PROJECT REPORT

COURSE NAME: OBJECT ORIENTED PROGRAMMING

COURSE CODE: 24SJPBCST304

Group Members: Karthika Bensilal

Alwin M Benny

Karthika Unni

Deborah Elsa Halu

SwiftServe

Digital Canteen Management System

SwiftServe is a campus canteen management system built using Java SE with a Swing/AWT frontend for intuitive user interaction. Structured around a 3-tier architecture, it separates business logic and database management for maintainability. The application applies object-oriented design, connecting to MySQL for secure order and payment management via JDBC (MySQL Connector/J). With adaptive layouts, real-time status updates, and Git/GitHub for version control, SwiftServe streamlines food ordering, reduces wait times, and supports collaborative development.

INDEX

1. Introduction

SwiftServe is a digital canteen order management system designed to revolutionize campus food services by eliminating queues and streamlining the order, payment, and collection process. The central vision of SwiftServe is to empower students to order and pay for their food through a digital platform and collect their items using instant token assignment, drastically reducing wait times and crowding in campus canteens. SwiftServe benefits three types of users:

- **Students**: Primary customers who order food.
- **Canteen Staff/Admins**: Food preparers and supervisors who manage orders and mark them complete.
- **IT Developers**: Maintain, improve, and ensure system stability.

SwiftServe consists of two tightly integrated applications:

- **Order App**: For students, enabling menu browsing, order placement, payment, and receiving instant confirmation via a unique token.
- **Admin App**: For canteen staff, offering a dashboard to monitor incoming orders, update statuses, and notify students of completed orders.

Both apps communicate in real time, ensuring continuous data flow and resilience even if an application or network connection fails.

2. Objectives

- **Remove queues:** Allow students to order and pay without physical waiting.
- **Provide instant feedback:** Generate a unique token for each order within two seconds.
- **Enable seamless order management:** Staff receive real-time notifications for new orders and completion requirements.
- **Preserve order and transaction history:** Maintain complete logs for future audits and tracking.
- **Ensure robust data security:** Orders and transactions are safely stored, preserving data even in case of system crashes or connection issues.
- **Deliver an intuitive user interface:** Both apps are designed for easy accessibility and operation by students and staff.

- **Support future scalability:** Architecture allows for features like stock management to be added later.
- **Maintain modular and maintainable code:** Design separates logic, interface, and data layers, supporting easy upgrades and changes.

3. Requirement Specification

Hardware Requirements:

- Student smartphones/tablets/computers for accessing the Order App.
- Canteen staff monitor/tablet for the Kitchen App.
- Central server for backend processing and database management.
- Reliable WiFi or LAN connectivity across campus for real-time syncing.

Software Requirements:

- Operating System: Windows/Linux (server/backend), Android/iOS (student app/client devices).
- Database: MySQL for secure, persistent data storage.
- Programming Language: Java, with SOLID principles for maintainability and reliability.
- Front-end: Java Swing (desktop apps) or Android Studio for mobile environments.
- Secure payment API, supporting digital transactions (Paytm, Google Pay, campus wallet, etc.).
- Real-time communication modules/APIs for syncing orders between student and kitchen apps.

4. Design Document

System Architecture Overview:

- Clearly divides logic between interface, processing, and data storage layers.
- Two main apps (Order App & Admin App) communicate via a shared SQL database; real-time sync is enabled using efficient backend services.
- The architecture uses SOLID principles to separate user interface, business logic, and database modules.

SwiftServe Class Diagram Table:

Class Name	**Attributes / Fields**	**Key Methods or Responsibilities**
CanteenOrderSystem	-	Main entry, initializes services and UI
Order	token, itemName, quantity, price, orderTime, isComplete	getTotalPrice(), getFormattedOrderTime(), setters/getters
OrderRepository	-	save(Order), findByToken(int), findAllPending(), updateStatus(int, boolean)
MySQLOrderRepository	JDBC_URL, USER, PASS	getConnection(), save(), findByToken(), findAllPending(), updateStatus()
OrderSystemLogic	repository, listeners	addOrder(), markOrderAsComplete(), getPendingOrders(), addOrderUpdateListener()
AuthService	STUDENT_USER, STUDENT_PASS, CANTEEN_USER, CANTEEN_PASS	authenticate(username, password), UserRole enum
AdminUI	orderSystemLogic, orderTable, tableModel, buttons	Displays/administers orders, marks completion, refresh
UI	orderSystemLogic, itemNameField, quantitySpinner, priceField, etc	Customer-facing UI, places order, handles payment
AdminTokenSlip	detailsPanel	Displays token/order details in UI

