

## 

## AIRLINE MANAGEMENT SYSTEM

## A MINI PROJECT REPORT

|  |  |
| --- | --- |
| **Submitted** | **By** |
| **AMALA ENCILIN T** | **231801011** |
|  |  |
|  |  |

### In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS) THANDALAM

CHENNAI-602105

2024 - 2025

# ABSTRACT

The **Airline Management System** is a robust Java-based application designed to streamline the operations involved in managing airline bookings, flights, and passenger information. This system employs Java for both front-end and back-end development, while utilizing MySQL as the database management system to ensure efficient data storage, retrieval, and management. By integrating these technologies through XAMPP, the project offers a reliable and interactive platform to handle tasks such as flight scheduling, passenger registration, booking management, and generating ticket details.

The application features a user-friendly interface that simplifies the process of data entry and retrieval for airline staff. Users can easily navigate through the system to manage flights, update passenger information, and process bookings efficiently. The backend logic automates critical operations, minimizing manual work and reducing the likelihood of errors. With robust database connectivity, the system ensures data persistence, integrity, and seamless integration.

Developed using tools such as Notepad, CMD, and XAMPP, this project highlights the versatility of Java in building standalone applications with integrated database functionality. It demonstrates a practical approach to software development, focusing on clean code design, efficient database operations, and a smooth user experience. This system serves as a scalable solution for airline operators, enabling them to effectively manage their operations with precision and ease.

# TABLE OF CONTENTS

1. **INTRODUCTION**
   1. [INTRODUCTION](#_TOC_250013) 4
   2. [OBJECTIVES](#_TOC_250012) 5
   3. [MODULES](#_TOC_250011) 6
2. SURVEY OF TECHNOLOGIES
   1. [SOFTWARE DESCRIPTION](#_TOC_250010) 8
3. REQUIREMENTS AND ANALYSIS
   1. [REQUIREMENT SPECIFICATION](#_TOC_250003) 10
   2. [HARDWARE AND SOFTWARE REQUIREMENTS](#_TOC_250002) 10
   3. [DATA DICTIONARY…](#_TOC_250001) 12
   4. [ER DIAGRAM](#_TOC_250000) 13
4. PROGRAM CODE 14
5. RESULTS AND DISCUSSIONS 22
6. CONCLUSION 30
7. REFERENCES 31

# INTRODUCTION

## INTRODUCTION

In the airline industry, managing bookings, flights, and passenger information is a critical yet complex task. Manual processes for handling flight schedules, passenger data, and ticketing are prone to errors, inefficiencies, and inconsistencies. To address these challenges, the **Airline Management System** is developed as an efficient and automated solution for handling airline operations.

This system is a standalone application built entirely using Java for both the front-end and back-end, with MySQL serving as the database for secure and structured data storage. The application allows users to manage flight schedules, register passengers, process bookings, and generate tickets. By leveraging XAMPP for database integration and connectivity, the project ensures seamless interaction between the application and the underlying database.

The **Airline Management System** emphasizes simplicity and usability, featuring an intuitive interface for effortless data entry and retrieval. It automates critical processes such as booking management and passenger data updates, thereby eliminating manual errors and reducing workload. Designed with scalability in mind, the system can adapt to the needs of various airline operations, from small-scale carriers to larger airlines.

This project not only showcases the power and versatility of Java in building real-world applications but also highlights the importance of database integration in ensuring data reliability and consistency. By focusing on effective design, functionality, and usability, the **Airline Management System** aims to enhance the efficiency of airline operations while providing a robust framework for managing flights, bookings, and passenger data.

**1.2 OBJECTIVES**

### **Primary Objectives:**

### ** Automate Airline Operations**: Provide a digital platform to manage flight schedules,

### bookings, and passenger information accurately and efficiently.

### ** Simplify Data Entry and Retrieval:** Enable users to input and manage flight details,

### passenger records, and booking information seamlessly through an intuitive graphical user

### interface.

### ** Dynamic Booking Management:** Automatically update seat availability, generate

### tickets, and process bookings in real-time.

### ** Visual Data Representation:** Display flight schedules, passenger lists, and booking

### statuses in an organized table format for better clarity and quick access.

### ** Database Integration:** Securely store all airline-related data in a MySQL database to ensure

### efficient, scalable, and reliable data management.

**Educational Objectives:**

1. **Promote Operational Efficiency**: Minimize manual tasks like flight scheduling and booking management, reducing errors and saving time.
2. **Enhance Passenger Management**: Facilitate airlines in maintaining accurate passenger records and improving customer service.
3. **Encourage Digital Transformation**: Build familiarity with digital platforms for managing airline operations and logistics.
4. **Support Modular Expansion**: Establish a foundation for adding advanced features like dynamic pricing, flight analytics, or real-time updates.
5. **Improve User Experience**: Ensure a smooth, user-friendly interface for navigating and managing airline functionalities.

* 1. **MODULES**

**Flight Management Module (Admin Role)**

* **Login & Dashboard**:
  + Secure access for authorized users to manage airline operations.
  + Display a summary of flights, bookings, and passenger data.
* **Flight Scheduling**:
  + Add, update, or delete flight schedules, including departure and arrival times, destinations, and flight statuses.
  + Validate input fields to ensure accurate schedule entry.
* **Passenger Management**:
  + Record passenger details such as name, contact information, and travel documents.
  + Prevent duplication by validating passenger IDs or booking references.
* **Booking Management**:
  + Facilitate ticket booking, cancellation, and modification.
  + Update seat availability dynamically and generate booking confirmations.
* **Ticket Generation**:
  + Automatically create digital tickets with flight details and passenger information.
  + Provide options to view, download, or email tickets.
* **Record Management**:
  + Maintain a comprehensive database of flights, passengers, and bookings.
  + Enable admins to add, update, or delete records efficiently.

**Display Module (Admin and Passenger Role)**

* **View Flights**:
  + Display all scheduled flights in an organized table format, including flight number, destination, departure time, and status.
  + Highlight flight statuses with specific colors for clarity (e.g., green for "On Time," yellow for "Delayed").
* **Booking Details**:
  + Provide an option for passengers or admins to view booking details, including passenger name, seat number, and ticket status.
* **Detailed Flight View**:
  + Include comprehensive flight information such as seat availability, crew details, and flight duration.
  + Offer a drill-down feature for viewing passenger manifests (admin role only).
* **Back Navigation**:
  + Provide intuitive navigation buttons to return to the dashboard or previous pages seamlessly.

