Preface

Project Name: MitchCloud

Project Description: a POS system to help simplify and streamline sales of Mitchell’s Homemade Ice Cream products to the customers, clerks and managers.

Project Sponsor: Mitchell’s

Project Author: Jessica Gehring

Project Submission date: Dec. 29th, 2017

Project Unexpected Events: Possibility of log in failure and events logging failure

Project Lessons Learned: associating classes with other classes and logging to a file

Project Performance: Project performs as expected, runs with few errors

Contents

Requirements

- POS system will help create a faster and smoother checkout process. It will resolve payment and product choices along with being able to log in.
- System Name: MitchCloud

Artifact Name: vision.txt

Create Date: Nov 10, 2017

Author: Jessica Gehring

Version: 1.0

Introduction:

- Creating a cloud based system (MitchCloud) via website, point-of-sale (POS) or an app, that will be flexible and be able to support varying customer orders,
- different payment types and also would support the existing and the future expansions of the system. They system will be able to to keep track of inventory,

- descriptions of the products itself along with having transactions with third party systems.

Positioning:

- Business Opportunity:
    - Current POS product is not able to take customer's buisness regarding the companies available products and is unable to keep track of inventory and third party systems.
    - The existing POS product also does not scale well as buisness increase, can only work online but never offline, or during off hours.
    - The POS system that is current does not work well with customers, managers, clerks or third party systems.
    - Current POS only allows for mobile PDAs and laptops, but has not integrated any new technologies.
    - Market is inflexable due to the limited functionalities of the current POS

Problem Statement:

- Current POS system only limits clients to certain products to order, causing a lack of sales in edible products.
    - effects the clients, managers and clerks.

Stakeholders:

- Customer: wants icecream, cakes, special requests and fst service in a short amount of time.
- Clerk: wants accurate, no errors, and quick processes for payment and orders
- Company: wants to accurately record transactions, inventory, and satisfy customers.

Summary of System Features:

- System will give fast service to customers and clerks. Keep track of sales, inventory, the different payments customers used.
- Authentification of users, handles offline orders and will have transactions with third party systems.

Product Overview:

- Will provide Customers with fast service.
- Will provide Customers the ability to filter through products through keywords suck as "vegan", "no sugar added".
- Will allow the Customers to view the products price.
- Customer will be allowed to view the products ingredients, sources, pictures and calories.
- Customer will have many payment options, card, cash and more.
- Customer will have the option to choose many flavors along with toppings and to be able to adjust their purchases at chekcout

- Will provide Clerk with fast transactions, and product filters

Functional Requirements Specification

- Stake holders
  - Clerk
  - Managers
  - Sponsors
  - Customers
- Actors and Goals
  - Clerk will log in and take customers orders and check out customer's orders in a reasonable amount of time
  - Customers have the ability to log in and place orders and checkout in a reasonable amount of time
- Use Cases
  - Description:
    - Each use case defines the uses of each aspect of the system.
  - Fully-Dressed Description:
    - Use case checkout plays an important role, Clerk should have no errors with payment transactions and customers should have fast payment options.
    - Use case managing users plays an important role. The user must input their username and password in order to place an order, or for clerk to create and checkout a customer.
    - Use case managing products plays an important role. Keeps track of products as the customer or clerk makes a purchase or order

Use Cases

- usecase_01_process_sale.txt:

  System Name: MitchCloud

  Artifact Name: usecase_01_process_sale.txt

  Create Date: Nov 21, 2017

  Author: Jessica Gehring

  Version: 1.0

  Use Case UC1: Process Sale

  Primary Actor: Clerk

  Stakeholders and Interests:

  - Clerk wants efficient and fast orders with no payment errors, want correct shipping information, and to be able to view and filter through products/Customer orders.

  - Customer wants to have fast service and fast payment by using either credit, cash or debit. Would like to be able to filter and view products easily along with a proof of purchase (electronic or paper reciept).

- Local Manager wants to be able to manage customer orders by creating, reading, updating or deleting a Customers order. Want to create, read, update or delete discounts, and to keep an accurate summary of sales from that day.
- System Admin wants to handle any errors or failures with ease with payments.
- Sales Analysts want to generate a report of of the sales which entails a Customers Zip code, Address and more.
- Payment Authorization Service wants to receive authorization regarding the proper account for their sellee.
- Tax Service want to collect the tax from every sale the clerk makes.

Main Success Scenarion:

1. Customer goes to check out with the products they want to purchase.
2. The Clerk then procedes to start a new sale.
3. The Clerk scans the bar code or enters the UPC to identify the product.
4. The POS system records the item and will present what the product is, a description, price and the running total.

The Clerk repeats steps 1 through 4 until the process is done.

5. The system then procedes to present the final total, the price of the products all together along with the calculated tax.
6. The Clerk then procedes to tell the Customer the total and asks for the type of payment they will use along with shipping information.
7. The Customer pays and supplies the necessary shipping information to the Clerk. The system then handles the payment efficiently.
8. The system logs the complete sale including the payment information, and shipping information to the Accounting system, Shipping Management system.
9. The system logs the products sold so the Local Manager can take the inventory.
10. The system then either prints out a receipt or sends a receipt electronically to the customer.
11. Customer then leaves with a receipt and whethere or not they decided to purchase any products.

- Usecase_01_process_sale_v2.txt:

  System Name: MitchCloud
  Artifact Name: usecase_01_process_sale.txt
  Create Date: Nov 22, 2017
  Author: Jessica Gehring
  Version: 2.0

  Use Case UC1: Process Sale
  Scope: POS Application
  Level: SubFunction
  Primary Actor: Clerk
  Stakeholders and Interests:

- Clerk wants efficient and fast orders with no payment errors, want correct shipping information, and to be able to view and filter through products/Customer orders.
- Customer wants to have fast service and fast payment by using either credit, cash or debit. Would like to be able to filter and view products easily along with a proof of purchase (electronic or paper reciept).
- Local Manager wants to be able to manage customer orders by creating, reading, updating or deleting a Customers order. Want to create, read, update or delete discounts, and to keep an accurate summary of sales from that day.
- System Admin wants to handle any errors or failures with ease with payments.
- Sales Analysts want to generate a report of of the sales which entails a Customers Zip code, Address and more.
- Payment Authorization Service wants to receive authorization regarding the proper account for their sellee.
- Tax Service want to collect the tax from every sale the clerk makes.

Preconditions: Cashier and customer are identified and authenticated
PostConditions: Sale is logged, tax correctly calculate, final total correctly calculated. Inventory up-to-date and sales are recorded. The receipt is created and payment authorization approves.
Main Success Scenario:
1. Customer goes to check out with the products they want to purchase.
2. The Clerk then procedes to start a new sale.
3. The Clerk scans the bar code or enters the UPC to identify the product.
4. The POS system records the item and will present what the product is, a description, price and the running total.

The Clerk repeats steps 1 through 4 until the process is done.

5. The system then procedes to present the final total, the price of the products all together along with the calculated tax.
6. The Clerk then procedes to tell the Customer the total and asks for the type of payment they will use along with shipping information.
7. The Customer pays and supplies the necessary shipping information to the Clerk. The system then handles the payment efficiently.
8. The system logs the complete sale including the payment information, and shipping information to the Accounting system, Shipping Management system.
9. The system logs the products sold so the Local Manager can take the inventory.
10. The system then either prints out a receipt or sends a receipt electronically to the customer.
11. Customer then leaves with a receipt and whether or not they decided to purchase any products.

Extensions:
*a. Manager is able to override and operation
    1. The system will enter manager authorization mode
    2. A manager or cashier can perform one operation.
    3. system can revert back to Cashier-authorization

*b. When the system fails

1. cashier restarts the system , logs in, customer logs in, and recovers the lost state
2. System reconstructs to before the sale
    2a. System can detect a failure
        1. system notifies the cashier of an error. Records the error and begins to clean the system
        2. the cashier starts new sale

1a. Manager or a Customer choose to resume a sale
1. cashier resumes the operation, enters ID to retrieve the sale.
2. Customer enters ID to retrieve information
3. sale is retrieved, the system displays the items and the total
    2a. sale was not found
        1. system notifies cashier
        2. cashier starts new sale and re-enters customer items

2a. Invalid item ID
1. system signals error and rejects the item entry
2. casier responds to error
    2a. Human-readable item ID
        1. cashier reads the ID and phsyically enters
        2. system displays the item description and cost

3a. Invalid Payment Type
1. system signals error and rejects the Payment
2. cashier repsonds to error
    2a. Payment is Credit Card
        1. cashier manually enters card number into system
            1a. system issues an error. Cashier tries alternate method
    2b. Cashier asks for different type of payment
        1. cashier asks manager to override an operation
        2. manager performs override
        3. cashier indicates new payment type

Special Requirements:
- UI on a flat monitor. Text visible from a distance
- Credit authorization response within a short amount of time
- Quick recovery when system fails
- Language selection
- Pluggable business rules

Technology and Data Variations List:
*a. manager override by swiping a card through a reader or use of a passcode via screen
*2a. item identifier through bar code scanner
*3a. Payment read through a card reader or through keyboard

Frequency or Occurrence: continuous

Open Issues:
- tax law variations

- service recovery issue
- customization for different businesses
- must the cashier send the log after they clock out
- customers access to paying

- Usecase_02_checkout.txt:

  System Name: MitchCloud
  Artifact Name: usecase_02_checkout.txt
  Create Date: Nov 22, 2017
  Author: Jessica Gehring
  Version: 1.0

  Use Case UC2: Checkout
  Primary Actor: Clerk
  Stakeholders and Interests:
  - Clerk wants no errors or failures with payment transactions and for transactions to be fast and efficient. Also want to have correct shipping information without any errors.
  - Customer wants fast payment transactions with the choice between different payments like cash, credit or debit. Want the option to have stuff shipped quickly and properly.
  - Local Manager wants to be able to update, read, create or delete a users order before, during, or after checkout.
  - Cloud Manager wants no errors in payment transaction, no errors in shipping information and no problems with user authentification.
  - System Admin wants to make sure that the Customer is properly authentificated in order to use the POS system.
  - Sales Analysts want the correct payment and shipping information from the customer at the time of checkout in order to generate a report.

  Main Success Scenario:
  1. Customer is authenticated and is able to place an order.
  2. Customer proceeds to checkout.
  3. Clerk asks for the Customers payment and shipping information.
  4. Customer gives the Clerk their payment and shipping info and also any other useful information for checkout.
  5. Once Checkout is complete, the system prints a reciept or sends the customer a receipt electronically.

- Usecase_02_checkout_v2.txt:

  System Name: MitchCloud
  Artifact Name: usecase_02_checkout_v2.txt
  Create Date: Nov 22, 2017
  Author: Jessica Gehring

Version: 2.0

Use Case UC2: Checkout
Scope: POS application
Level: subfunction
Primary Actor: Clerk
Stakeholders and Interests:
- Clerk wants no errors or failures with payment transactions and for transactions to be fast and efficient. Also want to have correct shipping information without any errors.
- Customer wants fast payment transactions with the choice between different payments like cash, credit or debit. Want the option to have stuff shipped quickly and properly.
- Local Manager wants to be able to update, read, create or delete a users order before, during, or after checkout.
- Cloud Manager wants no errors in payment transaction, no errors in shipping information and no problems with user authentification.
- System Admin wants to make sure that the Customer is properly authentificated in order to use the POS system.
- Sales Analysts want the correct payment and shipping information from the customer at the time of checkout in order to generate a report.
Preconditions: Products and Payment Type identified
Postconditions: Sale successfully completed with no errors
Main Success Scenario:
1. Customer is authenticated and is able to place an order.
2. Customer adds items to their cart
3. Customer proceeds to checkout.
4. Clerk asks for the Customers payment and shipping information.
5. Customer gives the Clerk their payment and shipping info and also any other useful information for checkout.
6. Once Checkout is complete, the system prints a reciept or sends the customer a receipt electronically.
Extenstion:
*a. Manager requests and override operation:
    1. System enters manager-authorization mode
    2. manager or cashier performs the manager-mode operation
    3. system changes back to cashier-authorization mode
1a. Customer Id cannot be authenticated
    1. system signals error
    2. cashier handles the error
        2a. cashier manually enters Customer authorization
        2b. system displays user information
            1. user unable to be authenticated through manual entry;
system signals error. Cashier tries alternate method
2a. Credit Card cannot be authorized

1. system signals error
2. cashier handles error
    2a. cashier asks for other method of payment, overrides current payment method
    2b. customer supplies second payment method
        1. system signals error
        2. manager performs override
        3. cashier asks for another payment method
    2c. If payment method unauthorized enter manually

3a. Customer hands too much cash
1. system notifies cashier
2. cashier handles the problem

Special Requirements:
- sound and visual representation of error
- payment authorization quick

Technology and Data Variations List:
*a. Coding scheme for ID's
*b. Credit card information entered via card reader or keyboard
*c system for correct cash tendered
Frequency of Occurrence: continuous

- Usecase_03_profile_management.txt:

System Name: MitchCloud
Artifact Name: usecase_03_profile_management.txt
Create Date: Nov 22, 2017
Author: Jessica Gehring
Version: 1.0

Use Case UC3: Profile Management
Primary Actor: Clerk
Stakeholders and Interests:
- Clerk wants to be able to view information of customers, view previous orders, to view the hours that they are/have/will be working, and also to view the bankaccount information of customers.
- Customer wants to be able to view personal information and view their previous orders and to see their banking information and also their shipping information incase it needs to be updated.
- Local Manager wants to keep track of the customers orders, wants the ability to create, read, update and delete orders from customers. Also wants the ability to keep track of the clerk and the amount of sales they had.
- Cloud Manager wants to store customer orders, customer banking information and shipping for future use. Want to keep track of clerks hours and sales.

