1.Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.text.*;
import java.util.*;
import java.util.List;

class DBConnection {
   private static final String URL =
"jdbc:mysql://localhost:3306/IRCTC_System";
   private static final String USER = "root";
   private static final String PASSWORD = "root"; // Replace with your
MySQL password

   public static Connection getConnection() throws SQLException {
      return DriverManager.getConnection(URL, USER, PASSWORD);
   }

   public static void initializeDatabase() {
      try (Connection conn = getConnection()) {
         String createUsersTable = "CREATE TABLE IF NOT EXISTS Users
(" +
                              "user_id INT AUTO_INCREMENT PRIMARY KEY, "
+
                              "name VARCHAR(100) NOT NULL, " +
                              "mobile VARCHAR(15) NOT NULL, " +
                              "email VARCHAR(100) UNIQUE NOT NULL, " +
                              "password VARCHAR(100) NOT NULL, " +
                              "security_question VARCHAR(200) NOT NULL, " +
                              "security_answer VARCHAR(100) NOT NULL)";
```

```
        String createTrainsTable = "CREATE TABLE IF NOT EXISTS Trains
(" +
                        "train_id INT AUTO_INCREMENT PRIMARY KEY,
" +
                        "train_number VARCHAR(10) UNIQUE NOT NULL,
" +
                        "train_name VARCHAR(100) NOT NULL, " +
                        "source_station VARCHAR(100) NOT NULL, " +
                        "destination_station VARCHAR(100) NOT NULL, "
+
                        "departure_time VARCHAR(10) NOT NULL, " +
                        "arrival_time VARCHAR(10) NOT NULL, " +
                        "duration VARCHAR(10) NOT NULL, " +
                        "runs_on VARCHAR(50) NOT NULL)";

        String createBookingsTable = "CREATE TABLE IF NOT EXISTS
Bookings (" +
                         "booking_id INT AUTO_INCREMENT PRIMARY
KEY, " +
                         "user_id INT NOT NULL, " +
                         "train_number VARCHAR(10) NOT NULL, " +
                         "train_name VARCHAR(100) NOT NULL, " +
                         "from_station VARCHAR(100) NOT NULL, " +
                         "to_station VARCHAR(100) NOT NULL, " +
                         "journey_date DATE NOT NULL, " +
                         "class_type VARCHAR(50) NOT NULL, " +
                         "passenger_name VARCHAR(100) NOT NULL, "
+
                         "age INT NOT NULL, " +
                         "gender VARCHAR(10) NOT NULL, " +
                         "berth_preference VARCHAR(20), " +
                         "fare DECIMAL(10,2) NOT NULL, " +
                         "pnr_number VARCHAR(15) UNIQUE NOT NULL,
" +
                        "booking_status VARCHAR(20) DEFAULT
'Confirmed', " +
                        "seat_number VARCHAR(10), " +
                        "booking_date TIMESTAMP DEFAULT
CURRENT_TIMESTAMP, " +
                        "FOREIGN KEY (user_id) REFERENCES
Users(user_id))";

        String createAccountHolderTable = "CREATE TABLE IF NOT
EXISTS AccountHolder (" +
                          "account_holder_id INT AUTO_INCREMENT
PRIMARY KEY, " +
                          "name VARCHAR(100), " +
                          "age INT, " +
                          "account_number VARCHAR(20) UNIQUE
NOT NULL, " +
                          "balance DECIMAL(15, 2), " +
                          "interest DECIMAL(5, 2), " +
                          "user_id INT, " +
                          "FOREIGN KEY (user_id) REFERENCES
Users(user_id))";

        String createProfilesTable = "CREATE TABLE IF NOT EXISTS
Profiles (" +
                       "profile_id INT AUTO_INCREMENT PRIMARY
KEY, " +
                       "user_id INT NOT NULL, " +
                       "age INT, " +
                       "gender VARCHAR(10), " +
                       "preferred_class VARCHAR(50), " +
                       "gov_id VARCHAR(20), " +
                       "emergency_contact VARCHAR(15), " +
                       "wheelchair_access BOOLEAN DEFAULT FALSE,
" +
                                "medical_condition BOOLEAN DEFAULT FALSE, "
+
                                "FOREIGN KEY (user_id) REFERENCES
Users(user_id))";

          String createSessionLogsTable = "CREATE TABLE IF NOT EXISTS
SessionLogs (" +
                                  "log_id INT AUTO_INCREMENT PRIMARY KEY,
" +
                                  "user_id INT, " +
                                  "action VARCHAR(100) NOT NULL, " +
                                  "timestamp TIMESTAMP DEFAULT
CURRENT_TIMESTAMP, " +
                                  "session_id VARCHAR(50))";

          Statement stmt = conn.createStatement();
          stmt.execute(createUsersTable);
          stmt.execute(createTrainsTable);
          stmt.execute(createBookingsTable);
          stmt.execute(createAccountHolderTable);
          stmt.execute(createProfilesTable);
          stmt.execute(createSessionLogsTable);

          String checkTrains = "SELECT COUNT(*) FROM Trains";
          ResultSet rs = stmt.executeQuery(checkTrains);
          rs.next();
          if (rs.getInt(1) == 0) {
             insertSampleTrainData(conn);
          }

          String checkAccounts = "SELECT COUNT(*) FROM
AccountHolder";
          rs = stmt.executeQuery(checkAccounts);
          rs.next();
          if (rs.getInt(1) == 0) {
            insertSampleAccountHolderData(conn);
        }

    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Database initialization
failed: " + e.getMessage(),
                                    "Database Error",
JOptionPane.ERROR_MESSAGE);
    }
}

private static void insertSampleTrainData(Connection conn) throws
SQLException {
    String sql = "INSERT INTO Trains (train_number, train_name,
source_station, destination_station, " +
                "departure_time, arrival_time, duration, runs_on) VALUES (?,
?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement stmt = conn.prepareStatement(sql);

    Object[][] trains = {
        {"12001", "Shatabdi Express", "New Delhi", "Bhopal", "08:10",
"16:25", "08:15", "Daily"},
        {"12309", "Rajdhani Express", "New Delhi", "Kolkata", "17:00",
"10:55", "17:55", "Daily"},
        {"12621", "Tamil Nadu Express", "New Delhi", "Chennai", "22:30",
"07:10", "32:40", "Daily"},
        {"12295", "Duronto Express", "Mumbai", "Kolkata", "17:15", "18:50",
"25:35", "Mon,Wed,Fri"},
        {"12055", "Jan Shatabdi", "Dehradun", "New Delhi", "15:20",
"21:10", "05:50", "Daily"},
        {"12951", "Rajdhani Express", "Mumbai", "Delhi", "16:35", "08:15",
"15:40", "Daily"},
        {"12952", "Rajdhani Express", "Delhi", "Mumbai", "16:55", "08:30",
"15:35", "Daily"},
        {"12301", "Duronto Express", "Howrah", "Delhi", "22:05", "08:00",
"09:55", "Tue,Thu,Sat"},
        {"12302", "Duronto Express", "Delhi", "Howrah", "22:45", "08:40",
"09:55", "Mon,Wed,Fri"},
        {"12305", "Duronto Express", "Sealdah", "Delhi", "22:50", "09:40",
"10:50", "Daily"}
    };

    for (Object[] train : trains) {
        for (int i = 0; i < train.length; i++) {
            stmt.setString(i + 1, (String) train[i]);
        }
        stmt.addBatch();
    }
    stmt.executeBatch();
}

private static void insertSampleAccountHolderData(Connection conn)
throws SQLException {
    String sql = "INSERT INTO AccountHolder (name, age,
account_number, balance, interest, user_id) " +
                "VALUES (?, ?, ?, ?, ?, ?)";
    PreparedStatement stmt = conn.prepareStatement(sql);

    Object[][] accounts = {
        {"John Doe", 30, "ACC1001", 50000.00, 3.5, null},
        {"Jane Smith", 25, "ACC1002", 75000.00, 3.5, null}
    };

    for (Object[] account : accounts) {
        stmt.setString(1, (String) account[0]);
        stmt.setInt(2, (Integer) account[1]);
        stmt.setString(3, (String) account[2]);
        stmt.setDouble(4, (Double) account[3]);
        stmt.setDouble(5, (Double) account[4]);
```

```
          stmt.setObject(6, account[5], Types.INTEGER);
          stmt.addBatch();
      }
      stmt.executeBatch();
   }

   public static void logSession(int userId, String action, String sessionId)
throws SQLException {
      try (Connection conn = getConnection()) {
          String sql = "INSERT INTO SessionLogs (user_id, action,
session_id) VALUES (?, ?, ?)";
          PreparedStatement stmt = conn.prepareStatement(sql);
          stmt.setInt(1, userId);
          stmt.setString(2, action);
          stmt.setString(3, sessionId);
          stmt.executeUpdate();
      }
   }

   public static double getAccountBalance(int userId) throws SQLException
{
      try (Connection conn = getConnection()) {
          String sql = "SELECT balance FROM AccountHolder WHERE
user_id = ?";
          PreparedStatement stmt = conn.prepareStatement(sql);
          stmt.setInt(1, userId);
          ResultSet rs = stmt.executeQuery();
          if (rs.next()) {
             return rs.getDouble("balance");
          }
          return 0.0;
      }
   }
```

```
    public static boolean deductBalance(int userId, double amount) throws
SQLException {
        try (Connection conn = getConnection()) {
            conn.setAutoCommit(false);
            try {
                String checkSql = "SELECT balance FROM AccountHolder
WHERE user_id = ?";
                PreparedStatement checkStmt =
conn.prepareStatement(checkSql);
                checkStmt.setInt(1, userId);
                ResultSet rs = checkStmt.executeQuery();

                if (rs.next()) {
                    double currentBalance = rs.getDouble("balance");
                    if (currentBalance >= amount) {
                        String updateSql = "UPDATE AccountHolder SET balance =
balance - ? WHERE user_id = ?";
                        PreparedStatement updateStmt =
conn.prepareStatement(updateSql);
                        updateStmt.setDouble(1, amount);
                        updateStmt.setInt(2, userId);
                        updateStmt.executeUpdate();
                        conn.commit();
                        return true;
                    }
                }
                conn.rollback();
                return false;
            } catch (SQLException e) {
                conn.rollback();
                throw e;
            } finally {
                conn.setAutoCommit(true);
            }
        }
    }

    public static void refundBalance(int userId, double amount) throws
SQLException {
        try (Connection conn = getConnection()) {
            String sql = "UPDATE AccountHolder SET balance = balance + ?
WHERE user_id = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setDouble(1, amount);
            stmt.setInt(2, userId);
            stmt.executeUpdate();
        }
    }
}

class User {
    private int userId;
    private String name;
    private String mobile;
    private String email;
    private String password;
    private String securityQuestion;
    private String securityAnswer;
    private String sessionId;

    public User(String name, String mobile, String email, String password,
                String securityQuestion, String securityAnswer) {
        this.name = name;
        this.mobile = mobile;
        this.email = email;
        this.password = password;
        this.securityQuestion = securityQuestion;
        this.securityAnswer = securityAnswer;
        this.sessionId = UUID.randomUUID().toString();
    }
    private User(int userId, String name, String mobile, String email, String
password,
                String securityQuestion, String securityAnswer, String sessionId)
{
        this.userId = userId;
        this.name = name;
        this.mobile = mobile;
        this.email = email;
        this.password = password;
        this.securityQuestion = securityQuestion;
        this.securityAnswer = securityAnswer;
        this.sessionId = sessionId;
    }

    public void register() throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "INSERT INTO Users (name, mobile, email, password,
security_question, security_answer) " +
                         "VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS);
            stmt.setString(1, name);
            stmt.setString(2, mobile);
            stmt.setString(3, email);
            stmt.setString(4, password);
            stmt.setString(5, securityQuestion);
            stmt.setString(6, securityAnswer);
            stmt.executeUpdate();

            ResultSet rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                this.userId = rs.getInt(1);
            }
            DBConnection.logSession(userId, "register", sessionId);

            String accountSql = "INSERT INTO AccountHolder (name, age,
account_number, balance, interest, user_id) " +
                                "VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement accountStmt =
conn.prepareStatement(accountSql);
            accountStmt.setString(1, name);
            accountStmt.setInt(2, 0);
            accountStmt.setString(3, "ACC" + (1000 + new
Random().nextInt(9000)));
            accountStmt.setDouble(4, 10000.0);
            accountStmt.setDouble(5, 3.5);
            accountStmt.setInt(6, userId);
            accountStmt.executeUpdate();
        }
    }

    public static User login(String email, String password) throws
SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "SELECT * FROM Users WHERE email = ? AND
password = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, email);
            stmt.setString(2, password);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                String sessionId = UUID.randomUUID().toString();
                User user = new User(
                    rs.getInt("user_id"),
                    rs.getString("name"),
                    rs.getString("mobile"),
                    rs.getString("email"),
                    rs.getString("password"),
                    rs.getString("security_question"),
                    rs.getString("security_answer"),
                    sessionId
                );
                DBConnection.logSession(user.userId, "login", sessionId);
                return user;
            }
            return null;
        }
    }

    public boolean hasCompleteProfile() throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "SELECT COUNT(*) FROM Profiles WHERE user_id =
?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, userId);
            ResultSet rs = stmt.executeQuery();
            rs.next();
            return rs.getInt(1) > 0;
        }
    }

    public void logout() {
        try {
            DBConnection.logSession(userId, "logout", sessionId);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        sessionId = null;
    }

    public static String getSecurityQuestion(String email) throws
SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "SELECT security_question FROM Users WHERE
email = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getString("security_question");
            }
            return null;
        }
    }

    public static boolean verifySecurityAnswer(String email, String answer)
throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "SELECT security_answer FROM Users WHERE email
= ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return
rs.getString("security_answer").equalsIgnoreCase(answer);
            }
            return false;
        }
    }

    public static void resetPassword(String email, String newPassword)
throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "UPDATE Users SET password = ? WHERE email = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, newPassword);
            stmt.setString(2, email);
            stmt.executeUpdate();
        }
    }

    public int getUserId() { return userId; }
    public String getName() { return name; }
    public String getSessionId() { return sessionId; }
}

class Profile {
    private int userId;
    private int age;
    private String gender;
    private String preferredClass;
    private String govId;
    private String emergencyContact;
    private boolean wheelchairAccess;
    private boolean medicalCondition;

    public Profile(int userId, int age, String gender, String preferredClass,
String govId,
                   String emergencyContact, boolean wheelchairAccess, boolean
medicalCondition) {
        this.userId = userId;
        this.age = age;
        this.gender = gender;
        this.preferredClass = preferredClass;
        this.govId = govId;
        this.emergencyContact = emergencyContact;
        this.wheelchairAccess = wheelchairAccess;
        this.medicalCondition = medicalCondition;
    }

    public void save() throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "INSERT INTO Profiles (user_id, age, gender,
preferred_class, gov_id, " +
                        "emergency_contact, wheelchair_access,
medical_condition) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, userId);
            stmt.setInt(2, age);
            stmt.setString(3, gender);
            stmt.setString(4, preferredClass);
            stmt.setString(5, govId);
            stmt.setString(6, emergencyContact);
            stmt.setBoolean(7, wheelchairAccess);
            stmt.setBoolean(8, medicalCondition);
            stmt.executeUpdate();
        }
    }

    public static Profile getProfile(int userId) throws SQLException {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "SELECT * FROM Profiles WHERE user_id = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, userId);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new Profile(
                    userId,
                    rs.getInt("age"),
                    rs.getString("gender"),
                    rs.getString("preferred_class"),
                    rs.getString("gov_id"),
                    rs.getString("emergency_contact"),
                    rs.getBoolean("wheelchair_access"),
                    rs.getBoolean("medical_condition")
                );
            }
            return null;
        }
    }

    public int getAge() { return age; }
    public String getGender() { return gender; }
    public String getPreferredClass() { return preferredClass; }
    public String getGovId() { return govId; }
    public String getEmergencyContact() { return emergencyContact; }
    public boolean isWheelchairAccess() { return wheelchairAccess; }
    public boolean isMedicalCondition() { return medicalCondition; }
}

class Train {
    private String trainNumber;
    private String trainName;
    private String sourceStation;
    private String destinationStation;
    private String departureTime;
    private String arrivalTime;
    private String duration;
    private String runsOn;

    public Train(String trainNumber, String trainName, String sourceStation,
String destinationStation,
                 String departureTime, String arrivalTime, String duration, String
runsOn) {
        this.trainNumber = trainNumber;
        this.trainName = trainName;
        this.sourceStation = sourceStation;
        this.destinationStation = destinationStation;
        this.departureTime = departureTime;
        this.arrivalTime = arrivalTime;
        this.duration = duration;
        this.runsOn = runsOn;
    }

    public static List<Train> searchTrainsWithFilters(String from, String to,
String journeyDate,
                                                      String classType, String trainType, String
timeRange) throws SQLException {
        List<Train> trains = new ArrayList<>();
        try (Connection conn = DBConnection.getConnection()) {
            StringBuilder sql = new StringBuilder(
                "SELECT * FROM Trains WHERE source_station LIKE ? AND
destination_station LIKE ?"
            );

            if (!trainType.equals("All")) {
                sql.append(" AND train_name LIKE ?");
            }

            PreparedStatement stmt = conn.prepareStatement(sql.toString());
            stmt.setString(1, "%" + from + "%");
            stmt.setString(2, "%" + to + "%");

            int paramIndex = 3;
            if (!trainType.equals("All")) {
                stmt.setString(paramIndex++, "%" + trainType + "%");
            }

            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Train train = new Train(
                    rs.getString("train_number"),
                    rs.getString("train_name"),
                    rs.getString("source_station"),
                    rs.getString("destination_station"),
                    rs.getString("departure_time"),
                    rs.getString("arrival_time"),
                    rs.getString("duration"),
                    rs.getString("runs_on")
                );

                SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
                try {
                    java.util.Date date = sdf.parse(journeyDate);
                    String[] days = train.runsOn.split(",");
                    String journeyDay = new
SimpleDateFormat("EEE").format(date).toLowerCase();
                    boolean runsOnDay = false;
                    for (String day : days) {
                        if (day.trim().toLowerCase().startsWith(journeyDay)) {
                            runsOnDay = true;
                            break;
                        }
                    }
                    if (runsOnDay) {
                        if (timeRange != null && !timeRange.equals("All")) {
                            String[] times = timeRange.split("-");
                            int startHour = Integer.parseInt(times[0].split(":")[0]);
                            int endHour = Integer.parseInt(times[1].split(":")[0]);
                            int departureHour =
Integer.parseInt(train.departureTime.split(":")[0]);
                            if (departureHour >= startHour && departureHour <=
endHour) {
                            trains.add(train);
                            }
                        } else {
                            trains.add(train);
                        }
                    }
                } catch (ParseException e) {
                    throw new SQLException("Invalid date format: " + journeyDate,
e);
                }
            }
        }
        return trains;
}

public boolean isClassAvailable(String classType) {
        return getAvailabilityForClass(classType).equals("Available");
}

public String getAvailabilityForClass(String classType) {
        Random random = new Random();
        int availability = random.nextInt(100);

        if (availability > 80) {
            return "WL " + (random.nextInt(50) + 1);
        } else if (availability > 60) {
            return "RAC " + (random.nextInt(20) + 1);
        } else {
            return "Available";
        }
}

public String getTrainNumber() { return trainNumber; }
public String getTrainName() { return trainName; }
public String getSourceStation() { return sourceStation; }
public String getDestinationStation() { return destinationStation; }
public String getDepartureTime() { return departureTime; }
public String getArrivalTime() { return arrivalTime; }
public String getDuration() { return duration; }
public String getRunsOn() { return runsOn; }
}
```

