Sale Module

Linda is looking at expanding into online business, and since she’s looking at loyalty programs, recording customer information is important to her. Recording sales from her website is easy, but recording sales at their kiosks is problematic. Some customers simply do not want give their personal information, so Linda is comfortable with recording these sales to one ‘customer’ by location.

At the end of the day, Linda would like the ability to query purchases – minus sales by location so that she can adjust inventory to account for unrecorded sales.

Your task is to create a class diagram and sequence diagrams to support the following user stories and systems use case specifications.

Use Case: Maintain Customer

User Story

As the owner of this business, I would like to record contact information for my customers.

Acceptance Criteria:

1. Must be able to flag customers as no longer active.
2. Must be able to query customers.
3. Must be able to query sales by company.

A diagram of a product

Description automatically generated with medium confidence

Use Case Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name | Create Customer | | |
| Triggering Event | New customer requests sale. | | |
| Brief Description | Allows the Owner to record a new customer’s information. | | |
| Actors | Owner | | |
| Related Use Cases |  | | |
| Preconditions | Owner has opened the Main Menu. | | |
| Post Conditions | Customer is saved to the database. Sales can be created for the Customer | | |
| Flow of activities | Actor | | System |
|  |  | Requests to add a new Customer | Prompts for Customer name, address, telephone and email address |
|  |  | Enters customer name, address, telephone number and email address | Verifies that name, address, telephone number and email address have been entered.  Creates a unique customer identifier  Displays the customer’s identifier, name, address, telephone number and email address  Prompts to save |
|  | 3. | Request to save | Saves the customer |
| Exception Conditions | * Owner chooses to cancel adding the customer | | |

A diagram of a diagram

Description automatically generatedUser Story

As the owner of this business, I would like to record my sales so that I can quickly see how much money I’ve made.

Acceptance Criteria:

1. Must be able to record sales by selecting products by product type and enter quantity sold.

Use Case Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name | Create Sale | | |
| Triggering Event | Customer has to purchase products | | |
| Brief Description | Allows the Owner to record a new sale | | |
| Actors | Owner | | |
| Related Use Cases |  | | |
| Preconditions | Owner has opened the Main Menu. | | |
| Post Conditions | Sale is created and can now be queried | | |
| Flow of activities | Actor | | System |
|  | 1. | Requests to add a new sale | Displays a list of customers and prompts for selection. Displays a list of locations and prompts for selection. Prompts for sales date |
|  | 2. | Selects a location  Selects a customer  Enters sales date | Verifies that customer and location were selected.  Verifies that sale date has been entered and generates a unique sale number.  Displays Sale identifier, date and customer. Display a list of product types and prompts for selection. |
|  | 3. | Selects a product type | Displays a list of products including product name and price. Prompts for selection and quantity sold |
|  | 3. | Selects a product and enters quantity sold | Verifies that quantity sold was entered  Adds product to the sale and calculates extended price \* quantity  Calculates HST  Updates totals.  Display a list of products sold. Prompts for another product from the list and quantity sold. Prompts for another product type |
|  | 4. | Repeats above step until all products entered | Prompts to select a product type. Prompts to confirm sale |
|  | 5. | Repeats above 2 steps until all product types are selected. | Prompts to confirm sale |
|  | 5. | Chooses to save | Saves the sale.  Return to the main menu |
| Exception Conditions | * Owner chooses to cancel adding the sale | | |

A diagram of a project

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name | Query Purchase - Sale | | |
| Triggering Event | Owner requires a list of remaining inventory (end of day processing) | | |
| Brief Description | Allows the Owner to retrieve purchases – sales so that inventory can be reconciled | | |
| Actors | Owner | | |
| Related Use Cases |  | | |
| Preconditions | Owner has opened the Main Menu. | | |
| Post Conditions | Purchases – sales are displayed. | | |
| Flow of activities | Actor | | System |
|  | 1. | Retrieve locations | Displays a list of locations and prompts for selection |
|  | 2. | Requests to purchases – sales for location | Retrieves Purchases by product assigned to the location. Retrieves Sales by product assigned to the location. Retrieves Inventory adjustments by product assigned to the location. Subtracts purchases – sales and inventory adjustments) and displays quantity in inventory. |
|  |  | Request to exit | returns to the main menu |
| Exception Conditions | Owner requests to cancel query | | |

A diagram of a computer program

Description automatically generated

User Story

As the owner of this business, I would like to record inventory adjustments to accommodate for the cash sales that happen quickly and aren’t recorded.

