# E-Commerce System

**A MINI-PROJECT BY:**

**Aswinkumar J 230701044**

**Bharath B 230701052**

***in partial fulfillment of the award of the degree***

***OF***

## *BACHELOR OF ENGINEERING*

**IN**

## COMPUTER SCIENCE AND ENGINEERING



**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

**CHENNAI**

**NOVEMBER 2024**

# BONAFIDE CERTIFICATE

Certified that this project **“E-Commerce System”** is the bonafide work of **“Aswinkumar J, Bharath B”** who carried out the project work under my supervision.

Submitted for the practical examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SIGNATURE SIGNATURE

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

This project focuses on the development of an e-commerce application using **Java Swing** for the graphical user interface and **MySQL** as the database management system. The application aims to provide a functional platform for users to browse products, add them to a shopping cart, and place orders seamlessly.

The project demonstrates a robust integration between the front-end and back-end, ensuring efficient user authentication, product management, and transaction handling. The modular design of the application allows for easy navigation and scalability. Key features include user registration and login, product catalogue display, cart management, and order tracking.

The choice of Java Swing ensures a flexible and customizable user interface, while MySQL facilitates reliable data storage and retrieval. This report outlines the architecture, implementation details, testing results, and challenges faced during the development process. The project concludes with potential future enhancements, such as incorporating payment gateways and refining the user interface to improve usability and functionality.

***TABLE OF CONTENTS***

1. **INTRODUCTION**

**2**. **Requirements**

2.1 Software Requirement

2.2 Hardware Requirement Specifications

|  |
| --- |

**3.Entity Relationship Diagram**

**4. Schema Diagram**

**5.Implementation**

**6.Snapshot**

**7.Conclusion**

**1.INTRODUCTION**

The rapid growth of e-commerce has transformed the way businesses operate and customers shop. From buying daily essentials to luxury products, online shopping has become an integral part of modern life. The ability to browse products, compare prices, and make purchases conveniently from anywhere has made e-commerce platforms a necessity in today’s fast-paced world.

This project focuses on developing an e-commerce application using **Java Swing** for the front-end and **MySQL** for the back-end. The application is designed to provide users with an intuitive platform to explore products, add them to a shopping cart, and place orders effortlessly. By leveraging the versatility of Java Swing for graphical user interfaces and the robust data management capabilities of MySQL, this project ensures a seamless and reliable shopping experience.

The primary objective of this project is to design and implement a functional e-commerce application that emphasizes simplicity, usability, and scalability. The app features core modules such as user registration and login, product catalog display, cart management, and order processing. These modules collectively simulate the operations of a real-world e-commerce system, offering practical insights into software design and development.

Java Swing was chosen for its ability to create lightweight, customizable, and platform-independent graphical interfaces, making it suitable for desktop applications. On the other hand, MySQL provides a powerful and efficient relational database system to handle large volumes of data while ensuring consistency and security.

This report provides an in-depth overview of the project, detailing its system architecture, implementation process, and testing outcomes. Additionally, it highlights the challenges encountered during development and proposes future enhancements to improve the functionality and user experience of the application.

**REQUIREMENTS**

**2.1 Software Requirement Specifications**

Operating System Front End Back End Server Documentation : Windows 11

Frontend Software: Java

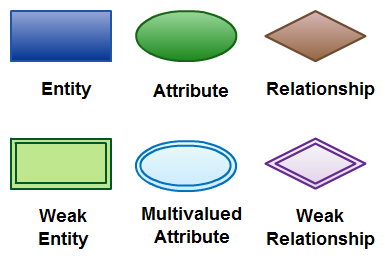
Backend Software: MySQL

**2.2 Hardware Requirement Specifications**

Computer Processor Core i3 Processor Speed 2.3 GHz Processor Hard Disk 400 GB or more RAM Min 2GB

**ENTITY RELATIONSHIP DIAGRAM**

An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes. If the application is primarily a database application, the entity-relationship approach can be used effectively for modeling some parts of the problem. The main focus in ER modeling is the Data Items in the system and the relationship between them. It aims to create conceptual scheme for the Data from the user’s perspective. The model thus created is independent of any database model. The ER models are frequently represented as ER diagram. Here we present the ER diagram of the above mentioned project.



**SCHEMA DIAGRAM**

A database schema is the skeleton structure that represents the logical view of the entire database. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and make it useful.

