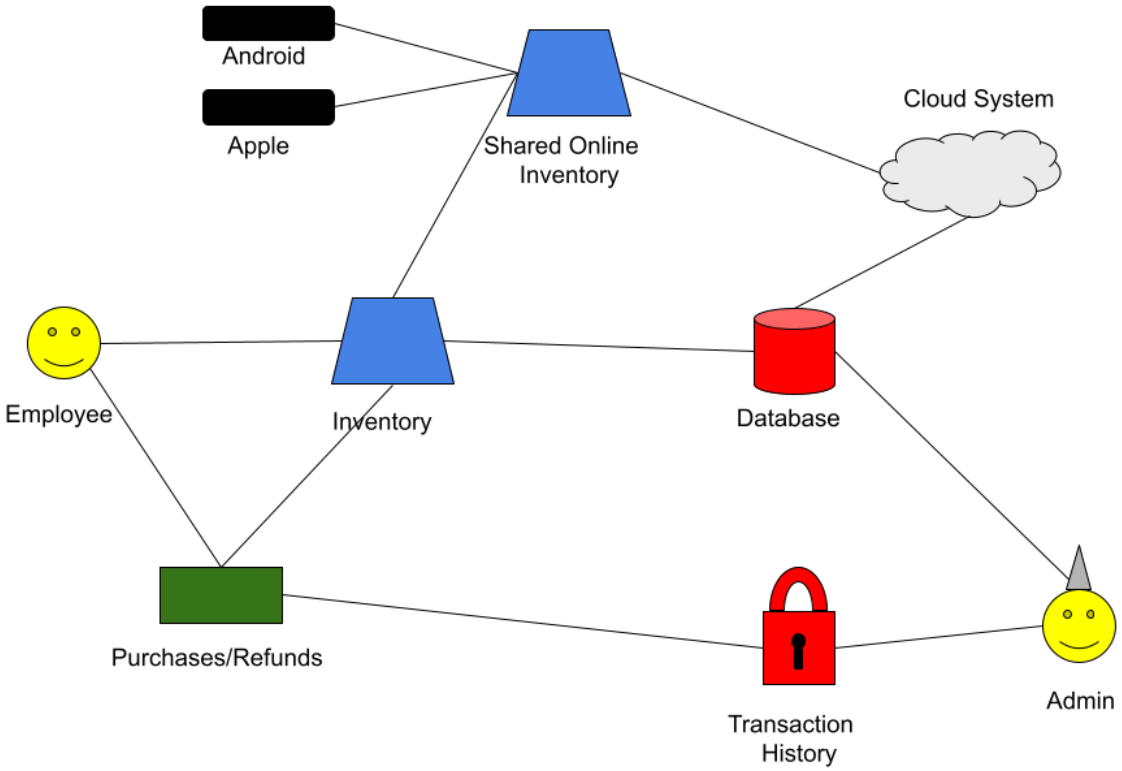
Clothing Point of Sale System

Created by: Nick Ingargiola and Itzel Orozco

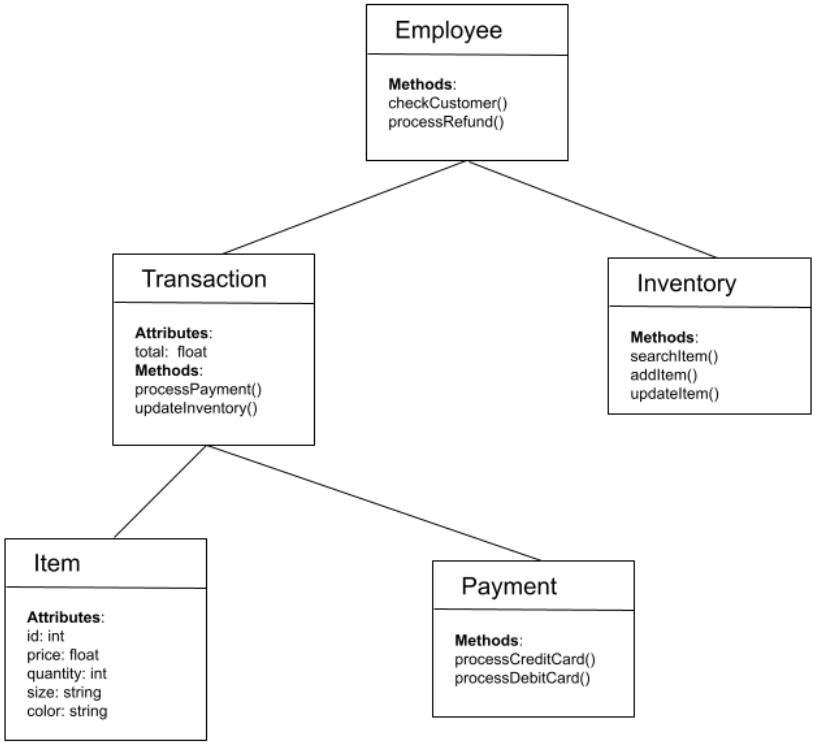
System Description

The Point of Sale System is a strong solution designed to make smooth sale processes and manage inventory in a competent manner. It supports various payment methods, including credit card and debit card to ensure accuracy in transaction totals and inventory updates. Employees can easily process refunds and returns, and the system allows for ideal inventory management, including adding new items and searching by various criteria. This will be compatible with Apple and Android devices, the system guarantees accessibility and flexibility, while data is securely stored in a database with cloud backup. All in all, it empowers employees to deliver exceptional customer service while optimizing store operations.

Software Architecture Overview



Architecture Diagram of all major components



UML Class Diagram

Description of Classes

Employee: This represents the employee at the clothing store. They are responsible for checking out customers(credit, debit, or cash) and processing refunds (cash only). They further are associated with the inventory and are responsible for searching, adding, and ensuring that the inventory is updated after all transactions

Transaction: The transaction class represents all the steps within a transaction. Some of these steps include purchasing and returning. It is associated with the Item, Employee, and Payment classes by calculating the total cost, updating inventory, and processing refunds. The transaction history is secure and can only be accessed by administrators.

Item: This class represents the items which are for sale in the store. Each item is set with unique attributes such as item ID, price, quantity, size, and color. These items are either scanned using a barcode or manually entered by the employee. This class is essential for the sales process and inventory management.