- System Admin wants to keep track of customer and clerk authorization to make sure the POS is being properly used.
- Sales Analyst needs to have the correct payment and shipping information so they can put together a report.
- Payment Authorization Service wants no errors in payment information and transactions.
- Shipping System wants no errors with the Customers shipping information so products can be delivered quickly and on time.
- Delivery and Shipping Operations Manager also wants no error with the customers shipping information so that the products they ordered can be delivered quickly and on time.

Main Success Scenario:
1. Customer checks out and gives the clerk their payment information, shipping information and their order.
2. Clerk takes the customers information and proceeds to store it in the system.
3. System then keeps the customers payment/banking information, shipping information and also the orders.
4. Clerk then is able to go and review the customers information in the system along with any previous orders.

- Usecase_03_profile_management_v2.txt:

  System Name: MitchCloud
  Artifact Name: usecase_03_profile_management.txt
  Create Date: Nov 22, 2017
  Author: Jessica Gehring
  Version: 2.0

  Use Case UC3: Profile Management
  Scope: POS Application
  Level: subfucntion
  Primary Actor: Clerk
  Stakeholders and Interests:
  - Clerk wants to be able to view information of customers, view previous orders, to view the hours that they are/have/will be working, and also to view the bankaccount information of customers.
  - Customer wants to be able to view personal information and view their previous orders and to see their banking information and also their shipping information incase it needs to be updated.
  - Local Manager wants to keep track of the customers orders, wants the ability to create, read, update and delete orders from customers. Also wants the ability to keep track of the clerk and the amount of sales they had.
  - Cloud Manager wants to store customer orders, customer banking information and shipping for future use. Want to keep track of clerks hours and sales.

- System Admin wants to keep track of customer and clerk authorization to make sure the POS is being properly used.
- Sales Analyst needs to have the correct payment and shipping information so they can put together a report.
- Payment Authorization Service wants no errors in payment information and transactions.
- Shipping System wants no errors with the Customers shipping information so products can be delivered quickly and on time.
- Delivery and Shipping Operations Manager also wants no error with the customers shipping information so that the products they ordered can be delivered quickly and on time.

Preconditions: Track user information
Postconditions: Log and store user information successfully
Main Success Scenario:
1. Customer checks out and gives the clerk their payment information, shipping information and their order.
2. Clerk takes the customers information and proceeds to store it in the system.
3. System then keeps the customers payment/banking information, shipping information and also the orders.
4. Clerk then is able to go and review the customers information in the system along with any previous orders.

Extension:
a* system fails
    1. cashier restarts the system, logs in and recovers data that has been lost
    2. system reconstructs
        2a. system detects an error preventing recovery
        2b. cashier starts a new log

1a. Customer gives clerk incorrect information
    1. system sends an error
    2. cashier handles the error manually
        2a. cashier asks for customer to reenter information correctly
        2b. customer enters their information correctly
        2c. system is still unable to identify customer and their information

2a. system does not log the customers information
    1. system fails
    2. cashier handles the failure
        2a. cashier reboots system and recovers data
        2b. cashier resends current sale to system to log

Special Requirements:
- customer information logging be quick and efficient
- neat organized sales logs

Technology and Data variations LIst:
- Log to properly keep track of sales
- system to store customers information to be used in future sales

- quick sales
Frequency of Ocurrence: nearly continuous
Open Issues:
- wrong information
- recovery issues
- keeping log of multiple sales at once

- Usecase_04_managing_inventory.txt:

  System Name: MitchCloud
  Artifact Name: usecase_04_managing_inventory.txt
  Create Date: Nov 22, 2017
  Author: Jessica Gehring
  Version: 1.0

  Use Case UC4: Managing Inventory
  Primary Actor: Local Manager
  Stakeholders and Interests:
  - Local Manager wants to have the ability to keep track of expiration dates of the products being sold. Want the ability to create a summary of sales and to also keep track of the products that have been made.
  - Clerk wants the ability to be able to view previous sales and orders and products that haven't been sold.
  - Customer wants the ability to see their previous orders and current orders and the products that are currently available to be purchased. And also to be able to view product descriptions, and the ability to sort through subcategories of products.
  - Cloud Manager wants correct information regarding Inventory and no errors.
  - Raw and Goods Suppliers want correct inventory reports in order to determine the amount of new supplies needed.
  Main Success Scenarion:
  1. Clerk makes a sale.
  2. The sale is then stored with previous sales in the system.
  3. Local Manager takes the sales of the day and begins to make a sales summary.
  4. After the sales the Local Manager checks the inventory to see how much is remaining and the expiration dates of the products.
  5. Local Manager stores the result of the check to the system.
  The Local Manager repeats 1 through 5 until the system indicates it is done.
  4. The system then presents the Local Manager's summary of the sales from that day and the Inventory report.
  5. The Local Manager then can send the Summary of sales and Inventory to the Sales Analysts to use to generate reports.

- Usecase_04_managing_inventory_v2.txt:

  System Name: MitchCloud

Artifact Name: usecase_04_managing_inventory.txt
Create Date: NOv 22, 2017
Author: Jessica Gehring
Version: 2.0

Use Case UC4: Managing Inventory
Scope: POS Application
Level: subfunction
Primary Actor: Local Manager
Stakeholders and Interests:
- Local Manager wants to have the ability to keep track of expiration dates of the products being sold. Want the ability to create a summary of sales and to also keep track of the products that have been made.
- Clerk wants the ability to be able to view previous sales and orders and products that haven't been sold.
- Customer wants the ability to see their previous orders and current orders and the products that are currently available to be purchased. And also to be able to view product descriptions, and the ability to sort through subcategories of products.
- Cloud Manager wants correct information regarding Inventory and no errors.
- Raw and Goods Suppliers want correct inventory reports in order to determine the amount of new supplies needed.
Preconditions: Log invnetory
Postconditions: Correclty logged inventory after sales
Main Success Scenarion:
1. Clerk makes a sale.
2. The sale is then stored with previous sales in the system.
3. Local Manager takes the sales of the day and begins to make a sales summary.
4. After the sales the Local Manager checks the inventory to see how much is remaining and the expiration dates of the products.
5. Local Manager stores the result of the check to the system.
The Local Manager repeats 1 through 5 until the system indicates it is done.
4. The system then presents the Local Manager's summary of the sales from that day and the Inventory report.
5. The Local Manager then can send the Summary of sales and Inventory to the Sales Analysts to use to generate reports.
Extension:
*a. system fails to log sales correctly
    1. system sends error message to cashier and manager
    2. cashier handles all sales logs manually
1a. Inventory was incorrectly logged in the system
    1. system sends messasge stating inventory is not correct
    2. manager and cashier handle the error
        2a. manager overrides system, reviews past sales and logs inventory
        2b. system sends error due to unreconizable item and quanity

1. manager then finds alternate fix

Special Requirements:
- Sales log easily readable
- sales log neatly logged
- sales logged right after sale is through

Technology and Data Variations List:
- manager override inventory system
- system to log sales and transactions

Frequency of Occurrence: not continuous

Open Issues:
- Logging system is accessed by cashier or manager or both

- Usecase_05_managing_orders.txts:

  System Name: MitchCloud
  Artifact Name: usecase_05_managing_orders.txt
  Create Date: Nov 23, 2017
  Author: Jessica Gehring
  Version: 1.0

  Use Case UC5: Managing Orders
  Primary Actor: Local Manager
  Stakeholders and Interests:
  - Local Manager wants the ability to create, read, update or delete any existing orders from customers, either from the past or the present without having or causing any problems or having any issues.
  - Clerk wants to be able to use the information from past or present orders in order to create a fast and reliable service to the customer.
  - Customer wants to be able to have fast and efficient order taking with out any problems or errors.
  - Cloud Manager wants to be able to view users orders so it can be stored into the system for future use.
  - Sales Analysts want to be able to use the information from the customers order to create a sales report.

  Main Success Scenario:
  1. Customer proceeds to order.
  2. Local Manager creates the customers order and reads the products that the customer has selected and submits the information into the system.
  3. Customer then proceeds to change one of the products they want to order and removes one product from their order.
  4. Local Manager then follows suit and updates the order by changing the product and deleting the other product from the user and then proceeds to update the system.
  Steps 1 through 4 are then repeated until the customer is ready to checkout.
  5. Customer proceeds to checkout with no errors occuring during order taking.

- Usecase_05_managing_orders_v2.txt:

System Name: MitchCloud
Artifact Name: usecase_05_managing_orders.txt
Create Date: NOv 23, 2017
Author: Jessica Gehring
Version: 2.0

Use Case UC5: Managing Orders
Scope: POS Application
Level: subfunction
Primary Actor: Local Manager
Stakeholders and Interests:
- Local Manager wants the ability to create, read, update or delete any existing orders from customers, either from the past or the present without having or causing any problems or having any issues.
- Clerk wants to be able to use the information from past or present orders in order to create a fast and reliable service to the customer.
- Customer wants to be able to have fast and efficient order taking with out any problems or errors.
- Cloud Manager wants to be able to view users orders so it can be stored into the system for future use.
- Sales Analysts want to be able to use the information from the customers order to create a sales report.

Preconditions: Obtain orders
Postconditions: Obtained correct and compeleted orders
- Sales Analysts want to be able to use the information from the customers order to create a sales report.

Main Success Scenario:
1. Customer proceeds to order.
2. Local Manager creates the customers order and reads the products that the customer has selected and submits the information into the system.
3. Customer then proceeds to change one of the products they want to order and removes one product from their order.
4. Local Manager then follows suit and updates the order by changing the product and deleting the other product from the user and then proceeds to update the system.

Steps 1 through 4 are then repeated until the customer is ready to checkout.

5. Customer proceeds to checkout with no errors occuring during order taking.

Extenstions:
*a. System fails to create a new order for customer, sends error message to manager
    1. manager handles error
    2. manager manually creates an order for the customer
        2a. system does not allow the new order to be created
        2b. manager then proceeds to reboot the system

1. manager logs in and restores the order

1a. Customer is unable to change their order

   1. manager goes to override the operation
   2. manager manually changes customers order

2a. System fails with payment transaction

   1. error is sent to the manager
   2. manager handles the error

      2a. manager manually handles the payment transaction

Special Requirements:
- fast order creating and changing
- direct communication with manager

Technology and Data Variations List:
- manager override for order creations, deletions and changes

Frequency of Occurrence: nearly continuous

Open Issues:
- who can handle order changes directly
- who can create an order directly

- Usecase_06_managing_user.txt:

  System Name: MitchCloud
  Artifact Name: usecase_06_managing_users.txt
  Create Date: Nov 23, 2017
  Author: Jessica Gehring
  Version: 1.0

  Use Case UC6: Managing users
  Primary Actor: Local Manager
  Stakeholders and Interests:
  - Local Manager wants the ability to create, read, update or delete clerk and customer information from the system.
  - Clerk wants to be able to obtain customer orders quickly without any errors.
  - Customer wants to be able to add, delete or change their order fast and without any problems.
  - Local Cloud wants a correct representation of the customers orders in order to keep it in the system.
  - System Admin wants the ability to keep track of user authorization.
  - Sales Analysts want to keep track of user information each time a sale is made.

  Main Success Scenario:
  1. Clerk clocks in to work.
  2. Local Manager Updates the system to say that the clerk clocked in at this time and day.
  3. Customer logs in and is authenticated.
  4. Customer begins to order, first chooses chocolate icecream.

5. Local Manager then creates a new order in the system and reads that the customer has ordered chocolate icecream and adds it to the system.
6. Customer then proceeds to add mint chocolate chip icecream to their order and gets rid of the chocolate icecream.
7. Local Manager reads the changes to the customers updated order and proceeds to update the system.
8. Customer then proceeds to checkout.

- Usecase_06_managing_users_v2.txt:

System Name: MitchCloud
Artifact Name: usecase_06_managing_users.txt
Create Date: NOv 23, 2017
Author: Jessica Gehring
Version: 2.0

Use Case UC6: Managing users
Scope: POS Application
Level: subfunction
Primary Actor: Local Manager
Stakeholders and Interests:
- Local Manager wants the ability to create, read, update or delete clerk and customer information from the system.
- Clerk wants to be able to obtain customer orders quickly without any errors.
- Customer wants to be able to add, delete or change their order fast and without any problems.
- Local Cloud wants a correct representation of the customers orders in order to keep it in the system.
- System Admin wants the ability to keep track of user authorization.
- Sales Analysts want to keep track of user information each time a sale is made.

Preconditions: Authenticate and log users names and passwords
Postconditions: Correctly authenticate users and log names
Main Success Scenario:
1. Clerk clocks in to work.
2. Local Manager Updates the system to say that the clerk clocked in at this time and day.
3. Customer logs in and is authenticated.
4. Customer begins to order, first chooses chocolate icecream.
5. Local Manager then creates a new order in the system and reads that the customer has ordered chocolate icecream and adds it to the system.
6. Customer then proceeds to add mint chocolate chip icecream to their order and gets rid of the chocolate icecream.
7. Local Manager reads the changes to the customers updated order and proceeds to update the system.

