



**National University of Sciences and Technology (NUST)**  
**School of Electrical Engineering and Computer Science**

**CS-212: Object Oriented Programming (3+1)**

**SEMESTER PROJECT**

**Submitted To: Dr. Nazia Perwaiz**

**Submitted By:**  
**BESE-14B**

NAME	CMS ID
Ushba Fatima	467212
Anoosheh Arshad	459658
Ahmad Shahmeer	475027



**Smart Cart System**

## ABSTRACT

Stepping into the future of shopping with *ScanDash* – a cutting-edge Smart Cart System programmed using SQL for reliable database management, Java for versatile programming capabilities, and Arduino for seamless hardware integration. With dedicated interfaces for both store managers and customers, streamline inventory management and allow customers to manage their own carts. For customers, enjoy effortless product scanning, real-time bill calculation, and interactive payment simulations. The primary motivation of *ScanDash* is to enhance retail efficiency and optimize the tedious process of inventory management, which will allow for a more seamless customer experience and in turn greater commercial success.

It's not just shopping; it's a seamless, innovative journey.

## INTRODUCTION

*ScanDash* provides real-time transaction tracking and data-driven decision-making, making it an adaptable solution for various retail environments, from small shops to large chains. OOP principles enable a modular design, code reusability, and maintainability. Key OOP concepts include:

- **Encapsulation:** By encapsulating data and functions within classes, *ScanDash* ensures that the internal workings of each component are hidden from other parts of the system. This leads to better data security and simplifies maintenance.
- **Inheritance:** The use of inheritance allows for the creation of generalized classes (such as a generic user class) from which more specific classes (such as manager and customer classes) can be derived. This promotes code reuse and reduces redundancy.
- **Polymorphism:** Polymorphism enables the system to process objects differently based on their data type or class. This is particularly useful in handling different categories of grocery items within the system, allowing for flexible interaction handling.

### Manager Interface

The manager interface in *ScanDash* provides tools for managing inventory and customer accounts. Managers can add, update, and remove products from the inventory. Additionally, they can manage customer data, including addition and removal of customers.

- **Inventory Management:** Utilizing a MySQL database, managers can efficiently track product quantities, categorize items, and manage pricing. This ensures that inventory data is always up-to-date and accessible.

- **Customer Management:** Managers have the ability to create and delete customer profiles. This feature supports customer relationship management (CRM) and enhances the overall customer experience.

### Customer Interface

The customer interface is designed to streamline the shopping process using RFID technology integrated with Arduino. RFID technology is used as an alternative to barcode readers for this project. Customers can easily add items to their cart by scanning RFID tags.

- **RFID Shopping:** RFID readers connected to Arduino allows customers to scan items and add them to their virtual shopping cart. This reduces checkout times and minimizes human error.
- **Database Integration:** Customer data is stored in a database to ensure reusability of their customer cards and create a loyal customer-base.

## PROBLEM STATEMENT

Traditional retail environments suffer from inefficient inventory management, long checkout queues, and high operational costs, leading to reduced customer satisfaction and profitability.

Implementation of a solution to this problem is necessary:

- It allows self-checkout for customers, improving store-customer relationship.
- The increased efficiency in the shopping system will promote the customers to use the technology more considering their fast paced lives.
- The digitalization of the cart system will help stores decrease expenses on sales personnel and for a direct customer to store relationship.
- The cart is self-managed, reducing the hassle of delayed billing.
- The products are divided into numerous categories for easier management and understanding of the product itself based on the specific product description.
- The presence of cart cards for each customers promotes cart security and also helps understand trends in shopping from customers.

We wish to introduce a system which is user-friendly. *ScanDash* is a one for all system and aims to make the shopping experience simpler and reduce disruptions making it a smooth experience

## PROJECT OBJECTIVES

- **Efficient Inventory Management:** Real-time tracking using MySQL databases ensures accurate stock levels and reduces overstocking or stockouts.
- **Transparency of Product Description:** The products are divided into categories based on their nature and have specific characteristics (expiry dates, warranty dates, weight etc) so all the details are displayed to the customer with just one click.
- **Reduced Checkout Times:** RFID technology with Arduino enables quick, self-service checkout and automated bill calculation, eliminating long queues.
- **Cost Reduction:** Automation decreases the need for extensive sales personnel, lowering labor costs and increasing profitability.
- **Enhanced Customer Experience:** User-friendly interfaces and quick transactions improve customer satisfaction and loyalty.
- **Data-Driven Decision Making:** Detailed analytics from the MySQL database allow for informed decisions to optimize sales and inventory.

## METHODOLOGY

### Software Requirements:

- a. Java Development Kit (JDK)
- b. Eclipse IDE with WindowBuilder plugin

Windows Builder 1.13.0: [Eclipse downloads - Select a mirror | The Eclipse Foundation](#)

- c. MySQL database

[MySQL :: MySQL Community Downloads](#) (My SQL Installer for Windows)

- d. MySQL Connector/J 8.3.0

[MySQL :: Download Connector/J](#) (Platform Independent)

- e. FlatLaf - Flat Look and Feel for GUI

[FlatLaf - Flat Look and Feel | FormDev](#)

The general approach utilized while developing the project was to implement an online alternative to the shopping cart system as accurately and fluidly as possible, and provide a user-friendly experience. To that end, many technologies and tools were procedurally implemented and integrated into the program.

It was important to store, retrieve and display data in multiple instances in an organized, tabular manner. Information like the ScanDash Customer and Manager details, Product details and Customer's Cart needed to be gathered in one place so that it can be observed and analyzed. This purpose was achieved through the use of databases. For this project, a single database was created, consisting of different tables that stored information related to the certain requirement. The database software used for this project is MySQL. To connect the MySQL database to the Java Project, the following code was used, which accesses the JDBC driver, an application programming interface (API) for the Java which defines how a client may access a database. For local implementations, a separate .jar file called MySQL Connector was also installed and added as a Library to the project:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DBConnect {
    static Connection con = null;

    // Method to establish connection to the database
    public static Connection ConnectToUserDB() {
        try {
            // Establish connection to the MySQL database named "library" on localhost at port
3306
            // with username "root" and password "MySQL@2773778"
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/user", "root",
"MySQL@2773778");
        } catch (SQLException ex) {
            // Log any SQLException that occurs during the connection attempt
            Logger.getLogger(DBConnect.class.getName()).log(Level.SEVERE, null, ex);
        }
        return con; // Return the established connection (may be null if connection fails)
    }

    public static Connection ConnectToProductDB() {
        try {
            // Establish connection to the MySQL database named "library" on localhost at port
3306
            // with username "root" and password "MySQL@2773778"
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/products", "root",
"MySQL@2773778");
        } catch (SQLException ex) {
            // Log any SQLException that occurs during the connection attempt
            Logger.getLogger(DBConnect.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```

        return con; // Return the established connection (may be null if connection fails)
    }

    public static Connection ConnectToCartDB() {
        try {
            // Establish connection to the MySQL database named "library" on localhost at port
3306
            // with username "root" and password "MySQL@2773778"
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/cart", "root",
"MySQL@2773778");
        } catch (SQLException ex) {
            // Log any SQLException that occurs during the connection attempt
            Logger.getLogger(DBConnect.class.getName()).log(Level.SEVERE, null, ex);
        }
        return con; // Return the established connection (may be null if connection fails)
    }
}

```

Mainly SQL Script in the MySQL command line was used for the creation of the database and database tables.

After achieving database connectivity, it was important to make the user interface simple and elegant, for an enhanced user experience. To implement this, the Java Swing API was utilized to provide the GUI and make the system interactive and presentable. To improve the appearance of our program output further, the Java Look and Feel utility, a part of Swing, was employed. In particular, an external custom Look and Feel named FlatLaf was used: it is a modern open-source cross-platform Look and Feel for Java Swing desktop applications. To do so the .jar file for FlatLaf Look and Feel was installed and added as a Library to the project. The import statement `import com.formdev.flatlaf.FlatDarkLaf;` allows for the use of all the available tools for this Look and Feel. The following code then implements the FlatLaf Look and Feel on all Swing GUI components used, using the UIManager class:

```

try {
    UIManager.setLookAndFeel(new FlatDarkLaf());
} catch (Exception ex) {
    System.err.println("Failed to initialize LaF");
}

```

## SYSTEM ARCHITECTURE

1. **User:** Represents a fundamental entity within the system, encompassing both customers and store managers.
  - **authenticate:** Validates the user's identity based on the provided username, granting access to system functionalities.
2. **Customer (subclass of User):** Embodies an individual utilizing the Smart Cart system for shopping.
  - **authenticate:** An overridden method that validates the customer's identity based on their assigned card number and username.
  - **scanProduct:** Allows customers to effortlessly scan desired products, to add/remove them into their digital shopping cart.
  - **addProduct:** Facilitates the addition of selected products to the customer's virtual shopping cart by specifying product quantity, ensuring a personalized shopping experience.
  - **removeProduct:** Enables customers to remove unwanted items from their digital cart, ensuring convenience during the shopping process.
  - **displayBill:** Provides customers with a overview of their current shopping expenses.
  - **checkout:** Initiates the checkout process, culminating in the completion of the transaction and the customer's exit from the smart cart system.
3. **StoreManager (subclass of User):** Represents a key managerial role responsible for overseeing operations within the retail environment.
  - **authenticate:** An overridden method that authenticates the store manager based on the username and password
  - **manageInventory:** Method to allow the store manager to manage the inventory, such as adding new products, updating quantities, removing items or adding any discount to a product.
  - **addUser:** Grants store managers the ability to onboard new users into the system by providing them with cards to access the smart cart.
  - **removeUser:** Facilitates the removal of existing users from the system.



#### 4. Cart

Responsible for managing the product database within the Smart Cart system, enabling seamless interaction and transaction processing.

- **addProductToDB:** Adds a new product entry to the cart database with the specified name, price, and quantity, ensuring accurate inventory management.
- **removeProductFromDB:** Removes a product from the database of cart based on its name, facilitating flexible cart management and inventory control.
- **calculateBill:** Computes the total bill amount based on the products stored in the cart database and their respective quantities and discounts, providing customers with accurate billing information.
- **clearCart:** Resets the cart database, clearing all entries after the completion of a successful transaction.
- **displayCart:** Displays the contents of the cart database, allowing customers to review their selected products and quantities before finalizing their purchase.

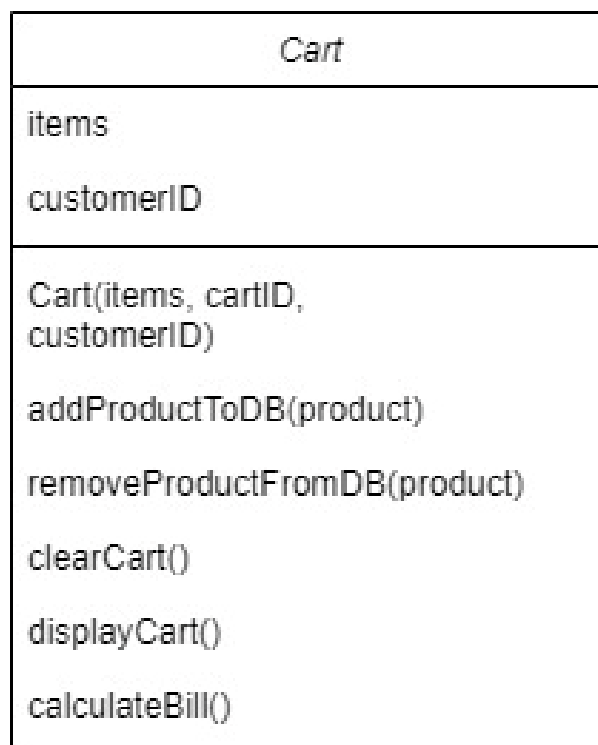
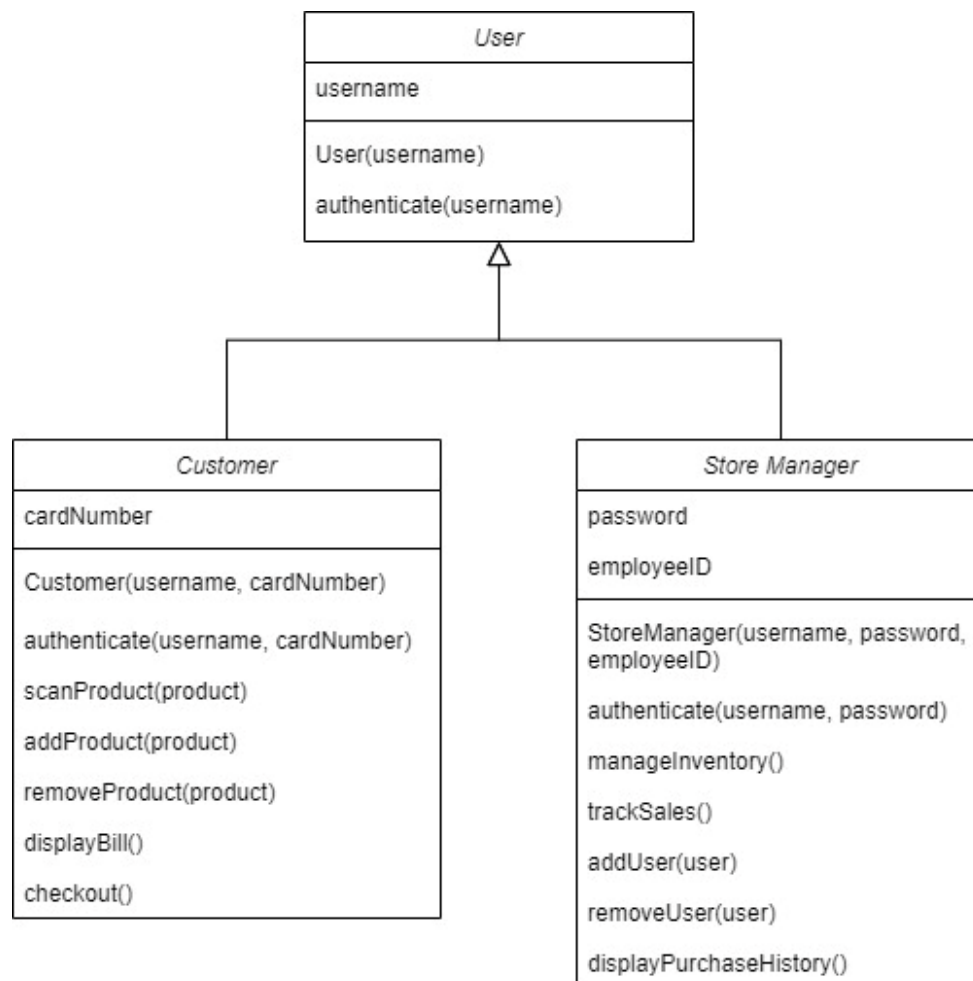
#### 5. Product: The Product class serves as a parent class for various types of products, providing a foundational structure and common attributes. Key attributes and functionalities of the Product class include:

- **Name:** Represents the name of the product.
- **Product ID (ProdID):** A unique identifier assigned to each product for tracking and management purposes.
- **Price:** Denotes the price of the product.
- **Quantity:** Represents the quantity of the product available in the inventory.
- **Discount:** Specifies any discount applied to the product's price, if applicable.
- **Category:** Indicates which category (subclass) the product belongs to.

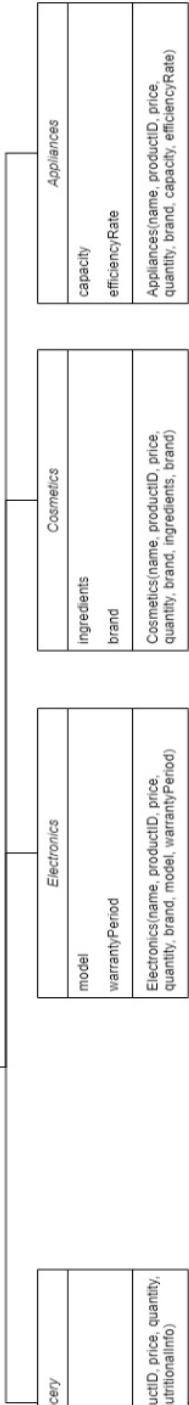
#### 6. Grocery(subclass of Product): Represents grocery items including both fresh and packaged items. Additional attributes include **expiryDate** and **nutritionalInfo**.

#### 7. FreshGrocery (subclass of GroceryProduct): Represents products fresh grocery items. Additional attributes include **weight**.

8. **FreshProduce (subclass of FreshGrocery)**: Represents items such as fruits and vegetables. Additional attributes include **organic** (indicates if product is organic or not).
9. **BakeryItems (subclass of FreshGrocery)**: Represents all bakery items like bread, croissants etc. Additional attributes include **gluterFree** (indicates if the product has gluten present or not).
10. **PackagedProduct (subclass of GroceryProduct)**: Represents all other packaged products. Additional attributes include **brand**.
11. **ElectronicsProduct (subclass of Product)**: Represents electronic items such as smartphones, laptops, and TVs. Additional attributes include **model** and **warranty period**.
12. **CosmeticsProduct (subclass of Product)**: Represents cosmetic or beauty products such as skincare and makeup. Additional attributes include **brand** and **ingredients**.
13. **ApplianceProduct**: Represents household appliances such as refrigerators, washing machines, and microwaves. Additional attributes may include **energy efficiency rating** and **capacity**.



Product	
name	
productID	
price	
quantity	
discount	
category	
Product(name, productID, price, quantity, brand)	



PackedGrocery	
brand	
PackedGrocery(name, productID, price, quantity, brand, expiryDate, nutritionalInfo, brand)	



FreshGrocery	
weight	
FreshGrocery(name, productID, price, quantity, brand, expiryDate, nutritionalInfo, weight)	

BakeryItems	
type	
BakeryItems(name, productID, price, quantity, brand, expiryDate, nutritionalInfo, weight, type, glutenFree)	

FreshProduce	
organic	
FreshProduce(name, productID, price, quantity, brand, expiryDate, nutritionalInfo, weight, organic)	

## DATABASE MANAGEMENT

This component handles the storage and retrieval of data within the system. It includes the following databases:

### 1. Store Manager Database:

- **Fields:** Username, Password
- **Purpose:** Stores authentication credentials for store managers.

### 2. Customer Database:

- **Fields:** Customer Name, Card Number
- **Purpose:** Stores information about customers, including their names and associated card numbers for smart cart access.

### 3. Product Database:

- **Fields:** Product Name, Product ID, Price, Available Quantity, Discount.
- **Purpose:** This database manages information about products available in the store, including their names, unique identifiers (IDs), prices, available quantities, and any applicable discounts. It serves as a central repository for product data, facilitating efficient inventory management and pricing strategies.

### 4. Category-Specific Databases:

These databases cater to distinct product categories like groceries, cosmetics, electronics, and appliances, storing category-specific details. This segregation ensures efficient management and tracking of products within each category.

### 5. Cart Database:

- **Fields:** Product Name, Price, Quantity, Discount
- **Purpose:** This runtime database holds information about the products added to the cart during a shopping session. It facilitates real-time calculation of the total bill and is emptied upon successful checkout, ensuring accurate billing and transaction processing.

### 6. Purchase History Database:

- **Fields:** Card ID, Bill Amount, Date, User Name.
- **Purpose:** Stores transactional data to track purchases made using the Smart Cart system, facilitating analysis of customer behavior and system effectiveness in enhancing store operations and sales.

*\*Note:*

*con= connection to database*

*pst= Prepared Statement*

*rs= Result Set*

<i>CartManagement</i>
con pst rs
CartManagement() addProductToCartDB(product) removeProductFromCartDB(product) clearCart() displayCart() calculateBill() checkout() closeResources() getCurrentQuantityInCart(product) isProductInCart(product)

<i>UserManagement</i>
con pst rs
UserManagement() findUser(primaryKey) getManagerFromDB(username) getCustomerFromDB(username) closeResources() authenticateUser(enteredUser, userInDB) addCustomerToDB(customer) removeCustomerFromDB(customer) isValidUsername(username) getAvailableCards()

<i>ProductManagement</i>
con pst rs
ProductManagement() findProductinDB(ID) addProductToDB(product) removeProductFromDB(product) updateProductInDB(product) createProductByCategory(product) copyCommonProductDetails(source, destination) addProductToCategoryDB(product) addGroceryDescription(groceryProduct) addElectronicsDescription(electronicsProduct) addApplianceDescription(applianceProduct) addFreshProduceDescription(freshProduce) addPackedGroceryDescription(packagedProduct) addCosmeticsDescription(cosmeticsProduct) addBakeryItemsDescription(bakeryItems) addFreshGroceryDescription(freshGrocery) findProductinCategoryDB(product) getApplianceProduct(applianceProduct) getFreshProduce(freshProduce) getFreshGroceryProduct(freshGrocery) getPackedProduct(packagedGroceryProduct) getCosmeticsProduct(cosmeticsProduct) getElectronicsProduct(electronicsProduct) getGroceryProduct(groceryProduct) closeResources()

## IMPLEMENTATION

After integrating all the necessary tools and technologies, the implementation of the Project involved three main aspects: the operation of Arduino to scan products, the storing and retrieving of data from the database tables, and the procedural building of the user interface and windows.

For the user interface, a separate .java file was made for each of the major windows. Each of these .java files implement the functionality of that particular window. These files are then connected together through the use of buttons so one can move from one window to the other, based on the practicality. The user is initially greeted with a window with the ScanDash logo and a loading sequence: this window was implemented in Progress.java. The following code snippet shows the implementation of a progress bar to give the loading effect:

```
progressBar = new JProgressBar();
progressBar.setStringPainted(true);
progressBar.setForeground(new Color( r: 102, g: 153, b: 204));
progressBar.setBounds( x: 52, y: 304, width: 397, height: 25);
contentPane.add(progressBar);
```

```
try {
    for (x = 0; x <= 100; x++) {
        Progress.progressBar.setValue(x);
        Thread.sleep( millis: 30);
        if (x == 100) {
            Thread.sleep( millis: 1000);
            frame.setVisible(false);
            stopMusic();
            FirstWindow firstWindow = new FirstWindow();
            firstWindow.setVisible(true);
            FirstWindow.playMusic( filePath: "C:\\Users\\hp\\IdeaProjects\\Semester Project\\src\\Nintendo Wii - Shop Channel Music (Extended) HQ.wav");
        }
    }
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

The program then moves to FirstWindow.java which is the starting point of the ScanDash interface. Here the user is given two options as buttons to choose between Customer and Manager. Clicking each button opens the respective window (Customer.java and Manager.java respectively). This was implemented using Action Listeners on the respective JButtons:



```

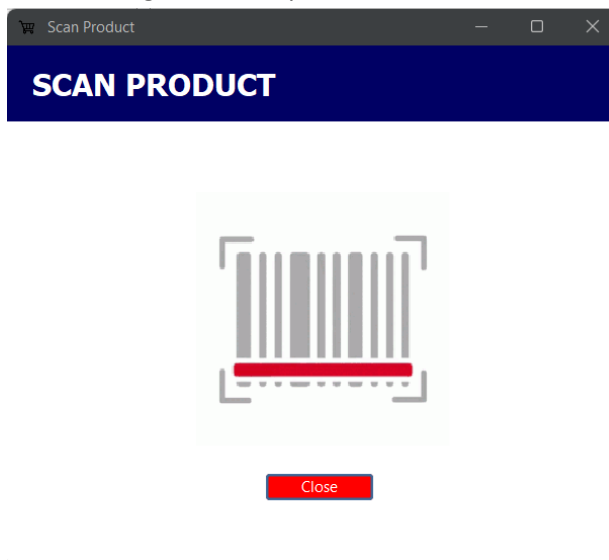
JButton btnManager = new JButton( text: "Manager");
btnManager.setForeground(Color.white);
btnManager.setBounds( x: 180, y: 220, width: 125, height: 25);
btnManager.setBackground(new Color( r: 0, g: 0, b: 100));
btnManager.setFocusPainted(false);
btnManager.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stopMusic();
        // Add code to open Manager window here
        new ManagerWindow().setVisible(true);
        FirstWindow.this.dispose();
    }
});
contentPane.add(btnManager);

JButton btnCustomer = new JButton( text: "Customer");
btnCustomer.setForeground(Color.white);
btnCustomer.setBounds( x: 180, y: 260, width: 125, height: 25);
btnCustomer.setBackground(new Color( r: 0, g: 0, b: 100));
btnCustomer.setFocusPainted(false);
btnCustomer.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stopMusic();
        // Add code to open Customer window here
        new CustomerWindow().setVisible(true);
        FirstWindow.this.dispose();
    }
});
contentPane.add(btnCustomer);

```

A similar methodology is applied for other windows wherever relevant.

Certain visual animations are displayed on the screen during certain parts of the program, like when the user is scanning their card and when the user has to pay the bill. This was done by importing .gif files which are added to the project. The following is an example:



Certain key aspects were implemented in each respective window, including manipulation of the database tables, error checking techniques, and different layouts for resizability of certain windows.

## Encapsulation

**Concept:** Encapsulation is the mechanism of restricting access to certain details of an object and only exposing essential features. It protects the object's integrity by preventing external code from accessing or modifying its internal state directly.

### **Implementation:**

- The **User** class uses private fields (**username**) and provides public getter and setter methods to control access to the username.

```
public abstract class User {
    private String username;

    // Constructors
    public User() {
    }

    public User(String username) {
        this.username = username;
    }

    // Getter for username
    public String getUsername() {
        return username;
    }

    // Setter for username
    public void setUsername(String username) {
        this.username = username;
    }
}
```

## Inheritance

**Concept:** Inheritance is a mechanism where a new class inherits properties and behaviors (fields and methods) from an existing class. It promotes code reusability and establishes a relationship between the parent and child classes.

### **Implementation:**

- The **Grocery** class extends the **Product** class, inheriting its properties and methods while adding new attributes (**expiryDate** and **manufactureDate**).

```
class Grocery extends Product {
    private String expiryDate;
    private String manufactureDate;

    // Constructor
    public Grocery(String name, String prodID, double price, int quantity, double discount, String expiryDate, String manufactureDate) {
        super(name, prodID, price, quantity, discount, "Grocery");
        this.expiryDate = expiryDate;
        this.manufactureDate = manufactureDate;
    }
}
```

## Abstraction

**Concept:** Abstraction is the process of hiding the complex implementation details and showing only the necessary features of an object. It simplifies interaction with the object by providing a clear interface.

**Implementation:**

- The **Arduino** class abstracts the functionality of scanning a product by providing a high-level **scanProduct** method, hiding the file operations and line counting.

```
public class Arduino {
    public static String scanProduct() {
        System.out.println("Hi From Arduino Scan Product");
        String filePath = "test.txt"; // specify your file path here

        // Initial line count
        int initialLineCount = countLines(filePath);
        int newLineCount = initialLineCount;

        // Loop until the line count increases
        while (true) {
            newLineCount = countLines(filePath);
            if (newLineCount > initialLineCount) {
                break;
            }
        }

        // Get and print the last Card UID
        String lastCardUID = getCardUID(filePath);
        return lastCardUID;
    }

    private static int countLines(String filePath) {
        int lines = 0;
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            while (br.readLine() != null) {
                lines++;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return lines;
    }

    private static String getCardUID(String filePath) {
        String lastCardUID = null;
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.startsWith("Card UID: ")) {
                    lastCardUID = line.substring(10).trim();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return lastCardUID;
    }
}
```

## Casting

**Concept:** Casting in Java refers to converting one data type into another. It is often used to treat an object of one type as another type, usually within an inheritance hierarchy.

