POS Machine Software Design Specification

Team Members:

Carlos Gonzalez

Brendan Shiroma

Melanie Vaknin

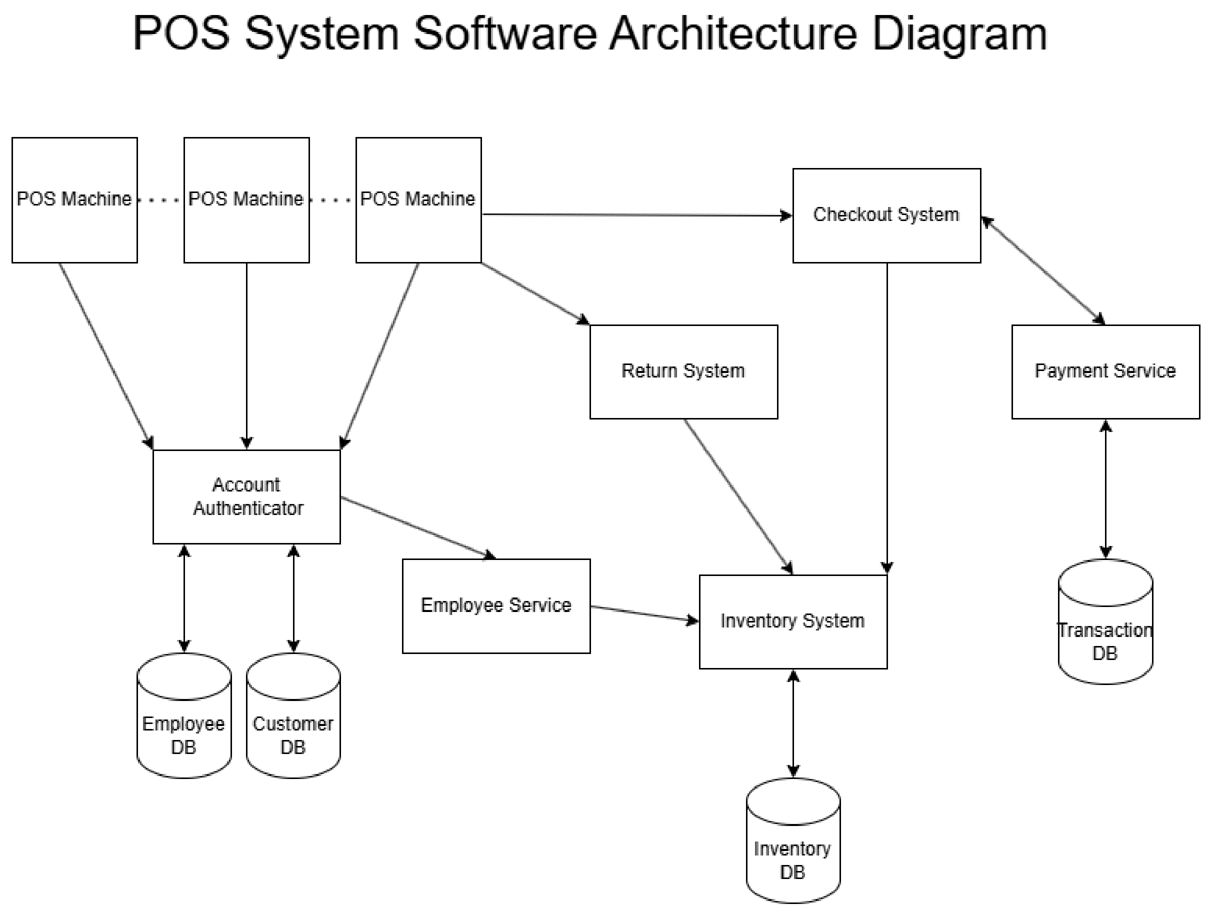
November 8, 2024

1. System Description

The POS system was intended to make the retail business easier by integrating main components: inventory control, sales, and employee management. Everything starts with the POS machines, which act as the prime touch point for customers and employees. These machines connect to subsystems like account authentication for employee access, checkout systems that handle customer purchases, and return systems that update inventory when items are returned. The payment service follows a policy of "secure transaction", offering the possibility to update stock levels immediately in real-time so that everything can balance out.