A database schema can be divided broadly into two categories −

* Physical Database Schema − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* Logical Database Schema − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**IMPLEMENTATION**

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.sql.\*;

public class SmallBasketApp extends JFrame {

    private static final String URL = "jdbc:mysql://localhost:3306/orders";

    private static final String USER = "aswin";

    private static final String PASSWORD = "aswin";

    // Updated prices for specified items

    double[] MRPS = {99999.00, 799.99, 749.00, 2999.00, 19.99, 699.00, 600.00, 9.99, 199.99, 5.99};

    public SmallBasketApp() {

        setTitle("SmallBasket"); // Updated title

        setSize(400, 300);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        // Create buttons

        JButton viewButton = new JButton("View Orders");

        JButton deleteButton = new JButton("Delete Order");

        JButton buyItemButton = new JButton("Buy Item");

        // Add action listeners

        viewButton.addActionListener(e -> openViewOrdersFrame());

        deleteButton.addActionListener(e -> openDeleteOrderFrame());

        buyItemButton.addActionListener(e -> openBuyItemFrame());

        // Set up GridLayout panel and add buttons

        JPanel panel = new JPanel(new GridLayout(3, 1, 10, 10));

        panel.add(viewButton);

        panel.add(deleteButton);

        panel.add(buyItemButton);

        // Add panel to frame

        add(panel);

    }

    private void openBuyItemFrame() {

        JFrame buyFrame = new JFrame("Buy Item - SmallBasket");

        buyFrame.setSize(400, 400);

        buyFrame.setLocationRelativeTo(null);

        buyFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        JPanel formPanel = new JPanel(new GridLayout(8, 2, 5, 5));

        String[] items = {"iPhone", "Asus Laptop", "Java Programming Book", "Logitech Mouse", "REC Cashew Nuts",

                          "Scientific Calculator", "Steel Waterbottle", "Teddy Bear", "Titan Watch", "Sunrise Coffee Powder"};

        String[] sellers = {"Apple Pvt.", "Asus India Pvt.", "Fatim Stores", "Logitech Technologies Pvt.",

                            "REC Botanical Garden", "Casio Pvt.", "Sitha Pvt.", "ABC Handcrafts Pvt.",

                            "Titan India Pvt.", "NESCAFE Pvt."};

        JTextField nameField = new JTextField();

        JTextField addressField = new JTextField();

        JComboBox<String> itemComboBox = new JComboBox<>(items);

        JComboBox<Integer> quantityComboBox = new JComboBox<>(new Integer[]{1, 2, 3, 4, 5, 10});

        JLabel sellerNameLabel = new JLabel(sellers[0]);

        JComboBox<String> paymentMethodComboBox = new JComboBox<>(new String[]{"GPay", "Net Banking", "Credit Card", "Debit Card", "Cash"});

        JLabel mrpLabel = new JLabel("MRP: ₹" + MRPS[0]);

        JLabel priceLabel = new JLabel("Total Price: ₹" + MRPS[0]);

        JButton buyButton = new JButton("Buy");

        itemComboBox.addActionListener(e -> {

            int selectedIndex = itemComboBox.getSelectedIndex();

            sellerNameLabel.setText(sellers[selectedIndex]);

            mrpLabel.setText("MRP: ₹" + MRPS[selectedIndex]);

            updatePrice(itemComboBox, quantityComboBox, priceLabel);

        });

        quantityComboBox.addActionListener(e -> updatePrice(itemComboBox, quantityComboBox, priceLabel));

        formPanel.add(new JLabel("Name:"));

        formPanel.add(nameField);

        formPanel.add(new JLabel("Address:"));

        formPanel.add(addressField);

        formPanel.add(new JLabel("Select Item:"));

        formPanel.add(itemComboBox);

        formPanel.add(new JLabel("MRP for One Item:"));

        formPanel.add(mrpLabel);

        formPanel.add(new JLabel("Quantity:"));

        formPanel.add(quantityComboBox);

        formPanel.add(new JLabel("Seller Name:"));

        formPanel.add(sellerNameLabel);

        formPanel.add(new JLabel("Payment Method:"));

        formPanel.add(paymentMethodComboBox);

        formPanel.add(priceLabel);

        formPanel.add(buyButton);

        buyFrame.add(formPanel);

        buyFrame.setVisible(true);

        buyButton.addActionListener(e -> {

            String name = nameField.getText().trim();

            String address = addressField.getText().trim();

            String selectedItem = (String) itemComboBox.getSelectedItem();

            int quantity = (Integer) quantityComboBox.getSelectedItem();

            String sellerName = sellerNameLabel.getText();

