VIETNAMESE – GERMAN UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER SCIENCE DEPARTMENT

JAVA PROJECT REPORT

Grocery Store Management

Module 61CSE215: Object Oriented Programming in JAVA

1. Vo Pham Khang Huy - 10421082
2. Ly Minh Hung - 10421079
3. Pham Phu Tuan Khoa - 10421129
4. Nguyen Tien Khoa - 10421085

Lecturer: Dr. Tran Hong Ngoc

Binh Duong, WS2023

Versions

Date	Version	In charge	Description
16-Oct-23	V1.0	Pham Phu Tuan Khoa, Vo Pham Khang Huy	Update Section I and II
18-Oct-23	V1.0	Vo Pham Khang Huy	Create diagram
19-Oct-23	V1.1	Nguyen Tien Khoa, Pham Phu Tuan Khoa	Import diagrams
20-Oct-23	V1.2	Pham Phu Tuan Khoa, Ly Minh Hung	Update Section II and III
21-Oct-23	V1.3	Nguyen Tien Khoa	Update Versions, Duty Roster, List of Figures/Tables, Abbreviation, Table of Contents
22-Oct-23	V1.4	Nguyen Tien Khoa	Check and fix mistakes
1-Nov-23	V1.5	Nguyen Tien Khoa Pham Phu Tuan Khoa Vo Pham Khang Huy	Update Section VII Update Section IV and V Update Section III (abstract classes, OOP techniques)
6-Nov-23	V2.0	Pham Phu Tuan Khoa, Vo Pham Khang Huy	Create GUI for product, employee, supplier, customer, login
11-Nov-23	V2.1	Ly Minh Hung	Create function (Save, Update, Delete, and Search) for PRODUCT GUI
12-Nov-23	V2.1	Ly Minh Hung	Update Search function for PRODUCT GUI
13-Nov-23	V2.1	Ly Minh Hyng, Pham Phu Tuan Khoa	FOR GUI PRODUCT Create and Update Refresh button, and function Update Table and Search function by name instead of ID.
13-Nov-23	V2.2	Ly Minh Hung	Create and Update GUI for Sales, Invoice, Reports Create Class CartItem

			Update, create function, and connect to database for Sales, Invoice, and Reports.
14-Nov-23	V2.3	Ly Minh Hung	Fix Error of Vector by Using ArrayList for Sales and Product
14-Nov-23	V2.4	Vo Pham Khang Huy, Nguyen Tien Khoa	Update Full Report

Abbreviation

GSM	Grocery Store Management

List of Figures

List of Tables

Table of Contents

I. INTRODUCTION:

In this project, we investigate the business functions of the grocery store management system and provide a desktop application supporting users.

This project provides the basic functions for the stakeholders/users as follows:

- **Input Customer Information**: Manager can input and store customer information, including customer ID, name, and contact details.
- **Input Product Details**: Manager can add and update product details, such as product name, price, and quantity in stock.
- **Input Employee Information**: Manager can input employee information, including employee ID, name, position, and salary.
- **Search Customer's information**: The system allows searching and retrieving past customer transactions using customer IDs or transaction IDs.
- **Search Employee's Information**: Manager can look up employee information, including their roles and contact details.
- **Search Product's Information**: Manager can look up product information, including their category and price.

II. CLASS ANALYSIS:

1. Objects:

No	Objects	State	Behavior
1	Customer	- ID: 111 - Jack - 01751075	- Purchasing Product
2	Cashier	- ID: 121 - Tom - 01591596	- Making payment
3	Warehouse_manager	- ID: 122 - Jerry	- Controlling Quality

		- 01501574	
4	Supplier	- ID: 111 - CorpFood - 197491	- Supplying Product
5	Product	- Banana - ID: 777 - 100 - inPrice:$3 - outPrice:$5	- Are offered for Sale

Table 1: List of Objects

2. Classes:

No	Classes	Attributes	Methods
1	Person(abstract class)	- ID - Name - PhoneNo	- getID() - getName() - getPhoneNo() - setID() - setName() - setIPhoneNo() - describe()

2	Customer(super class)	- ID - Name - PhoneNo	- getID() - setID() - getName() - setName() - getPhoneNo() - setPhoneNo() - describe()
3	Employee(super class)	- ID - Name - PhoneNo - Role	- getID() - setID() - getName() - setName() - getPhoneNo() - setPhoneNo() - getRoleo() - setRole() - describe()
4	Cashier(subclass)	- ID - Name - PhoneNo	- getID() - setID() - getName()

			- setName() - getPhoneNo() - setPhoneNo() - describe()
5	Product	- ID - Name - Category - Quantity - inPrice - outPrice	- getID() - setID() - getName() - setName() - getCategory() - setCategory() - getQuantity() - setQuantity() - getInPrice() - setInPrice() - getOutPrice() - setOutPrice() - describe()

6	Supplier	- ID - Name - Barcode	- getID() - setID() - getName() - setName() - getBarcode() - setBarcode() - describe()
7	WarehouseManager(subclass)	- ID - Name - PhoneNo	- getID() - setID() - getName() - setName() - getPhoneNo() - setPhoneNo() - describe()
8	CartItems	- invoiceID - productName - productQuantity - unitPrice - totalPrice	- getInvoiceID() - setInvoiceID() - getProductName() - setProductName() - getProductQuantity() - setProductQuantity()

			- getUnitPrice() - setUnitPrice() - getTotalPrice() - setTotalPrice()
9	InVoice	- saleId - invoiceId - customerId - customerName - totalQty - totalBill - status - balance	- getSaleId() - setSaleId() - getInvoiceId() - setInvoiceId() - getCustomerId() - setCustomerId() - getCustomerName() - setCustomerName() - getTotalQty() - setTotalQty() - getTotalBill() - setTotalBill() - getStatus() - setStatus() - getBalance() - setBalance()