Class Name	**Attributes / Fields**	**Key Methods or Responsibilities**
LoginFrame	usernameField, passwordField, loginButton, authService, orderSystemLogic	Handles user/admin login, launches appropriate UI
PaymentDialog	order, transactionIdField, submitButton	Handles secure payment and receipt printing

Database Design:

- **Orders Table**: Stores all placed orders and status.
- **Users Table**: Student/staff credentials.
- **Menu Table**: List of food items and availability.
- **Transactions Table**: Logs every payment for audit or refund.
- All tables are connected via orderID, userID, and itemID for simple and powerful queries.

5. Source Code

- **SwiftServe/ (Project Root)**
    - **src/**
        - **com/canteen/app/**
            - **CanteenOrderSystem.java**
        - **com/canteen/model/**
            - **Order.java**
        - **com/canteen/repository/**
            - **MySQLOrderRepository.java**
            - **OrderRepository.java**
        - **com/canteen/service/**
            - **AuthService.java**
            - **OrderSystemLogic.java**
        - **com/canteen/ui/**
            - **AdminTokenSlip.java**
            - **AdminUI.java**
            - **LoginFrame.java**
            - **PaymentDialog.java**
            - **UI.java**
    - **lib/**
        - **mysql-connector-j-8.0.x.jar**
    - **resources/**
        - **(images, logos, configs)**

A) App (.app)

Entry point and main orchestrator for the entire system (CanteenOrderSystem.java). Responsible for starting up the application, initializing core modules, and wiring dependencies together.

1.CanteenOrderSystem.java(app)

```
package com.canteen.app;


import com.canteen.repository.MySQLOrderRepository;

import com.canteen.repository.OrderRepository;

import com.canteen.service.OrderSystemLogic;

import com.canteen.ui.AdminUI;

import com.canteen.ui.UI;

import com.canteen.ui.LoginFrame;

import javax.swing.JOptionPane;

import javax.swing.SwingUtilities;


public class CanteenOrderSystem {

  public static void main(String[] args) {

    try {

      Class.forName("com.mysql.cj.jdbc.Driver");

    } catch (ClassNotFoundException e) {

      JOptionPane.showMessageDialog(null,

        "MySQL JDBC Driver not found. Ensure the connector JAR is in your classpath.",

        "Driver Error", JOptionPane.ERROR_MESSAGE);

      e.printStackTrace();

      return;

    }
```

```
        SwingUtilities.invokeLater(() -> {
            try {
                OrderRepository repository = new MySQLOrderRepository();
                OrderSystemLogic logic = new OrderSystemLogic(repository);
                LoginFrame login = new LoginFrame(logic);
                login.setVisible(true);

            } catch (Exception e) {
                JOptionPane.showMessageDialog(null,
                    "Failed to initialize application. Check database connection and credentials.",
                    "Initialization Error", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        });
    }
}
```

B) Model (.model)

Includes data structure classes that represent the core objects used throughout the application—such as orders, menu items, and any other domain entities (Order.java). Everything related to what data looks like and how it is stored in-memory.

2.Order.java(model)

```
package com.canteen.model;


import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;


public class Order {
    private Integer token;

    private String itemName;

    private int quantity;

    private float price;

    private LocalDateTime orderTime;

    private boolean isComplete;


    public Order(String itemName, int quantity, float price) {

        this.itemName = itemName;

        this.quantity = quantity;

        this.price = price;

        this.orderTime = LocalDateTime.now();

        this.isComplete = false;

    }

    public Order(int token, String itemName, int quantity, float price, LocalDateTime orderTime, boolean
isComplete) {

        this.token = token;

        this.itemName = itemName;

        this.quantity = quantity;

        this.price = price;

        this.orderTime = orderTime;

        this.isComplete = isComplete;

    }

    public Integer getToken() { return token; }

    public String getItemName() { return itemName; }

    public int getQuantity() { return quantity; }

    public float getPrice() { return price; }
```

```
  public LocalDateTime getOrderTime() { return orderTime; }

  public boolean isComplete() { return isComplete; }

  public float getTotalPrice() { return quantity * price; }


  public String getFormattedOrderTime() {

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    return orderTime.format(formatter);

  }

  public void setToken(int token) { this.token = token; }

  public void setComplete(boolean complete) { isComplete = complete; }

}
```

C) Repository (.repository)

Houses all Data Access Object (DAO) interfaces and implementations responsible for database transactions. Manages CRUD operations to the MySQL backend via JDBC connection (OrderRepository.java, MySQLOrderRepository.java). This is the gatekeeper between your service/business logic and the persistent storage.

