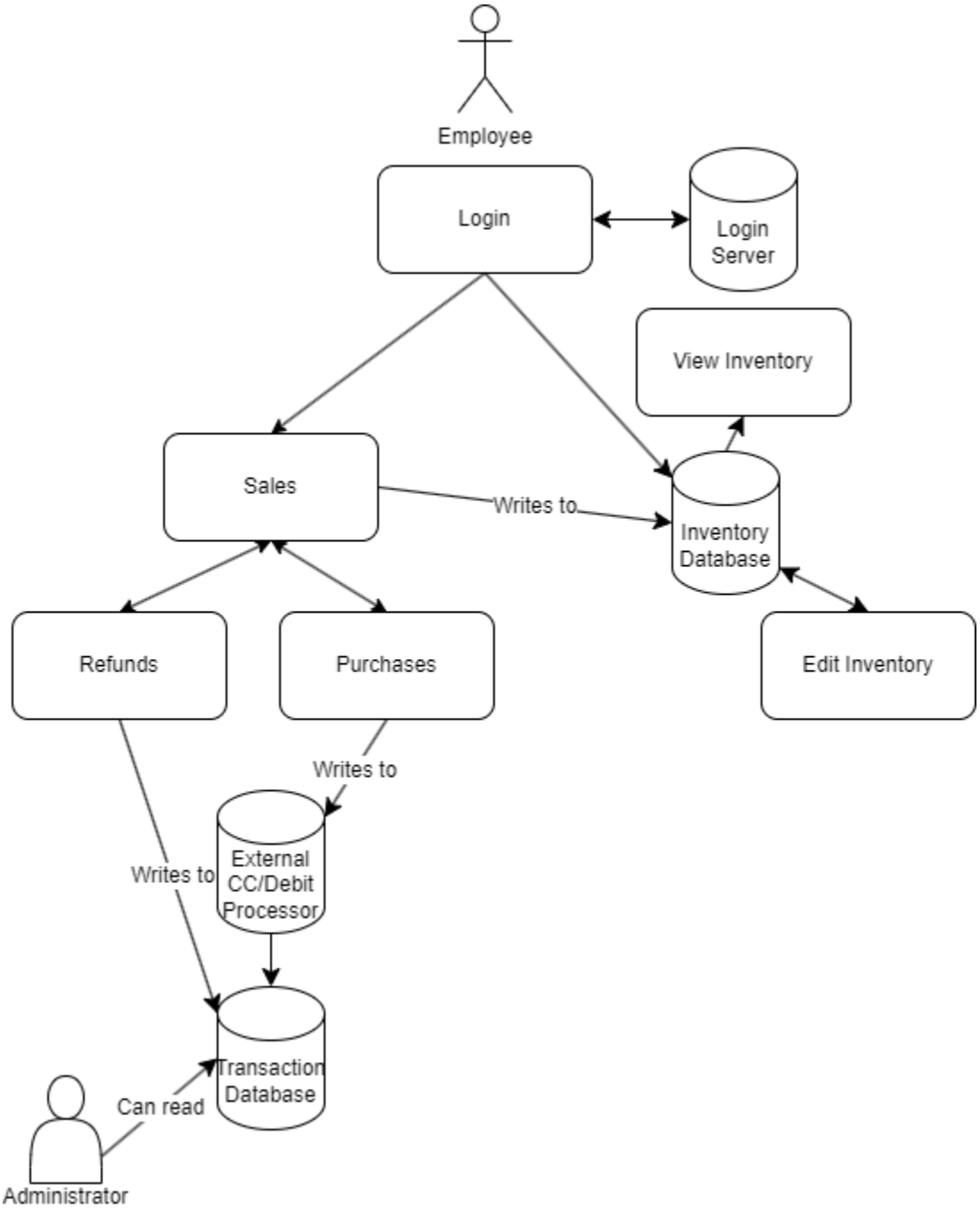
Clothing Store Point of Sale Software System

Javier Garcia Ramirez
Matthew Humphrey

The Clothing Store Point of Sale Software System (CSPSSS for short) is a software system intended for use by employees. Employees have the ability to organize and update inventory, as well as the ability to handle transactions such as purchases and returns.The system will be safe and efficient.