            String paymentMethod = (String) paymentMethodComboBox.getSelectedItem();

            double price = MRPS[itemComboBox.getSelectedIndex()] \* quantity;

            if (name.isEmpty() || address.isEmpty()) {

                JOptionPane.showMessageDialog(buyFrame, "Please enter your Name and Address.", "Input Error", JOptionPane.ERROR\_MESSAGE);

                return;

            }

            int rowsInserted = insertOrder(name, selectedItem, price, sellerName, address, paymentMethod);

            if (rowsInserted > 0) {

                JOptionPane.showMessageDialog(buyFrame, "Purchase was successful!");

                buyFrame.dispose();

            } else {

                JOptionPane.showMessageDialog(buyFrame, "Error processing purchase.");

            }

        });

    }

    private void updatePrice(JComboBox<String> itemComboBox, JComboBox<Integer> quantityComboBox, JLabel priceLabel) {

        // Get the selected item's index

        int selectedIndex = itemComboBox.getSelectedIndex();

        // Get the quantity selected

        int quantity = (Integer) quantityComboBox.getSelectedItem();

        // Calculate the total price based on the MRP and quantity

        double total = MRPS[selectedIndex] \* quantity;

        // Update the price label to show the calculated total price

        priceLabel.setText("Total Price: ₹" + String.format("%.2f", total));

    }

    private void openViewOrdersFrame() {

        JFrame viewFrame = new JFrame("View Orders - SmallBasket");

        viewFrame.setSize(600, 400);

        viewFrame.setLocationRelativeTo(null);

        viewFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        // Create a table model and set up the JTable

        DefaultTableModel tableModel = new DefaultTableModel(

            new String[]{"Order ID", "Name", "Product", "Price", "Seller", "Address", "Payment Method"}, 0);

        JTable table = new JTable(tableModel);

        // Populate the table with data from the database

        String sql = "SELECT id, name, product\_name, product\_price, seller\_name, address, method\_of\_payment FROM orders";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);

             Statement statement = connection.createStatement();

             ResultSet resultSet = statement.executeQuery(sql)) {

            while (resultSet.next()) {

                int id = resultSet.getInt("id");

                String name = resultSet.getString("name");

                String product = resultSet.getString("product\_name");

                double price = resultSet.getDouble("product\_price");

                String seller = resultSet.getString("seller\_name");

                String address = resultSet.getString("address");

                String paymentMethod = resultSet.getString("method\_of\_payment");

                tableModel.addRow(new Object[]{id, name, product, price, seller, address, paymentMethod});

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(viewFrame, "Error retrieving orders from the database.", "Error", JOptionPane.ERROR\_MESSAGE);

            e.printStackTrace();

        }

        // Add the table to a JScrollPane and add it to the frame

        JScrollPane scrollPane = new JScrollPane(table);

        viewFrame.add(scrollPane);

        viewFrame.setVisible(true);

    }

    private int deleteOrder(int orderId) {

        String sql = "DELETE FROM orders WHERE id = ?";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);

             PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setInt(1, orderId);

            return preparedStatement.executeUpdate();

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return 0;

    }

    private void openDeleteOrderFrame() {

        JFrame deleteFrame = new JFrame("Delete Order - SmallBasket");

        deleteFrame.setSize(600, 400);

        deleteFrame.setLocationRelativeTo(null);

        deleteFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        // Create a table model and set up the JTable

        DefaultTableModel tableModel = new DefaultTableModel(

            new String[]{"Order ID", "Name", "Product", "Price", "Seller", "Address", "Payment Method"}, 0);

        JTable table = new JTable(tableModel);

        // Populate the table with data from the database

        String sql = "SELECT id, name, product\_name, product\_price, seller\_name, address, method\_of\_payment FROM orders";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);

             Statement statement = connection.createStatement();

             ResultSet resultSet = statement.executeQuery(sql)) {

            while (resultSet.next()) {

                int id = resultSet.getInt("id");

                String name = resultSet.getString("name");

                String product = resultSet.getString("product\_name");

                double price = resultSet.getDouble("product\_price");

                String seller = resultSet.getString("seller\_name");

                String address = resultSet.getString("address");

                String paymentMethod = resultSet.getString("method\_of\_payment");

                tableModel.addRow(new Object[]{id, name, product, price, seller, address, paymentMethod});

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(deleteFrame, "Error retrieving orders from the database.", "Error", JOptionPane.ERROR\_MESSAGE);

            e.printStackTrace();

        }

        // Add the table to a JScrollPane

        JScrollPane scrollPane = new JScrollPane(table);