```
class Booking {
   private static final String[] CLASS_TYPES = {
      "Anubhuti Class (EA)", "AC First Class (1A)", "Vistadome AC (EV)",
      "Exec. Chair Car (EC)", "AC 2 Tier (2A)", "AC 3 Tier (3A)",
      "AC 3 Economy (3E)", "Sleeper (SL)", "Second Sitting (2S)", "General
(GN)"
   };

   private static final String[] QUOTA_TYPES = {"General", "Tatkal",
"Premium Tatkal", "Ladies", "Senior Citizen"};

   private static final Map<String, Double> CLASS_FARES = new
HashMap<>();
   private static final Map<String, Double> CONCESSION_RATES = new
HashMap<>();

   static {
      CLASS_FARES.put("Anubhuti Class (EA)", 4500.0);
      CLASS_FARES.put("AC First Class (1A)", 3500.0);
      CLASS_FARES.put("Vistadome AC (EV)", 3200.0);
      CLASS_FARES.put("Exec. Chair Car (EC)", 2800.0);
      CLASS_FARES.put("AC 2 Tier (2A)", 2500.0);
      CLASS_FARES.put("AC 3 Tier (3A)", 1800.0);
      CLASS_FARES.put("AC 3 Economy (3E)", 1500.0);
      CLASS_FARES.put("Sleeper (SL)", 800.0);
      CLASS_FARES.put("Second Sitting (2S)", 400.0);
      CLASS_FARES.put("General (GN)", 300.0);

      CONCESSION_RATES.put("Senior Citizen", 0.5);
      CONCESSION_RATES.put("Student", 0.25);
      CONCESSION_RATES.put("Person with Disability", 0.4);
      CONCESSION_RATES.put("Child (5-12 years)", 0.5);
      CONCESSION_RATES.put("Armed Forces", 0.3);
   }
```

```
    public static Map<String, Double> getClassFares() {
        return CLASS_FARES;
    }

    public static Map<String, Double> getConcessionRates() {
        return CONCESSION_RATES;
    }

    public static String bookTickets(int userId, String trainNumber, String
trainName,
                                   String fromStation, String toStation, String
journeyDate,
                                   String classType, String quota, List<Passenger>
passengers,
                                   String paymentMethod) throws SQLException {
        Connection conn = DBConnection.getConnection();

        String pnr = generatePNR();
        double totalFare = 0.0;
        Random random = new Random();

        double baseFare = getClassFares().getOrDefault(classType, 0.0);
        if (quota.equals("Tatkal")) baseFare *= 1.3;
        if (quota.equals("Premium Tatkal")) baseFare *= 1.5;

        for (Passenger passenger : passengers) {
            double fare = baseFare;
            if (passenger.getConcessionType() != null &&
!passenger.getConcessionType().isEmpty() &&

getConcessionRates().containsKey(passenger.getConcessionType())) {
                double discount = fare *
getConcessionRates().get(passenger.getConcessionType());
                fare -= discount;
            }
```

```
          totalFare += fare;
        }

        boolean paymentSuccess = DBConnection.deductBalance(userId,
totalFare);
        if (!paymentSuccess) {
            throw new SQLException("Insufficient balance to complete the
booking.");
        }

        try {
            String sql = "INSERT INTO Bookings (user_id, train_number,
train_name, from_station, " +
                        "to_station, journey_date, class_type, passenger_name,
age, gender, " +
                        "berth_preference, fare, pnr_number, booking_status,
seat_number) " +
                        "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

            PreparedStatement stmt = conn.prepareStatement(sql);
            for (Passenger passenger : passengers) {
                double fare = baseFare;
                if (passenger.getConcessionType() != null &&
!passenger.getConcessionType().isEmpty()) {
                    double discount = fare *
getConcessionRates().get(passenger.getConcessionType());
                    fare -= discount;
                }

                String bookingStatus = "Confirmed";
                String seatNumber = String.format("%03d",
random.nextInt(1000));
                Train tempTrain = new Train(trainNumber, trainName,
fromStation, toStation, "", "", "", "");
                String availability = tempTrain.getAvailabilityForClass(classType);
                if (availability.startsWith("WL")) {
                    bookingStatus = "Waiting List";
                    seatNumber = availability;
                } else if (availability.startsWith("RAC")) {
                    bookingStatus = "RAC";
                    seatNumber = availability;
                }

                stmt.setInt(1, userId);
                stmt.setString(2, trainNumber);
                stmt.setString(3, trainName);
                stmt.setString(4, fromStation);
                stmt.setString(5, toStation);
                stmt.setString(6, convertDate(journeyDate));
                stmt.setString(7, classType);
                stmt.setString(8, passenger.getName());
                stmt.setInt(9, passenger.getAge());
                stmt.setString(10, passenger.getGender());
                stmt.setString(11, passenger.getBerthPreference());
                stmt.setDouble(12, fare);
                stmt.setString(13, pnr);
                stmt.setString(14, bookingStatus);
                stmt.setString(15, seatNumber);

                stmt.addBatch();
            }
            stmt.executeBatch();

            DBConnection.logSession(userId, "book_tickets via " +
paymentMethod, pnr);
            return pnr;
        } catch (SQLException e) {
            DBConnection.refundBalance(userId, totalFare);
            throw e;
        }
    }

    private static String generatePNR() {
        Random random = new Random();
        return "PNR" + (100000 + random.nextInt(900000));
    }

    private static String convertDate(String journeyDate) throws
SQLException {
        try {
            SimpleDateFormat inputFormat = new
SimpleDateFormat("dd/MM/yyyy");
            SimpleDateFormat outputFormat = new
SimpleDateFormat("yyyy-MM-dd");
            return outputFormat.format(inputFormat.parse(journeyDate));
        } catch (ParseException e) {
            throw new SQLException("Invalid date format: " + journeyDate, e);
        }
    }

    public static ResultSet getBookingHistory(int userId) throws
SQLException {
        Connection conn = DBConnection.getConnection();
        String sql = "SELECT * FROM Bookings WHERE user_id = ? ORDER
BY booking_date DESC";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setInt(1, userId);
        return stmt.executeQuery();
    }

    public static boolean cancelTicket(String pnrNumber) throws
SQLException {
        Connection conn = DBConnection.getConnection();
        conn.setAutoCommit(false);
        try {
            String getBookingSql = "SELECT user_id, fare FROM Bookings
WHERE pnr_number = ? AND booking_status = 'Confirmed'";
            PreparedStatement getBookingStmt =
conn.prepareStatement(getBookingSql);
            getBookingStmt.setString(1, pnrNumber);
            ResultSet rs = getBookingStmt.executeQuery();

            if (!rs.next()) {
                conn.rollback();
                return false;
            }

            int userId = rs.getInt("user_id");
            double totalFare = 0.0;

            do {
                totalFare += rs.getDouble("fare");
            } while (rs.next());

            String updateSql = "UPDATE Bookings SET booking_status =
'Cancelled' WHERE pnr_number = ?";
            PreparedStatement updateStmt =
conn.prepareStatement(updateSql);
            updateStmt.setString(1, pnrNumber);
            int rowsAffected = updateStmt.executeUpdate();

            if (rowsAffected > 0) {
                DBConnection.refundBalance(userId, totalFare);
                DBConnection.logSession(userId, "cancel_ticket", pnrNumber);
                conn.commit();
                return true;
            }

            conn.rollback();
            return false;
```

```
        } catch (SQLException e) {
            conn.rollback();
            throw e;
        } finally {
            conn.setAutoCommit(true);
        }
    }

    public static ResultSet getPNRStatus(String pnrNumber) throws
SQLException {
        Connection conn = DBConnection.getConnection();
        String sql = "SELECT * FROM Bookings WHERE pnr_number = ?";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setString(1, pnrNumber);
        return stmt.executeQuery();
    }

    public static String[] getClassTypes() {
        return CLASS_TYPES;
    }

    public static String[] getQuotaTypes() {
        return QUOTA_TYPES;
    }

    public static String[] getConcessionTypes() {
        return CONCESSION_RATES.keySet().toArray(new String[0]);
    }

    public static class Passenger {
        private String name;
        private int age;
        private String gender;
        private String berthPreference;
        private String concessionType;
        public Passenger(String name, int age, String gender, String
berthPreference, String concessionType) {
            this.name = name;
            this.age = age;
            this.gender = gender;
            this.berthPreference = berthPreference;
            this.concessionType = concessionType;
        }

        public String getName() { return name; }
        public int getAge() { return age; }
        public String getGender() { return gender; }
        public String getBerthPreference() { return berthPreference; }
        public String getConcessionType() { return concessionType; }
    }
}

public class TrainManagementSystem extends JFrame {
    private User currentUser;
    private JPanel currentPanel;
    private JTextArea outputArea;
    private boolean darkTheme = false;
    private JLabel userInfoLabel;
    private List<String> recentSearches = new ArrayList<>();

    public TrainManagementSystem() {
        super("IRCTC Train Booking System");
        setSize(1200, 800);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        DBConnection.initializeDatabase();

        JPanel headerPanel = createHeaderPanel();
        add(headerPanel, BorderLayout.NORTH);

        currentPanel = new JPanel();
        add(currentPanel, BorderLayout.CENTER);

        JPanel footerPanel = createFooterPanel();
        add(footerPanel, BorderLayout.SOUTH);

        showLoginPanel();

        setLocationRelativeTo(null);
        setVisible(true);
    }

    private JPanel createHeaderPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBackground(new Color(0, 100, 0));
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        JLabel titleLabel = new JLabel("IRCTC - Indian Railway Catering and
Tourism Corporation", SwingConstants.CENTER);
        titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
        titleLabel.setForeground(Color.WHITE);
        panel.add(titleLabel, BorderLayout.CENTER);

        JPanel rightPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));
        rightPanel.setBackground(new Color(0, 100, 0));

        userInfoLabel = new JLabel("Welcome, Guest");
        userInfoLabel.setForeground(Color.WHITE);
        userInfoLabel.setFont(new Font("Arial", Font.PLAIN, 14));
        rightPanel.add(userInfoLabel);
        JLabel dateLabel = new JLabel(new
SimpleDateFormat("dd-MMM-yyyy [HH:mm:ss]").format(new
java.util.Date()));
        dateLabel.setForeground(Color.WHITE);
        dateLabel.setFont(new Font("Arial", Font.PLAIN, 14));
        dateLabel.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
        rightPanel.add(dateLabel);

        panel.add(rightPanel, BorderLayout.EAST);

        return panel;
    }

    private JPanel createFooterPanel() {
        JPanel panel = new JPanel();
        panel.setBackground(new Color(0, 100, 0));
        panel.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));

        JLabel footerLabel = new JLabel("Customer Care Numbers: 1464 /
80904687989 / 08035734999 | " +
                                "BEWARE OF FRAUDSTERS: Always download
official IRCTC Rail Connect App from the Google Play Store or Apple App
Store only.");
        footerLabel.setForeground(Color.WHITE);
        footerLabel.setFont(new Font("Arial", Font.PLAIN, 10));
        panel.add(footerLabel);

        return panel;
    }

    private void applyTheme() {
    if (darkTheme) {
        currentPanel.setBackground(new Color(60, 63, 65));
        currentPanel.setForeground(Color.WHITE);
        if (outputArea != null) {
            outputArea.setBackground(new Color(60, 63, 65));
            outputArea.setForeground(Color.WHITE);
        }
    } else {
        currentPanel.setBackground(new Color(238, 238, 238));
        currentPanel.setForeground(Color.BLACK);
        if (outputArea != null) {
            outputArea.setBackground(Color.WHITE);
            outputArea.setForeground(Color.BLACK);
        }
    }
}

private void showLoginPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Login to IRCTC",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel emailLabel = new JLabel("Email:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(emailLabel, gbc);

    JTextField emailField = new JTextField(20);
```

```
gbc.gridx = 1;
gbc.gridy = 1;
currentPanel.add(emailField, gbc);

JLabel passwordLabel = new JLabel("Password:");
gbc.gridx = 0;
gbc.gridy = 2;
currentPanel.add(passwordLabel, gbc);

JPasswordField passwordField = new JPasswordField(20);
gbc.gridx = 1;
gbc.gridy = 2;
currentPanel.add(passwordField, gbc);

JButton loginButton = new JButton("Login");
gbc.gridx = 0;
gbc.gridy = 3;
gbc.gridwidth = 2;
currentPanel.add(loginButton, gbc);

JButton registerButton = new JButton("Register New User");
gbc.gridx = 0;
gbc.gridy = 4;
currentPanel.add(registerButton, gbc);

JButton forgotPasswordButton = new JButton("Forgot Password?");
gbc.gridx = 0;
gbc.gridy = 5;
currentPanel.add(forgotPasswordButton, gbc);

JButton themeToggleButton = new JButton("Toggle Dark Theme");
gbc.gridx = 0;
gbc.gridy = 6;
currentPanel.add(themeToggleButton, gbc);
```

```
    loginButton.addActionListener(e -> {
        try {
            String email = emailField.getText();
            String password = new String(passwordField.getPassword());
            currentUser = User.login(email, password);
            if (currentUser != null) {
                userInfoLabel.setText("Welcome, " + currentUser.getName());
                showMainMenu();
            } else {
                JOptionPane.showMessageDialog(this, "Invalid email or
password", "Login Failed", JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Database error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    registerButton.addActionListener(e -> showRegistrationPanel());
    forgotPasswordButton.addActionListener(e ->
showForgotPasswordPanel());
    themeToggleButton.addActionListener(e -> {
        darkTheme = !darkTheme;
        applyTheme();
    });

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}

private void showRegistrationPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("New User Registration",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    String[] securityQuestions = {
        "What was your first pet's name?",
        "What was the name of your first school?",
        "What is your mother's maiden name?",
        "What city were you born in?",
        "What was your favorite food as a child?"
    };

    JLabel nameLabel = new JLabel("Full Name:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(nameLabel, gbc);

    JTextField nameField = new JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(nameField, gbc);

    JLabel mobileLabel = new JLabel("Mobile Number:");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(mobileLabel, gbc);
```

```
JTextField mobileField = new JTextField(20);
gbc.gridx = 1;
gbc.gridy = 2;
currentPanel.add(mobileField, gbc);

JLabel emailLabel = new JLabel("Email:");
gbc.gridx = 0;
gbc.gridy = 3;
currentPanel.add(emailLabel, gbc);

JTextField emailField = new JTextField(20);
gbc.gridx = 1;
gbc.gridy = 3;
currentPanel.add(emailField, gbc);

JLabel passwordLabel = new JLabel("Password:");
gbc.gridx = 0;
gbc.gridy = 4;
currentPanel.add(passwordLabel, gbc);

JPasswordField passwordField = new JPasswordField(20);
gbc.gridx = 1;
gbc.gridy = 4;
currentPanel.add(passwordField, gbc);

JLabel confirmPasswordLabel = new JLabel("Confirm Password:");
gbc.gridx = 0;
gbc.gridy = 5;
currentPanel.add(confirmPasswordLabel, gbc);

JPasswordField confirmPasswordField = new JPasswordField(20);
gbc.gridx = 1;
gbc.gridy = 5;
currentPanel.add(confirmPasswordField, gbc);
```

```
    JLabel securityQuestionLabel = new JLabel("Security Question:");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(securityQuestionLabel, gbc);

    JComboBox<String> securityQuestionCombo = new
JComboBox<>(securityQuestions);
    gbc.gridx = 1;
    gbc.gridy = 6;
    currentPanel.add(securityQuestionCombo, gbc);

    JLabel securityAnswerLabel = new JLabel("Security Answer:");
    gbc.gridx = 0;
    gbc.gridy = 7;
    currentPanel.add(securityAnswerLabel, gbc);

    JTextField securityAnswerField = new JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 7;
    currentPanel.add(securityAnswerField, gbc);

    JButton registerButton = new JButton("Register");
    gbc.gridx = 0;
    gbc.gridy = 8;
    gbc.gridwidth = 2;
    currentPanel.add(registerButton, gbc);

    JButton backButton = new JButton("Back to Login");
    gbc.gridx = 0;
    gbc.gridy = 9;
    currentPanel.add(backButton, gbc);

    registerButton.addActionListener(e -> {
        String name = nameField.getText();
        String mobile = mobileField.getText();
        String email = emailField.getText();
        String password = new String(passwordField.getPassword());
        String confirmPassword = new
String(confirmPasswordField.getPassword());
        String securityQuestion = (String)
securityQuestionCombo.getSelectedItem();
        String securityAnswer = securityAnswerField.getText();

        if (!password.equals(confirmPassword)) {
            JOptionPane.showMessageDialog(this, "Passwords do not match",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        if (name.isEmpty() || mobile.isEmpty() || email.isEmpty() ||
password.isEmpty() || securityAnswer.isEmpty()) {
            JOptionPane.showMessageDialog(this, "All fields are required",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            User newUser = new User(name, mobile, email, password,
securityQuestion, securityAnswer);
            newUser.register();
            JOptionPane.showMessageDialog(this, "Registration successful!
Please login.", "Success", JOptionPane.INFORMATION_MESSAGE);
            showLoginPanel();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Registration failed: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    backButton.addActionListener(e -> showLoginPanel());
    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}

private void showForgotPasswordPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Password Recovery",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel emailLabel = new JLabel("Registered Email:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(emailLabel, gbc);

    JTextField emailField = new JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(emailField, gbc);

    JButton nextButton = new JButton("Next");
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.gridwidth = 2;
    currentPanel.add(nextButton, gbc);

    JButton backButton = new JButton("Back to Login");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(backButton, gbc);

    nextButton.addActionListener(e -> {
        String email = emailField.getText();
        try {
            String question = User.getSecurityQuestion(email);
            if (question == null) {
                JOptionPane.showMessageDialog(this, "Email not found",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            currentPanel.removeAll();
            currentPanel.setLayout(new GridBagLayout());

            JLabel questionLabel = new JLabel("Security Question: " +
question);
            gbc.gridx = 0;
            gbc.gridy = 0;
            gbc.gridwidth = 2;
            currentPanel.add(questionLabel, gbc);

            JLabel answerLabel = new JLabel("Answer:");
            gbc.gridx = 0;
            gbc.gridy = 1;
            gbc.gridwidth = 1;
            currentPanel.add(answerLabel, gbc);

            JTextField answerField = new JTextField(20);
```

```
gbc.gridx = 1;
gbc.gridy = 1;
currentPanel.add(answerField, gbc);

JLabel newPasswordLabel = new JLabel("New Password:");
gbc.gridx = 0;
gbc.gridy = 2;
currentPanel.add(newPasswordLabel, gbc);