Table 2: List of Classes

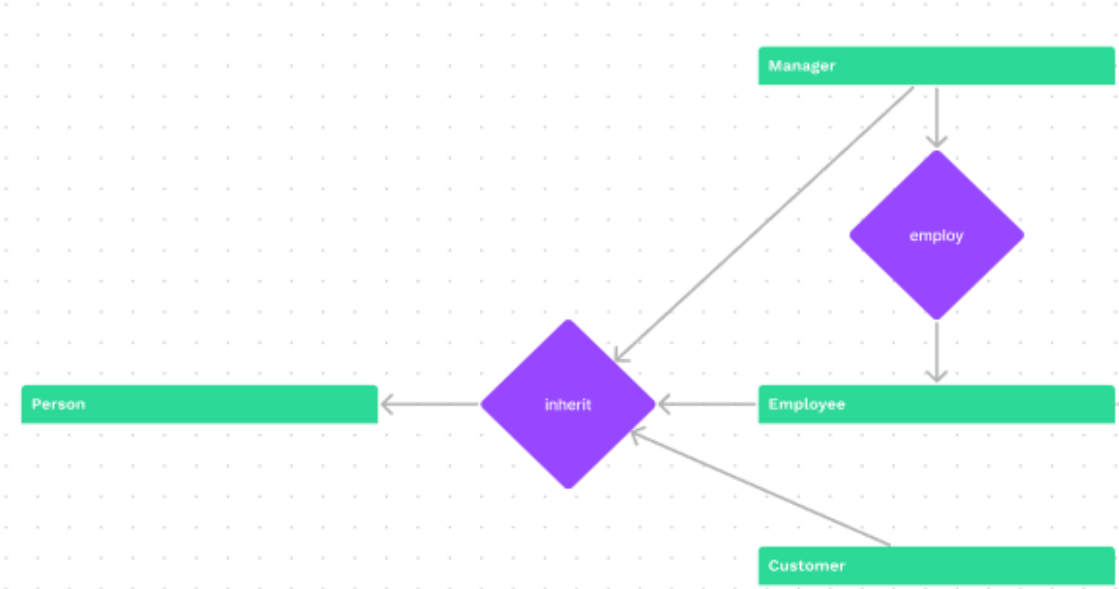

Figure 1: Inheritance diagram

III. CLASS DESIGN

1. Classes:

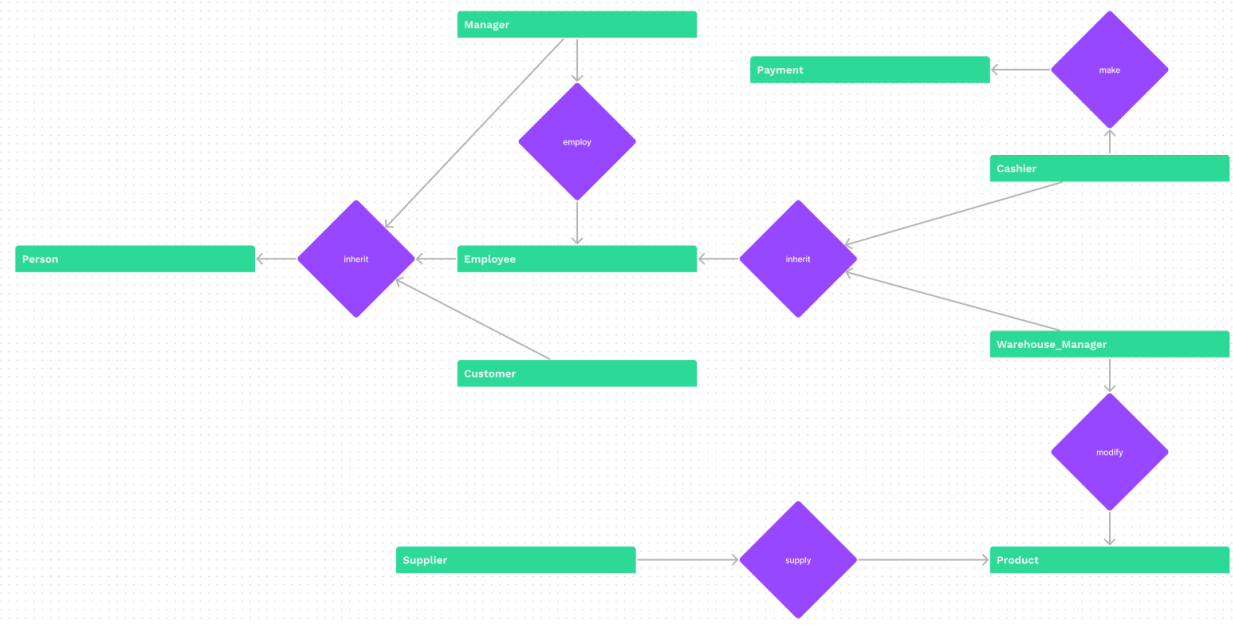

Figure 2: Class Diagram of Project

No	Class	Instance Variable	Methods	Description
1	Person	private String **name** private String **ID** private String **phoneNo**	public String getName() public String getID() public String getPhone() public String public void setName() public void setID() public void setPhoneNo() public void describe()	This class is an abstract class used to manage the information of a Person(maybe Employee or Customer).
2	Employee	super(name, ID, phoneNo) public String **position**	public String getRole() public void setRole() public void describe()	This class is used to get information and behavior of an Employee.
3	Customer	super(name, ID, phoneNo)	public void describe()	This class is used to manage Customer information and behavior.
4	Cashier	super(name, ID, phoneNo)	public void accessCO() public void accessWH	This class is used to manage Cashier information and behavior.
5	Product	private Integer id	public Integer getId() public void setId()	This class is used to manage

		private String name private String category private Integer quantity private Double inPrice private Double outPrice	public String getName() public void setName() public String getCategory() public void setCategory() public Integer getQuantity() public void setQuantity() public Double getInPrice() public void setInPrice() public Double getOutPrice() public void setOutPrice() public void describe()	Product information and behavior.
6	Supplier	private int Id private String name private String barCode	public int getId() public void setId() public String getName() public void setName() public String getBarCode() public void setBarCode()	This class is used to manage Supplier information and behavior.
7	WarehouseManager	super(name, ID, phoneNo)	public void accessWH()	This class is used to manage WarehouseManager

				information and behavior
8	CartItems	private String invoiceID; private String productName; private int productQuantity; private Double unitPrice; private Double totalPrice;	Getter and Setter()	This class is used to manage cardItems
9	Invoice	private Integer saleId; private String invoiceId; private String customerId; private String customerName; private int totalQty; private Double totalBill; private String status; private Double balance;	Getter and Setter()	This class is used to manage Invoice

Table 3: Details of Classes

- Abstract classes:

No	Abstract Class	Abstract Methods	Concrete Methods	Description
1	Person	getId() getName() getPhoneNo() setId() setName() setPhoneNo() abstract describe()	Util()	The Person abstract class is used by subclasses Customer and Employee to define common methods and properties for individuals in the grocery store management system.

Table 4: Details of Abstract Classes

```java
package Classes;

abstract public class Person {
    private int Id;
    private String name;
    private String phoneNo;
    public Person(int Id, String name, String phoneNo) {
        this.Id = Id;
        this.name = name;
        this.phoneNo = phoneNo;
    }

    public int getId() {
        return Id;
    }

    public void setId(int Id) {
        this.Id = Id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPhoneNo() {
        return phoneNo;
    }

    public void setPhoneNo(String PhoneNo) {
        this.phoneNo = phoneNo;
    }
    abstract void describe();
}
```

Figure 3: Person class

2. Some OOP Techniques:

2.1. Overloading method:

- In the Grocery Store Management System code, overloading methods are not explicitly demonstrated.

2.2. Overriding method:

- Overriding methods are demonstrated in the Customer and Employee classes. They override the abstract methods from the Person abstract class to provide specific implementations for each subclass.

1. Inheritance

- Inheritance is evident in the Customer and Employee classes, which inherit properties and methods from the Person abstract class, demonstrating the inheritance relationship.

Customer:

```java
package Classes;

public class Customer extends Person{

    public Customer(int id, String name, String phoneNo) {
        super(id, name, phoneNo);
    }
    @Override
    void describe(){
        System.out.println("Customer's name: " + getName() + ", Customer's ID: " + getId() + ", Customer's PhoneNo: " + getPhoneNo());
    }
}
```

Figure 4: Customer class

Employee:

```java
package Classes;

public class Employee extends Person{
    private String role;

    public Employee(int id, String name, String phoneNo, String role) {
        super(id, name, phoneNo);
        this.role = role;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
    void describe(){
        System.out.println("Employee's ID: " + getId() + ", Employee's name: " + getName() + "Employee's PhoneNo: " + getPhoneNo() + "Employee's role: " + getRole());
    }

}
```

Figure 5: Employee class

Cashier:

```java
package Classes;

public class Cashier extends Employee implements accessCheckOut, accessWarehouse{

    public Cashier(int id, String name, String phoneNo, String role) {
        super(id, name, phoneNo, role);
    }

    @Override
    public void accessCO(int a) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void accessWH() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

}
```

Figure 6: Cashier class

Warehouse_manager:

```java
package Classes;

public class WarehouseManager extends Employee implements accessCheckOut, accessWarehouse{

    public WarehouseManager(int id, String name, String phoneNo, String role) {
        super(id, name, phoneNo, role);
    }

    @Override
    public void accessCO(int a) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void accessWH() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
}
```