It makes up a big part of the system with the ability for in-store and online sales to align seamlessly because of data syncing. In addition, it provides comprehensive sales, inventory, and customer activity reporting to help managers make more informed decisions. User management functionality in the POS bolsters efficiency reduces errors, and smooths the running of the business by managing the employee's roles and permissions.

2. Software Architecture Diagram



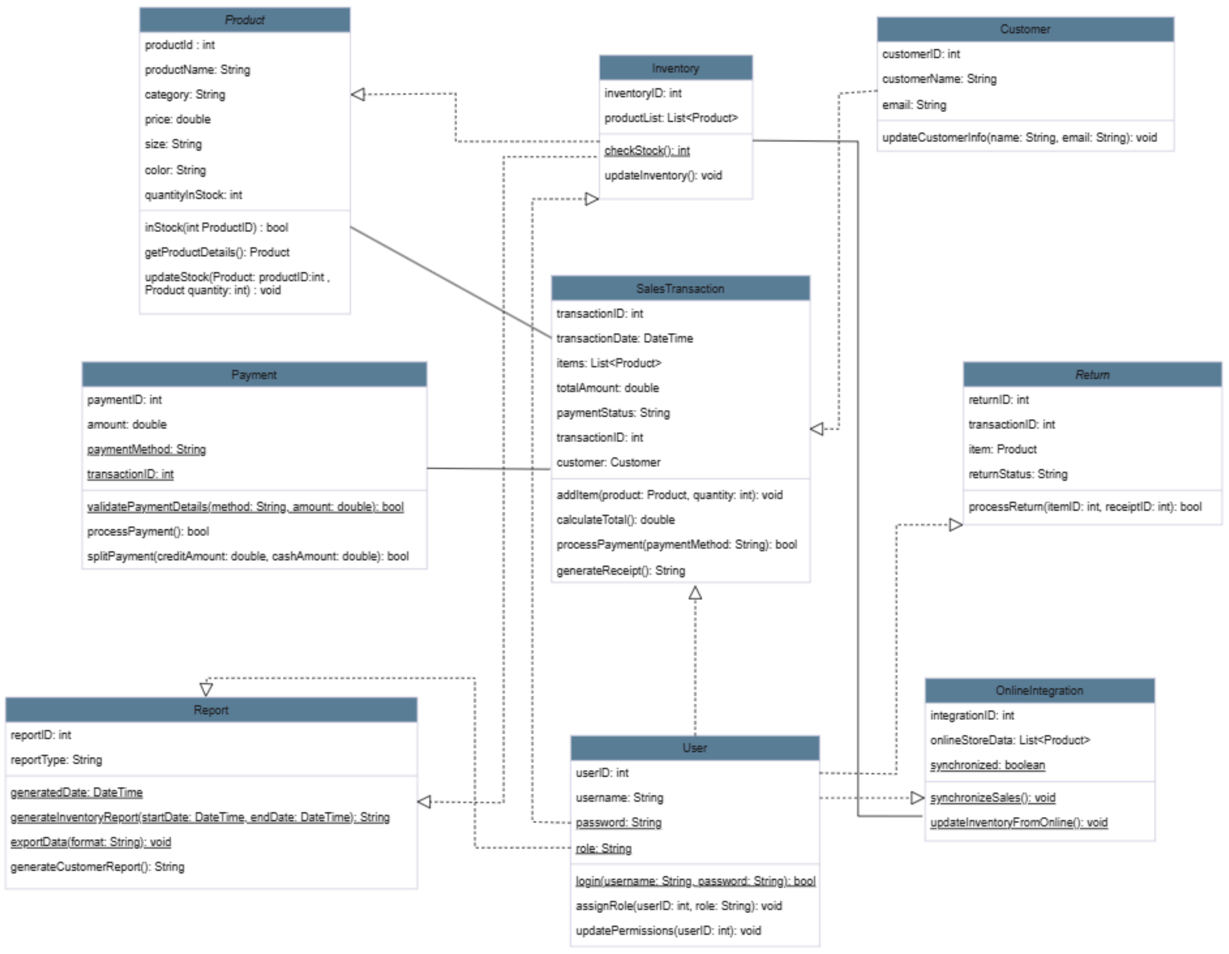
2.1 Diagram Description

The POS System Software Architecture Diagram starts with the POS machines. There are multiple of them to show how they can be used simultaneously with no consequences or drawbacks. Then the POS machine can either go to the account authenticator, the return system, or the checkout system. The account authenticator is used for employees to be able to log in to their accounts and access certain services. This then can go back and forth between the account database, which stores all the data of the employees such as their unique ID, their password, and any other information.

The account authenticator can also go to the employee service, which is where employees can do things such as process returns, update sales, process sales, and manage the inventory. From the employee service, they can modify and view inventory status in the inventory system, which also has a database connected to it where data can be sent back and forth from. Going back to the POS machine and the return system, the return system is a dependency of the POS machine. The inventory system then relies on the return system as well because when an item is returned, the inventory system needs to be updated for that item.

Finally going back to the POS machine again and the checkout system, the checkout system also depends on the POS machine. Then from the checkout system, it has a mutual relationship with the payment service, where data can be sent back and forth. The payment service also can send data back and forth from the transaction database. This is to ensure that the payment was successful so that it can go back to the checkout system and the customer can get their item, and the company can not get scammed. The checkout system also goes to the inventory system to update the inventory after a customer checks out with their items.

3. UML Class Diagram


Product
productId : int
productName: String
category: String
price: double
size: String
color: String
quantityInStock: int
inStock(int ProductID) : bool
getProductDetails(): Product
updateStock(Product: productID:int , Product quantity: int) : void
Inventory
inventoryID: int
productList: List<Product>
checkStock(): int
updateInventory(): void
Customer
customerID: int
customerName: String
email: String