3.MySQLOrderRespository.java

```
package com.canteen.repository;


import com.canteen.model.Order;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import java.time.LocalDateTime;

public class MySQLOrderRepository implements OrderRepository {

  private final String JDBC_URL =
"jdbc:mysql://localhost:3306/canteen_db?useSSL=false&serverTimezone=UTC&allowPublicKeyRetrieval=tr
ue";
```

```
    private final String USER = "root";

    private final String PASS = "alwin123";


    private Connection getConnection() throws SQLException {

        return DriverManager.getConnection(JDBC_URL, USER, PASS);

    }


    @Override

    public Order save(Order order) {

        String sql = "INSERT INTO orders (item_name, quantity, price, order_time, is_complete) VALUES (?, ?, ?, ?,
?)";

        try (Connection conn = getConnection();

             PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {


            stmt.setString(1, order.getItemName());

            stmt.setInt(2, order.getQuantity());

            stmt.setFloat(3, order.getPrice());

            stmt.setTimestamp(4, Timestamp.valueOf(order.getOrderTime()));

            stmt.setBoolean(5, order.isComplete());


            stmt.executeUpdate();


            try (ResultSet rs = stmt.getGeneratedKeys()) {

                if (rs.next()) {

                    order.setToken(rs.getInt(1));

                }

            }
        return order;

    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("Database error during order save.", e);
    }
}

@Override
public Optional<Order> findByToken(int token) {
    String sql = "SELECT * FROM orders WHERE token = ?";
    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, token);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return Optional.of(createOrderFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return Optional.empty();
}
@Override
public List<Order> findAllPending() {
  List<Order> orders = new ArrayList<>();
  String sql = "SELECT * FROM orders WHERE is_complete = FALSE ORDER BY order_time ASC";
  try (Connection conn = getConnection();
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
      orders.add(createOrderFromResultSet(rs));
    }
  } catch (SQLException e) {
    e.printStackTrace();
  }
  return orders;
}
@Override
public void updateStatus(int token, boolean isComplete) {
  String sql = "UPDATE orders SET is_complete = ? WHERE token = ?";
  try (Connection conn = getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {

    stmt.setBoolean(1, isComplete);
    stmt.setInt(2, token);
    stmt.executeUpdate();

  } catch (SQLException e) {
    e.printStackTrace();
  }
```

```
    }


    private Order createOrderFromResultSet(ResultSet rs) throws SQLException {
        return new Order(
            rs.getInt("token"),
            rs.getString("item_name"),
            rs.getInt("quantity"),
            rs.getFloat("price"),
            rs.getTimestamp("order_time").toLocalDateTime(),
            rs.getBoolean("is_complete")
        );
    }
}
```

4.OrderRepository.java

```
package com.canteen.repository;


import com.canteen.model.Order;
import java.util.List;
import java.util.Optional;


public interface OrderRepository {
    Order save(Order order);
    Optional<Order> findByToken(int token);
    List<Order> findAllPending();
    void updateStatus(int token, boolean isComplete);
}
```

D) Service (.service)

Handles core business logic, application processes, and workflows. Classes here manage authentication, order processing, and coordination between the UI, Model, and Repository (OrderSystemLogic.java, AuthService.java). This is the "brains" connecting UI actions to backend data changes.

5.AuthService.java(service)

```
package com.canteen.service;


public class AuthService {


  private static final String STUDENT_USER = "student";

  private static final String STUDENT_PASS = "pass";


  private static final String CANTEEN_USER = "admin";

  private static final String CANTEEN_PASS = "canteen123";


  public enum UserRole {

    STUDENT,

    CANTEEN_ADMIN,

    INVALID

  }

  public UserRole authenticate(String username, String password) {

    if (username.equals(STUDENT_USER) && password.equals(STUDENT_PASS)) {

      return UserRole.STUDENT;

    } else if (username.equals(CANTEEN_USER) && password.equals(CANTEEN_PASS)) {

      return UserRole.CANTEEN_ADMIN;

    } else {
            return UserRole.INVALID;
        }
    }
}
```

