Clothing Store Point-of-Sale System

Software Design Specification

Spring 2024 Group 7

Triet Lieu, Software Engineer

Konrad Kapusta, Software Architect

Prepared for

CS 250: Introduction to Software System

Instructor: Bryan Donyanavard

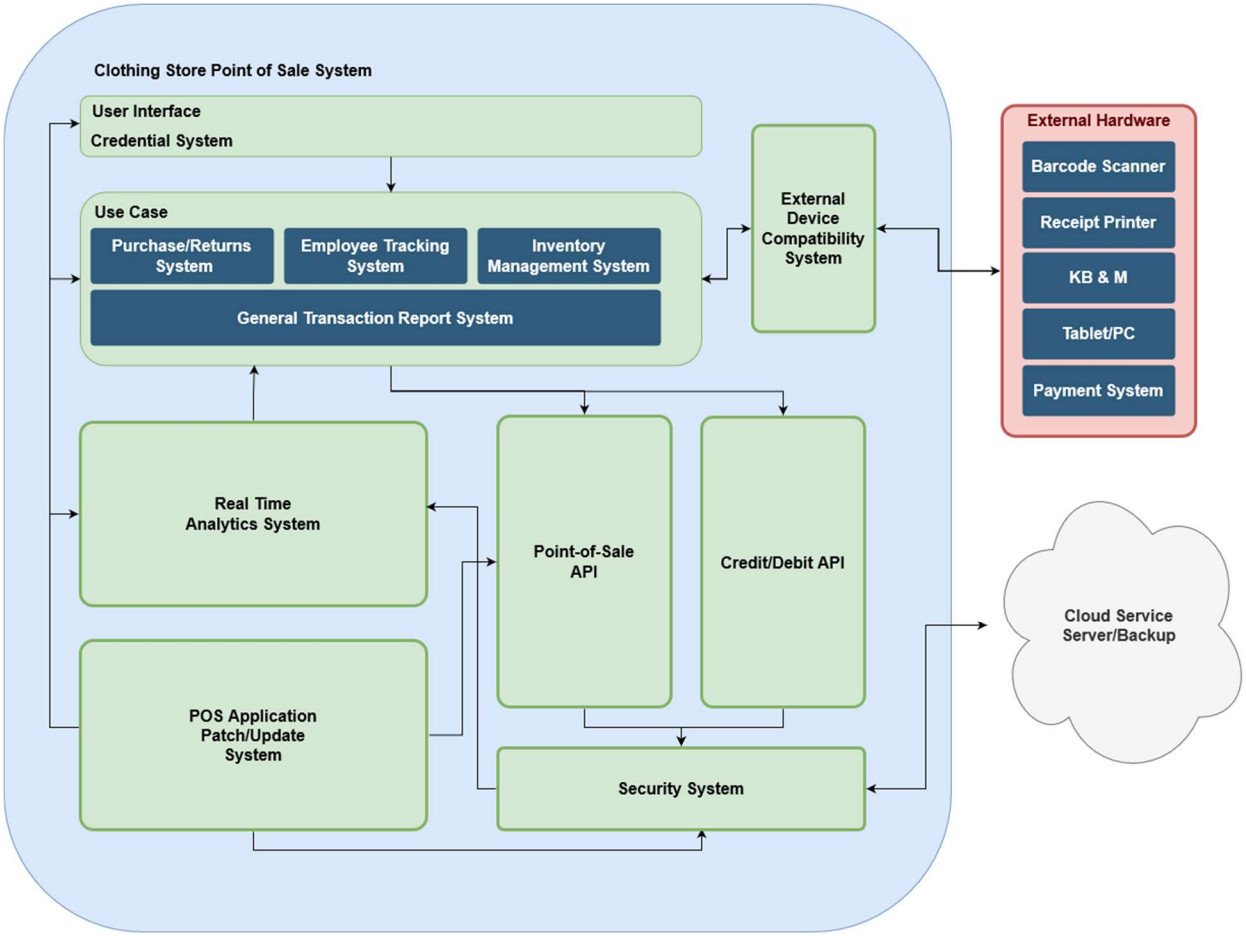
1. System Description

This clothing store point-of-sale system provides a time-saving tool for retail personnel to efficiently conduct all tasks needed to run a physical store, from recording inventory and transactions, to processing purchases, to looking up items. Inventory is automatically updated by integrating with purchases and returns. Items are searchable by the item's ID, price, size, and other attributes a manger sees fit to define.

The system can be installed onto supported barcode scanners and allows for processing card payment through a third-party vendor. An app that supports both iOS and Android can be downloaded into phones and tablets. Both connect to a secure cloud database that backs up and synchronizes data exchange across different store locations.