**Database Module**

* **Flight Records**:
  + Securely store flight details, including flight numbers, destinations, schedules, and statuses, in a MySQL database.
  + Ensure unique identification for each flight using flight IDs to prevent conflicts.
* **Passenger Records**:
  + Maintain a comprehensive database of passenger information, including names, contact details, and travel documents.
  + Ensure unique booking references or IDs for each passenger record to avoid duplication.
* **Booking Records**:
  + Store booking details, such as flight ID, passenger ID, seat assignment, and ticket status.
  + Link bookings to corresponding flights and passengers for efficient data management.
* **Data Retrieval**:
  + Enable efficient retrieval of flight schedules, passenger details, and booking statuses for viewing, editing, or reporting.
  + Handle large data sets, ensuring high performance and minimal latency during searches or updates.

**Security Module**

* **Authentication System**:
  + Restrict access to the system's administrative functionality using secure login credentials.
  + Implement role-based access control (e.g., Admin for managing flights and bookings, Passengers for viewing tickets).
  + Protect sensitive data, such as passenger information and booking details, with robust authentication mechanisms.
* **Database Security**:
  + Safeguard stored flight, booking, and passenger data using secure SQL operations to prevent unauthorized access or SQL injection.
  + Encrypt sensitive information, such as passenger contact details and payment data, where applicable.
* **Error Handling**:
  + Ensure smooth operations by managing input errors (e.g., invalid flight times or duplicate passenger IDs) effectively.
  + Handle database connection issues gracefully, providing informative error messages without exposing sensitive system details.

# SURVEY OF TECHNOLOGIES

**2.1 SOFTWARE DESCRIPTION**

**2.1.1 Java**

Java is a versatile, object-oriented programming language widely known for its platform independence and robust performance. In this project, Java is the primary language used to develop both the backend logic and the graphical user interface (GUI). With its Swing library, Java provides the tools to create an intuitive and visually appealing interface, enabling efficient interaction between users and the system. Additionally, Java's JDBC (Java Database Connectivity) ensures seamless integration with the MySQL database, enabling reliable data storage and retrieval operations.

**2.1.2 MySQL**

MySQL is an open-source Relational Database Management System (RDBMS) designed for secure and efficient data handling. It plays a critical role in this project by storing student details, subject-wise marks, and corresponding grades in a structured manner. The system uses SQL queries for performing essential operations like inserting, updating, and retrieving data. MySQL ensures data integrity, reliability, and scalability, making it an ideal choice for managing the records of a Student Grading System. The database is managed locally using the XAMPP server for ease of use and accessibility during development.

**2.1.3 XAMPP**

XAMPP is an open-source, cross-platform server solution that simplifies the setup and management of a MySQL database server. In this project, XAMPP is used to host and manage the MySQL database locally, providing a convenient interface to configure and test database operations. It ensures smooth communication between the Java application and the database, enabling real-time updates and secure data handling. XAMPP's lightweight design and ease of use make it an essential tool for this project.

**2.1.4 Java Swing**

Java Swing is a GUI toolkit used to build interactive graphical user interfaces for desktop applications. In this project, Swing forms the foundation for creating user-friendly pages, such as the front page, grade input forms, and the grade display table. Swing provides prebuilt components like buttons, text fields, labels, and tables, which are styled and customized to enhance the user experience. Its ability to handle events and create dynamic interfaces ensures smooth navigation and functionality throughout the application.

**2.1.5 Command Prompt (CMD)**

The Command Prompt is a command-line interface used for compiling and executing Java programs in this project. It acts as the primary tool for running the application during the development phase, ensuring proper execution and debugging of code. The use of CMD ensures lightweight and straightforward operations without requiring heavy Integrated Development Environments (IDEs), aligning with the simplicity of the development environment.

**2.1.6 Notepad**

Notepad, a lightweight text editor, is used for writing the Java source code in this project. It serves as a basic tool for creating and editing the program files, ensuring a minimalistic yet functional development approach. Combined with CMD for execution, Notepad eliminates the dependency on specialized IDEs, demonstrating that robust applications can be developed with simple tools.

**III. REQUIREMENTS AND ANALYSIS**

**3.1 User Requirements:**

The Airline Management System is designed to simplify and automate the management of flight schedules, bookings, and passenger information. The system caters to both administrative staff and passengers, ensuring efficient operations and secure access. The primary requirements focus on effective data handling, an intuitive user interface, and seamless communication between the frontend and backend.

**Key Features:**

* Login and Authentication:
  + Role-based access control for administrators and passengers to ensure secure operations.
* Flight Management:
  + Administrators can add, update, and view flight schedules and statuses.
* Passenger and Booking Management:
  + Handle passenger records and manage bookings, cancellations, and ticket modifications with ease.
* Database Integration:
  + All flight, booking, and passenger data is securely stored in a MySQL database for efficient retrieval and reporting.
* Dynamic Seat Allocation:
  + Automatic updating of seat availability and ticket generation during the booking process.
* User-Friendly Interface:
  + Interactive GUI built using Java Swing for enhanced usability and seamless navigation.

**3.2 Hardware and Software Requirements**

**Software Requirements**

* **Operating System**:
  + Windows 10 or higher.
* **Programming Language**:
  + Java (JDK 8 or higher).
* **Frontend**:
  + Java Swing for the graphical user interface.
* **Backend**:
  + MySQL for database operations.
* **Database Management System (DBMS)**:
  + MySQL 8.0 or higher, hosted locally using XAMPP.
* **Development Tools**:
  + **Text Editor**: Notepad or any other lightweight code editor for writing source code.
  + **Command Line**: CMD for compiling and running the application.
* **Libraries/Packages**:
  + Java JDBC for database connectivity.