6.OrderSystemLogic.java

```
package com.canteen.service;


import com.canteen.model.Order;
import com.canteen.repository.OrderRepository;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;


public class OrderSystemLogic {
    private final OrderRepository repository;
    private final List<OrderUpdateListener> listeners;


    public OrderSystemLogic(OrderRepository repository) {
        this.repository = repository;
        this.listeners = new ArrayList<>();
    }


    public void addOrder(Order order) {
        Order persistedOrder = repository.save(order);
        System.out.println("Order added to DB: Token #" + persistedOrder.getToken() + " - " +
persistedOrder.getItemName());
        notifyOrderAdded(persistedOrder);
}


public void markOrderAsComplete(int token) {
   repository.updateStatus(token, true);
   Optional<Order> orderToComplete = repository.findByToken(token);
   orderToComplete.ifPresent(order -> {
      System.out.println("Order marked complete in DB: Token #" + order.getToken());
      notifyOrderStatusChanged(order);
   });
}
public List<Order> getPendingOrders() {
   return repository.findAllPending();
}
public void addOrderUpdateListener(OrderUpdateListener listener) {
   listeners.add(listener);
}


public interface OrderUpdateListener {
   void orderAdded(Order order);
   void orderStatusChanged(Order order);
}
private void notifyOrderAdded(Order order) {
   for (OrderUpdateListener listener : listeners) {
      listener.orderAdded(order);
   }
}
```

```
    private void notifyOrderStatusChanged(Order order) {

        for (OrderUpdateListener listener : listeners) {

            listener.orderStatusChanged(order);

        }

    }

}
```

E) UI (.ui)

Contains all Java JFrame and JPanel user interface classes, including login screens, admin and student dashboards, payment dialogs, and UI for order/task management (LoginFrame, UI, AdminUI, AdminTokenSlip, PaymentDialog). This is the presentation layer that interacts directly with users.

7.AdminTokenSlip.java

```
package com.canteen.ui;



import com.canteen.model.Order;

import javax.swing.*;

import java.awt.*;

class AdminTokenSlip extends JFrame {

    private JPanel detailsPanel;

    public AdminTokenSlip() {

        setTitle("Order Token Slip");

        setSize(300, 250);

        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        setLocationRelativeTo(null);

        setResizable(false);



        detailsPanel = new JPanel();

        detailsPanel.setLayout(new BoxLayout(detailsPanel, BoxLayout.Y_AXIS));
        detailsPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));


        JScrollPane scrollPane = new JScrollPane(detailsPanel);
        add(scrollPane, BorderLayout.CENTER);
    }


    public void displayTokenSlip(Order order) {
        detailsPanel.removeAll();


        JLabel titleLabel = new JLabel("--- Canteen Order Slip ---");
        titleLabel.setFont(new Font("SansSerif", Font.BOLD, 16));
        titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        detailsPanel.add(titleLabel);
        detailsPanel.add(Box.createRigidArea(new Dimension(0, 10)));


        addDetailRow("Token Number:", order.getToken() != null ? String.valueOf(order.getToken()) : "N/A");
        addDetailRow("Item Name:", order.getItemName());
        addDetailRow("Quantity:", String.valueOf(order.getQuantity()));
        addDetailRow("Price per Item:", String.format("$%.2f", order.getPrice()));
        detailsPanel.add(Box.createRigidArea(new Dimension(0, 5)));


        JLabel totalLabel = new JLabel(String.format("Total: $%.2f", order.getTotalPrice()));
        totalLabel.setFont(new Font("SansSerif", Font.BOLD, 14));
        totalLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        detailsPanel.add(totalLabel);
        detailsPanel.add(Box.createRigidArea(new Dimension(0, 5)));
        addDetailRow("Order Time:", order.getFormattedOrderTime());
        addDetailRow("Status:", order.isComplete() ? "Completed" : "Pending");
        detailsPanel.add(Box.createVerticalGlue());


        detailsPanel.revalidate();
        detailsPanel.repaint();
        setVisible(true);
    }


    private void addDetailRow(String labelText, String valueText) {
        JPanel rowPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setFont(new Font("SansSerif", Font.BOLD, 12));
        JLabel value = new JLabel(valueText);
        value.setFont(new Font("SansSerif", Font.PLAIN, 12));
        rowPanel.add(label);
        rowPanel.add(value);
        detailsPanel.add(rowPanel);
    }
}
```