8. Customer then proceeds to checkout.
Extenstions:
*a. System fails to log a clerk or customer in
    1. system sends failure message to clerk or customer.
    2. clerk or customer handles the error message
        2a. clerk/customer reneter their ID and password
        2b. system is unable to authorize them
            1. manager is then contacted to authenticate users
1a. system fails to log user name and password
    1. manager then reboots the system
    2. manager retrieves login information
        2a. system does not properly log login information
            1. manager finds alternate way of logging login information
                1a. manager logs logins manually
2a. User incorrectly types user name or password
    1. system sends error message to user
    2. user then handles the error
        2a. error manually enters their user name or password again
        2b. system does not recognize password or name
            1. user contacts manager for alternate way of authorization
Special Requirements:
- quick response time for username and password
- clear visual of user name and password
Technology and Data Variations List:
- system for logging users in by an ID and password
- system to log all logins, time, date, name
Frequency of Occurrence: nearly continuous
Open Issues:
- who can view the user names and passwords log
- who can handle user name and password failures
- who can create user names and passwords

- Usecase_07_managing_products.txt:

System Name: MitchCloud
Artifact Name: usecase_07_managing_products.txt
Create Date: Nov 23, 2017
Author: Jessica Gehring
Version: 1.0

Use Case UC7: Managing Products
Primary Actor: Cloud Manager
Stakeholders and Interests:

- Cloud Manager wants the ability to create, read, update or delete products, prices of the products and wants to deal with clerks, local managers and customers in the system.
- Local Manager wants to access the prices and products quickly without any failures or errors.
- Clerk Wants the ability to view and filter through the products quickly by finding prices or using a keyword.
- Customer wants the ability to filter throught the products by price, specialty items, flavors and so forth.
- Raw and Goods Suppliers want to know what products will be needed and what will no longer be needed.

Main Success Scenario:

1. The Christmas season is over and there is no need to have the seasonal product anymore.
2. The Cloud Manager then creates a discount for the seasonal product and discontinues the product.
3. The Cloud Manager then updates the clerks, customers and local managers system of the new update to the seasonal product.
4. The Cloud Manager then updates the system to show there is a discount and that the product will be discontinued.

- Usecase_07_managing_products_v2.txt:

  System Name: MitchCloud
  Artifact Name: usecase_07_managing_products.txt
  Create Date: NOv 23, 2017
  Author: Jessica Gehring
  Version: 2.0

  Use Case UC7: Managing Products
  Scope: POS Application
  Level: subfunction
  Primary Actor: Cloud Manager
  Stakeholders and Interests:
  - Cloud Manager wants the ability to create, read, update or delete products, prices of the products and wants to deal with clerks, local managers and customers in the system.
  - Local Manager wants to access the prices and products quickly without any failures or errors.
  - Clerk Wants the ability to view and filter through the products quickly by finding prices or using a keyword.
  - Customer wants the ability to filter throught the products by price, specialty items, flavors and so forth.
  - Raw and Goods Suppliers want to know what products will be needed and what will no longer be needed.

  Preconditions: Create, update delete products

Postconditions: successfully created and updated products prices, descriptions, availability and discounts
Main Success Scenario:
1. The Christmas season is over and there is no need to have the seasonal product anymore.
2. The Cloud Manager then creates a discount for the seasonal product and discontinues the product.
3. The Cloud Manager then updates the clerks, customers and local managers system of the new update to the seasonal product.
4. The Cloud Manager then updates the system to show there is a discount and that the product will be discontinued.
Extenstions:
*a. system fails to remove discount
    1. system sends failure message to manager
    2. manager handles the error
        2a. manager reboots the system
        2b. manager starts updating discounts and products again
Special Requirements:
- change discounts
- a clear screen to display products
Technology and Data Variations List:
- managers ability to override discounts and products
Frequency of Occurrence: slightly continuous
Open Issues:
- who gets to create new products
- who gets to create a discount
- who gets to delete products
- who gets to adjust the prices of products

- Usecase_08_managing_discounts.txt:

System Name: MitchCloud
Artifact Name: usecase_08_managing_discounts.txt
Create Date: Nov 23, 2017
Author: Jessica Gehring
Version: 1.0

Use Case UC8: Managing Discounts
Primary Actor: Cloud Manager
Stakeholders and Interests:
- Cloud Manager wants to be able to create discounts for products that will be discontinued, for veterans, seasonal discounts and custome discounts.
- Customer wants to have discounts for special occasions, student discounts etc.
- Local Manager wants to be able to access discounts so it can be used on products.

- Clerk wants to be able to access discounts easily in order to quickly apply the discount to the customers order.
Main Success Scenario:
1. Its the Christmas season.
2. Cloud Manager goes into the system to create a discount for certain products.
3. The Cloud Manager has no errors or failures with creating the discount.
4. The Cloud Manager then updates the system with the newly created discount for clerks and customers to view.
5. At the time of viewing and filtering products, the customers and clerks see there is a discount on a certain product.
6. A Customer chooses a product with the discount.
7. Customer then proceeds to check out.
8. The Clerk then proceeds to process the customers order and applies the discount.
9. Clerk asks for customers payment type and shipping information.
10. Customer gives the clerk accurate payment and shipping information.
11. The customers payment was authorized.
12. The customer then receives a receipt with the total and the discount given.

- Usecase_08_managing_discounts_v2.txt:
System Name: MitchCloud
Artifact Name: usecase_08_managing_discounts.txt
Create Date: NOv 23, 2017
Author: Jessica Gehring
Version: 2.0

Use Case UC8: Managing Discounts
Scope: POS Application
Level: subfunction
Primary Actor: Cloud Manager
Stakeholders and Interests:
- Cloud Manager wants to be able to create discounts for products that will be discontinued, for veterans, seasonal discounts and custome discounts.
- Customer wants to have discounts for special occasions, student discounts etc.
- Local Manager wants to be able to access discounts so it can be used on products.
- Clerk wants to be able to access discounts easily in order to quickly apply the discount to the customers order.
Preconditions: create discounts for products
Postconditions: Successfully created discounts for seasonal and off season products, and discounts for veterans, students
Main Success Scenario:
1. Its the Christmas season.
2. Cloud Manager goes into the system to create a discount for certain products.
3. The Cloud Manager has no errors or failures with creating the discount.

4. The Cloud Manager then updates the system with the newly created discount for clerks and customers to view.
5. At the time of viewing and filtering products, the customers and clerks see there is a discount on a certain product.
6. A Customer chooses a product with the discount.
7. Customer then proceeds to check out.
8. The Clerk then proceeds to process the customers order and applies the discount.
9. Clerk asks for customers payment type and shipping information.
10. Customer gives the clerk accurate payment and shipping information.
11. The customers payment was authorized.
12. The customer then receives a receipt with the total and the discount given.
Extenstions:
*a. system fails to apply a discount to a product at time of checkout
    1. error message is sent to cashier
    2. cashier has manager override the error
        2a. manager overrides the product to apply discount
1a. manager cannont apply discount
    1. manager reboots the system
    2. manager enters ID to get list of products
        2a. manager proceeds to enter discounts for selected products
        2b. manager updates the system, the system does not successfully update
            1. system sends error
            2. manager finds alternate ways to fix system discounts
Special Requirements:
- clear display of discounts
- clear display of products that require discount
- functioning system for discounts
Technology and Data Variations List:
- system for creating discounts
Frequency of Occurrence: slighlty continuous
Open Issues:
- who is allowed to create discounts for products

- Usecase_09_managing_sales_reports.txt:
  System Name: MitchCloud
  Artifact Name: usecase_09_creating_sales_reports.txt
  Create Date: Nov 23, 2017
  Author: Jessica Gehring
  Version: 1.0

  Use Case UC9: Creating Sales Reports
  Primary Actor: Sales Analysts

Stakeholders and Interests:
- Sales Analysts want the ability to obtain customer information easily without any problems to generate a sales report. The report would include zip codes, addresses, time period and more.
- Local and Cloud Managers want to keep track of customer information.
- Clerk wants the ability to view the sales and keep track of customer information.
Main Success Scenario:
1. Customer goes to checkout.
2. The clerk processes a sale.
3. A new sale has occured.
4. The sale then is logged into the system.
5. The local manager then takes that sale from the system and starts to create a sales summary.
6. Local Manager finishes the Sales summary and saves it in the system.
One through seven repeat until there are no more sales occuring.
7. Sales Analysts then take the information from the sales summary in the system and logs all the information to create a Sales Report from the day.
8. The Sales Analysts then take the sales report and saves it to the system.

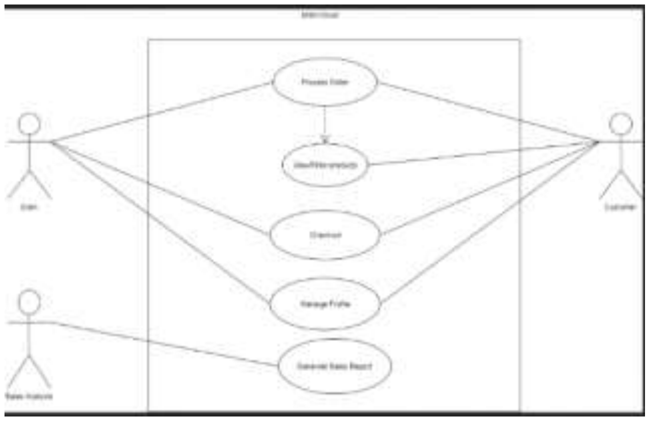
- Usecase_09_creating_sales_reports_v2.txt:

System Name: MitchCloud
Artifact Name: usecase_09_creating_sales_reports_v2.txt
Create Date: NOv 23, 2017
Author: Jessica Gehring
Version: 2.0

Use Case UC9: Creating Sales Reports
Scope: POS Application
Level: subfunction
Primary Actor: Sales Analysts
Stakeholders and Interests:
- Sales Analysts want the ability to obtain customer information easily without any problems to generate a sales report. The report would include zip codes, addresses, time period and more.
- Local and Cloud Managers want to keep track of customer information.
- Clerk wants the ability to view the sales and keep track of customer information.
Preconditions: obtain customer information through logs
Postconditions: Customer information is correctly sent through logs, showing all transactions and products purchased
Main Success Scenario:
1. Customer goes to checkout.
2. The clerk processes a sale.
3. A new sale has occured.

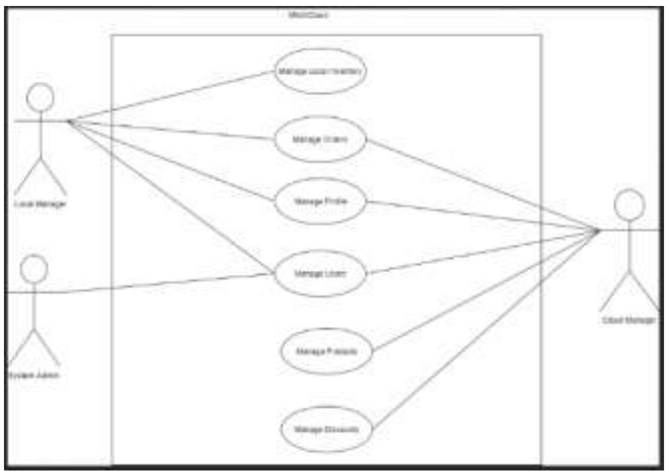
4. The sale then is logged into the system.
5. The local manager then takes that sale from the system and starts to create a sales summary.
6. Local Manager finishes the Sales summary and saves it in the system.
One through seven repeat until there are no more sales occuring.
7. Sales Analysts then take the information from the sales summary in the system and logs all the information to create a Sales Report from the day.
8. The Sales Analysts then take the sales report and saves it to the system.
Extenstions:
*a. system fails to send the log to the sales anaylst
    1. sends error
    2. sales analyst requests to have the log resent
        2a. log is resent but system sends error once again
        2b. system is set to reboot
            1. sales manager then requests again to send the log
Special Requirements:
- clear visualization of customer information
- neat logs
- visualization of past sales and transactions
- logs sent in a fast time period
Technology and Data Variations List:
- system that will send logs
- system to keep track of information and sales
Frequency of Occurrence: slighlty continuous
Open Issues:
- who can send the logs
- who can view the logs

Use Case Diagrams



UseCase Diagram 1

UseCase Diagram 2



Actors and Goals

[Actors]	[Goals]
1. Clerk	- Process orders - Checkout order, involves handling payments and shipping - Deal with payment transactions, errors or failures - View filter products - Manage personal information - Manage previous orders - Manage working hours - Manager bank account information
2. Local Manager	- Manage sales summaries - Keep track of expiration dates by taking inventory - Keep track of the current inventory and products that are readymade - Create, read, update or delete existing orders in the POS system - Create, read, update or delete a clerk or customer in the POS system - Edit and new business rules - Put customer information into the system - Manage payment transactions
3. Cloud Manager	- Create, read, update and delete inventory that is in the system - Create, read, update and delete customer, clerk, local manager information - Create, read, update and delete product prices, product information and product discounts
4. System Admin	- Process different payments - Deal with system start up - Deal with system shutdown - Deal with errors or failures in the system - Deal with requests and responses from managers, clearks and customers - Manager the users information in the system
5. Customer	- Deal with payment options - View and order products - Customer authentification - Deal with shipping information - Manage their personal information
6. Sales Analyst	- Manage inventory reports