Inventory: This class represents the store’s inventory and provides functionalities to search items(by ID, name, or date added), add new items, and update item attributes. The inventory's transaction history and sales numbers are stored within a database, which is backed up in a cloud system and synchronized across different stores.

Payment: This class will handle the processing of credit or debit card transactions. It ensures that the payments are processed securely and efficiently.

Description of Attributes

Transaction:

total: This attribute is used to calculate the total amount for the item including sales tax.

Item:

id: This attribute is used to provide an item ID number which is unique to each specific item

price: This attribute is used to set a price for each item

quantity: This attribute specifies how many of each item is left in stock

size: This attribute is used to mark what size each item is

color: This attribute is used to represent what color every item is

Description of Operations

Employee:

checkCustomer(): This method is used by the employee to check out the customer

processRefund(): This method is used by the employee to process a refund for a customer

Transaction:

processPayment(): This method is used to process a payment by a customer

updateInventory(): This method is used to update the inventory of the items that are purchased

Inventory:

searchItem(): This method is used to search an item in the inventory

addItem(): This method is used to add an item to the inventory

updateItem(): This method is used to update an item in the inventory

Payment:

processCreditCard: This method is used to process a purchase made by credit card

processDebitCard: This method is used to process a purchase made by debit card

Development Plan and Timeline

1 → 2 → 3 → 4 → 5

1 PLANNING & REQUIREMENTS

Duration: 2 weeks

TASKS:

- Define project scope, objectives, and success criteria.
- Gather requirements from stakeholders, including store managers and employees.
- Conduct market research and competitor analysis.

Team Responsibilities:

- Project Manager: Oversees planning phase and ensures alignment with stakeholder expectations.
- Business Analyst: Facilitates requirements gathering sessions and documents business needs.

