FUNCITY MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

Monish D.Y. **230701195**

Kasilingam M. **230701145**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM, CHENNAI-602105

2024 – 2025

BONAFIDE CERTIFICATE

Certified that this project report **“FUNCITY MANAGEMENT SYSTEM”** is the bonafide work of **“KASILINGAM M. (230701145) and MONISH D.Y. (230701195)”** who carried out the project work under my supervision.

Submitted for the Practical Examination held on 26.11.2024

SIGNATURE

Mrs. K. MAHESMEENA,

Assistant Professor,

Computer Science and Engineering,

Rajalakshmi Engineering College (Autonomous), Thandalam, Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The FunCity Management System is a Java-based mini-project designed to simplify and enhance the operations of a mall's gaming section. Its main goal is to create an efficient system for managing customer interactions, ticket bookings, game reservations, and rewards, all while improving the overall visitor experience.

The system is divided into several key modules: Customer Management, Machine Management, Ticket Management, Points Management, and Game Management. Each module is tailored to handle specific tasks such as registering customers, dispensing tickets, managing gaming machines, tracking loyalty points, and overseeing game availability. The data is stored in a well-structured MySQL database, ensuring that all information related to customers, machines, tickets, and points is easily accessible and well-organized.

The database schema is carefully designed, using tables like Customer, Machine, Ticket, Points, and Game, all normalized to ensure minimal redundancy and maximum efficiency. Relationships between these tables are established through foreign keys, ensuring seamless integration of customer records, tickets, and reward points.

Java's object-oriented capabilities, such as collections, exception handling, and multi-threading, are leveraged to make the system responsive and scalable. The user interface is intuitive, allowing administrators to efficiently manage operations, monitor ticket sales, and track machine/game availability.

Ultimately, the FunCity Management System automates routine tasks, reduces human error, and provides valuable insights for administrators, creating a more enjoyable and streamlined gaming experience for visitors. The system is designed to be scalable, with future enhancements like mobile app support and real-time game tracking. Through this project, the practical application of Java and relational database management in solving real-world challenges is demonstrated, offering a modern solution for managing gaming sections in malls.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Introduction

The FunCity Management System is a Java-based project aimed at efficiently managing the gaming section of a mall. It streamlines operations like customer registration, ticketing, game reservations, and loyalty point tracking. With its intuitive user interface and robust MySQL database, the system enhances the gaming experience for visitors and simplifies administrative tasks for staff.

1.2. Objective

The objective of the FunCity Management System is to automate the management of a mall's gaming section, minimize manual errors, improve operational efficiency, and provide a seamless and enjoyable experience for both visitors and administrators.

1.3. Modules

1. Customer Management: Manages customer registration, profiles, and interaction history.

2. Machine Management: Tracks the status and availability of gaming machines.

3. Ticket Management: Handles ticket booking, issuance, and tracking for games.

4. Points Management: Tracks and updates customer loyalty points and rewards.

5. Game Management: Manages game reservations, schedules, and availability in real time.

2. SYSTEM ARCHITECTURE

2.1. Frontend Layer (Java Swing):

- Use Java Swing components like frames, panels, tables, and buttons for creating forms and visualizing data.
- Include navigation and interactive features through event handling.
- Ensure input validation and support background tasks for smooth UI operations.

2.2. Backend Layer (MySQL Database):

- Design a normalized database schema with foreign key constraints for integrity.
- Optimize performance using indexes and stored procedures for complex operations.
- Manage the database using MySQL tools or scripts.

2.3. Integration Layer (Database Connection):

- Implement a database connector module using JDBC for communication with MySQL.
- Ensure secure data handling by preventing SQL injection and managing resources properly.
- Abstract database operations for reusable and maintainable code.

3. REQUIREMENTS AND ANALYSIS

3.1. Requirement Specification

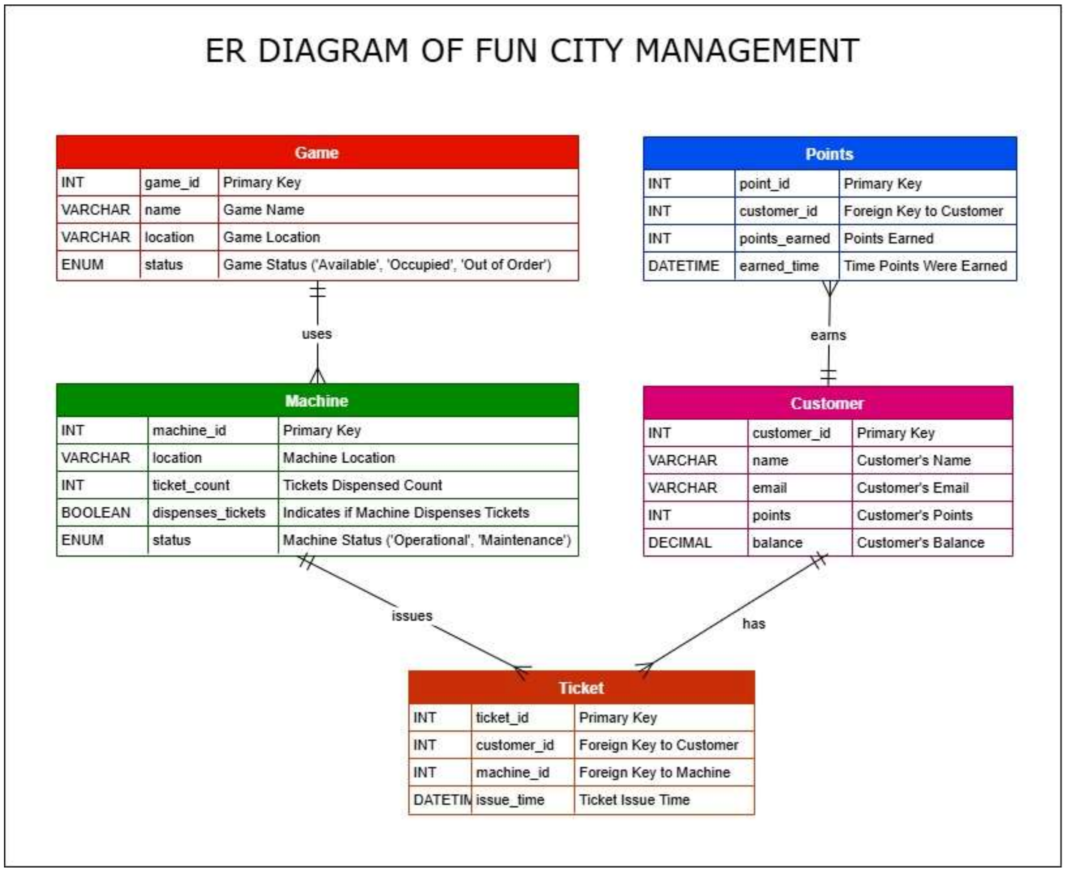
The FunCity Management System manages customer interactions, ticket bookings, games, machines, and loyalty points efficiently.

