ALPHA STATIONARY SHOP INVENTORY MANAGEMNT SYSTEM

A MINI PROJECT

SUBMITTED BY

LOGAPRIYA S G - 230701164

MADHUMITHA B M - 230701168

In partial fulfilment for the award of the Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI- 602105

2024-25

BONAFIDE CERTIFIACTE

Certified that this project report "**Alpha Stationary Shop INVENTORY MANAGEMENT SYSTEM**" is a Bonafide work of "**B M MADHUMITHA (230701168) & S G LOGAPRIYA (230701164)**" who carried out the project work under the Supervision of **Mrs.K.Maheshmeena - Assistant Professor**

Submitted for the Practical Examination held on ___________

Signature of

Mrs.K.Maheshmeena

Assistant Professor

Computer Science and Engineering

Rajalakshmi Engineering College

Thandalam, Chennai-602 105

TABLE OF CONTENTS

ABSTRACT

The Alpha Stationary Shop Inventory Management System is a comprehensive solution designed to optimize and automate the shop's inventory management processes. By leveraging modern technologies, including JavaFX for an intuitive graphical user interface and PostgreSQL for a secure and efficient database backend, the system addresses the key challenges of traditional manual inventory handling. It empowers the shop to streamline operations across multiple domains, including inventory tracking, supplier relationship management, customer data handling, and invoice generation.

This system ensures accurate monitoring of stock levels, enabling timely replenishment and minimizing both overstocking and stockouts. Suppliers and customers are managed effectively through centralized records, facilitating seamless procurement and personalized customer interactions. Additionally, automated invoice management enhances operational efficiency by reducing manual effort and potential errors in billing and transactions.

Key features of the system include real-time inventory updates, integration of loyalty programs to boost customer retention, and detailed supply chain tracking. These functionalities not only save time but also provide valuable data insights to support strategic decision-making. The scalable and modular design of the system ensures that it can adapt to future business needs and technological advancements, making it a future-proof investment for the shop.

In summary, the Alpha Stationary Shop Inventory Management System represents a paradigm shift from traditional inventory practices to a technology-driven, automated, and efficient approach. It empowers the shop to improve customer satisfaction, enhance operational productivity, and drive overall business growth through the intelligent use of data and automation.

Objectives of the System

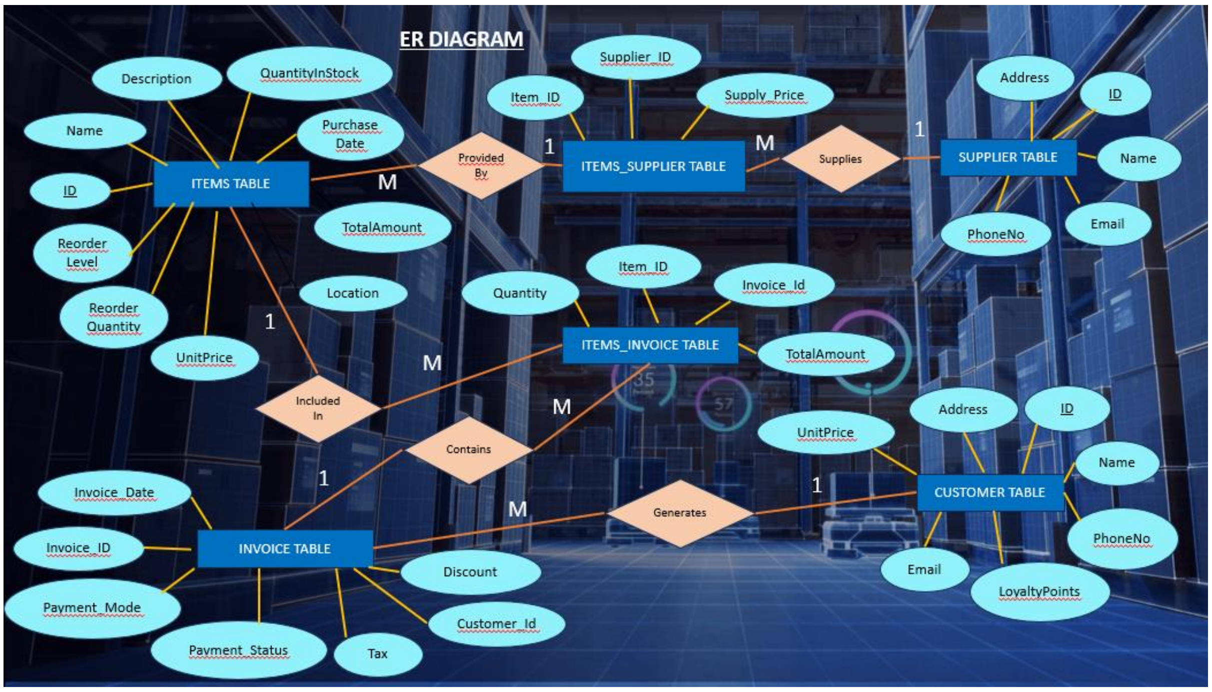
1. **Track Inventory Levels: Maintain real-time stock status to prevent overstocking or shortages.**
2. **Supplier Management: Ensure timely procurement by maintaining supplier records and linking them to inventory.**
3. **Customer Relationship Management: Track customer loyalty and enhance the purchasing experience.**
4. **Invoice Handling: Automate invoice creation for both suppliers and customers.**
5. **Data Security: Ensure secure access and accurate data storage through a structured database.**

System Architecture

The architecture consists of three primary layers:

1. Frontend Layer:
    - JavaFX-based GUI with forms for data entry and tables for data visualization.
    - Interactive buttons for navigation and feature access.
2. Backend Layer:
    - PostgreSQL database designed with normalized schema and foreign key constraints.
    - SQL queries and stored procedures for efficient operations.
3. Integration Layer:
    - dbconnect module acts as the bridge, ensuring secure database communication.

ER Diagram



Detailed ER Diagram

The ER diagram visually represents the database schema, showing relationships and cardinalities. It includes:

- Entities:
  - Items, Suppliers, Customers, Invoices, SupplyInvoices, and Items-Invoice.
- Attributes:
  - Each entity’s columns such as item_id, item_name, supplier_name, etc.
- Relationships:
  - One-to-many relationships between customers and invoices.
  - Many-to-many relationships between items and invoices using a junction table.