Figure 7: Warehouse_manager class

Cart Items:

```java
/**
public class CartItems {
    private String invoiceID;
    private String productName;
    private int productQuantity;
    private Double unitPrice;
    private Double totalPrice;

    public CartItems(String invoiceID, String productName, int productQuantity, double unitPrice, double totalPrice) {
        this.invoiceID = invoiceID;
        this.productName = productName;
        this.productQuantity = productQuantity;
        this.unitPrice = unitPrice;
        this.totalPrice = totalPrice;
    }
```

Figure 8: CartItems class

In Voice:

```java
/
public class Invoice {
    private Integer saleId;
    private String invoiceId;
    private String customerId;
    private String customerName;
    private int totalQty;
    private Double totalBill;
    private String status;
    private Double balance;
```

Figure 9: Invoice class

IV. Package Design

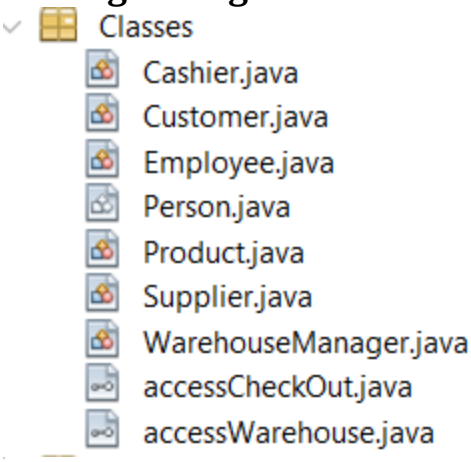

Figure 10.1: Package hierarchy

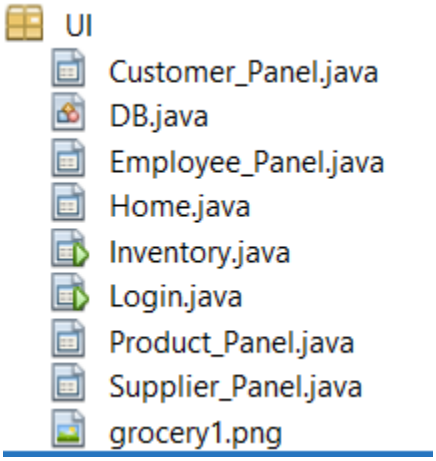

Figure 10.2: Package hierarchy

The packages used in our project split into 2 parts:

Built-in Packages stored in the JRE System Library:

1) **java.lang:** Contains language support classes(e.g. classes which define primitive data types, math operations). This package is automatically imported.

2) **java.io:** Contains classes for supporting input / output operations.

3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.

4) **java.applet:** Contains classes for creating Applets.

5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like buttons, menus, etc).

6) **java.net:** Contain classes for supporting networking operations.

7) **java.sql:** is a package that provides the API for accessing and manipulating relational databases. It is a part of the Java Standard Edition (SE) and is used for database connectivity.

Defined packages are Classes and UI.

- Classes contain Cashier, Customer, Employee, Person, Product, Supplier, WarehouseManager, accessCheckOut and accessWarehouse.
- UI contains Home Frame, Login Frame, Customer's panel, DB(Databases), Product's panel, Employee's panel, Supplier's panel.

V. Interface Design

```
package Classes;

public interface accessCheckOut {
    public void accessCO(int a);
}
```

Figure 11.1: CheckOut interface

```
package Classes;

public interface accessWarehouse {
    public void accessWH();
}
```

Figure 11.2: Warehouse interface

We created 2 interfaces aimed to split access permission for 2 positions of employees.

- **Cashier** can access both *Warehouse* and *CheckOut*
- **Warehouse manager** can only access *warehouse*

VI. Access Control

Table 5: Data Access Control Table

No	Class	Access Control Modifier	Description
1	Person	- private: Id, name, phoneNo - pubic: constructor, getter and setter method.	In summary, the Person class provides encapsulation by using private fields with public getter and setter methods, and it serves as a base class for other classes to inherit from. The describe method, being abstract, enforces that any concrete subclass must provide its own implementation.
2	Cashier	- public constructor, accessCO, accessWH.	The Cashier class represents a type of Employee and implements specific interfaces (AccessCheckOut and AccessWarehouse). It includes placeholder implementations for the methods declared in these interfaces, suggesting that a concrete implementation is expected in subclasses or later stages of development.

			In terms of access control, the class is declared as public, indicating that it is accessible from outside its package. The use of inheritance and interface implementation suggests that the Cashier class is designed to reuse code from the Employee class and provide specific functionality related to checkout and warehouse access through the implemented interfaces.
3	Customer	- public constructor, describe.	The Customer class is a public class that represents a specific type of person, extending the abstract Person class. It provides a concrete implementation of the describe method, offering a description of a customer's name, ID, and phone number. This class is designed to

			be used when you need to model and describe customers in your application. The use of inheritance allows for code reuse and a consistent structure across different types of people in the system.
4	Employee	- private: role - public constructor, getter and setter method.	The Employee class is a public class representing a specific type of person, extending the Person class. It introduces a new field (role) to capture the role of the employee within the organization. The describe method offers a specific description for an employee, combining information from the Person class with the additional role information from the Employee class. The encapsulation of the role field using private access and public getter/setter methods follows good object-oriented design practices.

5	Product	- private: id, name, category, quantity, inPrice, outPrice. - public constructor, getter and setter method, describe.	The Product class is a public class representing a product, with private fields encapsulating its attributes. The class includes a constructor for initializing instances and getter/setter methods for accessing and modifying the state. The describe method offers a way to print a description of the product, providing a convenient way to display information about a Product object.
6	Supplier	- private id, name, barCode. - public constructor, getter and setter method.	The Supplier class is a public class representing a supplier. It encapsulates the supplier's attributes (id, name, barCode) with private fields and provides controlled access to these fields through getter and setter methods. The constructor initializes the state of a Supplier object when it is created. This class is designed to be used

			when modeling and managing information about suppliers in a system.
7	WarehouseManager	- public constructor, accessWH.	The WarehouseManager class is a public class representing a warehouse manager. It extends the Employee class, inheriting its attributes and methods. Additionally, it implements the AccessCheckOut and AccessWarehouse interfaces, signifying that a WarehouseManager has specific responsibilities related to checkout and warehouse access. The class includes placeholder implementations for the methods declared in the interfaces, suggesting that concrete implementations are expected in subclasses or later stages of development. This

			class is designed to be used when modeling and managing information about warehouse managers in a system.

Table 6: Method Access Control Table

No	Method	Class	Modifier
1	Constructor Getter() Setter() describe()	Person	public
2	Constructor accessCO() accessWH()	Cashier	public
3	Constructor describe()	Customer	public
4	Constructor Getter() Setter() describe()	Employee	public
5	Constructor Getter() Setter() describe()	Product	public

6	Constructor Getter() Setter()	Supplier	public
7	Constructor accessWH()	WarehouseManager	public
8	Constructor() Getter() Setter()	CartItems	public
9	Constructor() Getter() Setter()	InVoice	public

VII. Encapsulation vs Inheritance vs Polymorphism

1. Encapsulation

- Encapsulation is the practice of keeping fields within a class *private*, so that they can only be manipulated through methods in their own class. It's a protective barrier that prevents the data from being accessed directly by methods from outside the class.
- Take the *Customer* class as an example from *Figure 2*. It has a private field which is *customer_id*. This means that it can not be accessed by any methods unless they are declared within its own class like *ViewProduct(), MakePayment(), AddToCart()*, and *DeleteFromCart()*.

2. Inheritance

- Inheritance allows us to define a new class based on an existing one, inheriting its attributes and methods, and allowing us to add new ones or modify the existing ones.

- For example in *Figure 2*, two subclasses *Payment* and *Cart* extend the superclass *Customer*, and they inherit all attributes and methods from their superclass. Also, we can add more specific attributes like *card_type* and *card_no* for *Payment*, and *cart_id* for *Cart*.

