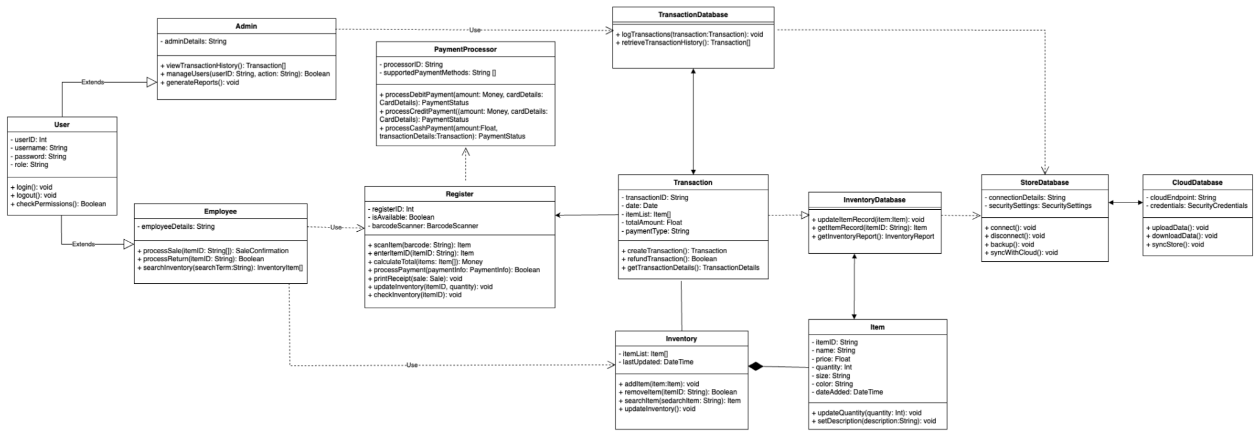
Test Plan

Updated Software Design Specifications



Admin
- adminDetails: String
+ viewTransactionHistory(): Transaction[]
+ manageUsers(userID: String, action: String): Boolean
+ generateReports(): void
Use
Extends
User
- userID: Int
- username: String
- password: String
- role: String
+ login(): void
+ logout(): void
+ checkPermissions(): Boolean
Extends
Employee
- employeeDetails: String
+ processSale(itemID: String[]): SaleConfirmation
+ processReturn(itemID: String): Boolean
+ searchInventory(searchTerm:String): InventoryItem[]
Use
PaymentProcessor
- processorID: String
- supportedPaymentMethods: String []
+ processDebitPayment(amount: Money, cardDetails: CardDetails): PaymentStatus
+ processCreditPayment((amount: Money, cardDetails: CardDetails): PaymentStatus
+ processCashPayment(amount:Float, transactionDetails:Transaction): PaymentStatus
Register
- registerID: Int
- isAvailable: Boolean
- barcodeScanner: BarcodeScanner
+ scanItem(barcode: String): Item
+ enterItemID(itemID: String): Item
+ calculateTotal(items: Item[]): Money
+ processPayment(paymentInfo: PaymentInfo): Boolean
+ printReceipt(sale: Sale): void
+ updateInventory(itemID, quantity): void
+ checkInventory(itemID): void
Use
TransactionDatabase
+ logTransactions(transaction:Transaction): void
+ retrieveTransactionHistory(): Transaction[]
Transaction
- transactionID: String
- date: Date
- itemList: Item[]
- totalAmount: Float
- paymentType: String
+ createTransaction(): Transaction
+ refundTransaction(): Boolean
+ getTransactionDetails(): TransactionDetails
InventoryDatabase
+ updateItemRecord(item:Item): void
+ getItemRecord(itemID: String): Item
+ getInventoryReport(): InventoryReport
StoreDatabase
- connectionDetails: String
- securitySettings: SecuritySettings
+ connect(): void
+ disconnect(): void
+ backup(): void
+ syncWithCloud(): void
CloudDatabase
- cloudEndpoint: String
- credentials: SecurityCredentials
+ uploadData(): void
+ downloadData(): void
+ syncStore(): void
Inventory
- itemList: Item[]
- lastUpdated: DateTime
+ addItem(item:Item): void
+ removeItem(itemID: String): Boolean
+ searchItem(sedarchItem: String): Item
+ updateInventory(): void
Item
- itemID: String
- name: String
- price: Float
- quantity: Int
- size: String
- color: String
- dateAdded: DateTime
+ updateQuantity(quantity: Int): void
+ setDescription(description:String): void

Changes Made:

- In the user class, the userID was changed from a String to an Int. This change was made since the username would take in a string while the userID would only include a numerical ID.

Overall, there were not any major changes to our previous design. We decided to not make any more modifications because we felt that our Software Architecture Overview sufficiently addressed all necessary components without inaccuracies or missing information.

Test Set 1 : User Account Management

Unit Testing

Test Case: checkPermissions()
Validate that the checkPermissions() method accurately identifies an admin user's permissions

Input: Instantiate a User object for admin Greg with userID, username, password, and role as Admin. Then, instantiate a User object for employee Sue with userID, username, password, and role as Employee.

```
User checkPermissions()
User a
a.userID = "123" // changed userID type to Int on UML
a.username = "greg"
a.password = "test"
a.role = "Admin"

User b
```

```
b.userID = “456” // changed userID type to Int on UML
b.username = “sue”
b.password = “test”
b.role = “Employee”

IF(a.checkPermissions() == true)
    Return PASS
ELSE
    Return FAIL
```

Expected output: The method returns a pass for Greg, confirming he has admin permissions, and a fail for Sue since she has employee permissions.