	- Create sales reports through the data collected in the system

System Events

- Se_checkOut_02.txt:

  System Name: MitchCloud
  Artifact Name: se_checkout_02.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. getTotal
  2. getNumberItemsOrdered
  3. displaySale
  4. makePayment

- Se_creating_sales_reports_09.txt:

  System Name: MitchCloud
  Artifact Name: se_creating_sales_report_09.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. openTransactionLog
  2. closeTransactionLog

- Se_managing_discounts_08.txt:

  System Name: MitchCloud
  Artifact Name: se_managing_discounts_08.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. displayItem
  2. getDescription
  3. getCost

- Se_managing_inventory_04.txt:

  System Name: MitchCloud
  Artifact Name: se_managing_inventory_04.txt
  Create Date: Nov 21, 2017

Author: Jessica Gehring
Version: 1.0

1. getUnits

- se_managing_orders_05.txt:
  System Name: MitchCloud
  Artifact Name: se_managing_orders_05.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. startSale
  2. addSaleItem
  3. createSaleItem

- se_managing_products_07.txt:
  System Name: MitchCloud
  Artifact Name: se_managing_products_07.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. getCost
  2. getDescription
  3. getUnits
  4. subTotal
  5. getTotal

- se_managing_users_06.txt:
  System Name: MitchCloud
  Artifact Name: se_managing_users_06.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. login

- se_process_sale_01.txt:
  System Name: MitchCloud
  Artifact Name: se_process_sale_01.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version 1.0

1. interactiveTransaction
2. displayItem
3. getDescription
3. getSubTotal

- se_profile_management_03.txt:

  System Name: MitchCloud
  Artifact Name: se_profile_management_03.txt
  Create Date: Nov 21, 2017
  Author: Jessica Gehring
  Version: 1.0

  1. checkUsername
  2. checkPassword
  3. getUsername
  4. getPassword

Supplementary Specification

System Name: MitchCloud

Artifact Name: supplementary_specification.txt

Create Date: Nov 20, 2017

Author: Jessica Gehring

Version: 1.0

[Supplementary Specification]

Logging:

- Logs all interactions including errors/failures and stores them into storage
- Logs customer interactions into storage
- Logs clerks interactions and puts them in storage
- Logs the managers interactions and puts them in storage

Security:

- Usage of MitchCloud requires authentification
- Need authentification from the customer and the clerk

Usability:

- Customer, Managers, Clerk will have a clear display of the POS

- The text size and color will be visible to everyone
- Shipping, payments and orders will be fast, easy and error-free
- Along with visual messages, there will be sounds to alert the clerk of an error
- Have the ability to access the POS from multiple devices like, laptops and mobile device

Reliability:

- Recoverability: when there is a failure of external services, use local services
- Performance: To have speedy checkout, order processing and payment transactions so that customers can leave quickly. One problem is having errors/failures with user authentification, orders and payment transactions.

Supportability:

- Adaptability: The POS has to adjust to the customers demands and payment methods
- Configurability: Customers want different platform/networks for this POS system. Be able to use this POS system on mobile devices. The POS system has to be flexible to handle the changing business.

Implementation:

- C++ technology will help improve supportability, reliability and usability along with developing made easier.

Purchased Components:

- Payment Authorization Service: company uses this service to authorize customer payments
- Tax calculator: to suport the different taxes around the country/world
- Accounting System: used to keep track of companies sales and spending
- Shipping Management System: used to manage shipping customers orders to get the products to the customer quickly
- Raw and Goods Suppliers: Company uses this service to supply the ingredients/products to sell and use.
- Chatting Manager: (customer service)
- Delivery and Shipment Operation Manager: company uses to manage shipping and delivery services to assure timely delivery and shipment of products

Free Open Source Components:

- Maximization of C++ use

Interfaces:

- Hardware: user devices, monitors, bar code scanner.
- Software: Inventory system, customer/clerk/manager information system, payment system, tax system, discounts system, Accounting system, shipping system

Domain (Business) Rules:

- Rule 1: User Authentification
- Rule 2: Tax Rule

- Rule 3: Signature for credit card payments
- Rule 4: Discounts (Customer, Manager, Clerk, sales, products)

Legal Issues:

- Tax rules need to be applied to each order at time of check out
- Having to obtain certain software licensing

Domains of Interest:

- Pricing: the original price of products or products with discounts
- Credit and Debit payment handling: credit authorization service has to authorize the customers card
- Sales tax: tax calculations from a third party software. Should take into account tax exemptions.
- Item Identifiers: POS should support UPCs and barcodes to help identify a product

Glossary

- Defines the terms used in the System

[Term]	[Definition and Information]	[Format]	[Validation Rules]	Aliases
Product	Ice cream, cakes, pies and specials that are available			
User Authentication	The validation of a user or the validation of an employee			
Payment Authorization	Validates the payment from an external service, known as the payment authorization service, to guarantee payment to the sellee			
Bar Code	Code, or lines that are used to identify a products price and a brief description of the product			
UPC	Is a numeric code that is used for identifying a	Is a 12 digit code	Digit 12 will be the check digit	Is a universal product code

	products price, and a short description of the product			
Payment	Either credit, debit, cash or phone pay. Supplied by the customer			
Product Price	How much a certain product will cost to buy			
Discounts	Special price mark downs on certain products or by special request or during certain seasons or days			
Inventory	Log of available products for sale			

Business Rules

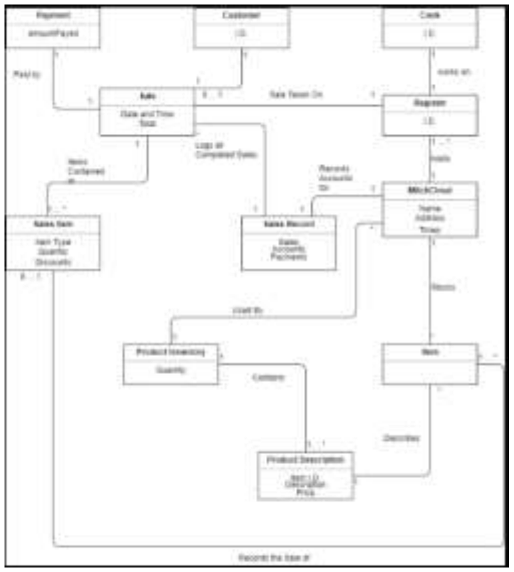
[ID]	[Rule]	[Changeability]	[Source]
Rule 1:	User Authentication	Can change to an online authentication	From Mitchells
Rule 2:	Tax rules. Sales have to have tax applied to them by law	Tax laws can change annually	Law
Rule 3:	Signature for Credit card payments	Can change from physical paper signature to an electronic signature	Required by all credit card companies
Rule 4:	discounts	Discounts can change periodically due to seasons changing, specific days, veterans, students and more	Mitchells

- Are the parts of the system that has the ability to change

Domain Analysis

- Domain Model:

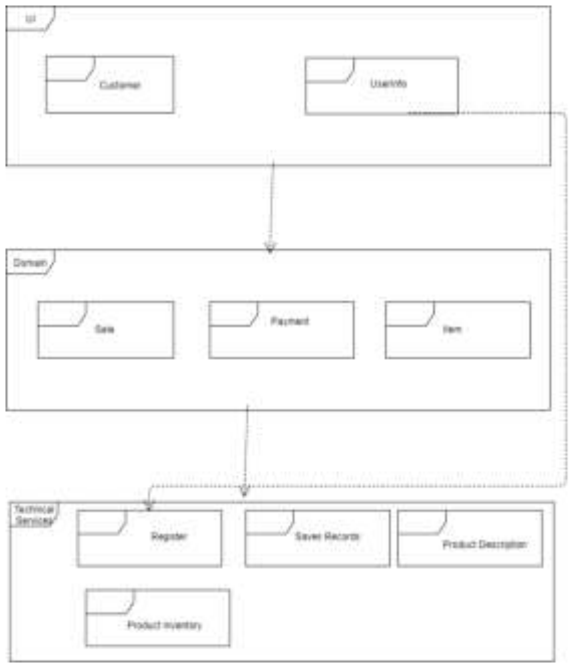
Domain Model 1



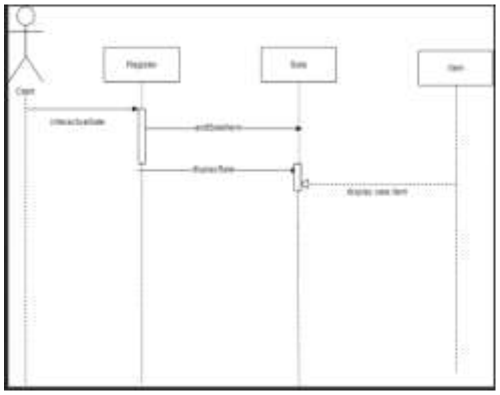
- Concept of the system is to use information supplied by some class into one class for the register to have an efficient and simple POS system
- Classes that are associated with each other are the items, the product inventory, register, userInfo/clerk, sale item and also the sales record