Key **features** include modules for customer registration, ticketing, game reservations, machine tracking, and points management, with admin tools for reports and role management.

Non-functional needs focus on usability, scalability, reliability, security, and performance, ensuring seamless and secure operations.

Built with Java and MySQL, it requires minimal hardware and supports future expansions, automating tasks and enhancing user experience.

3.2. ER Diagram



Game		
INT	game_id	Primary Key
VARCHAR	name	Game Name
VARCHAR	location	Game Location
ENUM	status	Game Status ('Available', 'Occupied', 'Out of Order')

Points		
INT	point_id	Primary Key
INT	customer_id	Foreign Key to Customer
INT	points_earned	Points Earned
DATETIME	earned_time	Time Points Were Earned

Machine		
INT	machine_id	Primary Key
VARCHAR	location	Machine Location
INT	ticket_count	Tickets Dispensed Count
BOOLEAN	dispenses_tickets	Indicates if Machine Dispenses Tickets
ENUM	status	Machine Status ('Operational', 'Maintenance')

Customer		
INT	customer_id	Primary Key
VARCHAR	name	Customer's Name
VARCHAR	email	Customer's Email
INT	points	Customer's Points
DECIMAL	balance	Customer's Balance

Ticket		
INT	ticket_id	Primary Key
INT	customer_id	Foreign Key to Customer
INT	machine_id	Foreign Key to Machine
DATETIM	issue_time	Ticket Issue Time

This **ER Diagram** represents the interconnection of your database tables.

Entities and Attributes

1. **Customer**: Represents the individuals interacting with the system.
   - Attributes:
     - customer_id (Primary Key): Unique identifier for each customer.
     - name: Name of the customer.
     - email: Contact email address of the customer.
     - points: Points accumulated by the customer through transactions.
     - balance: Current balance in the customer's account.

2. **Machine**: Represents ticket-dispensing machines in the amusement park.
   - Attributes:
     - machine_id (Primary Key): Unique identifier for each machine.
     - location: Physical location of the machine.
     - ticket_count: Number of tickets dispensed by the machine.
     - dispenses_tickets: Boolean indicating if the machine dispenses tickets.
     - status: Operational status of the machine (Operational or Maintenance).

3. **Ticket**: Tracks ticket issuance to customers.
   - Attributes:
     - ticket_id (Primary Key): Unique identifier for each ticket.
     - customer_id (Foreign Key): Links to the Customer table.
     - machine_id (Foreign Key): Links to the Machine table.
     - issue_time: Timestamp when the ticket was issued.

4. **Points**: Represents points earned by customers.
   - Attributes:
     - point_id (Primary Key): Unique identifier for the points record.
     - customer_id (Foreign Key): Links to the Customer table.
     - points_earned: Number of points earned during a transaction.
     - earned_time: Timestamp for when points were earned.

5. **Game**: Represents various games available in the amusement park.
   - Attributes:
     - game_id (Primary Key): Unique identifier for each game.
     - name: Name of the game.
     - location: Location of the game within the amusement park.
     - status: Status of the game (Available, Occupied, Out of Order).

Relationships

- **Customer-Ticket**: One customer can be associated with multiple tickets. The foreign key customer_id in the Ticket table creates this relationship.
- **Customer-Points**: One customer can earn multiple points records, linked through the foreign key customer_id in the Points table.
- **Machine-Ticket**: A machine can issue multiple tickets. The foreign key machine_id in the Ticket table defines this.
- **Game-Machine**: Games may be associated with ticket-dispensing machines. This connection can be modeled through logical association if needed.

3.3. Normalization

Step 1: First Normal Form (1NF)

1NF ensures all tables contain atomic (indivisible) values, and there are no repeating groups:

- Each column contains unique data types.
- Example: In Customer, name holds a single value like "John Doe" rather than multiple values.

Step 2: Second Normal Form (2NF)

2NF eliminates partial dependencies, ensuring all non-key attributes are fully dependent on the primary key:

- Each table has a primary key, and no attribute depends only on part of a composite key.
- Example:
    - In Ticket, customer_id and machine_id depend fully on ticket_id. No partial dependency exists.

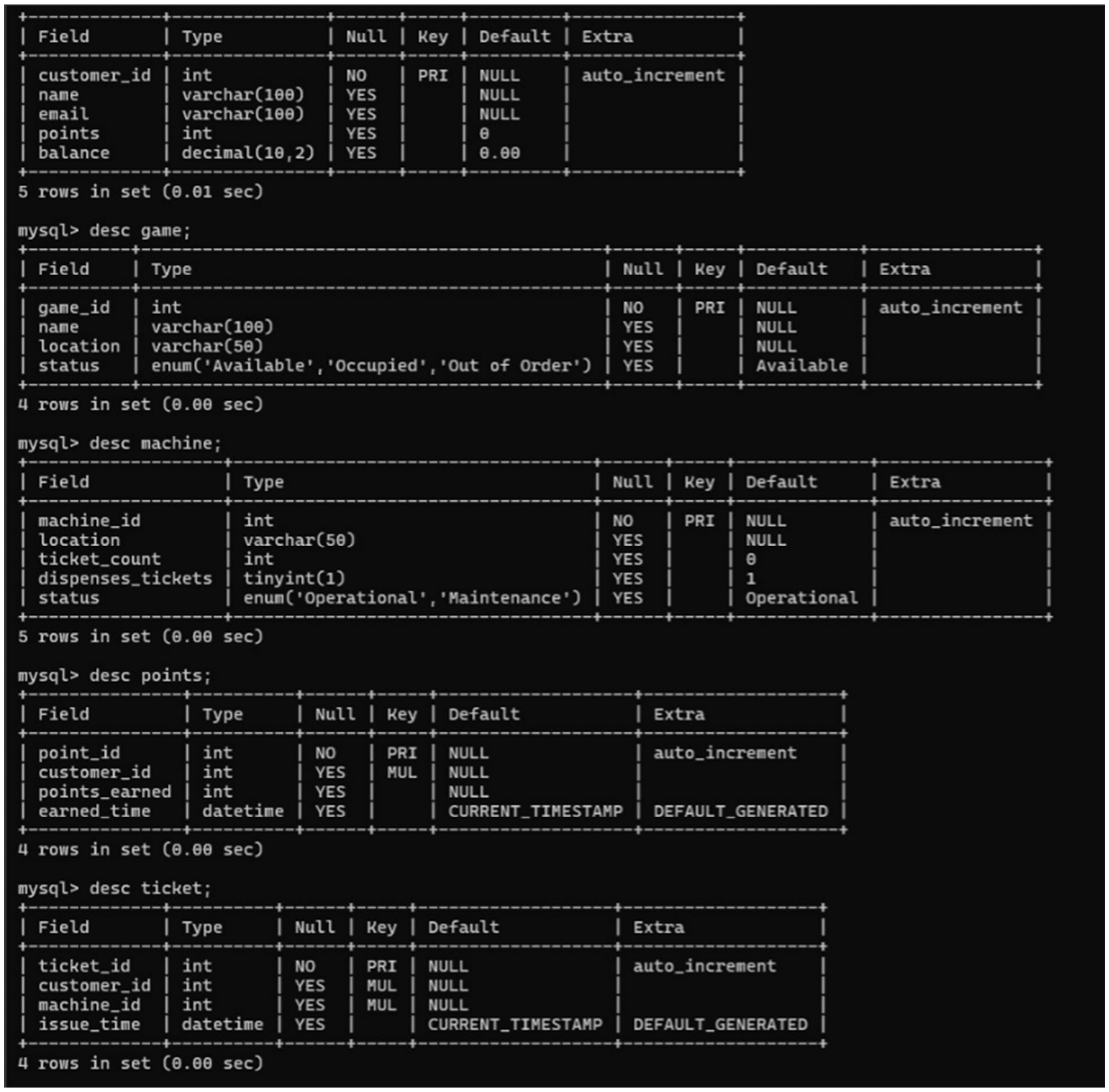
Step 3: Third Normal Form (3NF)

