TABLE OF CONTENTS

ABSTRACT

The Airline Reservation Management System (ARMS) is a comprehensive software solution designed to streamline and optimize the process of booking, managing, and tracking airline reservations. This system automates the complex workflows involved in passenger booking, flight scheduling, ticketing, and seat allocation, thereby reducing manual effort and minimizing human error. Built with a user-friendly interface, ARMS enables both customers and airline staff to access essential features,

such as real-time flight status updates, seat availability, and automated fare calculations. The system integrates with a MySQL database to securely store and retrieve passenger data, flight information, and transaction details, ensuring data consistency and integrity. Enhanced by Java-based backend functionality, ARMS provides robust performance and scalability to accommodate high volumes of concurrent bookings. By implementing ARMS, airlines can improve operational efficiency, increase customer satisfaction, and maintain a high level of accuracy in their reservation services.

PROBLEM STATEMENT

The traditional airline reservation system has long been a foundational component of the travel industry, serving as a means of booking flights, managing customer information, and handling ticketing processes. However, as technology rapidly advances, the limitations of these conventional systems have become more apparent. Traditional reservation methods often rely on manual data entry, in-person booking channels, and siloed systems that require significant time and resources from both customers and airline staff. These legacy systems restrict booking and reservation management to specific times and locations, limiting flexibility for travelers who may need access to booking and flight changes from anywhere. Moreover, manual data entry and processing increase the potential for human error, affecting booking accuracy and customer satisfaction. In an increasingly digital world, the demand for more accessible, flexible, and efficient reservation systems has grown significantly, particularly for an online airline reservation application capable of meeting the diverse needs of both airlines and travelers.

Historically, airline reservations were confined to physical locations, like ticketing offices or travel agencies, which restricted the accessibility and adaptability of booking options. Traditional systems demanded extensive preparation, including manual data input, phone-based customer service, and in-person booking confirmations. For airline staff managing high volumes of bookings, this process could be time-consuming and lead to delays in confirming customer reservations. As digital technology began to emerge in the early 2000s, some airlines implemented computerized reservation systems to reduce these inefficiencies. However, these early solutions often lacked user-friendly interfaces and offered limited functionality; moreover, they still required customers to be physically present in specific booking locations. These challenges underscored the need for a reservation system that could provide real-time availability, remote access, and an easy-to-use interface.

In recent years, many airlines have adopted online booking platforms, often integrated with large travel websites, but these platforms come with their own challenges and limitations. Although they offer remote booking options, they are generally designed as comprehensive travel booking tools rather than dedicated airline reservation systems, making them complex and sometimes challenging for users only seeking basic flight reservations. Additionally, while some platforms offer basic booking confirmation features, they often lack real-time updates, support for rapid schedule changes, or user-friendly interfaces that cater to the needs of both airlines and customers. Current platforms also struggle to offer an engaging, seamless booking experience that keeps travelers informed and empowered throughout the process. These gaps reveal the need for a dedicated airline reservation system that prioritizes accessibility, usability, and flexibility for both airline staff and travelers.

The goal of our project is to address these challenges by developing a dedicated airline reservation application that provides a streamlined, flexible, and user-friendly experience for both airline staff and passengers. This application aims to reduce the workload on airline employees by offering automated booking management, real-time seat availability updates, and customizable flight options tailored to diverse passenger

needs. Additionally, the application will enable airlines to manage reservations, track passenger preferences, and make adjustments in real time, helping identify booking trends and enhancing customer satisfaction. Unlike broader travel platforms, our application will focus specifically on delivering an intuitive reservation experience that prioritizes functionality and ease of use for both ends. Through these features, we aim to support a more efficient and accessible approach to travel booking, allowing passengers to access flight information anytime, anywhere, while facilitating a more streamlined workflow for airline staff. By incorporating innovative features and a user-friendly design, our online airline reservation application is positioned to become a valuable tool in modernizing airline services and meeting the evolving demands of today's travel industry.

METHODOLOGY

In developing this airline reservation system, a structured methodology was adopted to ensure seamless functionality across essential features, including user authentication, MySQL database integration, flight search, booking management, and ticket generation. The system also includes a flight history feature to allow users to track their bookings. This methodology is divided into several phases: requirements gathering, database setup, backend logic for reservation flow, feedback mechanisms, and flight history functionality. Below is a breakdown of each phase.

1. Requirements Gathering and Analysis

The first phase involved identifying and analyzing the specific requirements for the airline reservation system to ensure a comprehensive understanding of its features.

This included the need for a MySQL database to manage flight details, reservations, user accounts, and flight history.

- **Output**: A detailed list of requirements, including user login functionality, flight search, booking, ticket issuance, and flight history tracking.
- **Pros**:
    - Ensures all essential features are defined before development.
    - Minimizes the risk of missing core functionality.
- **Cons**:
    - Requirements may need adjustments if the scope evolves during development.
    - Collecting feedback may take time if multiple stakeholders are involved.