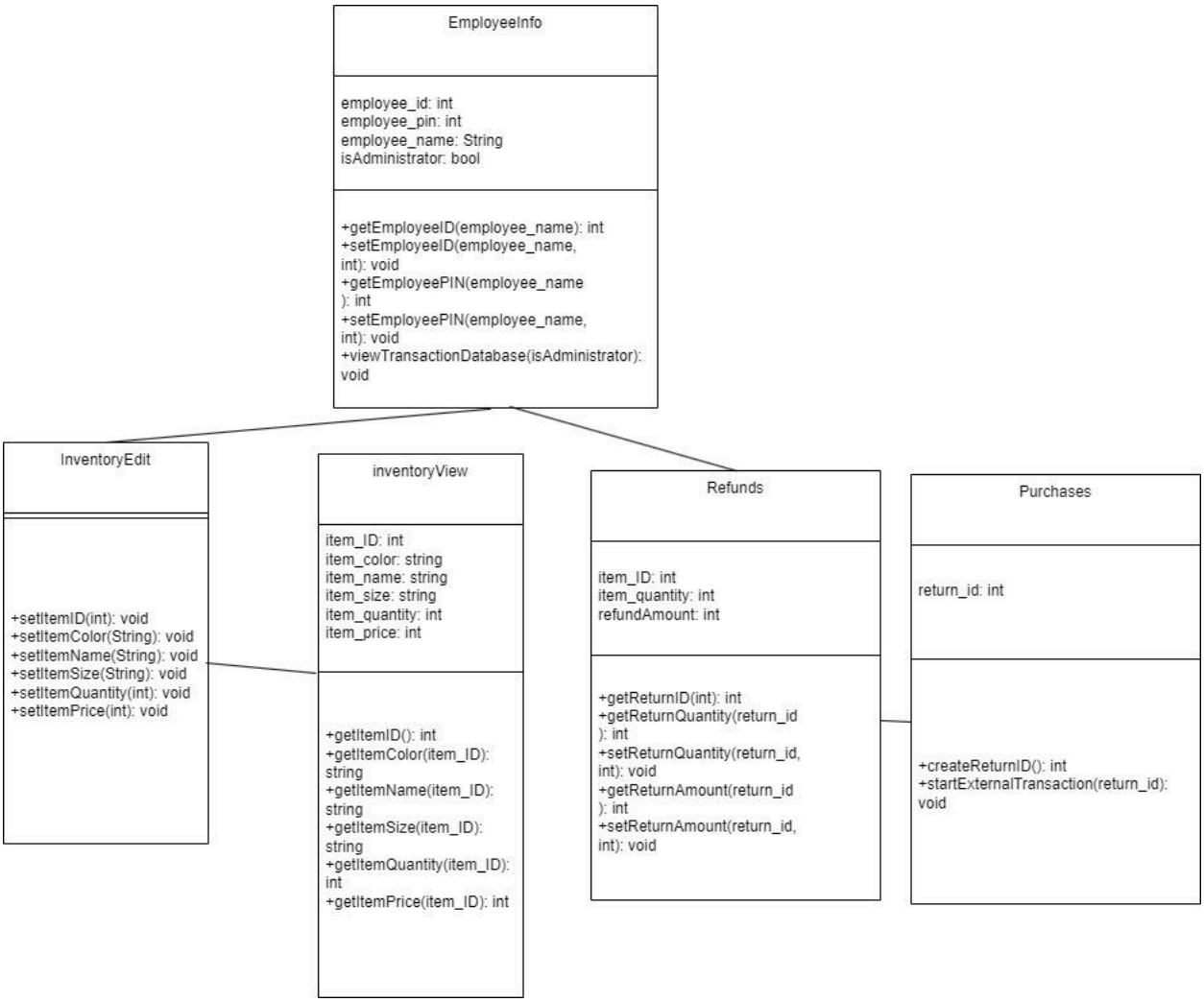
Architectural Diagram:



SWA Diagram description:

The diagram begins at the main intended user: The employee. The employee first enters the login component, which interacts with an external login server. This gives access to the two main fields, sales and inventory interaction. Employees can handle refunds, which directly writes to a transaction database, or they handle purchases, which writes to the external credit card/debit card processor, and then to the transaction database. The transaction database can only be read by the Administrator. Sales and refunds will then interact with the inventory database to update the quantity of products and such. The inventory field is the other main component. Logging in gives access to the inventory database. Employees can scan barcodes to retrieve item ID’s, from which they can either edit the inventory or view available stock. Employees are also able to manually enter item IDs.

UML Diagram:


EmployeeInfo
employee_id: int
employee_pin: int
employee_name: String
isAdministrator: bool
+getEmployeeID(employee_name): int
+setEmployeeID(employee_name, int): void
+getEmployeePIN(employee_name): int
+setEmployeePIN(employee_name, int): void
+viewTransactionDatabase(isAdministrator): void
InventoryEdit
+setItemID(int): void
+setItemColor(String): void
+setItemName(String): void
+setItemSize(String): void
+setItemQuantity(int): void
+setItemPrice(int): void
inventoryView
item_ID: int
item_color: string
item_name: string
item_size: string
item_quantity: int
item_price: int
+getItemID(): int
+getItemColor(item_ID): string
+getItemName(item_ID): string
+getItemSize(item_ID): string
+getItemQuantity(item_ID): int
+getItemPrice(item_ID): int
Refunds
item_ID: int
item_quantity: int
refundAmount: int
+getReturnID(int): int
+getReturnQuantity(return_id): int
+setReturnQuantity(return_id, int): void
+getReturnAmount(return_id): int
+setReturnAmount(return_id, int): void
Purchases
return_id: int
+createReturnID(): int
+startExternalTransaction(return_id): void

Class/Method Descriptions:

EmployeeInfo: This class interfaces with the external login server, using the employee_id and employee_pin credentials. The external server then updates the status of isAdministrator, to allow the user access to the transaction database (if they're an admin).
Administrator methods:
getEmployeeID(): retrieves the employee_id stored on the logon server.
setEmployeeID(): creates an employee_id that will be stored on the logon server.
getEmployeePIN(): retrieves the employee_pin stored on the logon server.
setEmployeePIN(): creates the employee_pin that will be used to login.
viewTransactionDatabase(): checks if the user is an administrator, and if they are, returns a copy of the transaction database.