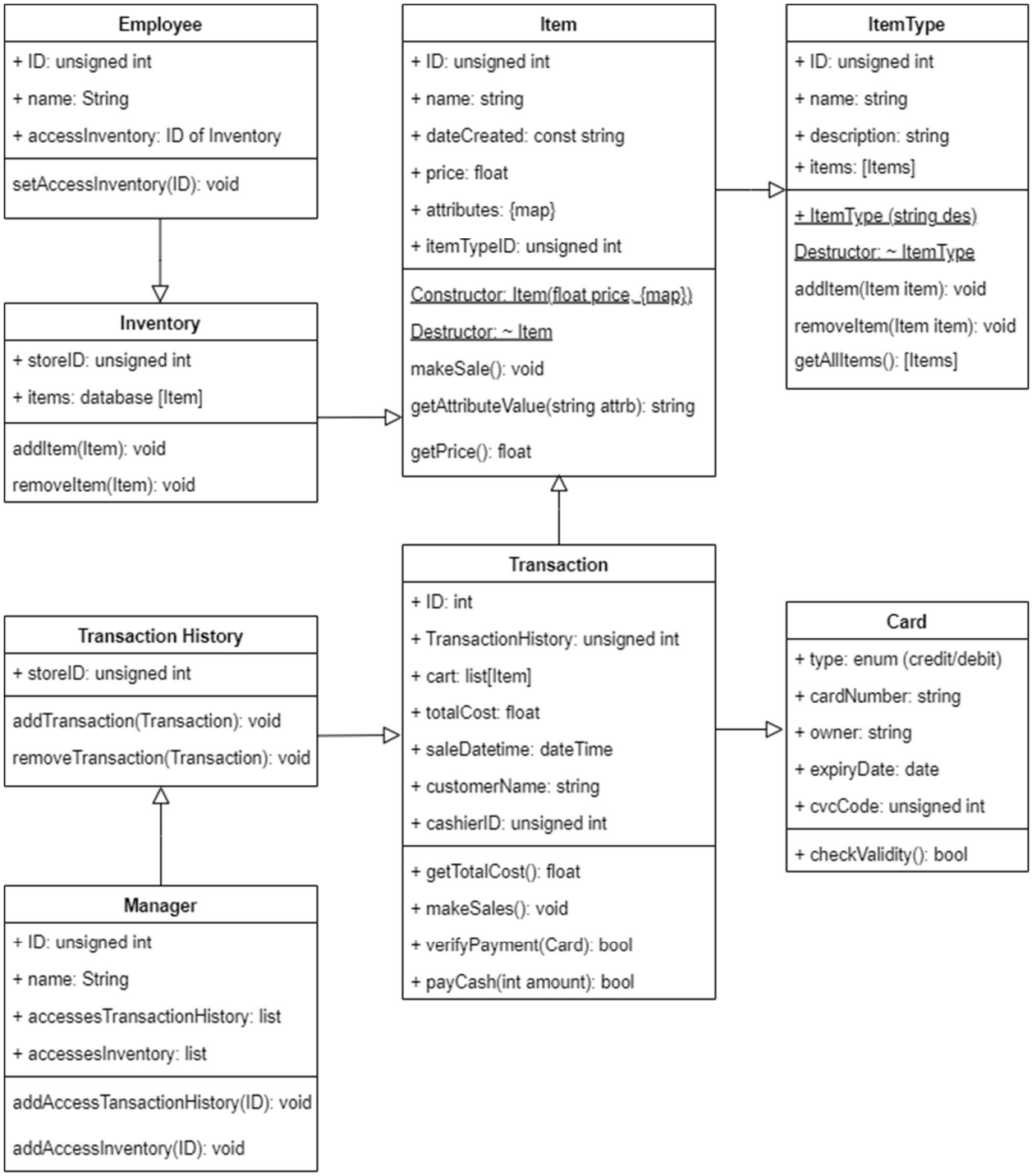
2. Software Architecture Overview

A. Architecture Model



	Description
User Interface	The “view” component of the system with which a user interacts, it should be intuitive and responsive. Receives commands from “controller components”.
Credential System	Manages logins. Use to create or remove users as either Manager or Employee. Must be robust because it grants access to the rest of the system
Purchases/ Returns	Integrates with Inventory and Transaction components and write to their databases automatically.
Employee Tracking	Keeps logs of employee attendance. Ensures actions allowed on the system are only those that relate to authorized store business.
Transaction Report	Give summary accounting of sales, loss, and other metrics that inform the business of which products to stock and how it can improve sales.
Point-of-Sale API	Allows an authorized developer to write software to affect and enhance the functioning of the other components, especially relating to sale and inventory
Credit/Debit Card API	External API developed by third-party vendor that the system calls to verify and charge cards for transaction payments.
External Device	Driver firmware that can be installed into external devices to allow them to integrate with the system. Design requires knowledge of hardware.
Security	Ensures data and the system cannot be hacked.
Cloud Service	Database system to store inventory, transaction history, and other data. Allows the owner of the head of a chain of affiliate business to view the data contributed by individual stores