3. Polymorphism

- Polymorphism allows us to perform a single action in different ways. In Java, we use method '*overloading*' and method '*overriding*' to achieve polymorphism.
    + Method '*overloading*': This occurs when two or more methods in the same class have the same name but different parameters.
    + Method '*overriding*': This occurs when a subclass provides a specific implementation of a method that is already provided by its superclass. The method in the subclass must have the same name, return type, and parameters as the one in its superclass.

- We take the *Supplier* and *Product* classes as an example in *Figure 2*. We can see that both classes have the *description()* function. The *Supplier* class might have a method of displaying its shipping information. However, for the *Product* class, it inherits the *description()* function, but it prints out its product information instead.
- This is an example of method '*overriding*', since the *description()* method is polymorphic because it behaves differently depending on whether it's called on an object of type *Supplier* or an object of type *Product*.

VIII. Experiment

1. Environment and Tools

a. Environment:

- The code is created from Netbeans, one of the common Java environments, and we use 3 different PCs to code each section of code.
- After that, we connect the codes together and test it many times to make some adjustments.

b. Tools:

- Here is a list of tools used in this project:
    + MySQL Connector.
    + MySQL Workbench.
    + MySQL Shell.
    + MySQL Server.
    + Swing.

- *Swing* is already existed inside the Java environment, so to declare it, we just need to type:

```
import javax.swing.*;
```

2. Project functions

Employee:

- Look for the name of the employee.
- Update the Employee's Panel view.
- Keep the new Employee's details safe.
- Update the employee's details.
- Delete the employee's details.

Product:

- Look for the name of the Product.
- Update the Product's Panel view.
- Keep the new Product's details safe.
- Update the Product's details.
- Delete the Product's details.

Customer:

- Look for the name of the Customer.
- Update the Customer's Panel view.
- Keep the new Customer's details safe.
- Update the Customer's details.
- Delete the Customer's details.

Supplier:

- Look for the name of the Supplier.
- Update the Supplier's Panel view.
- Keep the new Supplier's details safe.
- Update the Supplier's details.
- Delete the Supplier's details.

3. Database

We use MySQL Workbench for storing data of grocery store:

cartId	INID	productName	qty	unitPrice	totalPrice
1	1	Pizza	2	12	24
2	1	Cola	5	3	15
3	1	mango	5	29	145
4	1	Pizza	3	12	36
5	1	fish	5	15	75
6	1	Cola	10	3	30
7	1	Pizza	3	12	36
8	1	Cola	10	3	30
9	1	fish	5	15	75
10	1	Cola	10	3	30
11	1	Pizza	5	12	60
12	1	mango	2	29	58
13	1	Cola	10	3	30
14	1	Pizza	5	12	60
15	1	mango	2	29	58

Figure 12.1: Cart.csv

customerId	customerName	customerPhoneNo
1	Khoa	1234112
3	Pix	1234111
4	Essmus	334115
5	Trixy	0238122
6	Bixs	09921
NULL	NULL	NULL

Figure 12.2: Customer.csv

employeeId	employeeName	phoneNo	role
1	Peterr	098823	Cashier
4	Essmu	334115	Warehouse Manager
5	Pinky	000111	Cashier
6	Blackin	1112222	Cashier
10	Pete	098823	Cashier
11	Khoa	01409147	Warehouse Manager
NULL	NULL	NULL	NULL

Figure 12.3: Employee.csv

	exid	val
▶	1	12

Figure 12.4: Extra.csv

	productId	productName	category	quantity	inPrice	outPrice
▶	1	Milk	Food	100	22	5
	2	Banana	Food	100	11	23
	4	Orange	Fruit	200	8	10
	5	Sandwich	Food	100	5	7
	11	Tea	Drinks	50	10	15
	12	BlackTea	Food	50	10	15
	13	Sausage	Food	3	110	200
	14	mango	Food	155	66	99
	15	fish	Food	177	88	170
	17	Cola	Food	444	79	100
	18	Pizza	Food	277	999	18
*	NULL	NULL	NULL	NULL	NULL	NULL

Figure 12.5: Product.csv

	exid	val
▶	1	4

Figure 12.6: Product.csv

	supplierId	supplierName	barCode
▶	1	ADCcorp	1244
	2	AMDcorp	1277
	4	KCAcorp	1213
	5	BigC	13344
*	NULL	NULL	NULL

Figure 12.7: Supplier.csv

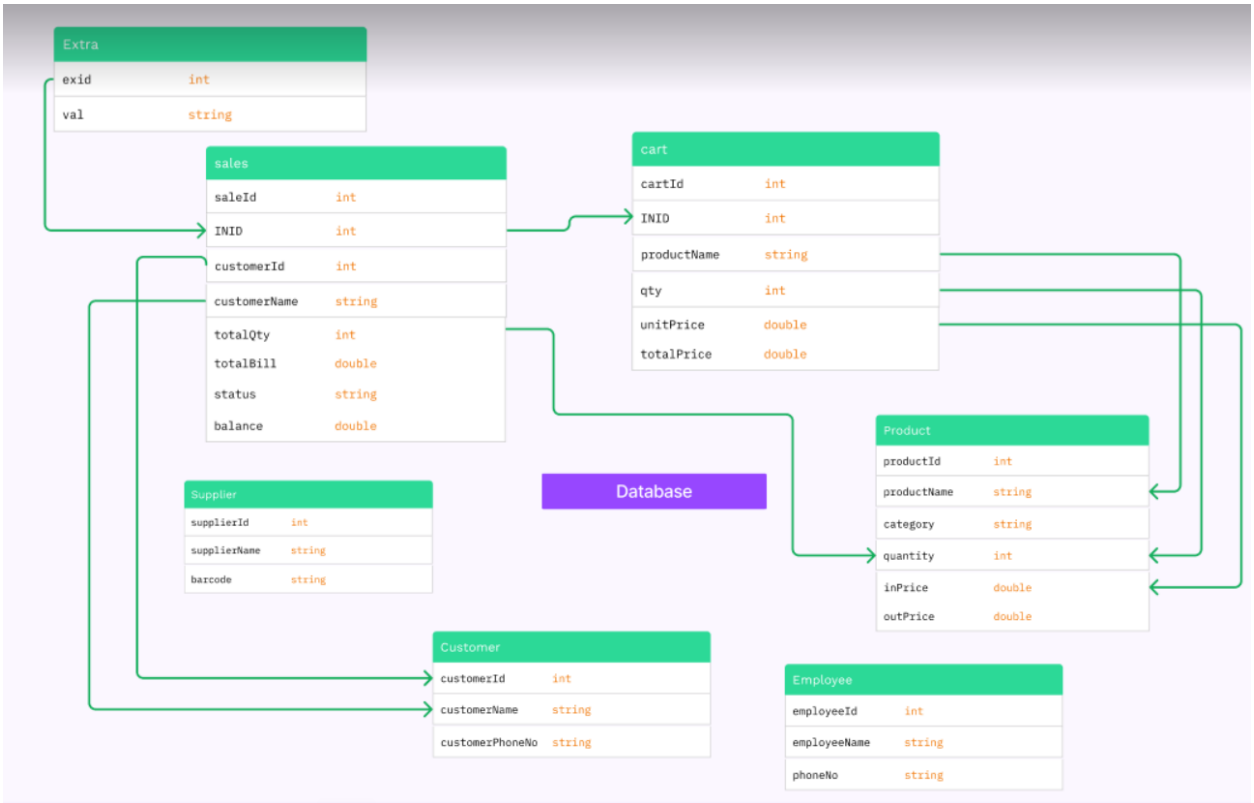

Figure 12.8: Relational database

4. GUI

Interface's Name	Order	Description
Login frame	1	- The required step at which you wish to log into the system.
Home frame	2	- A basic representation of the data you wish to interact with
Employee panel	3	- A view of

		employee data that the manager can edit.
Product panel	4	- A view of Product data that the manager can edit.
Customer panel	5	- A view of Customer data that the manager can edit.
Supplier panel	6	- A view of Supplier data that the manager can edit.
Sales panel	7	view of sales
invoice panel	8	view of invoice
report	9	view of reports

