RESTAURANT MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

ARAVINDAN SG	**230701031**
AJAY SRINIVAS R	**230701017**

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024 - 2025

BONAFIDE CERTIFICATE

Certified that this project report **"RESTAURANT MANAGEMENT SYSTEM"** is the bonafide work of **" AJAY SRINIVAS R (230701017) , ARAVINDAN SG (230701031)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Mrs.B.Deepa
Professor(SS) CSE,
Computer Science and Engineering,
Rajalakshmi Engineering College,
Thandalam, Chennai 602 105.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The **Restaurant Management System** is a comprehensive software solution designed to automate and streamline the billing process in restaurants, ensuring efficiency, accuracy, and improved customer service. The system provides a user-friendly interface for staff to manage customer orders, process payments, and generate detailed bills in real-time. It enables secure user authentication, allowing authorised personnel to access and operate the system.

Through the integration of Java Swing for the graphical user interface, JDBC for database connectivity, and MySQL for data storage, the system supports dynamic menu management, real-time calculation of totals, and printing of formatted bills with customer details. It also captures essential customer information, such as names and mobile numbers, which can be used for future reference or marketing purposes.

The **Restaurant Billing System** ensures accurate billing by automatically calculating item prices, discounts, and totals, reducing the risk of human errors. With features like menu item search, bill printing, and secure login, the system provides a seamless experience for both restaurant staff and customers. It is highly scalable, allowing for future enhancements such as inventory management and advanced reporting.

In summary, this system is designed to improve operational efficiency, reduce errors, and enhance the overall dining experience, making it an invaluable tool for modern restaurant management.

INTRODUCTION:

The Restaurant Billing System is a streamlined software solution designed to simplify and automate the core aspects of restaurant sales management - menu item handling and customer billing. In any restaurant, the ability to quickly process orders and generate accurate bills is essential for maintaining customer satisfaction and efficient operations. This system aims to eliminate manual billing errors while providing a straightforward approach to menu management.

The Restaurant Billing System focuses on three essential features that form the backbone of daily restaurant transactions. First, it enables easy addition of new menu items to the system with their respective prices. Second, it allows for quick removal or updating of menu items when they become unavailable or require modifications. Finally, it provides a robust billing module that calculates total charges, including taxes, to generate accurate customer bills efficiently.

Developed using Java, MySQL, and Java Swing, this system creates a reliable and user-friendly platform. Java provides the core functionality for the system's operations, MySQL handles the secure storage and retrieval of menu and billing data, and Java Swing delivers an intuitive graphical interface that restaurant staff can master quickly with minimal training.

This report will detail the design and implementation of the Restaurant Billing System, demonstrating how these focused features work together to create an efficient and practical solution for everyday restaurant billing operations. The emphasis on simplicity and core functionality makes this system particularly suitable for small to medium-sized restaurants looking to modernise their billing process.

OBJECTIVES:

- Provide an intuitive and user-friendly interface for restaurant staff to generate customer bills quickly and accurately.
- Automate the calculation of itemised costs and totals to eliminate manual errors during billing.
- Implement a login system to ensure that only authorised personnel can access the billing application.
- Allow users to easily add, update, and remove menu items, reflecting real-time changes in the application.
- Enable searching and filtering of menu items to expedite item selection during billing.
- Collect and store customer details such as names and mobile numbers for record-keeping and customer relationship management.
- Generate clear and professional itemised bills for customers, including details such as serial number, item name, price, and total cost.
- Use MySQL to store and manage data for users, menu items, and customer transactions, ensuring scalability and reliability.
- Leverage JDBC to facilitate seamless interaction between the application and the database.
- Provide options to modify the application to suit unique requirements, such as adding categories for menu items or customising bill templates.
- Allow expansion to include tax calculations or loyalty programs in the future.

SYSTEM FEATURES AND FUNCTIONALITIES:

1. **Login Module**
    - Validates user credentials using a database.
    - Provides access to the billing system for authenticated users.
2. **Billing Module**
    - Allows users to add menu items to a customer's bill.
    - Displays an itemised bill with serial numbers and a total.
    - Generates a printable bill with customer details and items purchased.
3. **Menu Management Module**
    - Retrieves menu items from the database.
    - Allows searching, filtering, and adding new items to the menu.
4. **Customer Module**
    - Captures customer name and mobile number.
    - Stores customer data in the database for future reference.
5. **Database Integration**
    - **Tables:**
        - `users`: For storing user credentials (username, password).
        - `menu_items`: For storing menu item names and prices.
        - `customers`: For saving customer details (name, mobile).
    - **Database Operations**: **C**reate, **R**etrieve, **U**pdate and **D**elete data.

TECHNOLOGIES USED:

- **Java (Swing):**
  - Java is used to develop the core application logic and graphical user interface (GUI) using the Swing framework.
  - Platform independence allows the application to run on any system with Java Runtime Environment (JRE).
  - Swing provides pre-built components like buttons, labels, text fields, and panels, enabling the creation of an intuitive and interactive UI.
  - Event-driven programming in Swing simplifies handling user interactions, such as button clicks and text input.
  - Robust and versatile programming language with a vast ecosystem.
  - Seamless integration with databases through JDBC.