3NF eliminates transitive dependencies, ensuring all attributes depend only on the primary key:

- There are no attributes dependent on other non-primary attributes.
- Example:
    - In Machine, location depends only on machine_id, not on other attributes like status.

    - depending only on game_id.

4. DATABASE SCHEMA



```
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| customer_id | int           | NO   | PRI | NULL    | auto_increment |
| name        | varchar(100)  | YES  |     | NULL    |                |
| email       | varchar(100)  | YES  |     | NULL    |                |
| points      | int           | YES  |     | 0       |                |
| balance     | decimal(10,2) | YES  |     | 0.00    |                |
+-------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)

mysql> desc game;
+----------+---------------------------------------------+------+-----+-----------+----------------+
| Field    | Type                                        | Null | Key | Default   | Extra          |
+----------+---------------------------------------------+------+-----+-----------+----------------+
| game_id  | int                                         | NO   | PRI | NULL      | auto_increment |
| name     | varchar(100)                                | YES  |     | NULL      |                |
| location | varchar(50)                                 | YES  |     | NULL      |                |
| status   | enum('Available','Occupied','Out of Order') | YES  |     | Available |                |
+----------+---------------------------------------------+------+-----+-----------+----------------+
4 rows in set (0.00 sec)

mysql> desc machine;
+-------------------+----------------------------------+------+-----+-------------+----------------+
| Field             | Type                             | Null | Key | Default     | Extra          |
+-------------------+----------------------------------+------+-----+-------------+----------------+
| machine_id        | int                              | NO   | PRI | NULL        | auto_increment |
| location          | varchar(50)                      | YES  |     | NULL        |                |
| ticket_count      | int                              | YES  |     | 0           |                |
| dispenses_tickets | tinyint(1)                       | YES  |     | 1           |                |
| status            | enum('Operational','Maintenance') | YES  |     | Operational |                |
+-------------------+----------------------------------+------+-----+-------------+----------------+
5 rows in set (0.00 sec)

mysql> desc points;
+---------------+----------+------+-----+-------------------+-------------------+
| Field         | Type     | Null | Key | Default           | Extra             |
+---------------+----------+------+-----+-------------------+-------------------+
| point_id      | int      | NO   | PRI | NULL              | auto_increment    |
| customer_id   | int      | YES  | MUL | NULL              |                   |
| points_earned | int      | YES  |     | NULL              |                   |
| earned_time   | datetime | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+---------------+----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)

mysql> desc ticket;
+-------------+----------+------+-----+-------------------+-------------------+
| Field       | Type     | Null | Key | Default           | Extra             |
+-------------+----------+------+-----+-------------------+-------------------+
| ticket_id   | int      | NO   | PRI | NULL              | auto_increment    |
| customer_id | int      | YES  | MUL | NULL              |                   |
| machine_id  | int      | YES  | MUL | NULL              |                   |
| issue_time  | datetime | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)
```

5. PROGRAM CODE

1. SQL QUERIES

We got 5 tables in this miniproject. These are the queries for those tables

```
CREATE TABLE IF NOT EXISTS Customer (
   customer_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   email VARCHAR(100),
   points INT DEFAULT 0,
   balance DECIMAL(10, 2) DEFAULT 0.00
);


CREATE TABLE IF NOT EXISTS Machine (
   machine_id INT PRIMARY KEY AUTO_INCREMENT,
   location VARCHAR(50),
   ticket_count INT DEFAULT 0,
   dispenses_tickets BOOLEAN DEFAULT TRUE,
   status ENUM('Operational', 'Maintenance') DEFAULT 'Operational'
);


CREATE TABLE IF NOT EXISTS Ticket (
   ticket_id INT PRIMARY KEY AUTO_INCREMENT,
   customer_id INT,
   machine_id INT,
   issue_time DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
   FOREIGN KEY (machine_id) REFERENCES Machine(machine_id)
);


CREATE TABLE IF NOT EXISTS Points (
   point_id INT PRIMARY KEY AUTO_INCREMENT,
   customer_id INT,
   points_earned INT,
   earned_time DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);


CREATE TABLE IF NOT EXISTS Game (
   game_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   location VARCHAR(50),
   status ENUM('Available', 'Occupied', 'Out of Order') DEFAULT 'Available'
);
```

2. JAVA CODE

i. Main.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.table.DefaultTableModel;

public class Main extends JFrame {
  private JTabbedPane tabbedPane;

  public Main() {
    setTitle("Fun City Management System");
    setSize(600, 500);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);

    tabbedPane = new JTabbedPane();

    // Add each management panel to the tabbedPane
    tabbedPane.add("Customer Management", new CustomerPanel());
    tabbedPane.add("Machine Management", new MachinePanel());
    tabbedPane.add("Ticket Management", new TicketPanel());
    tabbedPane.add("Points Management", new PointsPanel());
    tabbedPane.add("Game Management", new GamePanel());

    add(tabbedPane);
  }
```

```
    public static void main(String[] args) {
        // Show the login page first
        SwingUtilities.invokeLater(() -> new LoginPage().setVisible(true));
    }
}

class LoginPage extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private JLabel messageLabel;

    public LoginPage() {
        // Set JFrame properties
        setTitle("Login Page");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);  // Center the window on the screen

        // Use GridBagLayout for better component arrangement
        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5); // Add some space between components

        // Create components
        JLabel usernameLabel = new JLabel("Username:");
        usernameField = new JTextField(20);

        JLabel passwordLabel = new JLabel("Password:");
        passwordField = new JPasswordField(20);

        loginButton = new JButton("Login");
```

```
messageLabel = new JLabel("");