**Hardware Requirements**

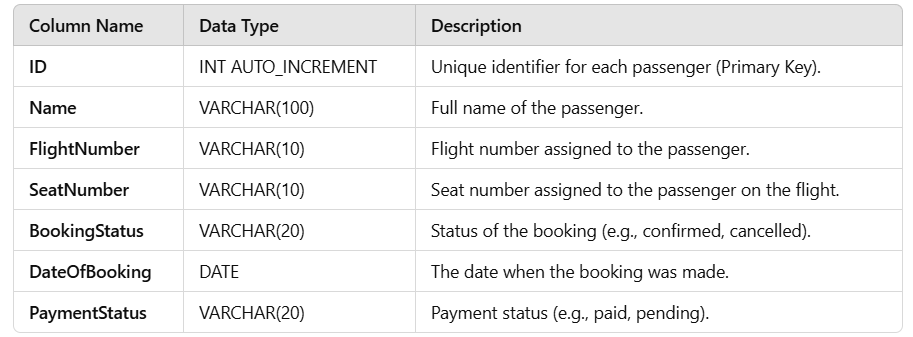
* **System Type**:
  + Desktop PC or Laptop.
* **Operating System**:
  + Windows 10 or higher.
* **Processor**:
  + Intel® Core™ i3-6100 or higher (or equivalent AMD processor).
* **RAM**:
  + 4 GB or higher for smooth operation.
* **Storage**:
  + 200 MB available disk space (for application files and database).
* **Display**:
  + Monitor with a minimum resolution of 1280 x 720 pixels.
* **Input Devices**:
  + Standard Keyboard and Mouse for user interaction.

**3.3 DATA DICTIONARY**

**1. Grading Table**

**Table Name:** students

**Description:**  
This table stores the details of students, their individual subject marks, and the overall grade.

**  
Usage:**

 **ID**: A unique identifier for each passenger (Primary Key).

 **Name**: Full name of the passenger.

 **Flight Number**: The flight number assigned to the passenger.

 **Seat Number**: The seat number assigned to the passenger on the flight.

 **Booking Status**: Indicates whether the booking is confirmed, cancelled, or pending.

 **Date Of Booking**: The date when the passenger made the booking.

 **Payment Status**: Indicates the payment status of the ticket (paid, pending).

**IV. PROGRAM CODE:**

**DatabaseConnection:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

public static Connection getConnection() throws SQLException {

try {

// Load the JDBC driver

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

return DriverManager.getConnection("jdbc:mysql://localhost:3308/AirlineDB", "root", "");

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

throw new SQLException("Error while connecting to the database");

}

}

}

**HomeFrame:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class HomeFrame extends JFrame {

private String username;

public HomeFrame(String username) {

this.username = username;

// Frame title

setTitle("Airline Management System - Home");

// Main Panel with background color and padding

JPanel mainPanel = new JPanel() {

// Custom paintComponent to add a background gradient

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

Color color1 = new Color(240, 248, 255);

Color color2 = new Color(0, 102, 204);

GradientPaint gp = new GradientPaint(0, 0, color1, 0, getHeight(), color2);

g2d.setPaint(gp);

g2d.fillRect(0, 0, getWidth(), getHeight());

}

};

mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));

mainPanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 30));

// Title label with custom font and alignment

JLabel titleLabel = new JLabel("Welcome to Airline Management");

titleLabel.setFont(new Font("Arial", Font.BOLD, 24));

titleLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

titleLabel.setForeground(Color.BLACK);

// Subtitle label for personalized welcome

JLabel welcomeLabel = new JLabel("Hello, " + username);

welcomeLabel.setFont(new Font("Arial", Font.PLAIN, 18));

welcomeLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

welcomeLabel.setForeground(Color.YELLOW);

// Panel for buttons with GridBagLayout for alignment

JPanel buttonPanel = new JPanel(new GridBagLayout());

buttonPanel.setOpaque(false); // Transparent to show gradient background

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(15, 15, 15, 15); // Space around buttons

// Book Flight Button

JButton bookFlightButton = new JButton("Book a Flight");

styleButton(bookFlightButton);

gbc.gridx = 0;

gbc.gridy = 0;

buttonPanel.add(bookFlightButton, gbc);

bookFlightButton.addActionListener(e -> new BookingFrame(username).setVisible(true));

// Cancel Booking Button

JButton cancelBookingButton = new JButton("Cancel Booking");

styleButton(cancelBookingButton);

gbc.gridy = 1;

buttonPanel.add(cancelBookingButton, gbc);

cancelBookingButton.addActionListener(e -> new CancelBookingPage(username).setVisible(true));

// Logout Button

JButton logoutButton = new JButton("Logout");

styleButton(logoutButton);

gbc.gridy = 2;

buttonPanel.add(logoutButton, gbc);

logoutButton.addActionListener(e -> {

new LoginFrame().setVisible(true);

dispose();

});

// Add components to the main panel

mainPanel.add(titleLabel);

mainPanel.add(Box.createRigidArea(new Dimension(0, 20))); // Spacer

mainPanel.add(welcomeLabel);

mainPanel.add(Box.createRigidArea(new Dimension(0, 30))); // Spacer

mainPanel.add(buttonPanel);

// Set up the frame

add(mainPanel);

setSize(450, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null); // Center the frame

setResizable(false);

}

// Helper method to style buttons with rounded edges, shadow effect, and improved colors

private void styleButton(JButton button) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(new Color(0, 153, 255)); // Brighter blue background

button.setForeground(Color.WHITE); // White text

button.setFocusPainted(false); // Remove focus border

button.setPreferredSize(new Dimension(250, 45)); // Set button size

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(0, 102, 204), 2),

BorderFactory.createEmptyBorder(10, 20, 10, 20)

));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR)); // Change cursor on hover

}

// Main method to run the home frame (for testing)

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new HomeFrame("user123").setVisible(true); // Example username

});

}

}

**BookingFrame:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public class BookingFrame extends JFrame {

private String username;

public BookingFrame(String username) {

this.username = username;

setTitle("Book Flight - Airline Management System");

// Custom main panel with gradient background

GradientPanel mainPanel = new GradientPanel();

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

mainPanel.setLayout(new BorderLayout());

// Title Label at the top

JLabel titleLabel = new JLabel("Book Your Flight");

titleLabel.setFont(new Font("Arial", Font.BOLD, 20));

titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

titleLabel.setForeground(new Color(255, 255, 255)); // White color for contrast

mainPanel.add(titleLabel, BorderLayout.NORTH);

