Point of Sale System Report

Juan Pablo Zendejas

8 December 2023

Contents

1 User Manual

This chapter focuses on the interaction between the software system and the end user. We will go over how the employee, manager, and customer interact with the system and provide basic instructions.

1.1 For the Employee

Hello GorpCore Employee! This section will teach you everything you need to know in order to provide customers with a unique shopping experience only the GorpCore Group can provide. As a cashier, you will be the mediator between the customer and the proprietary point of sale technology that powers our stores. It's important to always maintain a smile and a positive attitude to keep customers coming back for more of their favorite clothes.

The cashiering kiosk is where you will spend most of your time. Your manager will provide you with an employee ID card that has a log-in barcode on the back. When you are manning a register, you simply have to scan your ID on the register to authenticate yourself and get it ready to process transactions. You should see the cashiering view on the touch-screen, which simply gives you an option to start a new transaction.

When a customer comes to purchase something, use the touch-screen to initiate a new transaction. Scan the barcodes on the items that they want to purchase, and make sure that the correct item name is displayed on the screen. The total cost is added up for you and the customer. Once the customer is ready to pay, use the touch-screen to tap the "Finalize Purchase" button. On the customer's side, they will see the appropriate prompts and submit their payment. If the customer wants to pay in cash, the kiosk comes with an automated cash machine that will accept bills and coins and dispense appropriate change. There is no need for intervention from the employee.

After the transaction is finalized, be sure to give the customer a warm goodbye and hand them their goods. When you're ready to end your shift or leave the station, be sure to press the Log Out button to ensure software security.

1.2 For the Manager

As a manager, you handle normal employee duties and can view the status of the store you manage. You can access individual transaction logs from the kiosks, and also use a web interface to view total transaction logs from your entire store. Finally, you have the ability to manage the price of merchandise sold.

On the kiosk, you just have to scan you own employee ID. Then, there is a unique option to view logs from the kiosk. These logs are only for the specific machine and include the items purchased, which employee processed the transaction, payment method, and other important transaction data.

On the web interface, simply use your assigned username and password to log in. There, you can access store-wide transaction logs which have aggregate data from every in-store kiosk. Finally, you have a unique ability to update merchandise data. The website displays a list of all items that are sold in the store, along with their price. In this list, you are able to delete an item or modify the details and price of it. Finally, there is a button to add a new entry for merchandise.

Finally, the manager is also able to manage employee accounts. On the same web interface, there are menus for showing a list of employees and their information like ID and name. Similarly to the merchandise page, there exist buttons to both add and remove employees or update their information.

1.3 For the Customer

As a customer, your role should be easy and enjoyable. When you have the goods you'd like to purchase, simply hand them to the cashier so they can scan it on the kiosk and add up the total cost. When the cashier asks you to pay, simply follow the instructions on the touch-screen. You can choose to either pay with a credit/debit card, or insert cash in the machine. You will then be prompted if you'd like a physical receipt or not. After that, you're good to go home with your amazing clothes.

In addition, there is also an online web store where you can purchase clothes and goods to be delivered straight to your home. On the website, you can use the links provided to register an account by providing a username, password, and personal information. Once you have an account and a payment method linked, you can browse the online selection of clothes styles. Click the "Add to Cart" button to save an item into your virtual shopping cart. When you're ready to submit your order, click the big Checkout button available on every page of the website to jump straight to the order process. There, you can verify that your order is complete, and then charge your payment method so we can process your order.

Your online account can also be used in-store to earn deals and rewards. Simply tap the button on the touchscreen before you pay and input your account details to link your purchase with your rewards account.

2 Software Requirements

This chapter serves as a base specification for the new Point of Sales system that we will be implementing for the client Gorpcore Group. It will be used by the cashiers for handling daily customer transactions, interfaced by the corporate offices to handle price setting, and used by managers to keep track of their store's performance and revenue.

The client requirements are divided into 4 sections, User Requirements, Functional Requirements, Non-Functional Requirements, and Other Requirements. The User Requirements are a quick, non-technical overview of the system for the client. The functional and non-functional requirements are a technical overview of how the system should feel and operate. Lastly, the Other Requirements section details caveats and other considerations during the creation of the system.

2.1 User Requirements

The planned Point of Sale system will integrate the physical PoS machines that the customer and cashier will interact with, along with the remote, corporate machines that it will send important data to. However, one of the goals is to still function without internet required.

The PoS system will be designed to include a touchscreen and keyboard combo that the cashier will use, along with a customer-facing touchscreen and card reader which will be used to retrieve data from the customer. It should also be able to automatically accept cash. The system will be designed to fit in to the existing gorp-core fashion style present in the stores, along with exuding a sense of fashion that will make the customers feel good for shopping. We aim for a maximalist style, opposing industry trends of minimalism; this means no sharp lines, using smooth curves and yet not being too busy.

Finally, the managers will be able to use their assigned credentials to access customer and transaction records to monitor the performance of their storefront. This will be done either manually through the use of the kiosks, or from the corporate web portal.

2.2 Functional Requirements

In this section, we will focus on the technical requirements required for the functionality of the system. A graphical overview of how the system will function is displayed in Figure 2.1. The customer can register an account, and with the help of the employed cashier, can purchase clothes from the store or for delivery from other stores and the warehouse.

Included in Figure 2.1 is a database with 3 tables for important data: the Product Table, Transaction Table, and the Account Table. The product table will be a list of every product sold by Gorp-Core stores. The transaction table will contain a record of every customer transaction. Finally, the Account Table will hold both customer account records and employee account records for the cashier and the managers.

The system is then further divided into a Kiosk and a Web Interface. The Kiosk will be used by the customer and cashier, while only managers will have access to the web interface. The web interface will let managers update merchandise details (including pricing), access transaction logs, and access customer data.

This document will only go over the Register Account, Purchase Clothes, and Update Merchandise Details use-cases.

2.2.1 Kiosk

The Kiosk will run custom-built software based on an ARM Architecture. The processor will handle the input from the touch-screen and keyboard, along with drawing the cashier's interface to the cashier-facing screen. In addition, it will also handle input and output to the customer-facing screen, which is only a touchscreen. One of the requirements is that should the screen be damaged, the machine should still be easily repairable and operable again. For this, we will provide simple USB and HDMI connections to allow the use of an external display in the event of damage.