### Implementation:

- The **createProductByCategory** method casts **Product** objects to specific subclasses based on the product category.

```
public static Product createProductByCategory(Product product) {
    Product resultProduct;
    System.out.println(product.getCategory()); // Using println for better
    visibility

    switch (product.getCategory()) {
        case "Electronics":
            resultProduct = new ElectronicsProduct();
            break;
        case "Fresh Produce":
            resultProduct = new FreshProduce();
            break;
        case "Bakery Items":
            resultProduct = new BakeryItems();
            break;
        case "Packaged Product":
            resultProduct = new PackagedProduct();
            break;
        case "Cosmetics":
            resultProduct = new CosmeticsProduct();
            break;
        case "Appliances":
            resultProduct = new ApplianceProduct();
            break;
        case "Fresh Grocery":
            resultProduct = new FreshGrocery();
            break;
        case "Grocery":
            resultProduct = new Grocery();
            break;
        default:
            // For unknown categories, return a generic Product instance
            resultProduct = new Product();
            break;
    }

    // Copy common details only if a specific subclass instance was created
    if (!resultProduct.getClass().equals(Product.class)) {
        copyCommonProductDetails(product, resultProduct);
    }

    return resultProduct;
}

public static Boolean addProductToCategoryDB(Product product) {
    if (product instanceof ElectronicsProduct) {
        return addElectronicsDescription((ElectronicsProduct) product);
    }
}
```

```

    } else if (product instanceof ApplianceProduct) {
        return addApplianceDescription((ApplianceProduct) product);
    } else if (product instanceof FreshProduce) {
        return addFreshProduceDescription((FreshProduce) product);
    } else if (product instanceof BakeryItems) {
        return addBakeryItemsDescription((BakeryItems) product);
    } else if (product instanceof PackagedProduct) {
        return addPackedGroceryDescription((PackagedProduct) product);
    } else if (product instanceof CosmeticsProduct) {
        return addCosmeticsDescription((CosmeticsProduct) product);
    } else if (product instanceof FreshGrocery) {
        return addFreshGroceryDescription((FreshGrocery) product);
    } else if (product instanceof Grocery) {
        System.out.println("hiii");
        return addGroceryDescription((Grocery) product);
    } else {
        return false; // Unsupported product type
    }
}

```

## Exception Handling

**Concept:** Exception handling is a mechanism to handle runtime errors, allowing a program to continue execution or terminate gracefully. It uses **try**, **catch**, and **finally** blocks to manage exceptions.

**Implementation:**

- The **findProductinDB** method uses a **try-catch-finally** block to handle SQL exceptions when querying the database.

```

public static Product findProductinDB(String ID) {
    Product product = null;
    try {
        String query = "SELECT * FROM products WHERE prod_id = ?";
        pst = con.prepareStatement(query);
        pst.setString(1, ID);
        rs = pst.executeQuery();

        if (rs.next()) {
            product = new Product();
            product.setProdID(rs.getString("prod_id"));
            product.setName(rs.getString("product_name"));
            product.setPrice(rs.getDouble("price"));
            product.setQuantity(rs.getInt("quantity"));
            product.setDiscount(rs.getDouble("discount"));
            product.setCategory(rs.getString("category"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        closeResources();
    }
}

```

```
        return product;
    }
```

## Interfaces

**Concept:** Interfaces define a contract of methods that implementing classes must provide. They allow for polymorphism by letting different classes implement the same set of methods.

**Implementation:**

- The **AddDescriptionWindow** interface declares methods **isEmptyFields** and **validateDescriptions** that must be implemented by any class that implements this interface.

```
import javax.swing.JButton;

public interface AddDescriptionWindow {
    JButton addDescriptionbtn = new JButton();
    public Boolean isEmptyFields();
    public Boolean validateDescriptions();
}
```

## Composition

**Concept:** Composition is a design principle where a class is composed of one or more objects of other classes, allowing for more flexible and modular code reuse.

### **Implementation:**

- The **CartManagement** class uses composition by holding a connection (**con**) to a database, allowing it to manage cart items.

```
public class CartManagement {
    private static Connection con = DBConnect.ConnectToCartDB();
    private static PreparedStatement pst;
    private static ResultSet rs;

    public CartManagement() {
        // Initialize the connection, statement, and result set to null
        con = null;
        pst = null;
        rs = null;
    }

    public static Boolean addProductToCartDB(Product product) {
        String querySelect = "SELECT * FROM cart_items WHERE prod_id = ?";
        String queryUpdate = "UPDATE cart_items SET quantity = quantity + ? WHERE
prod_id = ?";
        String queryInsert = "INSERT INTO cart_items (prod_id, prod_name, price,
quantity, discount) VALUES (?, ?, ?, ?, ?)";

        PreparedStatement pst = null;
        ResultSet rs = null;

        try {
            // Check if the product exists in the cart
            pst = con.prepareStatement(querySelect);
            pst.setString(1, product.getProdID());
            rs = pst.executeQuery();

            if (rs.next()) { // If the product exists in the cart
                pst.close();
                pst = con.prepareStatement(queryUpdate);
                pst.setInt(1, product.getQuantity());
                pst.setString(2, product.getProdID());
                pst.executeUpdate();
            } else { // If the product doesn't exist in the cart
                pst.close();
                pst = con.prepareStatement(queryInsert);
                pst.setString(1, product.getProdID());
                pst.setString(2, product.getName());
                pst.setDouble(3, product.getPrice());
                pst.setInt(4, product.getQuantity());
                pst.setDouble(5, product.getDiscount());
                pst.executeUpdate();
            }
        }
    }
}
```

```

    }

    return true; // Product added successfully
} catch (SQLException e) {
    e.printStackTrace();
    return false; // Failed to add product
} finally {
    try {
        if (rs != null) rs.close();
        if (pst != null) pst.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}

```

## Polymorphism

**Concept:** Polymorphism allows objects of different classes to be treated as objects of a common superclass. It supports method overriding and dynamic method invocation.

**Implementation:**

- The **addWindowListener** method demonstrates polymorphism by allowing different implementations of the **windowClosing** method to be called based on the actual type of the **WindowListener**.

```

addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        int response = JOptionPane.showConfirmDialog(
            AddElectronicsDescriptionWindow.this,
            "Do you want to quit Scan Dash?", "Confirm Exit",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE);

        if (response == JOptionPane.YES_OPTION) {
            // Exit the program
            System.exit(0);
        }
        // If the response is NO_OPTION, do nothing and stay in the current
        window
    }
});

```



# ARDUINO X JAVA

## Circuit Design

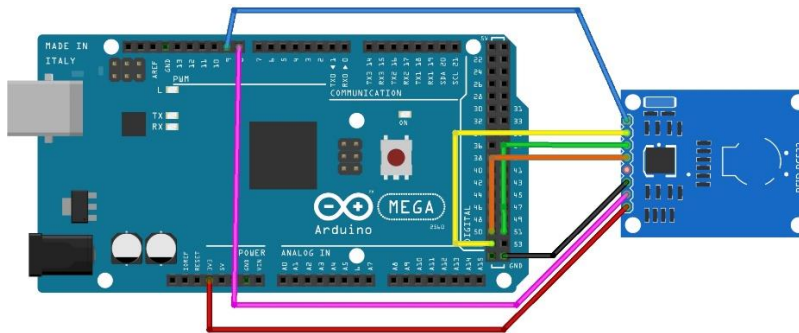
### **Description:**

In this setup, the RFID card sensor serves as an alternative to a barcode reader. The project was originally designed with a barcode reader in mind. However, due to its unavailability, an RFID sensor was used instead. RFID cards function similarly to barcodes, serving as unique identifiers for individual products.

### **Functionality:**

- **Feedback Mechanism:** When an RFID card is scanned by the RFID-RC522 module, the buzzer and LED light produce indications, signaling that a card was successfully scanned. This feedback mechanism ensures that the scanning process is acknowledged by the user.
- **UID Capture:** Upon scanning an RFID card, the RFID-RC522 module captures its unique identifier (UID), facilitating distinct identification.
- **Managerial and Customer Operations:** The UID is used for various operations like inventory management by managers and cart management by customers, allowing efficient addition or removal of products.

### **Circuit Diagram:**



### **Hardware Required:**

- Arduino Mega or Arduino Uno (**I used Mega**)
- RFID-RC522
- jumper wires
- Some ID cards (**optional**)
- Buzzer
- Led Light

### Pin Out for Uno/Nano and Mega

RC522	MODULE	Uno/Nano	MEGA
SDA	D10	D9	
SCK	D13	D52	
MOSI	D11	D51	
MISO	D12	D50	
IRQ	N/A	N/A	
GND	GND	GND	
RST	D9	D8	
3.3V	3.3V	3.3V	

### Arduino and Java Integration

#### Arduino Code:

```
#include <String.h>
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN      5           // Configurable, see typical pin layout above
#define SS_PIN       53          // Configurable, see typical pin layout above
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance

void setup() {
  Serial.begin(9600); // Initialize serial communications with the PC
  while (!Serial);    // Do nothing if no serial port is opened (added for Arduinos
based on ATMEGA32U4)
  SPI.begin();        // Init SPI bus
  mfrc522.PCD_Init(); // Init MFRC522
  delay(4);           // Optional delay. Some board do need more time after init to be
ready, see Readme
  mfrc522.PCD_DumpVersionToSerial(); // Show details of PCD - MFRC522 Card Reader
details
  Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
  pinMode(12, OUTPUT);
}

void loop() {
  // Reset the loop if no new card present on the sensor/reader. This saves the
entire process when idle.
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
    return;
  }
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()) {
    return;
  }
}
```

```
digitalWrite(12, HIGH);  
// Optional delay to keep it high for a certain period  
delay(100);  
digitalWrite(12, LOW);  
// Dump debug info about the card; PICC_HaltA() is automatically called  
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));  
// Convert UID to a string representation  
String uidString = "";  
for (byte i = 0; i < mfrc522.uid.size; i++) {  
    uidString += String(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");  
    uidString += String(mfrc522.uid.uidByte[i], HEX);  
}  
}
```

### Capturing UID Data in a Text File:

CoolTerm serves as an intermediary tool in our project, enabling the seamless capture of Arduino's serial monitor output.

- **Functionality:** CoolTerm is configured to capture the scanned UID of RFID cards in real-time and store it in a text file named "test.txt". We established a serial connection between CoolTerm and the Arduino board, ensuring reliable data transfer.
- **Data Logging:** Scanned UID data is automatically logged by CoolTerm into the designated text file, simplifying subsequent processing.
- **Workflow Integration:** The data captured by CoolTerm seamlessly integrates into our project's workflow, allowing easy access for further processing within the Java program.

Furthermore, leveraging file handling in Java, we can efficiently read the captured UID data from the text file, enabling various operations and functionalities within our application.

### Reading The Data from The Text File In Java:

#### 1. Method Invocation:

- The **scanProduct()** method serves as the entry point for data capture.
- It then specifies the file path where the data is stored (in this case, "test.txt").

#### 2. Initial Line Count:

- The method **countLines()** is used to determine the initial line count of the file.
- This count helps establish a baseline for detecting new data entries.

#### 3. Continuous Monitoring:

- The program enters a loop to continuously monitor the file for new data.
- It compares the initial line count with the current line count (**newLineCount**) until new data is detected.

#### 4. Retrieving Card UID:

- Once new data is found, the **getCardUID()** method is invoked.
- This method reads the file line by line to locate the latest entry where the card UID was written.
- It extracts and returns the UID value from the identified line.

#### 5. Returning Captured Data:

- Finally, the captured UID value is returned to the caller, allowing further processing or utilization within the Java application.

## DATABASES

### Cart Management Database:

- **Purpose:** This database manages the cart items in the system, facilitating efficient handling of product-related data during the shopping process.
- **Tables:**
  - a. **cart\_items:** Stores information about products added to the cart, including product ID, name, price, quantity, and discount. Each product is uniquely identified by its product ID.

```
CREATE DATABASE `cart` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
CREATE TABLE `cart_items` (
  `sr_no` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(100) NOT NULL,
  `prod_name` varchar(255) NOT NULL,
  `price` decimal(10,2) NOT NULL,
  `quantity` int NOT NULL,
  `discount` decimal(5,2) DEFAULT '0.00',
  PRIMARY KEY (`sr_no`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=29 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

### Product Management Database:

- **Purpose:** This database oversees the management of various product categories, ensuring organized storage and retrieval of product details.
- **Tables:**
  - a. **products:** Contains essential details of all products, such as product ID, name, price, quantity, discount, and category. Each product is assigned a unique product ID for identification.
  - b. **grocery\_items:** Stores information specific to grocery products, including product ID, expiry date, and manufacture date. Each entry is uniquely identified by its product ID.
  - c. **fresh\_produce:** Manages data related to fresh produce items, including product ID, expiry date, manufacture date, weight, and organic/inorganic classification. Each entry is uniquely identified by its product ID.
  - d. **packed\_grocery\_items:** Stores details of packed grocery products, such as product ID, expiry date, manufacture date, and brand. Each entry is uniquely identified by its product ID.
  - e. **fresh\_grocery:** Manages information regarding fresh grocery items, including product ID, expiry date, manufacture date, and weight. Each entry is uniquely identified by its product ID.
  - f. **electronics:** Stores data specific to electronics products, including product ID, model, and warranty period. Each entry is uniquely identified by its product ID.
  - g. **cosmetics:** Manages details of cosmetic products, including product ID, ingredients, and brand. Each entry is uniquely identified by its product ID.

- h. **bakery\_items:** Stores information about bakery items, including product ID, expiry date, manufacture date, weight, and gluten information. Each entry is uniquely identified by its product ID.
- i. **appliances:** Manages data related to appliances, including product ID, capacity, and efficiency rate. Each entry is uniquely identified by its product ID.