JPasswordField newPasswordField = new JPasswordField(20);
gbc.gridx = 1;
gbc.gridy = 2;
currentPanel.add(newPasswordField, gbc);

JLabel confirmPasswordLabel = new JLabel("Confirm Password:");
gbc.gridx = 0;
gbc.gridy = 3;
currentPanel.add(confirmPasswordLabel, gbc);

JPasswordField confirmPasswordField = new JPasswordField(20);
gbc.gridx = 1;
gbc.gridy = 3;
currentPanel.add(confirmPasswordField, gbc);

JButton resetButton = new JButton("Reset Password");
gbc.gridx = 0;
gbc.gridy = 4;
gbc.gridwidth = 2;
currentPanel.add(resetButton, gbc);

JButton cancelButton = new JButton("Cancel");
gbc.gridx = 0;
gbc.gridy = 5;
currentPanel.add(cancelButton, gbc);
```

```
            resetButton.addActionListener(ev -> {
                String answer = answerField.getText();
                String newPassword = new
String(newPasswordField.getPassword());
                String confirmPassword = new
String(confirmPasswordField.getPassword());

                if (!newPassword.equals(confirmPassword)) {
                    JOptionPane.showMessageDialog(this, "Passwords do not
match", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }

                try {
                    if (User.verifySecurityAnswer(email, answer)) {
                        User.resetPassword(email, newPassword);
                        JOptionPane.showMessageDialog(this, "Password reset
successful!", "Success", JOptionPane.INFORMATION_MESSAGE);
                        showLoginPanel();
                    } else {
                        JOptionPane.showMessageDialog(this, "Incorrect security
answer", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(this, "Error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                }
            });

            cancelButton.addActionListener(ev -> showLoginPanel());

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
        }
```

```
            applyTheme();
            currentPanel.revalidate();
            currentPanel.repaint();
        });

        backButton.addActionListener(e -> showLoginPanel());

        applyTheme();
        currentPanel.revalidate();
        currentPanel.repaint();
    }

    private void showMainMenu() {
        currentPanel.removeAll();
        currentPanel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        JLabel welcomeLabel = new JLabel("Welcome, " +
    currentUser.getName(), SwingConstants.CENTER);
        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 24));
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 2;
        currentPanel.add(welcomeLabel, gbc);

        JButton bookTicketButton = new JButton("Book Ticket");
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 1;
        currentPanel.add(bookTicketButton, gbc);

        JButton bookingHistoryButton = new JButton("Booking History");
```

```
gbc.gridx = 1;
gbc.gridy = 1;
currentPanel.add(bookingHistoryButton, gbc);

JButton cancelTicketButton = new JButton("Cancel Ticket");
gbc.gridx = 0;
gbc.gridy = 2;
currentPanel.add(cancelTicketButton, gbc);

JButton pnrStatusButton = new JButton("PNR Status");
gbc.gridx = 1;
gbc.gridy = 2;
currentPanel.add(pnrStatusButton, gbc);

JButton manageProfileButton = new JButton("Manage Profile");
gbc.gridx = 0;
gbc.gridy = 3;
currentPanel.add(manageProfileButton, gbc);

JButton accountBalanceButton = new JButton("Account Balance");
gbc.gridx = 1;
gbc.gridy = 3;
currentPanel.add(accountBalanceButton, gbc);

JButton logoutButton = new JButton("Logout");
gbc.gridx = 0;
gbc.gridy = 4;
gbc.gridwidth = 2;
currentPanel.add(logoutButton, gbc);

bookTicketButton.addActionListener(e -> showTrainSearchPanel());
bookingHistoryButton.addActionListener(e -> showBookingHistory());
cancelTicketButton.addActionListener(e -> showCancelTicketPanel());
pnrStatusButton.addActionListener(e -> showPNRStatusPanel());
manageProfileButton.addActionListener(e -> showProfilePanel());
```

```
        accountBalanceButton.addActionListener(e -> showAccountBalance());
        logoutButton.addActionListener(e -> {
            currentUser.logout();
            currentUser = null;
            userInfoLabel.setText("Welcome, Guest");
            showLoginPanel();
        });

        applyTheme();
        currentPanel.revalidate();
        currentPanel.repaint();
    }

    private void showTrainSearchPanel() {
        currentPanel.removeAll();
        currentPanel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        JLabel titleLabel = new JLabel("Search Trains",
    SwingConstants.CENTER);
        titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 4;
        currentPanel.add(titleLabel, gbc);

        JLabel fromLabel = new JLabel("From:");
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 1;
        currentPanel.add(fromLabel, gbc);

        JTextField fromField = new JTextField(15);
        gbc.gridx = 1;
        gbc.gridy = 1;
        currentPanel.add(fromField, gbc);

        JLabel toLabel = new JLabel("To:");
        gbc.gridx = 2;
        gbc.gridy = 1;
        currentPanel.add(toLabel, gbc);

        JTextField toField = new JTextField(15);
        gbc.gridx = 3;
        gbc.gridy = 1;
        currentPanel.add(toField, gbc);

        JLabel dateLabel = new JLabel("Journey Date (dd/mm/yyyy):");
        gbc.gridx = 0;
        gbc.gridy = 2;
        currentPanel.add(dateLabel, gbc);

        JTextField dateField = new JTextField(15);
        gbc.gridx = 1;
        gbc.gridy = 2;
        currentPanel.add(dateField, gbc);

        JLabel classLabel = new JLabel("Class:");
        gbc.gridx = 2;
        gbc.gridy = 2;
        currentPanel.add(classLabel, gbc);

        JComboBox<String> classCombo = new
    JComboBox<>(Booking.getClassTypes());
        gbc.gridx = 3;
        gbc.gridy = 2;
        currentPanel.add(classCombo, gbc);
        JLabel trainTypeLabel = new JLabel("Train Type:");
        gbc.gridx = 0;
        gbc.gridy = 3;
        currentPanel.add(trainTypeLabel, gbc);

        String[] trainTypes = {"All", "Express", "Rajdhani", "Shatabdi", "Duronto",
    "Jan Shatabdi"};
        JComboBox<String> trainTypeCombo = new JComboBox<>(trainTypes);
        gbc.gridx = 1;
        gbc.gridy = 3;
        currentPanel.add(trainTypeCombo, gbc);

        JLabel timeRangeLabel = new JLabel("Departure Time:");
        gbc.gridx = 2;
        gbc.gridy = 3;
        currentPanel.add(timeRangeLabel, gbc);

        String[] timeRanges = {"All", "00:00-06:00", "06:00-12:00", "12:00-18:00",
    "18:00-24:00"};
        JComboBox<String> timeRangeCombo = new
    JComboBox<>(timeRanges);
        gbc.gridx = 3;
        gbc.gridy = 3;
        currentPanel.add(timeRangeCombo, gbc);

        JButton searchButton = new JButton("Search Trains");
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.gridwidth = 4;
        currentPanel.add(searchButton, gbc);

        JButton backButton = new JButton("Back to Main Menu");
        gbc.gridx = 0;
        gbc.gridy = 5;
        currentPanel.add(backButton, gbc);
        outputArea = new JTextArea(15, 60);
        outputArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(outputArea);
        gbc.gridx = 0;
        gbc.gridy = 6;
        gbc.gridwidth = 4;
        gbc.fill = GridBagConstraints.BOTH;
        currentPanel.add(scrollPane, gbc);

        searchButton.addActionListener(e -> {
            String from = fromField.getText();
            String to = toField.getText();
            String date = dateField.getText();
            String classType = (String) classCombo.getSelectedItem();
            String trainType = (String) trainTypeCombo.getSelectedItem();
            String timeRange = (String) timeRangeCombo.getSelectedItem();

            try {
                List<Train> trains = Train.searchTrainsWithFilters(from, to, date,
    classType, trainType, timeRange);
                displayTrainResults(trains, classType);
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(this, "Error searching trains: " +
    ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

        backButton.addActionListener(e -> showMainMenu());

        applyTheme();
        currentPanel.revalidate();
        currentPanel.repaint();
    }
```

```
private void displayTrainResults(List<Train> trains, String classType) {
    outputArea.setText("");
    if (trains.isEmpty()) {
        outputArea.append("No trains found for the given criteria.\n");
        return;
    }

    outputArea.append(String.format("%-10s %-30s %-20s %-20s %-10s
%-10s %-10s %-15s\n",
            "Train No", "Train Name", "Departure", "Arrival", "Duration", "From",
"To", "Availability"));

outputArea.append("----------------------------------------------------------------------
------------------------------------------\n");

    for (Train train : trains) {
        String availability = train.getAvailabilityForClass(classType);
        outputArea.append(String.format("%-10s %-30s %-20s %-20s %-10s
%-10s %-10s %-15s\n",
                train.getTrainNumber(),
                train.getTrainName(),
                train.getSourceStation() + " (" + train.getDepartureTime() + ")",
                train.getDestinationStation() + " (" + train.getArrivalTime() + ")",
                train.getDuration(),
                train.getSourceStation(),
                train.getDestinationStation(),
                availability));
    }

    if (trains.size() > 0) {
        JButton bookButton = new JButton("Book Selected Train");
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 7;
        gbc.gridwidth = 4;
        gbc.insets = new Insets(10, 10, 10, 10);
        currentPanel.add(bookButton, gbc);

        bookButton.addActionListener(e -> {
            String selectedTrain = JOptionPane.showInputDialog(this, "Enter
Train Number to book:");
            if (selectedTrain != null && !selectedTrain.isEmpty()) {
                for (Train train : trains) {
                    if (train.getTrainNumber().equals(selectedTrain)) {
                        showPassengerDetailsPanel(train);
                        break;
                    }
                }
            }
        });

        currentPanel.revalidate();
        currentPanel.repaint();
    }
}

private void showPassengerDetailsPanel(Train train) {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Passenger Details for " +
train.getTrainName(), SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 4;
    currentPanel.add(titleLabel, gbc);
    JLabel journeyLabel = new JLabel(train.getSourceStation() + " to " +
train.getDestinationStation());
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 4;
    currentPanel.add(journeyLabel, gbc);

    JLabel dateLabel = new JLabel("Journey Date (dd/mm/yyyy):");
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.gridwidth = 1;
    currentPanel.add(dateLabel, gbc);

    JTextField dateField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(dateField, gbc);

    JLabel classLabel = new JLabel("Class:");
    gbc.gridx = 2;
    gbc.gridy = 2;
    currentPanel.add(classLabel, gbc);

    JComboBox<String> classCombo = new
JComboBox<>(Booking.getClassTypes());
    gbc.gridx = 3;
    gbc.gridy = 2;
    currentPanel.add(classCombo, gbc);

    JLabel quotaLabel = new JLabel("Quota:");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(quotaLabel, gbc);
    JComboBox<String> quotaCombo = new
JComboBox<>(Booking.getQuotaTypes());
    gbc.gridx = 1;
    gbc.gridy = 3;
    currentPanel.add(quotaCombo, gbc);

    JLabel passengersLabel = new JLabel("Number of Passengers:");
    gbc.gridx = 2;
    gbc.gridy = 3;
    currentPanel.add(passengersLabel, gbc);

    SpinnerModel spinnerModel = new SpinnerNumberModel(1, 1, 6, 1);
    JSpinner passengersSpinner = new JSpinner(spinnerModel);
    gbc.gridx = 3;
    gbc.gridy = 3;
    currentPanel.add(passengersSpinner, gbc);

    JPanel passengersPanel = new JPanel();
    passengersPanel.setLayout(new BoxLayout(passengersPanel,
BoxLayout.Y_AXIS));
    JScrollPane scrollPane = new JScrollPane(passengersPanel);
    gbc.gridx = 0;
    gbc.gridy = 4;
    gbc.gridwidth = 4;
    gbc.fill = GridBagConstraints.BOTH;
    currentPanel.add(scrollPane, gbc);

    JButton addPassengersButton = new JButton("Add Passenger Details");
    gbc.gridx = 0;
    gbc.gridy = 5;
    gbc.gridwidth = 4;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    currentPanel.add(addPassengersButton, gbc);

    JButton bookButton = new JButton("Proceed to Payment");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(bookButton, gbc);

    JButton backButton = new JButton("Back to Search");
    gbc.gridx = 1;
    gbc.gridy = 6;
    currentPanel.add(backButton, gbc);

    List<JPanel> passengerForms = new ArrayList<>();

    addPassengersButton.addActionListener(e -> {
        int numPassengers = (Integer) passengersSpinner.getValue();
        passengersPanel.removeAll();
        passengerForms.clear();

        for (int i = 0; i < numPassengers; i++) {
            JPanel formPanel = new JPanel(new GridLayout(0, 2, 5, 5));
            formPanel.setBorder(BorderFactory.createTitledBorder("Passenger
" + (i + 1)));

            formPanel.add(new JLabel("Name:"));
            JTextField nameField = new JTextField();
            formPanel.add(nameField);

            formPanel.add(new JLabel("Age:"));
            JSpinner ageSpinner = new JSpinner(new
SpinnerNumberModel(18, 1, 120, 1));
            formPanel.add(ageSpinner);

            formPanel.add(new JLabel("Gender:"));
            JComboBox<String> genderCombo = new JComboBox<>(new
String[]{"Male", "Female", "Other"});
            formPanel.add(genderCombo);
            formPanel.add(new JLabel("Berth Preference:"));
            JComboBox<String> berthCombo = new JComboBox<>(new
String[]{"No Preference", "Lower", "Middle", "Upper", "Side Lower", "Side
Upper"});
            formPanel.add(berthCombo);

            formPanel.add(new JLabel("Concession:"));
            JComboBox<String> concessionCombo = new
JComboBox<>(Booking.getConcessionTypes());
            concessionCombo.insertItemAt("None", 0);
            concessionCombo.setSelectedIndex(0);
            formPanel.add(concessionCombo);

            passengersPanel.add(formPanel);
            passengerForms.add(formPanel);
        }

        passengersPanel.revalidate();
        passengersPanel.repaint();
    });

    bookButton.addActionListener(e -> {
        String journeyDate = dateField.getText();
        String classType = (String) classCombo.getSelectedItem();
        String quota = (String) quotaCombo.getSelectedItem();

        if (journeyDate.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter journey
date", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        List<Booking.Passenger> passengers = new ArrayList<>();
        for (JPanel form : passengerForms) {
            Component[] components = form.getComponents();
            String name = ((JTextField) components[1]).getText();
            int age = (Integer) ((JSpinner) components[3]).getValue();
            String gender = (String) ((JComboBox<?>)
components[5]).getSelectedItem();
            String berthPreference = (String) ((JComboBox<?>)
components[7]).getSelectedItem();
            String concessionType = (String) ((JComboBox<?>)
components[9]).getSelectedItem();

            if (name.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please enter name for
all passengers", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            if ("None".equals(concessionType)) {
                concessionType = null;
            }

            passengers.add(new Booking.Passenger(name, age, gender,
berthPreference, concessionType));
        }

        if (passengers.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please add passenger
details first", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            String pnr = Booking.bookTickets(
                currentUser.getUserId(),
                train.getTrainNumber(),
                train.getTrainName(),
                train.getSourceStation(),
                train.getDestinationStation(),
                journeyDate,
                classType,
                quota,
                passengers,
                "Account Balance"
            );

            JOptionPane.showMessageDialog(this,
                "Booking successful!\nPNR: " + pnr + "\nTotal Fare: " +
calculateTotalFare(passengers, classType, quota),
                "Success", JOptionPane.INFORMATION_MESSAGE);
            showMainMenu();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Booking failed: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
      });

      backButton.addActionListener(e -> showTrainSearchPanel());

      applyTheme();
      currentPanel.revalidate();
      currentPanel.repaint();
}

private double calculateTotalFare(List<Booking.Passenger> passengers,
String classType, String quota) {
      double total = 0.0;
      double baseFare = Booking.getClassFares().getOrDefault(classType,
0.0);

      if (quota.equals("Tatkal")) baseFare *= 1.3;
      if (quota.equals("Premium Tatkal")) baseFare *= 1.5;
```

```
    for (Booking.Passenger passenger : passengers) {
        double fare = baseFare;
        if (passenger.getConcessionType() != null &&
!passenger.getConcessionType().isEmpty()) {
            double discount = fare *
Booking.getConcessionRates().get(passenger.getConcessionType());
            fare -= discount;
        }
        total += fare;
    }

    return total;
}

private void showBookingHistory() {
    currentPanel.removeAll();
    currentPanel.setLayout(new BorderLayout());

    JLabel titleLabel = new JLabel("Your Booking History",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    currentPanel.add(titleLabel, BorderLayout.NORTH);

    outputArea = new JTextArea();
    outputArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(outputArea);
    currentPanel.add(scrollPane, BorderLayout.CENTER);

    JButton backButton = new JButton("Back to Main Menu");
    currentPanel.add(backButton, BorderLayout.SOUTH);

    try {
        ResultSet rs = Booking.getBookingHistory(currentUser.getUserId());
        outputArea.setText(String.format("%-10s %-15s %-30s %-15s %-15s
%-12s %-10s %-10s %-10s\n",
            "PNR", "Train No", "Train Name", "From", "To", "Journey Date",
"Class", "Status", "Fare"));

outputArea.append("--------------------------------------------------------------------
----------------------------------------------\n");

        while (rs.next()) {
            outputArea.append(String.format("%-10s %-15s %-30s %-15s
%-15s %-12s %-10s %-10s ₹%.2f\n",
                rs.getString("pnr_number"),
                rs.getString("train_number"),
                rs.getString("train_name"),
                rs.getString("from_station"),
                rs.getString("to_station"),
                rs.getDate("journey_date"),
                rs.getString("class_type"),
                rs.getString("booking_status"),
                rs.getDouble("fare")));
        }
    } catch (SQLException ex) {
        outputArea.setText("Error retrieving booking history: " +
ex.getMessage());
    }

    backButton.addActionListener(e -> showMainMenu());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}

private void showCancelTicketPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Cancel Ticket",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel pnrLabel = new JLabel("Enter PNR Number:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(pnrLabel, gbc);

    JTextField pnrField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(pnrField, gbc);

    JButton checkButton = new JButton("Check PNR");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(checkButton, gbc);

    JButton cancelButton = new JButton("Cancel Ticket");
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(cancelButton, gbc);

    JButton backButton = new JButton("Back to Main Menu");
    gbc.gridx = 0;
    gbc.gridy = 3;
    gbc.gridwidth = 2;
    currentPanel.add(backButton, gbc);

    outputArea = new JTextArea(5, 30);
    outputArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(outputArea);
    gbc.gridx = 0;
    gbc.gridy = 4;
    gbc.gridwidth = 2;
    gbc.fill = GridBagConstraints.BOTH;
    currentPanel.add(scrollPane, gbc);

    checkButton.addActionListener(e -> {
        String pnr = pnrField.getText().trim();
        if (pnr.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter a PNR
number", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
        try {
            ResultSet rs = Booking.getPNRStatus(pnr);
            StringBuilder status = new StringBuilder();
            boolean found = false;
            while (rs.next()) {
                found = true;
                status.append("PNR: ").append(rs.getString("pnr_number"))
                      .append("\nTrain: ").append(rs.getString("train_name"))
                      .append("
(").append(rs.getString("train_number")).append(")")
                      .append("\nFrom: ").append(rs.getString("from_station"))
                      .append("\nTo: ").append(rs.getString("to_station"))
                      .append("\nDate: ").append(rs.getDate("journey_date"))
                      .append("\nClass: ").append(rs.getString("class_type"))
                      .append("\nPassenger:
").append(rs.getString("passenger_name"))
```