Database Design and Schema

Table Descriptions:

1. **Items Table:**
   - **Purpose: Stores details about inventory items.**
   - **Key Columns:**
     - **item_id: Primary key.**
     - **quantity: Tracks stock level.**
   - **Example Data:**

item_id	item_name	quantity	unit_price
101	Pen	200	10.5

2. **Suppliers Table:**
   - **Purpose: Tracks supplier information.**
   - **Key Columns:**
     - **supplier_id: Primary key.**
     - **email: Ensures uniqueness.**
3. **Customers Table:**
   - **Includes loyalty tracking (loyalty_points).**
4. **Invoices and Junction Tables:**
   - **Link items to invoices, managing quantities and pricing.**

Normalization:

- **Ensures no redundancy.**
- **Example: Splitting Items and Suppliers into separate tables to avoid duplication.**

Features and Functionalities

Inventory Management:

- **Add, update, and delete inventory records.**
- **Real-time stock level visualization.**

Supplier Management:

- **Add supplier details.**
- **Link suppliers to supply invoices for tracking.**

Customer Management:

- **Maintain loyalty points and purchase history.**

Invoice Management:

- **Generate customer and supply invoices.**
- **Automatically update item quantities based on sales.**

Frontend

JavaFX Interface

The frontend leverages JavaFX, a modern UI toolkit for building rich client applications, to provide an interactive and user-friendly interface. It includes:

1. **Input Forms:**
    - **AddItemPage: A dedicated form where users can add new inventory items by filling in details such as item name, description, quantity, reorder level, unit price, and purchase date.**
    - **AddSupplierPage: Enables the addition of supplier details, including name, contact information, and address.**
    - **AddCustomerPage: Captures customer information like name, contact details, and loyalty points, with options to track their last purchase date.**
    - **Other forms such as AddInvoicePage and AddSupplyInvoicePage streamline invoice creation and linking items to suppliers.**
2. **TableViews for Data Visualization:**
    - **Dynamic tables display real-time data fetched from the database, allowing users to visualize and manage records efficiently:**
        - **displayItems: Lists inventory items with columns for item ID, name, quantity, unit price, and reorder level.**
        - **displayCustomers: Presents customer data, including loyalty points and purchase history.**
        - **displaySuppliers: Shows supplier details such as contact number and address.**

- Each table is designed for easy navigation and is dynamically populated using JavaFX's ObservableList.

Navigation

The App.java file serves as the central hub for navigation, connecting all the system's functionalities into a unified application.

1. **Feature Access:**
    - Intuitive buttons allow users to quickly navigate to forms or data views.
    - Buttons are styled using CSS to ensure visual consistency and user appeal. For example:

css

Copy code

```
-fx-font-size: 14px;

-fx-padding: 10px 20px;

-fx-background-color: #555555;

-fx-text-fill: white;

-fx-border-radius: 5px;
```

    - The modular navigation system ensures that adding new features requires minimal changes.
2. **Responsive Design:**
    - Layouts such as VBox and HBox provide responsive alignment, ensuring compatibility across different screen sizes.
    - Buttons like "Back to Home Page" enable easy return to the main menu, enhancing usability.

Frontend Advantages:

- Provides a clean and visually appealing interface for users to interact with the system.
- Reduces the learning curve for users with its intuitive design and organized navigation structure.

Backend

PostgreSQL Database

The backend utilizes PostgreSQL, a powerful open-source relational database management system, to handle all data-related operations securely and efficiently.

1. Normalized Schema:
    - The database schema follows normalization principles to eliminate data redundancy and ensure consistency.
    - Relationships are established through foreign keys, creating a structured and maintainable schema:
        - Example: The items_invoice table links the items and invoices tables to maintain a many-to-many relationship.
2. Foreign Keys and Constraints:
    - Enforce referential integrity:
        - Example: Deleting a supplier automatically cascades to related supply invoices, preventing orphaned records.
    - Primary keys (item_id, supplier_id, etc.) uniquely identify rows, ensuring accuracy during data retrieval.

Integration

The dbconnect module serves as the middleware between the frontend and the database, enabling secure and efficient query execution.

1. Connection Management:
    - Establishes and maintains a connection pool to handle multiple user requests without performance bottlenecks.

    - **Utilizes secure authentication credentials to connect to the PostgreSQL database.**

2. **Query Execution:**
    - **Encapsulates SQL operations like INSERT, UPDATE, DELETE, and SELECT into reusable methods:**

Data Security:

- **Adopts best practices for data security, such as parameterized queries to prevent SQL injection.**
- **Regular backups ensure data durability in case of system failures.**

<u>Integration Between Frontend and Backend</u>

The interaction between the JavaFX-based frontend and the PostgreSQL backend is seamless, facilitated by the following:

1. **Dynamic Data Binding:**
    - **Updates in the database are immediately reflected in the JavaFX tables using ObservableList.**
    - **Example: Adding a new item dynamically updates the displayItems table.**
2. **Modularity:**
    - **Each JavaFX page corresponds to a specific backend operation, maintaining a clear separation of concerns.**
    - **Example: The AddItemPage exclusively interacts with the items table, while AddSupplierPage handles the supplier table.**

SQL Operations and Queries

Key Queries:

1. **Insert Data:**

sql

```
INSERT INTO customers (customer_name, contact_no, email, address)
VALUES (?, ?, ?, ?);
```

2. **Update Stock:**

sql

```
UPDATE items SET quantity = quantity - ? WHERE item_id = ?;
```

3. **Join Query for Analysis:**

sql

```
SELECT i.item_name, ii.quantity, ii.total_amount

FROM items i

JOIN items_invoice ii ON i.item_id = ii.item_id

WHERE ii.invoice_id = ?;
```

Query Performance Optimization:

- **Indexing on frequently queried columns like item_id and invoice_id.**

CODE-IMPLEMENTATION

ADDCUSTOMER-PAGE

```
package project;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.Date;
import java.time.LocalDate;

public class AddCustomerPage extends Application {

    @SuppressWarnings("unused")
    @Override
    public void start(Stage primaryStage) {
        // Labels and TextFields
        Label customerNameLabel = new Label("Customer Name:");
        TextField customerNameField = new TextField();

        Label contactNoLabel = new Label("Contact No:");
        TextField contactNoField = new TextField();

        Label emailLabel = new Label("Email:");
        TextField emailField = new TextField();

        Label addressLabel = new Label("Address:");
        TextField addressField = new TextField();

        Label loyaltyPointsLabel = new Label("Loyalty Points:");
        TextField loyaltyPointsField = new TextField();

        Label lastPurchaseDateLabel = new Label("Last Purchase Date:");
        DatePicker lastPurchaseDatePicker = new DatePicker(LocalDate.now());

        // Submit button
        Button submitButton = new Button("Add Customer");

        submitButton.setOnAction(e -> {
            String customerName = customerNameField.getText();
            String contactNo = contactNoField.getText();
            String email = emailField.getText();
            String address = addressField.getText();
            int loyaltyPoints = Integer.parseInt(loyaltyPointsField.getText());
            Date lastPurchaseDate =
Date.valueOf(lastPurchaseDatePicker.getValue());

            // Create dbconnect object and insert the customer
            dbconnect db = new dbconnect();
            Connection conn = db.connect();
            db.addCustomer(conn, customerName, contactNo, email, address,
loyaltyPoints, lastPurchaseDate);
            primaryStage.close();
        });

        // Layout setup
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(8);
        grid.setHgap(10);

        // Add components to grid
        grid.add(customerNameLabel, 0, 0);
        grid.add(customerNameField, 1, 0);
        grid.add(contactNoLabel, 0, 1);
        grid.add(contactNoField, 1, 1);
        grid.add(emailLabel, 0, 2);
        grid.add(emailField, 1, 2);
        grid.add(addressLabel, 0, 3);
        grid.add(addressField, 1, 3);
        grid.add(loyaltyPointsLabel, 0, 4);
        grid.add(loyaltyPointsField, 1, 4);
        grid.add(lastPurchaseDateLabel, 0, 5);
        grid.add(lastPurchaseDatePicker, 1, 5);
        grid.add(submitButton, 1, 6);
        // Scene setup
        Scene scene = new Scene(grid, 400, 350);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add Customer");
        primaryStage.show();
    }
}
```

AddInvoice-Page

```
package project;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.time.LocalDate;

public class AddInvoicePage extends Application {
```

```
@Override
public void start(Stage primaryStage) {
  // Labels and TextFields
  Label invoiceDateLabel = new Label("Invoice Date:");
  DatePicker invoiceDatePicker = new DatePicker(LocalDate.now());

  Label customerIdLabel = new Label("Customer ID:");
  TextField customerIdField = new TextField();

  Label totalAmountLabel = new Label("Total Amount:");
  TextField totalAmountField = new TextField();

  // Submit button
  Button submitButton = new Button("Create Invoice");

  submitButton.setOnAction(e -> {
    // Get form input values
    try {
      int customerId = Integer.parseInt(customerIdField.getText());
      float totalAmount = Float.parseFloat(totalAmountField.getText());
      Date invoiceDate = Date.valueOf(invoiceDatePicker.getValue());

      // Create dbconnect object and insert the invoice
      dbconnect db = new dbconnect();
      Connection conn = db.connect();
```

```
            // SQL query to insert the new invoice
            String sqlInsertInvoice = "INSERT INTO invoice (invoice_date,
customer_id, bill_amount) VALUES (?, ?, ?)";
            PreparedStatement stmtInsertInvoice =
conn.prepareStatement(sqlInsertInvoice);

            // Set parameters for the prepared statement
            stmtInsertInvoice.setDate(1, invoiceDate);  // Set invoice_date
            stmtInsertInvoice.setInt(2, customerId);    // Set customer_id
            stmtInsertInvoice.setFloat(3, totalAmount); // Set bill_amount

            // Execute the query to insert the invoice
            int rowsAffected = stmtInsertInvoice.executeUpdate();
            if (rowsAffected > 0) {
               System.out.println("Invoice created successfully.");

               // Update the last_purchase_date in the customer table
               String sqlUpdateCustomer = "UPDATE customer SET
last_purchase_date = ? WHERE customer_id = ?";
               PreparedStatement stmtUpdateCustomer =
conn.prepareStatement(sqlUpdateCustomer);

               // Set parameters for the prepared statement
               stmtUpdateCustomer.setDate(1, invoiceDate);  // Set the invoice
date as the last purchase date
               stmtUpdateCustomer.setInt(2, customerId);    // Set customer_id

               // Execute the update query
```

```
                int updateRowsAffected = stmtUpdateCustomer.executeUpdate();
                if (updateRowsAffected > 0) {
                    System.out.println("Customer's last purchase date updated
successfully.");
                    primaryStage.close();
                } else {
                    System.out.println("Failed to update the customer's last
purchase date.");
                }

                // Close the window after success
                primaryStage.close();
            } else {
                System.out.println("Failed to create invoice.");
            }

            // Close the connection
            conn.close();
        } catch (SQLException sqlEx) {
            System.out.println("Database error: " + sqlEx.getMessage());
            sqlEx.printStackTrace();
        } catch (Exception ex) {
            System.out.println("Error: " + ex.getMessage());
            ex.printStackTrace();
        }
    });
```

```
        // Layout setup
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(8);
        grid.setHgap(10);

        // Add components to grid
        grid.add(invoiceDateLabel, 0, 0);
        grid.add(invoiceDatePicker, 1, 0);

        grid.add(customerIdLabel, 0, 2);
        grid.add(customerIdField, 1, 2);

        grid.add(totalAmountLabel, 0, 5);
        grid.add(totalAmountField, 1, 5);
        grid.add(submitButton, 1, 6);

        // Scene setup
        Scene scene = new Scene(grid, 400, 350);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Create Invoice");
        primaryStage.show();
    }
}
```