Package Diagram

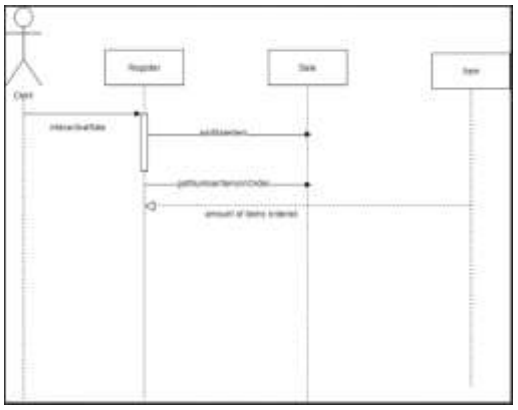
- Shows the layers of the system



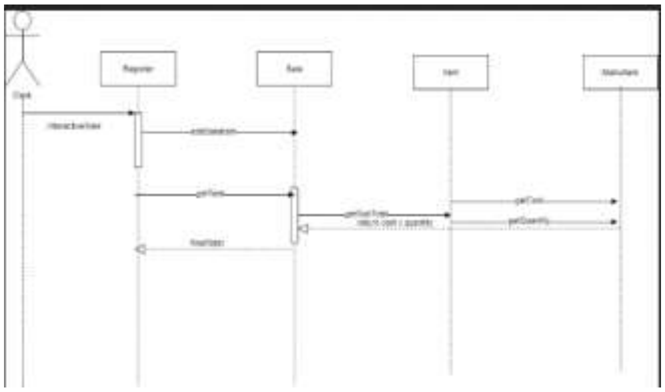
Sequence Diagrams



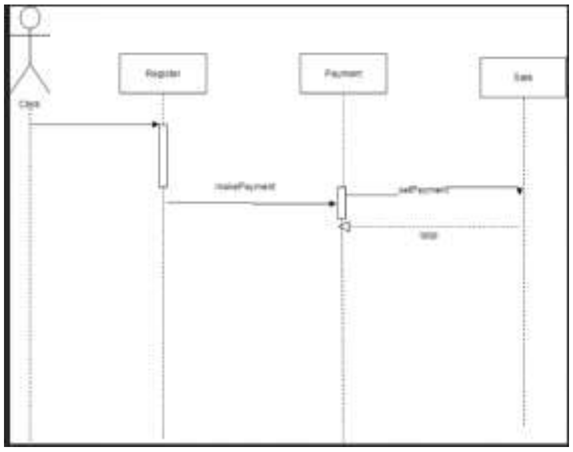
Sequence Diagram 2



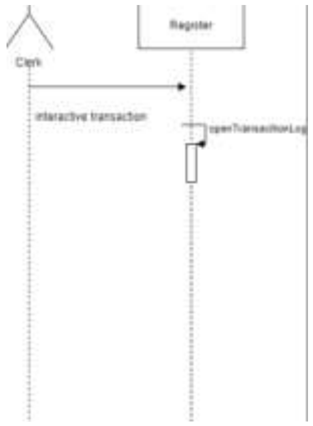
Sequence Diagram 1



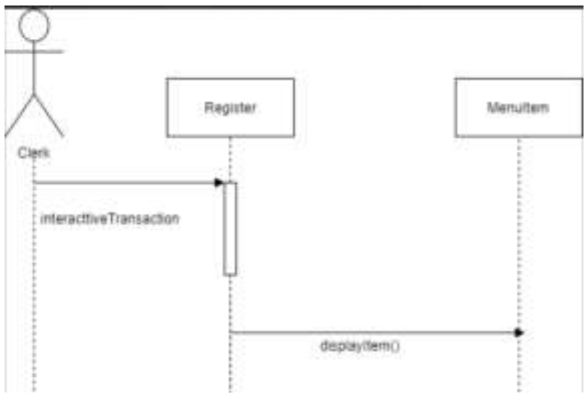
Sequence Diagram 4



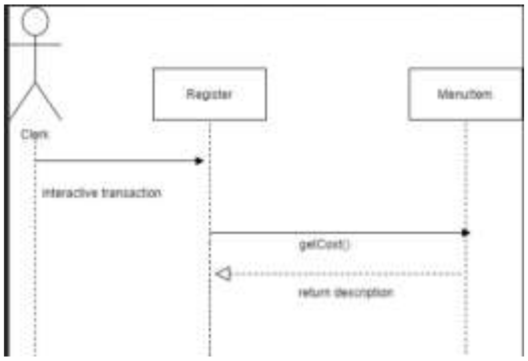
Sequence Diagram 3



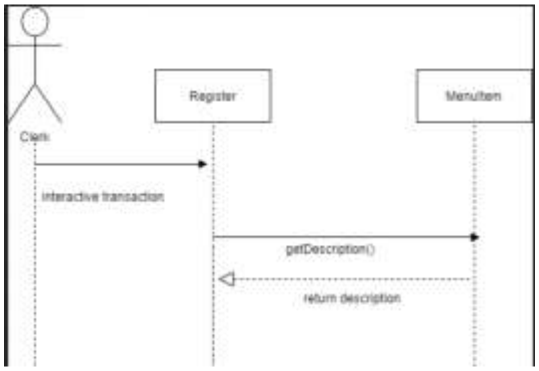
Sequence Diagram 6



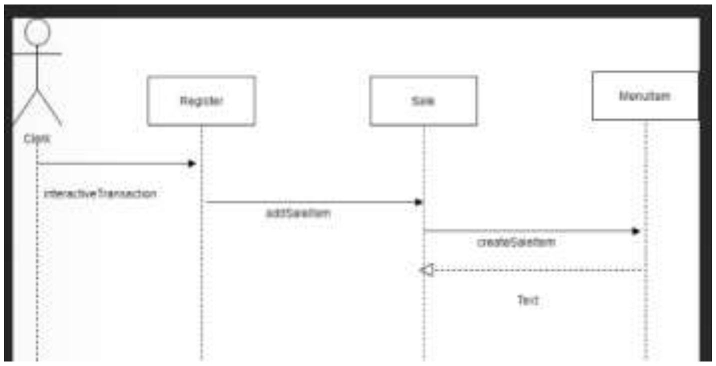
Sequence Diagram 5



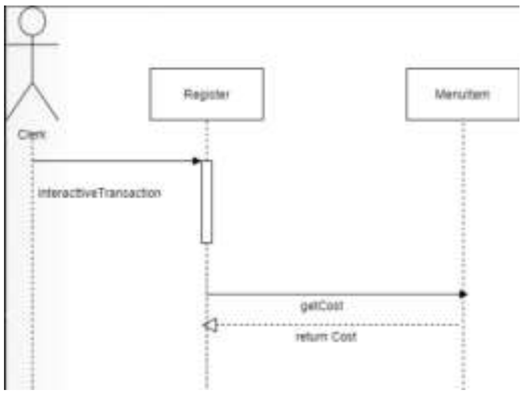
Sequence Diagram 8



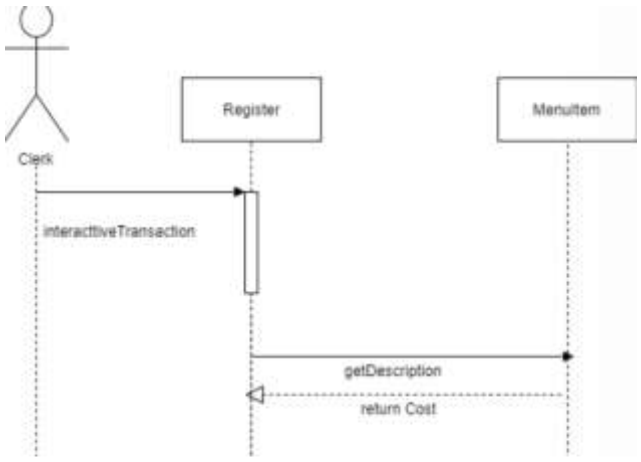
Sequence Diagram 7



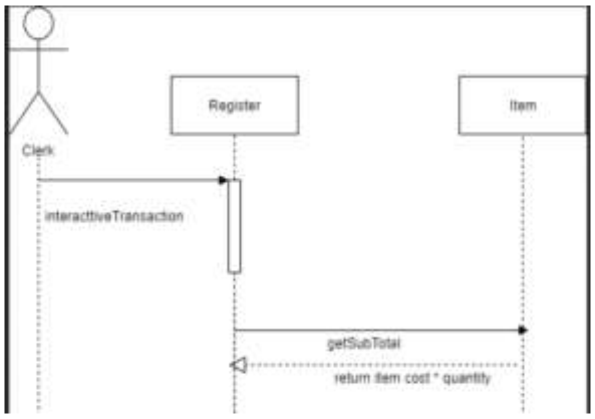
Sequence Diagram 10



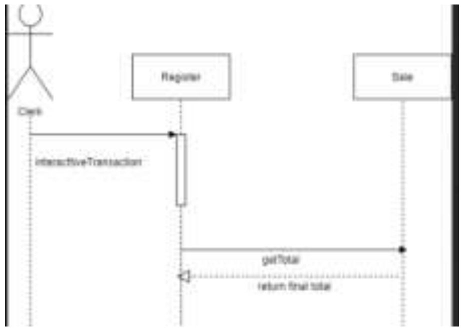
Sequence Diagram 9



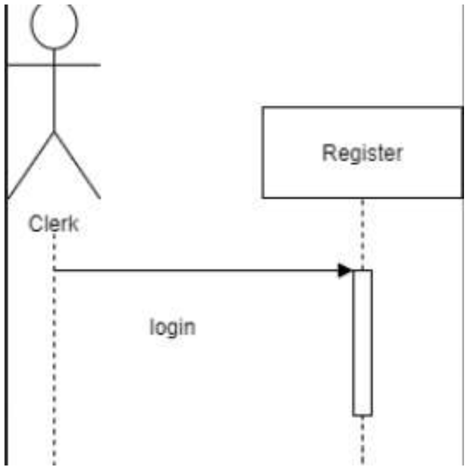
Sequence Diagram 11



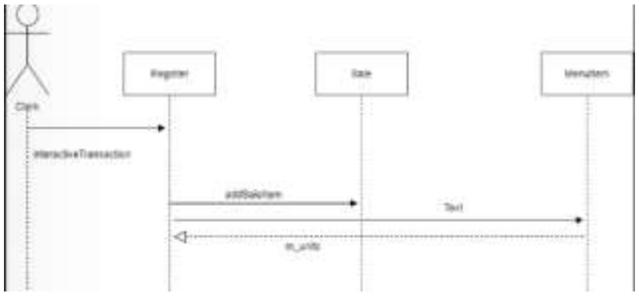
Sequence Diagram 12



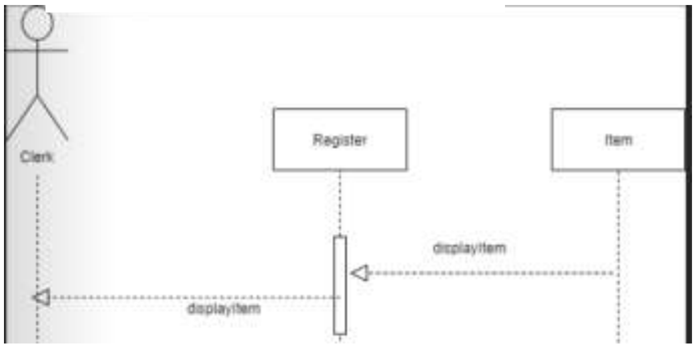
Sequence Diagram 14



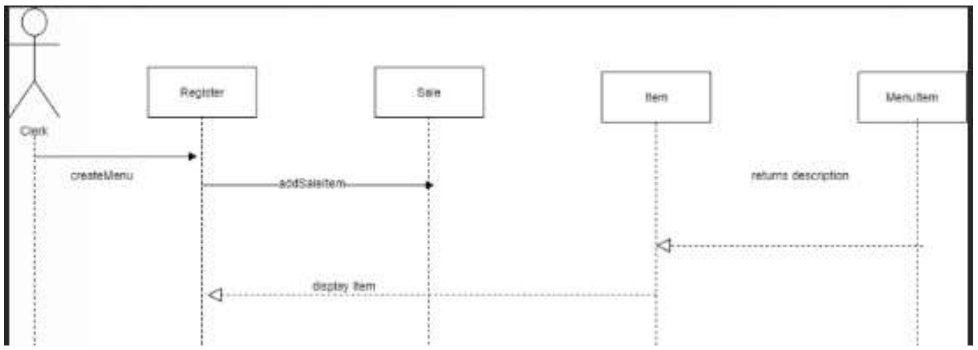
Sequence Diagram 13



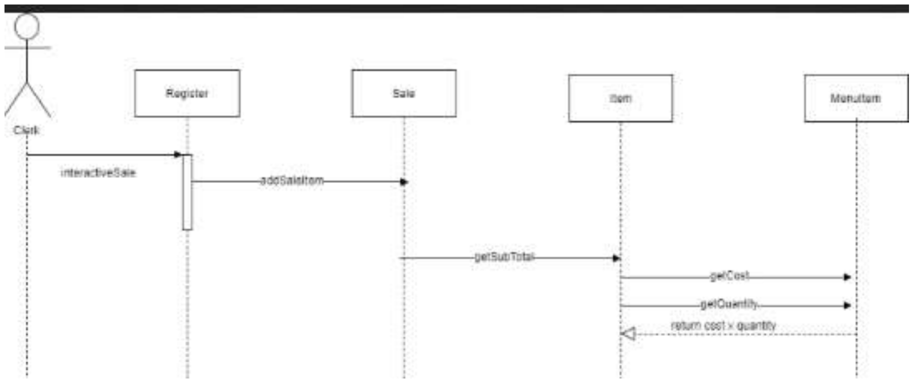
Sequence Diagram 15



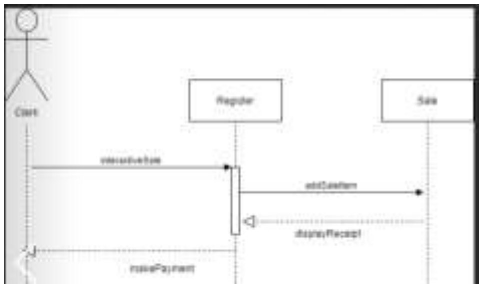
Sequence Diagram 16



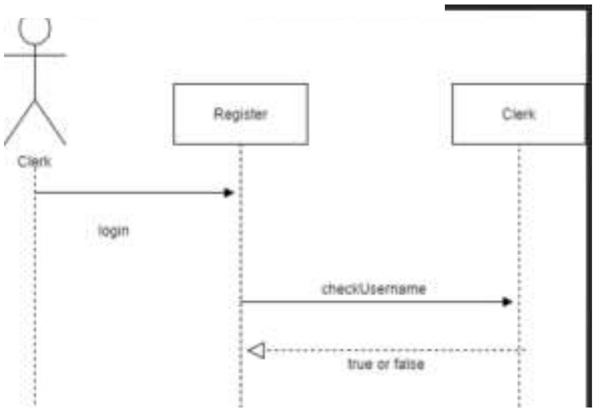
Sequence Diagram 17



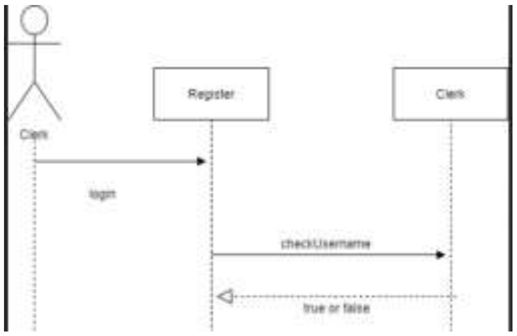
Sequence Diagram 18



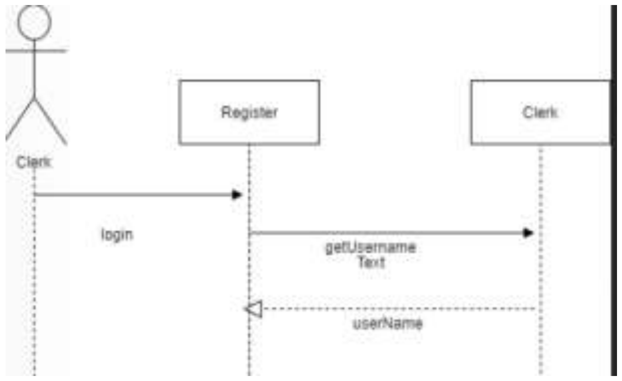
Sequence Diagram 20



Sequence Diagram 19



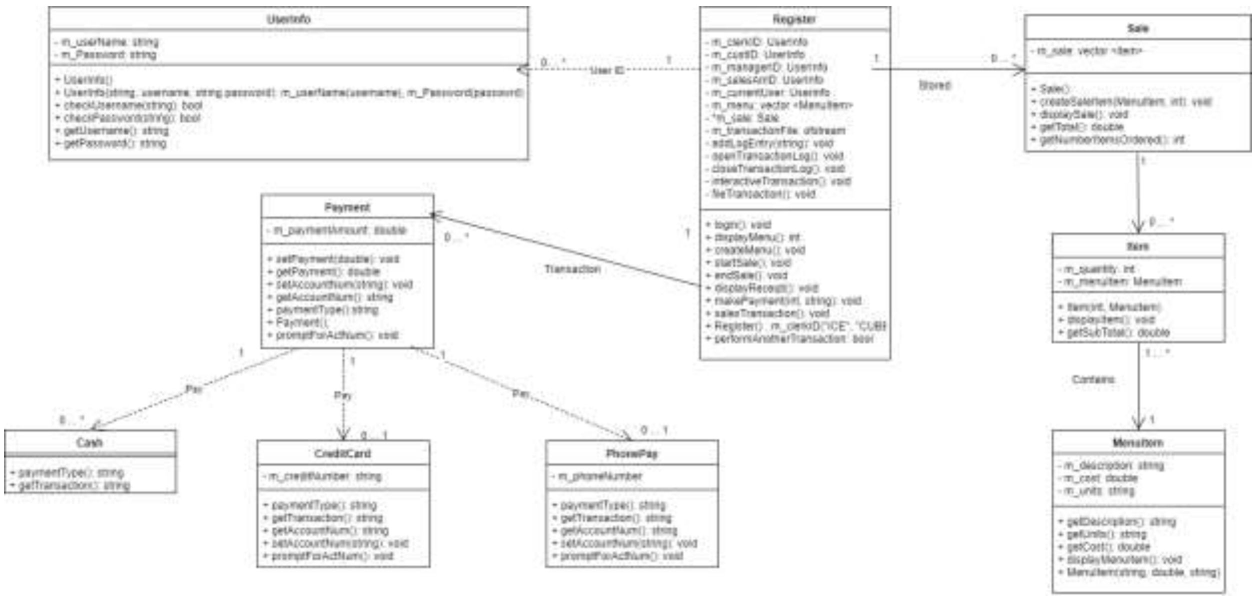
Sequence Diagram 22



Sequence Diagram 21

Class Diagram

- Class diagram breaks down the classes and their member variables and functions


UserInfo
- m_userName: string
- m_Password: string
+ UserInfo()
+ UserInfo(string: username, string password): m_userName(username), m_Password(password)
+ checkUsername(string): bool
+ checkPassword(string): bool
+ getUsername(): string
+ getPassword(): string
0 ... *
User ID
1
Register
- m_clerkID: UserInfo
- m_custID: UserInfo
- m_managerID: UserInfo
- m_salesAmID: UserInfo
- m_currentUser: UserInfo
- m_menu: vector <MenuItem>
- *m_sale: Sale
- m_transactionFile: ofstream
- addLogEntry(string): void
- openTransactionLog(): void
- closeTransactionLog(): void
- interactiveTransaction(): void
- fileTransaction(): void
+ login(): void
+ displayMenu(): int
+ createMenu(): void
+ startSale(): void
+ endSale(): void
+ displayReceipt(): void
+ makePayment(int, string): void
+ salesTransaction(): void
+ Register() : m_clerkID("ICE", "CUBE
+ performAnotherTransaction: bool
1
Stored
0 ... *
Sale
- m_sale: vector <Item>
+ Sale()
+ createSaleItem(MenuItem, int): void
+ displaySale(): void
+ getTotal(): double
+ getNumberItemsOrdered(): int
1
0 ... *
Item
- m_quantity: int
- m_menuItem: MenuItem
+ Item(int, MenuItem)
+ displayItem(): void
+ getSubTotal(): double
1 ... *
Contains
1
MenuItem
- m_description: string
- m_cost: double
- m_units: string
+ getDescription(): string
+ getUnits(): string
+ getCost(): double
+ displayMenuItem(): void
+ MenuItem(string, double, string)
Payment
- m_paymentAmount: double
+ setPayment(double): void
+ getPayment(): double
+ setAccountNum(string): void
+ getAccountNum(): string
+ paymentType() string
+ Payment()
+ promptForActNum(): void
0 ... *
Transaction
1
1
Pay
0 ... *
Cash
+ paymentType(): string
+ getTransaction(): string
1
Pay
0 ... 1
CreditCard
- m_creditNumber: string
+ paymentType(): string
+ getTransaction(): string
+ getAccountNum(): string
+ setAccountNum(string): void
+ promptForActNum(): void
1
Pay
0 ... 1
PhonePay
- m_phoneNumber
+ paymentType(): string
+ getTransaction(): string
+ getAccountNum(): string
+ setAccountNum(string): void
+ promptForActNum(): void

Source Code: cpp files

- Holds the implementation of the classes :

```
#include "stdafx.h"
#include "mitchcloud.h"
#include <iostream>
#include <time.h>


//REGISTER CLASS IMPLEMENTATION//////////////////////////////////////////////////

void Register::login()
{
	string username;
	string password;

	cout << "Please enter your username" << std::endl;
	cin >> username;
	cout << "Please enter your password" << std::endl;
	cin >> password;
	do
	{
		if (m_clerkID.checkUsername(username) == true &&
m_clerkID.checkPassword(password) == true)
		{
			m_currentUser = m_clerkID;
			break;
		}

		 if (m_custID.checkUsername(username) == true &&
m_custID.checkPassword(password) == true)
		{
			 m_currentUser = m_custID;
			 break;
		}
```

```
			if (m_managerID.checkUsername(username) == true &&
	m_managerID.checkPassword(password) == true)
			{
				m_currentUser = m_managerID;
				break;
			}
	if (m_salesAnID.checkUsername(username) == true &&
m_salesAnID.checkPassword(password) == true)
		{
			m_currentUser = m_salesAnID;
			break;
		}
		cout << "Please enter your username" << std::endl;
		cin >> username;
		cout << "Please enter your password" << std::endl;
		cin >> password;

	} while (true);
}

int Register::displayMenu()
{
	int userInput;
	do
	{
		m_transactionFile << "User: " << m_currentUser.getUsername() << " shows the
menu of items to order " << endl;

		cout << "Menu" << endl;
		cout << "====" << endl;

		for (int i = 0; i < m_menu.size(); i++)
		{
			cout << i + 1 << ") ";
			m_menu[i].displayMenuItem();
			cout << endl;
		}
		cout << "Select Product from menu or 0 to end" << endl;
		cin >> userInput;

	} while ((userInput < 0) || (userInput >= m_menu.size()));
	if (userInput == 0)
	{
		m_transactionFile << "User: " << m_currentUser.getUsername() << " selected
to end order " << endl;
	}
	else
	{
		m_transactionFile << "User: " << m_currentUser.getUsername() << " selected
item #  " << userInput << endl;
	}
	return userInput;
}

void Register::createMenu()
{
	m_menu.push_back(MenuItem("Vanilla Bean IceCream", 2.50, "Scoop"));
```

```
        m_menu.push_back(MenuItem("Chocolate IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Butter Pecan IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Dark Roast Coffee IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Coffee Chocolate Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Double Chocolate Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Banana Cream Pie IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Caramel Fudge Brownie IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Caramel Sea Salt IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Cookies & Cream IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Key Lime Pie IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Bing Cherry Chocolate Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Cookie Dough IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Fresh Strawberry IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Rocky Road IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Fresh Mint Choco Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Chocolate Peanut Butter Cup IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Blue Cosmo IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Black Raspberry Choco Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Peanut Butter Choco Chunk IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Pralines & Cream IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Pistachio IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Vanilla Bean Yogurt", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Mango Sorbet", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Raspberry Sorbet", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Vegan Salted Caramel Pecan", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Vegan Chocolate", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("PeanutButter Pretzel Yogurt", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Mint Choco Chunk", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Wildberry Crumble Icecream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Lemon Blackberry Yogurt", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Amaretto Cookie IceCream", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Coffee & Cookies Yogurt", 2.50, "Scoop"));
        m_menu.push_back(MenuItem("Turtle", 26.95, "Pie"));
        m_menu.push_back(MenuItem("Cookies & Cream", 26.95, "Pie"));
        m_menu.push_back(MenuItem("Chocolate Peanut Butter Cup", 26.95, "Cake"));
}
        void Register::displayReceipt()
        {
                m_sale->displaySale();
                m_transactionFile << "Receipt Printed" << endl;
        }

        void Register::startSale()
        {
                m_sale = new Sale;
                openTransactionLog();
                m_transactionFile << "User: " << m_currentUser.getUsername() << " logged