```
                  .append("\nStatus: ").append(rs.getString("booking_status"))
                  .append("\nFare: ₹").append(rs.getDouble("fare"))
                  .append("\n\n");
          }
          if (!found) {
              outputArea.setText("No booking found for PNR: " + pnr);
          } else {
              outputArea.setText(status.toString());
          }
      } catch (SQLException ex) {
          outputArea.setText("Error retrieving PNR status: " +
ex.getMessage());
      }
  });

  cancelButton.addActionListener(e -> {
      String pnr = pnrField.getText().trim();
      if (pnr.isEmpty()) {
          JOptionPane.showMessageDialog(this, "Please enter a PNR
number", "Error", JOptionPane.ERROR_MESSAGE);
          return;
      }
      try {
          boolean success = Booking.cancelTicket(pnr);
          if (success) {
              outputArea.setText("Ticket with PNR " + pnr + " cancelled
successfully. Amount refunded.");
              JOptionPane.showMessageDialog(this, "Ticket cancelled
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
          } else {
              outputArea.setText("No valid ticket found for PNR: " + pnr);
              JOptionPane.showMessageDialog(this, "Cancellation failed: No
valid ticket found", "Error", JOptionPane.ERROR_MESSAGE);
          }
      } catch (SQLException ex) {
          outputArea.setText("Error cancelling ticket: " + ex.getMessage());
          JOptionPane.showMessageDialog(this, "Error cancelling ticket: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
      }
  });

  backButton.addActionListener(e -> showMainMenu());

  applyTheme();
  currentPanel.revalidate();
  currentPanel.repaint();
}

private void showPNRStatusPanel() {
  currentPanel.removeAll();
  currentPanel.setLayout(new GridBagLayout());
  GridBagConstraints gbc = new GridBagConstraints();
  gbc.insets = new Insets(10, 10, 10, 10);
  gbc.fill = GridBagConstraints.HORIZONTAL;

  JLabel titleLabel = new JLabel("PNR Status",
SwingConstants.CENTER);
  titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
  gbc.gridx = 0;
  gbc.gridy = 0;
  gbc.gridwidth = 2;
  currentPanel.add(titleLabel, gbc);

  JLabel pnrLabel = new JLabel("Enter PNR Number:");
  gbc.gridx = 0;
  gbc.gridy = 1;
  gbc.gridwidth = 1;
  currentPanel.add(pnrLabel, gbc);

  JTextField pnrField = new JTextField(15);
  gbc.gridx = 1;
  gbc.gridy = 1;
  currentPanel.add(pnrField, gbc);

  JButton checkButton = new JButton("Check Status");
  gbc.gridx = 0;
  gbc.gridy = 2;
  gbc.gridwidth = 2;
  currentPanel.add(checkButton, gbc);

  JButton backButton = new JButton("Back to Main Menu");
  gbc.gridx = 0;
  gbc.gridy = 3;
  currentPanel.add(backButton, gbc);

  outputArea = new JTextArea(10, 30);
  outputArea.setEditable(false);
  JScrollPane scrollPane = new JScrollPane(outputArea);
  gbc.gridx = 0;
  gbc.gridy = 4;
  gbc.gridwidth = 2;
  gbc.fill = GridBagConstraints.BOTH;
  currentPanel.add(scrollPane, gbc);

  checkButton.addActionListener(e -> {
      String pnr = pnrField.getText().trim();
      if (pnr.isEmpty()) {
          JOptionPane.showMessageDialog(this, "Please enter a PNR
number", "Error", JOptionPane.ERROR_MESSAGE);
          return;
      }
      try {
          ResultSet rs = Booking.getPNRStatus(pnr);
          StringBuilder status = new StringBuilder();
          boolean found = false;
            while (rs.next()) {
                found = true;
                status.append("PNR: ").append(rs.getString("pnr_number"))
                      .append("\nTrain: ").append(rs.getString("train_name"))
                      .append("
(").append(rs.getString("train_number")).append(")")
                      .append("\nFrom: ").append(rs.getString("from_station"))
                      .append("\nTo: ").append(rs.getString("to_station"))
                      .append("\nDate: ").append(rs.getDate("journey_date"))
                      .append("\nClass: ").append(rs.getString("class_type"))
                      .append("\nPassenger:
").append(rs.getString("passenger_name"))
                      .append("\nStatus: ").append(rs.getString("booking_status"))
                      .append("\nSeat: ").append(rs.getString("seat_number"))
                      .append("\nFare: ₹").append(rs.getDouble("fare"))
                      .append("\n\n");
            }
            if (!found) {
                outputArea.setText("No booking found for PNR: " + pnr);
            } else {
                outputArea.setText(status.toString());
            }
        } catch (SQLException ex) {
            outputArea.setText("Error retrieving PNR status: " +
ex.getMessage());
        }
    });

    backButton.addActionListener(e -> showMainMenu());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

```
private void showProfilePanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Manage Profile",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel ageLabel = new JLabel("Age:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(ageLabel, gbc);

    JTextField ageField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(ageField, gbc);

    JLabel genderLabel = new JLabel("Gender:");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(genderLabel, gbc);

    JComboBox<String> genderCombo = new JComboBox<>(new
String[]{"Male", "Female", "Other"});
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(genderCombo, gbc);

    JLabel classLabel = new JLabel("Preferred Class:");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(classLabel, gbc);

    JComboBox<String> classCombo = new
JComboBox<>(Booking.getClassTypes());
    gbc.gridx = 1;
    gbc.gridy = 3;
    currentPanel.add(classCombo, gbc);

    JLabel govIdLabel = new JLabel("Government ID:");
    gbc.gridx = 0;
    gbc.gridy = 4;
    currentPanel.add(govIdLabel, gbc);

    JTextField govIdField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 4;
    currentPanel.add(govIdField, gbc);

    JLabel emergencyLabel = new JLabel("Emergency Contact:");
    gbc.gridx = 0;
    gbc.gridy = 5;
    currentPanel.add(emergencyLabel, gbc);

    JTextField emergencyField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 5;
    currentPanel.add(emergencyField, gbc);

    JLabel wheelchairLabel = new JLabel("Wheelchair Access:");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(wheelchairLabel, gbc);

    JCheckBox wheelchairCheck = new JCheckBox();
    gbc.gridx = 1;
    gbc.gridy = 6;
    currentPanel.add(wheelchairCheck, gbc);

    JLabel medicalLabel = new JLabel("Medical Condition:");
    gbc.gridx = 0;
    gbc.gridy = 7;
    currentPanel.add(medicalLabel, gbc);

    JCheckBox medicalCheck = new JCheckBox();
    gbc.gridx = 1;
    gbc.gridy = 7;
    currentPanel.add(medicalCheck, gbc);

    JButton saveButton = new JButton("Save Profile");
    gbc.gridx = 0;
    gbc.gridy = 8;
    gbc.gridwidth = 2;
    currentPanel.add(saveButton, gbc);

    JButton backButton = new JButton("Back to Main Menu");
    gbc.gridx = 0;
    gbc.gridy = 9;
    currentPanel.add(backButton, gbc);

    // Load existing profile if available
    try {
        Profile profile = Profile.getProfile(currentUser.getUserId());
        if (profile != null) {
            ageField.setText(String.valueOf(profile.getAge()));
            genderCombo.setSelectedItem(profile.getGender());
```

```
            classCombo.setSelectedItem(profile.getPreferredClass());
            govIdField.setText(profile.getGovId());
            emergencyField.setText(profile.getEmergencyContact());
            wheelchairCheck.setSelected(profile.isWheelchairAccess());
            medicalCheck.setSelected(profile.isMedicalCondition());
        }
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this, "Error loading profile: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }

    saveButton.addActionListener(e -> {
        try {
            int age = Integer.parseInt(ageField.getText().trim());
            String gender = (String) genderCombo.getSelectedItem();
            String preferredClass = (String) classCombo.getSelectedItem();
            String govId = govIdField.getText().trim();
            String emergencyContact = emergencyField.getText().trim();
            boolean wheelchairAccess = wheelchairCheck.isSelected();
            boolean medicalCondition = medicalCheck.isSelected();

            if (govId.isEmpty() || emergencyContact.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please fill all required
fields", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            Profile profile = new Profile(
                currentUser.getUserId(),
                age,
                gender,
                preferredClass,
                govId,
                emergencyContact,
                wheelchairAccess,
                medicalCondition
            );
            profile.save();
            JOptionPane.showMessageDialog(this, "Profile saved
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            showMainMenu();
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Invalid age format",
"Error", JOptionPane.ERROR_MESSAGE);
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error saving profile: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    backButton.addActionListener(e -> showMainMenu());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}

private void showAccountBalance() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Account Balance",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel balanceLabel = new JLabel("Current Balance: ");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(balanceLabel, gbc);

    JTextField balanceField = new JTextField(15);
    balanceField.setEditable(false);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(balanceField, gbc);

    JButton backButton = new JButton("Back to Main Menu");
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.gridwidth = 2;
    currentPanel.add(backButton, gbc);

    try {
        double balance =
DBConnection.getAccountBalance(currentUser.getUserId());
        balanceField.setText(String.format("₹%.2f", balance));
    } catch (SQLException ex) {
        balanceField.setText("Error retrieving balance");
        JOptionPane.showMessageDialog(this, "Error retrieving balance: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }

    backButton.addActionListener(e -> showMainMenu());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
```

```
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new TrainManagementSystem());
}
}
```

1.2.About Code
import

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.text.*;
import java.util.*;
import java.util.List;
```

- Purpose: These imports bring in the necessary libraries for the application.
- Details:
    - javax.swing.*: Provides Swing components (e.g., JFrame, JButton, JTextField) for building the GUI.
    - java.awt.* and java.awt.event.*: Used for GUI layout (GridBagLayout, BorderLayout) and event handling (ActionListener).
    - java.sql.*: Enables database connectivity and operations via JDBC (Connection, PreparedStatement).
    - java.text.*: Used for date formatting (SimpleDateFormat).
    - java.util.* and java.util.List: Provides utility classes like ArrayList, HashMap, and Random.

Class: DBConnection

This class handles all database-related operations, including connection management, table creation, and data manipulation.

Fields

java

Copy

```
private static final String URL =
"jdbc:mysql://localhost:3306/IRCTC_System";
private static final String USER = "root";
private static final String PASSWORD = "root";
```

- Purpose: These are constants for database connection details.
- Why Used:
  - URL: Specifies the database location and name (IRCTC_System on localhost at port 3306).
  - USER and PASSWORD: Credentials for MySQL authentication (root/root).
  - Declared as private static final to ensure they are immutable and shared across all instances, promoting encapsulation and security.

Method: getConnection()

java

Copy

```
public static Connection getConnection() throws
SQLException {
    return DriverManager.getConnection(URL, USER,
PASSWORD);
}
```

- Purpose: Establishes a connection to the MySQL database.
- Why Used:

  - Provides a reusable method to get a database connection using JDBC’s DriverManager.
  - Throws SQLException to handle connection errors (e.g., wrong credentials, database not running).
  - static so it can be called without creating an instance of DBConnection, making it a utility method.

Method: initializeDatabase()

java

Copy

```
public static void initializeDatabase() {
    try (Connection conn = getConnection()) {
        String createUsersTable = "CREATE TABLE IF NOT
EXISTS Users (...)";
        String createTrainsTable = "CREATE TABLE IF NOT
EXISTS Trains (...)";
        String createBookingsTable = "CREATE TABLE IF
NOT EXISTS Bookings (...)";
        String createAccountHolderTable = "CREATE TABLE
IF NOT EXISTS AccountHolder (...)";
        String createProfilesTable = "CREATE TABLE IF
NOT EXISTS Profiles (...)";
        String createSessionLogsTable = "CREATE TABLE
IF NOT EXISTS SessionLogs (...)";

        Statement stmt = conn.createStatement();
        stmt.execute(createUsersTable);
        stmt.execute(createTrainsTable);
        stmt.execute(createBookingsTable);
        stmt.execute(createAccountHolderTable);
        stmt.execute(createProfilesTable);
        stmt.execute(createSessionLogsTable);
        String checkTrains = "SELECT COUNT(*) FROM
Trains";
        ResultSet rs = stmt.executeQuery(checkTrains);
        rs.next();
        if (rs.getInt(1) == 0) {
            insertSampleTrainData(conn);
        }

        String checkAccounts = "SELECT COUNT(*) FROM
AccountHolder";
        rs = stmt.executeQuery(checkAccounts);
        rs.next();
        if (rs.getInt(1) == 0) {
            insertSampleAccountHolderData(conn);
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Database
initialization failed: " + e.getMessage(),
                                    "Database Error",
JOptionPane.ERROR_MESSAGE);
    }
}
```

- Purpose: Initializes the database by creating necessary tables and populating them with sample data if empty.
- Why Used:
    - Table Creation: Creates six tables (Users, Trains, Bookings, AccountHolder, Profiles, SessionLogs) using SQL CREATE

TABLE statements. The IF NOT EXISTS clause prevents errors if tables already exist.

- Foreign Keys: Ensures referential integrity (e.g., FOREIGN KEY (user_id) REFERENCES Users(user_id) in Bookings).
- Sample Data: Checks if Trains and AccountHolder tables are empty, and if so, calls insertSampleTrainData() and insertSampleAccountHolderData() to populate them.
- Error Handling: Uses a try-catch block to handle SQLException and displays an error message via JOptionPane.
- Called during application startup (in TrainManagementSystem constructor) to ensure the database is ready.

Method: insertSampleTrainData(Connection conn)

java

Copy

```
private static void insertSampleTrainData(Connection
conn) throws SQLException {
    String sql = "INSERT INTO Trains (...) VALUES (?,
?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement stmt =
conn.prepareStatement(sql);

    Object[][] trains = {
        {"12001", "Shatabdi Express", "New Delhi",
"Bhopal", "08:10", "16:25", "08:15", "Daily"},
        // ... more trains ...
    };

    for (Object[] train : trains) {
        for (int i = 0; i < train.length; i++) {
            stmt.setString(i + 1, (String) train[i]);
        }
        stmt.addBatch();
    }
    stmt.executeBatch();
}
```

- Purpose: Inserts sample train data into the Trains table.
- Why Used:
    - Uses a PreparedStatement to safely insert data, preventing SQL injection.
    - Batch execution (addBatch(), executeBatch()) improves performance by sending multiple inserts in one go.
    - Provides initial data for testing the train search feature.

Method: insertSampleAccountHolderData(Connection conn)
java
Copy

```
private static void
insertSampleAccountHolderData(Connection conn) throws
SQLException {
    String sql = "INSERT INTO AccountHolder (...)
VALUES (?, ?, ?, ?, ?, ?)";
    PreparedStatement stmt =
conn.prepareStatement(sql);

    Object[][] accounts = {
        {"John Doe", 30, "ACC1001", 50000.00, 3.5,
null},
        {"Jane Smith", 25, "ACC1002", 75000.00, 3.5,
null}
    };

    for (Object[] account : accounts) {
        stmt.setString(1, (String) account[0]);
        stmt.setInt(2, (Integer) account[1]);
        stmt.setString(3, (String) account[2]);
        stmt.setDouble(4, (Double) account[3]);
        stmt.setDouble(5, (Double) account[4]);
        stmt.setObject(6, account[5], Types.INTEGER);
        stmt.addBatch();
    }
    stmt.executeBatch();
}
```

- Purpose: Inserts sample account holder data into the AccountHolder table.
- Why Used:
    - Similar to insertSampleTrainData(), uses a PreparedStatement with batch execution for efficiency.
    - Provides initial account data with balances for testing payment and refund functionality.
    - user_id is set to null for these sample accounts, as they are not linked to users initially.

Method: logSession(int userId, String action, String sessionId)

java

Copy

```
public static void logSession(int userId, String
action, String sessionId) throws SQLException {
    try (Connection conn = getConnection()) {
        String sql = "INSERT INTO SessionLogs (user_id,
action, session_id) VALUES (?, ?, ?)";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setInt(1, userId);
        stmt.setString(2, action);
        stmt.setString(3, sessionId);
        stmt.executeUpdate();
    }
}
```

- Purpose: Logs user actions (e.g., login, booking) in the SessionLogs table.
- Why Used:
  - Tracks user activities for auditing and debugging purposes.
  - Used throughout the application (e.g., during login, booking, cancellation) to maintain a record of events.
  - Uses a PreparedStatement to safely insert data.

Method: getAccountBalance(int userId)

```
public static double getAccountBalance(int userId)
throws SQLException {
    try (Connection conn = getConnection()) {
        String sql = "SELECT balance FROM AccountHolder
WHERE user_id = ?";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setInt(1, userId);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getDouble("balance");
        }
        return 0.0;
    }
}
```

- Purpose: Retrieves the account balance for a given user.

- Why Used:
  - Used to display the user’s balance in the “Account Balance” panel.
  - Returns 0.0 if no account is found, ensuring the application doesn’t crash.
  - Uses a PreparedStatement for secure querying.

Method: deductBalance(int userId, double amount)

java

Copy

```
public static boolean deductBalance(int userId, double
amount) throws SQLException {
    try (Connection conn = getConnection()) {
        conn.setAutoCommit(false);
        try {
            String checkSql = "SELECT balance FROM
AccountHolder WHERE user_id = ?";
            PreparedStatement checkStmt =
conn.prepareStatement(checkSql);
            checkStmt.setInt(1, userId);
            ResultSet rs = checkStmt.executeQuery();

            if (rs.next()) {
                double currentBalance =
rs.getDouble("balance");
                if (currentBalance >= amount) {
                    String updateSql = "UPDATE
AccountHolder SET balance = balance - ? WHERE user_id =
?";
                    PreparedStatement updateStmt =
conn.prepareStatement(updateSql);
                    updateStmt.setDouble(1, amount);
                    updateStmt.setInt(2, userId);
                    updateStmt.executeUpdate();
                    conn.commit();
                    return true;
                }
            }
            conn.rollback();
            return false;
        } catch (SQLException e) {
            conn.rollback();
            throw e;
        } finally {
            conn.setAutoCommit(true);
        }
    }
}
```

- Purpose: Deducts a specified amount from the user's balance during booking.
- Why Used:
  - Ensures transactional integrity using setAutoCommit(false), commit(), and rollback().
  - Checks if the user has sufficient balance before deducting.
  - Returns true if the deduction is successful, false otherwise (e.g., insufficient balance).
  - Used during ticket booking to process payments.

Method: refundBalance(int userId, double amount)

```
public static void refundBalance(int userId, double
amount) throws SQLException {
    try (Connection conn = getConnection()) {
        String sql = "UPDATE AccountHolder SET balance
= balance + ? WHERE user_id = ?";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setDouble(1, amount);
        stmt.setInt(2, userId);
        stmt.executeUpdate();
    }
}
```

- Purpose: Refunds a specified amount to the user's balance.
- Why Used:
    - Used during ticket cancellation or if booking fails after payment.
    - Updates the balance by adding the refund amount using a PreparedStatement.

Class: User

This class manages user-related operations such as registration, login, logout, and password recovery.