<u>AddItem-Page</u>

```
package project;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.*;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;


import java.sql.Connection;

import java.sql.Date;

import java.time.LocalDate;


public class AddItemPage extends Application {


    @SuppressWarnings("unused")

    @Override

    public void start(Stage primaryStage) {

        // Labels and TextFields

        Label itemNameLabel = new Label("Item Name:");

        TextField itemNameField = new TextField();


        Label descriptionLabel = new Label("Description:");

        TextField descriptionField = new TextField();


        Label quantityLabel = new Label("Quantity:");
        TextField quantityField = new TextField();

        Label reorderLevelLabel = new Label("Reorder Level:");
        TextField reorderLevelField = new TextField();

        Label unitPriceLabel = new Label("Unit Price:");
        TextField unitPriceField = new TextField();

        Label purchaseDateLabel = new Label("Purchase Date:");
        DatePicker purchaseDatePicker = new DatePicker(LocalDate.now());

        // Submit button
        Button submitButton = new Button("Add Item");

        submitButton.setOnAction(e -> {
            String itemName = itemNameField.getText();
            String description = descriptionField.getText();
            int quantity = Integer.parseInt(quantityField.getText());
            int reorderLevel = Integer.parseInt(reorderLevelField.getText());
            float unitPrice = Float.parseFloat(unitPriceField.getText());
            Date purchaseDate = Date.valueOf(purchaseDatePicker.getValue());

            // Create dbconnect object and insert the item
            dbconnect db = new dbconnect();
            Connection conn = db.connect();
            db.addNewItem(conn, itemName, description, quantity, reorderLevel,
    unitPrice, purchaseDate);
            primaryStage.close();
        });

        // Layout setup
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(8);
        grid.setHgap(10);

        // Add components to grid
        grid.add(itemNameLabel, 0, 0);
        grid.add(itemNameField, 1, 0);
        grid.add(descriptionLabel, 0, 1);
        grid.add(descriptionField, 1, 1);
        grid.add(quantityLabel, 0, 2);
        grid.add(quantityField, 1, 2);
        grid.add(reorderLevelLabel, 0, 3);
        grid.add(reorderLevelField, 1, 3);
        grid.add(unitPriceLabel, 0, 4);
        grid.add(unitPriceField, 1, 4);
        grid.add(purchaseDateLabel, 0, 5);
        grid.add(purchaseDatePicker, 1, 5);
        grid.add(submitButton, 1, 6);

        // Scene setup
        Scene scene = new Scene(grid, 400, 350);
```

```
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add Item");
        primaryStage.show();
    }
}
```

Add Items To Invoice-Page

```
package project;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class AddItemsToInvoicePage extends Application {

    @Override
    public void start(Stage primaryStage) {
        // UI Components
```

```
Label itemIdLabel = new Label("Item ID:");
TextField itemIdField = new TextField();

Label invoiceIdLabel = new Label("Invoice ID:");
TextField invoiceIdField = new TextField();

Label quantityLabel = new Label("Quantity:");
TextField quantityField = new TextField();

Label unitPriceLabel = new Label("Unit Price:");
TextField unitPriceField = new TextField();

Button submitButton = new Button("Submit");
submitButton.setOnAction(e -> {
   // Retrieve input values
   int itemId = Integer.parseInt(itemIdField.getText());
   int invoiceId = Integer.parseInt(invoiceIdField.getText());
   int quantity = Integer.parseInt(quantityField.getText());
   double unitPrice = Double.parseDouble(unitPriceField.getText());
   double totalAmount = unitPrice * quantity;

   // Create DB connection and execute the necessary queries
   dbconnect db = new dbconnect();
   try (Connection con = db.connect()) {
      // 1. Check if enough quantity is available in the items table
      if (checkItemAvailability(con, itemId, quantity)) {
```

```
                // 2. Add item to the invoice table
                db.addItemsToInvoice(con, itemId, invoiceId, quantity, unitPrice,
totalAmount);

                // 3. Reduce the item quantity in the items table
                reduceItemQuantity(con, itemId, quantity);

                // Close the stage after successful operation
                primaryStage.close();
            } else {
                // Show an alert or message if not enough items are available
                System.out.println("Not enough items available.");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    });

    // Layout
    GridPane grid = new GridPane();
    grid.setPadding(new Insets(10));
    grid.setVgap(10);
    grid.setHgap(10);

    grid.add(itemIdLabel, 0, 0);
    grid.add(itemIdField, 1, 0);
    grid.add(invoiceIdLabel, 0, 1);
```

```
        grid.add(invoiceIdField, 1, 1);
        grid.add(quantityLabel, 0, 2);
        grid.add(quantityField, 1, 2);
        grid.add(unitPriceLabel, 0, 3);
        grid.add(unitPriceField, 1, 3);
        grid.add(submitButton, 1, 4);

        Scene scene = new Scene(grid, 400, 250);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add Items to Invoice");
        primaryStage.show();
    }

    /**
     * Checks if the requested quantity is available in the items table.
     * @param con The database connection.
     * @param itemId The ID of the item.
     * @param quantity The quantity to check.
     * @return true if the quantity is available, false otherwise.
     */
    private boolean checkItemAvailability(Connection con, int itemId, int
quantity) throws SQLException {
        String query = "SELECT quantity FROM items WHERE item_id = ?";
        try (PreparedStatement stmt = con.prepareStatement(query)) {
            stmt.setInt(1, itemId);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
```

```
                int availableQuantity = rs.getInt("quantity");
                return availableQuantity >= quantity;
            }
        }
    }
    return false;
}

/**
 * Reduces the quantity of the item in the items table after it has been
added to the invoice.
 * @param con The database connection.
 * @param itemId The ID of the item.
 * @param quantity The quantity to reduce.
 */
private void reduceItemQuantity(Connection con, int itemId, int quantity)
throws SQLException {
    String query = "UPDATE items SET quantity = quantity - ? WHERE item_id
= ?";
    try (PreparedStatement stmt = con.prepareStatement(query)) {
        stmt.setInt(1, quantity);
        stmt.setInt(2, itemId);
        stmt.executeUpdate();
    }
}
}
```