```
CREATE DATABASE `products` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
```

```
CREATE TABLE `products` (  
  `sr_no` int NOT NULL AUTO_INCREMENT,  
  `prod_id` varchar(255) DEFAULT NULL,  
  `product_name` varchar(255) DEFAULT NULL,  
  `price` decimal(10,2) DEFAULT NULL,  
  `quantity` int DEFAULT NULL,  
  `discount` decimal(5,2) DEFAULT NULL,  
  `category` varchar(50) DEFAULT NULL,  
  PRIMARY KEY (`sr_no`),  
  UNIQUE KEY `prod_id` (`prod_id`)  
) ENGINE=InnoDB AUTO_INCREMENT=30 DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `grocery_items` (  
  `sr_no` int NOT NULL AUTO_INCREMENT,  
  `prod_id` varchar(20) DEFAULT NULL,  
  `expiry_date` varchar(20) DEFAULT NULL,  
  `manufacture_date` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`sr_no`),  
  UNIQUE KEY `prod_id` (`prod_id`)  
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `fresh_produce` (  
  `sr_number` int NOT NULL AUTO_INCREMENT,  
  `prod_id` varchar(20) DEFAULT NULL,  
  `expiry_date` varchar(20) DEFAULT NULL,  
  `manufacture_date` varchar(20) DEFAULT NULL,  
  `weight` varchar(100) DEFAULT NULL,  
  `organic_or_inorganic` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`sr_number`),  
  UNIQUE KEY `prod_id` (`prod_id`)  
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `packed_grocery_items` (
  `sr_no` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
  `expiry_date` varchar(20) DEFAULT NULL,
  `manufacture_date` varchar(20) DEFAULT NULL,
  `brand` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`sr_no`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `fresh_grocery` (
  `sr_number` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
  `expiry_date` varchar(20) DEFAULT NULL,
  `manufacture_date` varchar(20) DEFAULT NULL,
  `weight` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`sr_number`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `electronics` (
  `sr_number` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
  `model` varchar(50) DEFAULT NULL,
  `warranty_period` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`sr_number`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `cosmetics` (
  `sr_number` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
  `ingredients` text,
  `brand` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`sr_number`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `bakery_items` (
  `sr_number` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
```

```

    `expiry_date` varchar(20) DEFAULT NULL,
    `manufacture_date` varchar(20) DEFAULT NULL,
    `weight` varchar(100) DEFAULT NULL,
    `gluten_info` varchar(20) DEFAULT NULL,
    PRIMARY KEY (`sr_number`),
    UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;

```

```

CREATE TABLE `appliances` (
  `sr_number` int NOT NULL AUTO_INCREMENT,
  `prod_id` varchar(20) DEFAULT NULL,
  `capacity` decimal(10,2) DEFAULT NULL,
  `efficiency_rate` decimal(5,2) DEFAULT NULL,
  PRIMARY KEY (`sr_number`),
  UNIQUE KEY `prod_id` (`prod_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;

```

### User Management Database:

- **Purpose:** This database handles user-related information, including customer and manager data, facilitating user authentication and access control.
- **Tables:**
  - a. **customer:** Stores customer details, such as username and card number. Each customer is uniquely identified by their ID, and both username and card number are maintained as unique keys.
  - b. **manager:** Manages manager credentials for authentication and access control, with the username serving as the primary key.
  - c. **scandash\_cards:** Stores information about RFID cards used in the system, including card number and availability status. Each card entry is uniquely identified by its ID, and the

card number is maintained as a unique key.

```

CREATE DATABASE `user` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;

```

```

CREATE TABLE `customer` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `card_number` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`),
  UNIQUE KEY `uc_card_number` (`card_number`)
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;

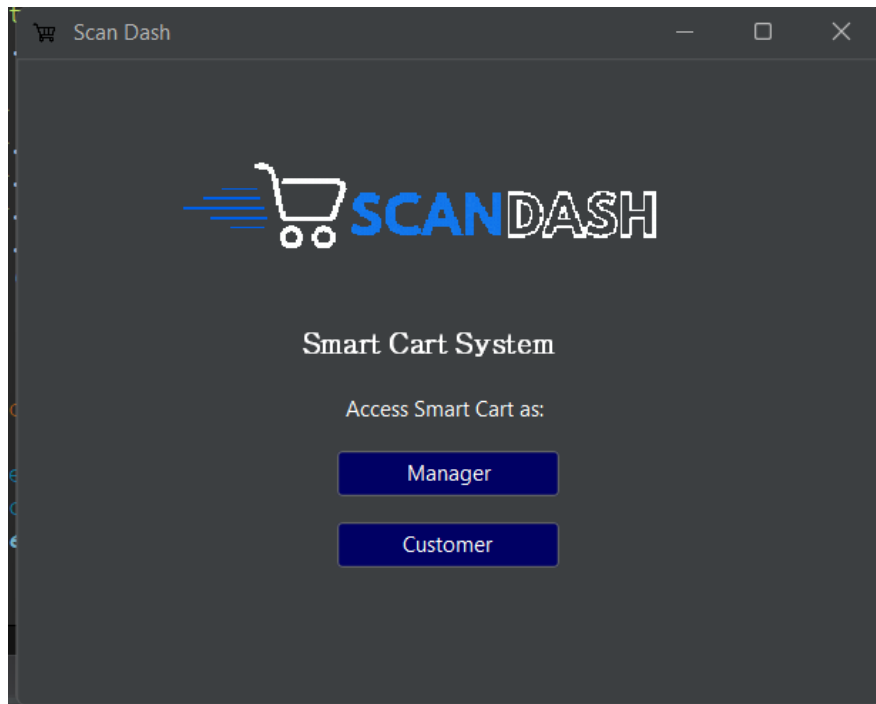
```



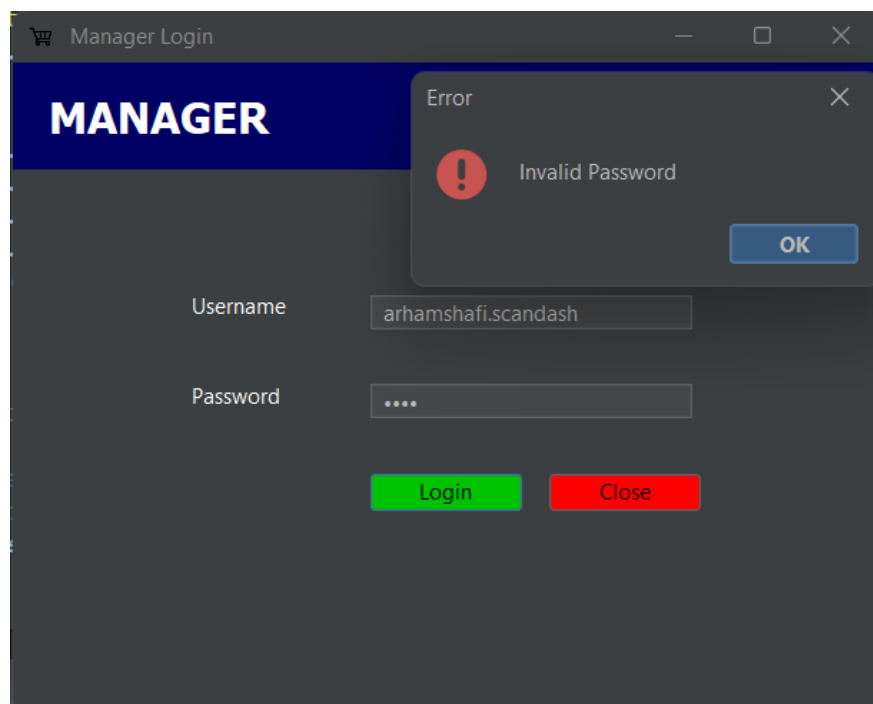
```
CREATE TABLE `manager` (  
  `ID` varchar(20) DEFAULT NULL,  
  `username` varchar(50) NOT NULL,  
  `password` varchar(50) DEFAULT NULL,  
  PRIMARY KEY (`username`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `scandash_cards` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `card_number` varchar(255) DEFAULT NULL,  
  `availability` tinyint(1) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `card_number` (`card_number`)  
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_0900_ai_ci;
```

## RESULTS AND EVALUATION



*First Window displayed to user for choosing user*



*Login window for manager/ error message for incorrect password input*

```
mysql> select * from manager;
```

ID	username	password
1	arhamshafi.scandash	user123
2	charlesdarwin.scandash	user123
3	sufibanaspati.scandash	user123

*Preassigned managers in the database*

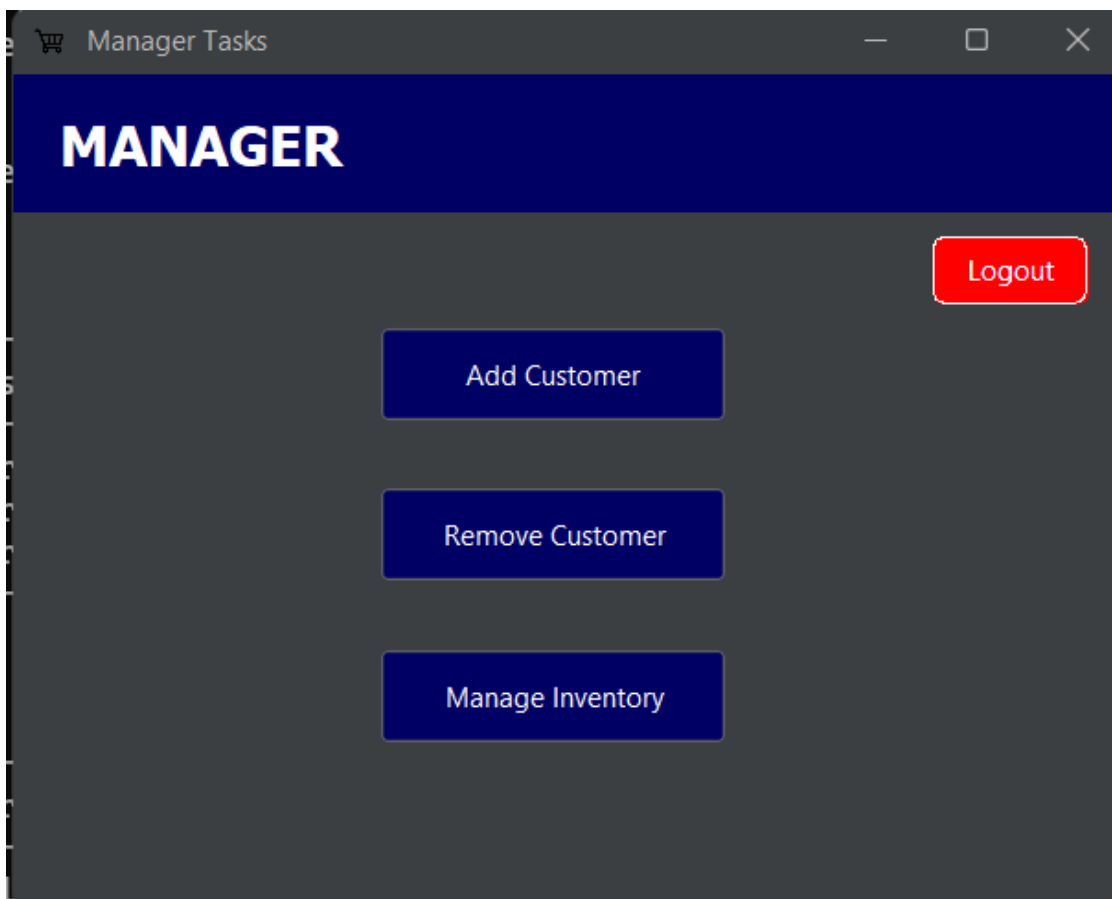
The screenshot shows a 'Customer Login' window with a dark blue header and a dark gray body. The header contains a shopping cart icon and the text 'Customer Login'. Below the header, the word 'CUSTOMER' is displayed in large white letters. The login form has two input fields: 'Username' with the value 'ushba.scandash' and 'Card Number' with the value '1231'. At the bottom of the form are two buttons: a green 'Login' button and a red 'Close' button. An error dialog box is overlaid on the form, titled 'Error', with a red exclamation mark icon and the text 'Invalid Card Number'. The dialog box has a blue 'OK' button.

*Login window for customer/ error shown for invalid card number input*

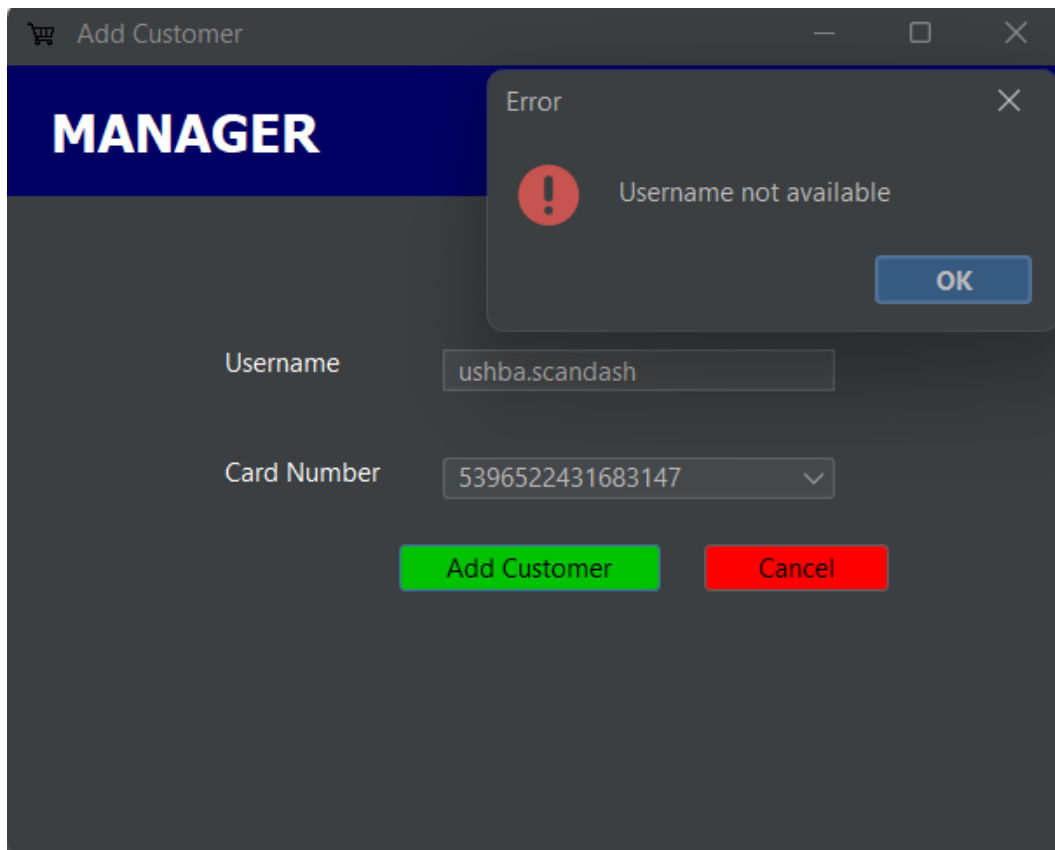
```
mysql> select * from customer;
```

id	username	card_number
2	ushba.scandash	5185790713134221
3	shahmeer.scandash	5140106223523635
4	anoosheh.scandash	5311414367228075
18	yeetzan.scandash	5140995712104471

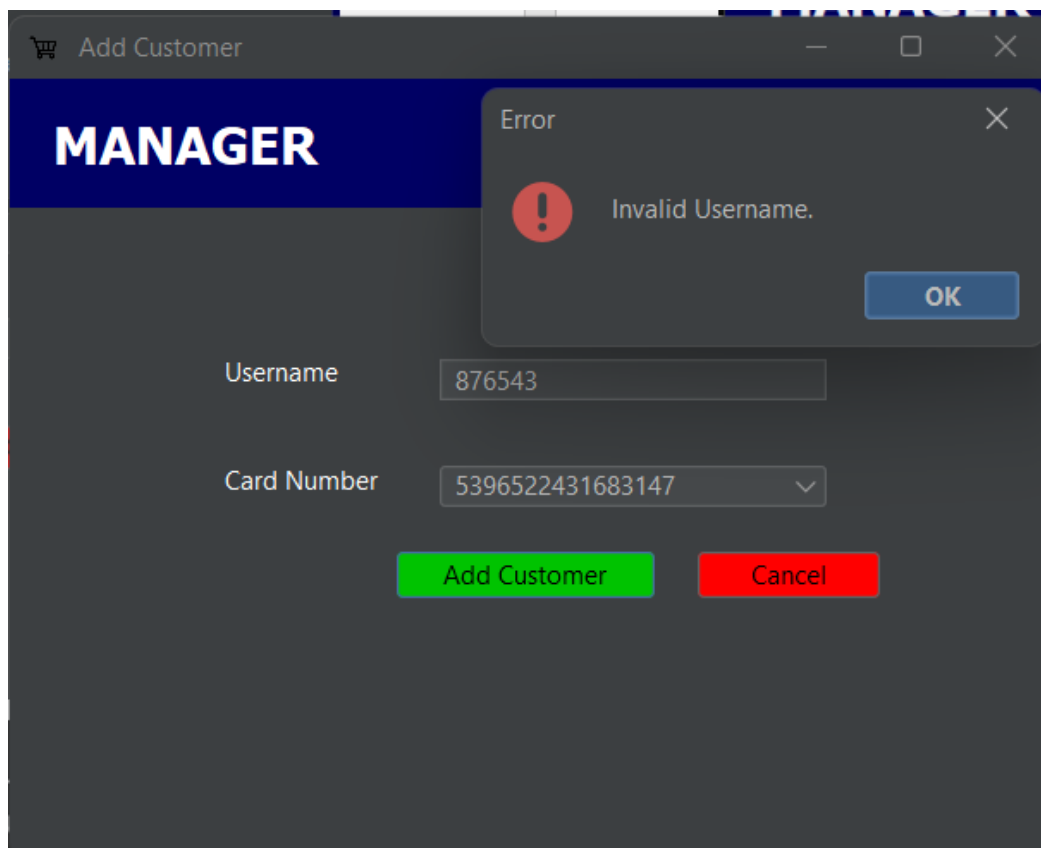
*Preassigned customers in the database*



*Manager tasks window along with logout option*



*Adding new customers/ error shown for invalid username (preexisting usernames and number input)*



Add Customer

MANAGER

Error

Invalid Username.

OK

Username

ali.scan

Card Number

5396522431683147

Add Customer

Cancel

*Adding new customers/error shown when username of format other than .scandash is used*

Add Customer

MANAGER

Username

ali.scandash

Card Number

5396522431683147

5396522431683147

5144355062085224

5581731490133747

4938751224340650

*Adding new customers/ display of all available cards*