- **JDBC (Java Database Connectivity):**
  - JDBC is used to connect the application to the MySQL database for seamless data retrieval, insertion, and updates.
  - Supports prepared statements to prevent SQL injection attacks.
  - Provides APIs for executing SQL queries and managing transactions.
  - Offers flexibility to connect to various relational databases.
  - Ensures a smooth flow of data between the application and database.

- **MySQL:**
  - MySQL is used as the relational database management system (RDBMS) for storing and managing application data, including users, menu items, and customer details.
  - Structured data storage in tables with primary and foreign key relationships.
  - SQL queries enable efficient data manipulation and retrieval.
  - Open-source, highly scalable, and reliable for handling large datasets.
  - Provides support for advanced features like indexing and normalisation to improve query performance.

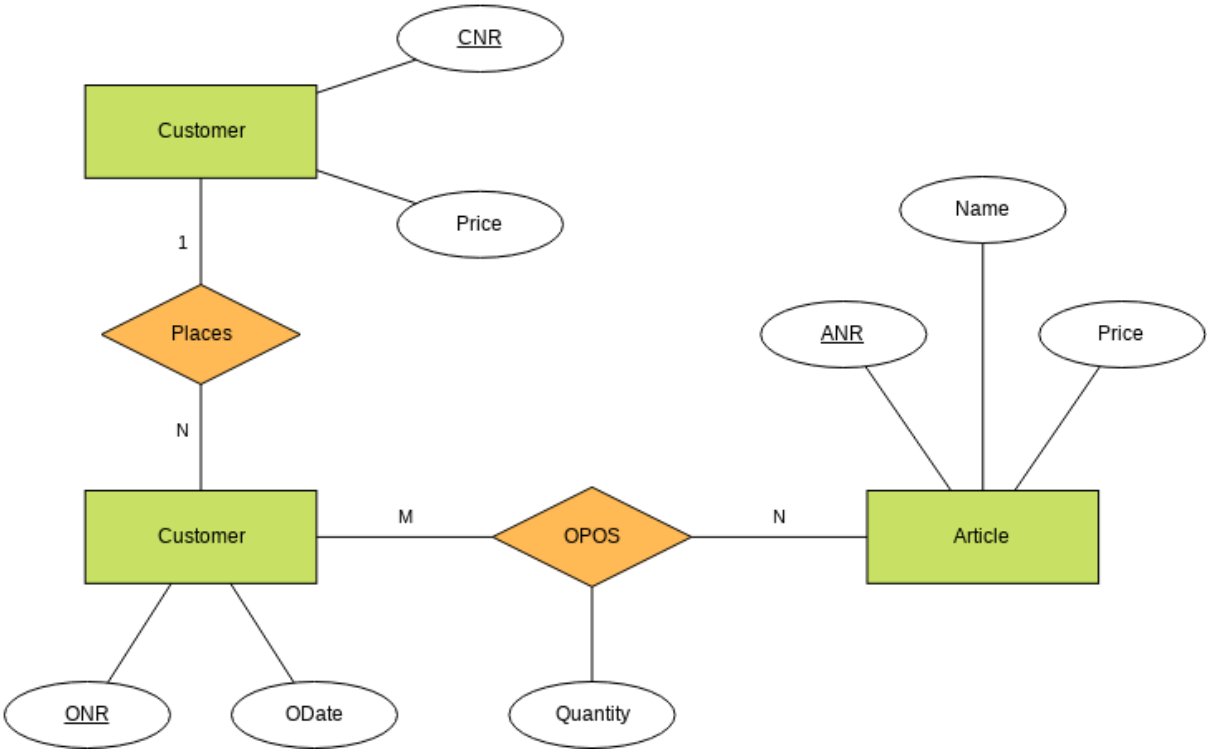
- **Java IDEs (Eclipse, IntelliJ IDEA, or NetBeans):**
  - IDEs are used for developing, debugging, and testing the Java code.
  - Code editors with syntax highlighting, autocompletion, and refactoring tools.
  - Integrated debugging tools to identify and fix runtime errors.
  - Streamlines the development process with built-in tools for version control and project management.
  - Supports plugins and frameworks for enhanced productivity.

- **JDK (Java Development Kit):**
  - The JDK provides the necessary tools and libraries to compile and execute Java applications.
  - Includes the Java compiler (`javac`) and Java Virtual Machine (JVM).
  - Contains essential libraries for building robust applications.
  - Enables developers to write and test Java applications efficiently.
  - Ensures compatibility across different operating systems.

- **JRE (Java Runtime Environment):**
  - JRE provides the runtime environment needed to execute Java applications on client machines.
  - Includes the JVM and runtime libraries required to run compiled Java programs.
  - Handles cross-platform execution seamlessly.
  - Ensures portability of Java applications.
  - Offers a reliable and secure runtime environment.

- **JDBC Driver for MySQL (Connector/J)**
  - The JDBC Driver acts as a bridge between the Java application and the MySQL database.
  - Provides Java APIs for executing SQL statements.
  - Optimised for high-performance database connectivity.
  - Facilitates secure and efficient database communication.
  - Reduces the complexity of writing low-level database access code.