        // Create a delete button

        JButton deleteButton = new JButton("Delete Selected Order");

        deleteButton.addActionListener(e -> {

            int selectedRow = table.getSelectedRow();

            if (selectedRow == -1) {

                JOptionPane.showMessageDialog(deleteFrame, "Please select an order to delete.", "Error", JOptionPane.WARNING\_MESSAGE);

                return;

            }

            int orderId = (int) tableModel.getValueAt(selectedRow, 0); // Get the Order ID

            int confirm = JOptionPane.showConfirmDialog(deleteFrame,

                "Are you sure you want to delete this order?", "Confirm Delete", JOptionPane.YES\_NO\_OPTION);

            if (confirm == JOptionPane.YES\_OPTION) {

                int rowsDeleted = deleteOrder(orderId);

                if (rowsDeleted > 0) {

                    tableModel.removeRow(selectedRow);

                    JOptionPane.showMessageDialog(deleteFrame, "Order deleted successfully.");

                } else {

                    JOptionPane.showMessageDialog(deleteFrame, "Error deleting order.", "Error", JOptionPane.ERROR\_MESSAGE);

                }

            }

        });

        // Set up the frame layout

        JPanel panel = new JPanel(new BorderLayout());

        panel.add(scrollPane, BorderLayout.CENTER);

        panel.add(deleteButton, BorderLayout.SOUTH);

        deleteFrame.add(panel);

        deleteFrame.setVisible(true);

    }

    private int insertOrder(String name, String productName, double productPrice, String sellerName, String address, String paymentMethod) {

        String sql = "INSERT INTO orders (name, product\_name, product\_price, seller\_name, address, method\_of\_payment) VALUES (?, ?, ?, ?, ?, ?)";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);

             PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setString(1, name);

            preparedStatement.setString(2, productName);

            preparedStatement.setDouble(3, productPrice);

            preparedStatement.setString(4, sellerName);

            preparedStatement.setString(5, address);

            preparedStatement.setString(6, paymentMethod);

            return preparedStatement.executeUpdate();

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return 0;

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            SmallBasketApp app = new SmallBasketApp();

            app.setVisible(true);

        });