8.AdminUI.java

```
package com.canteen.ui;


import com.canteen.model.Order;
import com.canteen.service.OrderSystemLogic;
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableModel;

import java.awt.*;

import java.util.List;

import java.util.Vector;


public class AdminUI extends JFrame implements OrderSystemLogic.OrderUpdateListener {

  private OrderSystemLogic orderSystemLogic;

  private JTable orderTable;

  private DefaultTableModel tableModel;

  private JButton markCompleteButton;

  private JButton refreshButton;

  private JButton logoutButton;


  public AdminUI(OrderSystemLogic orderSystemLogic) {

    this.orderSystemLogic = orderSystemLogic;

    this.orderSystemLogic.addOrderUpdateListener(this);


    setTitle("Canteen Order System - Admin");

    setSize(700, 500);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setLocationRelativeTo(null);

    setLayout(new BorderLayout(10, 10));

    ((JPanel) getContentPane()).setBorder(new EmptyBorder(10, 10, 10, 10));


    initComponents();

    layoutComponents();

    addEventHandlers();
    refreshOrderTable();
  }
  private void initComponents() {
    String[] columnNames = {"Token", "Item", "Quantity", "Price", "Total", "Status", "Time"};
    tableModel = new DefaultTableModel(columnNames, 0) {
      @Override
      public boolean isCellEditable(int row, int column) {
        return false;
      }
    };
    orderTable = new JTable(tableModel);
    orderTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

    markCompleteButton = new JButton("Mark as Complete");
    refreshButton = new JButton("Refresh Orders");
    logoutButton = new JButton("Logout");
  }

  private void layoutComponents() {
    JScrollPane scrollPane = new JScrollPane(orderTable);
    add(scrollPane, BorderLayout.CENTER);
    JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 15, 5));
    buttonPanel.add(logoutButton);
    buttonPanel.add(refreshButton);
    buttonPanel.add(markCompleteButton);
    add(buttonPanel, BorderLayout.SOUTH);
```

```
        JLabel headerLabel = new JLabel("Pending Canteen Orders", SwingConstants.CENTER);

        headerLabel.setFont(new Font("SansSerif", Font.BOLD, 18));

        add(headerLabel, BorderLayout.NORTH);

    }


    private void addEventHandlers() {

        markCompleteButton.addActionListener(e -> {

            int selectedRow = orderTable.getSelectedRow();

            if (selectedRow != -1) {

                int token = (int) tableModel.getValueAt(selectedRow, 0);

                orderSystemLogic.markOrderAsComplete(token);

            } else {

                JOptionPane.showMessageDialog(AdminUI.this, "Please select an order to mark complete.", "No
Order Selected", JOptionPane.WARNING_MESSAGE);

            }

        });

        refreshButton.addActionListener(e -> refreshOrderTable());

        logoutButton.addActionListener(e -> {

            this.dispose();

            LoginFrame.relaunch(orderSystemLogic);

        });

    }

    private void refreshOrderTable()

        SwingUtilities.invokeLater(() -> {

            tableModel.setRowCount(0);

            List<Order> pendingOrders = orderSystemLogic.getPendingOrders();

            for (Order order : pendingOrders) {
```

```
            Vector<Object> rowData = new Vector<>();
            rowData.add(order.getToken());
            rowData.add(order.getItemName());
            rowData.add(order.getQuantity());
            rowData.add(String.format("%.2f", order.getPrice()));
            rowData.add(String.format("%.2f", order.getTotalPrice()));
            rowData.add(order.isComplete() ? "Completed" : "Pending");
            rowData.add(order.getFormattedOrderTime());
            tableModel.addRow(rowData);
        }
    });
}
@Override
public void orderAdded(Order order) {

    if (!order.isComplete()) {
        SwingUtilities.invokeLater(() -> {
            Vector<Object> rowData = new Vector<>();
            rowData.add(order.getToken());
            rowData.add(order.getItemName());
            rowData.add(order.getQuantity());
            rowData.add(String.format("%.2f", order.getPrice()));
            rowData.add(String.format("%.2f", order.getTotalPrice()));
            rowData.add("Pending");
            rowData.add(order.getFormattedOrderTime());
            tableModel.addRow(rowData);
        });
        }
    }

    @Override
    public void orderStatusChanged(Order order) {
        refreshOrderTable();
    }
}
```