ER DIAGRAM:


CNR
Customer
Price
1
Places
N
Customer
M
OPOS
N
Article
Name
ANR
Price
ONR
ODate
Quantity

DATABASE:

1. **Items Table**

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(50)	NO		NULL	
price	decimal(5,2)	NO		NULL	

2. **Users Table**

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO	UNI	NULL	
password	varchar(50)	NO		NULL	

3. **Customers Table**

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
mobile	varchar(15)	NO		NULL	

PROGRAM CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class RestaurantBillingApp {
public static void main(String[] args) {
SwingUtilities.invokeLater(() -> new LoginFrame());
}
// JDBC Database connection method
public static Connection getConnection() throws SQLException {
String url = "jdbc:mysql://localhost:3306/res_db";
String user = "root";
String password = "password";
return DriverManager.getConnection(url, user, password);
}
}
//Login Frame
class LoginFrame extends JFrame {
private JTextField usernameField;
private JPasswordField passwordField;
public LoginFrame() {
```

```
setTitle("Login");
setSize(350, 200);
setLayout(new GridBagLayout());
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(10, 10, 10, 10);
JLabel titleLabel = new JLabel("Restaurant Login", JLabel.CENTER);
titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
titleLabel.setForeground(new Color(34, 139, 34));
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
add(titleLabel, gbc);
gbc.gridwidth = 1;
gbc.gridy++;
gbc.gridx = 0;
add(new JLabel("Username:"), gbc);
usernameField = new JTextField("username");
usernameField.setPreferredSize(new Dimension(150, 25)); // width, height
gbc.gridx = 1;
add(usernameField, gbc);
gbc.gridy++;
gbc.gridx = 0;
add(new JLabel("Password:"), gbc);
gbc.gridx = 1;
```

```
passwordField = new JPasswordField("password");
add(passwordField, gbc);
JButton loginButton = new JButton("Login");
loginButton.setBackground(new Color(60, 179, 113));
loginButton.setForeground(Color.WHITE);
gbc.gridy++;
gbc.gridx = 0;
gbc.gridwidth = 20;
add(loginButton, gbc);
loginButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String username = usernameField.getText();
String password = new String(passwordField.getPassword());
if (authenticateUser(username, password)) {
dispose();
new BillingFrame();
} else {
JOptionPane.showMessageDialog(null, "Invalid credentials");
}
}
});
setVisible(true);
}
private boolean authenticateUser(String username, String password) {
try (Connection conn = RestaurantBillingApp.getConnection()) {
String query = "SELECT * FROM users WHERE username = ? AND password = ?";
```

```
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
return rs.next();
} catch (SQLException e) {
e.printStackTrace();
return false;
}
}
}
// Billing Frame
// Billing Frame with enhanced bill format and search functionality
class BillingFrame extends JFrame {
private JTextArea billArea;
private List<MenuItem> menuItems;
private double totalAmount = 0;
private JLabel totalLabel;
private JTextField customerNameField;
private JTextField customerMobileField;
private JTextField searchField;
private JPanel itemPanel;
private int serialNumber = 1;
private List<MenuItem> billItems;
public BillingFrame() {
setTitle("Billing System");
```

```
setSize(700, 600);
setLayout(new BorderLayout());
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
billItems = new ArrayList<>();
menuItems = fetchMenuItems();
// Top panel for customer details
JPanel customerPanel = new JPanel(new GridLayout(2, 2, 10, 10));
customerPanel.setBorder(BorderFactory.createTitledBorder("Customer Details"));
customerPanel.add(new JLabel("Customer Name:"));
customerNameField = new JTextField();
customerPanel.add(customerNameField);
customerPanel.add(new JLabel("Mobile Number:"));
customerMobileField = new JTextField();
customerPanel.add(customerMobileField);
add(customerPanel, BorderLayout.NORTH);
// Left panel for search and menu items
// Left panel for search and menu items
JPanel leftPanel = new JPanel();
leftPanel.setLayout(new BorderLayout()); // Use BorderLayout for better arrangement
// Search bar
searchField = new JTextField("Search items...");
searchField.setToolTipText("Search menu items...");
leftPanel.add(searchField, BorderLayout.NORTH);
// Menu item panel (list of menu items)
itemPanel = new JPanel();
```

```
itemPanel.setLayout(new GridLayout(0, 1, 5, 5)); // Display items vertically

itemPanel.setBackground(new Color(240, 248, 255)); // Light background for the items

updateMenuItemsList(menuItems); // Call to update the list of menu items

JScrollPane itemScrollPane = new JScrollPane(itemPanel); // Wrap the item panel in a scroll pane

leftPanel.add(itemScrollPane, BorderLayout.CENTER);

// Adding the left panel to the main layout (East panel remains as before)

add(leftPanel, BorderLayout.WEST);

// Right panel (East) for the action buttons: Add, Update, Remove items

JPanel eastPanel = new JPanel();

eastPanel.setLayout(new BoxLayout(eastPanel, BoxLayout.Y_AXIS)); // Stack buttons vertically

eastPanel.setPreferredSize(new Dimension(200, getHeight())); // Set width of the east panel to 200px