InventoryEdit: The purpose of this class is to change attributes of items in the inventory, such as the item ID, color, name, price, size and quantity. The inventoryEdit serves as a subclass to the InventoryView class.

Employee Methods:

setItemID(): Creates/changes the ID of an item
setItemColor(): Creates/Changes color of an item
setItemName(): Creates/Changes name of an item
setItemSize(): Creates/Changes specific size of item
setItemQuantity(): Creates/Changes the quantity of an item
setItemPrice(): Creates/changes the price of an item

InventoryView: The purpose of this class is to look up items and retrieve their corresponding attributes. This class contains the variables for ID, Color, name, size, quantity and price. This class serves as the parent class to InventoryEdit.

Employee Methods:

getItemID(): When the barcode is scanned, getItemID() runs, and returns the ID of a specific item.
getItemColor(): returns color of a specific item, given an item_ID.
getItemName(): returns the name of a specific item, given an item_ID.
getItemSize(): returns the size of a specific item, given an item_ID.
getItemQuantity(): returns the quantity of a specific item, given an item_ID.

getItemPrice(): returns the price of a specific item, given an item_ID.

Refunds: This class is responsible for handling returns, and also interacting with the external payment processor for purchases. Each return is created as an object that has a unique return_id, which allows for employees to edit information about the return.

Employee methods:

getReturnID(): the return_ID is retrieved from being manually entered in
getReturnQuantity(): Retrieves the number of items being returned, given a return_id
setReturnQuantity(): Sets how many items are being returned, given a return_id
getReturnAmount(): Retrieves the cost refunded, given a return_id
setReturnAmount(): Sets the cost to be refunded given a return_id

Purchases: This class is for handling the purchases in the store. Every time a purchase is made, a return_id is created, in case a refund needs to happen. The Refunds class can then reference this number for its own purposes.

Employee methods:

createReturnID(): This creates a return_id variable in order for the Refunds class to reference it.
startExternalTransaction(): This method takes return_id in order to get information about the items in the purchase, and sends this information to an external payment processor. The validation of credit/debit cards and the rest of the payment procedure is not handled by this method, but by the external service.

Development Plan/Timeline:

Initial Planning (Weeks 1-2):

- Communicate with store staff to set expectations
- Identify what methods will take the longest to create
- Divide team into groups

Database Relations (Weeks 3-5):

- Ensure the integrity of external and internal databases
- Create a group to maintain the internal databases

Class Creation/Implementation (Weeks 6-10):

- Create the EmployeeInfo class, and ensure security.
- Test and implement the InventoryView, InventoryEdit, Purchases, and Refunds classes

Method Testing/Implementation (Weeks 11-15):

- Perform tests for individual methods to verify functionality.
- Make sure inter-class operations perform as expected.

UI/App Creation (Weeks 16-20):

- Design an employee-centric interface
- Make input fields, connected menus, and graphical elements to make the software simple to use

System Testing (Weeks 21-23):

- Integrate the existing classes and logic with the new user interface.
- Create test cases for the system that include not only common workflows, but edge cases as well.

Documentation (Week 24):

- Write material that can be used by the store to train their employees

Maintenance (Ongoing):

- Communicate with the store as needed, in order to address bugs or workflow improvements.

System Test Plan

Test Set 1: EmployeeInfo

The unit testing will consist of trying the getEmployeeID and setEmployeeID functions. If the system works correctly, setEmployeeID should correctly change the value of employee_id, and the getEmployeeID should return the intended employee_id.

The Functional/Integration testing will consist of using the viewTransactionDatabase administrator function. Should the system work correctly, the method will allow only the administrator to view the Transaction database. If the method works without being an administrator, the system is integrated fine but unit testing should be reintroduced to ensure the method only works with the administrator.

The system test will consist of using EmployeeInfo to successfully set up a test employee, and then using functions from InventoryEdit, InventoryView, Refunds and Purchases to ensure that the employee has access to all the necessary classes.

Test Set 2: InventoryView

Examples of unit tests for the InventoryView class would be those that verify the functionality of the methods that belong only to InventoryView.

For example, you could test getItemID(), and make sure it returns the correct item_ID.
You could also test getItemColor(), and verify that it returns the correct color.

Examples of integration tests for the InventoryView class could be ones that use methods from InventoryView and InventoryEdit. To test integration between these classes, we should be able to verify that changes made by InventoryEdit will be reflected in the data retrieved by InventoryView's methods.

This can include using methods like setItemSize and then getItemSize, or setItemQuantity and then getItemQuantity. Using these methods in succession will allow us to see if data integrity is maintained between the two classes.

System tests have the largest scope out of our 3 granularities. These tests focus on using the entire system, and considering the interaction between multiple classes.

One system test that we could perform would be to create our inventory system with multiple items and verify that InventoryView, along with InventoryEdit, can be accessed by an employee to retrieve information for all items.