9.LoginFrame.java

```
package com.canteen.ui;


import com.canteen.service.AuthService;
import com.canteen.service.OrderSystemLogic;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
public class LoginFrame extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private AuthService authService;
    private OrderSystemLogic orderSystemLogic;
    public LoginFrame(OrderSystemLogic logic) {
        this.authService = new AuthService();
        this.orderSystemLogic = logic;
        setTitle("SwiftServe Login");

        setSize(350, 200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setLocationRelativeTo(null);

        setLayout(new BorderLayout(10, 10));


        initComponents();

        layoutComponents();

        addEventHandlers();

    }

    private void initComponents() {

        usernameField = new JTextField(15);

        passwordField = new JPasswordField(15);

        loginButton = new JButton("Login");

    }


    private void layoutComponents() {

        JPanel formPanel = new JPanel(new GridLayout(3, 2, 5, 5));

        formPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 10, 20));


        formPanel.add(new JLabel("Username (student/admin):"));

        formPanel.add(usernameField);


        formPanel.add(new JLabel("Password (pass/canteen123):"));

        formPanel.add(passwordField);


        formPanel.add(new JLabel()); // Placeholder
        formPanel.add(loginButton);


        add(formPanel, BorderLayout.CENTER);

    }


    private void addEventHandlers() {

        ActionListener loginAction = e -> attemptLogin();

        loginButton.addActionListener(loginAction);

        passwordField.addActionListener(loginAction);

    }


    private void attemptLogin() {

        String username = usernameField.getText();

        String password = new String(passwordField.getPassword());

        AuthService.UserRole role = authService.authenticate(username, password);


        if (role == AuthService.UserRole.INVALID) {

            JOptionPane.showMessageDialog(this, "Invalid Username or Password.", "Authentication Failed",
JOptionPane.ERROR_MESSAGE);

            return;

        }

        this.dispose();

        if (role == AuthService.UserRole.STUDENT) {

            UI customerUI = new UI(orderSystemLogic);

            customerUI.setVisible(true);

        } else if (role == AuthService.UserRole.CANTEEN_ADMIN) {

            AdminUI adminUI = new AdminUI(orderSystemLogic);
            adminUI.setVisible(true);
        }
    }
    public static void relaunch(OrderSystemLogic logic) {
        SwingUtilities.invokeLater(() -> {
            LoginFrame newLogin = new LoginFrame(logic);
            newLogin.setVisible(true);
        });
    }
}
```

10.PaymentDialog.java

```
package com.canteen.ui;


import com.canteen.model.Order;
import javax.swing.*;
import java.awt.*;
import java.time.LocalDateTime;
import java.awt.event.ActionListener;
public class PaymentDialog extends JDialog {
    private Order order;
    private JTextField transactionIdField;
    private JButton submitButton;
    private boolean paymentSuccessful = false;


    public PaymentDialog(JFrame parent, Order order) {
        super(parent, "Secure Payment Gateway", true);
        this.order = order;


        setSize(400, 200);
        setLocationRelativeTo(parent);
        setLayout(new BorderLayout(10, 10));


        initComponents();
        layoutComponents();
        addEventHandlers();
        ((JLabel) ((JPanel) getContentPane().getComponent(0)).getComponent(1)).setText(String.format("$%.2f",
order.getTotalPrice()));
    }


    private void initComponents() {
        transactionIdField = new JTextField(15);
        submitButton = new JButton("Submit Payment");
    }


    private void layoutComponents() {
        JPanel infoPanel = new JPanel(new GridLayout(2, 2, 5, 5));
        infoPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 5, 15));


        infoPanel.add(new JLabel("Order Total:"));
        JLabel totalLabel = new JLabel();
        totalLabel.setFont(new Font("SansSerif", Font.BOLD, 14));
        infoPanel.add(totalLabel);
        infoPanel.add(new JLabel("Transaction ID (e.g., wallet, card):"));
        infoPanel.add(transactionIdField);

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(submitButton);

        add(infoPanel, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);
    }

    private void addEventHandlers() {
        ActionListener processAction = e -> processPayment();
        submitButton.addActionListener(processAction);
        transactionIdField.addActionListener(processAction);
    }

    private void processPayment() {
        String transactionId = transactionIdField.getText().trim();

        if (transactionId.isEmpty() || transactionId.length() < 5) {
            JOptionPane.showMessageDialog(this, "Please enter a valid Transaction ID.", "Input Error", 
JOptionPane.ERROR_MESSAGE);
            return;
        }
        paymentSuccessful = true;
        generateReceipt(transactionId);
        dispose();
```

```
}

private void generateReceipt(String transactionId) {
  JDialog receipt = new JDialog(this, "Order Receipt - Token # Pending...", false);
  receipt.setSize(350, 270);
  receipt.setLocationRelativeTo(this);

  JTextArea textArea = new JTextArea();
  textArea.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 12));
  textArea.setEditable(false);
  String receiptText = String.format(
    "==================================\n" +
    "     SWIFTSERVE RECEIPT        \n" +
    "==================================\n" +
    "TOKEN NO:     [Assigned on Save]\n" +
    "ITEM:        %s\n" +
    "QTY:         %d\n" +
    "UNIT PRICE:   $%.2f\n" +
    "----------------------------------\n" +
    "TOTAL PAID:   $%.2f\n" +
    "TRANS. ID:    %s\n" +
    "TIME:        %s\n" +
    "==================================\n" +
    "    THANK YOU FOR YOUR ORDER!    \n",
    order.getItemName(),
    order.getQuantity(),
    order.getPrice(),
            order.getTotalPrice(),
            transactionId,
            LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPattern("HH:mm:ss"))
        );
        textArea.setText(receiptText);
        receipt.add(new JScrollPane(textArea), BorderLayout.CENTER);
        receipt.setVisible(true);
    }
    public boolean isPaymentSuccessful() {
        return paymentSuccessful;
    }
}
```