// Add Item Button

JButton addItemButton = new JButton("Add Item");

addItemButton.setBackground(new Color(70, 130, 180));

addItemButton.setForeground(Color.WHITE);

addItemButton.addActionListener(e -> showAddItemDialog());

eastPanel.add(addItemButton);

// Update Item Button

JButton updateItemButton = new JButton("Update Item Price");

updateItemButton.setBackground(new Color(30, 144, 255)); // Dodger Blue

updateItemButton.setForeground(Color.WHITE);

updateItemButton.addActionListener(e -> showUpdateItemDialog());

eastPanel.add(updateItemButton);

// Remove Item Button

JButton removeItemButton = new JButton("Remove Item");
```

```
removeItemButton.setBackground(new Color(255, 69, 0)); // Orange-Red
removeItemButton.setForeground(Color.WHITE);
removeItemButton.addActionListener(e -> showRemoveItemDialog());
eastPanel.add(removeItemButton);
// Adding the East panel to the layout
add(eastPanel, BorderLayout.EAST);
// Center panel for the bill area and total amount
JPanel billPanel = new JPanel(new BorderLayout());
billArea = new JTextArea();
billArea.setEditable(false);
JScrollPane billScrollPane = new JScrollPane(billArea);
totalLabel = new JLabel("Total: $0.00");
totalLabel.setFont(new Font("Arial", Font.BOLD, 16));
totalLabel.setHorizontalAlignment(JLabel.CENTER);
JButton printButton = new JButton("Print Bill");
printButton.addActionListener(e -> printBill());
billPanel.add(billScrollPane, BorderLayout.CENTER);
billPanel.add(totalLabel, BorderLayout.SOUTH);
billPanel.add(printButton, BorderLayout.NORTH);
add(billPanel, BorderLayout.CENTER);
setVisible(true);
}
// Method to fetch menu items from the database
private List<MenuItem> fetchMenuItems() {
List<MenuItem> items = new ArrayList<>();
try (Connection conn = RestaurantBillingApp.getConnection();
```

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM menu_items")) {
while (rs.next()) {
items.add(new MenuItem(rs.getString("name"), rs.getDouble("price")));
}
} catch (SQLException e) {
e.printStackTrace();
}
return items;
}
// Method to show a dialog for adding a new item
private void showAddItemDialog() {
JTextField itemNameField = new JTextField(20);
JTextField itemPriceField = new JTextField(10);
JPanel panel = new JPanel(new GridLayout(2, 2, 10, 10));
panel.add(new JLabel("Item Name:"));
panel.add(itemNameField);
panel.add(new JLabel("Item Price:"));
panel.add(itemPriceField);
int result = JOptionPane.showConfirmDialog(this, panel, "Add New Item",
JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);
if (result == JOptionPane.OK_OPTION) {
String itemName = itemNameField.getText().trim();
String itemPriceText = itemPriceField.getText().trim();
if (itemName.isEmpty() || itemPriceText.isEmpty()) {
JOptionPane.showMessageDialog(this, "Please enter both item name and price.");
```

```
return;
}
try {
double itemPrice = Double.parseDouble(itemPriceText);
addItemToDatabase(itemName, itemPrice);
refreshMenuItems();
JOptionPane.showMessageDialog(this, "Item added successfully!");
} catch (NumberFormatException e) {
JOptionPane.showMessageDialog(this, "Invalid price format. Please enter a valid number.");
}
}
}
// Method to insert a new item into the menu_items table
private void addItemToDatabase(String name, double price) {
String query = "INSERT INTO menu_items (name, price) VALUES (?, ?)";
try (Connection conn = RestaurantBillingApp.getConnection();
PreparedStatement stmt = conn.prepareStatement(query)) {
stmt.setString(1, name);
stmt.setDouble(2, price);
stmt.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
JOptionPane.showMessageDialog(this, "Failed to add item to the database.");
}
}
// // Method to update the menu list panel
```

```
private void updateMenuItemsList(List<MenuItem> items) {
itemPanel.removeAll();
for (MenuItem item : items) {
JButton itemButton = new JButton(item.getName() + " - ₹" + item.getPrice());
itemButton.setBackground(new Color(72, 209, 204));
itemButton.setForeground(Color.WHITE);
itemButton.addActionListener(e -> addItemToBill(item));
itemPanel.add(itemButton);
}
itemPanel.revalidate();
itemPanel.repaint();
}
// Method to search and update menu items
private void searchAndUpdateMenuList() {
String searchText = searchField.getText().toLowerCase();
List<MenuItem> filteredItems = new ArrayList<>();
for (MenuItem item : menuItems) {
if (item.getName().toLowerCase().contains(searchText)) {
filteredItems.add(item);
}
}
updateMenuItemsList(filteredItems);
}
//Remove item
private void showRemoveItemDialog() {
String[] itemNames = menuItems.stream()
```

```
.map(MenuItem::getName)
.toArray(String[]::new);
String selectedItem = (String) JOptionPane.showInputDialog(
this,
"Select Item to Remove:",
"Remove Item",
JOptionPane.PLAIN_MESSAGE,
null,
itemNames,
itemNames[0]
);
if (selectedItem != null) {
removeItemFromDatabase(selectedItem);
refreshMenuItems();
JOptionPane.showMessageDialog(this, "Item removed successfully!");
}
}
private void removeItemFromDatabase(String name) {
String query = "DELETE FROM menu_items WHERE name = ?";
try (Connection conn = RestaurantBillingApp.getConnection();
PreparedStatement stmt = conn.prepareStatement(query)) {
stmt.setString(1, name);
stmt.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
JOptionPane.showMessageDialog(this, "Failed to remove item from the database.");
```

```
}
}
//Update item
private void showUpdateItemDialog() {
String[] itemNames = menuItems.stream()
.map(MenuItem::getName)
.toArray(String[]::new);
JPanel panel = new JPanel(new GridLayout(2, 2, 10, 10));
JComboBox<String> itemComboBox = new JComboBox<>(itemNames);
JTextField newPriceField = new JTextField();
panel.add(new JLabel("Select Item:"));
panel.add(itemComboBox);
panel.add(new JLabel("New Price:"));
panel.add(newPriceField);
int result = JOptionPane.showConfirmDialog(
this,
panel,
"Update Item Price",
JOptionPane.OK_CANCEL_OPTION,
JOptionPane.PLAIN_MESSAGE
);
if (result == JOptionPane.OK_OPTION) {
String selectedItem = (String) itemComboBox.getSelectedItem();
String newPriceText = newPriceField.getText().trim();
try {
double newPrice = Double.parseDouble(newPriceText);
```

```
updateItemPriceInDatabase(selectedItem, newPrice);
refreshMenuItems();
JOptionPane.showMessageDialog(this, "Price updated successfully!");
} catch (NumberFormatException e) {
JOptionPane.showMessageDialog(this, "Invalid price format. Please enter a valid number.");
}
}
}
private void updateItemPriceInDatabase(String name, double newPrice) {
String query = "UPDATE menu_items SET price = ? WHERE name = ?";
try (Connection conn = RestaurantBillingApp.getConnection();
PreparedStatement stmt = conn.prepareStatement(query)) {
stmt.setDouble(1, newPrice);
stmt.setString(2, name);
stmt.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
JOptionPane.showMessageDialog(this, "Failed to update item price in the database.");
}
}
// Method to add item to bill with serial number
private void addItemToBill(MenuItem item) {
billItems.add(item);
billArea.append(serialNumber + ". " + item.getName() + "\t₹" + item.getPrice() + "\n");
totalAmount += item.getPrice();
totalLabel.setText("Total: ₹" + String.format("%.2f", totalAmount));
```