Add Supplier-Page

```
package project;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.*;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;


import java.sql.Connection;


public class AddSupplierPage extends Application {


   @SuppressWarnings("unused")

   @Override

   public void start(Stage primaryStage) {

      // Labels and TextFields

      Label supplierNameLabel = new Label("Supplier Name:");

      TextField supplierNameField = new TextField();


      Label contactNoLabel = new Label("Contact No:");

      TextField contactNoField = new TextField();


      Label emailLabel = new Label("Email:");

      TextField emailField = new TextField();
```

```
Label addressLabel = new Label("Address:");
TextField addressField = new TextField();

// Submit button
Button submitButton = new Button("Add Supplier");

submitButton.setOnAction(e -> {
   String supplierName = supplierNameField.getText();
   String contactNo = contactNoField.getText();
   String email = emailField.getText();
   String address = addressField.getText();

   // Create dbconnect object and insert the supplier
   dbconnect db = new dbconnect();
   Connection conn = db.connect();
   db.addSupplier(conn, supplierName, contactNo, email, address);
   primaryStage.close();
});

// Layout setup
GridPane grid = new GridPane();
grid.setPadding(new Insets(10, 10, 10, 10));
grid.setVgap(8);
grid.setHgap(10);
```

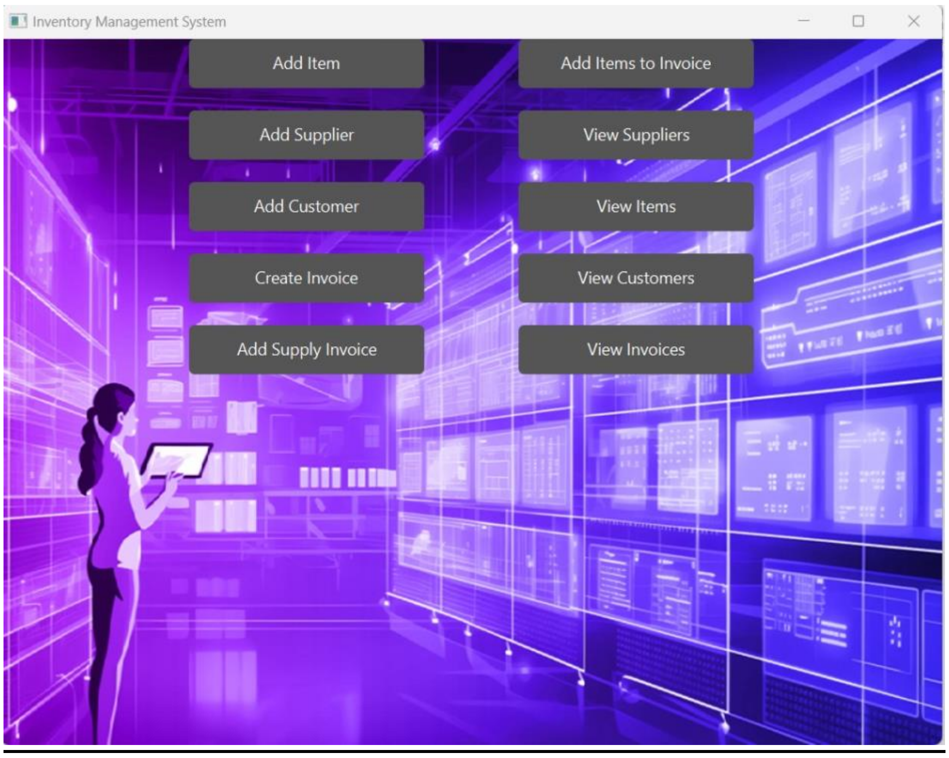
```
        // Add components to grid
        grid.add(supplierNameLabel, 0, 0);
        grid.add(supplierNameField, 1, 0);
        grid.add(contactNoLabel, 0, 1);
        grid.add(contactNoField, 1, 1);
        grid.add(emailLabel, 0, 2);
        grid.add(emailField, 1, 2);
        grid.add(addressLabel, 0, 3);
        grid.add(addressField, 1, 3);
        grid.add(submitButton, 1, 4);

        // Scene setup
        Scene scene = new Scene(grid, 400, 350);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add Supplier");
        primaryStage.show();
    }
}
```