// Form panel for inputs

JPanel formPanel = new JPanel();

formPanel.setLayout(new GridBagLayout());

formPanel.setOpaque(false); // Make form panel transparent

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(10, 10, 10, 10);

gbc.fill = GridBagConstraints.HORIZONTAL;

// Source Place

JLabel sourceLabel = new JLabel("Source Place:");

JTextField sourceField = new JTextField();

styleLabel(sourceLabel);

styleTextField(sourceField);

addToFormPanel(formPanel, gbc, sourceLabel, sourceField, 0);

// Destination Place

JLabel destinationLabel = new JLabel("Destination Place:");

JTextField destinationField = new JTextField();

styleLabel(destinationLabel);

styleTextField(destinationField);

addToFormPanel(formPanel, gbc, destinationLabel, destinationField, 1);

// Flight Date

JLabel dateLabel = new JLabel("Flight Date (YYYY-MM-DD):");

JTextField dateField = new JTextField();

styleLabel(dateLabel);

styleTextField(dateField);

addToFormPanel(formPanel, gbc, dateLabel, dateField, 2);

// Flight Time

JLabel timeLabel = new JLabel("Flight Time (HH:MM):");

JTextField timeField = new JTextField();

styleLabel(timeLabel);

styleTextField(timeField);

addToFormPanel(formPanel, gbc, timeLabel, timeField, 3);

// Book Button at the bottom

JButton bookButton = new JButton("Book Flight");

styleButton(bookButton);

bookButton.addActionListener(e -> handleBooking(sourceField, destinationField, dateField, timeField));

// Adding form and button to the main panel

mainPanel.add(formPanel, BorderLayout.CENTER);

mainPanel.add(bookButton, BorderLayout.SOUTH);

// Set up the frame

add(mainPanel);

setSize(450, 400);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

}

private void handleBooking(JTextField sourceField, JTextField destinationField, JTextField dateField, JTextField timeField) {

String source = sourceField.getText();

String destination = destinationField.getText();

String date = dateField.getText();

String time = timeField.getText();

if (source.isEmpty() || destination.isEmpty() || date.isEmpty() || time.isEmpty()) {

JOptionPane.showMessageDialog(this, "Please fill in all fields.");

return;

}

try {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm");

java.sql.Date flightDate = new java.sql.Date(dateFormat.parse(date).getTime());

java.sql.Time flightTime = new java.sql.Time(timeFormat.parse(time).getTime());

Connection conn = DatabaseConnection.getConnection();

String query = "INSERT INTO bookings (username, source\_place, destination\_place, flight\_date, flight\_time) VALUES (?, ?, ?, ?, ?)";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, username);

stmt.setString(2, source);

stmt.setString(3, destination);

stmt.setDate(4, flightDate);

stmt.setTime(5, flightTime);

stmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Flight booked successfully!");

dispose();

} catch (ParseException ex) {

JOptionPane.showMessageDialog(this, "Invalid date or time format. Please use YYYY-MM-DD for date and HH:MM for time.");

} catch (SQLException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());

}

}

private void addToFormPanel(JPanel panel, GridBagConstraints gbc, JLabel label, JTextField field, int row) {

gbc.gridx = 0;

gbc.gridy = row;

panel.add(label, gbc);

gbc.gridx = 1;

gbc.weightx = 1.0;

panel.add(field, gbc);

}

private void styleLabel(JLabel label) {

label.setFont(new Font("Arial", Font.PLAIN, 16));

label.setForeground(new Color(255, 255, 255)); // White for visibility on gradient

}

private void styleTextField(JTextField textField) {

textField.setFont(new Font("Arial", Font.PLAIN, 16));

textField.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(255, 255, 255), 1),

BorderFactory.createEmptyBorder(5, 5, 5, 5)

));

textField.setPreferredSize(new Dimension(250, 30));

}

private void styleButton(JButton button) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(new Color(0, 123, 255));

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setPreferredSize(new Dimension(200, 40));

button.setBorder(BorderFactory.createLineBorder(new Color(255, 255, 255), 2));

}

// Custom panel class for gradient background

class GradientPanel extends JPanel {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

int width = getWidth();

int height = getHeight();

Color color1 = new Color(0, 102, 204); // Dark blue

Color color2 = new Color(153, 204, 255); // Light blue

GradientPaint gp = new GradientPaint(0, 0, color1, 0, height, color2);

g2d.setPaint(gp);

g2d.fillRect(0, 0, width, height);

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new BookingFrame("user123").setVisible(true));

}

}

CancelPage:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

public class CancelBookingPage extends JFrame {

private JTextField bookingIdField;

private JButton cancelButton, backButton;

public CancelBookingPage(String username) {

setTitle("Cancel Booking");

setSize(450, 300); // Adjusted size for more space

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Set up the panel and components

JPanel cancelPanel = new JPanel();

cancelPanel.setLayout(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();

// Set the background color of the cancel panel to match the home UI background color

cancelPanel.setBackground(new Color(255, 255, 255)); // Light background similar to Home UI

// Add padding around content

cancelPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

// Title Label with Font and Color

JLabel titleLabel = new JLabel("Cancel Your Booking");

titleLabel.setFont(new Font("Arial", Font.BOLD, 24));

titleLabel.setForeground(new Color(0, 102, 204)); // Blue color for consistency with buttons

gbc.gridx = 0;

gbc.gridy = 0;

gbc.gridwidth = 2;

gbc.insets = new Insets(0, 0, 20, 0); // Space below the title

gbc.anchor = GridBagConstraints.CENTER;

cancelPanel.add(titleLabel, gbc);

// Booking ID Label

JLabel bookingIdLabel = new JLabel("Enter Booking ID:");

bookingIdLabel.setFont(new Font("Arial", Font.PLAIN, 16));

bookingIdLabel.setForeground(new Color(0, 102, 204)); // Matching color with the title

gbc.gridx = 0;

gbc.gridy = 1;

gbc.gridwidth = 1;

gbc.insets = new Insets(10, 0, 5, 0); // Space between elements

cancelPanel.add(bookingIdLabel, gbc);

// Create a JPanel to hold the booking ID label and field together

JPanel bookingPanel = new JPanel();

bookingPanel.setLayout(new BorderLayout());

bookingPanel.setBackground(new Color(255, 255, 255)); // Light background to match home UI