2.2.2 Web Interface

The web interface will be accessible anywhere by any manager. It will handle the updating of prices and merchandise, in addition to letting the manager both access transaction logs and customer data. One important function it will serve is to allow the manager to create cashier accounts and the barcodes that cashiers will use to authenticate themselves to the system. Of course, managers will have to authenticate as well.

2.2.3 Use Cases

The first use case is the Register Account use-case, shown in Figure 2.2. The user will provide their information into the Kiosk's touch-screen, which will then be stored into the Account table for future use. The Account Table will hold user personal information, along with their transactions completed with their account and other relevant data for the rewards system. Alternate flows should be considered for, e.g., an invalid e-mail or phone number or when the connection to the database is down; in such cases, an account should not be required. Exceptional flows, such as a system failure, should present a display that asks customers to use another register.

Next, we will look at the Ordering Merchandise use-case displayed in Figure 2.3. In this use-case, the customer approaches the kiosk and cashier with the goods they would like to purchase. The cashier then begins the transaction, and scans the clothes provided by the user. The kiosk fetches the information such as product price and name from

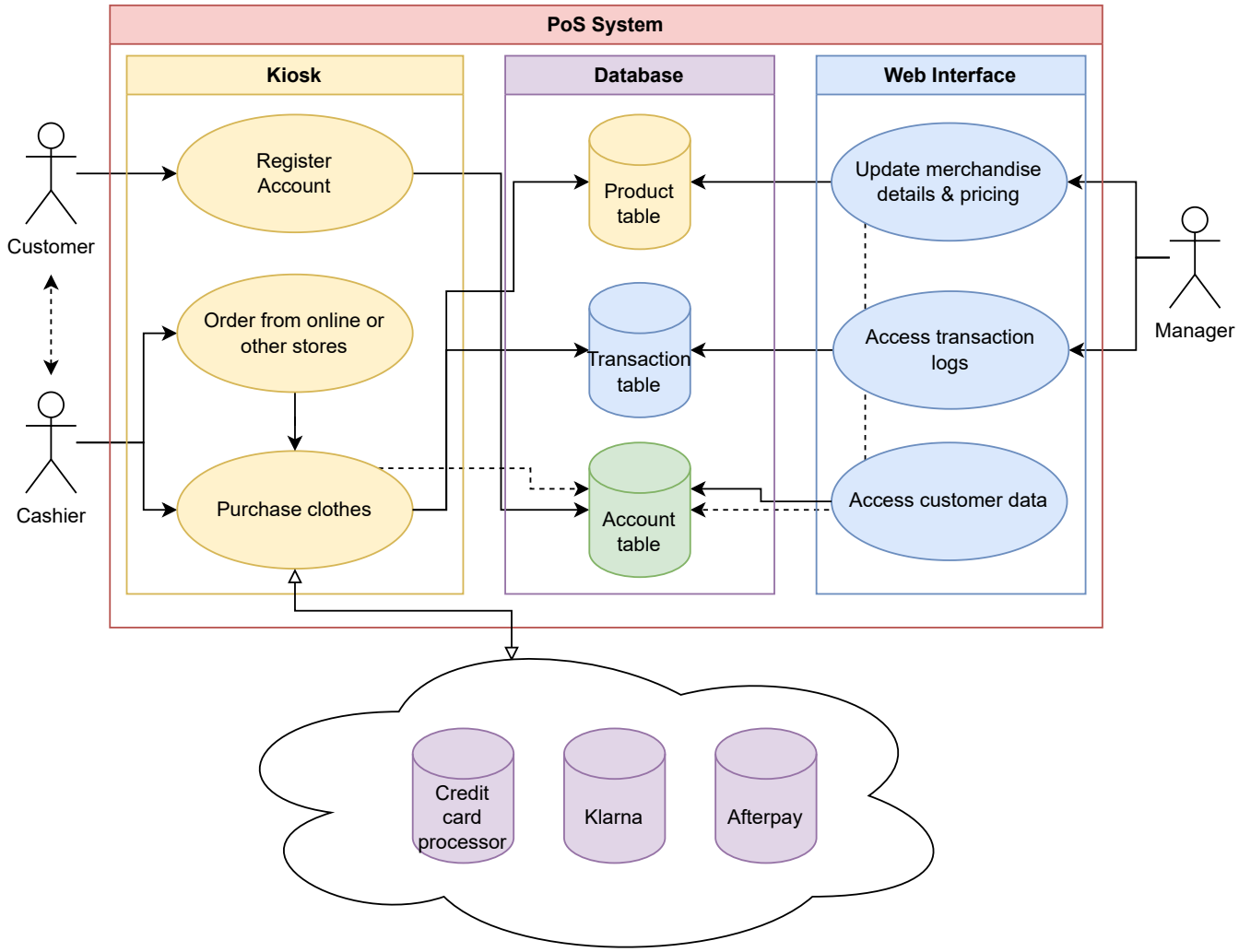

Figure 2.1: Software Architecture Overview

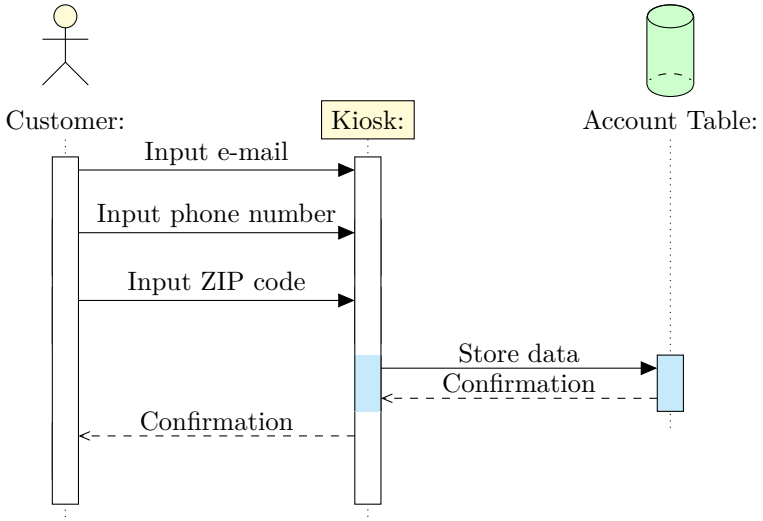

Figure 2.2: Use-Case: Register Account

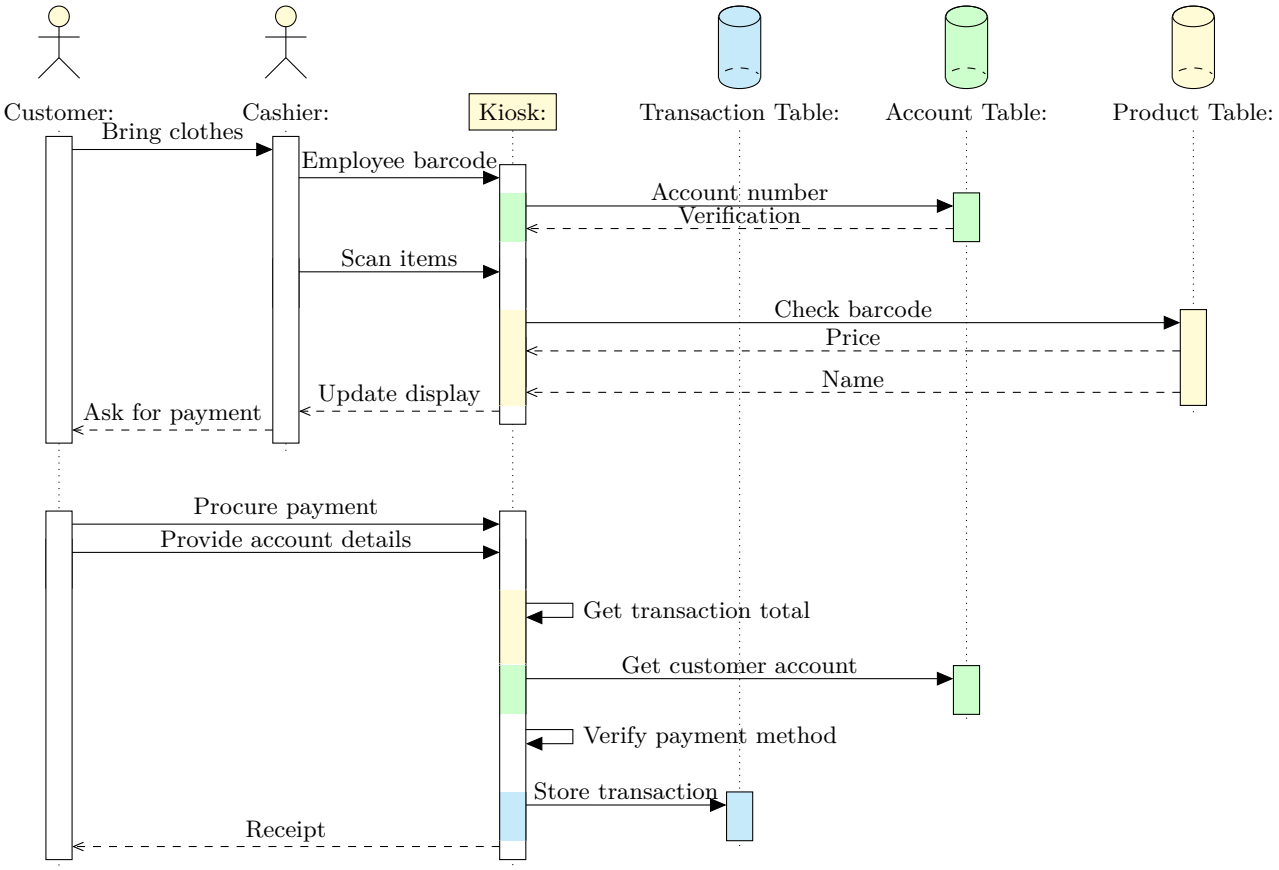

Figure 2.3: Use-Case: Ordering merchandise

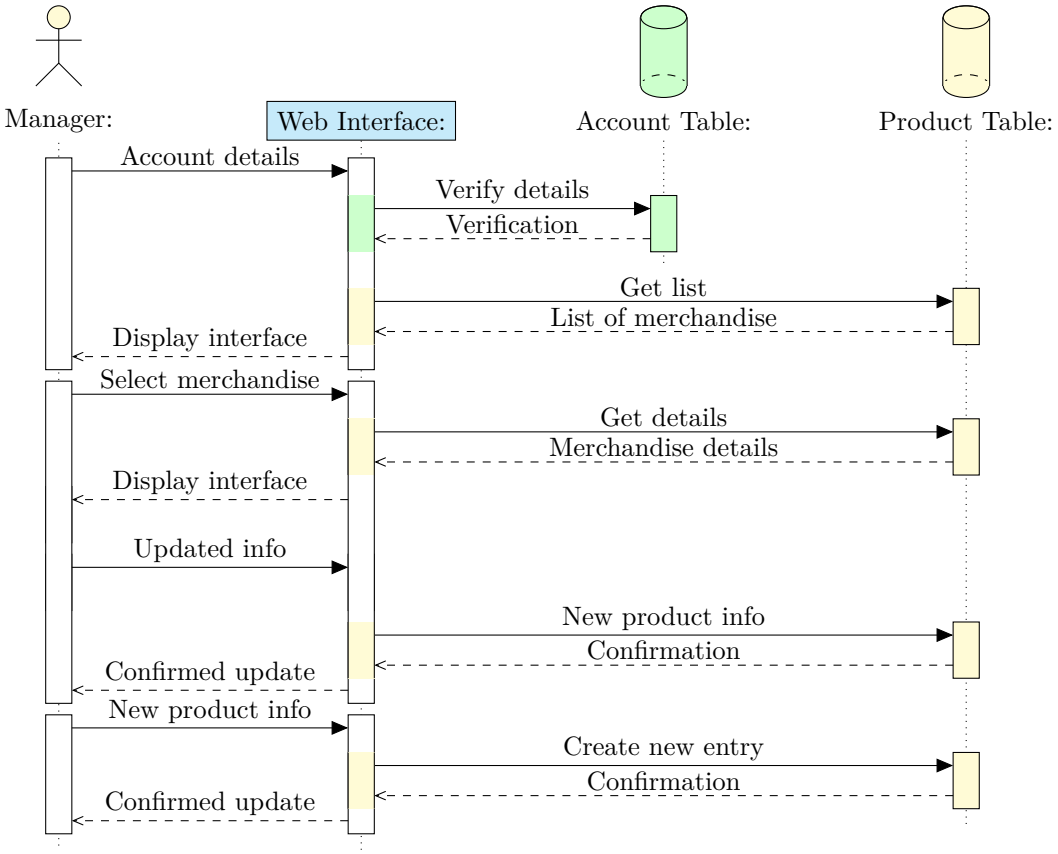

Figure 2.4: Use-Case: Updating merchandise details

the product table, and displays this running total on the customer's screen. Finally, the cashier asks for payment.