updateCustomerInfo(name: String, email: String): void
SalesTransaction
transactionID: int
transactionDate: DateTime
items: List<Product>
totalAmount: double
paymentStatus: String
transactionID: int
customer: Customer
addItem(product: Product, quantity: int): void
calculateTotal(): double
processPayment(paymentMethod: String): bool
generateReceipt(): String
Payment
paymentID: int
amount: double
paymentMethod: String
transactionID: int
validatePaymentDetails(method: String, amount: double): bool
processPayment(): bool
splitPayment(creditAmount: double, cashAmount: double): bool
Return
returnID: int
transactionID: int
item: Product
returnStatus: String
processReturn(itemID: int, receiptID: int): bool
Report
reportID: int
reportType: String
generatedDate: DateTime
generateInventoryReport(startDate: DateTime, endDate: DateTime): String
exportData(format: String): void
generateCustomerReport(): String
User
userID: int
username: String
password: String
role: String
login(username: String, password: String): bool
assignRole(userID: int, role: String): void
updatePermissions(userID: int): void
OnlineIntegration
integrationID: int
onlineStoreData: List<Product>
synchronized: boolean
synchronizeSales(): void
updateInventoryFromOnline(): void

3.1 Product

- **Class Description:**
    - The product class holds all of the information that is necessary to know about a product.
- **Attribute Descriptions:**
    - productId: the unique ID of the product
    - productName: the name of the product

    - category: the category of the product
    - price: the price of the product
    - size: the dimension of the product, so the employees know where they can store the product
    - color: the color of the product
    - quantityInStock: the amount of product in stock that
- **Operation Descriptions:**
    - inStock(): returns true or false if the item is in stock
    - getProductDetails: returns the attributes of the product (i.e. productId, productName)
    - updateStock(int): updates the number of items in stock for a specific product

3.2 Inventory

- **Class Description:**
    - The inventory class holds the attributes of all the items in inventory, which includes all the items not on the sales floor and the items that are on the sales floor. This is also accessible for users who are shopping online.
- **Attribute Description:**
    - inventoryID: holds the unique ID of an inventory item
    - productList: a list of all the products
- **Operation Descriptions:**
    - checkStock(): returns the number of items in stock for a specific product
    - updateInventory(): updates the number of items in stock for a product

3.3 Customer

- **Class Description:**
    - This class holds the information about the customer, and allows the customer to update their information too.