        in" << endl;
                m_transactionFile << "Sale started" << endl;
        }

        void Register::endSale()
        {
                m_transactionFile << "Sale is finished" << endl;
                delete m_sale;
                closeTransactionLog();
        }
```

```
void Register::addSaleItem(int menuSelection, int quantity)
{
	m_transactionFile << "User: " << m_currentUser.getUsername() << " is 
ordering Item # " << m_sale->getNumberItemsInOrder() + 1 << endl;
	m_sale->createSaleItem(m_menu[menuSelection], quantity);

	m_transactionFile << "User: " << m_currentUser.getUsername() << " ordered " 
<< m_menu[menuSelection].getDescription() << endl;
	m_transactionFile << "User: " << m_currentUser.getUsername() << " selected 
the amount " << quantity << " " << m_menu[menuSelection].getUnits() << endl;

	m_transactionFile << "Price is $" << m_menu[menuSelection].getCost() * 
quantity << endl;
	m_transactionFile << "subTotal is $" << m_sale->getTotal() << endl;
}

void Register::makePayment()
{
	int type;
	do
	{
		cout << "Please enter payment type, 1 for cash, 2 for credit and 3 
for phone pay." << endl;
		cin >> type;
	} while (type < 1 || type > 3);

	Payment *payment;
	switch (type)
	{
	case 1:
		payment = new Cash();
		break;
	case 2:
		payment = new CreditCard();
		break;
	case 3:
		payment = new PhonePay();
		break;
	}
	payment->setPayment(m_sale->getTotal());
	payment->promptForActNum();
	cout << "Payment transaction is" << payment->getTransaction() << endl;
	m_transactionFile << "Payment chosen: " << payment->getTransaction() << " 
successful" << endl;
	delete payment;

}

void Register::openTransactionLog()
{

	time_t now;
	const int MAX_FILE_NAME = 128;
	char fileName[MAX_FILE_NAME];

	fileName[0] = '\0';

	now = time(NULL);
	if (now != -1)
	{
		strftime(fileName, MAX_FILE_NAME, "Log_%m_%d_%Y_%H_%M_%S.txt",
gmtime(&now));
	}
	m_transactionFile.open(fileName);
}

void Register::closeTransactionLog()
{
	m_transactionFile.close();
}

void Register::makePayment(int paymentType, string accountNum)
{
	Payment *payment;
	switch(paymentType)
	{
	case 1:
		payment = new Cash();
		break;
	case 2:
		payment = new CreditCard();
		break;
	case 3:
		payment = new PhonePay();
		break;
	}
	payment->setPayment(m_sale->getTotal());
	payment->setAccountNum(accountNum);
	cout << "Payment transaction is" << payment->getTransaction() << endl;
	m_transactionFile << "Payment chosen: " << payment->getTransaction() << "
successful" << endl;
delete payment;
}

void Register::interactiveSale()
{
	m_transactionFile << "User: " << m_currentUser.getUsername() << " selected
Interactive Transaction" << endl;

	int choice;
	int quantity;

	do
	{
		choice = displayMenu();
		if (choice == 0)
		{
			break;
		}
		cout << "Enter quantity of product" << endl;
		cin >> quantity;
		system("cls");
		addSaleItem(choice, quantity);
		displayReceipt();
	} while (choice != 0);
```

```
		makePayment();
}

void Register::salesTransaction()
{
		int input;
		do
		{
			cout << "What type of transaction, File or Interactive? Enter 1 for 
File or 2 for Interactive." << endl;
			cin >> input;

		} while (input > 2 || input < 1);

		if (input == 1)
		{
			fileSale();
		}
		else
		{
			interactiveSale();
		}
}

void Register::fileSale()
{
		m_transactionFile << "User: " << m_currentUser.getUsername() << " selected 
File Transaction" << endl;

		string fileName;
		cout << "Enter file name to input data from" << endl;
		cin >> fileName;

		ifstream inputFile(fileName, ifstream::in);
		if (!inputFile.is_open())
		{
			cout << "Error, file: " << fileName << " unable to open" << endl;
			return;
		}
		int commandNumber;
		int item;
		int quantity;
		int paymentType;
		string accountNumber;

		while (inputFile.good())
		{
			inputFile >> commandNumber;
			switch (commandNumber)
			{
			case 1:
				inputFile >> item >> quantity;
				m_transactionFile << "Read Command 1, selection type, item 
read is " << item << " quantity of item is " << quantity << endl;
				addSaleItem(item, quantity);
				break;
			case 2:
				inputFile >> paymentType;
```

```
                    m_transactionFile << "Read Command 2, selected Payment Type is 
" << paymentType << endl;
                    accountNumber = " ";
                    if (paymentType != 1) //not a cash payment get account number
                    {
                           inputFile >> accountNumber;
                    }
                    makePayment(paymentType, accountNumber);
                    break;

             default:
                    m_transactionFile << "Invalid command read from file. Command 
number read was: " << commandNumber << endl;
             }
       }
       inputFile.close();
}

bool Register::performAnotherTransaction()
{
       string input;
       cout << "Would you like to perform another transaction? [Yes/No]" << endl;
       cin >> input;
       for (int i = 0; i < input.size(); i++)
       {

       input.at(i) = toupper(input.at(i));
}
       if (input == "YES")
       {
             return true;
       }
       else
       {
             return false;
       }
}
//CLERK USERINFO 
IMPLEMENTATION////////////////////////////////////////////////////////
bool UserInfo::checkUsername(string name)
{
       if (name != m_userName)
       {
             return false;
       }
       else
       {
             return true;
       }
}
bool UserInfo::checkPassword(string password)
{
       if (password != m_Password)
       {
             return false;
       }
       else
       {
```

```
                        return true;

                }
        }
//SALE CLASS
IMPLEMENTATION///////////////////////////////////////////////////////////////////////
    void Sale::createSaleItem(MenuItem item, int amount)
    {
            m_sale.push_back(Item(amount, item));
    }

    void Sale::displaySale()
    {
            cout << endl << endl;
            cout << "Reciept" << endl;
            cout << "=======" << endl;
            for (int i = 0; i < m_sale.size(); i++)
            {

                    m_sale[i].displayItem();
            }
            cout << "Final Total is $" << getTotal();
            cout << endl << endl;
    }

    double Sale::getTotal()
    {
            double finalTotal = 0;
            for (int i = 0; i < m_sale.size(); i++)
            {
                    finalTotal += m_sale[i].getSubTotal();
            }
            return finalTotal;
    }

    //MENUITEM CLASS
    IMPLEMENTATION///////////////////////////////////////////////////////////////////////
    string MenuItem::getDescription()
    {
            return m_description;
    }

    double MenuItem::getCost()
    {
            return m_cost;
    }

    void MenuItem::displayMenuItem()
    {
            cout << m_description << " $" << m_cost << " per " << m_units;
    }

    MenuItem::MenuItem(string description, double cost, string unit)
    {
            m_description = description;
            m_cost = cost;
            m_units = unit;
    }
```

```
string MenuItem::getUnits()
{
	return m_units;
}
//ITEM CLASS IMPLEMENTATION////////////////////////////////////////////////////////
double Item::getSubTotal()
{
	return m_menuItem.getCost() *m_quantity;
}

void Item::displayItem()
{
	m_menuItem.displayMenuItem();
	cout << "    Quantity is " << m_quantity << " Sub Total is $" << m_quantity
* m_menuItem.getCost() << endl;
}

//CASH CLASS
IMPLEMENTATION////////////////////////////////////////////////////////////
string Cash::getTransaction()
{
	string transaction;
	transaction = "Cash Transaction: Amount = $";
	transaction += to_string(getPayment());

	return transaction;
}

//CREDITCARD CLASS IMPLEMENTATION//////////////////////////////////////////////////
string CreditCard::getTransaction()
{
	string transaction;
	transaction = "Credit Card Transaction: Amount = $";
	transaction += to_string(getPayment());
	transaction += " Card Number = ";
	transaction += m_creditNumber;

	return transaction;
}

void CreditCard::promptForActNum()
{
	cout << "Please enter Credit Card Number" << endl;
	cin >> m_creditNumber;
}

//PHONEPAY CLASS
IMPLEMENTATION//////////////////////////////////////////////////////////
string PhonePay::getTransaction()
{
	string transaction;
	transaction = "Phone Pay Transaction: Amount = $";
	transaction += to_string(getPayment());
	transaction += " Phone Number = ";
	transaction += m_phoneNumber;

	return transaction;
}

void PhonePay::promptForActNum()
{
	cout << "Please enter Phone Number" << endl;
	cin >> m_phoneNumber;
}
//Code by Jessica Gehring
```