// Create a JTextField for Booking ID with custom style to match other boxes

bookingIdField = new JTextField();

bookingIdField.setFont(new Font("Arial", Font.PLAIN, 16));

bookingIdField.setPreferredSize(new Dimension(300, 40)); // Adjusted size for readability

bookingIdField.setBorder(BorderFactory.createLineBorder(new Color(0, 102, 204), 2)); // Add border with color

bookingIdField.setBackground(new Color(240, 248, 255)); // Light background similar to other input fields

bookingIdField.setEditable(true); // Ensure the text field is editable

bookingIdField.requestFocusInWindow(); // Set focus to the text field on window open

// Add the JTextField to the JPanel

bookingPanel.add(bookingIdField, BorderLayout.CENTER);

// Adjust GridBagConstraints for the JTextField to make it responsive

gbc.gridx = 1;

gbc.gridy = 1;

gbc.fill = GridBagConstraints.HORIZONTAL; // Allows the text field to stretch horizontally

gbc.weightx = 1.0; // Expands horizontally within the available space

gbc.insets = new Insets(10, 0, 5, 0); // Space between elements

cancelPanel.add(bookingPanel, gbc);

// Cancel Button with Styling

cancelButton = new JButton("Cancel Booking");

styleButton(cancelButton);

gbc.gridx = 0;

gbc.gridy = 2;

gbc.gridwidth = 2;

gbc.insets = new Insets(20, 0, 10, 0); // Space around button

cancelPanel.add(cancelButton, gbc);

// Action listener for the Cancel button

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

int bookingId = Integer.parseInt(bookingIdField.getText());

cancelBooking(bookingId);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(CancelBookingPage.this, "Please enter a valid booking ID.");

}

}

});

// Back to Home Button with Styling

backButton = new JButton("Back to Home");

styleButton(backButton);

gbc.gridx = 0;

gbc.gridy = 3;

gbc.gridwidth = 2;

gbc.insets = new Insets(10, 0, 0, 0); // Space around button

cancelPanel.add(backButton, gbc);

// Action listener for the Back button

backButton.addActionListener(e -> {

new HomeFrame(username).setVisible(true);

dispose(); // Close current page

});

// Add panel to frame

add(cancelPanel);

}

// Method to style buttons with rounded edges, shadow effect, and improved colors

private void styleButton(JButton button) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(new Color(0, 102, 204)); // Blue background

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setPreferredSize(new Dimension(250, 45));

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(0, 102, 204), 2),

BorderFactory.createEmptyBorder(10, 20, 10, 20)

));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR)); // Change cursor on hover

// Add Hover effect

button.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

button.setBackground(new Color(0, 153, 255)); // Lighter blue on hover

}

public void mouseExited(java.awt.event.MouseEvent evt) {

button.setBackground(new Color(0, 102, 204)); // Original color

}

});

}

// Method to cancel the booking by booking ID

private void cancelBooking(int bookingId) {

try {

// Establish connection to the database

Connection conn = DatabaseConnection.getConnection();

// Check if the booking exists in the database

String checkQuery = "SELECT \* FROM bookings WHERE booking\_id = ?";

PreparedStatement checkStmt = conn.prepareStatement(checkQuery);

checkStmt.setInt(1, bookingId);

ResultSet rs = checkStmt.executeQuery();

if (rs.next()) {

// Booking exists, now cancel it by updating the status to 'canceled'

String cancelQuery = "UPDATE bookings SET status = 'canceled' WHERE booking\_id = ?";

PreparedStatement cancelStmt = conn.prepareStatement(cancelQuery);

cancelStmt.setInt(1, bookingId);

cancelStmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Booking ID " + bookingId + " canceled successfully.");

} else {

JOptionPane.showMessageDialog(this, "Booking ID not found.");

}

} catch (SQLException e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage());

}

}

public static void main(String[] args) {

// Run the cancel booking page

SwingUtilities.invokeLater(() -> {

new CancelBookingPage("user123").setVisible(true);

});

}

}

**LoginFram:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

public class LoginFrame extends JFrame implements ActionListener {

private JTextField usernameField;

private JPasswordField passwordField;

private JButton loginButton;

public LoginFrame() {

setTitle("Login");

// Create main panel with padding and background color

JPanel mainPanel = new JPanel();

mainPanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 30)); // Add padding around the form

mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));

mainPanel.setBackground(new Color(240, 248, 255)); // Light background color

// Title label with custom font

JLabel titleLabel = new JLabel("Airline Management Login");

titleLabel.setFont(new Font("Arial", Font.BOLD, 20));

titleLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

titleLabel.setForeground(new Color(0, 123, 255)); // Blue color

// Form Panel using GridBagLayout for better control over alignment

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

formPanel.setBackground(new Color(240, 248, 255)); // Match background color

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(10, 10, 10, 10); // Spacing between components

// Username Label and Field

gbc.gridx = 0;

gbc.gridy = 0;

gbc.anchor = GridBagConstraints.EAST;

formPanel.add(new JLabel("Username:"), gbc);

usernameField = new JTextField(15);

gbc.gridx = 1;

gbc.anchor = GridBagConstraints.WEST;

formPanel.add(usernameField, gbc);

// Password Label and Field

gbc.gridx = 0;

gbc.gridy = 1;

gbc.anchor = GridBagConstraints.EAST;

formPanel.add(new JLabel("Password:"), gbc);

passwordField = new JPasswordField(15);

gbc.gridx = 1;

gbc.anchor = GridBagConstraints.WEST;

formPanel.add(passwordField, gbc);

// Login Button with custom styling

loginButton = new JButton("Login");

loginButton.setFont(new Font("Arial", Font.BOLD, 16));

loginButton.setBackground(new Color(0, 123, 255)); // Blue color

loginButton.setForeground(Color.WHITE);

loginButton.setFocusPainted(false); // Remove focus border

loginButton.setPreferredSize(new Dimension(150, 40)); // Consistent button size

loginButton.setAlignmentX(Component.CENTER\_ALIGNMENT); // Center-align the button

loginButton.addActionListener(this);