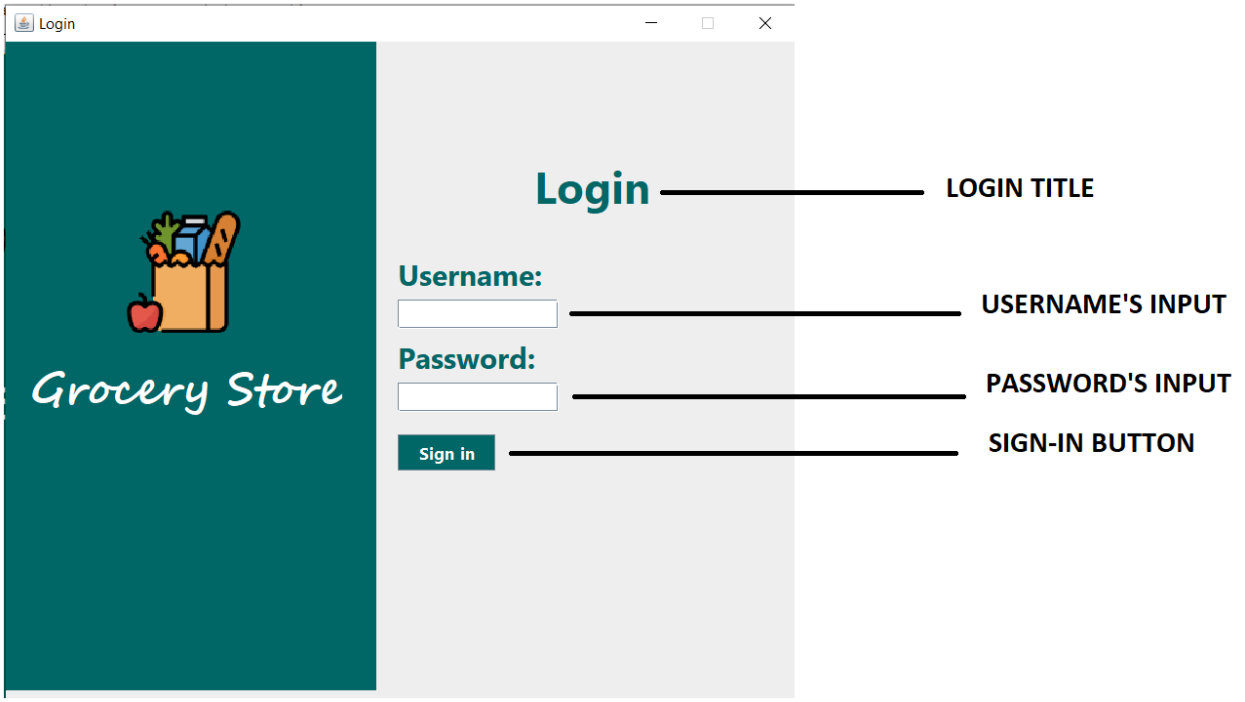

Figure 13: Login Page

Username's input: A username is a special identification that a user selects or is given to set them apart on a certain system. It aids in the system's ability to identify and link certain activities or pieces of data to particular users.
Password's input: A password is a private piece of data linked to a username that acts as an authentication mechanism. Only the user and the system are aware of it.
Sign-in button: A graphical element known as the "login" button is pressed to start the process of sending the password and username input into the system for validation.
The server-side authentication processes are initiated when the login button is clicked. The system then verifies that the password and username entered match the credentials that are on file for a legitimate user. They would appear like this if they didn't match the information:

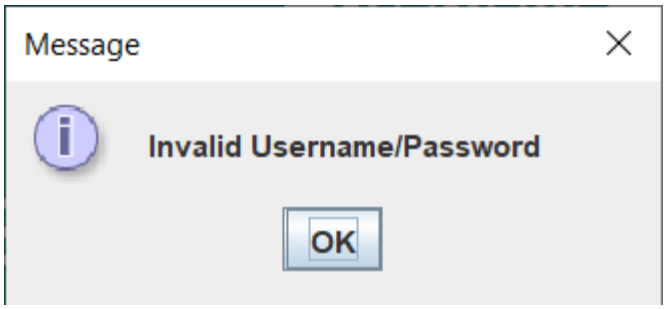

Figure 14: Invalid Username/Password dialog

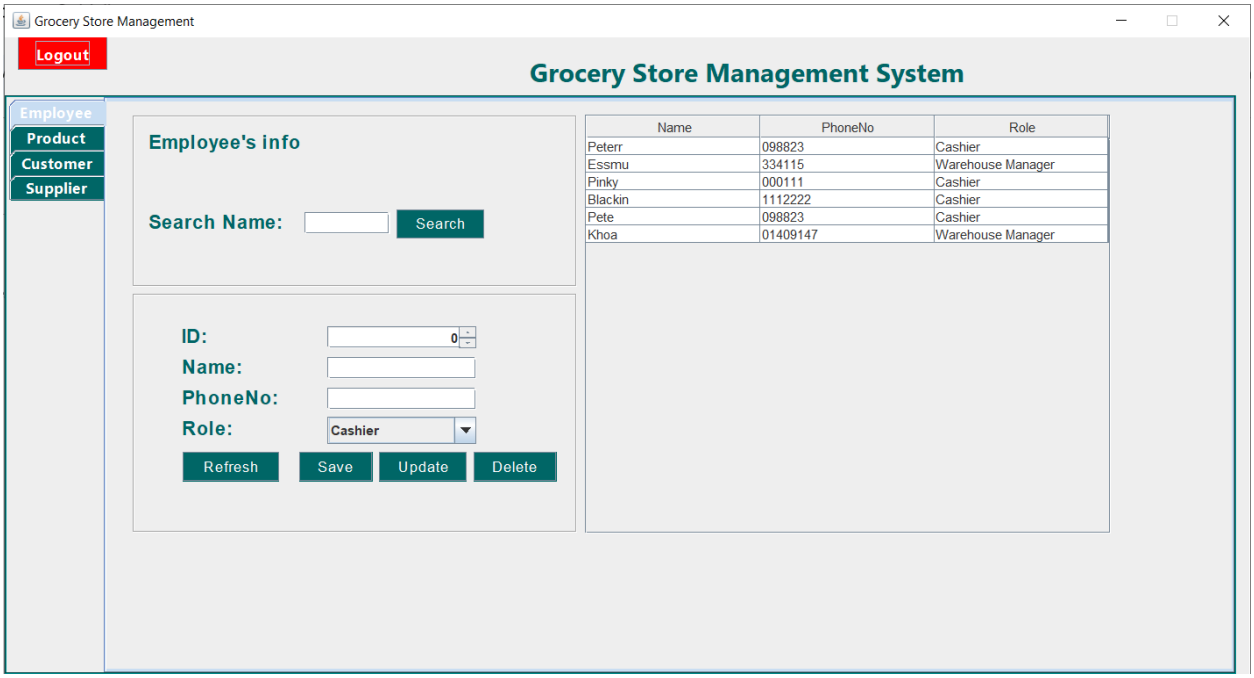

Figure 15: Employee panel

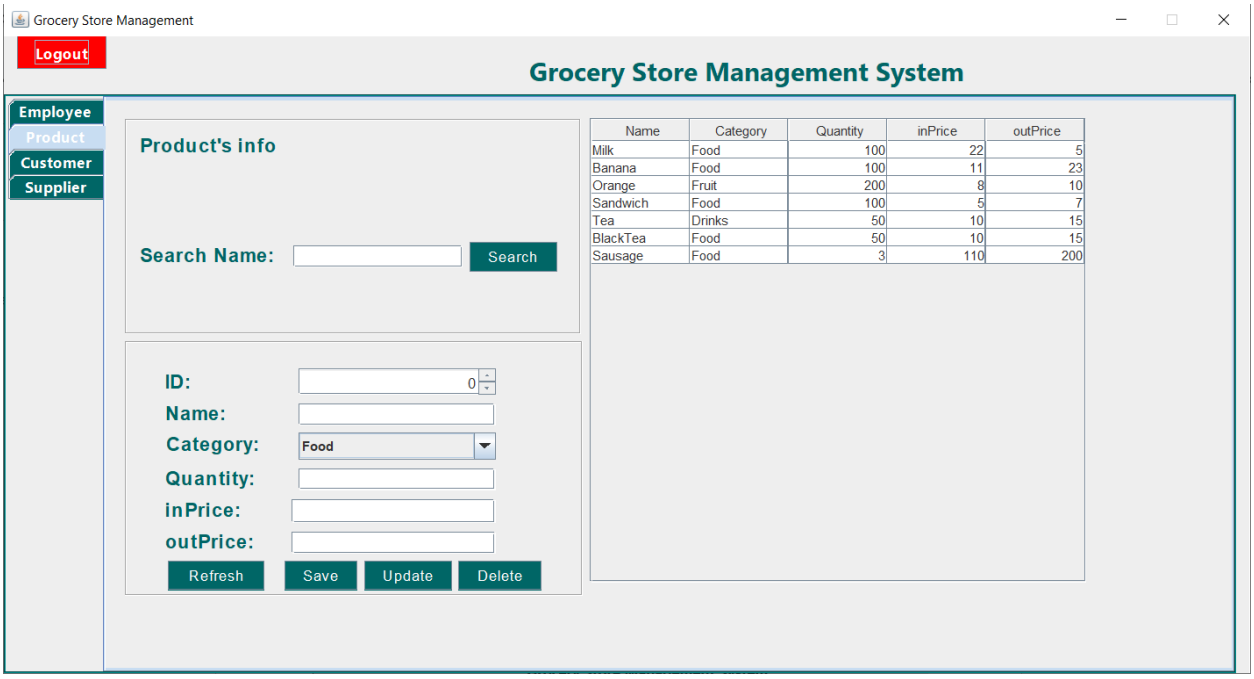

Figure 16: Product panel

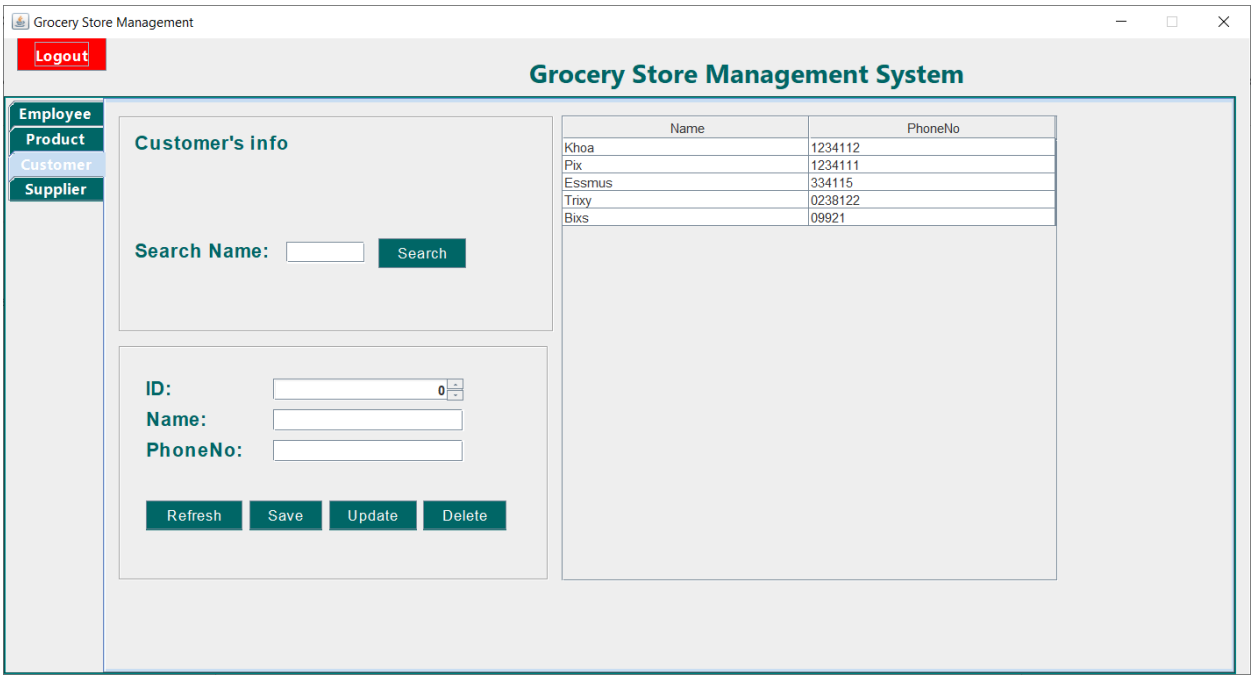

Figure 17: Customer panel

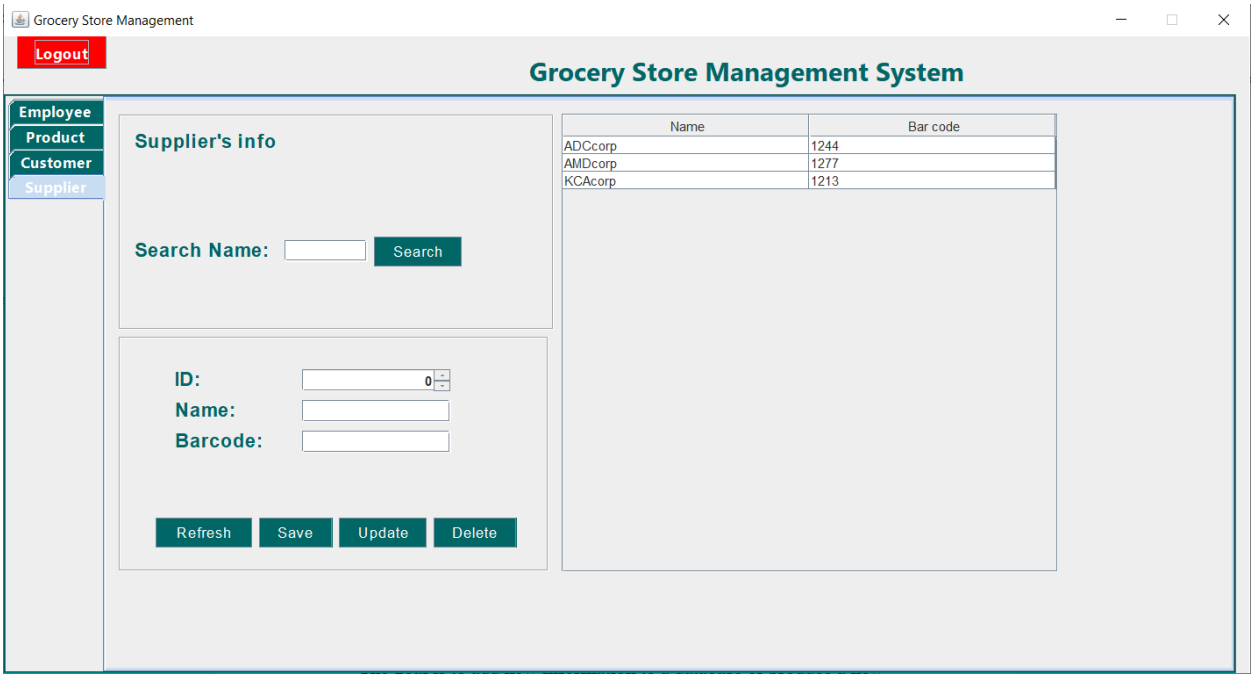

Figure 18: Supplier panel

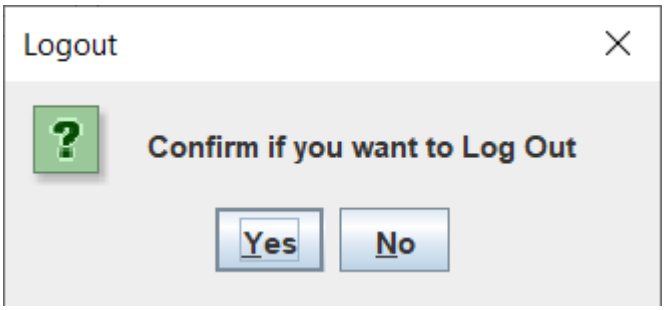

Figure 19: Logout dialog

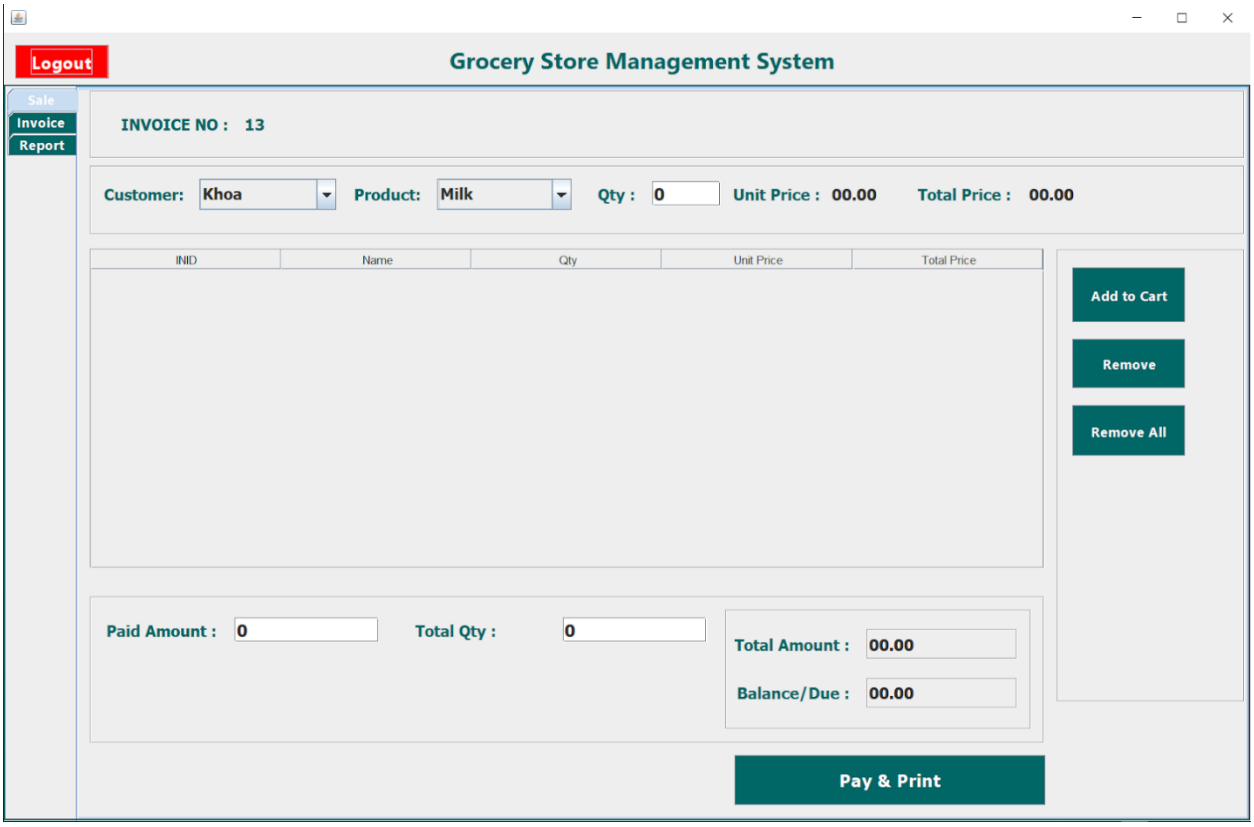

Figure 20: Invoice panel

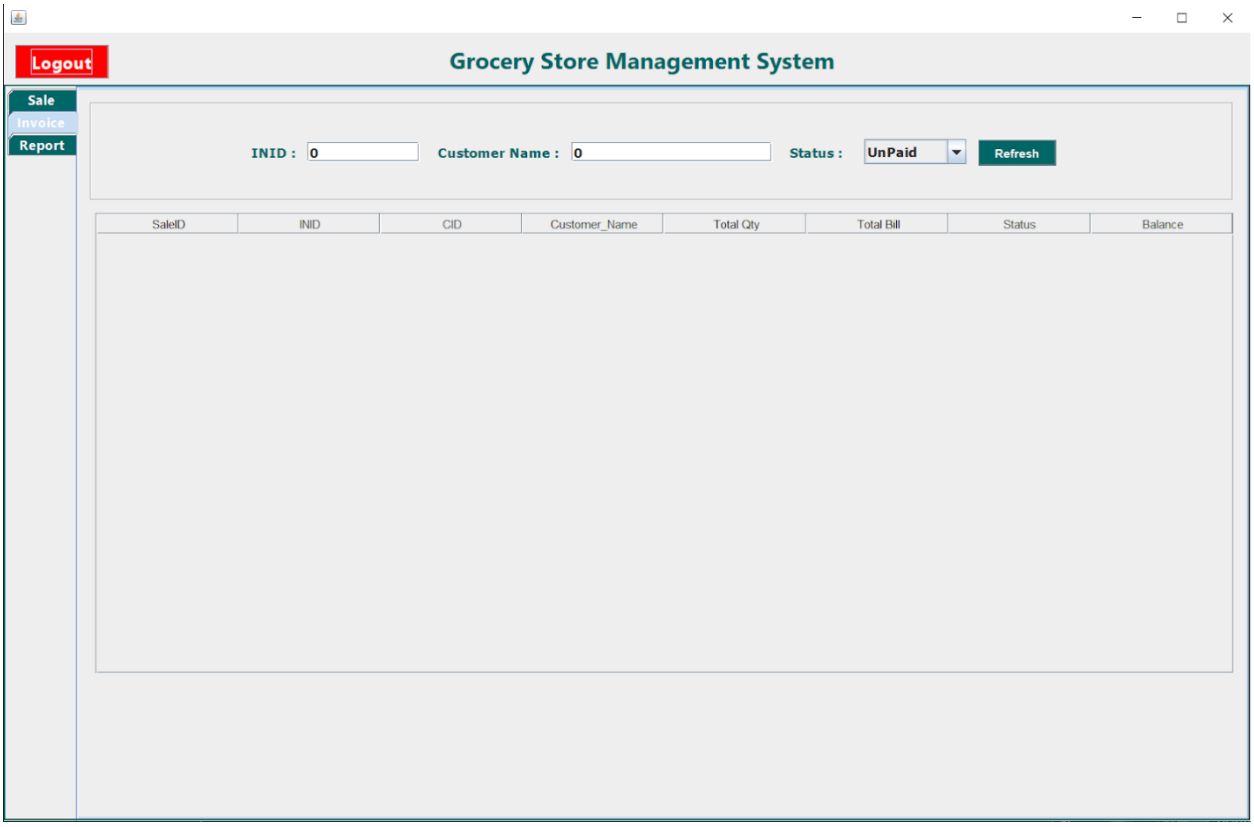

Figure 21: Sales panel

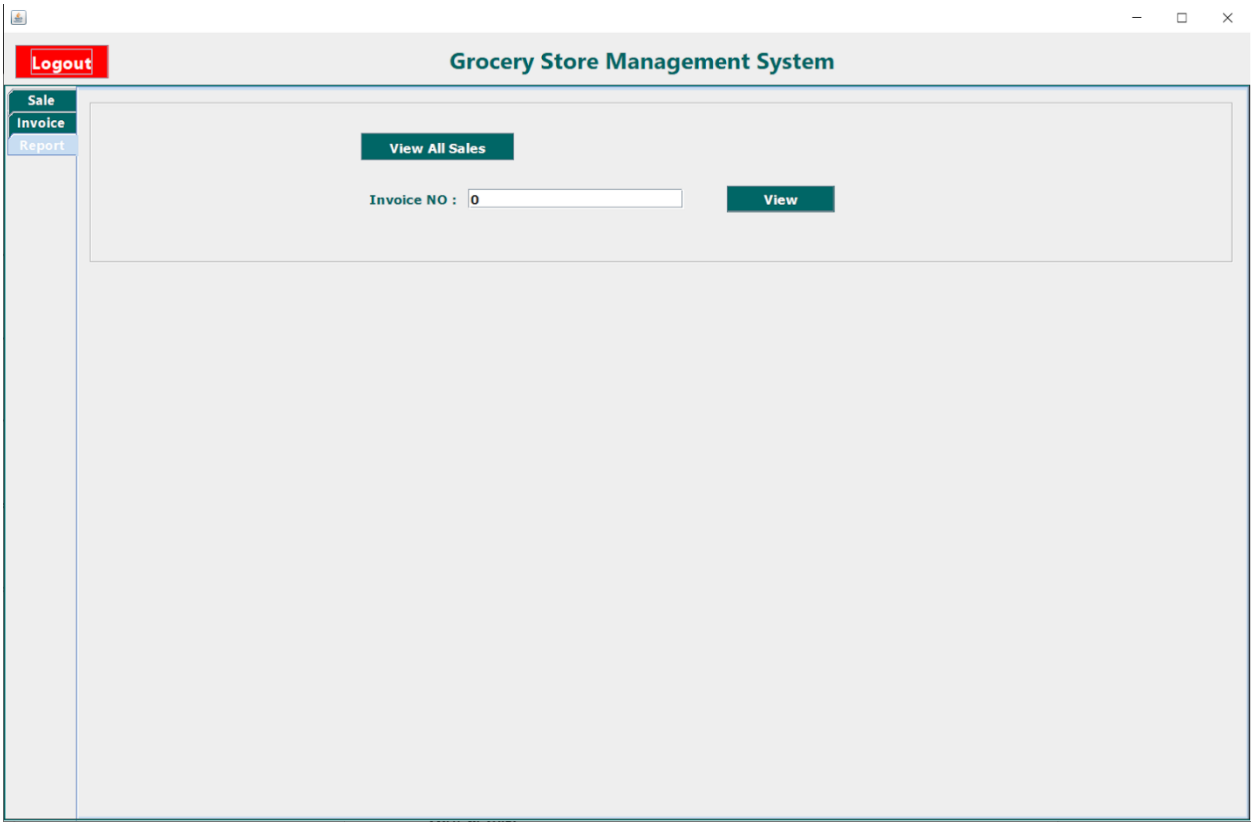

Figure 22: Report panel

Search Function:

- The goal is to get particular data using predetermined criteria from a collection or database.

How to Use:

- Typically, users enter search parameters (filters, keywords, etc.) to locate the needed data.
- After that, the system searches the database to find and show the pertinent information.

Refresh Function:

- The goal is to reload or update the content displayed on the screen, bringing in the latest data or information.

How to Use:

- In this case, you can use the refresh button by locating it in the user interface and clicking or tapping on it.

Save Function:

- The goal is to add new information to a database or produce a new record.
- How to Use:
  + Users enter the data required to create a new record.
  + After that, the system creates a new entry in the database by adding this data.

Update Function:

- The goal is to update a record or alter current data in a database.

How to Use:

- Usually, users pick the record they wish to change.
- The adjustments that needed to be made (such as changing wording or values) were then input.
- The updated data is added to the database by the system.