2. Database Design and Integration

The database was designed and integrated using MySQL to store flight schedules, user accounts, reservation details, and flight history. This structure supports searching available flights, storing bookings, and retrieving user history.

- **Output**: A MySQL database with tables for users, flights, reservations, and flight history.
- **Pros**:
    - Enables efficient storage and retrieval of flight and booking data.
    - Supports data persistence, ensuring user bookings and history are retained across sessions.
- **Cons**:
    - Requires efficient query optimization to maintain quick response times.
    - Needs regular maintenance, particularly for updating flight schedules and handling bookings.

3. Backend Logic for Reservation Flow

The backend logic, written in Java, manages the main reservation functionality. This includes flight searches, seat selection, booking confirmation, and ticket generation. The sequence follows:

- The application prompts the user to log in or register.
- Users can search for available flights based on dates and destinations.
- After selecting a flight, the application presents seat availability and booking options.
- Upon confirmation, a booking record is created, and a ticket is generated.
- **Output**: A dynamic reservation flow that handles each booking step and generates a ticket.
- **Pros**:
    - Enhances user experience by guiding them through each step of the reservation process.
    - Immediate ticket generation adds convenience.
- **Cons**:
    - Handling multiple bookings simultaneously may cause performance issues if not optimized.
    - Requires robust error handling for cases like overbooking or flight cancellations.

4. Ticket Generation and Booking Confirmation

Once a reservation is confirmed, the application generates an electronic ticket and displays booking details for the user. This ticket includes flight information, seat number, and booking reference. The confirmation also triggers an update in the flight's seat availability.

- **Output**: A confirmation and ticket generation mechanism that provides a booking reference and e-ticket.
- **Pros**:
    - Ensures users receive instant booking confirmation and flight details.

    - Provides a booking reference that can be used for check-in and flight history tracking.
- **Cons**:
    - Complex formatting might be needed to ensure ticket readability on various devices.
    - Requires handling edge cases, such as last-minute seat changes or cancellations.

5. Looping Mechanism for Multiple Bookings

The system allows users to make multiple bookings in one session. After each booking, users can choose to make another reservation or exit. The application resets to the initial search screen if a new booking is initiated.

- **Output**: A loop that enables users to make multiple bookings in one session.
- **Pros**:
    - Enhances user convenience by allowing multiple bookings without re-logging.
    - Facilitates group bookings by letting users make consecutive reservations.
- **Cons**:
    - Additional logic is needed to reset variables and clear data between bookings.
    - Storing multiple reservations in one session may impact data handling requirements.

6. Flight History and Record-Keeping

The flight history feature maintains a record of each user's past bookings, which they can view for reference. Each time a booking is completed, it's saved in the user's flight history, allowing them to see details of their previous flights.

- **Output**: A dynamic history feature that stores completed bookings and displays past flight information.
- **Pros**:
    - Provides users with a convenient way to track their booking history.
    - Useful for frequent flyers who may want to revisit past reservations.
- **Cons**:
    - Requires careful data handling to ensure accurate record-keeping.
    - Data volume may grow significantly over time, necessitating efficient data management.

7. Testing and Quality Assurance

Extensive testing was performed to verify the functionality of each component, including flight search, seat selection, booking confirmation, and flight history tracking. Testing included unit tests for each function, integration testing to ensure smooth component interaction, and user testing for feedback on usability and performance.

- **Output**: A stable, user-tested airline reservation system with reliable features.
- **Pros**:
    - Reduces bugs and ensures reliable booking and record-keeping.
    - User feedback during testing offers insights for further refinement.
- **Cons**:
    - Testing cycles may be time-consuming, especially with multi-step booking logic.
    - Replicating all potential user interactions and edge cases can be challenging.

IMPLEMENTATION

The implementation of this airline reservation application involves multiple phases, from database setup and user handling to managing reservations, validating seat availability, and maintaining a frequent flyer feature. Each part is designed to work together to provide users with a seamless flight booking experience.

1. **Database Setup and Design**

   The MySQL database serves as the backbone for storing essential data, such as flight details, user profiles, and reservation information. The database includes tables for flights (with flight numbers, destinations, schedules, and seat availability), users (with personal information and frequent flyer points), and reservations (storing booking details). This setup ensures that each piece of data is organized and easily retrievable. This structure supports the application's needs, from managing seat reservations to updating frequent flyer points. The database is connected to the Java application using JDBC (Java Database Connectivity), which allows for efficient data handling and interaction.

2. **User Registration and Login Process**

   When the application begins, users are prompted to register with a unique username or log in if they are returning users. This step personalizes the experience, allowing users to access past booking information and track their flight history and frequent flyer points. For new users, the system creates a new entry in the database to initialize their records. For returning users, the application retrieves existing data, ensuring that frequent flyer points and booking history are retained and updated each time they make a reservation. This feature enhances user experience and promotes loyalty by rewarding frequent users.