// Add components to main panel

mainPanel.add(titleLabel);

mainPanel.add(Box.createRigidArea(new Dimension(0, 20))); // Spacer

mainPanel.add(formPanel);

mainPanel.add(Box.createRigidArea(new Dimension(0, 20))); // Spacer

mainPanel.add(loginButton);

// Set up the frame

add(mainPanel);

setSize(400, 300);

setLocationRelativeTo(null); // Center the frame

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setResizable(false);

}

@Override

public void actionPerformed(ActionEvent e) {

String username = usernameField.getText();

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

try {

Connection conn = DatabaseConnection.getConnection();

String query = "SELECT \* FROM users WHERE username = ? AND password = ?";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, username);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

JOptionPane.showMessageDialog(this, "Login Successful!");

new HomeFrame(username).setVisible(true);

dispose(); // Close login window

} else {

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

}

} catch (SQLException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new LoginFrame().setVisible(true);

});

}

}

**Compile each file:**

javac DatabaseConnection.java

javac StudentOperations.java

javac StudentGradingSystem.java

javac FrontPage.java

javac StudentGradingUI.java

javac DisplayGradesUI.java

javac -cp .;"C:\javamini\mysql-connector-j-9.1.0.jar" \*.java

**Run the main program:**

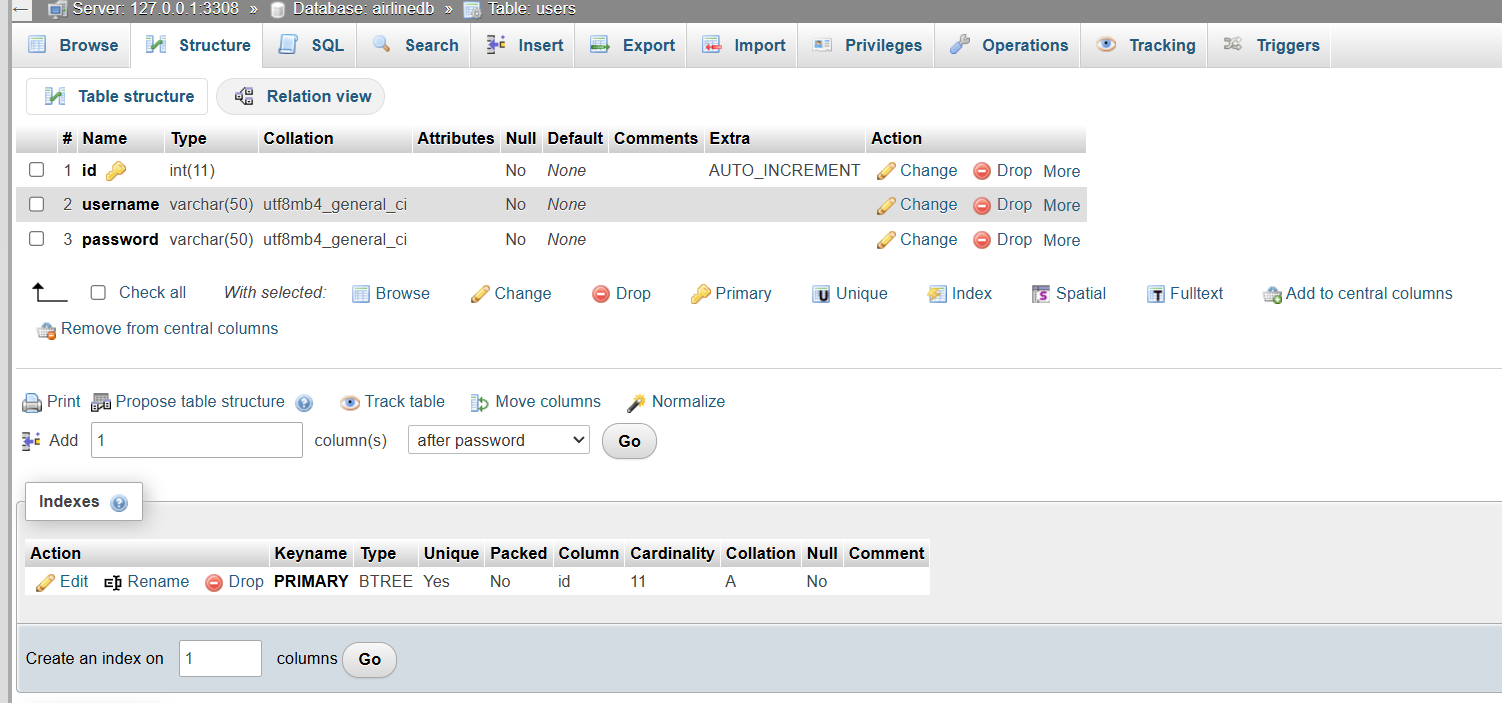
java -cp .;"C:\javamini\mysql-connector-j-9.1.0.jar" FrontPage