Delete Function:

- The goal is to delete information or entries from a database.

How to Use:

- Typically, users pick the record or piece of data they wish to remove.
- They attest to the request for deletion.
- After that, the system deletes the designated data from the database.

IX. Conclusion

1. Pros and cons:

- Pros:
  + Easy to set up and use: Because this project is designed with Java, which is a widely-used language with a large community, it helps us find solutions to some of the common problems like keeping track of different products, managing each employee, etc.
  + Stability: With Java, it helps to keep the system code maintain its stability and prevents crashing.
  + Offline functionality: The project can still work, even when there is no internet connection.
- Cons:
  + Configuration of advanced features: The advanced features included in Java may be difficult to configure, which leads to more time consumption when coding.
  + Bugs: The newly made project may contain bugs.

+ Limited functionality: Java may lack some advanced features, which can only be found in other languages, such as certain aspects of inventory management, customer relationship, etc.

2. Comments:

- After many attempts on planning the structure and purpose of the project, and getting it to run correctly, we can tell from the results that the project may lack some advanced features and contain bugs, but it still qualifies the required conditions and necessary aspects that it was supposed to do.
- The display is quite satisfactory, which fits the theme, and the users can easily interact with it. Overall, if this project is used in real life, it would bring a score of 7/10, since it is useful for small grocery stores. However, with bigger grocery stores, it might need more advanced features.

