output:
An error occurs as a Cook is not a possible job title at a clothing store

Input:
Employee Class
name: “Bob Johnson”
jobTitle: “Admin”
checkEmployeeInfo()
Expected Output:
Bob Johnson is successfully entered into the system as an Admin
Verify using checkEmployeeInfo() that Bob Johnson is found in the list of employees as an Admin and also verify that the employee has the abilities to do all administrative actions in the system.

Test Set 2: Adding Clothing Items into System
The feature that is being tested in this, is seeing how the POSSystem.InventoryLookup() class is adjusted with different ClothingItem objects.

Admin/employee adds to clothes into the system and is checked by POSSystem
Input:
Nothing
Expected Output:
Null

Input:
ClothingItem
idNum: “TR343FE3”
price: 25.68
size: “Medium”
quantity: 1
color: “black”
Name: “pants”
POSSystem.InventoryLookup()
Expected Output:
Item 1: [“TR343FE3”, 25.68, “Medium”, 1, “black”, “pants”]

Input:
ClothingItem
idNum: “FL219EF1”

price: 35.68
size: “Large”
quantity: 2
color: “blue”
Name: “hoodie”
POSSystem.InventoryLookup()
Expected Output:
Item 1: [“FL219EF1”, 35.68, “Large”, 2, “blue”, “hoodie”]
Input:
ClothingItem
idNum: “FL219EF1”
price: 35.68
size: “Large”
quantity: 2
color: “blue”
Name: “hoodie”
ClothingItem
idNum: “TR343FE3”
price: 25.68
size: “Medium”
quantity: 1
color: “black”
Name: “pants”
POSSystem.InventoryLookup()
Expected Output:
Item 1: [“FL219EF1”, 35.68, “Large”, 2, “blue”, “hoodie”]
Item 2: [“TR343FE3”, 25.68, “Medium”, 1, “black”, “pants”]

System Test

Test Set 1: Testing returning items
The target for this test set is seeing how the entire system reacts and operates when a customer is attempting to return an item back to the store. This set takes the possible inputs that could occur when a customer is returning an item, and is ensuring that the entire system functions properly based on the input.

Input: Customer wants to return an item back to the store
Expected Output:
Employee is able to log into the system and have the ability to process a return
The transaction amount will be a negative amount, as money is going back to the customer
Returns are only given in cash, so only available method of payment will be cash
Returns are still considered transactions, so the date of the return transaction will be accurate
The transaction will be successfully put into the transaction history list
System automatically puts the product back into the inventory
Products quantity will increase by 1 in the inventory based on its idNum, color, and size

Input: Customer wants to return an item back to the store and receive payment back on their credit card
Expected Output:
Returns are only given in cash, so only available method of payment will be cash therefore the system will not be able to accommodate the customer

Input: Customer wants to return an item back to the store that is not a product bought from that store
Expected Output:
An error will occur when attempting to return it, as that product is not in the stores inventory

Test Set 2: Checkout
The target for this test set is seeing how the entire system reacts and operates when a customer goes through the whole buying process illustrated in the system.

Input: Customer enters the name and password. They then add items that are uploaded to the system. An employee checks them out.
Expected Output:
Money added to company account (or bank register for cash).
Inventory is updated accordingly.
Success.

Input: Customer enters the name and incorrect password
Expected Output:
Error: incorrect password

Input: Customer enters the name and password. They then add items that are uploaded to the system. They then return all the items. They try to checkout

Expected Output:

Error: Can’t check out nothing