```
private int userId;
private String name;
private String mobile;
private String email;
private String password;
private String securityQuestion;
private String securityAnswer;
private String sessionId;
```

- Purpose: Stores user data.
- Why Used:

    - Encapsulates user information (private fields) to ensure data integrity.
    - sessionId tracks the user's session for logging purposes.

Constructors

java

Copy

```
public User(String name, String mobile, String email,
String password,
            String securityQuestion, String
securityAnswer) {
    this.name = name;
    this.mobile = mobile;
    this.email = email;
    this.password = password;
    this.securityQuestion = securityQuestion;
    this.securityAnswer = securityAnswer;
    this.sessionId = UUID.randomUUID().toString();
}

private User(int userId, String name, String mobile,
String email, String password,
             String securityQuestion, String
securityAnswer, String sessionId) {
    this.userId = userId;
    this.name = name;
    this.mobile = mobile;
    this.email = email;
    this.password = password;
    this.securityQuestion = securityQuestion;
    this.securityAnswer = securityAnswer;
    this.sessionId = sessionId;
}
```

- Purpose: Initializes User objects.
- Why Used:
    - First constructor: Used during registration to create a new user with a random sessionId.
    - Second constructor (private): Used internally by login() to create a User object from database data, including the userId and sessionId.
    - Overloaded constructors allow flexibility in object creation.

Method: register()

java

Copy

```
public void register() throws SQLException {
    try (Connection conn =
DBConnection.getConnection()) {
        String sql = "INSERT INTO Users (...) VALUES
(?, ?, ?, ?, ?, ?)";
        PreparedStatement stmt =
conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS);
        stmt.setString(1, name);
        // ... set other fields ...
        stmt.executeUpdate();

        ResultSet rs = stmt.getGeneratedKeys();
        if (rs.next()) {
            this.userId = rs.getInt(1);
        }
        DBConnection.logSession(userId, "register",
sessionId);

        String accountSql = "INSERT INTO AccountHolder
(...) VALUES (?, ?, ?, ?, ?, ?)";
        PreparedStatement accountStmt =
conn.prepareStatement(accountSql);
        accountStmt.setString(1, name);
        accountStmt.setInt(2, 0);
        accountStmt.setString(3, "ACC" + (1000 + new
Random().nextInt(9000)));
        accountStmt.setDouble(4, 10000.0);
        accountStmt.setDouble(5, 3.5);
        accountStmt.setInt(6, userId);
        accountStmt.executeUpdate();
    }
}
```

- Purpose: Registers a new user in the database and creates an associated account.
- Why Used:
    - Inserts user details into the Users table and retrieves the generated userId.
    - Logs the registration action in SessionLogs.
    - Creates an AccountHolder entry with an initial balance of 10,000 for the user.
    - Used in the registration panel to add new users.

Method: login(String email, String password)

java

Copy

```
public static User login(String email, String password)
throws SQLException {
    try (Connection conn =
DBConnection.getConnection()) {
        String sql = "SELECT * FROM Users WHERE email =
? AND password = ?";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setString(1, email);
        stmt.setString(2, password);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            String sessionId =
UUID.randomUUID().toString();
            User user = new User(
                rs.getInt("user_id"),
                rs.getString("name"),
                // ... other fields ...
                sessionId
            );
            DBConnection.logSession(user.userId,
"login", sessionId);
            return user;
        }
        return null;
    }
}
```

- Purpose: Authenticates a user and returns a User object if successful.

- Why Used:
  - Queries the Users table to verify the email and password.
  - Creates a new User object with a unique sessionId if authentication succeeds.
  - Logs the login action in SessionLogs.
  - Used in the login panel to authenticate users.

Method: hasCompleteProfile()

java

Copy

```
public boolean hasCompleteProfile() throws SQLException
{
    try (Connection conn =
DBConnection.getConnection()) {
        String sql = "SELECT COUNT(*) FROM Profiles
WHERE user_id = ?";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setInt(1, userId);
        ResultSet rs = stmt.executeQuery();
        rs.next();
        return rs.getInt(1) > 0;
    }
}
```

- Purpose: Checks if the user has a saved profile.
- Why Used:
  - Queries the Profiles table to see if a profile exists for the user.
  - Returns true if a profile is found, false otherwise.
  - Not explicitly used in the provided code but could be used to prompt users to complete their profile.

Method: logout

```
public void logout() {
    try {
        DBConnection.logSession(userId, "logout",
sessionId);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    sessionId = null;
}
```

- Purpose: Logs out the user.
- Why Used:
    - Logs the logout action in SessionLogs.
    - Clears the sessionId to end the session.
    - Used in the main menu to log out the user and return to the login panel.

Methods for Password Recovery

java

Copy

```
public static String getSecurityQuestion(String email)
throws SQLException { ... }
public static boolean verifySecurityAnswer(String
email, String answer) throws SQLException { ... }
public static void resetPassword(String email, String
newPassword) throws SQLException { ... }
```

- Purpose: Support password recovery functionality.
- Why Used:
    - getSecurityQuestion(): Retrieves the security question for a given email.

    - verifySecurityAnswer(): Verifies the user’s answer to the security question.
    - resetPassword(): Updates the user’s password in the database.
    - Used in the forgot password panel to allow users to reset their passwords.

Getters
java
Copy
```
public int getUserId() { return userId; }
public String getName() { return name; }
public String getSessionId() { return sessionId; }
```

- Purpose: Provide access to private fields.
- Why Used:
    - Encapsulation principle: Fields are private, and getters allow controlled access.
    - Used throughout the application (e.g., to display the user’s name in the header).

Class: Profile

This class manages user profiles, including saving and retrieving profile data.

Fields
java
Copy
```
private int userId;
private int age;
private String gender;
private String preferredClass;
private String govId;
private String emergencyContact;
private boolean wheelchairAccess;
private boolean medicalCondition;
```

- Purpose: Stores profile information for a user.
- Why Used:
  - Encapsulates profile data as private fields.
  - Used to save user preferences and accessibility needs.

Constructor

java

Copy

```
public Profile(int userId, int age, String gender,
String preferredClass, String govId,
               String emergencyContact, boolean
wheelchairAccess, boolean medicalCondition) {
    this.userId = userId;
    this.age = age;
    this.gender = gender;
    this.preferredClass = preferredClass;
    this.govId = govId;
    this.emergencyContact = emergencyContact;
    this.wheelchairAccess = wheelchairAccess;
    this.medicalCondition = medicalCondition;
}
```

- Purpose: Initializes a Profile object.
- Why Used:
  - Sets all fields during object creation.
  - Used when saving a new profile or loading an existing one.

Method: save()

java

Copy
```
public void save() throws SQLException {
    try (Connection conn =
DBConnection.getConnection()) {
        String sql = "INSERT INTO Profiles (...) VALUES
(?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        stmt.setInt(1, userId);
        // ... set other fields ...
        stmt.executeUpdate();
    }
}
```

- Purpose: Saves the profile to the Profiles table.
- Why Used:
  - Inserts profile data into the database using a PreparedStatement.
  - Used in the profile management panel to save user preferences.

Method: getProfile(int userId)
java
Copy
```
public static Profile getProfile(int userId) throws
SQLException {
    try (Connection conn =
DBConnection.getConnection()) {
        String sql = "SELECT * FROM Profiles WHERE
user_id = ?";
        PreparedStatement stmt =
conn.prepareStatement(sql);
```

```
        stmt.setInt(1, userId);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return new Profile(
                userId,
                rs.getInt("age"),
                // ... other fields ...
            );
        }
        return null;
    }
}
```

- Purpose: Retrieves a user's profile from the database.
- Why Used:
  - Queries the Profiles table to load existing profile data.
  - Returns null if no profile is found.
  - Used in the profile panel to prefill fields with existing data.

Getters

java

Copy

```
public int getAge() { return age; }
public String getGender() { return gender; }
// ... other getters ...
```

- Purpose: Provide access to profile fields.
- Why Used:
  - Encapsulation: Allows controlled access to private fields.
  - Used in the profile panel to display existing profile data.

Class: Train

This class represents a train and handles train search functionality.

```
private String trainNumber;
private String trainName;
private String sourceStation;
private String destinationStation;
private String departureTime;
private String arrivalTime;
private String duration;
private String runsOn;
```

- Purpose: Stores train details.
- Why Used:
  - Encapsulates train data as private fields.
  - Used to represent trains in search results and during booking.

Constructor

```
public Train(String trainNumber, String trainName,
String sourceStation, String destinationStation,
             String departureTime, String arrivalTime,
String duration, String runsOn) {
    this.trainNumber = trainNumber;
    this.trainName = trainName;
    this.sourceStation = sourceStation;
    this.destinationStation = destinationStation;
    this.departureTime = departureTime;
    this.arrivalTime = arrivalTime;
    this.duration = duration;
    this.runsOn = runsOn;
}
```

- Purpose: Initializes a Train object.

- Why Used:
    - Sets all fields during object creation.
    - Used when creating Train objects from database query results.

Method: searchTrainsWithFilters()

```
public static List<Train>
searchTrainsWithFilters(String from, String to, String
journeyDate,
                                                  String
classType, String trainType, String timeRange) throws
SQLException {
    List<Train> trains = new ArrayList<>();
    try (Connection conn =
DBConnection.getConnection()) {
        StringBuilder sql = new StringBuilder(
            "SELECT * FROM Trains WHERE source_station
LIKE ? AND destination_station LIKE ?"
        );

        if (!trainType.equals("All")) {
            sql.append(" AND train_name LIKE ?");
        }

        PreparedStatement stmt =
conn.prepareStatement(sql.toString());
        stmt.setString(1, "%" + from + "%");
        stmt.setString(2, "%" + to + "%");

        int paramIndex = 3;
        if (!trainType.equals("All")) {
            stmt.setString(paramIndex++, "%" +
trainType + "%");
        }

        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Train train = new Train(
                rs.getString("train_number"),
                // ... other fields ...
            );

            SimpleDateFormat sdf = new
SimpleDateFormat("dd/MM/yyyy");
            try {
                java.util.Date date =
sdf.parse(journeyDate);
                String[] days =
train.runsOn.split(",");
                String journeyDay = new
SimpleDateFormat("EEE").format(date).toLowerCase();
                boolean runsOnDay = false;
                for (String day : days) {
                    if
(day.trim().toLowerCase().startsWith(journeyDay)) {
                        runsOnDay = true;
                        break;
                    }
                }
                if (runsOnDay) {
                    if (timeRange != null &&
!timeRange.equals("All")) {
                        String[] times =
timeRange.split("-");
                        int startHour =
Integer.parseInt(times[0].split(":")[0]);
                        int endHour =
Integer.parseInt(times[1].split(":")[0]);
                        int departureHour =
Integer.parseInt(train.departureTime.split(":")[0]);
                        if (departureHour >= startHour
&& departureHour <= endHour) {
                            trains.add(train);
                        }
                    } else {
                        trains.add(train);
                    }
                }
            } catch (ParseException e) {
                throw new SQLException("Invalid date
format: " + journeyDate, e);
            }
        }
    }
    return trains;
}
```

- Purpose: Searches for trains based on user criteria (source, destination, date, etc.).
- Why Used:

    - Builds a dynamic SQL query using StringBuilder to filter trains by source and destination.
    - Optionally filters by trainType and timeRange.
    - Checks if the train runs on the journey date by parsing the date and comparing it with the runsOn field.
    - Returns a List<Train> for display in the train search panel.

Methods: isClassAvailable() and getAvailabilityForClass()

```
public boolean isClassAvailable(String classType) {
    return
getAvailabilityForClass(classType).equals("Available");
}

public String getAvailabilityForClass(String classType)
{
    Random random = new Random();
    int availability = random.nextInt(100);

    if (availability > 80) {
        return "WL " + (random.nextInt(50) + 1);
    } else if (availability > 60) {
        return "RAC " + (random.nextInt(20) + 1);
    } else {
        return "Available";
    }
}
```

- Purpose: Simulate seat availability for a given class.
- Why Used:
    - getAvailabilityForClass(): Randomly determines availability (Available, WL (Waiting List), or RAC (Reservation Against Cancellation)).

    - isClassAvailable(): Returns true if seats are available.
    - Used during booking to determine the booking status and seat number.

Getter

```
public String getTrainNumber() { return trainNumber; }
public String getTrainName() { return trainName; }
// ... other getters ...
```

- Purpose: Provide access to train fields.
- Why Used:
    - Encapsulation: Allows controlled access to private fields.
    - Used to display train details in the GUI.

Class: Booking

This class handles ticket booking, cancellation, and PNR status checking.

Fields

java

Copy

```
private static final String[] CLASS_TYPES = {
    "Anubhuti Class (EA)", "AC First Class (1A)",
"Vistadome AC (EV)",
    // ... other classes ...
};

private static final String[] QUOTA_TYPES = {"General",
"Tatkal", "Premium Tatkal", "Ladies", "Senior
Citizen"};
```

```
private static final Map<String, Double> CLASS_FARES =
new HashMap<>();
private static final Map<String, Double>
CONCESSION_RATES = new HashMap<>();

static {
    CLASS_FARES.put("Anubhuti Class (EA)", 4500.0);
    // ... other fares ...

    CONCESSION_RATES.put("Senior Citizen", 0.5);
    // ... other concessions ...
}
```

- Purpose: Define constants for class types, quotas, fares, and concession rates.
- Why Used:
    - CLASS_TYPES and QUOTA_TYPES: Used in dropdown menus for user selection.
    - CLASS_FARES: Stores base fares for each class.
    - CONCESSION_RATES: Stores discount rates for concessions (e.g., 50% for senior citizens).
    - static block initializes these maps at class loading time.

Getters for Constants

```
public static Map<String, Double> getClassFares() {
return CLASS_FARES; }
public static Map<String, Double> getConcessionRates()
{ return CONCESSION_RATES; }
public static String[] getClassTypes() { return
CLASS_TYPES; }
public static String[] getQuotaTypes() { return
QUOTA_TYPES; }
```

```
public static String[] getConcessionTypes() { return
CONCESSION_RATES.keySet().toArray(new String[0]); }
```

- Purpose: Provide access to the constant data.
- Why Used:
    - Used in the GUI to populate dropdowns and calculate fares during booking.

Method: bookTickets()

```
public static String bookTickets(int userId, String
trainNumber, String trainName,
                               String fromStation,
String toStation, String journeyDate,
                               String classType, String
quota, List<Passenger> passengers,
                               String paymentMethod)
throws SQLException {
    Connection conn = DBConnection.getConnection();

    String pnr = generatePNR();
    double totalFare = 0.0;
    Random random = new Random();

    double baseFare =
getClassFares().getOrDefault(classType, 0.0);
    if (quota.equals("Tatkal")) baseFare *= 1.3;
    if (quota.equals("Premium Tatkal")) baseFare *=
1.5;

    for (Passenger passenger : passengers) {
        double fare = baseFare;
        if (passenger.getConcessionType() != null &&
!passenger.getConcessionType().isEmpty() &&

getConcessionRates().containsKey(passenger.getConcessio
nType())) {
            double discount = fare *
getConcessionRates().get(passenger.getConcessionType())
;
            fare -= discount;
        }
        totalFare += fare;
    }

    boolean paymentSuccess =
DBConnection.deductBalance(userId, totalFare);
    if (!paymentSuccess) {
        throw new SQLException("Insufficient balance to
complete the booking.");
    }

    try {
        String sql = "INSERT INTO Bookings (...) VALUES
(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement stmt =
conn.prepareStatement(sql);
        for (Passenger passenger : passengers) {
            double fare = baseFare;
            if (passenger.getConcessionType() != null
&& !passenger.getConcessionType().isEmpty()) {
```

```
                double discount = fare *
getConcessionRates().get(passenger.getConcessionType())
;
                fare -= discount;
            }

            String bookingStatus = "Confirmed";
            String seatNumber = String.format("%03d",
random.nextInt(1000));
            Train tempTrain = new Train(trainNumber,
trainName, fromStation, toStation, "", "", "", "");
            String availability =
tempTrain.getAvailabilityForClass(classType);
            if (availability.startsWith("WL")) {
                bookingStatus = "Waiting List";
                seatNumber = availability;
            } else if (availability.startsWith("RAC"))
{
                bookingStatus = "RAC";
                seatNumber = availability;
            }

            stmt.setInt(1, userId);
            // ... set other fields ...
            stmt.addBatch();
        }
        stmt.executeBatch();

        DBConnection.logSession(userId, "book_tickets
via " + paymentMethod, pnr);
```

```
        return pnr;
    } catch (SQLException e) {
        DBConnection.refundBalance(userId, totalFare);
        throw e;
    }
}
```

- Purpose: Books tickets for the specified train and passengers.
- Why Used:
    - Generates a unique PNR number using generatePNR().
    - Calculates the total fare, applying quota surcharges (e.g., Tatkal) and concessions.
    - Deducts the fare from the user's balance using DBConnection.deductBalance().
    - Inserts booking details into the Bookings table for each passenger, determining the booking status (Confirmed, Waiting List, RAC) based on availability.
    - Logs the booking action in SessionLogs.
    - Refunds the balance if booking fails.
    - Used in the passenger details panel to complete the booking process.

Method: generatePNR()

java

Copy

```
private static String generatePNR() {
    Random random = new Random();
    return "PNR" + (100000 + random.nextInt(900000));
}
```

- Purpose: Generates a random PNR number.
- Why Used:
    - Creates a unique identifier for each booking.

        ◦ Used during ticket booking to assign a PNR to the booking.

Method: convertDate(String journeyDate)

```
private static String convertDate(String journeyDate)
throws SQLException {
    try {
        SimpleDateFormat inputFormat = new
SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat outputFormat = new
SimpleDateFormat("yyyy-MM-dd");
        return
outputFormat.format(inputFormat.parse(journeyDate));
    } catch (ParseException e) {
        throw new SQLException("Invalid date format: "
+ journeyDate, e);
    }
}
```

- Purpose: Converts a date string from dd/MM/yyyy to yyyy-MM-dd format.
- Why Used:
    - MySQL expects dates in yyyy-MM-dd format for the DATE type.
    - Used during booking to ensure the journey date is stored correctly in the database.

Method: getBookingHistory(int userId)

java

Copy

```
public static ResultSet getBookingHistory(int userId)
throws SQLException {
    Connection conn = DBConnection.getConnection();
    String sql = "SELECT * FROM Bookings WHERE user_id 
= ? ORDER BY booking_date DESC";
    PreparedStatement stmt = 
conn.prepareStatement(sql);
    stmt.setInt(1, userId);
    return stmt.executeQuery();
}
```

- Purpose: Retrieves the booking history for a user.
- Why Used:
    - Queries the Bookings table to get all bookings for the user, sorted by booking_date.
    - Used in the booking history panel to display past bookings.

Method: cancelTicket(String pnrNumber)

java

Copy