Acceptance Criteria:

1. Must be able to record inventory adjustments by location and product.

Use Case Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name | Create Inventory Adjustment | | |
| Triggering Event | End of day processing | | |
| Brief Description | Allows the Owner to adjust inventory to accommodate for unrecorded sales. | | |
| Actors | Owner | | |
| Related Use Cases |  | | |
| Preconditions | Owner has opened the Main Menu. | | |
| Post Conditions | Inventory adjustment is recorded and available to query | | |
| Flow of activities | Actor | | System |
|  | 1. | Requests to add an inventory adjustment | Displays a list of product types and prompts for selection. Displays a list of locations and prompts for selection. |
|  | 2. | Selects a location  Selects a product type. | Verifies that product type and location were selected.  Displays a list of products and prompts for selection. Prompts for adjustment quantity. |
|  | 3. | Selects a product and enters adjusted quantity. | Creates an adjustment identifier and assigns the system date to adjustment date. Displays the inventory adjustment |
|  | 4. | Repeats the above step until all adjustments for the product type are entered. Chooses to confirm. | Saves the inventory adjustments.  Returns to the main menu. |
| Exception Conditions | * Owner chooses to cancel adding the inventory adjustments. | | |

A diagram of a project

Description automatically generated

**.H File**

**Product.h**

Public Class Product{

Int productid;

String productName;

Double price;

ProductType productType;

SaleDetail saleDetail;

InventoryAdjustment inventoryAdjustment;

String purchaseDetail;

Int purchaseID;

}

**Customer.h**

Public class Customer{

Int customerID;

String name;

String address;

Int contact;

String email;

Sale saleSet;

}

**ProductType.h**

Public class ProductType{  
int productTypeID;

String description;

}

**Sale.h**

Public class Sale{

Int saleID;

String saleDate;

Customer customer;

Location location;

SaleDetail saleDetail;

}

**SaleDetail.h**

Public class SaleDetail{

Int saleDetailID;

Sale sale;

Product product;

Int quantity;

}

**Location.h**

Public class Location**{**

int locationID;

string address;

Sale sale;

SaleDetail saleDetail;

InventoryAdjustment inventoryAdjustment;

}

**InventoryAdjustment.h**

Public class InventoryAdjustment{

Int InventoryAdjustmentID;

String date;

Location location;

Int Quantity;

Product product;

}

**UIController.h**

Public class startNewCustomer(){

// requests to add new customer

}

Public class addCustomer(name,address,contact,email){

// passes entered data to the domain controller

// displays customer

}

Public class save(){

// sends request to domain controller to save

}

Public class startSale(){

// requests to add new sale

//displays list of customers and locations

}

Public class selectSale(customerID,locationID,saleDate){

// passes entered data to the domain controller

// displays sale and list of product types

}

Public class selectProductType(productTypeID){

// passes entered data to the domain controller

// displays list of product

}

Public class addSaleDetail(productID,quantity){

// passes entered data to the domain controller

// displays SaleDetail

}

Public class retrieveLocations(){  
// allows the user to request locations

// passes message to the Domain Controller

// displays list of locations to the user

}

Public class getPurchase(locationID,ProductID){

// passes entered data to the domain controller

// displays list of quantity in inventory

}

Public class exit(){

//owner choses to exit

}

Public class addInventoryAdjustment(){

//request to add InventoryAdjustment

}

Public class selectProducts(productTypeID,locationID){

// allows the user to request products

// passes message to the Domain Controller

// displays list of products to the user

}

Public class addInventoryAdjustment(productID,quantity){

// passes entered data to the domain controller

// displays InventoryAdjustment

}

**DomainController.h**

Public class createCustomer(name,address,contact,email){  
//sends request to create a customer

//receives new customer and returns to UI

}

Public class generateCustomerID(){

//creates purchaseid

}

Public class save(){

//sends save to entity manager

}

Public class getCustomers&Locations(){

// requests customers and locations from the entity manger

// retrieves customers and locations data and returns to the UIController

}

Public class createSale(customerID,locationID,saleDate){

//sends request to create a sale

//receives new sale and returns to UI

}

Public class getProducts(productTypeID){

// requests product for that producttypeID from the entity manger

// retrieves products data and returns to the UIController

}

Public class addSaleDetail(productID,quantity){

//sends request to add productID,quantity in that detail

//receives SaleDetailand returns to UI

}

Public class extendedPrice=calculateExtendedPrice(price\*quantity){

//calculates extendedprice

}

Public class Hst=calculateHST(){

//calculates HST

}

Public class updateTotals(){

//updates totals

}

Public class getLocations(){

// requests locations from the entity manger

// retrieves location data and returns to the UIController

}

Public class quantity=calculateInventory(){

//calculates quantity in inventory

}

Public class getProductTypes,Locations(){

// requests ProductTypes &Locations from the entity manger

// retrieves ProductTypes &Locations data and returns to the UIController

}

Public class createProducts(productTypeID,locationID){

//sends request to create a product

//receives list of products and returns to UI

}

Public class createInventoryAdjustment(productID,quantity,date){

//sends request to create a InventoryAdjustment

//receives new InventoryAdjustment and returns to UI

}