2 SYSTEM DESIGN

Duration: 3 weeks

TASKS:

- Design system architecture, including frontend, backend, and database components.
- Create wireframes and UI/UX designs for the point of sale interface.
- Define data models for inventory management and transaction processing.

Team Responsibilities:

- System Architect: Leads the design phase, defines system components and interactions.
- UI/UX Designer: Designs user interfaces and prototypes based on gathered requirements and industry standards.

3 DEVELOPMENT

Duration: 6 weeks

TASKS:

- Implement frontend interfaces for sales transactions, inventory management, and employee functions.
- Develop backend systems for transaction processing, inventory updates, and database management.
- Integrate external payment processors for credit card and debit card transactions.

Team Responsibilities:

- Frontend Developers: Implement user interfaces and barcode scanning functionality.
- Backend Developers: Develop backend systems and integrate with external services.

4 TESTING

Duration: 4 weeks

TASKS:

- Conduct unit testing for frontend and backend components.
- Perform integration testing to ensure seamless communication between modules.
- Implement user acceptance testing with store employees to validate system functionality.

Team Responsibilities:

- Quality Assurance Engineer: Develops test plans, conducts testing, and reports bugs to the development team for resolution.
- Frontend and Backend Developers: Collaborate with QA engineer to address and fix identified issues.

5 LAUNCH

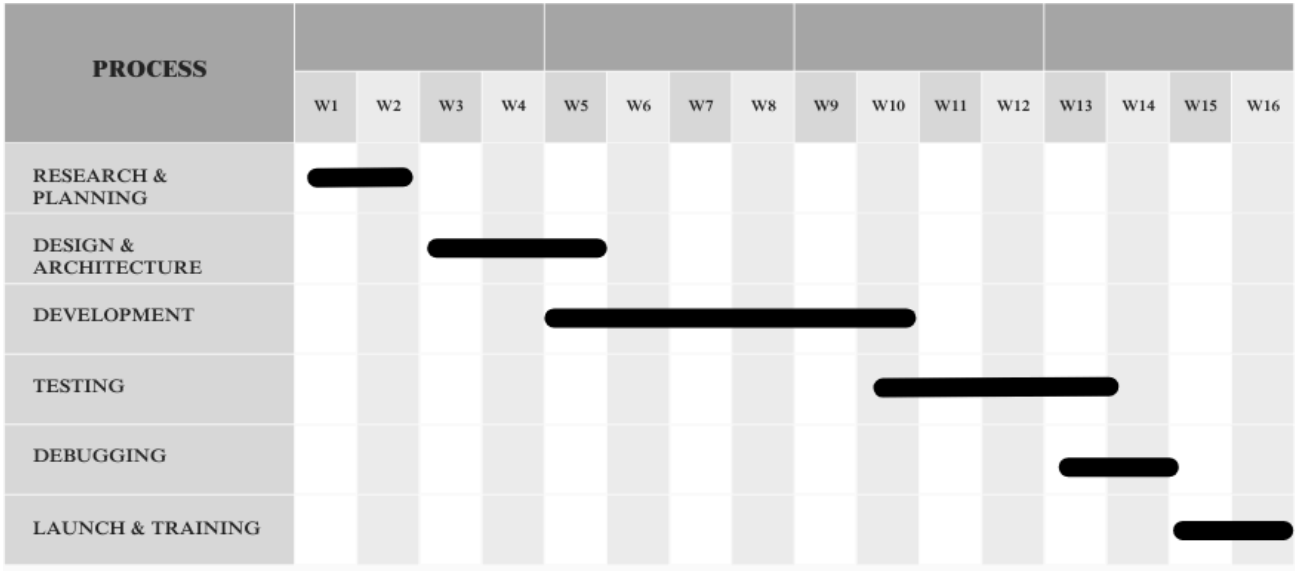
Duration: 2 weeks

TASKS:

- Deploy the system to production environment.
- Provide training sessions for store employees on system usage, including sales transactions, inventory management, and refund processing.
- Establish backup and recovery procedures for data stored in the centralized database.