```
mysql> select * from scandash_cards;
```

id	card_number	availability
1	5140995712104471	0
2	5185790713134221	0
3	5140106223523635	0
4	5311414367228075	0
5	5396522431683147	1
6	5144355062085224	1
7	5581731490133747	1
8	4938751224340650	1

*Database for all available card numbers*


Add Customer

# MANAGER

Username

Card Number

Saved

 Record Saved

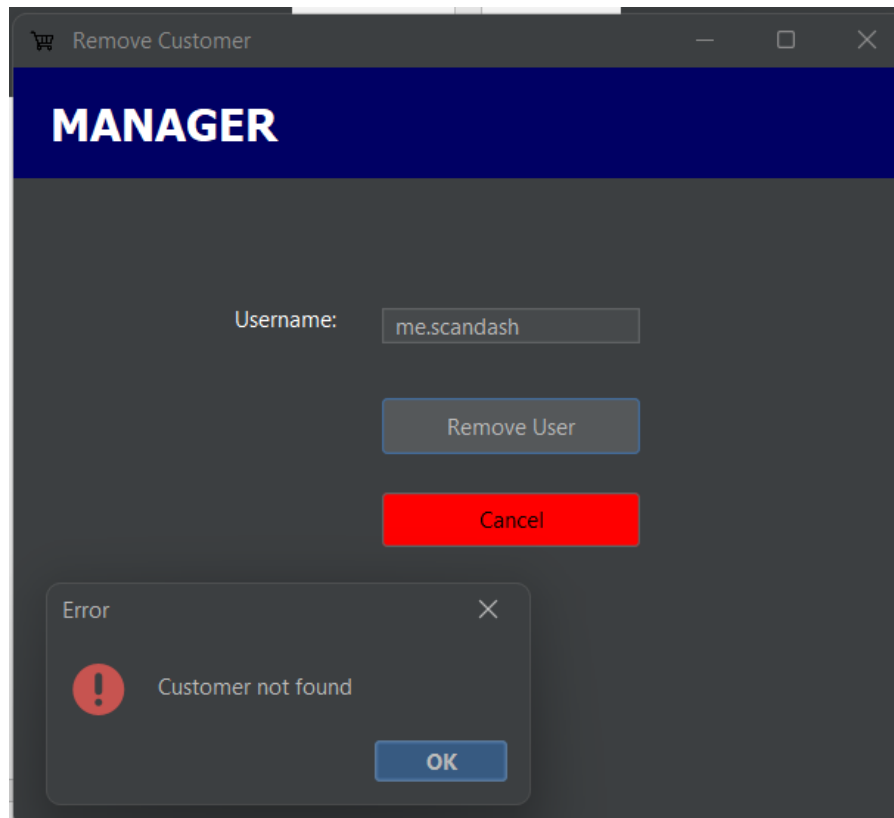
*Adding new customers/ successful addition of customer*