The customer can then use the kiosk to initiate payment through MasterCard, Visa, Contactless, EMV Chip, Cash, or even Klarna and AfterPay. The kiosk then loads the total and the customer's account, which is used to verify the payment method as valid and store the transaction both in the internal log and on the remote transaction table. Alternate flows include diverting the user to register an account if they do not have one, or applying discounts to certain articles of clothing. An exceptional flow to consider is the loss of internet access, in which case transaction logs should be stored to the kiosk's internal disk for later uploading.

Finally, we will examine the use-case for updating merchandise details as displayed in Figure 2.4. First, the manager must confirm their identity using their account details, which are verified through the account table as a manager's account. There are two operations displayed in the use-case diagram. The first is updating merchandise, in which the manager selects an item from the displayed list, its details are shown, and the manager then edits the details. These edited details are them stored in the Product Table by the web interface. A similar flow occurs for adding a new product; the manager

simply provides the new details immediately.

The use-case diagrams for accessing transaction logs and customer data are not included because they are simple use-cases. The manager authenticates to the system, and is then presented with options to access either. The logs are then displayed to the manager.

2.3 Non-Functional Requirements

One of the goals of this system is to be fast, easy to use, and intuitive for the cashier and the customer using it. This will reduce employee training time and improve customer satisfaction. To achieve this goal, it is important to take into considering the design of the kiosk's user interface. We aim to keep it in line with the store's branding, incorporating maximalism and smooth curves to feel inviting and natural.

The system must also be fault-tolerant and reliable. During times of internet outage, payments should be achieved through traditional means (cash), or deferred for later processing. Transaction logs generated during this time should be cached locally on the kiosk for later uploading; the cashiers should also be provided a receipt they can keep for physical records. This also means that the product table must be locally cached for redundancy.

The physical kiosks should be easy to repair as well. While we will be incorporating a custom design, all the internal components will be as stock as possible and easy to replace with off-the-shelf parts.

2.4 Other Requirements

Not every detail is described in this document. During the construction of the system, careful choices must be made to fill in the gaps in accordance to the client's needs. In addition, the system must be made to work the account details into a rewards system run by the store. Account details then must be shared with this other system in order to facilitate its functionality, and this should be considered during development.

3 Software Design Specification

This software design specification outlines the internal operations of the PoS system in greater detail, useful to system developers who will create most of it. It is divided into a description of each class and its fields and methods, then a development plan and timeline.

The UML class diagram in Figure 3.1 is designed to illustrate the system's structure and the interaction between each class, thus ensuring a streamlined user experience. All of the classes within the Clothing Store Point of Sales System are dependent upon one another, and the deletion of a class would break the point of sales system.

3.1 Description of Classes

3.1.1 Employee

The employee class holds data related to each employee. The employee's fields include an employee ID, a name, and a password. Employees have the ability to work the kiosk and search for items. The unique employee ID is used to authenticate the employee to the physical kiosks using a scanned barcode.

3.1.2 Manager

The manager class inherits the employee class. It has new abilities to find employees in the system, or add and delete employees from the system. Managers can also access any data inside any of the tables in the Database, which is useful for business operations.

3.1.3 Customer

The customer class represents the online profile a customer creates for the online and physical store. The customer class holds the customer's name, email, password, and address, and an assigned ID. After the customer profile is created, the customer is then able to close their account, change and verify their password, view their transaction history, and change their username.

3.1.4 Kiosk

The kiosk is a register, an in-person device that an employee must operate. The kiosk is given a serial number, and holds the scanned items for each transaction. The kiosk can also scan items, log in and log out employees from the kiosk, open the register, and check out customers using the shoppingCart array.