```
public static boolean cancelTicket(String pnrNumber) 
throws SQLException {
    Connection conn = DBConnection.getConnection();
    conn.setAutoCommit(false);
    try {
        String getBookingSql = "SELECT user_id, fare 
FROM Bookings WHERE pnr_number = ? AND booking_status = 
'Confirmed'";
        PreparedStatement getBookingStmt = 
conn.prepareStatement(getBookingSql);
        getBookingStmt.setString(1, pnrNumber);
        ResultSet rs = getBookingStmt.executeQuery();

        if (!rs.next()) {
            conn.rollback();
            return false;
        }

        int userId = rs.getInt("user_id");
        double totalFare = 0.0;

        do {
            totalFare += rs.getDouble("fare");
        } while (rs.next());

        String updateSql = "UPDATE Bookings SET
booking_status = 'Cancelled' WHERE pnr_number = ?";
        PreparedStatement updateStmt =
conn.prepareStatement(updateSql);
        updateStmt.setString(1, pnrNumber);
        int rowsAffected = updateStmt.executeUpdate();

        if (rowsAffected > 0) {
            DBConnection.refundBalance(userId,
totalFare);
            DBConnection.logSession(userId,
"cancel_ticket", pnrNumber);
            conn.commit();
            return true;
        }

        conn.rollback();
        return false;
    } catch (SQLException e) {
        conn.rollback();
        throw e;
    } finally {
        conn.setAutoCommit(true);
    }
}
```

- Purpose: Cancels a ticket and refunds the fare.
- Why Used:
    - Ensures transactional integrity using setAutoCommit(false), commit(), and rollback().
    - Only cancels tickets with status Confirmed.
    - Calculates the total fare to refund by summing fares for all passengers with the given PNR.
    - Updates the booking status to Cancelled, refunds the balance, and logs the action.
    - Used in the cancel ticket panel to process cancellations.

Method: getPNRStatus(String pnrNumber)

java

Copy

```
public static ResultSet getPNRStatus(String pnrNumber)
throws SQLException {
    Connection conn = DBConnection.getConnection();
    String sql = "SELECT * FROM Bookings WHERE
pnr_number = ?";
    PreparedStatement stmt =
conn.prepareStatement(sql);
    stmt.setString(1, pnrNumber);
    return stmt.executeQuery();
}
```

- Purpose: Retrieves the status of a booking by PNR number.
- Why Used:

    - Queries the Bookings table to get booking details for the given PNR.
    - Used in the PNR status panel to display booking information.

Inner Class: Passenger

java

Copy

```
public static class Passenger {
    private String name;
    private int age;
    private String gender;
    private String berthPreference;
    private String concessionType;

    public Passenger(String name, int age, String
gender, String berthPreference, String concessionType)
{
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.berthPreference = berthPreference;
        this.concessionType = concessionType;
    }

    public String getName() { return name; }
    // ... other getters ...
}
```

- Purpose: Represents a passenger in a booking.
- Why Used:
    - Nested within Booking to logically group passenger-related data.

    - Immutable (no setters) to ensure passenger details aren’t modified after creation.
    - Used during booking to store and save passenger information.

Class: TrainManagementSystem

This is the main class that extends JFrame and manages the GUI and user interactions.

```
private User currentUser;
private JPanel currentPanel;
private JTextArea outputArea;
private boolean darkTheme = false;
private JLabel userInfoLabel;
private List<String> recentSearches = new
ArrayList<>();
```

- Purpose: Store application state and UI components.
- Why Used:
    - currentUser: Tracks the logged-in user.
    - currentPanel: The main panel that dynamically changes to display different screens.
    - outputArea: Displays search results, booking history, etc.
    - darkTheme: Toggles between light and dark themes.
    - userInfoLabel: Displays the user’s name in the header.
    - recentSearches: Stores recent searches (not used in the provided code but could be for future enhancements).

Constructor

```
public TrainManagementSystem() {
    super("IRCTC Train Booking System");
```

```
        setSize(1200, 800);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        DBConnection.initializeDatabase();

        JPanel headerPanel = createHeaderPanel();
        add(headerPanel, BorderLayout.NORTH);

        currentPanel = new JPanel();
        add(currentPanel, BorderLayout.CENTER);

        JPanel footerPanel = createFooterPanel();
        add(footerPanel, BorderLayout.SOUTH);

        showLoginPanel();

        setLocationRelativeTo(null);
        setVisible(true);
    }
```

- Purpose: Initializes the GUI and starts the application.
- Why Used:
    - Sets up the main window (JFrame) with a title, size, and layout.
    - Calls DBConnection.initializeDatabase() to ensure the database is ready.
    - Adds header, main, and footer panels to the frame.
    - Displays the login panel as the initial screen.
    - Centers the window (setLocationRelativeTo(null)) and makes it visible.

Method: createHeaderPanel()

java
Copy
```java
private JPanel createHeaderPanel() {
    JPanel panel = new JPanel(new BorderLayout());
    panel.setBackground(new Color(0, 100, 0));
    panel.setBorder(BorderFactory.createEmptyBorder(10,
10, 10, 10));

    JLabel titleLabel = new JLabel("IRCTC - Indian
Railway Catering and Tourism Corporation",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    titleLabel.setForeground(Color.WHITE);
    panel.add(titleLabel, BorderLayout.CENTER);

    JPanel rightPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));
    rightPanel.setBackground(new Color(0, 100, 0));

    userInfoLabel = new JLabel("Welcome, Guest");
    userInfoLabel.setForeground(Color.WHITE);
    userInfoLabel.setFont(new Font("Arial", Font.PLAIN,
14));
    rightPanel.add(userInfoLabel);

    JLabel dateLabel = new JLabel(new
SimpleDateFormat("dd-MMM-yyyy [HH:mm:ss]").format(new
java.util.Date()));
    dateLabel.setForeground(Color.WHITE);
    dateLabel.setFont(new Font("Arial", Font.PLAIN,
14));

dateLabel.setBorder(BorderFactory.createEmptyBorder(0,
10, 0, 10));
    rightPanel.add(dateLabel);

    panel.add(rightPanel, BorderLayout.EAST);

    return panel;
}
```

- Purpose: Creates the header panel for the GUI.
- Why Used:
  - Displays the application title, user name (userInfoLabel), and current date/time.
  - Uses BorderLayout to organize components (title in the center, user info and date on the right).
  - Provides a consistent header across all screens.

Method: createFooterPanel()

java

Copy

```
private JPanel createFooterPanel() {
    JPanel panel = new JPanel();
    panel.setBackground(new Color(0, 100, 0));
    panel.setBorder(BorderFactory.createEmptyBorder(5,
10, 5, 10));

    JLabel footerLabel = new JLabel("Customer Care
Numbers: ...");
    footerLabel.setForeground(Color.WHITE);
    footerLabel.setFont(new Font("Arial", Font.PLAIN,
10));
    panel.add(footerLabel);

    return panel;
}
```

- Purpose: Creates the footer panel.
- Why Used:
  - Displays customer care information and a warning about fraud.
  - Provides a consistent footer across all screens.

Method: applyTheme()

java

Copy

```
private void applyTheme() {
    if (darkTheme) {
        currentPanel.setBackground(new Color(60, 63,
65));
        currentPanel.setForeground(Color.WHITE);
        if (outputArea != null) {
            outputArea.setBackground(new Color(60, 63,
65));
            outputArea.setForeground(Color.WHITE);
        }
    } else {
        currentPanel.setBackground(new Color(238, 238,
238));
        currentPanel.setForeground(Color.BLACK);
        if (outputArea != null) {
            outputArea.setBackground(Color.WHITE);
            outputArea.setForeground(Color.BLACK);
        }
    }
}
```

- Purpose: Applies the selected theme (light or dark) to the GUI.
- Why Used:
  - Toggles between light and dark themes based on the darkTheme flag.
  - Updates the background and foreground colors of currentPanel and outputArea.
  - Called whenever the panel changes or the theme is toggled to ensure consistent styling.

Method: showLoginPanel()

java

Copy

```
private void showLoginPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Login to IRCTC",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);
    JLabel emailLabel = new JLabel("Email:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(emailLabel, gbc);

    JTextField emailField = new JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(emailField, gbc);

    JLabel passwordLabel = new JLabel("Password:");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(passwordLabel, gbc);

    JPasswordField passwordField = new
JPasswordField(20);
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(passwordField, gbc);

    JButton loginButton = new JButton("Login");
    gbc.gridx = 0;
    gbc.gridy = 3;
    gbc.gridwidth = 2;
    currentPanel.add(loginButton, gbc);

    JButton registerButton = new JButton("Register New
User");
```

```
    gbc.gridx = 0;
    gbc.gridy = 4;
    currentPanel.add(registerButton, gbc);

    JButton forgotPasswordButton = new JButton("Forgot
Password?");
    gbc.gridx = 0;
    gbc.gridy = 5;
    currentPanel.add(forgotPasswordButton, gbc);

    JButton themeToggleButton = new JButton("Toggle
Dark Theme");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(themeToggleButton, gbc);

    loginButton.addActionListener(e -> {
        try {
            String email = emailField.getText();
            String password = new
String(passwordField.getPassword());
            currentUser = User.login(email, password);
            if (currentUser != null) {
                userInfoLabel.setText("Welcome, " +
currentUser.getName());
                showMainMenu();
            } else {
                JOptionPane.showMessageDialog(this,
"Invalid email or password", "Login Failed",
JOptionPane.ERROR_MESSAGE);
```

```
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this,
"Database error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    });

    registerButton.addActionListener(e ->
showRegistrationPanel());
    forgotPasswordButton.addActionListener(e ->
showForgotPasswordPanel());
    themeToggleButton.addActionListener(e -> {
        darkTheme = !darkTheme;
        applyTheme();
    });

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

- Purpose: Displays the login panel.
- Why Used:
    - Uses GridBagLayout for a structured layout of components (labels, text fields, buttons).
    - Provides fields for email and password input.
    - Includes buttons for login, registration, forgot password, and theme toggle.
    - Login Button: Calls User.login() to authenticate the user and navigates to the main menu if successful.
    - Register Button: Navigates to the registration panel.

  - Forgot Password Button: Navigates to the forgot password panel.
  - Theme Toggle Button: Toggles the darkTheme flag and applies the theme.
  - Calls revalidate() and repaint() to update the UI after changes.

Method: showRegistrationPanel()
java
Copy
```
private void showRegistrationPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("New User
Registration", SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    String[] securityQuestions = {
        "What was your first pet's name?",
        "What was the name of your first school?",
        "What is your mother's maiden name?",
        "What city were you born in?",
        "What was your favorite food as a child?"
    };
```

```
JLabel nameLabel = new JLabel("Full Name:");
gbc.gridx = 0;
gbc.gridy = 1;
gbc.gridwidth = 1;
currentPanel.add(nameLabel, gbc);

JTextField nameField = new JTextField(20);
gbc.gridx = 1;
gbc.gridy = 1;
currentPanel.add(nameField, gbc);

JLabel mobileLabel = new JLabel("Mobile Number:");
gbc.gridx = 0;
gbc.gridy = 2;
currentPanel.add(mobileLabel, gbc);

JTextField mobileField = new JTextField(20);
gbc.gridx = 1;
gbc.gridy = 2;
currentPanel.add(mobileField, gbc);

JLabel emailLabel = new JLabel("Email:");
gbc.gridx = 0;
gbc.gridy = 3;
currentPanel.add(emailLabel, gbc);

JTextField emailField = new JTextField(20);
gbc.gridx = 1;
gbc.gridy = 3;
```

```
    currentPanel.add(emailField, gbc);

    JLabel passwordLabel = new JLabel("Password:");
    gbc.gridx = 0;
    gbc.gridy = 4;
    currentPanel.add(passwordLabel, gbc);

    JPasswordField passwordField = new
JPasswordField(20);
    gbc.gridx = 1;
    gbc.gridy = 4;
    currentPanel.add(passwordField, gbc);

    JLabel confirmPasswordLabel = new JLabel("Confirm
Password:");
    gbc.gridx = 0;
    gbc.gridy = 5;
    currentPanel.add(confirmPasswordLabel, gbc);

    JPasswordField confirmPasswordField = new
JPasswordField(20);
    gbc.gridx = 1;
    gbc.gridy = 5;
    currentPanel.add(confirmPasswordField, gbc);

    JLabel securityQuestionLabel = new JLabel("Security
Question:");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(securityQuestionLabel, gbc);
    JComboBox<String> securityQuestionCombo = new
JComboBox<>(securityQuestions);
    gbc.gridx = 1;
    gbc.gridy = 6;
    currentPanel.add(securityQuestionCombo, gbc);

    JLabel securityAnswerLabel = new JLabel("Security
Answer:");
    gbc.gridx = 0;
    gbc.gridy = 7;
    currentPanel.add(securityAnswerLabel, gbc);

    JTextField securityAnswerField = new
JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 7;
    currentPanel.add(securityAnswerField, gbc);

    JButton registerButton = new JButton("Register");
    gbc.gridx = 0;
    gbc.gridy = 8;
    gbc.gridwidth = 2;
    currentPanel.add(registerButton, gbc);

    JButton backButton = new JButton("Back to Login");
    gbc.gridx = 0;
    gbc.gridy = 9;
    currentPanel.add(backButton, gbc);
    registerButton.addActionListener(e -> {
        String name = nameField.getText();
        String mobile = mobileField.getText();
        String email = emailField.getText();
        String password = new
String(passwordField.getPassword());
        String confirmPassword = new
String(confirmPasswordField.getPassword());
        String securityQuestion = (String)
securityQuestionCombo.getSelectedItem();
        String securityAnswer =
securityAnswerField.getText();

        if (!password.equals(confirmPassword)) {
            JOptionPane.showMessageDialog(this,
"Passwords do not match", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        if (name.isEmpty() || mobile.isEmpty() ||
email.isEmpty() || password.isEmpty() ||
securityAnswer.isEmpty()) {
            JOptionPane.showMessageDialog(this, "All
fields are required", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            User newUser = new User(name, mobile,
email, password, securityQuestion, securityAnswer);
            newUser.register();
            JOptionPane.showMessageDialog(this,
"Registration successful! Please login.", "Success",
JOptionPane.INFORMATION_MESSAGE);
            showLoginPanel();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this,
"Registration failed: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    });

    backButton.addActionListener(e ->
showLoginPanel());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

- Purpose: Displays the registration panel for new users.
- Why Used:
    - Provides fields for name, mobile, email, password, confirm password, security question, and answer.
    - Register Button:
        - Validates that passwords match and all fields are filled.
        - Creates a new User object and calls register() to save it to the database.
        - Shows a success message and returns to the login panel.
    - Back Button: Returns to the login panel.

- Uses GridBagLayout for a structured layout.

Method: showForgotPasswordPanel()
java
Copy

```
private void showForgotPasswordPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Password Recovery",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(titleLabel, gbc);

    JLabel emailLabel = new JLabel("Registered
Email:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(emailLabel, gbc);

    JTextField emailField = new JTextField(20);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(emailField, gbc);

    JButton nextButton = new JButton("Next");
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.gridwidth = 2;
    currentPanel.add(nextButton, gbc);

    JButton backButton = new JButton("Back to Login");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(backButton, gbc);

    nextButton.addActionListener(e -> {
        String email = emailField.getText();
        try {
            String question =
User.getSecurityQuestion(email);
            if (question == null) {
                JOptionPane.showMessageDialog(this,
"Email not found", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            currentPanel.removeAll();
            currentPanel.setLayout(new
GridBagLayout());

            JLabel questionLabel = new JLabel("Security
Question: " + question);
```

```
            gbc.gridx = 0;
            gbc.gridy = 0;
            gbc.gridwidth = 2;
            currentPanel.add(questionLabel, gbc);

            JLabel answerLabel = new JLabel("Answer:");
            gbc.gridx = 0;
            gbc.gridy = 1;
            gbc.gridwidth = 1;
            currentPanel.add(answerLabel, gbc);

            JTextField answerField = new
JTextField(20);
            gbc.gridx = 1;
            gbc.gridy = 1;
            currentPanel.add(answerField, gbc);

            JLabel newPasswordLabel = new JLabel("New
Password:");
            gbc.gridx = 0;
            gbc.gridy = 2;
            currentPanel.add(newPasswordLabel, gbc);

            JPasswordField newPasswordField = new
JPasswordField(20);
            gbc.gridx = 1;
            gbc.gridy = 2;
            currentPanel.add(newPasswordField, gbc);
            JLabel confirmPasswordLabel = new
JLabel("Confirm Password:");
            gbc.gridx = 0;
            gbc.gridy = 3;
            currentPanel.add(confirmPasswordLabel,
gbc);

            JPasswordField confirmPasswordField = new
JPasswordField(20);
            gbc.gridx = 1;
            gbc.gridy = 3;
            currentPanel.add(confirmPasswordField,
gbc);

            JButton resetButton = new JButton("Reset
Password");
            gbc.gridx = 0;
            gbc.gridy = 4;
            gbc.gridwidth = 2;
            currentPanel.add(resetButton, gbc);

            JButton cancelButton = new
JButton("Cancel");
            gbc.gridx = 0;
            gbc.gridy = 5;
            currentPanel.add(cancelButton, gbc);

            resetButton.addActionListener(ev -> {
                String answer = answerField.getText();
                String newPassword = new
String(newPasswordField.getPassword());
                String confirmPassword = new
String(confirmPasswordField.getPassword());

                if
(!newPassword.equals(confirmPassword)) {
                    JOptionPane.showMessageDialog(this,
"Passwords do not match", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }

                try {
                    if
(User.verifySecurityAnswer(email, answer)) {
                        User.resetPassword(email,
newPassword);

JOptionPane.showMessageDialog(this, "Password reset
successful!", "Success",
JOptionPane.INFORMATION_MESSAGE);
                        showLoginPanel();
                    } else {

JOptionPane.showMessageDialog(this, "Incorrect security
answer", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(this,
"Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
                }
            });

            cancelButton.addActionListener(ev ->
showLoginPanel());

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error:
" + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }

        applyTheme();
        currentPanel.revalidate();
        currentPanel.repaint();
    });

    backButton.addActionListener(e ->
showLoginPanel());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

- Purpose: Displays the password recovery panel.
- Why Used:

- First step: Asks for the user's email and retrieves the security question using User.getSecurityQuestion().
- Second step: Displays the security question, asks for the answer, and provides fields for a new password.
- Reset Button:
  - Validates that the new passwords match.
  - Verifies the security answer using User.verifySecurityAnswer().
  - Resets the password using User.resetPassword() and returns to the login panel.
- Cancel Button: Returns to the login panel.