// Arrange components using GridBagLayout
gbc.gridx = 0; gbc.gridy = 0; gbc.anchor = GridBagConstraints.EAST;
add(usernameLabel, gbc);

gbc.gridx = 1; gbc.gridy = 0; gbc.anchor = GridBagConstraints.WEST;
add(usernameField, gbc);

gbc.gridx = 0; gbc.gridy = 1; gbc.anchor = GridBagConstraints.EAST;
add(passwordLabel, gbc);

gbc.gridx = 1; gbc.gridy = 1; gbc.anchor = GridBagConstraints.WEST;
add(passwordField, gbc);

gbc.gridx = 1; gbc.gridy = 2; gbc.anchor = GridBagConstraints.CENTER;
add(loginButton, gbc);

gbc.gridx = 1; gbc.gridy = 3; gbc.anchor = GridBagConstraints.CENTER;
add(messageLabel, gbc);

// Add event listener to loginButton
loginButton.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    String username = usernameField.getText();
    String password = new String(passwordField.getPassword());

    // Call the validateLogin method from LoginManagement
    boolean loginSuccess = LoginManagement.validateLogin(username, password);
```

```
                if (loginSuccess) {
                    messageLabel.setText("Login Successful");
                    messageLabel.setForeground(Color.GREEN);
                    // Open the main application window after successful login
                    new Main().setVisible(true);
                    dispose();  // Close the login window
                } else {
                    messageLabel.setText("Invalid username or password");
                    messageLabel.setForeground(Color.RED);
                }
            }
        });
    }
}

class LoginManagement {
    public static boolean validateLogin(String username, String password) {
        // Add actual login validation logic, for now assuming success
        return "admin".equals(username) && "password".equals(password);  // Dummy check
for testing
    }
}



class MachinePanel extends JPanel {
    private JTextField locationField, ticketCountField;
    private JCheckBox dispensesCheckbox;
    private JComboBox<String> statusComboBox;
    private DefaultTableModel tableModel;
    private JTable machineTable;

    public MachinePanel() {
        setLayout(new BorderLayout(10, 10));


        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));


        // Fields
        JPanel locationPanel = createInputPanel("Location:", locationField = new JTextField());
        JPanel ticketCountPanel = createInputPanel("Ticket Count:", ticketCountField = new
JTextField());
        JPanel dispensesPanel = createInputPanel("Dispenses Tickets:", dispensesCheckbox =
new JCheckBox());
        JPanel statusPanel = createInputPanel("Status:", statusComboBox = new
JComboBox<>(new String[]{"Operational", "Maintenance"}));


        inputPanel.add(locationPanel);
        inputPanel.add(ticketCountPanel);
        inputPanel.add(dispensesPanel);
        inputPanel.add(statusPanel);


        // Buttons
        JPanel buttonPanel = new JPanel();
        JButton addButton = new JButton("Add Machine");
        JButton viewButton = new JButton("View Machines");
        buttonPanel.add(addButton);
        buttonPanel.add(viewButton);


        addButton.addActionListener(new AddMachineAction());
        viewButton.addActionListener(new ViewMachinesAction());


        inputPanel.add(buttonPanel);
        add(inputPanel, BorderLayout.NORTH);
        // Table
        tableModel = new DefaultTableModel(new String[]{"Machine ID", "Location", "Ticket
Count", "Dispenses", "Status"}, 0);
        machineTable = new JTable(tableModel);
        add(new JScrollPane(machineTable), BorderLayout.CENTER);
    }

    private JPanel createInputPanel(String labelText, JComponent field) {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setPreferredSize(new Dimension(100, 20));
        field.setPreferredSize(new Dimension(200, 25));
        panel.add(label);
        panel.add(field);
        return panel;
    }

    private class AddMachineAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String location = locationField.getText();
            int ticketCount = Integer.parseInt(ticketCountField.getText());
            boolean dispensesTickets = dispensesCheckbox.isSelected();
            String status = (String) statusComboBox.getSelectedItem();

            MachineManagement.addMachine(location, ticketCount, dispensesTickets, status);
            JOptionPane.showMessageDialog(MachinePanel.this, "Machine added successfully.",
"Success", JOptionPane.INFORMATION_MESSAGE);

            locationField.setText("");
            ticketCountField.setText("");
            dispensesCheckbox.setSelected(false);
            statusComboBox.setSelectedIndex(0);
        }
    }

    private class ViewMachinesAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            tableModel.setRowCount(0);
            Object[][] machines = MachineManagement.getMachines();
            for (Object[] machine : machines) {
                tableModel.addRow(machine);
            }
        }
    }
}
class TicketPanel extends JPanel {
    private JTextField customerIdField, machineIdField;
    private DefaultTableModel tableModel;
    private JTable ticketTable;

    public TicketPanel() {
        setLayout(new BorderLayout(10, 10));

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        // Fields
        JPanel customerIdPanel = createInputPanel("Customer ID:", customerIdField = new
JTextField());
        JPanel machineIdPanel = createInputPanel("Machine ID:", machineIdField = new
JTextField());
```

```
        inputPanel.add(customerIdPanel);
        inputPanel.add(machineIdPanel);

        // Buttons
        JPanel buttonPanel = new JPanel();
        JButton addButton = new JButton("Add Ticket");
        JButton viewButton = new JButton("View Tickets");
        buttonPanel.add(addButton);
        buttonPanel.add(viewButton);

        addButton.addActionListener(new AddTicketAction());
        viewButton.addActionListener(new ViewTicketsAction());

        inputPanel.add(buttonPanel);
        add(inputPanel, BorderLayout.NORTH);

        // Table
        tableModel = new DefaultTableModel(new String[]{"Ticket ID", "Customer ID",
"Machine ID"}, 0);
        ticketTable = new JTable(tableModel);
        add(new JScrollPane(ticketTable), BorderLayout.CENTER);
    }

    private JPanel createInputPanel(String labelText, JTextField field) {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setPreferredSize(new Dimension(100, 20));
        field.setPreferredSize(new Dimension(200, 25));
        panel.add(label);
        panel.add(field);
        return panel;
    }


    private class AddTicketAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            int customerId = Integer.parseInt(customerIdField.getText());
            int machineId = Integer.parseInt(machineIdField.getText());


            TicketManagement.addTicket(customerId, machineId);
            JOptionPane.showMessageDialog(TicketPanel.this, "Ticket added successfully.",
"Success", JOptionPane.INFORMATION_MESSAGE);


            customerIdField.setText("");
            machineIdField.setText("");
        }
    }


    private class ViewTicketsAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            tableModel.setRowCount(0);
            Object[][] tickets = TicketManagement.getTickets();
            for (Object[] ticket : tickets) {
                tableModel.addRow(ticket);
            }
        }
    }
}
class PointsPanel extends JPanel {
    private JTextField customerIdField, pointsField;
    private DefaultTableModel tableModel;
    private JTable pointsTable;
```

```
  public PointsPanel() {
    setLayout(new BorderLayout(10, 10));