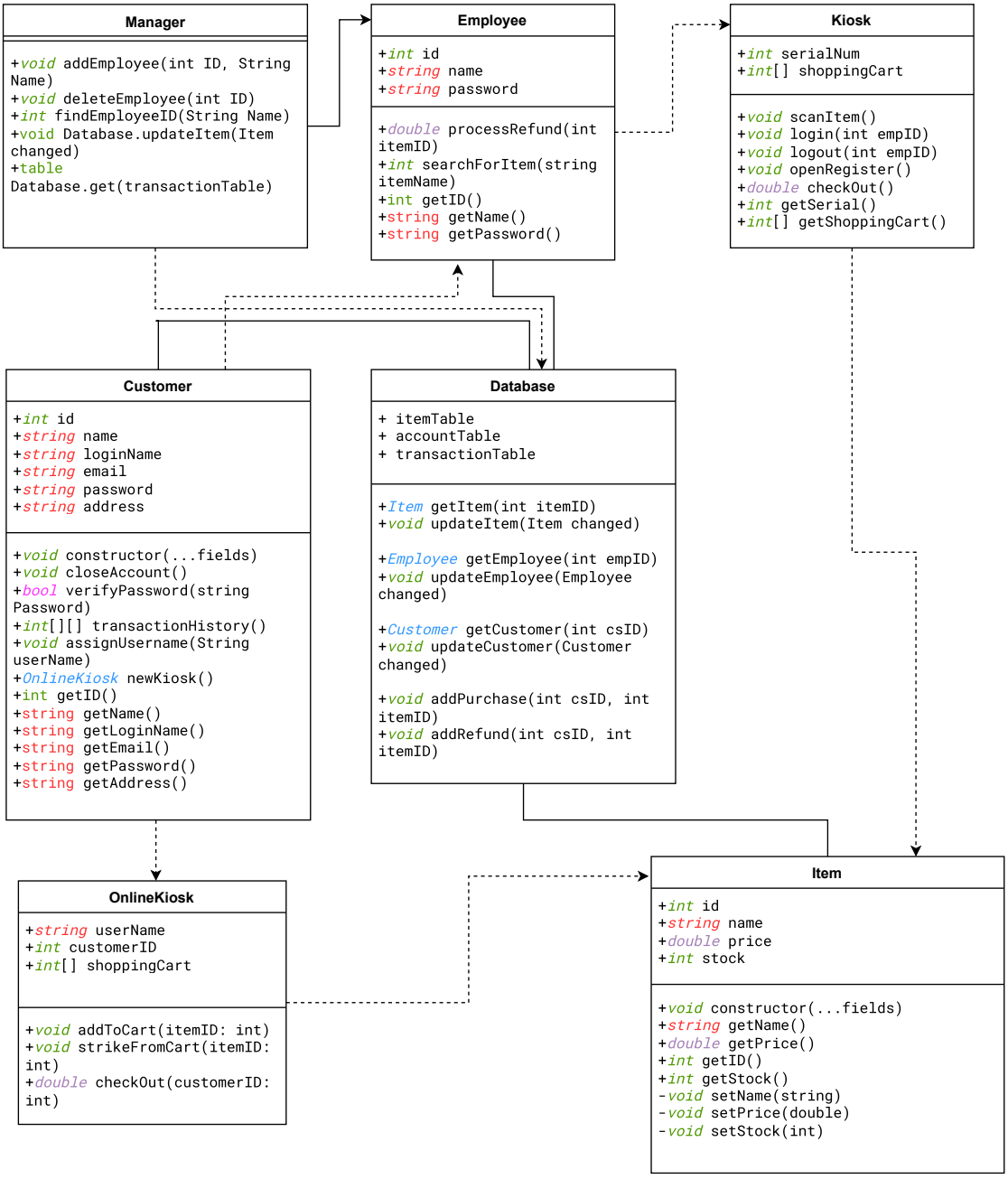

Figure 3.1: UML Class Diagram

3.1.5 Online Kiosk

The online self-checkout system the customer uses for the online store. It stores the runtime data and provides relevant operations for the web interface, like holding a shopping cart and processing the payment information. A new online kiosk instance is instantiated for each customer that wants to access the system. The customer's username and password are required in order to finalize transactions.

3.1.6 Database

This class holds all relevant business information needed both for bookkeeping and the operation of the software system. The database contains three separate tables, an Item table, a Transaction table, and an Account table. Each table is enumerated by unique IDs. The columns of the Item table include item ID, price, stock, name, and password. The account table has columns for ID, type (customer, employee, or manager), and name. For customer accounts, more data is held, including their address, payment methods, and login name. Finally, the Transaction table has columns customerID, date, itemID, price, and transactionType (refund or purchase).

3.1.7 Item

The item class is a representation of an item that is stored in the database. Each item is given a name, a unique item ID, a price, and a value that indicates that item's total stock in the store. Within the items class, the name, price, ID, and stock can be acquired through a getter method, while the name, price, and stock can be changed through a setter method. This includes name, price, and stock. The ID is printed on the tags as a barcode and used by the kiosks to identify what items are being purchased.

3.2 Description of Fields

1. Employee

   int employeeID Represents the unique employee ID.

   string name The employee's legal name.

   string password Used to authenticate the employee.

2. Manager
   - All fields are inherited from the employee class.

3. Customer

   int id An int that represents the customer's ID.

   string name A string that represents the customer's name.

   string loginName A string that represents the customer's username for their online account.

string email A string that holds the user's email address.

string password A string that holds the customer's password.

string address A string that holds the customer's home address.

4. Kiosk

   int serialNum Unique identifier for each physical kiosk.

   int[] shoppingCart An int array that stores the ID of each item that is scanned at checkout.

5. Database

   itemTable Stores a table of clothing items.

   accountTable Stores a table of all the employee and customer information.

   transactionTable Stores a table of all transactions, including refunds. This table can only be appended to.

6. Online Kiosk

   string userName A string that represents the username of the customer's account. This will be passed in when an instance of the online kiosk is created.

   int customerID The ID of the user that controls this kiosk.

   int[] shoppingCart An array of item IDs (integers) which represents the items currently in the user's shopping cart.

7. Item

   int id Identifies the item. Is present on each article's barcode.

   string name Display name of the product for register and online.

   double price Cost of item.

   int stock This keeps track of the total amount of this specific item the store has.

3.3 Description of Methods

1. Employee

   double processRefund(int itemID) Updates the total number of items the store holds, updates the transaction table so that the refund is logged, and returns the specific amount of money the customer is owed. An itemID is passed as an argument to process the refund.

   int searchForItem(string itemName) Finds an item based on the given name, and then return the found item ID from the table.

2. Manager

void addEmployee(int ID, String Name) Adds an employee to the accountTable. When adding an employee, if the employeeID already exists, the operation will be canceled, and the user will be informed of their error.

void deleteEmployee(int ID) Deletes an employee from the accountTable. The specific employee is found through their ID.

int findEmployeeID(String Name) Searches for an employee based on name. Returns the found ID from the accountTable.

void Database.updateItem(Item changed) Updates the pricing and quantity of items in the database.

table Database.get(transactionTable) Returns a table containing all of the store's transactions for use by the manager.

3. Customer

void constructor(...) Creates an instance of a customer and passes in the necessary data for the customer's ID, name, loginName, email, password, and address.

void closeAccount() Deletes the instance of the customer from the accountTable in database.

bool verifyPassword(string Password) Called when the customer logs in. If verify password returns as true, the user is admitted access to their account on the online kiosk. If verify password returns as false, the user will be denied access to their account.

int[][] transactionHistory() Returns a history of all of the purchases that a specific customer has made, from the transaction table.

void assignUsername(String userName) Changes the userName of the user.

OnlineKiosk newKiosk() Creates a new instance of the OnlineKiosk class that is specific to this user.

4. Kiosk

void scanItem() Called when a barcode is scanned. The scanned itemID is then added to the shoppingCart int array.

void constructor(int serial) Creates an instance of the Kiosk. It takes one parameter, the serial number for this kiosk.

void openRegister() If the register is closed, when this method is called the kiosk is opened.

double checkOut() Tallies up the total value of the items and return the amount the customer owes. If the customer fails to pay, the transaction is voided. It will then print an itemized list of all of the items in the shoppingCart[] array. The array is then emptied one item at a time, updating the number of items in itemTable, and then the transaction is logged in the transactionTable.

void login(int employeeID) When an employee works at a kiosk, they scan their employee badge so that management can keep track of who is working what registers, and when.

void logout(int employeeID) Close the register and tell the system that the employee is no longer working the register.

5. Database

Item getItem(int itemID) Returns the item held in the table at the specified index, which is the item ID.

void updateItem(Item changed) Overrides all of the data for the specific instance of the item.

Employee getEmployee(int empID) Returns the instance of the employee class that is stored in the accountTable at employeeID.

void updateEmployee(Employee changed) Overwrites the item's data by replacing the instance of the item class at the specified index with a new instance of the item class with updated data.