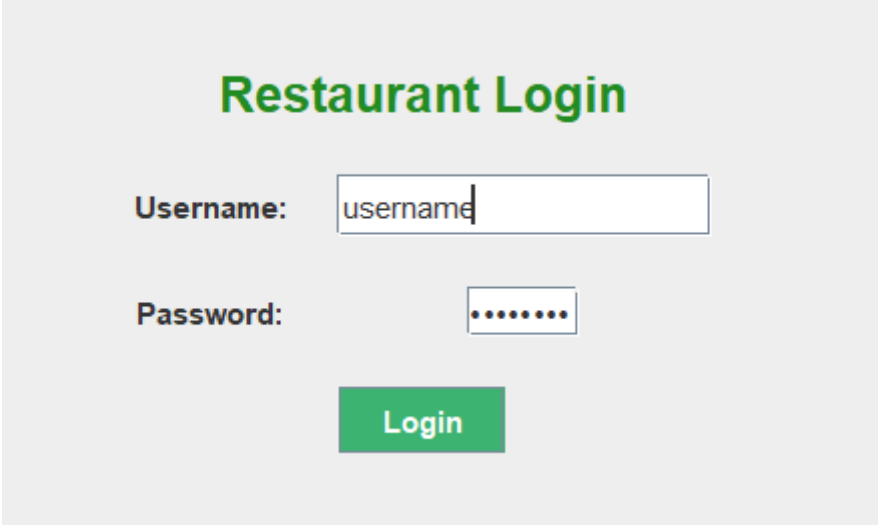
```
serialNumber++;
}
// Method to print bill with formatted details
private void printBill() {
if (customerNameField.getText().isEmpty() || customerMobileField.getText().isEmpty()) {
JOptionPane.showMessageDialog(this, "Please enter customer details");
return;
}
try (Connection conn = RestaurantBillingApp.getConnection()) {
String query = "INSERT INTO customers (name, mobile) VALUES (?, ?)";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, customerNameField.getText());
stmt.setString(2, customerMobileField.getText());
stmt.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
}
// Printing formatted bill
StringBuilder billContent = new StringBuilder();
billContent.append("******** ESTIMATE ********\n");
billContent.append("Restaurant XYZ\n");
billContent.append("Customer Name: ").append(customerNameField.getText()).append("\n");
billContent.append("Mobile Number:
").append(customerMobileField.getText()).append("\n\n");
billContent.append("Items:\n");
billContent.append("No.\tItem\tPrice\n");
int serialNo = 1;
```

```
for (MenuItem item : billItems) {
billContent.append(serialNo++).append(".\t").append(item.getName())
.append("\t₹").append(item.getPrice()).append("\n");
}
billContent.append("\nTotal: ₹").append(String.format("%.2f", totalAmount)).append("\n");
billArea.setText(billContent.toString());
try {
billArea.print();
} catch (Exception e) {
e.printStackTrace();
}
}
// Refresh menu items from the database
public void refreshMenuItems() {
menuItems = fetchMenuItems();
updateMenuItemsList(menuItems);
}
}
// MenuItem class
class MenuItem {
private String name;
private double price;
public MenuItem(String name, double price) {
this.name = name;
this.price = price;
}
```

```
public String getName() {
return name;
}
public double getPrice() {
return price;
}
```