3. Future plans:

- The project is currently a useful system to help users keep track of their records and product information. However, if this project will be used in the future, we may add more class functions such as Navigation, Report, etc.
- Also, we will adjust more security methods to ensure that sensitive data, like customer payment information, is stored with a lower chance of a breach.

DUTY ROSTER

ID	Task	In Charge	Start	End	State	Note
1	Report Section I, II	Pham Phu Tuan Khoa, Vo Pham Khang Huy	16-Oct-23	17-Oct-23	Done	
2	Report Section III	Lý Minh Hùng	21-Oct-23	23-Oct-23	Done	Only finished part 1 (Classes)
3	Diagrams	Pham Phu Tuan Khoa, Nguyen Tien Khoa Vo Pham Khang Huy	19-Oct-23	12-Nov-23	Done	
4	Report Section VII	Nguyen Tien Khoa	31-Oct-23	1-Nov-23	Done	
5	Update Report Section III	Vo Pham Khang Huy	31-Oct-23	1-Nov-23	Done	

6	Report Section IV, V	Pham Phu Tuan Khoa	31-Oct-23	1-Nov-23	Done	
7	Report Section VI	Ly Minh Hung	3-Nov-23	1-Nov-23	Done	
8	Report Section VIII Part 2, 4	Vo Pham Khang Huy	6-Nov-23	12-Nov-23	Done	
9	Report Section VII Part 1, and Section IX	Nguyen Tien Khoa	6-Nov-23	12-Nov-23	Done	
10	Code	Pham Phu Tuan Khoa, Vo Pham Khang Huy	31-Oct-23	12-Nov-23	Done	
11	Update Code	Ly Minh Hung	6-Nov-23	12-Nov-23	Done	Update function for buttons in GUI Product Create and Update function for buttons in

						GUI sales, invoice, reports.
12	Databases	Ly Minh Hung	6-Nov-23	12-Nov-23	Done	
13	Complete Report	Vo Pham Khang Huy, Nguyen Tien Khoa	13-Nov-23	14-Nov-23	Done	(All Sections)

REFERENCES

1. Sample objects: Grocery Management System Project for Final Year (lovelycoding.org)
2. Sample classes: Grocery-Store-Management-System-Database-Design.png (1301×633) (1000projects.org)