```
mysql> select * from customer;
```

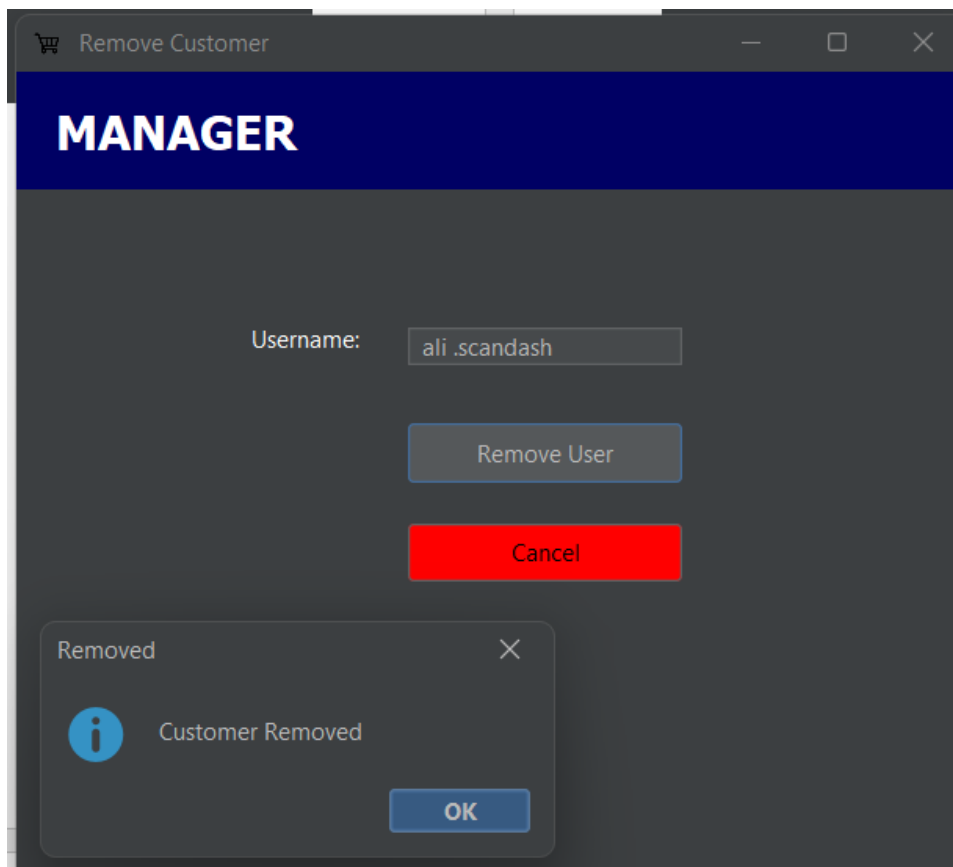
id	username	card_number
2	ushba.scandash	5185790713134221
3	shahmeer.scandash	5140106223523635
4	anoosheh.scandash	5311414367228075
18	yeetzan.scandash	5140995712104471
20	ali.scandash	5396522431683147

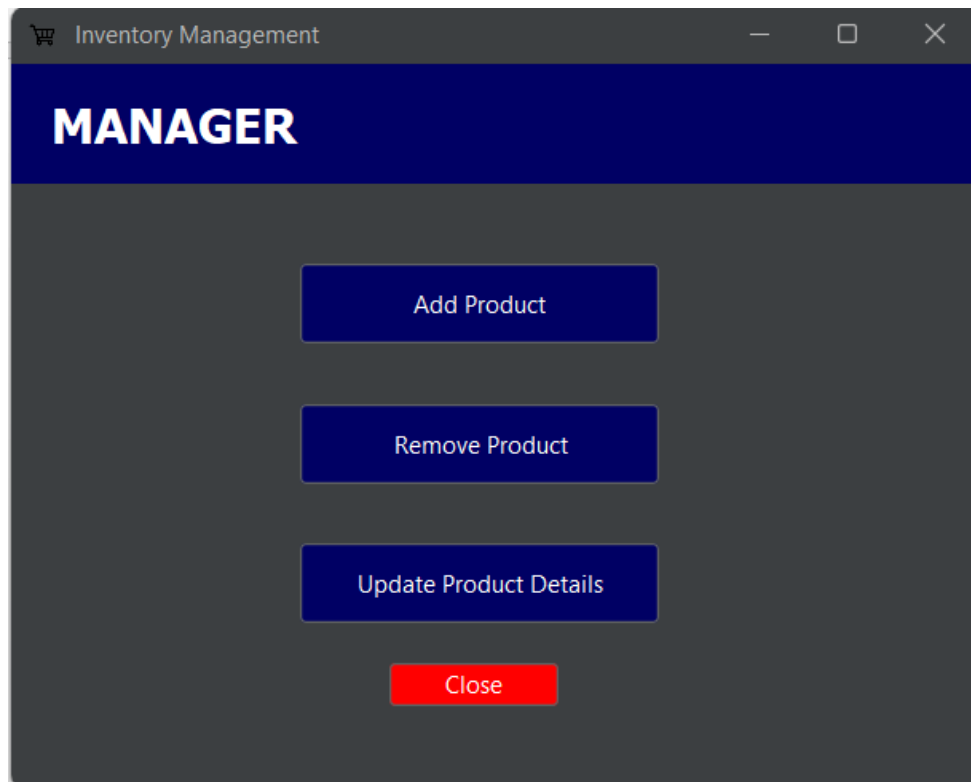
*Updated database showing new customer*



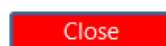


*Removing customer/ error shown when customer not found/ confirmation shown when customer is found*

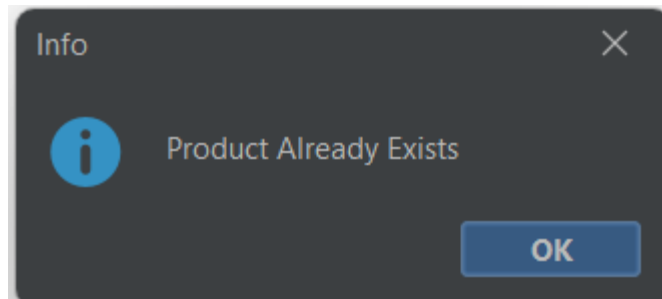




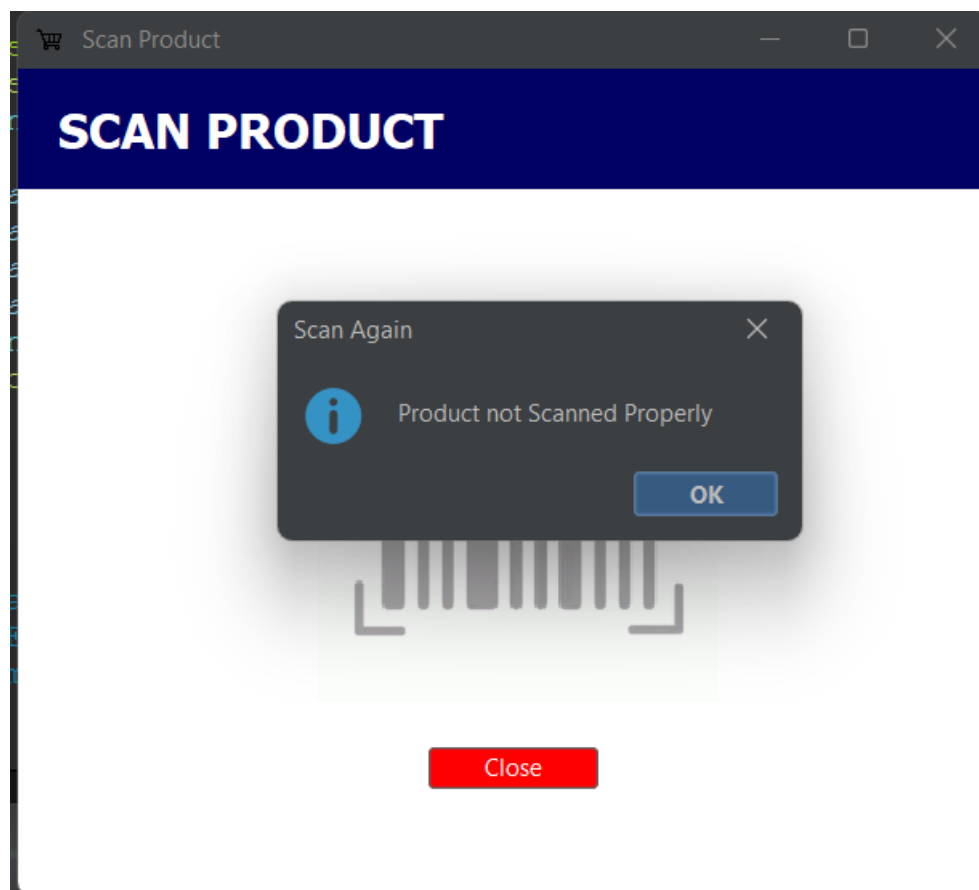
*Inventory management for the manager*



*Scan product window*



*Message appears when product is already in inventory*



*Error shown if product is not scanned accurately*

The screenshot shows a window titled "Add Product" with a dark gray background and a blue header bar containing the word "PRODUCT" in white. The form contains the following fields and controls:

- Product ID:
- Product Name:
- Product Price:  Rs
- Quantity:
- Discount:  %
- Category:  (dropdown arrow)

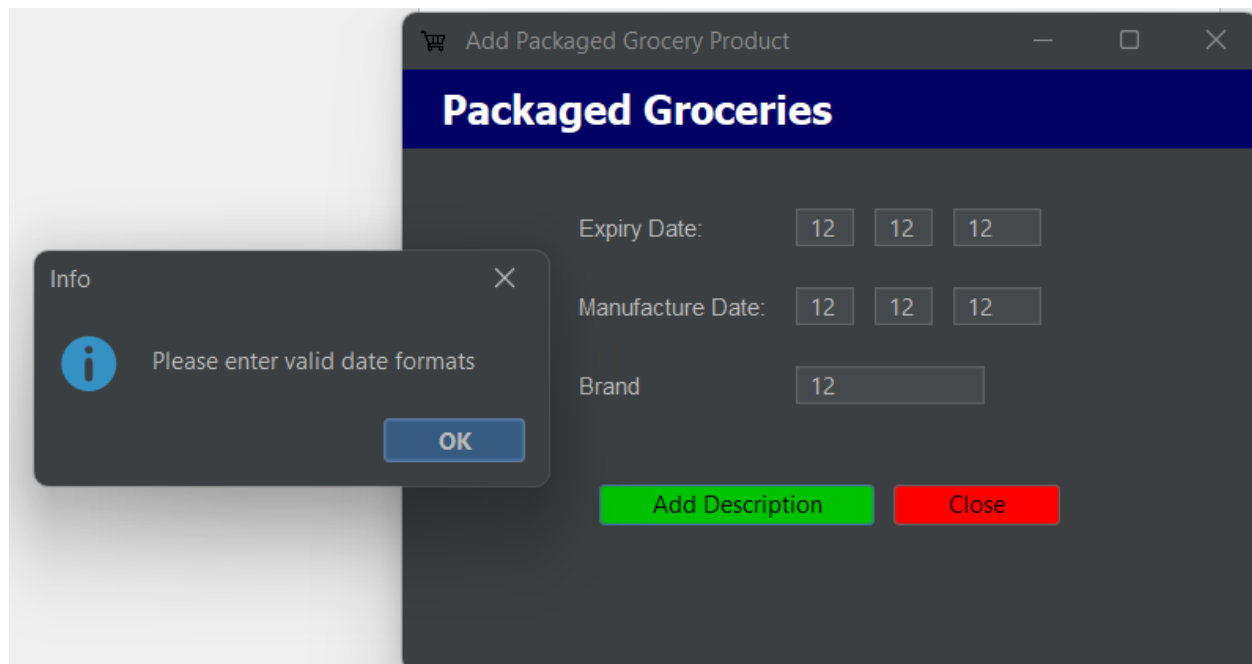
At the bottom, there are three buttons: "Add Product" (green), "Add Description" (green), and "Cancel" (red).

*Adding new product window*

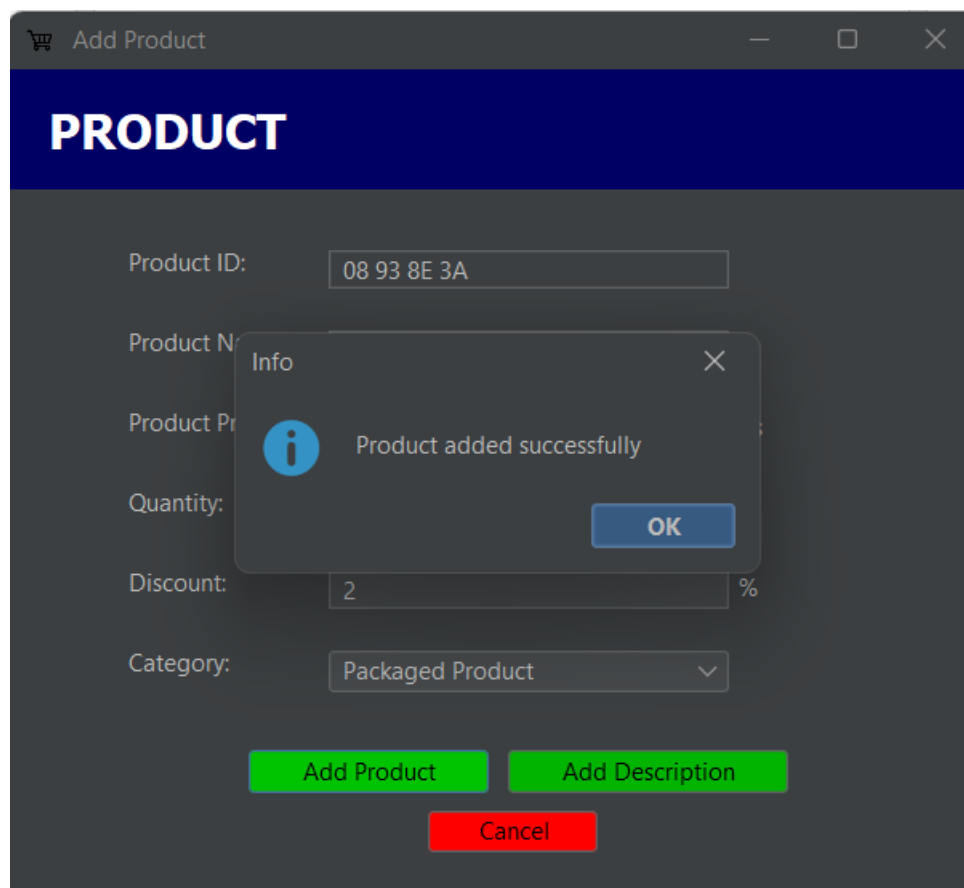
The screenshot shows a dropdown menu for product categories. The menu is open, displaying the following options:

- Grocery (highlighted with a blue background)
- Electronics
- Cosmetics
- Appliances
- Fresh Grocery
- Packaged Product
- Fresh Product
- Bakery Items

*Different categories of products*



*Adding new product/ error shown for invalid date format*



*Adding new product/ message shown when product is added*

28	B7 4F B3 81	Mobile Phone	60000.00	6	0.00	Electronics
29	97 ED 54 19	Hairbrush	550.00	8	5.00	Cosmetics
30	08 93 8E 3A	Juice	50.00	12	2.00	Packaged Product

*Database change after addition of new product*

Remove Product

PRODUCT

Product ID:

Product Name:

Product Price:

Rs

Quantity

Discount

%

Category:

Remove Product

Cancel

*Removing product*

Update Product

PRODUCT

Product ID:

A7 D4 B5 09

Product Name:

Chipser Masala

Product Price:

100

Rs

Quantity:

158

Discount:

5.0

%

Category:

Packaged Product

Update Details

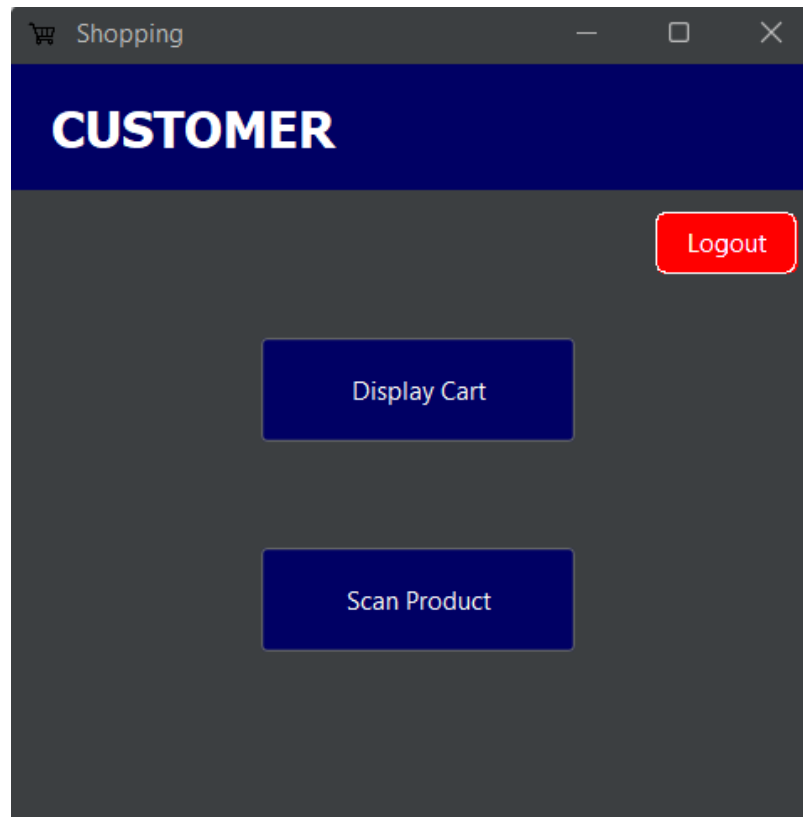
Cancel

Info

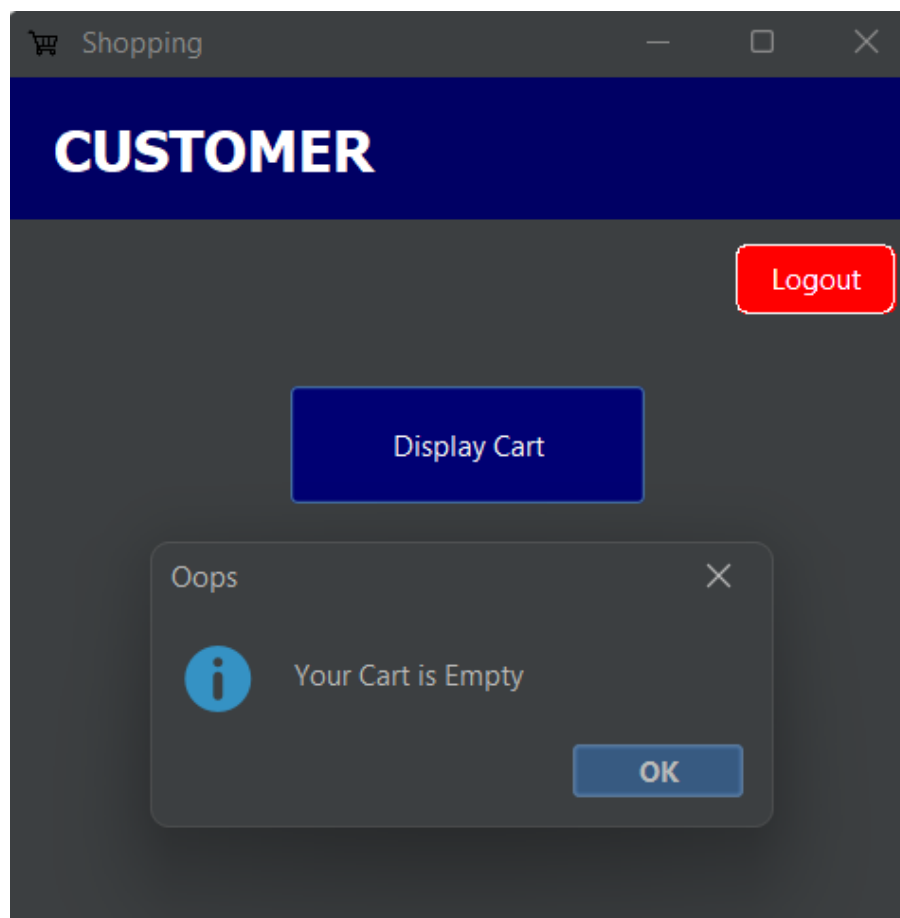