- **Attribute Description:**
    - customerID: the unique ID of each customer
    - customerName: the name of the customer, including first and last name
    - email: the email of the customer
- **Operation Description:**
    - updateCustomerInfo(name: String, email: String): update the customer's name and email

3.4 Payment

- **Class Description**:
    - The payment class will implement the type of payment for an item in the store.
- **Attributes Description:**
    - paymentID: The unique ID for the payment.
    - Amount: The total amount of the payment
    - paymentMethod: The different types of payments a customer can make (examples: Credit Card, Debit Card, Cash)
    - transactionID: This one references the Sales Transaction ID.
- **Operations:**
    - validatePaymentDetails(method: string, amount: double): Function that validates the information to process the payment
    - processPayment(): Function that returns if the validation was successful or not.

3.5 Sales Transaction

- **Class Description**:
    - This class will implement the process for the transaction of a sale, validating the information from the product, connection with processing the payment, and generating the receipt.
- **Attributes Description:**

    - transactionID: The unique ID for a single transaction.
    - transactionDate: It records the date the transaction was made
    - items: It records a list of the items on the transaction
    - totalAmount: The total of the sale.
    - paymentStatus: Indicates the status of the payment.
    - customer: the information of the customer
- **Operations:**
    - addItem(Product, quantity): Function that adds each item to a list.
    - calculateTotal(): Function that calculates the total of the sale
    - processPayment(method): Function that processes the payment based on the method chosen.
    - generateReceipt(): After the transaction is successful, this function will generate a receipt, the customer can choose between paper, email, or phone number.

3.6 Return

- **Class Description**:
    - This class will implement the process for returning an item to the store.
- **Attributes Description:**
    - returnID: The unique ID for a return transaction.
    - transactionID: The unique ID for a transaction.
    - item: it records which item is to be returned
    - totalAmount: The total of the sale.
    - returnStatus: Indicates the status of the payment.
    - customer: the information of the customer
- **Operations:**
    - processReturn(): Function that validates the ID of the item to be returned with the ID of the receipt to process the return.

3.7 Report Class

- **Description:**
    - Report class reports about which items are currently in the inventory of the store.
- **Attributes:**
    - reportID: holds the ID for the report
    - reportType: holds the type of file for the report
- **Operations:**
    - generatedDate(): Function that generates the date of the report
    - generateInventoryReport() : Function to generate the inventory report
    - exportData(): This function exports the data.
    - generateCustomerReport():Function that generates customer’s report

3.8 User Class

- **Description:**
    - User class is a class meant to hold all the information about the different employees working at the store.
- **Attributes:**
    - userID: unique ID for the users
    - username: holds the username of the user
    - password: holds the password of the user
    - role: type of role at the store(cashier, manager, etc)
- **Operations:**
    - login(username: string, password: string): Function to log in the system using the username and password.
    - assignRole(userID: int, role: string): Function that assigns the role to the user.
    - updatePermissions(userID: int): Function that can update the permissions for the users.

3.9 Online Integration

- **Description:**
  - Online integration class is meant to show all the products that are displayed online, and keep up with all inventory reports online.
- **Attributes:**
  - integrationID: unique ID for the integration
  - onlineStoreData: holds the list of the products available online.
  - synchronized: checks if the information from the store and online are synchronized
- **Operations:**
  - synchronizeSales(): Function that combines the sales from store and online to give a single amount of sales.
  - updateInventoryFromOnline(): Function that checks the inventory online, and verifies we have the items in stock.