Team Responsibilities:

- Deployment and Training Specialist: Manages deployment process, conducts training sessions, and ensures data backup procedures are in place.



Verification Test Plan

1. Unit Tests

- Target Feature: Frontend User Interface (UI) Components

- Test Set 1: UI Component Rendering
    a. Test Description: This test verifies that UI components render correctly and display the expected content.
    b. Test Vector:
        - Open the application on different devices (phones, tablets).
        - Navigate to different screens (sales transaction, inventory management, employee functions).
        - Verify that all UI components (buttons, input fields, labels) render correctly and are positioned as designed.
    c. Coverage: This test covers the rendering functionality of UI components, ensuring consistency across different devices and screens.

- Test Set 2: Input Validation
    a. Test Description: This test validates user input in UI forms to ensure data integrity and prevent errors.
    b. Test Vectors:
        - Input invalid data (e.g., alphanumeric characters in numeric fields, negative quantities).
        - Submit the form and observe error messages or validation prompts.
    c. Coverage: This test ensures that the system properly validates user input, preventing erroneous data entry and maintaining data integrity.

2. Integration Tests

- Target Feature: Payment Processing Integration

- Test Set 1: Credit Card Payment Integration
    a. Test Description: This test verifies the integration with external payment processors for credit card transactions.
    b. Test Vector:
        - Initiate a test transaction using a valid credit card.
        - Verify that the transaction is processed successfully and the payment is reflected in the system.
    c. Coverage: This test ensures that the system correctly communicates with external payment processors and accurately processes credit card payments.

- Test Set 2: Inventory Management Integration
    a. Test Description: This test validates the integration between inventory management functionality and database operations.
    b. Test Vector:
        - Add a new item to the inventory.
        - Verify that the item is correctly stored in the database and reflects in the inventory view.
    c. Coverage: This test ensures that inventory management operations are properly synchronized with the database, maintaining data consistency.

3. System Tests
- Target Feature: End-to-End Sales Transaction Workflow

- Test Set 1: Purchase Transaction
    a. Test Description: This test evaluates the entire sales transaction workflow from item selection to payment processing.
    b. Test Vector:
        - Select items for purchase.
        - Proceed to checkout and choose payment method (credit card, cash).
        - Complete the transaction and verify that inventory is updated, and the transaction is logged correctly.
    c. Coverage: This test ensures that the system handles the complete sales transaction process accurately and efficiently.

- Test Set 2: Refund Process
    a. Test Description: This test assesses the refund process for returned items.
    b. Test Vector:
        - Initiate a refund for a previously purchased item.
        - Verify that the inventory is updated accordingly, and the refund transaction is logged correctly.
    c. Coverage: This test ensures that the system correctly handles refunds, updating inventory and transaction records accurately.

Data Management Strategy

We will utilize an SQL database system to streamline data management and maintain consistency across all operations. The data is organized into several key categories; Employees, Transactions, Items, and Inventory. This allows for clear roles and responsibility within the database and simplifies maintenance. Each category is created for the optimization of its functionality. Using SQL created a balanced tradeoff between maintaining data and managing complexity making it

the best option for the system which requires accuracy and reliability. We could have chosen NoSQL but it would complicate data consistency and increase the amount of synchronization across services due to the different microservices provided.

Why?	● SQL, like PostgreSQL, will help with organization and keep data neat and easy to find. It'll keep data secured, accurate, and overall suitable to handle both sales and inventory. ● We will use a single database rather than multiple for the simple fact of having everything together and allow for easier data access, maintenance, and backup procedures. ● Several tables in our database will be for inventory items, payment information, transactions, etc.
Alternatives?	● NoSQL databases, such as MongoDB, are good at handling data; however, they may require additional effort in data consistency and transactional integrity.
Tradeoffs?	● Using a single database for our Sale System simplifies data management and system administration, but could limit scalability in the long run. On the other hand, switching to multiple databases could be complicated but offers better scalability.