How the test covers the targeted feature:
The checkPermissions() test case checks if the system correctly identifies user roles, such as distinguishing between admin Greg and employee Sue. By creating two users with different roles and testing if they have the correct permissions, we’re making sure that the system works as it should for role-based access. When Greg, the admin, is tested we expect the system to confirm that he has admin rights, which means he can access the transaction database. For Sue, the employee, we expect the system to show she doesn’t have permission. This allows us to test both scenarios and confirms that our system correctly gives access based on a user’s role. Which is key to making sure the right people have access to certain actions and information.

Functional Testing (Integration)

Test Case: viewTransactionHistory(): Transaction[] and retrieveTransactionHistory(): Transaction[] integration
Ensure that the viewTransactionHistory() method correctly retrieves the transaction history through the retrieveTransactionHistory() method and returns a correctly formatted list of transactions from the database.

Input:
Instantiate an Admin object for admin Greg with necessary details to log in and view transaction history.

```
SET admin = NEW Admin
admin.adminDetails = "Details about admin Greg"
SET transactionHistory = admin.viewTransactionHistory()

IF (transactionHistory != NULL)
     IF (transactionHistory is an array of Transaction)
        FOR each transaction in transactionHistory
            PRINT transaction
            EXIT
        PRINT "PASS: Transaction history successfully retrieved."
      ELSE
        PRINT "FAIL: Incorrect data type for transaction history."
ELSE
      PRINT "FAIL: Transaction history is null."
```

Expected output:
The expected output is non-null, and is correctly formatted as an array of Transaction objects. To confirm this, each transaction in the transactionHistory is printed, then the system will print "PASS: Transaction history successfully retrieved." If transactionHistory is null or not formatted as an array of Transaction objects, print “FAIL” with the appropriate failure reason.

How the test covers the targeted feature:
This test validates the integration between viewTransactionHistory() and retrieveTransactionHistory(), ensuring that the system can retrieve and display the transaction history accurately for an admin user like Greg. By checking that transactionHistory is non-null and correctly formatted we are also verifying that the data is presented in the expected structure. This is essential for admin users who rely on accessing and viewing transaction history for management and reporting purposes. The test confirms the systems ability to handle database interactions and present data correctly, which is fundamental for operational integrity.

System Testing

The system test should start with the admin attempting login into the Register using their ID, username, and password. Upon successful login, the system should verify the admin’s permissions before granting access to the transaction database. The admin should then be able to retrieve a list of transactions, select any transaction to view it’s details, and receive appropriate notifications in the event of retrieval issues, such as attempting to access transactions that don’t exist in the database. The test should cover logout procedures and ensure no unauthorized access is permitted once the session is terminated.

Test Set 2 - Inventory

Unit Testing

Test Case: searchItem()
Input: Input the itemID
Output: Item

```
Inventory i
i.addItem(“123”)
Inventory searchItem()
```

```
i.searchItem(“123”)
If (i.searchItem == i)
    Return Item
Else
    Return “Item is not found”
```

Features Tested:

1) Item Search - testing if employees are able to search for items in inventory
2) Accuracy - testing if the results are accurate and relevant to what is being searched

Test Sets:

1) Item that exists in inventory
2) Item that does not exist in inventory

How the Tests Cover the Targeted Features:
We first add an item to the inventory in order for it to be searched. After the item is added, we search the inventory using an itemID connected to the item. After the item is searched for, we either get the item or a message that states, “Item is not found”. This tests if the employees are actually able to search for items in inventory. Also, this tests the accuracy of the searches, because if we get a result that is not the item that we searched for, then we can conclude that the search feature is not accurate.

Functional Testing (Integration)

```
checkInventory (itemID): void
Inventory i
i.addItem(“123”)
Inventory searchItem()

i.searchItem(“123”)
checkInventory c
if (c.checkInventory(i) == searchItem)
    return item
else
    return “Item not found”
```

For the integration test, we are testing the feature for an employee or manager to be able to check on the stock of an item in the store on the kiosk as well as testing how accurate the system is at displaying the information. The selected test sets that will be applied will be to check whether the item is in stock or if the item does not exist/is out of stock. First, an item would be entered into the inventory information and the checkInventory function would exist within the program where an itemID can be entered into the system. The system will then compare the entered itemID with the existing itemID’s in the inventory and will return the item qualities such as size, color, stock, etc. If an itemID is entered that does not exist, the screen will display “item not found”. If the item does exist within the system but the stock count is 0, the screen would display “out of stock”.

System Testing

The employee or manager should be able to walk up to the kiosk and log in with their employee account. They are prompted to a screen where they can navigate to a screen where they can search for items in the inventory. They should be able to enter the itemID and click the search button where the screen will give them information about the stock of the item. If an item is not found, the system will display a message saying that the item is not found or that the item is not currently in stock. After the message is displayed, the employee is able to click a button that will return to the home screen.