11.UI.java

```
package com.canteen.ui;


import com.canteen.model.Order;
import com.canteen.service.OrderSystemLogic;
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.ActionListener;


public class UI extends JFrame implements OrderSystemLogic.OrderUpdateListener {
    private OrderSystemLogic orderSystemLogic;
    private AdminTokenSlip adminTokenSlip;


    private JTextField itemNameField;
private JSpinner quantitySpinner;

private JTextField priceField;

private JButton placeOrderButton;

private JButton logoutButton;

private JLabel confirmationLabel;


public UI(OrderSystemLogic orderSystemLogic) {

   this.orderSystemLogic = orderSystemLogic;

   this.orderSystemLogic.addOrderUpdateListener(this);

   this.adminTokenSlip = new AdminTokenSlip();


   setTitle("Canteen Order System - Customer");

   setSize(400, 350);

   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   setLocationRelativeTo(null);

   setLayout(new BorderLayout(10, 10));

   ((JPanel) getContentPane()).setBorder(new EmptyBorder(10, 10, 10, 10));


   initComponents();

   layoutComponents();

   addEventHandlers();

}


private void initComponents() {

   itemNameField = new JTextField(20);

   quantitySpinner = new JSpinner(new SpinnerNumberModel(1, 1, 100, 1));

   priceField = new JTextField(10);
```

```
        placeOrderButton = new JButton("Place Order");

        logoutButton = new JButton("Logout");

        confirmationLabel = new JLabel("Enter your order details.");

        confirmationLabel.setHorizontalAlignment(SwingConstants.CENTER);

        confirmationLabel.setFont(new Font("SansSerif", Font.ITALIC, 12));

    }


    private void layoutComponents() {

        JPanel formPanel = new JPanel(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();

        gbc.insets = new Insets(5, 5, 5, 5);

        gbc.fill = GridBagConstraints.HORIZONTAL;


        gbc.gridx = 0; gbc.gridy = 0; formPanel.add(new JLabel("Item Name:"), gbc);

        gbc.gridx = 1; gbc.gridy = 0; formPanel.add(itemNameField, gbc);


        gbc.gridx = 0; gbc.gridy = 1; formPanel.add(new JLabel("Quantity:"), gbc);

        gbc.gridx = 1; gbc.gridy = 1; formPanel.add(quantitySpinner, gbc);


        gbc.gridx = 0; gbc.gridy = 2; formPanel.add(new JLabel("Price ($):"), gbc);

        gbc.gridx = 1; gbc.gridy = 2; formPanel.add(priceField, gbc);


        add(formPanel, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 5));

        buttonPanel.add(placeOrderButton);

        buttonPanel.add(logoutButton);

        add(buttonPanel, BorderLayout.SOUTH);
        add(confirmationLabel, BorderLayout.NORTH);

    }

    private void addEventHandlers() {

        placeOrderButton.addActionListener(e -> placeOrder());

        logoutButton.addActionListener(e -> {

            this.dispose();

            LoginFrame.relaunch(orderSystemLogic);

        });

    }


    public void placeOrder() {

        String itemName = itemNameField.getText().trim();

        int quantity = (Integer) quantitySpinner.getValue();

        float price;

        if (itemName.isEmpty()) {

            JOptionPane.showMessageDialog(this, "Item Name cannot be empty.", "Input Error",
JOptionPane.ERROR_MESSAGE);

            return;

        }

        try {

            price = Float.parseFloat(priceField.getText().trim());

            if (price < 0) {

                JOptionPane.showMessageDialog(this, "Price cannot be negative.", "Input Error",
JOptionPane.ERROR_MESSAGE);

                return;

            }

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Invalid price format. Please enter a number.", "Input Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }
        Order newOrder = new Order(itemName, quantity, price);
        PaymentDialog paymentDialog = new PaymentDialog(this, newOrder);
        paymentDialog.setVisible(true);
        if (paymentDialog.isPaymentSuccessful()) {
            orderSystemLogic.addOrder(newOrder);


            itemNameField.setText("");
            quantitySpinner.setValue(1);
            priceField.setText("");


            confirmationLabel.setText("Order placed - Token #" + newOrder.getToken());


            SwingUtilities.invokeLater(() -> adminTokenSlip.displayTokenSlip(newOrder));
        } else {
            confirmationLabel.setText("Order aborted. Payment failed or cancelled.");
        }
    }
    @Override
    public void orderAdded(Order order) { /* ... */ }
    @Override
    public void orderStatusChanged(Order order) { /* ... */ }
}
```