Customer getCustomer(int csID) Returns the instance of the customer stored in the accountTable at the specified index.

void updateCustomer(Customer changed) Overwrites all of the data of the instance of the customer class that is stored in the accountTable at the specified index.

void addPurchase(int csID, int itemID) Appends the item ID to the transactionTable when an item is bought, flagged as a purchase.

void addRefund(int csID, int itemID) Append the item ID to the transaction table when an item is refunded, flagged as a refund.

6. Online Kiosk

void addToCart(int itemID) Appends the itemID to the shoppingCart array and updates the online display, along with the total.

void strikeFromCart(int itemID) Called when the customer decides they don't want to buy something anymore. Removes the item from shoppingCart and updates the online display.

double checkOut() Totals the dollar amount of the items within the shoppingCart[] array, and then empties the shopping cart one item at a time while updating both the itemTable and the transactionTable in the database.

7. Item

void constructor(...) Creates an instance of an item and fills it with a provided ID, name, price, and stock.

string getName() Returns the name of the specified item.

double getPrice() Returns the price of the specified item.

int getID() Returns the ID of the specified item.

int getStock() Returns the stock of the specified item.

void setName(string) Private method. Changes the name of the specified item.

void setPrice(double) Private method. Changes the price of the specified item.

void setStock(int) Private method. Changes the number of items in stock in the database.

3.4 Development Team and Partitioning

Project Management (PM) responsible for budget, timeline, and gathering requirements from the stakeholder.

Software Architects (SA) responsible for designing an efficient database and general architecture for this system. This is based on the stakeholder's requirements, functional and nonfunctional.

Software Developers (SD) responsible for writing the code that meets the specifications created by software architects.

Testing Team (TT) responsible for testing the quality of the software as well as reporting issues, defects, and unintended behavior.

Maintenance Team (MT) responsible for maintaining the system after launch. Includes bug fixes, clothing updates, and user support.

Security Engineers (SE) responsible for making data secure and protecting the system against online threats.

UI Designers (UID) responsible for creating an appealing user interface for the customer to browse through.

UX Designer (UXD) responsible for improving the total user experience. Involves considerations of accessibility and close collaboration with the UI designers and users.

3.5 Development Timeline

Months 1-2: Planning and Design

PM Interview the client and ask questions to clarify the different requirements.

SA Design UML class diagram for point of sale system.

1. Define attributes required for each entity.
2. Define relationships between each entity

SA Design software architecture diagram for point of sale system.

1. Database design should comply with business rules.
2. Ensure that prices are correct and clothing items are for our store brand.

Months 3-8: System Implementation

UID Create a website that allows customers to access our store (online kiosk).

1. Make online webpage beautiful and representative of the brand.
2. Visually appealing website.

UXD Modify website to improve user experience.

1. Structure website so it is easy to access, navigate, and intuitive to use.

SD Create a database to hold an inventory of items, accounts for employees and customers, and a list of transactions.

1. Whenever a transaction is completed, inventory should be updated.
2. When an employee is hired, account table should be updated.
3. Database is backed up through a cloud system, synchronized across different store locations.

SE Ensure that the database system is secure. Authorized users only, encrypting data, restricting access.

Months 9-11: System Testing

PM Validate if the system is what the client wants, if we have met their requirements.

1. Meet with the client and verify that the system's functionality aligns with their vision. If not, go back to system design.

TT Verify the design of the system. Is this the intended behavior? If not, software developers have more work to do.

1. Check database behavior.
2. Check employee, manager, and customer functions and accessibility.
3. Using a UML use case diagrams as a guide, check the quality of the system by checking the interaction between classes and functionalities.
4. Test online kiosk and physical kiosk functionality.

Month 12: System Rollout

SD Monitor system performance.

1. If there are issues, fix issues immediately.
2. Review customer and employee feedback, update system accordingly.

MT Continue maintenance on the system.

1. Update online kiosk to include trending clothes and upcoming sales.

4 Data Management Strategy

Every system holds data. This data is extremely important for the operation of the system, and thus it should also be planned well to prevent growing pains. For the Point of Sale system, we opted to go for a standard relational database setup using SQL. Our database will contain three tables, Accounts, Transactions, and Items, which will make up the operational data for the business that is used with the Point of Sale system. Figure 4.1 shows the entity relationship diagram for our system.

First, the Accounts table stores all accounts for managers, employees, and customers. Each type is differentiated with a type column that holds "EMP" for employee, "MGR" for manager, or "CSR" for customer. There are multiple columns for the person's name, password, login name for use online, and then the user's email and mailing address for purchases.

Then, the Items table contains all merchandise sold at our stores. Each item gets a unique ID, which is also the same code that is printed on the item barcodes and scanned by the kiosks. Stored with each item is its price, stock, and name.

Finally, the Transactions table pulls together all the data. Each transaction of an item is associated with an order ID. The type field denotes whether the transaction was a purchase or a refund. We have two foreign key relationships to the Item table or the Accounts tab to associate the transaction with both a customer and an item. To explain crow's foot notation in the ER diagram, only one account can make transactions; and these transactions can be either zero or many. For items, a transaction can consist of many items but has to be at least one; and there is one transaction for item purchase. We also store the date the transaction took place, and the price that it was purchased at. This is because discounts could have been applied, or the price could have changed since it was purchased.

4.1 Why SQL?

SQL is better for relational data. Our store POS system involves product details, customer information, transaction records (both refunds and purchases). Furthermore, the data in our database requires many relationships between entities. For example, customers can make many transactions, and these transactions are tied to the customer through a customer ID. Also, to choose what items are in the transaction, the transaction uses item IDs in the Item table.