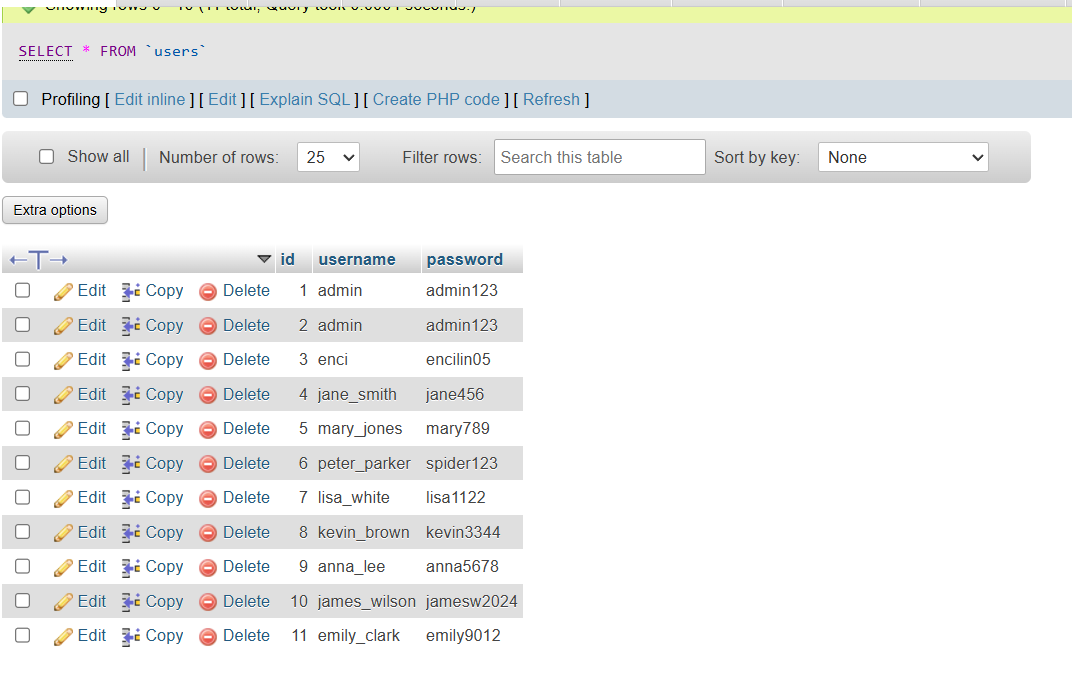
# V. RESULT AND DISCUSSION

# DATABASE

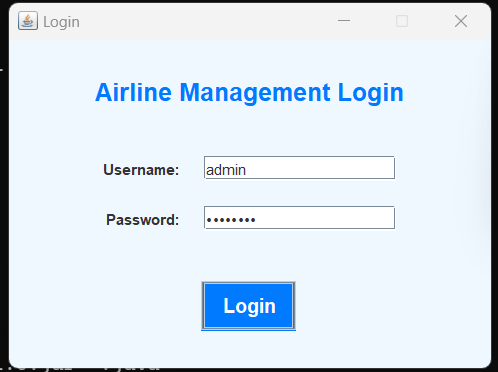
# 

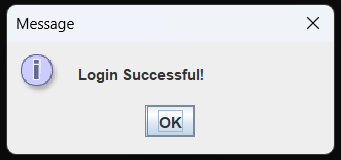
# 

****

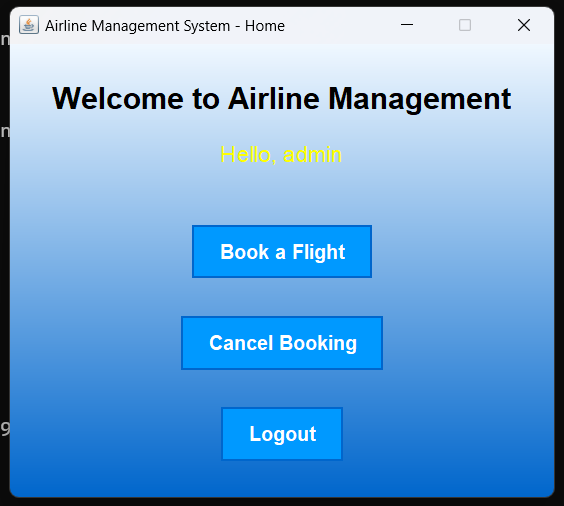
****

**LOGIN PAGE:**

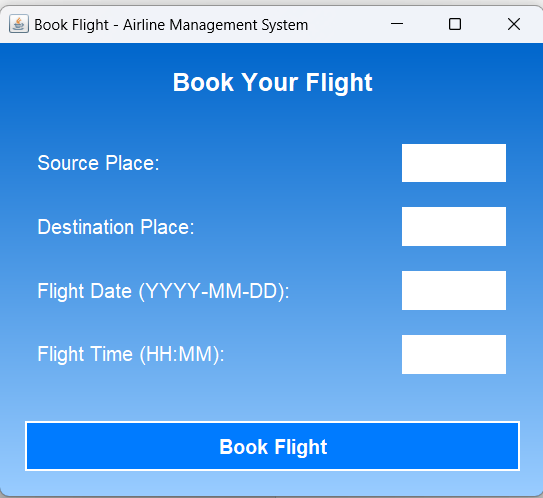
****

****

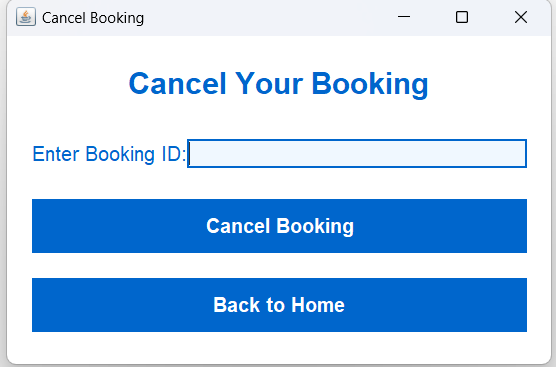
**HOME PAGE**



**BOOK** **PAGE:**



**CANCEL PAGE:**



# RESULTS

1. **User Features:**
   * **Login System:** A basic login system has been implemented for both passengers and airline staff to access their respective functionalities. After logging in, users are directed to their dashboards based on their role.
   * **Passenger Dashboard:** Passengers can view their booking status, flight details, seat assignments, and payment status.
   * **Staff Dashboard:** Airline staff can add new passenger bookings, update flight information, and view booking history for all passengers.
2. **Functionalities:**
   * **Booking Management:** Passengers can view available flights, select seats, and make payments. Airline staff can add, update, or cancel bookings.
   * **Flight Schedule:** The system allows for viewing, adding, or modifying flight schedules, including departure times, destinations, and available seats.
   * **Database Management:** All booking details, including passenger data, flight schedules, and seat assignments, are stored and retrieved from a MySQL database for persistent data management.
3. **Database Interaction:**
   * **Passenger Records:** The MySQL database table (passengers) is used to store passenger details, booking status, and flight assignments.
   * **Dynamic Updates:** Updates, such as booking new tickets or modifying flight schedules, are immediately reflected in the database.
   * **Error Handling:** The system includes error handling for invalid data entry, such as missing required fields or duplicate records.
4. **User Interface (UI):**
   * **Simple and Intuitive Design:** A user-friendly design has been created to ensure ease of navigation for both passengers and airline staff.
   * **Interactive Frontend:** Developed with Java Swing, the frontend includes buttons and input fields for booking flights and viewing flight statuses.
   * **Custom Branding:** Airline branding and logos have been incorporated to give the system a professional look.
5. **Performance:**
   * **Real-Time Feedback:** Operations such as booking a flight, updating a schedule, and displaying passenger details are executed in real-time, providing instant feedback to the user.
6. **Security Considerations:**
   * **Error Validation:** Input validation ensures that only valid data is entered, such as restricting seat selections to available seats and ensuring the accuracy of flight numbers.
   * **Basic User Authentication:** A login system with basic authentication is in place, though password encryption is not yet implemented in this version.
   * **Data Integrity:** Database constraints ensure that records are unique and prevent issues like duplicate passenger bookings.
7. **Next Steps for Improvement:**
   * **Password Security:** Implement password hashing to improve login security using Java libraries (e.g., BCrypt).
   * **Role-Based Permissions:** Enhance user roles for more granular access control (e.g., allowing airline staff to modify flight schedules but restricting passengers).
   * **Enhanced Validation:** Improve validation for seat availability and booking limits.
   * **UI Enhancements:** Upgrade the design using JavaFX for a more modern and responsive interface.
   * **Reports Generation:** Introduce downloadable flight booking reports for administrative purposes.
   * **Session Management:** Implement session-based login to enhance security and improve the user experience.