Method: showMainMenu()

java

Copy

```
private void showMainMenu() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel welcomeLabel = new JLabel("Welcome, " +
currentUser.getName(), SwingConstants.CENTER);
    welcomeLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    currentPanel.add(welcomeLabel, gbc);
    JButton bookTicketButton = new JButton("Book
Ticket");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(bookTicketButton, gbc);

    JButton bookingHistoryButton = new JButton("Booking
History");
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(bookingHistoryButton, gbc);

    JButton cancelTicketButton = new JButton("Cancel
Ticket");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(cancelTicketButton, gbc);

    JButton pnrStatusButton = new JButton("PNR
Status");
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(pnrStatusButton, gbc);

    JButton manageProfileButton = new JButton("Manage
Profile");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(manageProfileButton, gbc);
    JButton accountBalanceButton = new JButton("Account
Balance");
    gbc.gridx = 1;
    gbc.gridy = 3;
    currentPanel.add(accountBalanceButton, gbc);

    JButton logoutButton = new JButton("Logout");
    gbc.gridx = 0;
    gbc.gridy = 4;
    gbc.gridwidth = 2;
    currentPanel.add(logoutButton, gbc);

    bookTicketButton.addActionListener(e ->
showTrainSearchPanel());
    bookingHistoryButton.addActionListener(e ->
showBookingHistory());
    cancelTicketButton.addActionListener(e ->
showCancelTicketPanel());
    pnrStatusButton.addActionListener(e ->
showPNRStatusPanel());
    manageProfileButton.addActionListener(e ->
showProfilePanel());
    accountBalanceButton.addActionListener(e ->
showAccountBalance());
    logoutButton.addActionListener(e -> {
        currentUser.logout();
        currentUser = null;
        userInfoLabel.setText("Welcome, Guest");
        showLoginPanel();
    });

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

- Purpose: Displays the main menu for the logged-in user.
- Why Used:
  - Provides buttons for all main functionalities: book ticket, view booking history, cancel ticket, check PNR status, manage profile, view account balance, and logout.
  - Each button navigates to the corresponding panel.
  - Logout Button: Calls User.logout(), clears currentUser, and returns to the login panel.

Method: showTrainSearchPanel()

java

Copy

```
private void showTrainSearchPanel() {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Search Trains",
SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 4;
    currentPanel.add(titleLabel, gbc);

    JLabel fromLabel = new JLabel("From:");
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 1;
    currentPanel.add(fromLabel, gbc);

    JTextField fromField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 1;
    currentPanel.add(fromField, gbc);

    JLabel toLabel = new JLabel("To:");
    gbc.gridx = 2;
    gbc.gridy = 1;
    currentPanel.add(toLabel, gbc);

    JTextField toField = new JTextField(15);
    gbc.gridx = 3;
    gbc.gridy = 1;
    currentPanel.add(toField, gbc);

    JLabel dateLabel = new JLabel("Journey Date
(dd/mm/yyyy):");
    gbc.gridx = 0;
    gbc.gridy = 2;
    currentPanel.add(dateLabel, gbc);
    JTextField dateField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(dateField, gbc);

    JLabel classLabel = new JLabel("Class:");
    gbc.gridx = 2;
    gbc.gridy = 2;
    currentPanel.add(classLabel, gbc);

    JComboBox<String> classCombo = new
JComboBox<>(Booking.getClassTypes());
    gbc.gridx = 3;
    gbc.gridy = 2;
    currentPanel.add(classCombo, gbc);

    JLabel trainTypeLabel = new JLabel("Train Type:");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(trainTypeLabel, gbc);

    String[] trainTypes = {"All", "Express",
"Rajdhani", "Shatabdi", "Duronto", "Jan Shatabdi"};
    JComboBox<String> trainTypeCombo = new
JComboBox<>(trainTypes);
    gbc.gridx = 1;
    gbc.gridy = 3;
    currentPanel.add(trainTypeCombo, gbc);
    JLabel timeRangeLabel = new JLabel("Departure
Time:");
    gbc.gridx = 2;
    gbc.gridy = 3;
    currentPanel.add(timeRangeLabel, gbc);

    String[] timeRanges = {"All", "00:00-06:00",
"06:00-12:00", "12:00-18:00", "18:00-24:00"};
    JComboBox<String> timeRangeCombo = new
JComboBox<>(timeRanges);
    gbc.gridx = 3;
    gbc.gridy = 3;
    currentPanel.add(timeRangeCombo, gbc);

    JButton searchButton = new JButton("Search
Trains");
    gbc.gridx = 0;
    gbc.gridy = 4;
    gbc.gridwidth = 4;
    currentPanel.add(searchButton, gbc);

    JButton backButton = new JButton("Back to Main
Menu");
    gbc.gridx = 0;
    gbc.gridy = 5;
    currentPanel.add(backButton, gbc);

    outputArea = new JTextArea(15, 60);
    outputArea.setEditable(false);
    JScrollPane scrollPane = new
JScrollPane(outputArea);
    gbc.gridx = 0;
    gbc.gridy = 6;
    gbc.gridwidth = 4;
    gbc.fill = GridBagConstraints.BOTH;
    currentPanel.add(scrollPane, gbc);

    searchButton.addActionListener(e -> {
        String from = fromField.getText();
        String to = toField.getText();
        String date = dateField.getText();
        String classType = (String)
classCombo.getSelectedItem();
        String trainType = (String)
trainTypeCombo.getSelectedItem();
        String timeRange = (String)
timeRangeCombo.getSelectedItem();

        try {
            List<Train> trains =
Train.searchTrainsWithFilters(from, to, date,
classType, trainType, timeRange);
            displayTrainResults(trains, classType);
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error
searching trains: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    });
    backButton.addActionListener(e -> showMainMenu());

    applyTheme();
    currentPanel.revalidate();
    currentPanel.repaint();
}
```

- Purpose: Displays the train search panel.
- Why Used:
    - Provides fields for source, destination, journey date, class, train type, and departure time range.
    - Search Button:
        - Calls Train.searchTrainsWithFilters() to find matching trains.
        - Displays results using displayTrainResults().
    - Back Button: Returns to the main menu.
    - Uses JTextArea within a JScrollPane to display search results.

Method: displayTrainResults(List<Train> trains, String classType)
java
Copy
```
private void displayTrainResults(List<Train> trains,
String classType) {
    outputArea.setText("");
    if (trains.isEmpty()) {
        outputArea.append("No trains found for the
given criteria.\n");
        return;
    }
    outputArea.append(String.format("%-10s %-30s %-20s
%-20s %-10s %-10s %-10s %-15s\n",
            "Train No", "Train Name", "Departure",
"Arrival", "Duration", "From", "To", "Availability"));

outputArea.append("-------------------------------------
-------------------------------------------------------
-------------------------\n");

    for (Train train : trains) {
        String availability =
train.getAvailabilityForClass(classType);
        outputArea.append(String.format("%-10s %-30s
%-20s %-20s %-10s %-10s %-10s %-15s\n",
                train.getTrainNumber(),
                train.getTrainName(),
                train.getSourceStation() + " (" +
train.getDepartureTime() + ")",
                train.getDestinationStation() + " (" +
train.getArrivalTime() + ")",
                train.getDuration(),
                train.getSourceStation(),
                train.getDestinationStation(),
                availability));
    }

    if (trains.size() > 0) {
        JButton bookButton = new JButton("Book Selected
Train");
```

```
        GridBagConstraints gbc = new
GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 7;
        gbc.gridwidth = 4;
        gbc.insets = new Insets(10, 10, 10, 10);
        currentPanel.add(bookButton, gbc);

        bookButton.addActionListener(e -> {
            String selectedTrain =
JOptionPane.showInputDialog(this, "Enter Train Number
to book:");
            if (selectedTrain != null &&
!selectedTrain.isEmpty()) {
                for (Train train : trains) {
                    if
(train.getTrainNumber().equals(selectedTrain)) {

showPassengerDetailsPanel(train);
                        break;
                    }
                }
            }
        });

        currentPanel.revalidate();
        currentPanel.repaint();
    }
}
```

- Purpose: Displays the results of a train search.
- Why Used:
    - Formats train details (number, name, departure, arrival, etc.) in a tabular format in the outputArea.
    - Adds a “Book Selected Train” button if trains are found.
    - Book Button:
        - Prompts the user to enter the train number.
        - Navigates to the passenger details panel for the selected train.

Method: showPassengerDetailsPanel(Train train)

java

Copy

```
private void showPassengerDetailsPanel(Train train) {
    currentPanel.removeAll();
    currentPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;

    JLabel titleLabel = new JLabel("Passenger Details
for " + train.getTrainName(), SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD,
24));
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 4;
    currentPanel.add(titleLabel, gbc);

    JLabel journeyLabel = new
JLabel(train.getSourceStation() + " to " +
train.getDestinationStation());
```

```
    gbc.gridx = 0;
    gbc.gridy = 1;
    gbc.gridwidth = 4;
    currentPanel.add(journeyLabel, gbc);

    JLabel dateLabel = new JLabel("Journey Date
(dd/mm/yyyy):");
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.gridwidth = 1;
    currentPanel.add(dateLabel, gbc);

    JTextField dateField = new JTextField(15);
    gbc.gridx = 1;
    gbc.gridy = 2;
    currentPanel.add(dateField, gbc);

    JLabel classLabel = new JLabel("Class:");
    gbc.gridx = 2;
    gbc.gridy = 2;
    currentPanel.add(classLabel, gbc);

    JComboBox<String> classCombo = new
JComboBox<>(Booking.getClassTypes());
    gbc.gridx = 3;
    gbc.gridy = 2;
    currentPanel.add(classCombo, gbc);

    JLabel quotaLabel = new JLabel("Quota:");
    gbc.gridx = 0;
    gbc.gridy = 3;
    currentPanel.add(quotaLabel, gbc);

    JComboBox<String> quotaCombo = new
JComboBox<>(Booking.getQuotaTypes());
    gbc.gridx = 1;
    gbc.gridy = 3;
    currentPanel.add(quotaCombo, gbc);

    JLabel passengersLabel = new JLabel("Number of
Passengers:");
    gbc.gridx = 2;
    gbc.gridy = 3;
    currentPanel.add(passengersLabel, gbc);

    SpinnerModel spinnerModel = new
SpinnerNumberModel(1, 1, 6, 1);
    JSpinner passengersSpinner = new
JSpinner(spinnerModel);
    gbc.gridx = 3;
    gbc.gridy = 3;
    currentPanel.add(passengersSpinner, gbc);

    JPanel passengersPanel = new JPanel();
    passengersPanel.setLayout(new
BoxLayout(passengersPanel, BoxLayout.Y_AXIS));
    JScrollPane scrollPane = new
JScrollPane(passengersPanel);
    gbc.gridx = 0;
    gbc.gridy = 4;
    gbc.gridwidth = 4;
    gbc.fill = GridBagConstraints.BOTH;
    currentPanel.add(scrollPane, gbc);

    JButton addPassengersButton = new JButton("Add
Passenger Details");
    gbc.gridx = 0;
    gbc.gridy = 5;
    gbc.gridwidth = 4;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    currentPanel.add(addPassengersButton, gbc);

    JButton bookButton = new JButton("Proceed to
Payment");
    gbc.gridx = 0;
    gbc.gridy = 6;
    currentPanel.add(bookButton, gbc);

    JButton backButton = new JButton("Back to Search");
    gbc.gridx = 1;
    gbc.gridy = 6;
    currentPanel.add(backButton, gbc);

    List<JPanel> passengerForms = new ArrayList<>();

    addPassengersButton.addActionListener(e -> {
        int numPassengers = (Integer)
passengersSpinner.getValue();
        passengersPanel.removeAll();
        passengerForms.clear();
        for (int i = 0; i < numPassengers; i++) {
            JPanel formPanel = new JPanel(new
GridLayout(0, 2, 5, 5));

formPanel.setBorder(BorderFactory.createTitledBorder("P
assenger " + (i + 1)));

            formPanel.add(new JLabel("Name:"));
            JTextField nameField = new JTextField();
            formPanel.add(nameField);

            formPanel.add(new JLabel("Age:"));
            JSpinner ageSpinner = new JSpinner(new
SpinnerNumberModel(18, 1, 120, 1));
            formPanel.add(ageSpinner);

            formPanel.add(new JLabel("Gender:"));
            JComboBox<String> genderCombo = new
JComboBox<>(new String[]{"Male", "Female", "Other"});
            formPanel.add(genderCombo);

            formPanel.add(new JLabel("Berth
Preference:"));
            JComboBox<String> berthCombo = new
JComboBox<>(new String[]{"No Preference", "Lower",
"Middle", "Upper", "Side Lower", "Side Upper"});
            formPanel.add(berthCombo);

            formPanel.add(new JLabel("Concession:"));
            JComboBox<String> concessionCombo = new
JComboBox<>(Booking.getConcessionTypes());
            concessionCombo.insertItemAt("None", 0);
            concessionCombo.setSelectedIndex(0);
            formPanel.add(concessionCombo);

            passengersPanel.add(formPanel);
            passengerForms.add(formPanel);
        }

        passengersPanel.revalidate();
        passengersPanel.repaint();
    });

    bookButton.addActionListener(e -> {
        String journeyDate = dateField.getText();
        String classType = (String)
classCombo.getSelectedItem();
        String quota = (String)
quotaCombo.getSelectedItem();

        if (journeyDate.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please
enter journey date", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        List<Booking.Passenger> passengers = new
ArrayList<>();
```

```
        for (JPanel form : passengerForms) {
            Component[] components =
form.getComponents();
            String name = ((JTextField)
components[1]).getText();
            int age = (Integer) ((JSpinner)
components[3]).getValue();
            String gender = (String) ((JComboBox<?>)
components[5]).getSelectedItem();
            String berthPreference = (String)
((JComboBox<?>) components[7]).getSelectedItem();
            String concessionType = (String)
((JComboBox<?>) components[9]).getSelectedItem();

            if (name.isEmpty()) {
                JOptionPane.showMessageDialog(this,
"Please enter name for all passengers", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            if ("None".equals(concessionType)) {
                concessionType = null;
            }

            passengers.add(new Booking.Passenger(name,
age, gender, berthPreference, concessionType));
        }

        if (passengers.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please
add passenger details first", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            String pnr = Booking.bookTickets(
                currentUser.getUserId(),
                train.getTrainNumber(),
                train.getTrainName(),
                train.getSourceStation(),
                train.getDestinationStation(),
                journeyDate,
                classType,
                quota,
                passengers,
                "Account Balance"
            );

            JOptionPane.showMessageDialog(this,
                "Booking successful!\nPNR: " + pnr +
"\nTotal Fare: " + calculateTotalFare(passengers,
classType, quota),
                "Success",
JOptionPane.INFORMATION_MESSAGE);
            showMainMenu();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this,
"Booking failed: " + ex.getMessage
```

2.SQL code

```
-- Create the IRCTC_System database
CREATE DATABASE IF NOT EXISTS IRCTC_System;
USE IRCTC_System;

-- Create Users table
CREATE TABLE IF NOT EXISTS Users (
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   mobile VARCHAR(15) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   password VARCHAR(100) NOT NULL,
   security_question VARCHAR(200) NOT NULL,
   security_answer VARCHAR(100) NOT NULL
);

-- Create Trains table
CREATE TABLE IF NOT EXISTS Trains (
   train_id INT AUTO_INCREMENT PRIMARY KEY,
   train_number VARCHAR(10) UNIQUE NOT NULL,
   train_name VARCHAR(100) NOT NULL,
   source_station VARCHAR(100) NOT NULL,
   destination_station VARCHAR(100) NOT NULL,
   departure_time VARCHAR(10) NOT NULL,
   arrival_time VARCHAR(10) NOT NULL,
   duration VARCHAR(10) NOT NULL,
   runs_on VARCHAR(50) NOT NULL
);

-- Create Bookings table
CREATE TABLE IF NOT EXISTS Bookings (
   booking_id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   train_number VARCHAR(10) NOT NULL,
   train_name VARCHAR(100) NOT NULL,
   from_station VARCHAR(100) NOT NULL,
   to_station VARCHAR(100) NOT NULL,
   journey_date DATE NOT NULL,
   class_type VARCHAR(50) NOT NULL,
   passenger_name VARCHAR(100) NOT NULL,
   age INT NOT NULL,
   gender VARCHAR(10) NOT NULL,
   berth_preference VARCHAR(20),
   fare DECIMAL(10,2) NOT NULL,
   pnr_number VARCHAR(15) UNIQUE NOT NULL,
   booking_status VARCHAR(20) DEFAULT 'Confirmed',
   seat_number VARCHAR(10),
   booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

-- Check if seat_number column exists in Bookings table and add it if it
doesn't
SET @column_exists = (SELECT COUNT(*)
                      FROM INFORMATION_SCHEMA.COLUMNS
                      WHERE TABLE_NAME = 'Bookings'
                      AND COLUMN_NAME = 'seat_number'
                      AND TABLE_SCHEMA = 'IRCTC_System');
SET @sql = IF(@column_exists = 0,
              'ALTER TABLE Bookings ADD COLUMN seat_number
VARCHAR(10) AFTER booking_status',
              'SELECT 1');
PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

-- Create AccountHolder table
CREATE TABLE IF NOT EXISTS AccountHolder (
   account_holder_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   account_number VARCHAR(20) UNIQUE NOT NULL,
   balance DECIMAL(15, 2),
   interest DECIMAL(5, 2),
   user_id INT,
   FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

-- Create Profiles table
CREATE TABLE IF NOT EXISTS Profiles (
   profile_id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   age INT,
   gender VARCHAR(10),
   preferred_class VARCHAR(50),
   gov_id VARCHAR(20),
   emergency_contact VARCHAR(15),
   wheelchair_access BOOLEAN DEFAULT FALSE,
   medical_condition BOOLEAN DEFAULT FALSE,
   FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

-- Create SessionLogs table
CREATE TABLE IF NOT EXISTS SessionLogs (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT,
   action VARCHAR(100) NOT NULL,
   timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   session_id VARCHAR(50)
);

-- Insert sample train data
```

```
INSERT IGNORE INTO Trains (train_number, train_name, source_station,
destination_station, departure_time, arrival_time, duration, runs_on)
VALUES
('12001', 'Shatabdi Express', 'New Delhi', 'Bhopal', '08:10', '16:25', '08:15',
'Daily'),
('12309', 'Rajdhani Express', 'New Delhi', 'Kolkata', '17:00', '10:55', '17:55',
'Daily'),
('12621', 'Tamil Nadu Express', 'New Delhi', 'Chennai', '22:30', '07:10',
'32:40', 'Daily'),
('12295', 'Duronto Express', 'Mumbai', 'Kolkata', '17:15', '18:50', '25:35',
'Mon,Wed,Fri'),
('12055', 'Jan Shatabdi', 'Dehradun', 'New Delhi', '15:20', '21:10', '05:50',
'Daily'),
('12951', 'Rajdhani Express', 'Mumbai', 'Delhi', '16:35', '08:15', '15:40',
'Daily'),
('12952', 'Rajdhani Express', 'Delhi', 'Mumbai', '16:55', '08:30', '15:35',
'Daily'),
('12301', 'Duronto Express', 'Howrah', 'Delhi', '22:05', '08:00', '09:55',
'Tue,Thu,Sat'),
('12302', 'Duronto Express', 'Delhi', 'Howrah', '22:45', '08:40', '09:55',
'Mon,Wed,Fri'),
('12305', 'Duronto Express', 'Sealdah', 'Delhi', '22:50', '09:40', '10:50',
'Daily');