B. UML Class Diagram



Class	Description
Item	Profile of an item, not the specimen. **attributes** is a map whose keys are attributes like 'size', 'color', and values are specs like 'LG', 'red' for Item. **getPrice()** returns price float. **ID** can be assigned by a database. **dateCreated** is a constant string set as profile's creation date. **itemTypeID** links Item to ItemType via ItemType's unsigned int ID **Constructor** get inputs of price and map {attribute: spec, etc} to assign **delete Item** deletes Item, but must be separately removed from Inventory **makeSale()** decreases the count of the Item in inventory by 1 **getAttributeValue(attrb):** eg getAttributeValue('size') → 'small'
ItemType	eg "women shoes", "men dress pants". Use **getAllItems()** to field a list of related Items to suggest to customers. **description** is a string that explains this ItemType in greater detail **items** stores a list of Item profiles belonging to this ItemType **Constructor** takes string input and assigns it to description member. **Destructor** calls removeItem(..) on all Item references in items member list. **(add/remove)Item(Item)** puts input Item into or removes it from ItemType **getAllItems()** returns Items under this ItemType
Card	Denotes a card that is used to pay for a transaction. **type** is an enum that identifies card as of credit or debit type **owner**: string, **expiryDate**: date, **cvcCode**: unsigned int, and **cardNumber:** string are credential details linked to the card **checkValidity()** returns True if all the credential details pass verification.

Transaction	Denotes a register checkout. Items comprise a Transaction, a Card can be used as the payment method, and an Employee serves as the cashier. **cart** stores the list of item purchases. **saleDatetime** tracks the purchase time. customerName is a string. **cashierID** is unsigned int that links to an Employee. **getTotalCost()** returns the sum of all Item's prices. **makeSales()** calls makeSale() of all Items in cart to reduce their inventory count **verifyPayment(Card)** has input Card call checkValidity() to verify card **payCash(int amount)** returns True if the cash amount can cover the total cost
Transaction History	Stores Transactions for Store referenced by **storeID** unsigned int **[add/remove]Transaction(..)** [adds/removes] reference to that Transaction
Inventory	Tracks Items in **items** database for Store referenced by **storeID** unsigned int **[add/remove]Item (Item)** [adds/removes] reference to that Item
Manager	An admin user who can access multiple TransactionHistory and Inventory **accessesInventory**: list of ID of Inventory to which this Manager has access **accessesTransactionHistory**: list of ID of TransactionHistory granted access **addAccess[TransactionHistory/Inventory]** (ID): Grant this Manager access to [TransactionHistory/Inventory] referenced by input ID
Employee	A non-admin user with access to only 1 Inventory referenced by **permission** **setPermission(ID)**: Set Inventory that this Employee can access

3. Data Management

To devise the design and organization of our database systems, we follow these steps:

1. Recall the tasks we want the point-of-sale system to manage and execute.
2. Recall the tools and procedures we implemented and drew to solve such tasks.
3. Brainstorm the types of data that need to be feed into such tools and procedures for them to produce the results that qualify as solutions to the tasks we intend our point-of-sale system to solve. Such data, their content and their type, will be the data that our database systems must record.

Examples of Formulation

Task	Data	Format	Source
Allow employees and admins to login into the system	Login permissions (identity, username, password, storeID)	Password is encrypted. The rest will be string.	Manually chosen by user
Track which employee works at which store and their job title (trainee, regular, experienced, manager)	Store location, store's manager. Employee's store location, work hours, current length of service	Store location: string Store's manager: ID Employee's store location: ID Employee's work hours: string Current length of service: int	Most string types will be manually recorded. Current length of service is computed by the system

Implement a “loyal customer” program that grants and alerts such customers to chosen deals on items	Customer’s name, contact info, address, membership tier, TransactionID to link to the transaction history	Name: String Phone no: string Membership tier: int TransactionID: int	Data relating to customer’s identity will be manually recorded. TransactionID will be given by the database

For the tasks we anticipate, MySQL will be a reliable starting system. Being a relational database, SQL emphasizes structure to data and offers a simple, but effective language in structured query language for retrieving data based on their relations. Items, transactions, and inventory represent entities that are structures and so a point-of-sale system that works with such data will be well-managed by a relational database like SQL.

Some examples of the tasks that our point-of-sale system is intended to manage were described above, such as managing employee logins, employee’s store work location, and a customer list for a “loyal customer” program.

For each entity that we must manage, we will create a table for it in MySQL. For example, we will create a table for all these pieces of data: product, inventory, store, sale transaction, item return, refund. We are using one database with tables for all such entities so to maintain data integrity and help with query performances as multiple tables can be joined together much simpler in one database.

Employee Table has columns: ID (as primary key), name, login credentials, position, and contact information, and transaction id (for transactions done).

Customer Table has columns: ID, name, address, contact information, rewards status, transactionID (referencing foreign key in Transaction Table).

Item Table has columns: ID, name, description, price, quantity (inventory count), and category (referencing foreign key in Category Table).

Transaction Table has columns: ID, transaction date, employee id, customer id, amount paid, items, payment type, and transaction type (any of refund, sale, exchange).

To maintain the integrity and quality of our data, we use multiple tables with the aim to normalize our data and eliminate any redundancies in stored data. With more tables comes a more complex system to design, however, it allows us to make a more efficient system in terms of performance and, perhaps, data integrity.

A second consideration to our data management strategy is to store a hot copy of all the data in a separate cloud server backup. It is important to have multiple copies of data among different storage levels as a preventative measure in case something were to go wrong. We will also make sure to monitor database activities often.

4. Development Timeline