6. User Manual

A) Student Workflow:

- Open the Order App.
- Browse available menu items, add selections to cart, specify quantity.
- Confirm order and proceed to payment via integrated campus wallet/UPI.
- Immediately receive a token on-screen as an order confirmation and receipt.
- Wait for notification that food is ready, then collect food by showing the token.

B) Admin Workflow:

- Log into Admin App on dedicated staff device.
- View all incoming orders in live dashboard with status, items, and tokens.
- Prepare food, mark order as “Complete” once ready.
- Student and staff notified instantly of status change.

Admin Features

- Access admin panel to add/remove/edit menu items.
- View full order and transaction history.
- Prepare for planned stock management functionality.

7. Test Cases

1. **Valid Order Flow:** Place order, receive token, staff sees order, marks it complete, student notified.
2. **Order Validation:** Place order with missing item/quantity; error displayed, no order saved.
3. **Data Loss Protection:** Simulate app/network failure during order placement; order/transaction data remains intact when app restarts.
4. **Concurrency:** Multiple students submit orders; kitchen dashboard updates instantly in real time.
5. **Payment Verification:** Successful digital payment triggers token issuance and saves transaction.
6. **Menu Update:** Admin updates a menu item; change instantly visible to all users.
7. **Staff Actions:** Staff can mark multiple orders as complete simultaneously; notifications sent as expected.

8. Individual Contributions

Member Name	Area of Contribution	Key Responsibilities
Karthika Bensilal **(24CT042)**	Database Design & Implementation	Designed database schema, wrote SQL scripts, handled data migration and integration
Alwin M Benny **(24CT015)**	Backend Logic & Integration	Developed business/service layer, implemented DAO/repository logic, integrated backend to UI
Karthika Unni **(24CT043)**	Frontend/UI Development	Created login, student, and admin interfaces, handled event-driven programming and usability enhancements
Deborah Elsa Halu **(24CT024)**	Testing, Documentation & UX	Wrote user manuals, performed software and code testing, managed feedback cycle, improved UI/UX consistency

9. Conclusion

SwiftServe provides a robust, user-friendly solution to the inefficiencies of campus canteen management. By digitizing menu browsing, order placement, payment, and collection, it eliminates queues, ensures instant feedback for all users, and supports reliable, data-secure operations. Its modular system design and separation of core layers ensure lasting maintainability and readiness for future features like advanced inventory, analytics, and expanded admin controls.

10. Future Scope

SwiftServe offers a solid platform for digital canteen management and can be expanded in several ways:

- Multi-Branch Support: Manage several canteens or outlets from a single system.
- Mobile Integration: Develop mobile apps for easier student and staff access on the go.
- Advanced Payment Methods: Integrate popular payment gateways and campus wallets.
- Inventory & Analytics Features: Automate stock monitoring and provide reports on sales, menu performance, and peak usage times.
- Feedback & Notifications: Add features for student feedback, complaint management, and real-time notifications about orders or offers.
- Campus System Integration: Connect SwiftServe with student portals or hostel management for seamless service.

These upgrades will help SwiftServe support larger campuses, offer more convenience, and improve dining experiences for everyone.