# DISCUSSION

 User Experience:

* Strengths:
  + Intuitive Interface: The Airline Management System offers a simple, easy-to-navigate interface, allowing both passengers and staff to interact without needing technical expertise.
  + Efficient Booking Process: The system streamlines the booking process, making it easier for passengers to reserve seats, make payments, and check booking status.
  + Real-Time Updates: Changes such as flight updates or booking cancellations are reflected instantly in the system, offering a smooth user experience.
* Areas for Improvement:
  + Enhanced Interactivity: Adding drop-down menus for flight selection, auto-complete for passenger names, or seat selection could reduce errors during data entry.
  + UI Aesthetics: The design could benefit from a more modern look, with improved color schemes and the addition of icons for better clarity.
  + Mobile Compatibility: Optimizing the system for mobile devices would help users access the system more conveniently on smartphones or tablets.

 Database Integration:

* Strengths:
  + Efficient Data Handling: MySQL provides a robust database to store passenger information, flight schedules, and seat bookings.
  + Dynamic Data Interaction: Data changes are updated in real-time, ensuring that passengers and staff always view the latest booking details.
  + Real-Time Operations: The system operates dynamically with real-time flight booking and passenger record updates.
* Areas for Improvement:
  + Advanced Data Validation: Improved validation for booking limits, such as ensuring passengers cannot book a seat if the flight is already full.
  + Optimized Queries: With larger data sets, optimized database queries could improve performance, especially when retrieving flight schedules or booking histories.
  + Database Relationships: Adopting a relational database design with separate tables for passengers, flights, and bookings could improve modularity and scalability.

 Security Considerations:

* Strengths:
  + Role-Based Access: The system has distinct user roles for passengers and staff, limiting access to sensitive functionalities and ensuring secure interactions.
  + Error Handling: The application ensures that invalid or incomplete data does not enter the system, displaying error messages to guide users.
* Areas for Improvement:
  + Password Security: Currently, passwords are stored in plain text. Implementing encryption like bcrypt or another secure hashing algorithm would significantly improve login security.
  + Secure Communication: Adding SSL/TLS encryption for communication between the application and database would enhance confidentiality.
  + Two-Factor Authentication: Adding two-factor authentication (2FA) for logging in would add an extra layer of protection, especially for airline staff accessing sensitive data.

 Performance:

* Strengths:
  + Smooth Functionality: The system handles common use cases well, with fast response times for booking flights and updating records.
  + Minimal Latency: The application responds quickly to user inputs and database queries, offering a seamless experience.
* Areas for Improvement:
  + Scalability: As the system scales to handle more passengers and flights, introducing caching and optimizing queries will enhance performance.
  + Stress Testing: Conducting stress tests with multiple users making bookings at the same time will help identify potential bottlenecks.
  + Performance Monitoring: Adding tools to monitor database performance and track the system’s responsiveness will ensure that performance issues are addressed proactively.

 Future Enhancements:

* User Experience:
  + Improve passenger search features and enhance the booking process with a more interactive interface.
  + Implement downloadable flight booking and payment confirmation reports for both passengers and airline staff.
  + Make the system mobile-friendly to ensure passengers can manage bookings on the go.
* Admin Efficiency:
  + Extend the dashboard for airline staff with analytics features, such as seat occupancy and flight performance reports.
  + Add an administrator role for full system management, including the ability to oversee all bookings, users, and flight schedules.
* Security:
  + Implement stronger user authentication methods, such as two-factor authentication, to safeguard user accounts.
  + Use encryption for sensitive data stored in the database, particularly for user credentials and booking information.
  + Conduct regular vulnerability assessments to ensure the system is secure against potential threats.
* Scalability:
  + Normalize the database schema to handle a growing number of passengers and flights effectively.
  + Consider using cloud-based database solutions for improved scalability, performance, and data redundancy.

**VI. CONCLUSION**

The **Airline Management System** has successfully addressed core features like booking management, real-time updates, role-based access, and efficient database interactions. While the system is fully functional and user-friendly, there are areas for improvement, particularly in terms of security, scalability, and user interface enhancements. By focusing on optimizing the system for larger datasets, implementing better security measures, and enhancing the user experience, the platform can evolve into a comprehensive solution for managing flight bookings and passenger records in larger airline organizations.

# VII. REFERENCES

1. Jain, R. (2021). Designing a student grading system using SQL. International Journal ofComputer Science and Information Technologies*, 12*(2), 145-150.
2. Singh, A., & Sharma, P. (2020). Database design for student management and grading systems. Journal of Computer Science and Applications, *8*(4), 55-60.
3. Kumar, S., & Meena, S. (2019). Student information management system with grading automation. International Journal of Engineering and Technology, *10*(5), 1201-1210.
4. Patel, S. (2022). Developing a web-based grading system using MySQL and PHP. InternationalJournal of Web Applications*, 16*(3), 98-104.
5. Verma, P., & Yadav, R. (2018). Designing an efficient student grading system with MySQL and Java. International Journal of Software Engineering and Applications, *9*(1), 49-56.