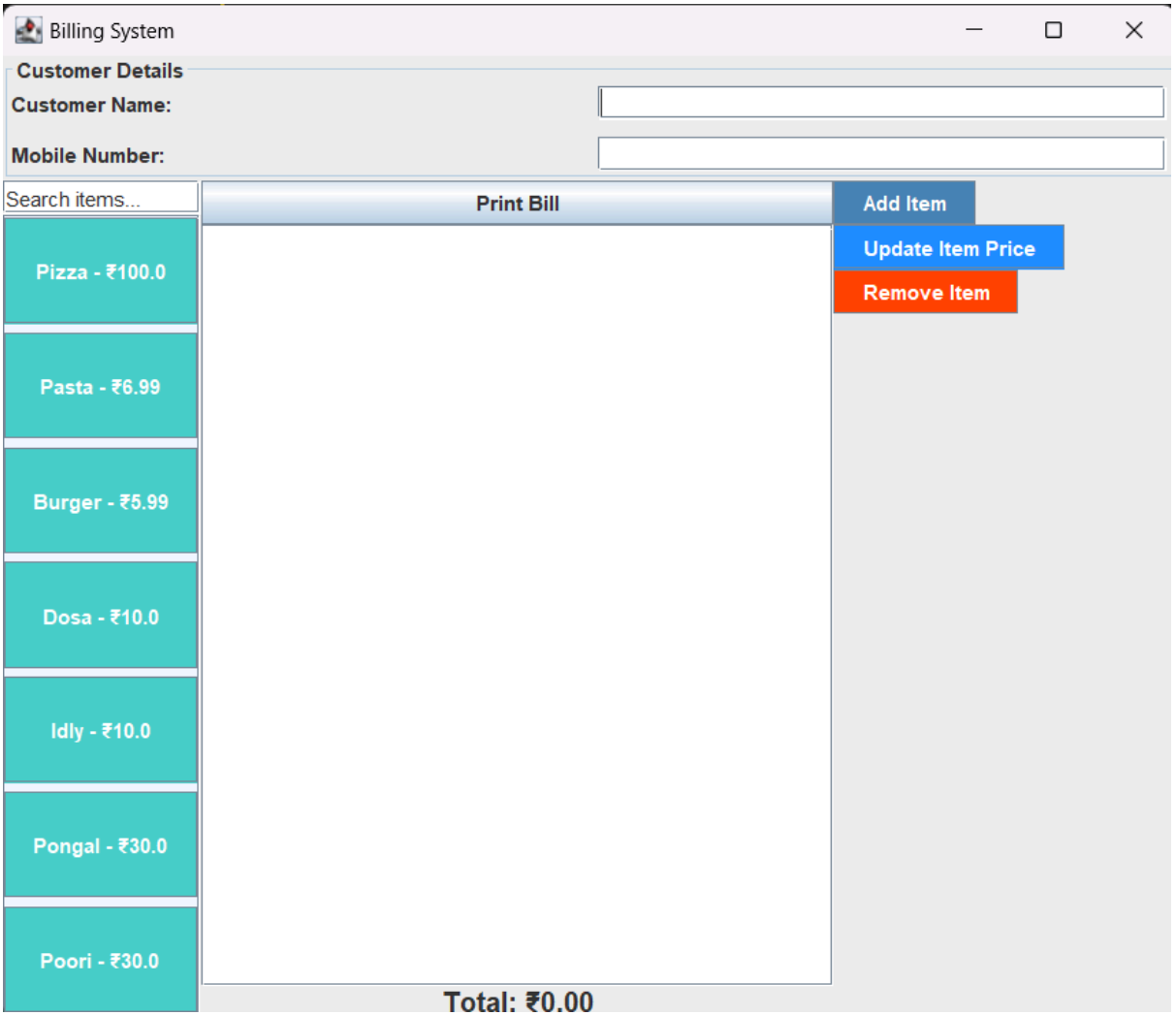
RESULTS:

1.User Authentication:

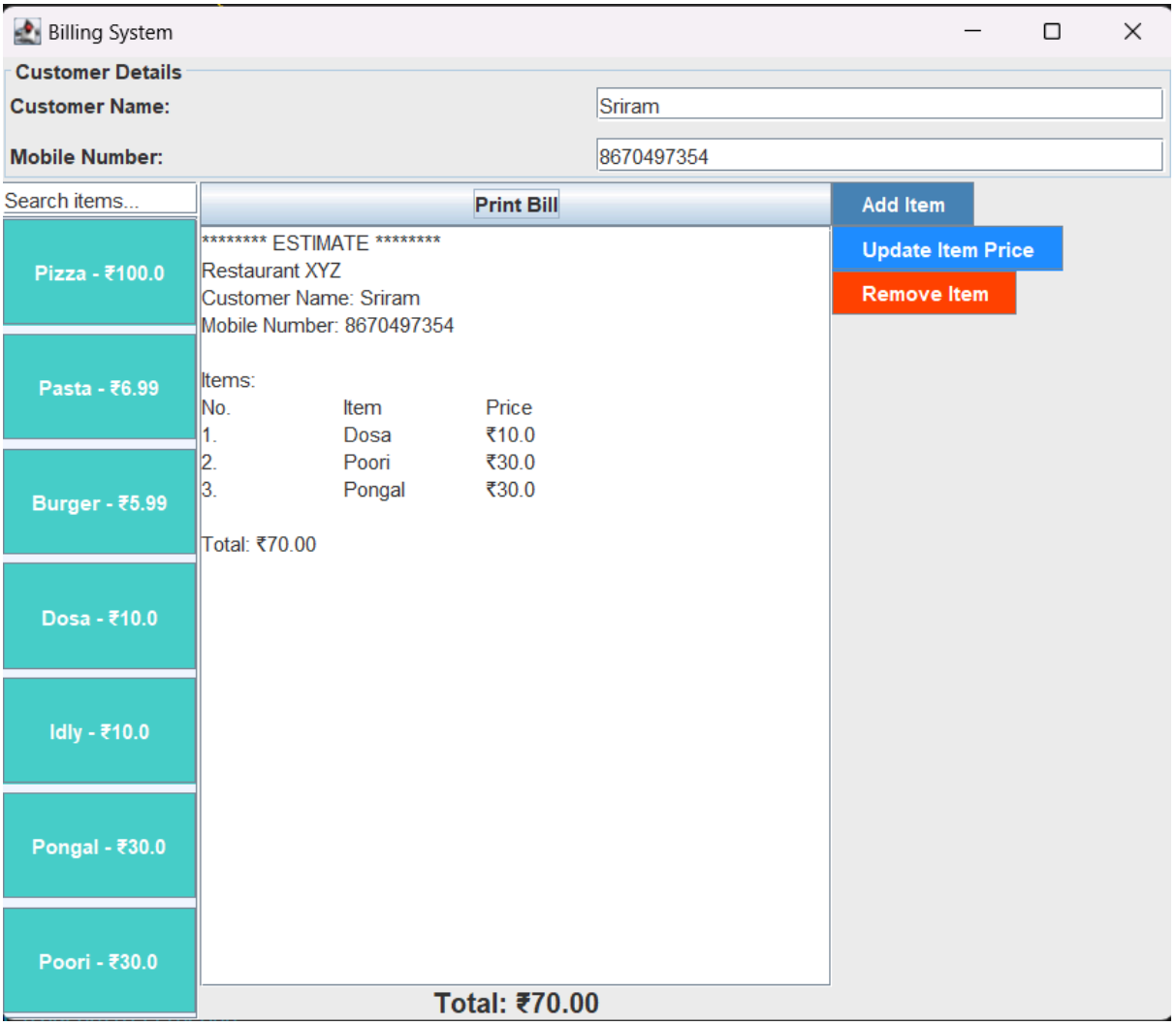



```
}
```

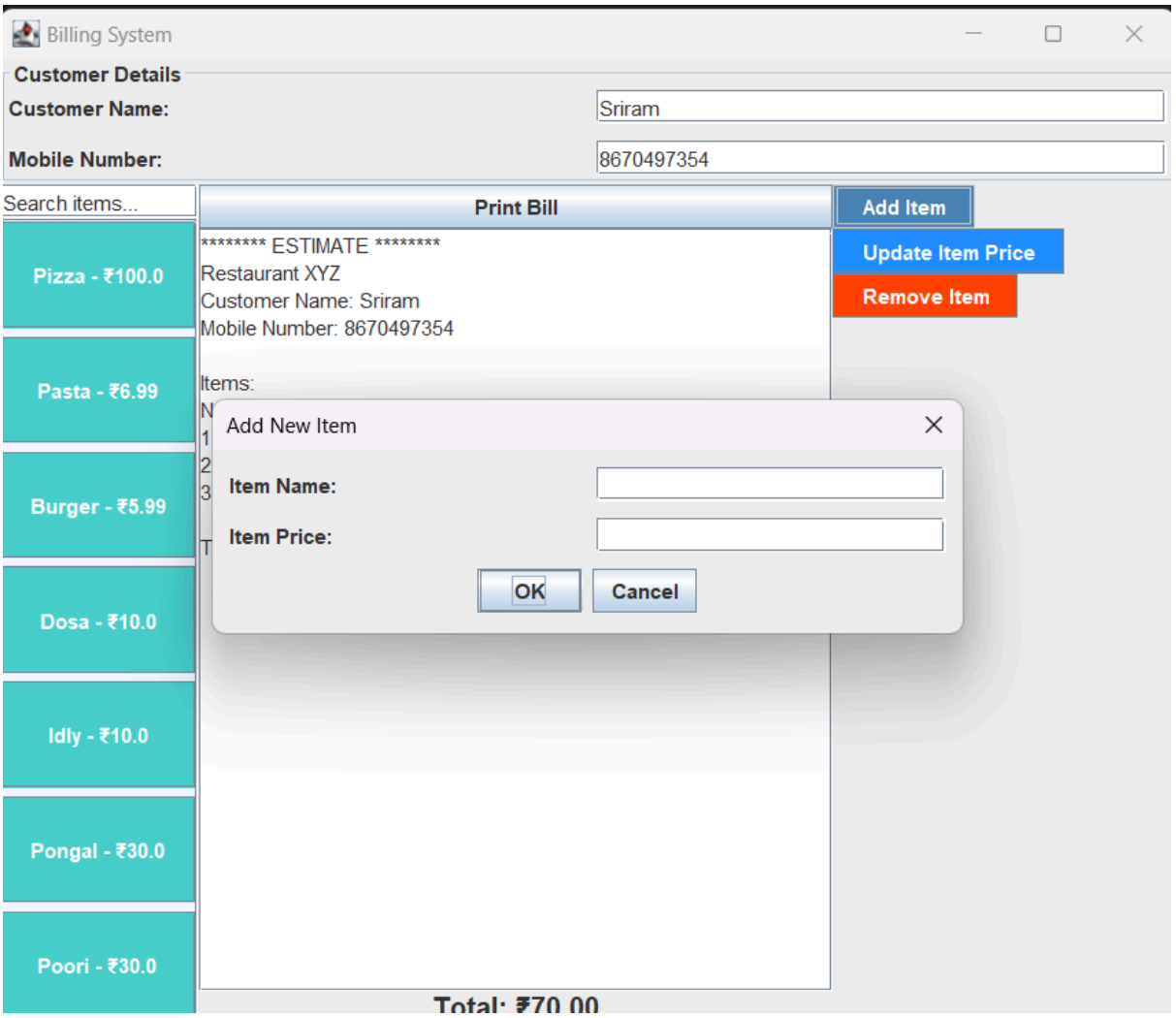
2.Home Page


Billing System
Customer Details
Customer Name:
Mobile Number:
Search items...
Print Bill
Add Item
Update Item Price
Remove Item
Pizza - ₹100.0
Pasta - ₹6.99
Burger - ₹5.99
Dosa - ₹10.0
Idly - ₹10.0
Pongal - ₹30.0
Poori - ₹30.0
Total: ₹0.00

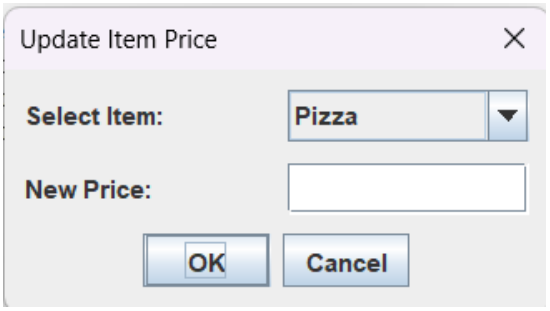
3.Billing Page



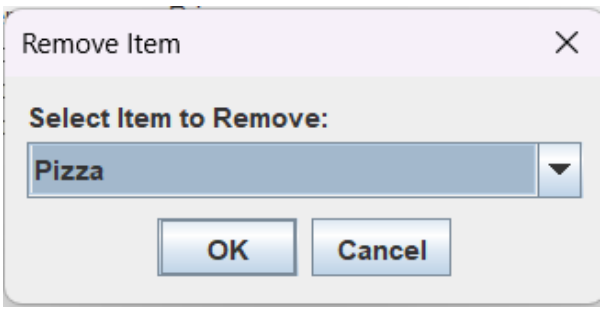
4.Add new item



5.Update



6.Remove



CONCLUSION:

The **Restaurant Billing System** is an innovative solution designed to streamline restaurant operations by automating the billing process, improving accuracy, and enhancing customer satisfaction. By leveraging modern technologies like Java Swing for the user interface, JDBC for database connectivity, and MySQL for data storage, the system provides a seamless and efficient platform for managing menu items, customer information, and transactions.

The system's key features, such as secure login, dynamic menu management, real-time billing, and the ability to generate formatted, printable bills, make it an indispensable tool for any restaurant. It ensures that customer orders are processed quickly and accurately, reducing the chances of human errors in billing. Moreover, by capturing and storing customer data, the system enables better customer relationship management, which can be used for marketing or loyalty programs.

The scalability of the system ensures that as the restaurant grows, the application can easily accommodate additional features like inventory tracking and advanced reporting tools. The integration of secure login and data protection mechanisms further ensures that sensitive business and customer information is safely handled.

In conclusion, the **Restaurant Billing System** significantly enhances operational efficiency, reduces time-consuming tasks, and provides a professional, error-free service, ultimately contributing to the restaurant's success and growth in a competitive market.

REFERENCES:

https://www.geeksforgeeks.org/introduction-to-java-swing/

https://docs.oracle.com/javase/tutorial/uiswing/

https://dev.mysql.com/doc/