Product details updated successfully

OK

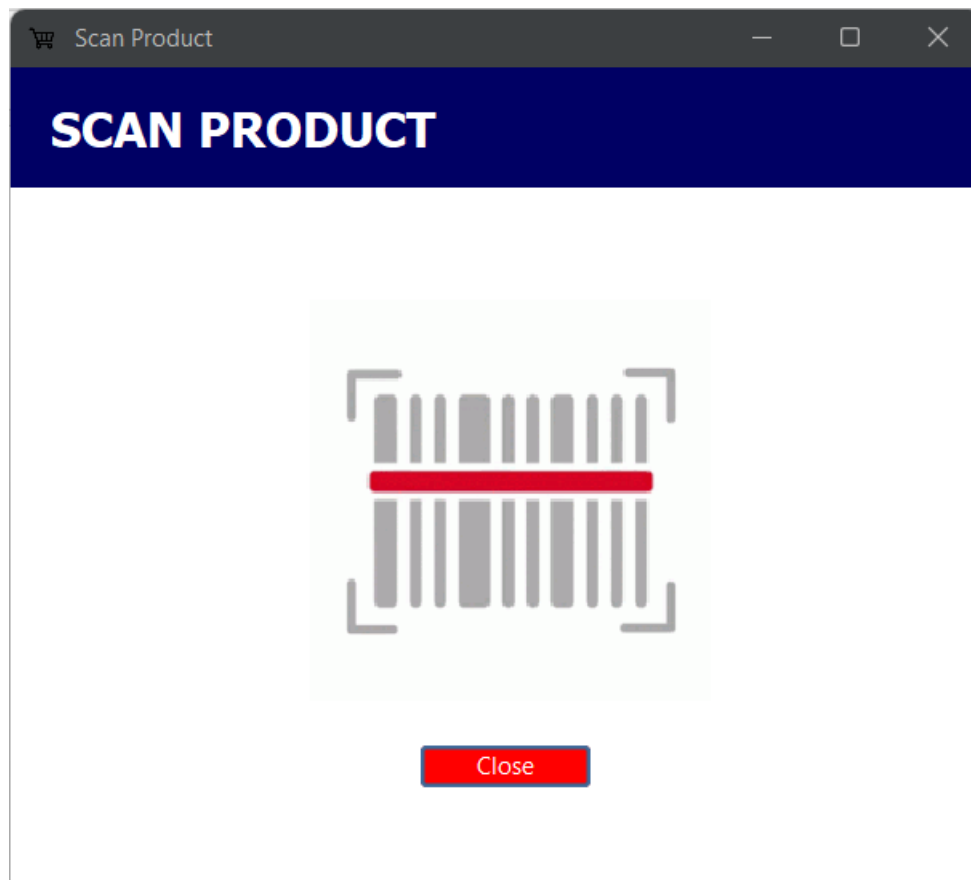
*Removing product/ message displayed upon success*



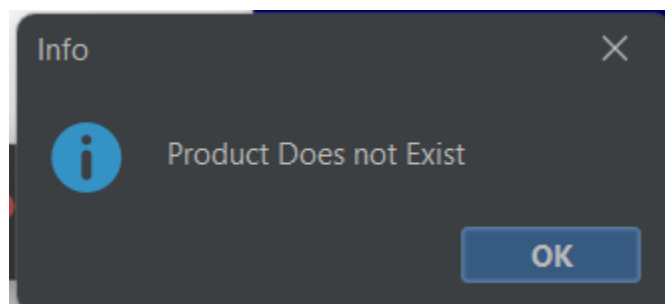
*Customer login window/ customer cart display*

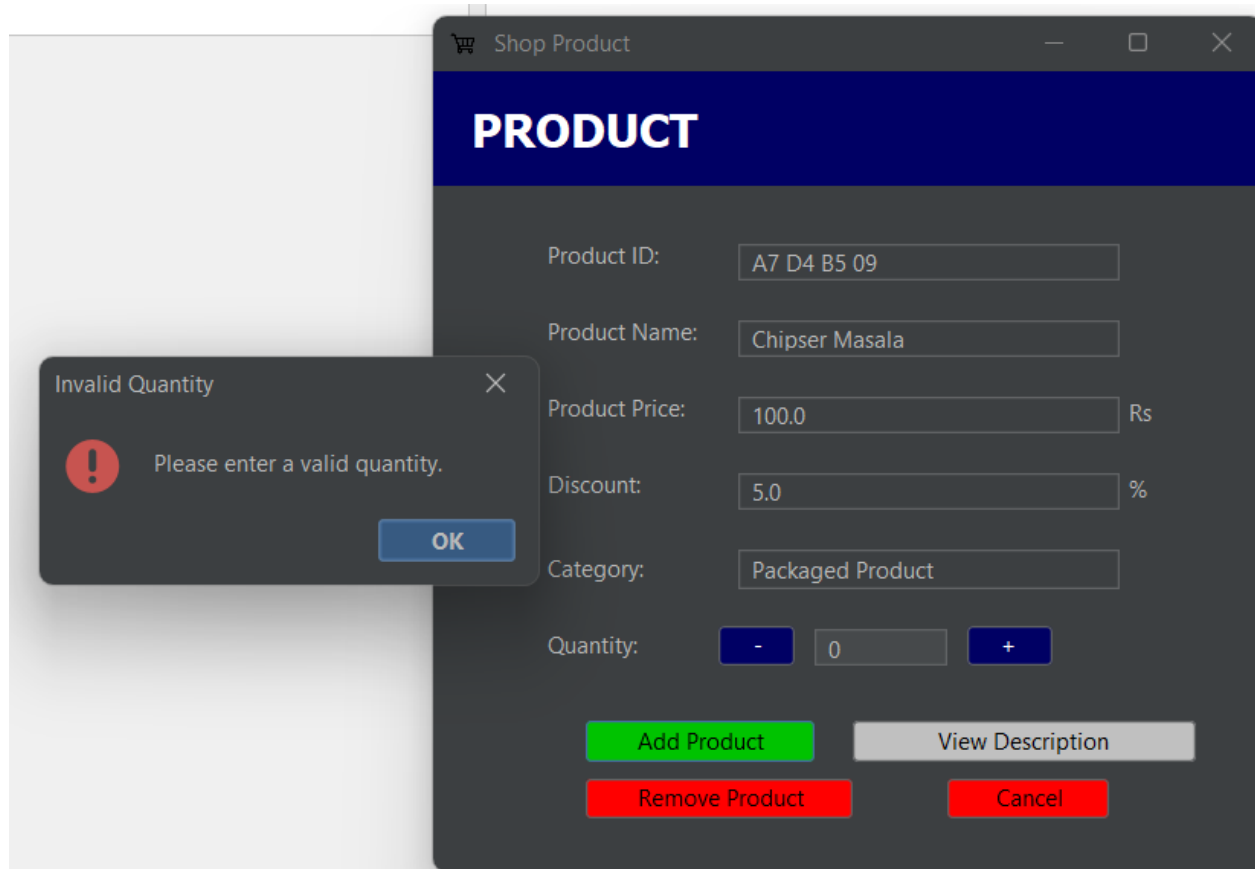




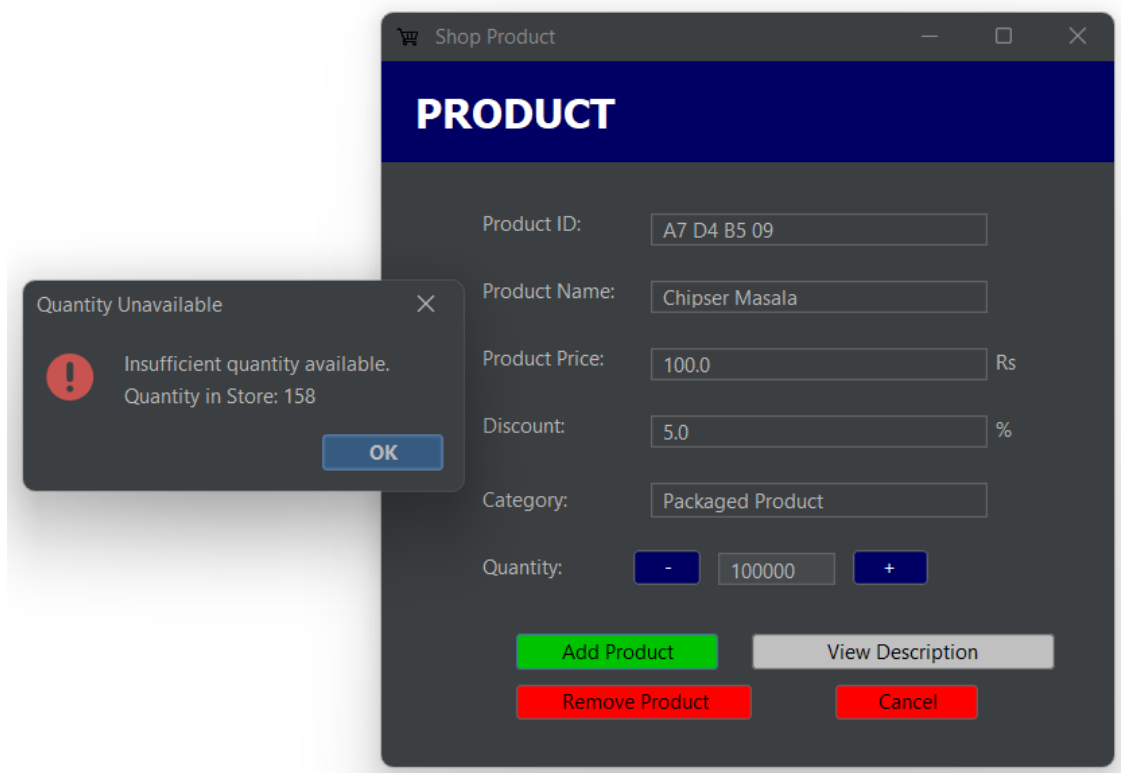


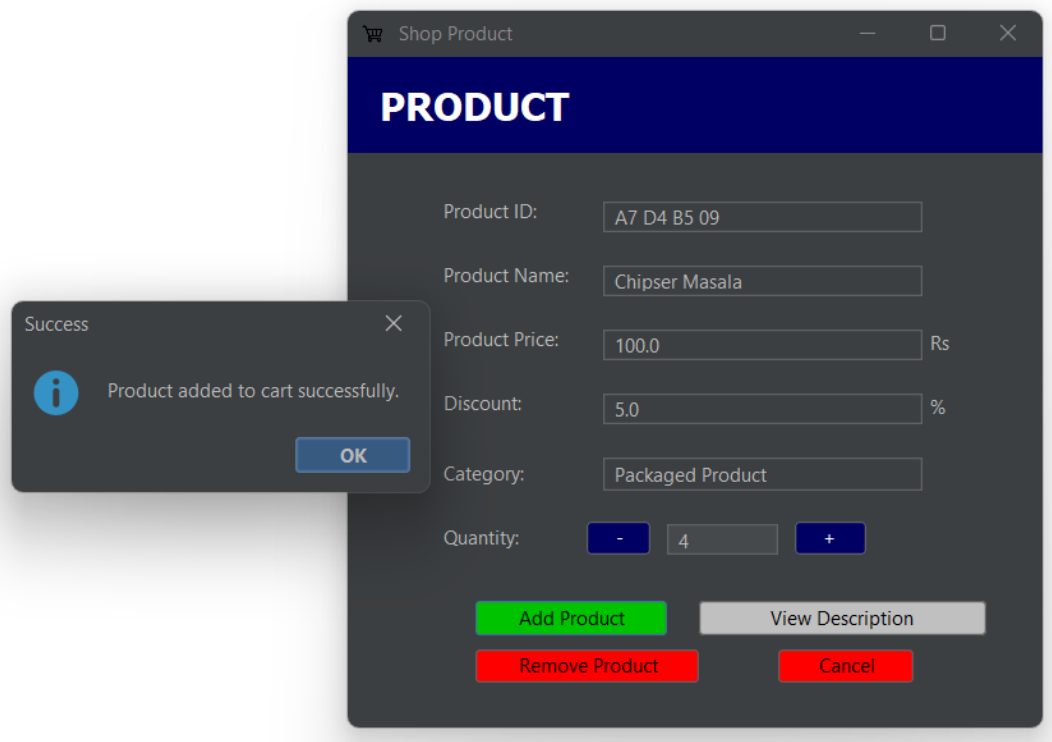
*Scan product window/ error displayed if product does not exist*



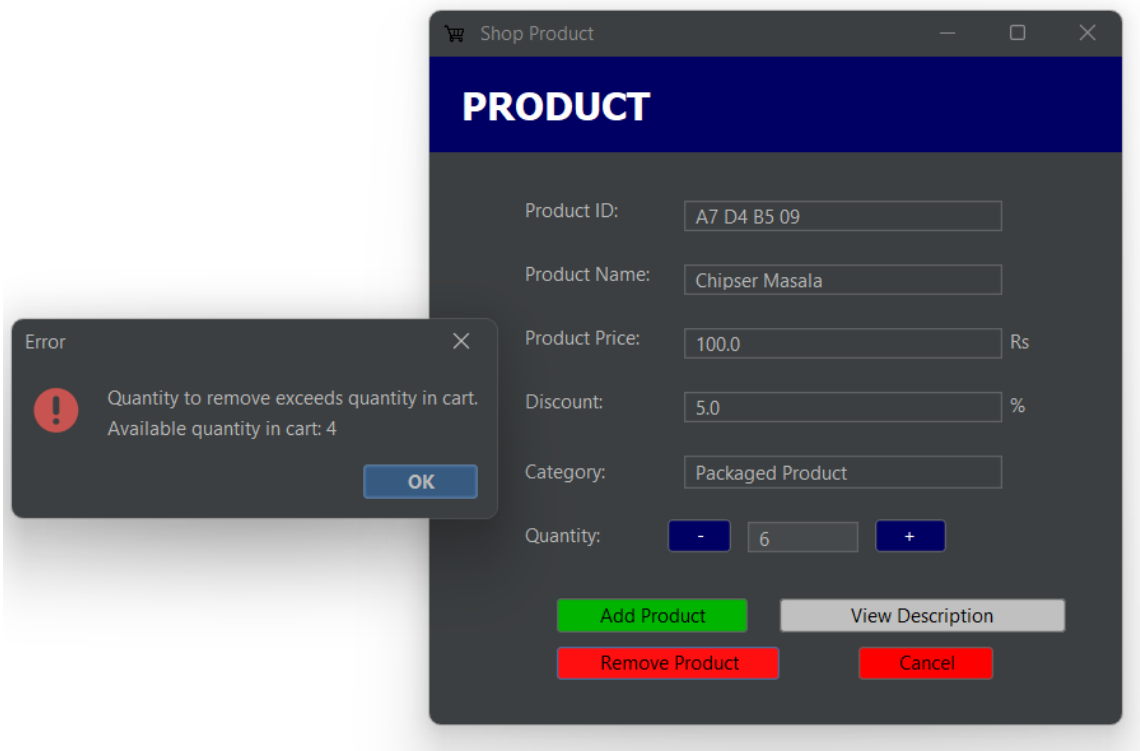


*Adding product to cart/ only quantities within the limit available in inventory allowed to be added/ not 0 or above available*






*Product successfully added to cart*



*Remove product form cart/ error shown if quantity of removal exceeds present quantity*

 View Packaged Grocery Product

## Packaged Groceries

Expiry Date:

3d

05

2024

Manufacture Date:

01

05

2024

Brand

Kashmir Foods


Close

Product details window

Success

Product removed from cart successfully.

OK

 Shop Product

## PRODUCT

Product ID:

A7 D4 B5 09

Product Name:

Chipsr Masala

Product Price:

100.0

Rs

Discount:

5.0

%

Category:

Packaged Product

Quantity:

-

1

+

Add Product

View Description

Remove Product

Cancel

Product successfully removed from cart

View Cosmetic Product

Cosmetics

Brand:

Ingredients:

Close

Shop Product

PRODUCT

Product ID:

Product Name:

Product Price:
Rs

Discount:
%

Category:

Quantity:

-

+

Add Product
View Description

Remove Product
Cancel

*Adding product/ product description along with category specific description*

View Cart

YOUR CART

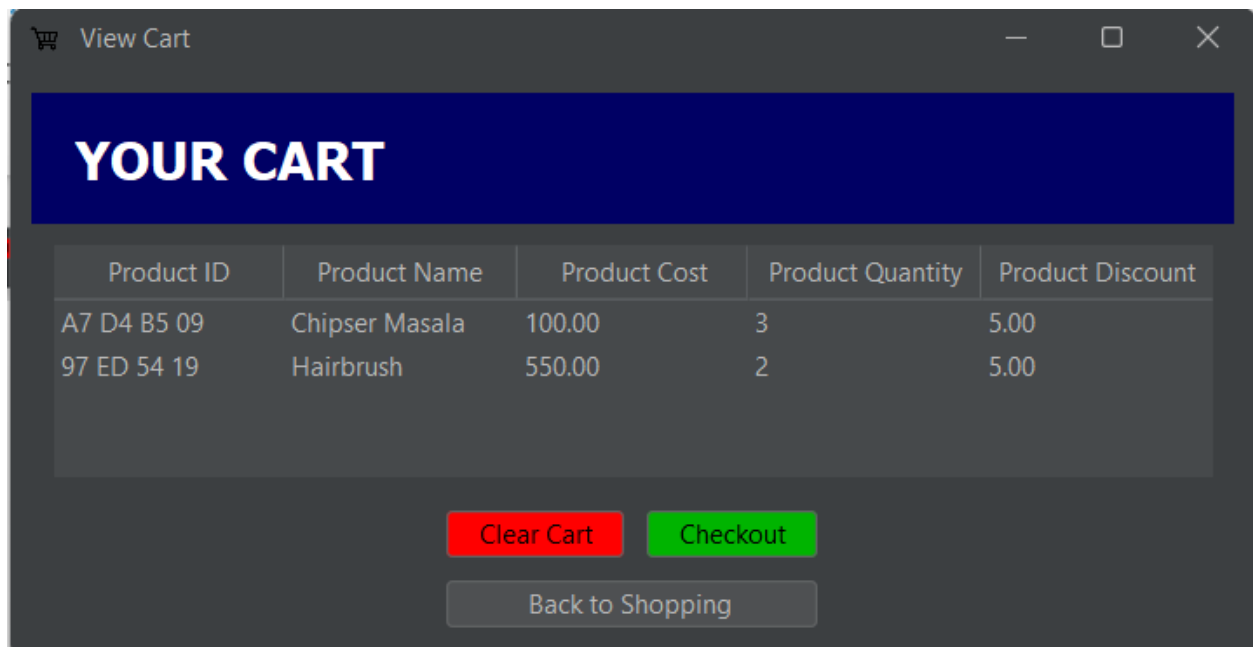
Product ID	Product Name	Product Cost	Product Quantity	Product Discount
A7 D4 B5 09	Chipser Masala	100.00	3	5.00
97 ED 54 19	Hairbrush	550.00	2	5.00

Clear Cart

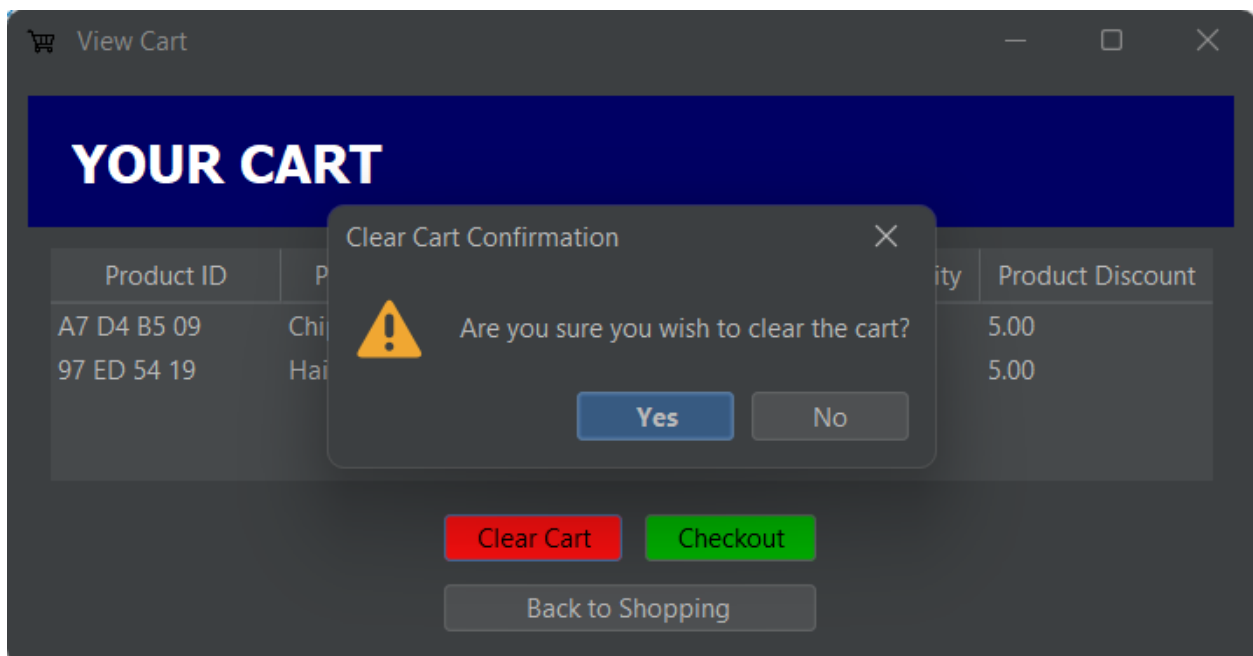
Checkout

Back to Shopping

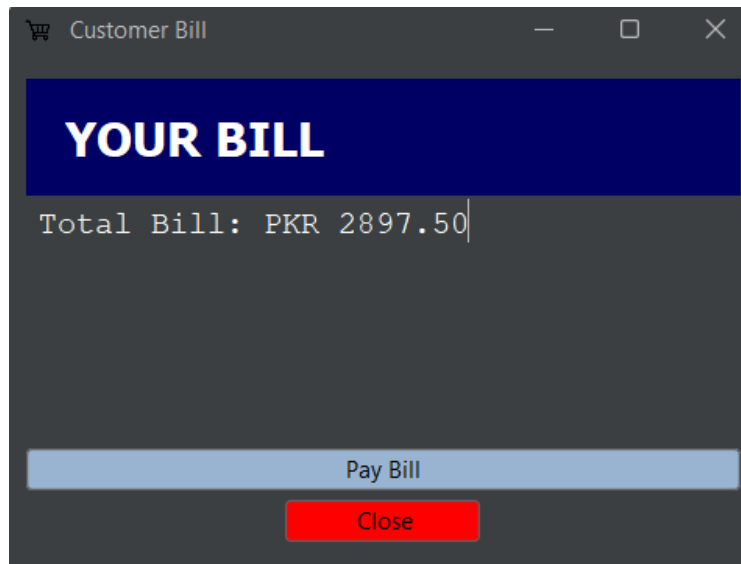
*View cart window*



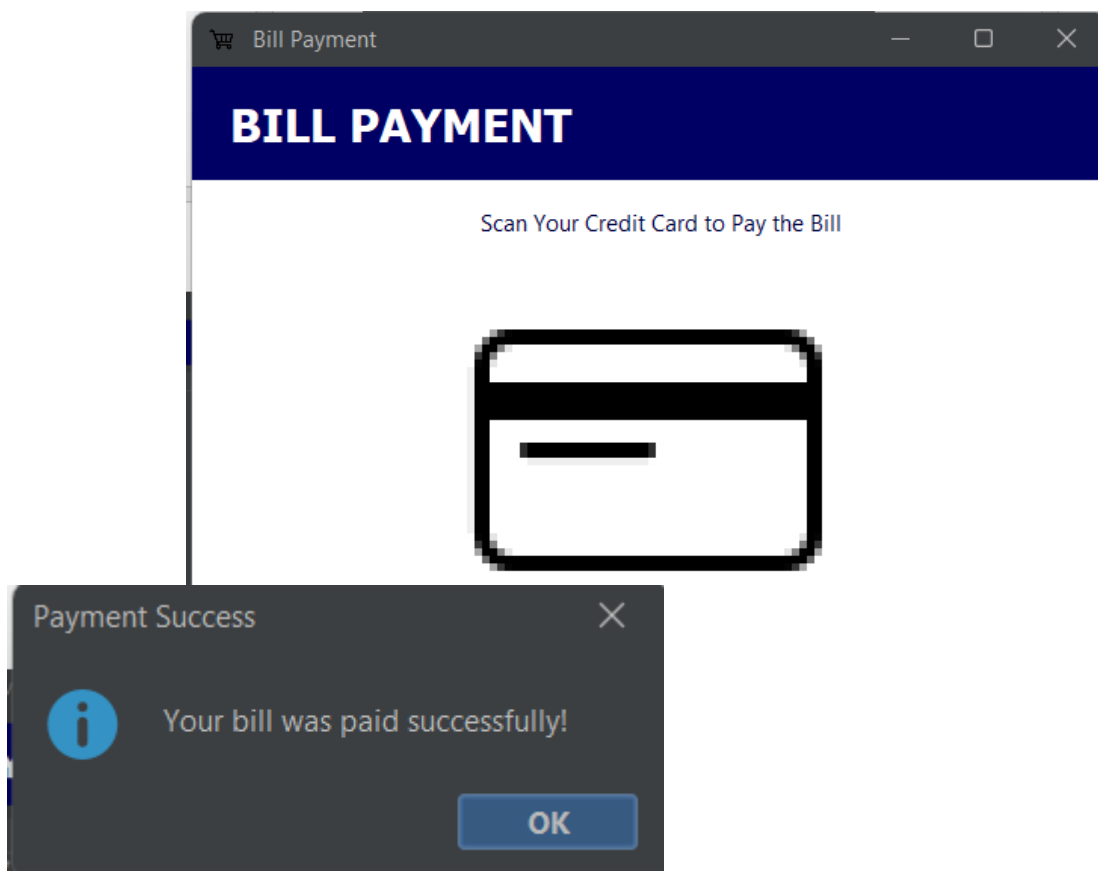
*Resize-able functionality of view cart window*



*Warning error displayed before clearing cart*



*Display Bill window*



*Simulation for bill payment using card read by the RFID reader/ message displayed*