4 Other Information

4.1 Partitioning of Tasks

- Weeks 1-4: Brendan
  - Implementing the account authenticator and link it to the account database to store all the information
    - Also work on the development of the employee system so that employees have role-based access to certain services
- Weeks 5-10: Benjamin
  - Implement the return and checkout systems and focus mainly on accuracy over efficiency
    - These are the main systems that the customer uses, so the team can take more time on them
- Weeks 11-14: Melanie

    - Work on the implementation of the payment service and link it to the transaction database and back to the checkout system
        - Need to ensure some kind of encryption on the data in the transaction database
- Weeks 15-18: Brendan
    - Work on the inventory system and the inventory database
        - Should be able to link to other stores and the main distribution center

5 Verification Test Plan

5.1 Test Set 1:

Unit Test: Stock Management with the Product Class.

```
Product productID = 123;
Product quantityInStock=0;
inStock(123);
if(inStock(123)){
        Return False
Else
        Return True
```

- **Unit Test:** Updating the Stock of a Product and verifying that the function instock() returns the correct boolean value.
    - Our scenario is checking if inStock(Product productID) properly verifies the stock availability
    - Steps:
        - Create a product with quantityInStock variable = 0
        - After the product is created, call the inStock(Product productID) method.
    - The expected result is going to be False, given the fact that no stock is available.

Integration Test: User updates stock of a product

```
User login(username, password)): void
User assignRole(string role): void
Product updateInventory(movie) :void
User employee;
Product ID = 123 ;
Product quantityInStock = 10
If( employee.updateInventory( p.ID, p.quantity)
        return PASS
Else
        Return FAIL
```

- **IntegrationTest:** An employee logs in and checks if inStock() properly verifies the stock availability for a product.
    - Steps:
        - An employee logs in with the login() method and then assignRole() method.
        - After log in, the employee calls the method updateInventory(Product ID, int quantityInStock)
    - We call the method updateInventory(Product productID, Product quantityInStock) for example: updateInventory(123, 10)
    - Call the inStock(int productID) function, and the expected result is going to be True because now the stock has been updated.

System Test: Customers can see the updated inventory of a product.

- A user(employee) approaches the store system and looks to update the inventory of the product with ID = 123. They choose to change the quantityInStock from 0 to 10. If the product updates the stock successfully, the employee will see a confirmation screen, and the stock of that specific product will be updated also for the customers.

5.2 Test set 2:

Unit Test

```
Unit test: calculateTotal()

customerTotal = totalAmount;

testTotal = calculateTotal()

If customerTotal == testTotal

        Return PASS

Return FAIL
```

- **Description Unit Test:** This unit test is checking to make sure that the function calculateTotal is calculating in an accurate way the total cost of all sales transactions. The test compares the expected customerTotal and the testTotal, if they are equal the test passes, if not, the test fails.

Integration Test

```
addItem(“shirt”, 17)

Product product;

updateStock(“shirt”, -17)

If (product.getInventory() == (quantityInStock - 17))

        Return PASS

Return FAIL
```

- **Integration Test Description:** This unit test is designed to add an item to the stock, and then to check the stock quantity of apples before, then it updates the stock to include the added apples, and checks the stock quantity apple, and then checks to see if it was updated properly.

5.3 System Test

An employee logs into the system using their unique username and password. They are then able to assist a customer with their purchase by scanning their items and adding it to the checkout. After they are done scanning all the items, the system calculates the total, and prompts the customer to enter in their desired payment method. The payment method gets validated, the payment gets processed, and the receipt gets printed with all the transaction details and products that were purchased. After the transaction has gone through, the stock gets updated in real time for each of the items that were purchased so employees and customers can see what is available, and how much of it is available.

5.4 UML Diagram Modifications

- **Updates:** Changed inStock() method to have a parameter of productID. This allows the user to check if a specific item is in stock, since the method before was not item-specific. Also changed updateStock to have a second parameter of productID so that the updateStock method updates the specific product.

7 Software Architecture Diagram.

Diagram Explanation

The updated Software Architecture Diagram for the POS System represents all the major components and their interactions with the database. This includes:

- **POS Machines**: Interface from which the processing of transactions, returns, and accesses by employees are made.
- **Checkout System**: This directly interacts with the Payment Service and Inventory Database to process purchases and update the stock in real time.
- **Return System**: Integrates into the Inventory Database in order to adjust stock in real time.
- **Employee Service**: Maintains the access and permissions of employees according to their roles and in concert with the Employee Database on user credentials.