Source Code: hpp file

- Holds all the classes that makes up the Mitchcloud POS system

```
#include <string>
#include <vector>
#include <fstream>
using namespace std;

#ifndef MITCHCLOUD_H
#define MITCHCLOUD_H

class Sale;

class MenuItem
{
public:
	string getDescription();
	string getUnits();
	double getCost();
	void displayMenuItem();
	MenuItem(string, double, string);

private:
	string m_description;
	double m_cost;
	string m_units;
};
////////////////////////////////////////////////////////////////////////////////
class UserInfo
{
public:
	UserInfo() {m_Password = " "; m_userName = " ";}
	UserInfo(string username, string password) : m_userName(username),
m_Password(password) {}
	bool checkUsername(string);
	bool checkPassword(string);
	string getUsername() { return m_userName;}
	string getPassword() { return m_Password; }

private:
	string m_Password;
	string m_userName;
};
////////////////////////////////////////////////////////////////////////////////
class Payment
{
public:
	void setPayment(double payment) { m_paymentAmount = payment; }
```

```
        double getPayment() { return m_paymentAmount; }
        virtual void setAccountNum(string accountNum) {};
        virtual string getAccountNum() { return " "; }
        virtual string paymentType() = 0;
        virtual string getTransaction() = 0;
        Payment() {m_paymentAmount = 0;}
        virtual void promptForActNum() {};
private:
        double m_paymentAmount;
};
///////////////////////////////////////////////////////////////////////////////////
class Cash : public Payment
{
public:
        virtual string paymentType() {return "cash";}
        virtual string getTransaction();
};
///////////////////////////////////////////////////////////////////////////////////
class CreditCard : public Payment
{
public:
        virtual string paymentType() { return "credit"; }
        virtual string getTransaction();
        virtual string getAccountNum() { return m_creditNumber; }
        virtual void setAccountNum(string creditNum) { m_creditNumber = creditNum;}
        virtual void promptForActNum();
private:
        string m_creditNumber;
};
///////////////////////////////////////////////////////////////////////////////////
class PhonePay : public Payment
{
public:
        virtual string paymentType() { return "phonePay"; }
        virtual string getTransaction();
        virtual string getAccountNum() { return m_phoneNumber; }
        virtual void setAccountNum(string phoneNum) { m_phoneNumber = phoneNum; }
        virtual void promptForActNum();
private:
        string m_phoneNumber;
};
///////////////////////////////////////////////////////////////////////////////////
class Register
{
public:
        void login();
        int displayMenu();
        void createMenu();
        void startSale();
        void endSale();
        void addSaleItem(int, int);
        void displayReceipt();
        void makePayment(); //interactive
        void makePayment(int, string); //non-interactive
        void salesTransaction();
        Register() : m_clerkID("ICE", "CUBE"), m_custID("FLOYD", "THINICE"),
m_managerID("METALLICA", "UNDERICE"), m_salesAnID("ZEPPELIN", "ICEANDSNOW") {}
        bool performAnotherTransaction();
```

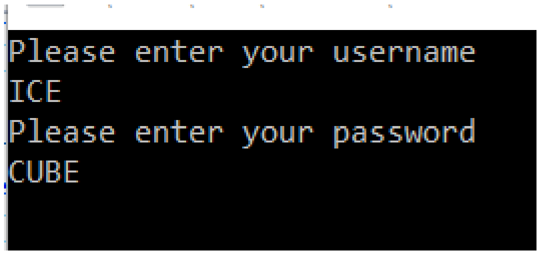
```
private:
        UserInfo m_clerkID;
        UserInfo m_custID;
        UserInfo m_managerID;
        UserInfo m_salesAnID;
        UserInfo m_currentUser;
        vector <MenuItem> m_menu;
        Sale *m_sale;
        ofstream m_transactionFile;

        //void addLogEntry(string);
        void openTransactionLog();
        void closeTransactionLog();
        void interactiveSale();
        void fileSale();
};
/////////////////////////////////////////////////////////////////////////////////
class Item
{
public:
        Item(int qty, MenuItem menuItem) : m_quantity(qty), m_menuItem(menuItem){};
        void displayItem();
        double getSubTotal();
private:
        int m_quantity;
        MenuItem m_menuItem;
};
/////////////////////////////////////////////////////////////////////////////////
class Sale
{
public:
        Sale() {};
        void createSaleItem(MenuItem, int);
        void displaySale();
        double getTotal();
        int getNumberItemsInOrder() { return m_sale.size();}
private:
        vector <Item> m_sale;
};
#endif
//Code by: Jessica Gehring
```

User Interface

- Requires a user to input a username and password correctly to get into the system
- User is able to select the type of interaction they would like to have with the system
- Hardware: user devices, monitors, bar code scanner
- Software: Inventory system, customer/clerk/manager information system, payment system, tax system, discount system, accounting system, shipping system

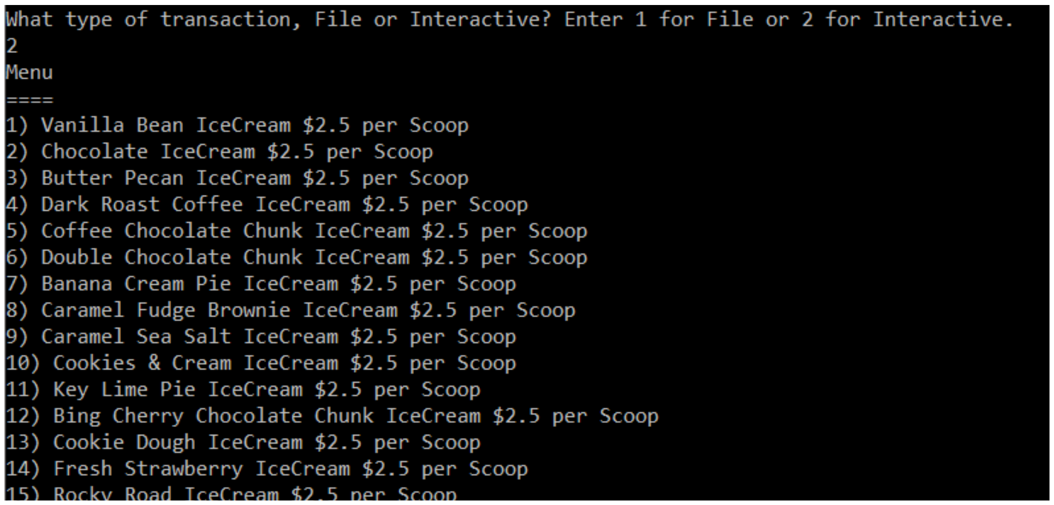
- User is able to select the type of product that they would like to purchase
- User is able to select the type of transaction they would like
- The user first is prompted to input their username, correctly.
- The user is then prompted to input their password, correctly



```
Please enter your username
ICE
Please enter your password
CUBE
```

Example 1

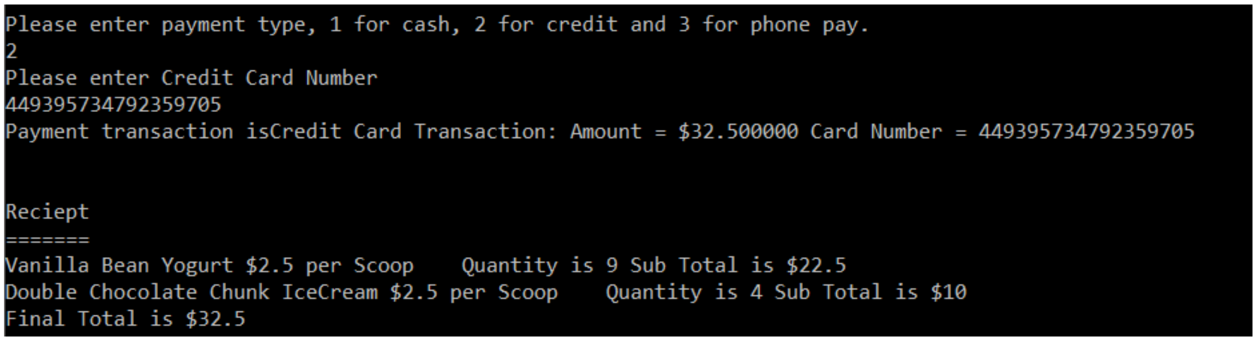
- The user is then prompted to choose what interaction they would like to choose from, a File or Interactive. Interactive gives them the menu of items and File reads from a file.



```
What type of transaction, File or Interactive? Enter 1 for File or 2 for Interactive.
2
Menu
====
1) Vanilla Bean IceCream $2.5 per Scoop
2) Chocolate IceCream $2.5 per Scoop
3) Butter Pecan IceCream $2.5 per Scoop
4) Dark Roast Coffee IceCream $2.5 per Scoop
5) Coffee Chocolate Chunk IceCream $2.5 per Scoop
6) Double Chocolate Chunk IceCream $2.5 per Scoop
7) Banana Cream Pie IceCream $2.5 per Scoop
8) Caramel Fudge Brownie IceCream $2.5 per Scoop
9) Caramel Sea Salt IceCream $2.5 per Scoop
10) Cookies & Cream IceCream $2.5 per Scoop
11) Key Lime Pie IceCream $2.5 per Scoop
12) Bing Cherry Chocolate Chunk IceCream $2.5 per Scoop
13) Cookie Dough IceCream $2.5 per Scoop
14) Fresh Strawberry IceCream $2.5 per Scoop
15) Rocky Road IceCream $2.5 per Scoop
```