<u>Add Supply Invoice-Page</u>

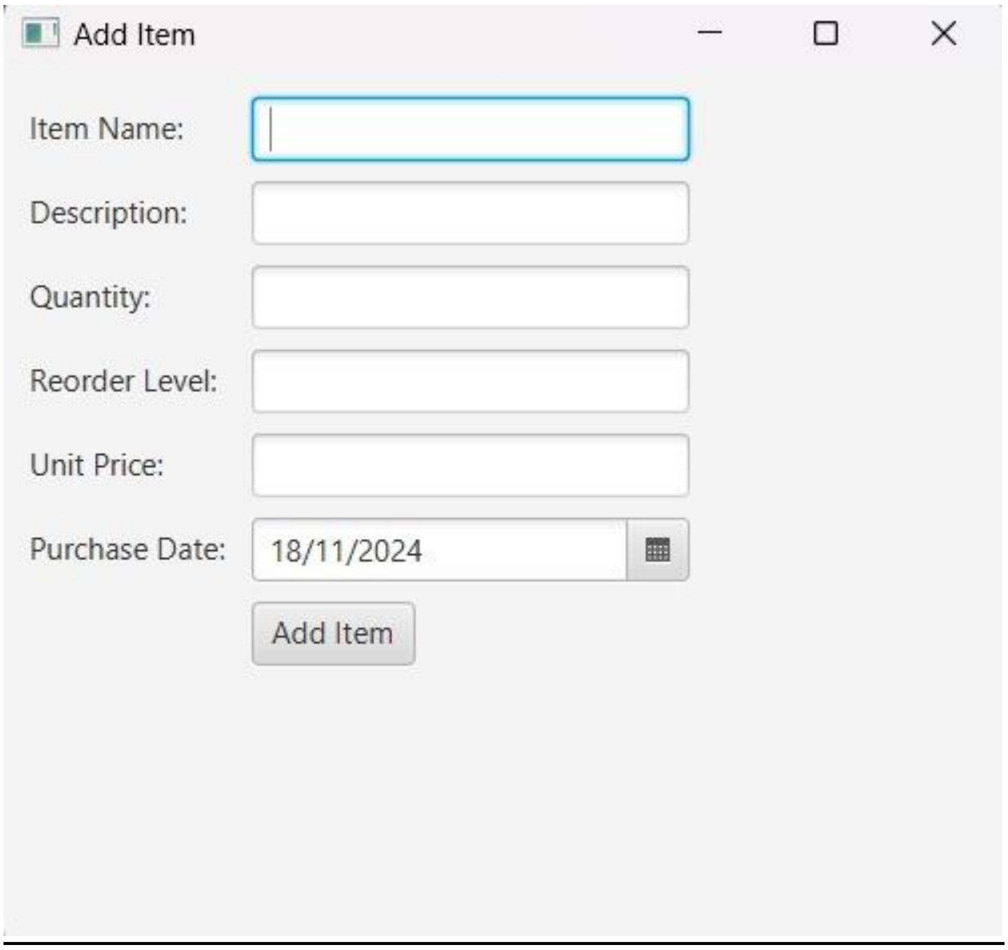
```
package project;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.Date;

public class AddSupplyInvoicePage extends Application {

    @SuppressWarnings("unused")
    @Override
    public void start(Stage primaryStage) {
        // UI Components
        Label itemIdLabel = new Label("Item ID:");
        TextField itemIdField = new TextField();

        Label supplierIdLabel = new Label("Supplier ID:");
        TextField supplierIdField = new TextField();

        Label supplyPriceLabel = new Label("Supply Price:");
        TextField supplyPriceField = new TextField();

        Label quantityLabel = new Label("Quantity:");
        TextField quantityField = new TextField();
        Label billDateLabel = new Label("Bill Date:");
        DatePicker billDatePicker = new DatePicker();

        Button submitButton = new Button("Submit");
        submitButton.setOnAction(e -> {
            int itemId = Integer.parseInt(itemIdField.getText());
            int supplierId = Integer.parseInt(supplierIdField.getText());
            double supplyPrice = Double.parseDouble(supplyPriceField.getText());
            int quantity = Integer.parseInt(quantityField.getText());
            Date billDate = Date.valueOf(billDatePicker.getValue());

            dbconnect db = new dbconnect();
            Connection con = db.connect();
            db.addSupplyInvoice(con, itemId, supplierId, supplyPrice, quantity,
billDate);
            primaryStage.close();
        });

        // Layout
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10));
        grid.setVgap(10);
        grid.setHgap(10);

        grid.add(itemIdLabel, 0, 0);
        grid.add(itemIdField, 1, 0);
        grid.add(supplierIdLabel, 0, 1);
        grid.add(supplierIdField, 1, 1);
        grid.add(supplyPriceLabel, 0, 2);
        grid.add(supplyPriceField, 1, 2);
        grid.add(quantityLabel, 0, 3);
        grid.add(quantityField, 1, 3);
        grid.add(billDateLabel, 0, 4);
        grid.add(billDatePicker, 1, 4);
        grid.add(submitButton, 1, 5);

        Scene scene = new Scene(grid, 400, 300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add Supply Invoice");
        primaryStage.show();
    }
}
```

SNAPSHOTS

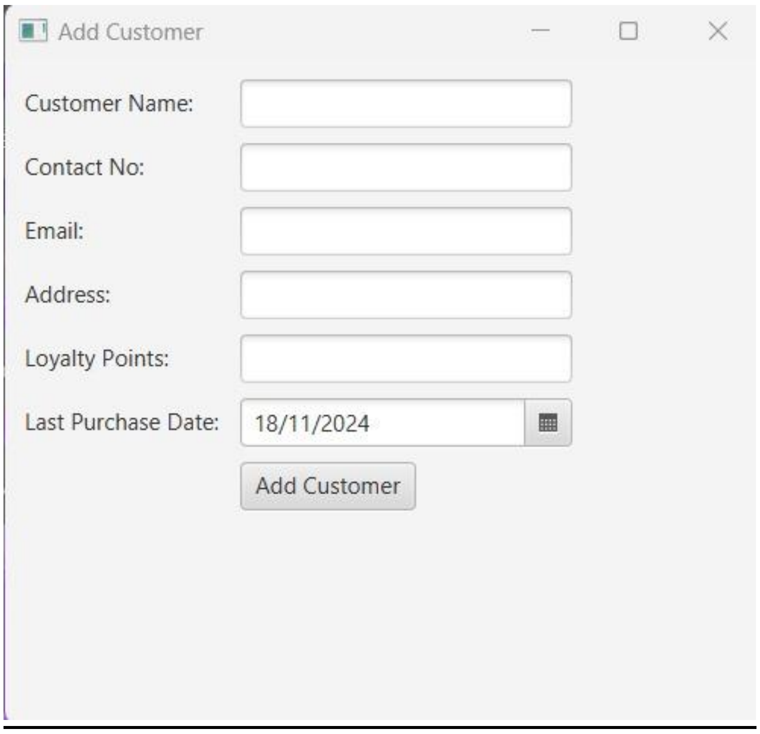
Home Page:



Add Items Page:



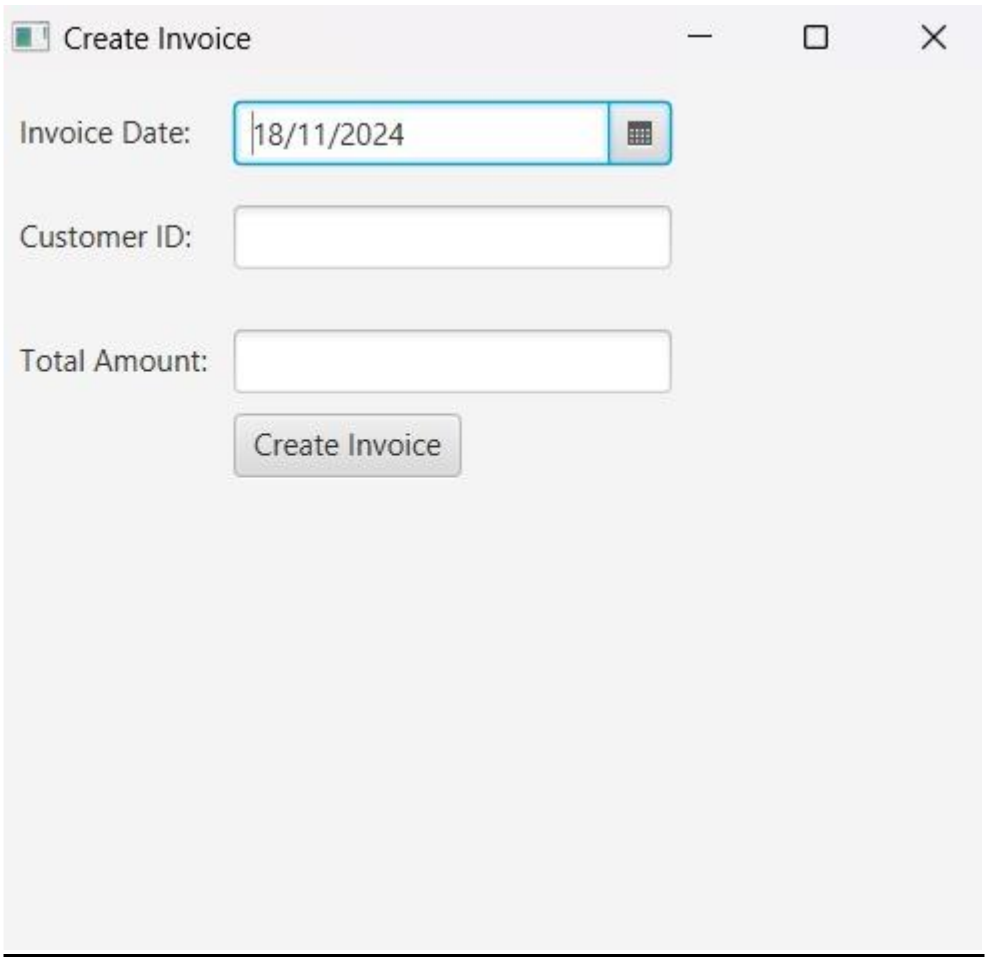
Add Customer Page:



Add Customer

Customer Name:	
Contact No:	
Email:	
Address:	
Loyalty Points:	
Last Purchase Date:	18/11/2024

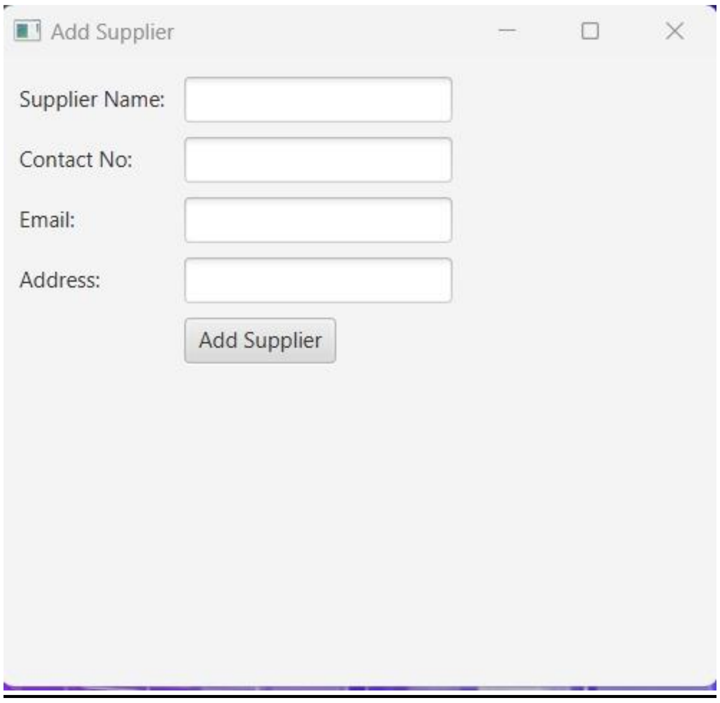
Add Invoice Page:



Create Invoice

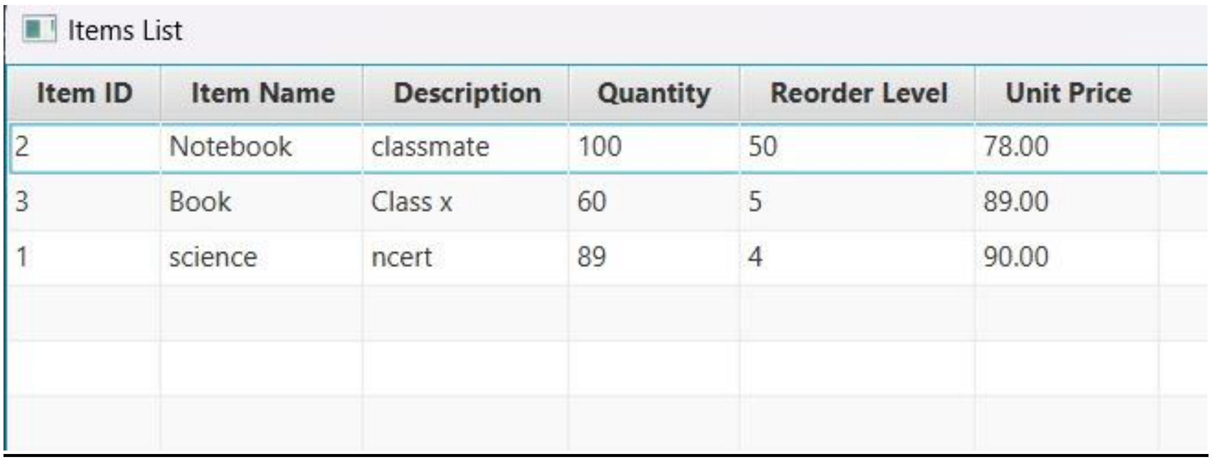
Invoice Date:	18/11/2024
Customer ID:	
Total Amount:	