- **Account Authentication**: Interacts with the Account Database for validation at times of login and access levels.
- **Payment Service**: Synchronizes with the Transaction Database to securely validate payments and complete them successfully.
- **Databases**: Inventory, Account, Employee, and Transaction databases manage the needed storage and data exchange between subsystems.

7. Data Management Strategy

SQL Databases

1. **Inventory Database**
2. **Transaction Database**
3. **Employee Database**
4. **Customer Database**

Each of these SQL databases contains tables for structured data that need consistency, such as inventory stock levels, employee details, and transaction records.

Table Descriptions and Relationships

7.1 Inventory Database

- Tables: Products, InventoryLevels
- Relationships: Products have a one-to-many relationship with InventoryLevels.

Tables:

- **Products**

productId	productName	category	price	size	color	quantityInStock
0001	T-Shirt Supreme	Clothing	21.99	M	White	50

0010	Blue Jeans	Clothing	25.99	34	Blue	30

- **InventoryLevels**

inventoryID	productId	location	quantity
001	0001	WareHouse	75
002	0005	Store 1	50
003	0010	Store 2	30

7.2.**Transaction Database**

- Tables: Transactions, TransactionItems, Payments
- Relationships: Transactions has a one-to-many relationship with TransactionItems. Payments has a one-to-one relationship with Transactions.

Tables:

- **Transactions**

transactionId	transactionDate	customerId	totalAmount	paymentStatus
4001	11-04-2024	1001	74.56	Complete

- **TransactionItems**

itemId	transactionId	productId	quantity	price
1	4001	0010	2	15.99
2	4001	1004	1	39.99

- **Payments**

paymentId	transactionId	paymentMethod	amount
5000	4001	Credit Card	74.56

7.3 **Employee Database**

- **Tables**: Employees, Roles, EmployeeRoles
- **Relationships**:
    - Employees have a many-to-many relationship with Roles through the EmployeeRoles table.

Tables:

- **Employees**

employeeId	username	password	name	email
101	carlos_gonz	gonz123	Carlos Gonzalez	cgonz@gmail.com

- **Roles**

roleId	roleName
1	Cashier
2	Manager

- **EmployeeRoles**

employeeId	roleId
101	1
202	2

7.4 Customer Database

- **Tables:** Customers

Tables:

- **Customers**

customerId	**name**	**email**
1001	Michael Jackson	mj@gmail.com

7.5 Data Flow and Synchronization

- **Synchronization with Online Sales**: Use regular synchronization schedules or event-based triggers to update the inventory and transaction records between online and in-store databases.
- **Real-Time Updates**: Implement triggers on the transaction and inventory databases to instantly update stock levels after each sale or return, ensuring data consistency across all platforms.

7.6 Data Security and Privacy

- **Encryption**: Encrypt sensitive data (passwords, payment details) within the transaction and account databases.
- **Access Control**: Implement role-based access to restrict database access based on employee roles. Only authorized personnel can access customer details or financial records.

7.7 Trade-Offs and Justification

- **SQL vs. NoSQL**: SQL was chosen for data integrity and robust querying capabilities. A NoSQL option could enhance flexibility but would lack the transactional safety required for POS systems. SQL databases are usually more sturdy and reliable and are table based, but there is a learning curve to learning that language. NoSQL is not table based, it is based on key-value pairs and documents, so it is easier to pick up and learn compared to SQL.

- **Single vs. Multiple Databases**: Using multiple databases allows logical separation (e.g., transactions, inventory), reducing contention on single databases and optimizing performance. However, a multi-database approach requires careful synchronization management.

7.8 Software Architecture Diagram Modifications

- **Updates:** Changed the Account Database that was previously there to split it into two databases: employee and customer. These each respectively contain different information, which makes it less cluttered, and more efficient to access a specific type of information.