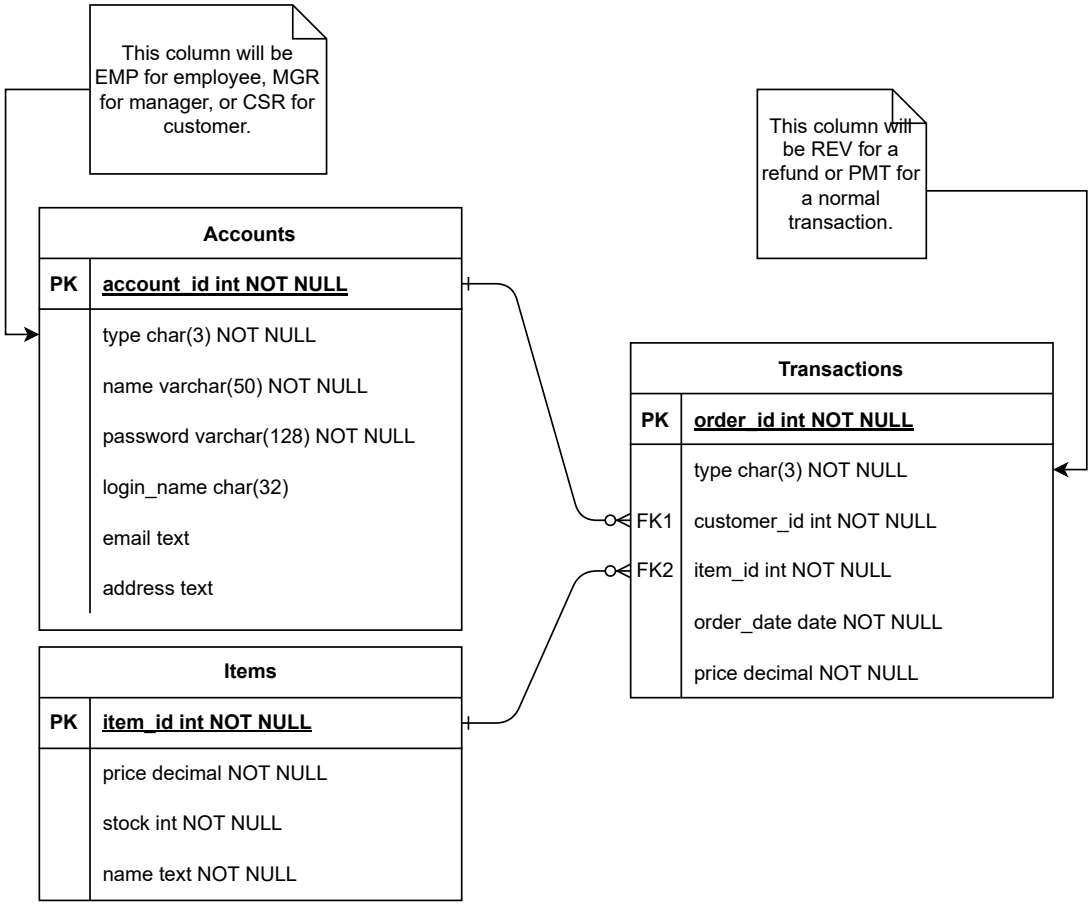

Figure 4.1: Entity Relationship Diagram

4.2 Data Format

We choose to consolidate all of our data into one SQL database, and then we split all of that into three separate tables based on datatype. We separated our data by customer, employee, and item.

The three tables were chosen to reduce data redundancy. By using foreign keys in the transaction table, we're able to reduce the duplication of the item name or the name of the customer who purchased the item. It also keeps related data of the same type in the same table, which helps with access speed.

An alternative organization would have been to separate the transaction and items table. We could have separated the stock field into an Inventory table, listing times when product was received and when product was sold, associated with a customer order ID. We opted not to go for this setup, because it would have increased the complexity of managing the data while not providing many benefits. The store already has bookkeeping for item stock delivery, which was not the goal of our Point of Sale system. Keeping the stock as one field makes it easy for managers to update the stock of each item when shipments of goods arrive.

4.3 Why not noSQL?

Our POS system is better used for highly structured data such as sales transactions, and customer and employee records, which SQL is better for. noSQL is useful for large data sets, and with our store POS system, there isn't a need for many read queries. However, if our POS system ever needs to handle a high volume of transactions noSQL would work better than SQL. In addition, since we don't have many locations, we have no need to handle problems like database distribution and consistency. It's enough to have one database handle all data traffic. This reduces initial system complexity and cost; as GorpCore Clothing Stores grows, the cost of the system can be re-evaluated.

5 Security Analysis

Since we handle plenty of customer data, it's important to keep this data secure from those who might want to steal and abuse it.

Our security considerations become mostly in terms of data and connectivity, since the hardware is bulky and heavy and runs software isolated from anything else.

5.1 Security by Design

Due to our control of the layout of the stores, each kiosk will be in a wired connection to an internal, unexposed network. This network will then have a single tunnel opening to another private network that is controlled by Corporate and has the database server. By maintaining the physical security of these cables, in addition to authentication techniques, the attack surface is reduced greatly. Keeping the connections wired and not on wireless broadcast also helps with the confidentiality of the customer data, along with improving availability if an attacker were to try and jam the wireless signals.

The online stores have the most attack surface and will need to be designed carefully. Using strong password hashing algorithms to keep confidentiality through user partitioning will help greatly. By associating users by ID and not by a user-provided choice, we can reduce chances of an SQL injection attack or internal system confusion over who's data is whos. We can ensure system integrity by locking down the web servers to run special read-only software, and using private tunnels to the database servers to reduce the database's internet exposure. Authentication methods using signing and encryption can ensure integrity of customer data. However, availability will always be a concern due to the fluctuating nature of the internet. Keeping redundant connections will help.

5.2 Vulnerabilities

One obvious vulnerability of integrity is the potential for spoofing scams that fool the cashier into marking down an item that normally is more expensive, perhaps by changing the barcode on an item to that of another. This can be remedied by proper employee training on checking that the name and picture of the item that the kiosk shows matches the actual item that was scanned.

Since cash is received automatically, there is a low risk of vulnerabilities that involve counterfeit currency or fooling the cashier into inputting the wrong cash amount. The kiosk machine will verify that the provided cash is valid by design.

As for the online systems, there is potential for vulnerabilities to arise in the underlying software and for malicious actors to break into our computer systems and access

confidential customer data like address, payment information, email, and phone number, or even transaction logs. By using safe security practises and constant maintenance, these vulnerabilities can be kept to exceptional situations and resolved quickly.

6 Software Testing

We performed many tests to ensure that our software aligned closely with the client's expectations and to reduce the amount of potential errors that could arise.

6.1 Unit Testing

6.1.1 Test Case 1: Create an instance of the employee class

For this test case, we aim to test the system's Employee class. To test this, we create an instance of the Employee class by using the class constructor to give an employee a unique int ID, string name, and string password. Now, to test that the class was properly instantiated, test getID(), getName(), and getPassword(). If the data returned by the getter methods matches the data passed to the constructor, then the employee class was correctly instantiated and declared. Since this test is self-contained within the Employee class, this is unit testing.