Add Supplier Page:


Add Supplier

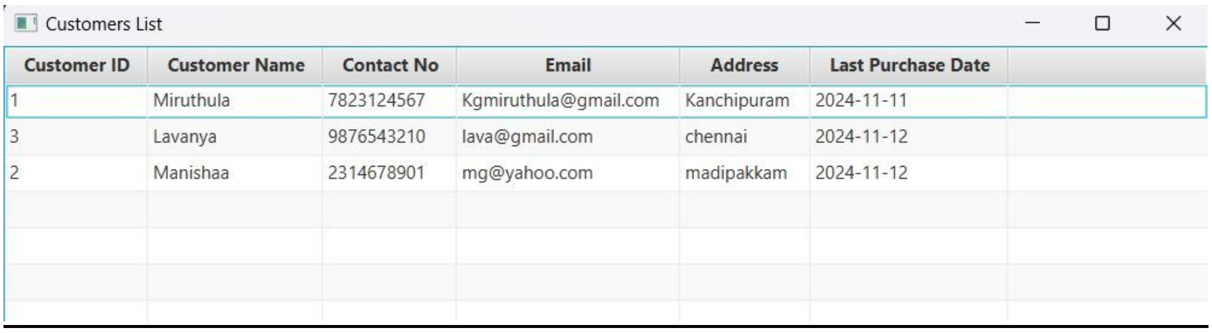
Supplier Name:	
Contact No:	
Email:	
Address:	

Display Items Page:


Items List

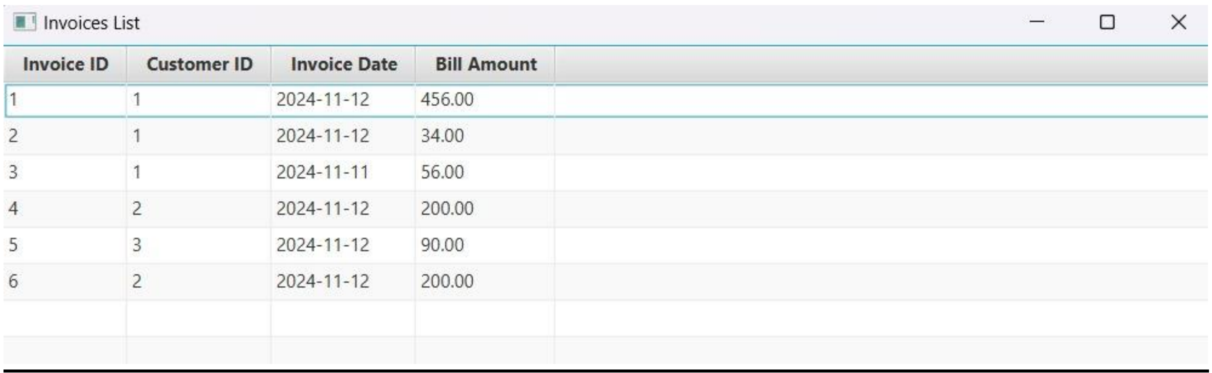
Item ID	Item Name	Description	Quantity	Reorder Level	Unit Price
2	Notebook	classmate	100	50	78.00
3	Book	Class x	60	5	89.00
1	science	ncert	89	4	90.00

Display Customer Page:


Customers List

Customer ID	Customer Name	Contact No	Email	Address	Last Purchase Date
1	Miruthula	7823124567	Kgmiruthula@gmail.com	Kanchipuram	2024-11-11
3	Lavanya	9876543210	lava@gmail.com	chennai	2024-11-12
2	Manishaa	2314678901	mg@yahoo.com	madipakkam	2024-11-12

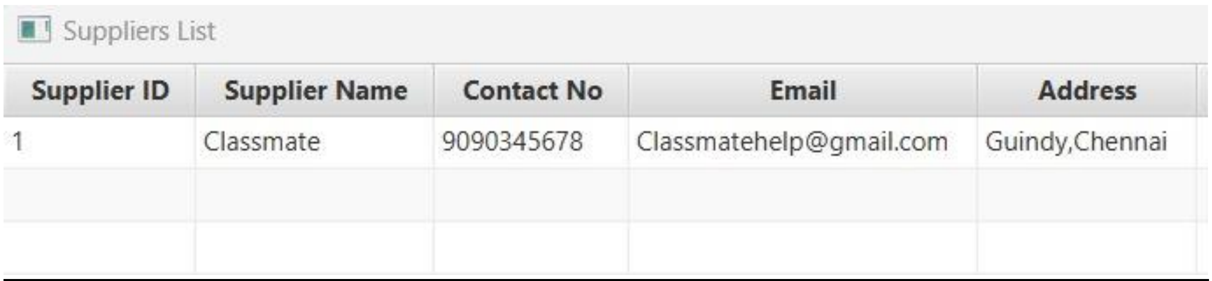
Display Invoice Page:



Invoices List

Invoice ID	Customer ID	Invoice Date	Bill Amount
1	1	2024-11-12	456.00
2	1	2024-11-12	34.00
3	1	2024-11-11	56.00
4	2	2024-11-12	200.00
5	3	2024-11-12	90.00
6	2	2024-11-12	200.00

Display Supplier Page:



Suppliers List

Supplier ID	Supplier Name	Contact No	Email	Address
1	Classmate	9090345678	Classmatehelp@gmail.com	Guindy,Chennai

Benefits of the System

1.Operational Efficiency: Automates routine tasks, reducing manual errors.

2.Cost-Effective: Optimizes inventory to prevent losses from overstocking or stockouts.

3.Improved Customer Experience: Loyalty tracking encourages repeat purchases.

4.Scalability: Modular design supports future enhancements.

CONCLUSION

The Alpha Stationary Shop Inventory Management System successfully addresses the challenges of manual inventory handling and retail operations by providing an efficient, automated solution tailored to the specific needs of a stationery shop. Through the use of JavaFX for an interactive, user-friendly graphical interface and PL/SQL for secure and robust database management, the project demonstrates the practical integration of frontend and backend technologies.