Example 2

- The user then can select the products they would like to purchase and can proceed to checkout.
- The user is then prompted to choose a payment option, from phone, cash or credit card.
- After a transaction the user is then prompted to enter if they would like another transaction or not



```
Please enter payment type, 1 for cash, 2 for credit and 3 for phone pay.
2
Please enter Credit Card Number
449395734792359705
Payment transaction isCredit Card Transaction: Amount = $32.500000 Card Number = 449395734792359705


Reciept
=======
Vanilla Bean Yogurt $2.5 per Scoop    Quantity is 9 Sub Total is $22.5
Double Chocolate Chunk IceCream $2.5 per Scoop    Quantity is 4 Sub Total is $10
Final Total is $32.5
```

Example 3

Data

- dataCash.txt:

  1 15 7

  1 20 19

  1 9 2

  1 42 1

  1 32 19

  2 1

- dataCredit.txt:

  1 2 4

  1 15 7

  1 42 1

  2 2 8890658

- dataPhone.txt:

  1 23 4

  1 40 2

  1 6 10

  1 5 9

  2 3 123-456-7890

- userClerk.txt:
  1 24 77
  1 25 100
  1 10 3

```
1 35 66
2 1 YES
FLOYD THINICE
```

- userCustomer.txt:

```
1 24 8
1 27 100
1 35 2
1 9 25
1 33 200
2 2 999090945
```

- userManager.txt:

```
1 25 5
1 40 33
1 20 1
2 3 123-456-7890
```

- userSales.txt:

```
1 23 4
1 34 1
1 42 23
1 2 4
1 4 12
2 1
```

Logs

- Log_12_02_2017_22_45_19.txt:

```
User: ICE Logged in

Sale Started

User: ICE selected Interactive Transaction

User: ICE shows the menu of items to order

User: ICE selected item # 10

User: ICE is ordering Item # 1

User: ICE Ordred Key Lime Pie IceCream

User: ICE Selected the amount 4 Scoop

Price is $10

SubTotal is $10

Receipt Printed
```

```
User: ICE shows the menu of items to order

User: ICE selected item # 34

User: ICE is ordering Item # 2

User: ICE Ordred Fresh Mint Choco Chunk

User: ICE Selected the amount 12 Pie

Price is $323.4

SubTotal is $333.4

Receipt Printed

User: ICE shows the menu of items to order

User: ICE selected item # 2

User: ICE is ordering Item # 3

User: ICE Ordred Butter Pecan IceCream

User: ICE Selected the amount 5 Scoop

Price is $12.5

SubTotal is $345.9

Receipt Printed

User: ICE shows the menu of items to order

User: ICE selected to end order.

Receipt Printed

Sale is Finished
```

- Log_12_02_2017_22_46_25.txt:

```
User: FLOYD Logged in

Sale Started

User: FLOYD selected Interactive Transaction

User: FLOYD shows the menu of items to order

User: FLOYD selected item # 59

User: FLOYD is ordering Item # 1

User: FLOYD Ordred Fresh Mint Choco Chunk IceCream

User: FLOYD Selected the amount 9 Scoop
```

```
Price is $22.5

SubTotal is $22.5

Receipt Printed

User: FLOYD shows the menu of items to order

User: FLOYD selected item # 41

User: FLOYD is ordering Item # 2

User: FLOYD Ordred Peanut Butter Fudge

User: FLOYD Selected the amount 2 Special

Price is $19.1

SubTotal is $41.6

Receipt Printed

User: FLOYD shows the menu of items to order

User: FLOYD selected to end order.

Receipt Printed

Sale is Finished
```

- Log_12_02_2017_22_47_24.txt:

```
User: METALLICA Logged in
Sale Started
User: METALLICA selected Interactive Transaction
User: METALLICA shows the menu of items to order
User: METALLICA selected item # 2
User: METALLICA is ordering Item # 1
User: METALLICA Ordred Butter Pecan IceCream
User: METALLICA Selected the amount 5 Scoop
Price is $12.5
SubTotal is $12.5
Receipt Printed
User: METALLICA shows the menu of items to order
User: METALLICA selected item # 32
User: METALLICA is ordering Item # 2
User: METALLICA Ordred Coffee & Cookies Yogurt
User: METALLICA Selected the amount 100 Scoop
Price is $250
SubTotal is $262.5
Receipt Printed
User: METALLICA shows the menu of items to order
```

```
User: METALLICA selected item # 34
User: METALLICA is ordering Item # 3
User: METALLICA Ordred Fresh Mint Choco Chunk
User: METALLICA Selected the amount 8 Pie
Price is $215.6
SubTotal is $478.1
Receipt Printed
User: METALLICA shows the menu of items to order
User: METALLICA selected to end order.
Receipt Printed
Sale is Finished
```

- Log_12_02_2017_22_49_57.txt:

```
User: ICE Logged in
Sale Started
User: ICE selected File Transaction
Read Command 1, selection type, item read is 24 quantity of item is 77
User: ICE is ordering Item # 1
User: ICE Ordred Raspberry Sorbet
User: ICE Selected the amount 77 Scoop
Price is $192.5
SubTotal is $192.5
Read Command 1, selection type, item read is 25 quantity of item is 100
User: ICE is ordering Item # 2
User: ICE Ordred Vegan Salted Caramel Pecan
User: ICE Selected the amount 100 Scoop
Price is $250
SubTotal is $442.5
Read Command 1, selection type, item read is 10 quantity of item is 3
User: ICE is ordering Item # 3
User: ICE Ordred Key Lime Pie IceCream
User: ICE Selected the amount 3 Scoop
Price is $7.5
SubTotal is $450
Read Command 1, selection type, item read is 35 quantity of item is 66
User: ICE is ordering Item # 4
User: ICE Ordred Choco Peanut Butter Cup
User: ICE Selected the amount 66 Pie
Price is $1778.7
SubTotal is $2228.7
Read command 2, selected payment type is 1
Payment chosen: Cash Transaction: Amount = $2228.700000 Successful Transaction.
Invalid command read from file. Command number read was: 0
Receipt Printed
Sale is Finished
```

- Log_12_02_2017_22_51_11.txt:

```
User: FLOYD Logged in
Sale Started
User: FLOYD selected File Transaction
Read Command 1, selection type, item read is 24 quantity of item is 8
User: FLOYD is ordering Item # 1
User: FLOYD Ordred Raspberry Sorbet
User: FLOYD Selected the amount 8 Scoop
Price is $20
SubTotal is $20
Read Command 1, selection type, item read is 27 quantity of item is 100
User: FLOYD is ordering Item # 2
User: FLOYD Ordred PeanutButter Pretzel Yogurt
User: FLOYD Selected the amount 100 Scoop
Price is $250
SubTotal is $270
Read Command 1, selection type, item read is 35 quantity of item is 2
User: FLOYD is ordering Item # 3
User: FLOYD Ordred Choco Peanut Butter Cup
User: FLOYD Selected the amount 2 Pie
Price is $53.9
SubTotal is $323.9
Read Command 1, selection type, item read is 9 quantity of item is 25
User: FLOYD is ordering Item # 4
User: FLOYD Ordred Cookies & Cream IceCream
User: FLOYD Selected the amount 25 Scoop
Price is $62.5
SubTotal is $386.4
Read Command 1, selection type, item read is 33 quantity of item is 200
User: FLOYD is ordering Item # 5
```

```
User: FLOYD Ordred Turtle

User: FLOYD Selected the amount 200 Pie

Price is $5390

SubTotal is $5776.4

Read command 2, selected payment type is 2

Payment chosen: Credit Card Transaction: Amount = $5776.400000 Card Number =
999090945 Successful Transaction.

Read command 2, selected payment type is 2

Payment chosen: Credit Card Transaction: Amount = $5776.400000 Card Number =
Successful Transaction.

Receipt Printed

Sale is Finished
```

- Log_12_02_2017_22_51_29.txt:

```
User: METALLICA Logged in
Sale Started
User: METALLICA selected File Transaction
Read Command 1, selection type, item read is 25 quantity of item is 5
User: METALLICA is ordering Item # 1
User: METALLICA Ordred Vegan Salted Caramel Pecan
User: METALLICA Selected the amount 5 Scoop
Price is $12.5
SubTotal is $12.5
Read Command 1, selection type, item read is 40 quantity of item is 33
User: METALLICA is ordering Item # 2
User: METALLICA Ordred Root Beer Float
User: METALLICA Selected the amount 33 Special
Price is $100.65
SubTotal is $113.15
Read Command 1, selection type, item read is 20 quantity of item is 1
User: METALLICA is ordering Item # 3
User: METALLICA Ordred Pralines & Cream IceCream
User: METALLICA Selected the amount 1 Scoop
Price is $2.5
SubTotal is $115.65
Read command 2, selected payment type is 3
Payment chosen: Phone Pay Transaction: Amount = $115.650000 Phone Number = 123-
456-7890 Successful Transaction.
Read command 2, selected payment type is 3
Payment chosen: Phone Pay Transaction: Amount = $115.650000 Phone Number =
Successful Transaction.
Receipt Printed
Sale is Finished
```

- Log_12_02_2017_22_54_20.txt:

```
User: FLOYD Logged in
Sale Started
User: FLOYD selected File Transaction
Read Command 1, selection type, item read is 15 quantity of item is 7
User: FLOYD is ordering Item # 1
User: FLOYD Ordred Fresh Mint Choco Chunk IceCream
User: FLOYD Selected the amount 7 Scoop
Price is $17.5
SubTotal is $17.5
Read Command 1, selection type, item read is 20 quantity of item is 19
User: FLOYD is ordering Item # 2
User: FLOYD Ordred Pralines & Cream IceCream
User: FLOYD Selected the amount 19 Scoop
Price is $47.5
SubTotal is $65
Read Command 1, selection type, item read is 9 quantity of item is 2
User: FLOYD is ordering Item # 3
User: FLOYD Ordred Cookies & Cream IceCream
User: FLOYD Selected the amount 2 Scoop
Price is $5
SubTotal is $70
Read Command 1, selection type, item read is 42 quantity of item is 1
User: FLOYD is ordering Item # 4
User: FLOYD Ordred Smoothie
User: FLOYD Selected the amount 1 Special
Price is $10.05
SubTotal is $80.05
Read Command 1, selection type, item read is 32 quantity of item is 19
User: FLOYD is ordering Item # 5
User: FLOYD Ordred Coffee & Cookies Yogurt
User: FLOYD Selected the amount 19 Scoop
Price is $47.5
SubTotal is $127.55
Read command 2, selected payment type is 1
Payment chosen: Cash Transaction: Amount = $127.550000 Successful Transaction.
Read command 2, selected payment type is 1
Payment chosen: Cash Transaction: Amount = $127.550000 Successful Transaction.
Receipt Printed
Sale is Finished
```

- Log_12_02_2017_22_54_20.txt:

```
User: FLOYD Logged in
Sale Started
User: FLOYD selected File Transaction
Read Command 1, selection type, item read is 2 quantity of item is 4
User: FLOYD is ordering Item # 1
User: FLOYD Ordred Butter Pecan IceCream
User: FLOYD Selected the amount 4 Scoop
```

```
        Price is $10
        SubTotal is $10
        Read Command 1, selection type, item read is 15 quantity of item is 7
        User: FLOYD is ordering Item # 2
        User: FLOYD Ordred Fresh Mint Choco Chunk IceCream
        User: FLOYD Selected the amount 7 Scoop
        Price is $17.5
        SubTotal is $27.5
        Read Command 1, selection type, item read is 42 quantity of item is 1
        User: FLOYD is ordering Item # 3
        User: FLOYD Ordred Smoothie
        User: FLOYD Selected the amount 1 Special
        Price is $10.05
        SubTotal is $37.55
        Read command 2, selected payment type is 2
        Payment chosen: Credit Card Transaction: Amount = $37.550000 Card Number =
        8890658 Successful Transaction.
        Read command 2, selected payment type is 2
        Payment chosen: Credit Card Transaction: Amount = $37.550000 Card Number =
        Successful Transaction.
        Receipt Printed
        Sale is Finished
```

- Log_12_02_2017_22_54_44.txt:

```
User: FLOYD Logged in
Sale Started
User: FLOYD selected File Transaction
Read Command 1, selection type, item read is 23 quantity of item is 4
User: FLOYD is ordering Item # 1
User: FLOYD Ordred Mango Sorbet
User: FLOYD Selected the amount 4 Scoop
Price is $10
SubTotal is $10
Read Command 1, selection type, item read is 40 quantity of item is 2
User: FLOYD is ordering Item # 2
User: FLOYD Ordred Root Beer Float
User: FLOYD Selected the amount 2 Special
Price is $6.1
SubTotal is $16.1
Read Command 1, selection type, item read is 6 quantity of item is 10
User: FLOYD is ordering Item # 3
User: FLOYD Ordred Banana Cream Pie IceCream
User: FLOYD Selected the amount 10 Scoop
Price is $25
SubTotal is $41.1
Read Command 1, selection type, item read is 5 quantity of item is 9
User: FLOYD is ordering Item # 4
User: FLOYD Ordred Double Chocolate Chunk IceCream
User: FLOYD Selected the amount 9 Scoop
```

```
Price is $22.5
SubTotal is $63.6
Read command 2, selected payment type is 3
Payment chosen: Phone Pay Transaction: Amount = $63.600000 Phone Number = 123-456-7890
Successful Transaction.
Read command 2, selected payment type is 3
Payment chosen: Phone Pay Transaction: Amount = $63.600000 Phone Number =   Successful
Transaction.
Receipt Printed
Sale is Finished
```