As an example, consider ID = 500, name = "Madeline", and password = "ilovestrawberries". We create an employee with emp = new Employee(ID, name, password). The test case succeeds if emp.getID() returns 500, emp.getName() returns "Madeline", and emp.getPassword() returns "ilovestrawberries".

6.1.2 Test Case 2: Create an instance of the Customer class

To test that the Customer class can be properly created, we will initialize an object instance of the Customer class. Do this by passing six pieces of data to the class constructor: an int ID, a string name, a string loginName, a string email, a string password, and a string address. Then, we will call six existing getter methods: getID(), getName(), getLoginName(), getEmail(), getPassword(), and getAddress(). If all of the data returned by the getter methods matches the data that we passed into the instance of the customer class, then that means the customer class is correctly instantiated and declared.

Consider a customer with ID = 1500, name = "Theo", loginName = "theounderstars", email = "theo@exok.com", password = "magicmirror", and address = "123 Park Place, Seattle, WA". We create an instance of the Customer class with cs = new Customer(ID, name, loginName, email, password, address). The test case succeeds if getID() returns 1500, getName() returns "Theo", getLoginName() returns "theounderstars", getEmail() returns "theo@exok.com", getPassword() returns "magicmirror", and getAddress() returns "123 Park Place, Seattle, WA".

6.2 Integration Testing

6.2.1 Test Case 1: Employee can use Kiosk

To test that employees can use the kiosk, we must properly create and declare an instance of both the Employee class and the Kiosk class. The kiosk is a system that will be physically accessed. For this test, the login(int empID) and logout(int empID) methods need to recognize the correct employee and log the employee in and out. In addition, the employees should be able to successfully use the kiosk. This is an example of integration testing because this test case relies on the successful functionality of two separate classes.

For this test, we will use the same instance of the Employee class as in Unit Test 1. However, we need an instance of the Kiosk class, which we will create with the serial number 1234: k = new Kiosk(1234). Then, we can test the login function with k.login(emp.getID()). The test succeeds if we can see that the kiosk has granted access to the employee, and is in a state ready to take transactions. Finally, calling k.logout(emp.getID()) should reset the state of the kiosk and leave it locked.

6.2.2 Test Case 2: Database can access Item class

The database accesses the item class in order to update the data of specific items. This test needs an instance of Item, and a working Database. We will call the void updateItem(Item changed) method, which passes in an instance of the item to be changed. This should add or update the specified item depending on whether that item already existed in the database. To check if the item has been properly updated, we will call the Item getItem(int itemID) class and compare the data we passed to the class with the data that was returned. This is an example of integration testing because this tests the combined functionality of two separate classes.

First, we create a new instance of the Item class. This will be the in-memory representation of the item which we test for proper storage. We will use the data id = 562578, name = “Levi’s Jeans”, price = 40.00, and stock = 50. The item is constructed with it = new Item(id, name, price, stock). Then, we store the data into the database using Database.updateItem(it). The test is successful if we can do it == Database.getItem(it.getID()) and the expression returns true.

6.3 System Testing

6.3.1 Test Case 1: Test customer online checkout feature

To test a customer’s ability to use the online checkout system, a multitude of functions must cooperate seamlessly. We must first create an instance of the customer class, and then fill the instance with data. We must then use the online website and browse through the online catalog until we find items we would like to add. When we choose to add an item to the shopping cart, the items will be held in an int[] array that contains the IDs of added items. This array is held in the online kiosk class. Every time an item is added to the shopping cart array, the dollar total of all the items in the array will be calculated in

the background by the checkout(int customerID) method. The customer class will then pass all of the relevant information to the online kiosk during checkout, and the total cost will be presented to the user. Next, the customer must manually input their payment information. Finally, the checkout is processed, the number of available items is updated in the database, a history of the transaction will be recorded in the database, and the order will be sent to a physical warehouse to be fulfilled. This is an example of system testing because it requires multiple pieces of the system to properly work. The customer class, the item class, the database class, and the online kiosk must all be completely functional for a customer to successfully checkout while using the online kiosk.

First, create an instance of a customer class and provide the required information (we did this in Unit Test 1). Then, that customer accesses the online kiosk using cs.newKiosk(321). When the customer adds an item to their shopping cart, call OnlineKiosk.addToCart(itemID). After the customer completes their purchase, the databases are updated: Database.addPurchase(customerID, itemID) is called in the transaction table, and Database.updateItem(itemID)is called in the item table. The test is a success when the customer physically receives the correct item, transaction history is updated correctly, and the item database has the correct number of remaining items.

6.3.2 Test Case 2: Test Manager's ability to update item prices

To test the Manager's ability to update the prices of items, three classes must work in tandem. First, the manager class is properly declared and instantiated. Then, the manager class will call the updateItem method in the Database class, which will call the Item class. When the Item class is called, a new instance of an item will be created, which will then override the data of the item already stored in the database. If all of these steps are properly executed, then the item data that was passed in the manager class will match the data that is returned by the Item getItem(int itemID) method. This is an example of system testing because this test case requires multiple classes to work in symmetry for proper output.

First, we instantiate the Manager, which is a subclass of Employee. We can use id = 1, name = "Jeff Bezos", and password = "1234". We create this by doing mg = new Manager(ID, name, password). Then, the manager calls Database.updateItem(it) to update the item class. This test succeeds if Database.getItem(it.getID()) returns the data provided by the manager.

7 Life Cycle Model

7.1 The model

During the semester and working with my team, I felt like we worked on a waterfall model life cycle that was fairly simple. Due to the nature of the lectures and assignments, we moved on from one stage to another in a very linear manner. First came requirements gathering, then software design, then data management and testing. There were a few times where we revisited older designs and updated them slightly, but it wasn't a true development cycle, it was just retrofitting the older designs to work with what we had now.

7.2 Reflection

For this small project, I'd say the waterfall development had mostly satisfactory performance. Since it was only 3 of us in a group, it was easy to coordinate and follow through together during the development of the software system. I don't think the waterfall cycle would work well for larger teams, however. A larger team would be tasked with more to do, and would have to constantly revisit their goals in order to keep them on track and following client requirements. I would have chosen something like the Spiral Model to keep iterating on good ideas and coordinating all the teams. If I were to just slightly modify the waterfall model we used, I'd probably ask for more time to revisit our requirements after the software design phase to check with the client and ensure that their needs could actually be met. It's one thing to request a feature, but after careful planning, could it feasibly be met? Those types of questions are why I think it's important to revisit earlier phases of the software life-cycle.