-- Insert a sample user
INSERT IGNORE INTO Users (name, mobile, email, password,
security_question, security_answer) VALUES
('Admin User', '9876543210', 'admin@irctc.com', 'password123', 'What was
your first pet''s name?', 'Fluffy');

-- Set user_id variable for sample data
SET @user_id = (SELECT user_id FROM Users WHERE email =
'admin@irctc.com' LIMIT 1);

-- Insert sample AccountHolder data
```

```
INSERT IGNORE INTO AccountHolder (name, age, account_number,
balance, interest, user_id) VALUES
('John Doe', 30, 'ACC1001', 50000.00, 3.5, NULL),
('Jane Smith', 25, 'ACC1002', 75000.00, 3.5, NULL);

-- Insert a sample booking for testing
INSERT IGNORE INTO Bookings (user_id, train_number, train_name,
from_station, to_station, journey_date, class_type, passenger_name, age,
gender, berth_preference, fare, pnr_number, booking_status, seat_number)
VALUES (@user_id, '12001', 'Shatabdi Express', 'New Delhi', 'Bhopal',
'2025-05-05', 'AC 2 Tier (2A)', 'John Doe', 30, 'Male', 'Lower', 2500.00,
'PNR123456', 'Confirmed', '001');

-- Sample queries for testing and verification
SELECT * FROM Users;
SELECT * FROM Trains;
SELECT * FROM Bookings;
SELECT * FROM AccountHolder;
SELECT * FROM Profiles;
SELECT * FROM SessionLogs;

-- Find bookings for the sample user
SELECT * FROM Bookings WHERE user_id = @user_id;

-- Find account details for the sample user
SELECT * FROM AccountHolder WHERE user_id = @user_id;

-- Find bookings by PNR number
SELECT * FROM Bookings WHERE pnr_number = 'PNR123456';

-- Find trains between two stations
SELECT * FROM Trains
WHERE source_station LIKE '%NEW DELHI%'
AND destination_station LIKE '%BHOPAL%';
```

-- Update a booking status
UPDATE Bookings SET booking_status = 'Cancelled' WHERE pnr_number = 'PNR123456';

-- Delete a booking
DELETE FROM Bookings WHERE pnr_number = 'PNR123456';

-- Delete a user (cascade deletes bookings, account holder, profiles, and session logs)
DELETE FROM Bookings WHERE user_id = @user_id;
DELETE FROM AccountHolder WHERE user_id = @user_id;
DELETE FROM Profiles WHERE user_id = @user_id;
DELETE FROM SessionLogs WHERE user_id = @user_id;
DELETE FROM Users WHERE user_id = @user_id;

-- Set the root password
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';

2.About code

Database Creation and Setup

sql
Copy
```
CREATE DATABASE IF NOT EXISTS IRCTC_System;
USE IRCTC_System;
```

- **Purpose**: Creates the database and sets it as the active database.
- **Explanation**:
  - CREATE DATABASE IF NOT EXISTS IRCTC_System: Creates a database named IRCTC_System if it doesn't already exist, preventing errors if the database is already present.
  - USE IRCTC_System: Sets IRCTC_System as the active database for subsequent operations.

- **Why Used**: This establishes the foundation for storing all data related to users, trains, bookings, accounts, profiles, and session logs.

Table Creation

Users Table

sql

Copy

```
CREATE TABLE IF NOT EXISTS Users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    mobile VARCHAR(15) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    security_question VARCHAR(200) NOT NULL,
    security_answer VARCHAR(100) NOT NULL
);
```

- **Purpose**: Stores user information for authentication and identification.
- **Explanation**:
    - user_id: Auto-incremented primary key to uniquely identify each user.
    - name, mobile, email, password, security_question, security_answer: Store user details, with NOT NULL ensuring these fields are mandatory.
    - email UNIQUE: Ensures no two users can have the same email address.
    - VARCHAR lengths are chosen to accommodate typical data (e.g., mobile as VARCHAR(15) to handle international numbers).

- **Why Used**: Required for user registration, login, and password recovery in the Train Management System. The user_id is referenced as a foreign key in other tables.

Trains Table

sql
Copy

```
CREATE TABLE IF NOT EXISTS Trains (
    train_id INT AUTO_INCREMENT PRIMARY KEY,
    train_number VARCHAR(10) UNIQUE NOT NULL,
    train_name VARCHAR(100) NOT NULL,
    source_station VARCHAR(100) NOT NULL,
    destination_station VARCHAR(100) NOT NULL,
    departure_time VARCHAR(10) NOT NULL,
    arrival_time VARCHAR(10) NOT NULL,
    duration VARCHAR(10) NOT NULL,
    runs_on VARCHAR(50) NOT NULL
);
```

- **Purpose**: Stores train details for searching and booking.
- **Explanation**:
    - train_id: Auto-incremented primary key for internal use.
    - train_number UNIQUE: Ensures each train has a unique identifier (e.g., "12001").
    - train_name, source_station, destination_station: Descriptive fields for the train.
    - departure_time, arrival_time, duration: Store timing details in HH:MM format (e.g., "08:10").
    - runs_on: Days the train operates (e.g., "Daily", "Mon,Wed,Fri").
    - All fields are NOT NULL to ensure complete train data.
- **Why Used**: Provides the core data for train search functionality in the application.

Bookings Table

sql
Copy
```sql
CREATE TABLE IF NOT EXISTS Bookings (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    train_number VARCHAR(10) NOT NULL,
    train_name VARCHAR(100) NOT NULL,
    from_station VARCHAR(100) NOT NULL,
    to_station VARCHAR(100) NOT NULL,
    journey_date DATE NOT NULL,
    class_type VARCHAR(50) NOT NULL,
    passenger_name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    gender VARCHAR(10) NOT NULL,
    berth_preference VARCHAR(20),
    fare DECIMAL(10,2) NOT NULL,
    pnr_number VARCHAR(15) UNIQUE NOT NULL,
    booking_status VARCHAR(20) DEFAULT 'Confirmed',
    seat_number VARCHAR(10),
    booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

- **Purpose**: Stores booking details for each passenger.
- **Explanation**:
    - booking_id: Auto-incremented primary key.
    - user_id: Links the booking to a user via a foreign key.
    - train_number, train_name, from_station, to_station: Train details for the booking.
    - journey_date: Date of travel in YYYY-MM-DD format.
    - class_type: Class of travel (e.g., "AC 2 Tier (2A)").

    - passenger_name, age, gender, berth_preference: Passenger details.
    - fare: Ticket cost with two decimal places (e.g., 2500.00).
    - pnr_number UNIQUE: Unique identifier for the booking (e.g., "PNR123456").
    - booking_status: Status of the booking (e.g., "Confirmed", "Waiting List", "Cancelled"), defaults to "Confirmed".
    - seat_number: Seat assignment (e.g., "001"), added later via a script.
    - booking_date: Timestamp of when the booking was made.
- **Why Used**: Essential for booking tickets, checking PNR status, viewing booking history, and cancelling tickets.

Add seat_number Column (Dynamic Check)

```
SET @column_exists = (SELECT COUNT(*)
                      FROM INFORMATION_SCHEMA.COLUMNS
                      WHERE TABLE_NAME = 'Bookings'
                      AND COLUMN_NAME = 'seat_number'
                      AND TABLE_SCHEMA =
'IRCTC_System');
SET @sql = IF(@column_exists = 0,
              'ALTER TABLE Bookings ADD COLUMN
seat_number VARCHAR(10) AFTER booking_status',
              'SELECT 1');
PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

- **Purpose**: Dynamically adds the seat_number column to the Bookings table if it doesn’t exist.
- **Explanation**:
    - Checks INFORMATION_SCHEMA.COLUMNS to see if seat_number exists in the Bookings table.

    - If it doesn't exist (@column_exists = 0), prepares an ALTER TABLE statement to add the column after booking_status.
    - If it exists, executes a dummy SELECT 1 statement to avoid errors.
- **Why Used**: Ensures backward compatibility if the database schema evolves (e.g., older versions might not have seat_number).

AccountHolder Table

sql

Copy

```sql
CREATE TABLE IF NOT EXISTS AccountHolder (
    account_holder_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    account_number VARCHAR(20) UNIQUE NOT NULL,
    balance DECIMAL(15, 2),
    interest DECIMAL(5, 2),
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

- **Purpose**: Stores account details for payment and refunds.
- **Explanation**:
    - account_holder_id: Auto-incremented primary key.
    - name, age: Account holder details (optional, as they can be NULL).
    - account_number UNIQUE: Unique account identifier (e.g., "ACC1001").
    - balance: Account balance for transactions (e.g., 50000.00).
    - interest: Interest rate (e.g., 3.5).
    - user_id: Links the account to a user (can be NULL for sample accounts not tied to users).

- **Why Used**: Manages user funds for booking payments and refunds in the system.

Profiles Table

```
CREATE TABLE IF NOT EXISTS Profiles (
    profile_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    age INT,
    gender VARCHAR(10),
    preferred_class VARCHAR(50),
    gov_id VARCHAR(20),
    emergency_contact VARCHAR(15),
    wheelchair_access BOOLEAN DEFAULT FALSE,
    medical_condition BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

- **Purpose**: Stores user profile information for preferences and accessibility needs.
- **Explanation**:
    - profile_id: Auto-incremented primary key.
    - user_id: Links the profile to a user.
    - age, gender, preferred_class, gov_id, emergency_contact: Optional user details.
    - wheelchair_access, medical_condition: Boolean flags for accessibility requirements, default to FALSE.
- **Why Used**: Allows users to save preferences (e.g., preferred class) and accessibility needs, enhancing user experience.

SessionLogs Table

```
CREATE TABLE IF NOT EXISTS SessionLogs (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    action VARCHAR(100) NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    session_id VARCHAR(50)
);
```

- **Purpose**: Logs user actions for auditing and debugging.
- **Explanation**:
    - log_id: Auto-incremented primary key.
    - user_id: Links the log to a user (can be NULL if the action isn't tied to a user).
    - action: Describes the action (e.g., "login", "book_tickets").
    - timestamp: Records when the action occurred.
    - session_id: Tracks the user session (e.g., a UUID).
- **Why Used**: Provides an audit trail for user activities, useful for troubleshooting and monitoring.

Sample Data Insertion

Insert Sample Trains

sql

Copy

```
INSERT IGNORE INTO Trains (train_number, train_name,
source_station, destination_station, departure_time,
arrival_time, duration, runs_on) VALUES
('12001', 'Shatabdi Express', 'New Delhi', 'Bhopal',
'08:10', '16:25', '08:15', 'Daily'),
('12309', 'Rajdhani Express', 'New Delhi', 'Kolkata',
'17:00', '10:55', '17:55', 'Daily'),
-- ... other trains ...
('12305', 'Duronto Express', 'Sealdah', 'Delhi',
'22:50', '09:40', '10:50', 'Daily');
```

- **Purpose**: Populates the Trains table with sample data for testing.
- **Explanation**:
    - INSERT IGNORE: Inserts data only if it doesn't already exist (based on train_number uniqueness).
    - Adds 10 trains with realistic data for routes, timings, and operating days.
- **Why Used**: Provides initial data for the train search feature, allowing users to test the system without manually adding trains.

Insert Sample User

sql

Copy

```
INSERT IGNORE INTO Users (name, mobile, email,
password, security_question, security_answer) VALUES
('Admin User', '9876543210', 'admin@irctc.com',
'password123', 'What was your first pet''s name?',
'Fluffy');
```

- **Purpose**: Adds a sample user for testing.
- **Explanation**:
    - INSERT IGNORE: Avoids duplicate entries if the email already exists.
    - Adds a user named "Admin User" with a known email and password.
    - Uses '' to escape the single quote in "pet's".
- **Why Used**: Allows testing of login, booking, and other user-related features.

Set user_id for Sample Data

sql

Copy

```
SET @user_id = (SELECT user_id FROM Users WHERE email =
'admin@irctc.com' LIMIT 1);
```

- **Purpose**: Retrieves the user_id of the sample user for use in subsequent inserts.
- **Explanation**:
    - Queries the Users table to get the user_id for the email admin@irctc.com.
    - Stores the result in the variable @user_id.
- **Why Used**: Ensures the sample booking and account data can be linked to the sample user.

Insert Sample AccountHolder Data

sql

Copy

```
INSERT IGNORE INTO AccountHolder (name, age,
account_number, balance, interest, user_id) VALUES
('John Doe', 30, 'ACC1001', 50000.00, 3.5, NULL),
('Jane Smith', 25, 'ACC1002', 75000.00, 3.5, NULL);
```

- **Purpose**: Adds sample account holders for testing payment functionality.
- **Explanation**:
    - INSERT IGNORE: Avoids duplicates based on account_number.
    - Adds two accounts with balances (50,000 and 75,000) and an interest rate of 3.5%.
    - user_id is NULL because these accounts aren't tied to users in this script (though the Java code links accounts to users during registration).
- **Why Used**: Provides initial accounts to test payment and refund features.

Insert Sample Booking

```
INSERT IGNORE INTO Bookings (user_id, train_number,
train_name, from_station, to_station, journey_date,
class_type, passenger_name, age, gender,
berth_preference, fare, pnr_number, booking_status,
seat_number)
VALUES (@user_id, '12001', 'Shatabdi Express', 'New
Delhi', 'Bhopal', '2025-05-05', 'AC 2 Tier (2A)', 'John
Doe', 30, 'Male', 'Lower', 2500.00, 'PNR123456',
'Confirmed', '001');
```

- **Purpose**: Adds a sample booking for testing.
- **Explanation**:
  - Links the booking to the sample user via @user_id.
  - Uses train "12001" (Shatabdi Express) for a journey on May 5, 2025.
  - Includes passenger details and a fare of 2,500.
  - pnr_number is set to "PNR123456" for easy lookup.
- **Why Used**: Allows testing of booking history, PNR status, and cancellation features.

Sample Queries for Testing

Retrieve All Data

sql

Copy

```
SELECT * FROM Users;
SELECT * FROM Trains;
SELECT * FROM Bookings;
SELECT * FROM AccountHolder;
SELECT * FROM Profiles;
SELECT * FROM SessionLogs;
```

- **Purpose**: Verifies that data was inserted correctly.

- **Explanation**: Simple SELECT * queries to display all rows from each table.
- **Why Used**: Helps developers confirm that the database is set up as expected.

Find Bookings for the Sample User

sql

Copy

```
SELECT * FROM Bookings WHERE user_id = @user_id;
```

- **Purpose**: Retrieves all bookings for the sample user.
- **Explanation**: Filters the Bookings table by user_id.
- **Why Used**: Tests the booking history feature for the sample user.

Find Account Details for the Sample User

sql

Copy

```
SELECT * FROM AccountHolder WHERE user_id = @user_id;
```

- **Purpose**: Retrieves the account details for the sample user.
- **Explanation**: Filters the AccountHolder table by user_id.
- **Why Used**: Tests the account balance feature (though the sample accounts in this script have user_id as NULL).

Find Bookings by PNR Number

sql

Copy

```
SELECT * FROM Bookings WHERE pnr_number = 'PNR123456';
```

- **Purpose**: Retrieves booking details for a specific PNR.
- **Explanation**: Filters the Bookings table by pnr_number.
- **Why Used**: Tests the PNR status feature.

Find Trains Between Two Stations

sql

Copy
```
SELECT * FROM Trains
WHERE source_station LIKE '%NEW DELHI%'
AND destination_station LIKE '%BHOPAL%';
```

- **Purpose**: Finds trains between New Delhi and Bhopal.
- **Explanation**: Uses LIKE with wildcards (%) for partial matching of station names.
- **Why Used**: Tests the train search functionality.

Sample Update and Delete Operations

Update Booking Status
sql
Copy
```
UPDATE Bookings SET booking_status = 'Cancelled' WHERE
pnr_number = 'PNR123456';
```

- **Purpose**: Changes the status of a booking to "Cancelled".
- **Explanation**: Updates the booking_status column for the specified pnr_number.
- **Why Used**: Tests the ticket cancellation feature.

Delete a Booking
sql
Copy
```
DELETE FROM Bookings WHERE pnr_number = 'PNR123456';
```

- **Purpose**: Removes a booking from the database.
- **Explanation**: Deletes rows from the Bookings table based on pnr_number.

- **Why Used**: Tests the ability to clean up bookings (though the Java code uses UPDATE for cancellations).

Delete a User (Cascading Deletes)

sql

Copy

```
DELETE FROM Bookings WHERE user_id = @user_id;
DELETE FROM AccountHolder WHERE user_id = @user_id;
DELETE FROM Profiles WHERE user_id = @user_id;
DELETE FROM SessionLogs WHERE user_id = @user_id;
DELETE FROM Users WHERE user_id = @user_id;
```

- **Purpose**: Deletes a user and all related data.
- **Explanation**:
    - Deletes dependent records first (Bookings, AccountHolder, Profiles, SessionLogs) to avoid foreign key constraint violations.
    - Finally deletes the user from the Users table.
- **Why Used**: Tests user deletion while maintaining referential integrity. Note that ON DELETE CASCADE could be added to foreign keys to automate this.

Set Root Password

sql

Copy

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';
```

- **Purpose**: Sets the password for the MySQL root user to "root".
- **Explanation**:
    - Changes the root user’s password to match the credentials used in the Java code (DBConnection class).
- **Why Used**: Ensures the Java application can connect to the MySQL database using the root user with password root. However, in a

production environment, using the root user with a simple password is insecure—better practice would be to create a dedicated user with limited privileges.

Justification and Design Choices

1. **Schema Design**:
    - **Normalization**: The schema is normalized to reduce redundancy (e.g., user_id is stored in Bookings rather than duplicating user details).
    - **Foreign Keys**: Enforce referential integrity (e.g., Bookings.user_id references Users.user_id).
    - **Data Types**: Chosen to match data requirements (e.g., DECIMAL for fare and balance to handle currency, VARCHAR for strings, DATE for journey_date).
2. **Sample Data**:
    - Provides a starting point for testing all features without requiring manual data entry.
    - Covers various scenarios (e.g., different train types, a confirmed booking).
3. **Dynamic Column Addition**:
    - The script to add seat_number ensures the database schema is updated safely, supporting backward compatibility.
4. **Testing Queries**:
    - Comprehensive set of queries to verify data, test functionality, and demonstrate how to interact with the database.
5. **Security Considerations**:
    - The script uses plain-text passwords, which is insecure. In a real system, passwords should be hashed (e.g., using bcrypt).
    - The root user with a simple password (root) is a security risk; a dedicated user with a strong password should be used.

Potential Improvements

- **Add ON DELETE CASCADE**: To foreign keys in Bookings, AccountHolder, Profiles, and SessionLogs to automatically delete dependent records when a user is deleted.
- **Password Hashing**: Store hashed passwords in the Users table instead of plain text.
- **Indexes**: Add indexes on frequently queried columns (e.g., Bookings.pnr_number, Trains.train_number) to improve query performance.
- **Validation Constraints**: Add CHECK constraints (e.g., age >= 0 in Bookings and Profiles).
- **Audit Triggers**: Create triggers on Bookings to log changes (e.g., cancellations) in SessionLogs.

This script provides a solid foundation for the Train Management System, enabling all core functionalities while leaving room for enhancements.