    JPanel inputPanel = new JPanel();
    inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS));
    inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    // Fields
    JPanel customerIdPanel = createInputPanel("Customer ID:", customerIdField = new
JTextField());
    JPanel pointsPanel = createInputPanel("Points Earned:", pointsField = new
JTextField());

    inputPanel.add(customerIdPanel);
    inputPanel.add(pointsPanel);

    // Buttons
    JPanel buttonPanel = new JPanel();
    JButton addButton = new JButton("Add Points");
    JButton viewButton = new JButton("View Points");
    buttonPanel.add(addButton);
    buttonPanel.add(viewButton);

    addButton.addActionListener(new AddPointsAction());
    viewButton.addActionListener(new ViewPointsAction());

    inputPanel.add(buttonPanel);
    add(inputPanel, BorderLayout.NORTH);

    // Table
    tableModel = new DefaultTableModel(new String[]{"Points ID", "Customer ID",
"Points"}, 0);
```

```
        pointsTable = new JTable(tableModel);
        add(new JScrollPane(pointsTable), BorderLayout.CENTER);
    }

    private JPanel createInputPanel(String labelText, JTextField field) {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setPreferredSize(new Dimension(100, 20));
        field.setPreferredSize(new Dimension(200, 25));
        panel.add(label);
        panel.add(field);
        return panel;
    }

    private class AddPointsAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            int customerId = Integer.parseInt(customerIdField.getText());
            int pointsEarned = Integer.parseInt(pointsField.getText());

            PointsManagement.addPoints(customerId, pointsEarned);
            JOptionPane.showMessageDialog(PointsPanel.this, "Points added successfully.",
"Success", JOptionPane.INFORMATION_MESSAGE);

            customerIdField.setText("");
            pointsField.setText("");
        }
    }

    private class ViewPointsAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            tableModel.setRowCount(0);
            Object[][] pointsData = PointsManagement.getPoints();
            for (Object[] points : pointsData) {
                tableModel.addRow(points);
            }
        }
    }
}

class GamePanel extends JPanel {
    private JTextField nameField, locationField, statusField;
    private DefaultTableModel tableModel;
    private JTable gameTable;

    public GamePanel() {
        setLayout(new BorderLayout(10, 10));  // Adding padding between components

        // Create input fields panel with BoxLayout for vertical alignment
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));  // Adding
padding around the form

        // Name, Location, Status fields
        JPanel namePanel = createInputPanel("Name:", nameField = new JTextField());
        JPanel locationPanel = createInputPanel("Location:", locationField = new JTextField());
        JPanel statusPanel = createInputPanel("Status:", statusField = new JTextField());

        inputPanel.add(namePanel);
        inputPanel.add(locationPanel);
        inputPanel.add(statusPanel);
        // Add Game and View Games buttons
        JPanel buttonPanel = new JPanel();
        JButton addButton = new JButton("Add Game");
        JButton viewButton = new JButton("View Games");
        buttonPanel.add(addButton);
        buttonPanel.add(viewButton);

        // Register button actions
        addButton.addActionListener(new AddGameAction());
        viewButton.addActionListener(new ViewGamesAction());

        inputPanel.add(buttonPanel);

        // Add the input panel to the top
        add(inputPanel, BorderLayout.NORTH);

        // Create table to display games
        tableModel = new DefaultTableModel(new String[]{"Game ID", "Name", "Location",
"Status"}, 0);
        gameTable = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(gameTable);
        add(scrollPane, BorderLayout.CENTER);
    }

    // Helper method to create individual input field panels
    private JPanel createInputPanel(String labelText, JTextField textField) {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setPreferredSize(new Dimension(80, 20));
        textField.setPreferredSize(new Dimension(200, 25));
        panel.add(label);
        panel.add(textField);
        return panel;
    }

    // Action to add a game
    private class AddGameAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String name = nameField.getText().trim();
            String location = locationField.getText().trim();
            String status = statusField.getText().trim();

            if (name.isEmpty() || location.isEmpty() || status.isEmpty()) {
                JOptionPane.showMessageDialog(GamePanel.this, "Please fill in all fields.", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Call the method to add the game to the database
            GameManagement.addGame(name, location, status);
            JOptionPane.showMessageDialog(GamePanel.this, "Game added successfully.",
"Success", JOptionPane.INFORMATION_MESSAGE);

            // Clear fields
            nameField.setText("");
            locationField.setText("");
            statusField.setText("");
        }
    }

    // Action to view games
    private class ViewGamesAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Clear existing rows
            tableModel.setRowCount(0);
            // Retrieve games from the database
            Object[][] games = GameManagement.getGames();
            for (Object[] game : games) {
                tableModel.addRow(game);
            }
        }
    }
}
class CustomerPanel extends JPanel {
    private JTextField nameField;
    private JTextField emailField;
    private JTextField pointsField; // Renamed for clarity
    private JTextField balanceField; // Renamed for clarity
    private JTable customerTable;
    private DefaultTableModel tableModel;

    public CustomerPanel() {
        setLayout(new BorderLayout(10, 10));  // Adding some padding between components

        // Create input fields panel using BoxLayout for cleaner alignment
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS)); // Vertical
stacking of components
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); // Adding
some padding around the form

        // Name, Email, Points, Balance fields
        JPanel namePanel = createInputPanel("Name:", nameField = new JTextField());
        JPanel emailPanel = createInputPanel("Email:", emailField = new JTextField());
        JPanel pointsPanel = createInputPanel("Points:", pointsField = new JTextField());
        JPanel balancePanel = createInputPanel("Balance:", balanceField = new JTextField());

        inputPanel.add(namePanel);
        inputPanel.add(emailPanel);
        inputPanel.add(pointsPanel);
        inputPanel.add(balancePanel);

        // Add Customer and View Customers buttons
        JPanel buttonPanel = new JPanel();
        JButton addButton = new JButton("Add Customer");
        JButton viewButton = new JButton("View Customers");
        buttonPanel.add(addButton);
        buttonPanel.add(viewButton);

        // Register button actions
        addButton.addActionListener(new AddCustomerAction());
        viewButton.addActionListener(new ViewCustomersAction());

        inputPanel.add(buttonPanel);

        // Add the input panel to the top
        add(inputPanel, BorderLayout.NORTH);

        // Create table to display customers
        tableModel = new DefaultTableModel(new String[]{"Customer ID", "Name", "Email",
"Points", "Balance"}, 0);
        customerTable = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(customerTable);
        add(scrollPane, BorderLayout.CENTER);
    }
    // Helper method to create individual input field panels
    private JPanel createInputPanel(String labelText, JTextField textField) {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        JLabel label = new JLabel(labelText);
        label.setPreferredSize(new Dimension(80, 20));
        textField.setPreferredSize(new Dimension(200, 25));
        panel.add(label);
        panel.add(textField);
        return panel;
    }