    }

}

**SNAPSHOT**

Figure 1:Home Page

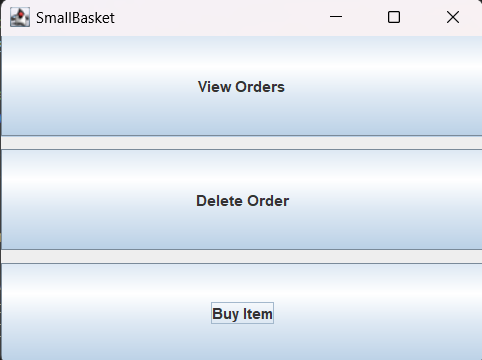


Figure 2: Buy Option

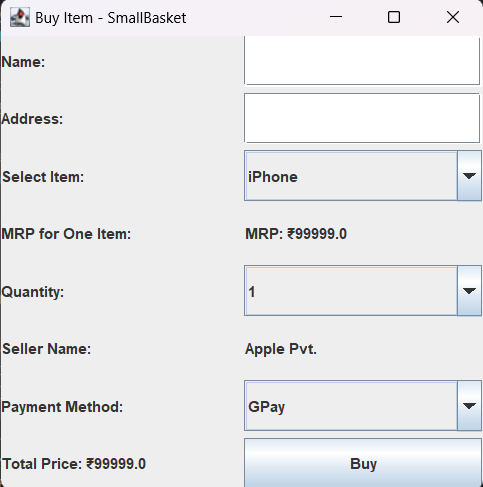


Figure 3: Deletion Option

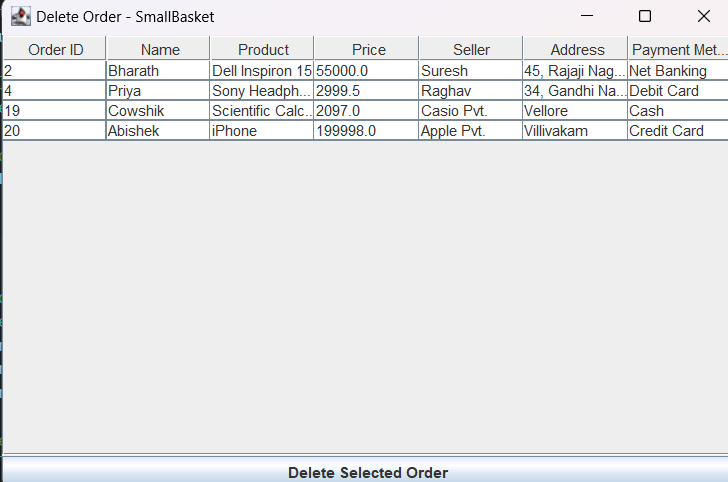


Figure 4: View Orders Option

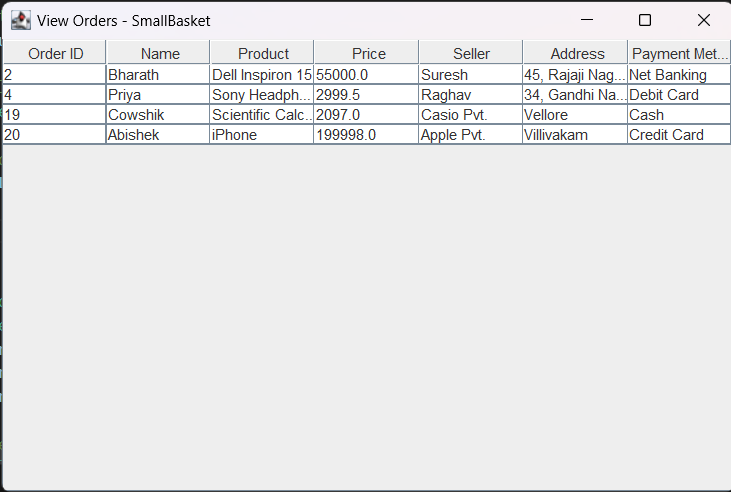


Figure 5: SQL Table

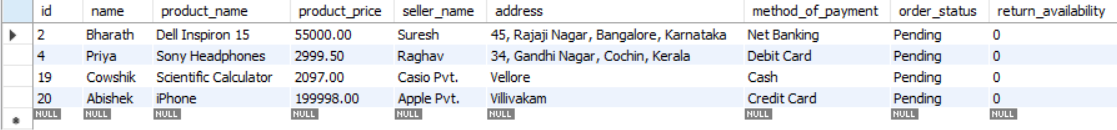
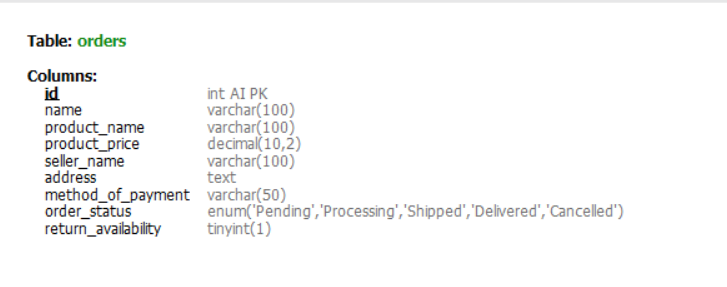


Figure 6: SQL Table Schema



**7.Conclusion**

The **SmallBasket** e-commerce application demonstrates the practical integration of **Java Swing** for user interface development and **MySQL** for database management in creating a functional online shopping platform. Throughout the project, we focused on building a simple yet efficient system that allows users to view, buy, and manage their orders, while also providing the option to delete orders as needed.

The application supports key e-commerce features, including user interaction through a graphical interface, product selection, price calculation based on quantity, order placement, and displaying order information retrieved from the database. Java Swing provided an intuitive and customizable front-end, while MySQL ensured reliable data storage and retrieval, supporting the backend operations like order insertion, deletion, and retrieval.

In terms of performance, the system works effectively for a small-scale e-commerce application, offering smooth interaction between the front-end and back-end. The program's modular design ensures that it can be easily extended to include additional features such as advanced search, payment gateway integration, and enhanced user authentication, making it adaptable to real-world e-commerce needs.

This project has also provided valuable insights into the complexities of developing a database-driven application, especially in the context of user interface design, data handling, and error management. Future improvements could include refining the user interface for a more engaging experience, implementing more complex transaction handling, and integrating real-time features such as product stock updates.

Overall, **SmallBasket** serves as a comprehensive foundation for an e-commerce platform, showcasing both the technical skills required for such development and the potential for future enhancements that could elevate the application to a more complete, commercial-grade system.

**REFERENCES**  
   1. <https://www.javatpoint.com/java-tutorial>

   2. <https://www.wikipedia.org/>

   3. <https://www.w3schools.com/sql/>

   4. SQL | Codecademy