## FURTHER IMPROVEMENTS

**Integration of Barcode Reader:** Incorporating a barcode reader alongside RFID scanning capabilities would provide users with multiple options for product identification, enhancing flexibility and convenience in the scanning process.

**Enhanced GUI Features:** Implementing a more user-friendly graphical user interface (GUI) with resizable windows would improve the overall usability of the application, catering to varying screen sizes and user preferences.

**Automated UID Capture:** Introducing a feature to automatically scan and capture UID data from customer cards issued to them, rather than relying on manual input, would streamline the checkout process and reduce the margin for errors.

**Optimization of File Management:** Implementing more efficient methods to manage file usage between CoolTerm and Java, such as automatic clearing of the file after execution or real-time data processing to minimize file storage requirements, would prevent issues arising from file overflow and manual intervention.

**Error Handling and Logging:** Enhancing error handling mechanisms and implementing comprehensive logging functionalities within the application would assist in identifying and troubleshooting issues more effectively, improving overall system reliability and maintenance.

## CONCLUSION AND REFERENCES

### 1. References:

- a. Utilize resources like Geek for Geeks for comprehensive study materials.
- b. Explore YouTube tutorials for step-by-step guidance on setting up libraries and learning SQL basics.
- c. Leverage GitHub for a wide range of ideas and solutions to enhance and expand the functionality of the system.

1. Arduino Hardware Integration: The Google Drive link provided demonstrates how Arduino hardware, including an Arduino Mega or Uno and an RFID-RC522 module, is utilized to scan unique identifiers (UIDs) from RFID cards. This integration showcases the hardware setup and illustrates the process of capturing UID data.

2. Simulation of Bill Payment: Within the project, a simulated bill payment process is implemented to provide users with a comprehensive experience. This simulation, triggered upon scanning a UID, presents users with bill-related information without deducting actual payments.



Instead, the bill payment process is simulated, ensuring a seamless user interaction while accurately reflecting the payment status.

[https://drive.google.com/drive/folders/1u6oUCvjd1M3sEUWJSYoSxC0gt1ia\\_8zL](https://drive.google.com/drive/folders/1u6oUCvjd1M3sEUWJSYoSxC0gt1ia_8zL)

**GIT HUB LINK**

<https://github.com/ushbafatima/ScanDash.git>