    // Action to add customer
    private class AddCustomerAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String name = nameField.getText().trim();
            String email = emailField.getText().trim();
            String pointsStr = pointsField.getText().trim(); // Retrieve points
            String balanceStr = balanceField.getText().trim(); // Retrieve balance

            if (name.isEmpty() || email.isEmpty() || pointsStr.isEmpty() || balanceStr.isEmpty()) {
                JOptionPane.showMessageDialog(CustomerPanel.this, "Please fill in all fields.",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            try {
                int points = Integer.parseInt(pointsStr); // Convert points to int
                double balance = Double.parseDouble(balanceStr); // Convert bal to double

                // Call the method to add the customer to the database
                CustomerManagement.addCustomer(name, email, points, balance);
                JOptionPane.showMessageDialog(CustomerPanel.this, "Customer added successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                nameField.setText("");
                emailField.setText("");
                pointsField.setText("");
                balanceField.setText("");
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(CustomerPanel.this, "Please enter valid numeric values for points and balance.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
    // Action to view customers
    private class ViewCustomersAction implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Clear existing rows
            tableModel.setRowCount(0);
            // Retrieve customers from the database
            Object[][] customers = CustomerManagement.getCustomers();
            for (Object[] customer : customers) {
                tableModel.addRow(customer);
            }
        }
    }
}
```

ii. Database.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class Database{
   private static final String URL = "jdbc:mysql://localhost:3306/FunCityManagement";
   private static final String USER = "root";
   private static final String PASSWORD = "Changeme@315";

   public static Connection getConnection() {
      Connection conn = null;
      try {
         conn = DriverManager.getConnection(URL, USER, PASSWORD);
         System.out.println("Connected to database!");
      } catch (SQLException e) {
         System.out.println("Error connecting to database: " + e.getMessage());
      }
      return conn;
   }
}
```

iii. CustomerManagement.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class CustomerManagement {

   // Method to add a customer to the database
   public static void addCustomer(String name, String email, int points, double balance)
{
      String sql = "INSERT INTO Customer VALUES (?, ?, ?, ?)";
      try (Connection conn = Database.getConnection();
           PreparedStatement pstmt = conn.prepareStatement(sql)) {
         pstmt.setString(1, name);
         pstmt.setString(2, email);
```

```
        pstmt.setInt(3, points);
        pstmt.setDouble(4, balance);
        pstmt.executeUpdate();
        System.out.println("Customer added successfully!");
    } catch (SQLException e) {
        System.out.println("Error adding customer: " + e.getMessage());
    }
}

// Method to view all customers in the database
public static void viewCustomers() {
    String sql = "SELECT * FROM Customer";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            System.out.println("Customer ID: " + rs.getInt("customer_id"));
            System.out.println("Name: " + rs.getString("name"));
            System.out.println("Email: " + rs.getString("email"));
            System.out.println("Points: " + rs.getInt("points"));
            System.out.println("Balance: " + rs.getDouble("balance"));
            System.out.println("----------------------");
        }
    } catch (SQLException e) {
        System.out.println("Error viewing customers: " + e.getMessage());
    }
}

// Method to delete a customer by their ID
public static void deleteCustomer(int customerId) {
    String sql = "DELETE FROM Customer WHERE customer_id = ?";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, customerId);
        pstmt.executeUpdate();
        System.out.println("Customer deleted successfully!");
    } catch (SQLException e) {
        System.out.println("Error deleting customer: " + e.getMessage());
    }
}

// Method to get all customers from the database
public static Object[][] getCustomers() {
    String sql = "SELECT * FROM Customer";
    List<Object[]> customerList = new ArrayList<>();
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Object[] customer = new Object[]{
                rs.getInt("customer_id"),
                rs.getString("name"),
                rs.getString("email"),
                rs.getInt("points"),
                rs.getDouble("balance")
            };
            customerList.add(customer);
        }
    } catch (SQLException e) {
        System.out.println("Error retrieving customers: " + e.getMessage());
    }
    return customerList.toArray(new Object[0][]); // Convert list to 2D array
  }
}
```

iv. MachineManagement.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class MachineManagement {

  // Method to add a machine to the database
  public static void addMachine(String location, int ticketCount, boolean
dispensesTickets, String status) {
    String sql = "INSERT INTO Machine VALUES (?, ?, ?, ?)";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, location);
        pstmt.setInt(2, ticketCount);
        pstmt.setBoolean(3, dispensesTickets);
        pstmt.setString(4, status);
        pstmt.executeUpdate();
        System.out.println("Machine added successfully!");
    } catch (SQLException e) {
        System.out.println("Error adding machine: " + e.getMessage());
    }
  }

  // Method to view all machines in the database
  public static void viewMachines() {
    String sql = "SELECT * FROM Machine";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                System.out.println("Machine ID: " + rs.getInt("machine_id"));
                System.out.println("Location: " + rs.getString("location"));
                System.out.println("Ticket Count: " + rs.getInt("ticket_count"));
                System.out.println("Dispenses Tickets: " + rs.getBoolean("dispenses_tickets"));
                System.out.println("Status: " + rs.getString("status"));
            }
        } catch (SQLException e) {
            System.out.println("Error viewing machines: " + e.getMessage());
        }
    }

    // Method to delete a machine by its ID
    public static void deleteMachine(int machineId) {
        String sql = "DELETE FROM Machine WHERE machine_id = ?";
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, machineId);
            pstmt.executeUpdate();
            System.out.println("Machine deleted successfully!");
        } catch (SQLException e) {
            System.out.println("Error deleting machine: " + e.getMessage());
        }
    }

    // Method to get all machines from the database
    public static Object[][] getMachines() {
        String sql = "SELECT * FROM Machine";
        List<Object[]> machineList = new ArrayList<>();
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                Object[] machine = new Object[]{
                    rs.getInt("machine_id"),
                    rs.getString("location"),
                    rs.getInt("ticket_count"),
                    rs.getBoolean("dispenses_tickets"),
                    rs.getString("status")
                };
                machineList.add(machine);
            }
        } catch (SQLException e) {
            System.out.println("Error retrieving machines: " + e.getMessage());
        }
        return machineList.toArray(new Object[0][]); // Convert list to 2D array
    }
}
```

v. GameManagement.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class GameManagement {

  // Method to add a game to the database
  public static void addGame(String name, String location, String status) {
    String sql = "INSERT INTO Game (name, location, status) VALUES (?, ?, ?)";
    try (Connection conn = Database.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
      pstmt.setString(1, name);
      pstmt.setString(2, location);
      pstmt.setString(3, status);
      pstmt.executeUpdate();
      System.out.println("Game added successfully!");
    } catch (SQLException e) {
      System.out.println("Error adding game: " + e.getMessage());
    }
  }

  // Method to view all games in the database
  public static void viewGames() {
    String sql = "SELECT * FROM Game";
    try (Connection conn = Database.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        ResultSet rs = pstmt.executeQuery()) {
      while (rs.next()) {
        System.out.println("Game ID: " + rs.getInt("game_id"));
        System.out.println("Name: " + rs.getString("name"));
        System.out.println("Location: " + rs.getString("location"));
        System.out.println("Status: " + rs.getString("status"));
        System.out.println("----------------------");
      }
    } catch (SQLException e) {
      System.out.println("Error viewing games: " + e.getMessage());
    }
  }

  // Method to get all games from the database
  public static Object[][] getGames() {
    String sql = "SELECT * FROM Game";
    List<Object[]> gameList = new ArrayList<>();
    try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                Object[] game = new Object[] {
                    rs.getInt("game_id"),
                    rs.getString("name"),
                    rs.getString("location"),
                    rs.getString("status")
                };
                gameList.add(game);
            }
        } catch (SQLException e) {
            System.out.println("Error retrieving games: " + e.getMessage());
        }
        return gameList.toArray(new Object[0][]); // Convert list to 2D array
    }
}
```

vi. PointsManagement.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PointsManagement {

    // Method to add points for a customer in the database
    public static void addPoints(int customerId, int pointsEarned) {
        String sql = "INSERT INTO Points (customer_id, points_earned) VALUES (?, ?)";
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, customerId);
            pstmt.setInt(2, pointsEarned);
            pstmt.executeUpdate();
            System.out.println("Points added successfully!");
        } catch (SQLException e) {
            System.out.println("Error adding points: " + e.getMessage());
        }
    }

    // Method to view all points records in the database
    public static void viewPoints() {
```

```
    String sql = "SELECT * FROM Points";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            System.out.println("Point ID: " + rs.getInt("point_id"));
            System.out.println("Customer ID: " + rs.getInt("customer_id"));
            System.out.println("Points Earned: " + rs.getInt("points_earned"));
            System.out.println("Earned Time: " + rs.getTimestamp("earned_time"));
            System.out.println("----------------------");
        }
    } catch (SQLException e) {
        System.out.println("Error viewing points: " + e.getMessage());
    }
}

// Method to delete points by their ID
public static void deletePoints(int pointId) {
    String sql = "DELETE FROM Points WHERE point_id = ?";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, pointId);
        pstmt.executeUpdate();
        System.out.println("Points record deleted successfully!");
    } catch (SQLException e) {
        System.out.println("Error deleting points: " + e.getMessage());
    }
}

// Method to get all points records from the database
public static Object[][] getPoints() {
    String sql = "SELECT * FROM Points";
    List<Object[]> pointsList = new ArrayList<>();
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Object[] points = new Object[]{
                rs.getInt("point_id"),
                rs.getInt("customer_id"),
                rs.getInt("points_earned"),
                rs.getTimestamp("earned_time")
            };
            pointsList.add(points);
        }
    } catch (SQLException e) {
        System.out.println("Error retrieving points records: " + e.getMessage());
    }
    return pointsList.toArray(new Object[0][]); // Convert list to 2D array
    }
}
```

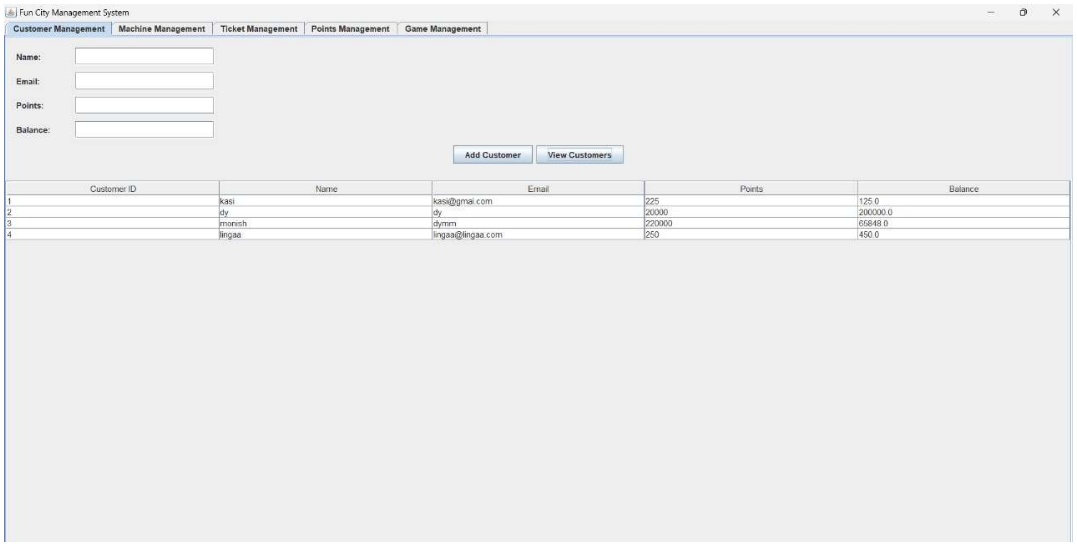
vii. TicketManagement.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class TicketManagement {

    // Method to add a ticket to the database
    public static void addTicket(int customerId, int machineId) {
        String sql = "INSERT INTO Ticket (customer_id, machine_id) VALUES (?, ?)";
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, customerId);
            pstmt.setInt(2, machineId);
            pstmt.executeUpdate();
            System.out.println("Ticket added successfully!");
        } catch (SQLException e) {
            System.out.println("Error adding ticket: " + e.getMessage());
        }
    }

    // Method to view all tickets in the database
    public static void viewTickets() {
        String sql = "SELECT * FROM Ticket";
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                System.out.println("Ticket ID: " + rs.getInt("ticket_id"));
                System.out.println("Customer ID: " + rs.getInt("customer_id"));
                System.out.println("Machine ID: " + rs.getInt("machine_id"));
                System.out.println("Issue Time: " + rs.getTimestamp("issue_time"));
                System.out.println("----------------------");
            }
        } catch (SQLException e) {
            System.out.println("Error viewing tickets: " + e.getMessage());
        }
    }

    // Method to delete a ticket by its ID
    public static void deleteTicket(int ticketId) {
        String sql = "DELETE FROM Ticket WHERE ticket_id = ?";
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, ticketId);
            pstmt.executeUpdate();
            System.out.println("Ticket deleted successfully!");
        } catch (SQLException e) {
            System.out.println("Error deleting ticket: " + e.getMessage());
        }
    }

    // Method to get all tickets from the database
    public static Object[][] getTickets() {
        String sql = "SELECT * FROM Ticket";
        List<Object[]> ticketList = new ArrayList<>();
        try (Connection conn = Database.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                Object[] ticket = new Object[]{
                    rs.getInt("ticket_id"),
                    rs.getInt("customer_id"),
                    rs.getInt("machine_id"),
                    rs.getTimestamp("issue_time")
                };
                ticketList.add(ticket);
            }
        } catch (SQLException e) {
            System.out.println("Error retrieving tickets: " + e.getMessage());
        }
        return ticketList.toArray(new Object[0][]); // Convert list to 2D array
    }
}
```

6. SNAPSHOTS

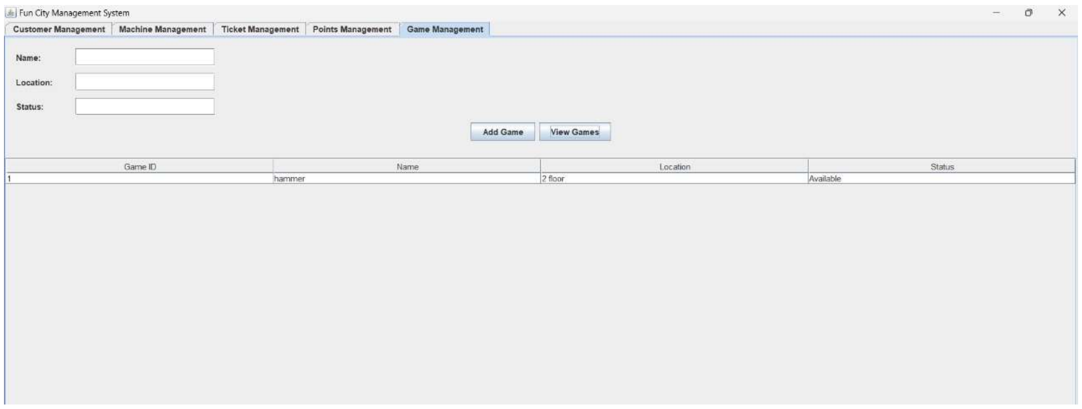
Here are the snapshots of this project UI

1. Customer Management



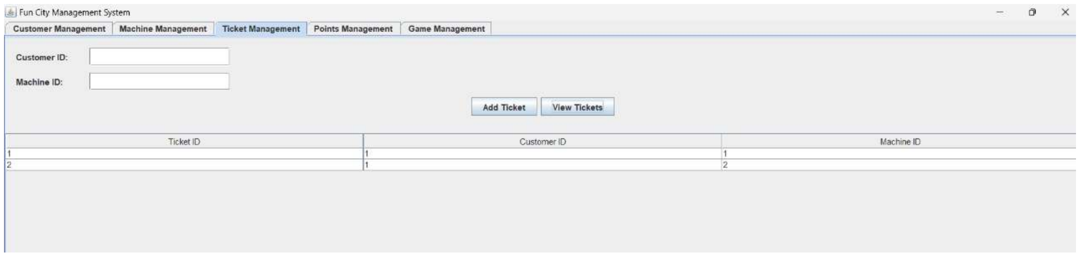
Customer ID	Name	Email	Points	Balance
1	kasi	kasi@gmai.com	225	125.0
2	dy	dy	20000	200000.0
3	monish	dymm	220000	65848.0
4	lingaa	lingaa@lingaa.com	250	450.0

2. Game Management

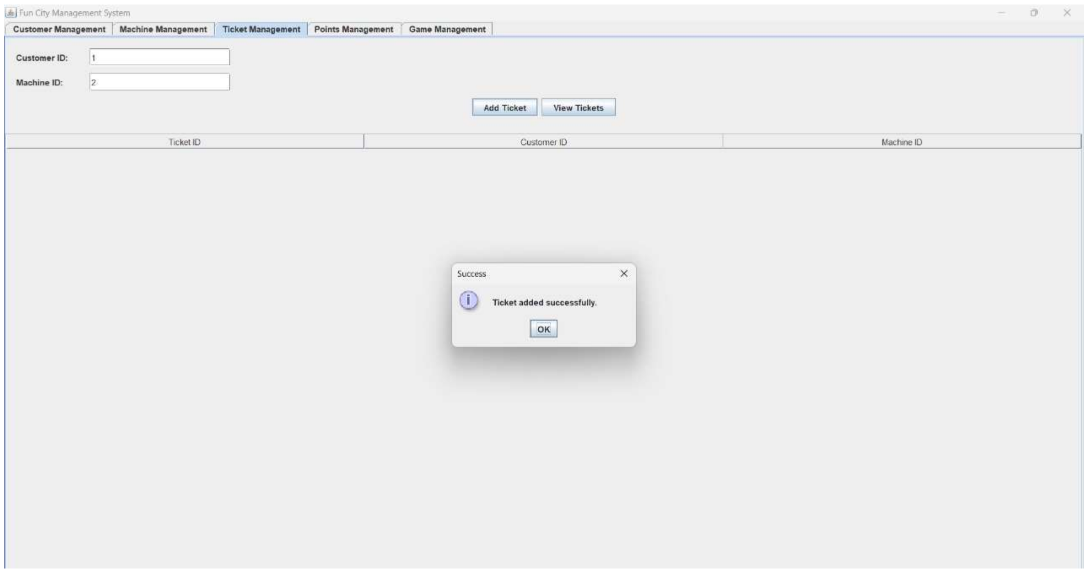


Game ID	Name	Location	Status
1	hammer	2 floor	Available

3. Ticket Management



Ticket ID	Customer ID	Machine ID
1	1	1
2	1	2



Customer ID: 1

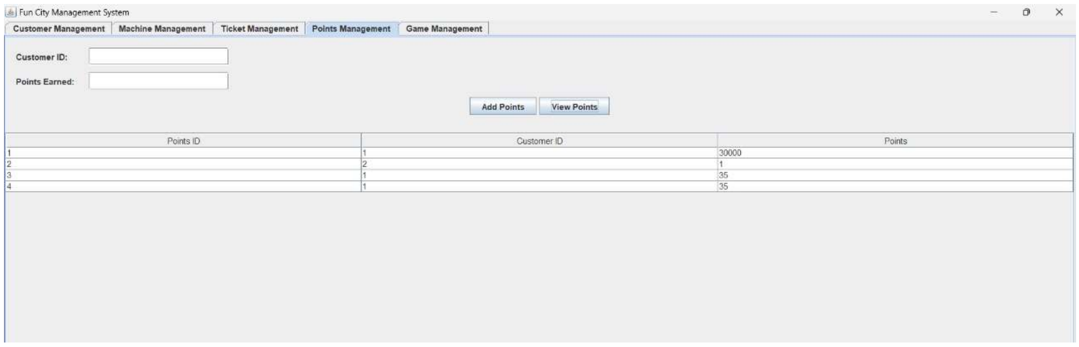
Machine ID: 2

Ticket ID	Customer ID	Machine ID

Success

Ticket added successfully.

4. Points Management



Customer ID:

Points Earned:

Points ID	Customer ID	Points
1	1	30000
2	2	1
3	1	35
4	1	35

7. CONCLUSION

The FunCity Management System successfully streamlines the management of a mall's gaming section by automating customer registration, ticket booking, game reservations, and points tracking. It reduces manual errors, improves operational efficiency, and enhances the visitor experience. The system's intuitive design and robust database integration ensure seamless data handling and scalability for future upgrades. By leveraging Java's capabilities and a well-structured MySQL database, the project demonstrates practical solutions for real-world management challenges.

8. REFERENCES

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

https://www.javatpoint.com/jdbc-tutorial