3. **Flight Search and Booking Process**

   The core of the application revolves around allowing users to search for flights and make reservations. The system retrieves available flights from the

database based on user search criteria (e.g., destination, date, preferred class) and displays them in a list. Users can select a flight and proceed with booking. This setup also supports additional features, such as filtering by seat class (economy, business) and selecting specific seats, which provides users with a personalized booking experience. The search and booking system includes options for users to confirm seat selection and receive immediate feedback on seat availability.

4. **Seat Validation and Reservation Confirmation**

   Once a user selects a flight, the application validates seat availability by checking the database. If seats are available, the reservation is recorded, and seat availability in the database is updated accordingly. Users receive a booking confirmation with all relevant flight details. This seat validation process ensures data accuracy and helps avoid overbooking. After booking, users can view their reservation details, including flight information and total cost. The application also allows users to cancel or modify bookings if needed, updating the database accordingly.

5. **Frequent Flyer Points and Loyalty Program**

   To encourage repeat use, the application includes a frequent flyer program. For each flight booking, users earn points that are stored in their profile. Over time, these points can be redeemed for discounts or upgrades. This loyalty feature is a major motivator, encouraging users to book additional flights to accumulate points. The frequent flyer points system also allows users to track their progress and see their eligibility for rewards, which enhances user satisfaction and promotes engagement.

6. **Reservation History and Management**

   Users can view their past reservations and manage current ones. This feature allows users to see all completed flights and any upcoming bookings, making it easy to access travel details. The reservation history management function is essential for frequent travelers, as it allows them to check previous flights,

manage travel schedules, and ensure they are accruing frequent flyer points. Additionally, this feature adds convenience by allowing users to quickly rebook frequently traveled routes.

7. **Fare Calculation and Payment Integration**

   The application includes an automatic fare calculator based on factors such as seat class, flight distance, and taxes. After confirming a flight selection, users are shown the total fare, and they proceed to the payment gateway for finalizing the booking. Payment integration ensures that users have a secure and convenient way to pay for flights. The fare calculation feature is accurate and transparent, allowing users to see a breakdown of costs before making a payment. This reduces the risk of hidden fees and builds trust with users.

RESULTS AND DISCUSSION

Results of the Airline Reservation Application

The airline reservation application achieved the goal of providing a user-friendly platform for booking and managing flights. Key features such as flight search, seat validation, booking management, and frequent flyer rewards were successfully implemented. Upon testing, the following results were observed:

1. **Efficient Flight Retrieval and Presentation**: The application quickly retrieves flight information from the database and presents it based on user search criteria, making it easy for users to find desired flights.
2. **Real-Time Seat Availability Validation**: The system validates seat availability immediately during booking, providing real-time feedback on availability and avoiding issues with overbooking.
3. **Personalized Experience with Frequent Flyer Points**: The frequent flyer feature allows users to track and accumulate points, enhancing loyalty and promoting engagement with the application.
4. **Reservation History for Convenient Management**: Users can easily view past and current bookings, allowing them to access important travel information at any time.

5. **Fare Transparency and Secure Payment**: Users see a clear breakdown of costs, and the integration of a payment gateway ensures secure transactions.

Comparison with Existing Methods

This application is more interactive and personalized than traditional booking methods. Compared with traditional systems, it offers advantages such as real-time seat validation, loyalty rewards, and reservation history tracking.

1. **Dynamic Database Integration**: Unlike traditional booking systems, this application allows for real-time updates to flight schedules and seat availability. This flexibility makes it superior to static systems that may require frequent updates.
2. **User-Specific Tracking and Rewards**: The frequent flyer system enables users to track rewards points, a feature not commonly available in traditional reservation methods.
3. **Immediate Feedback and Availability Checking**: Users receive immediate information on seat availability, which is an improvement over delayed confirmation methods in offline systems.
4. **User Engagement and Motivation**: The frequent flyer program and the easy interface encourage users to book more flights and engage with the platform more regularly.

Discussion on Improvements and Future Work

Although the current implementation is effective, there are areas where improvements could be made:

1. **Mobile Compatibility**: Expanding the application for mobile devices would allow users to book flights on the go, broadening the application's accessibility.
2. **Dynamic Pricing Based on Demand**: Implementing a dynamic pricing feature that adjusts fares based on demand and booking times would make the application competitive with other airline platforms.

3. **Enhanced User Interface**: Adding a more visually appealing interface with elements like seat selection visuals, custom themes, and real-time flight tracking could improve the user experience.
4. **Integration with Other Travel Services**: Adding options for users to book hotels, rental cars, and vacation packages would create a more comprehensive travel experience.

CONCLUSION

The development of this airline reservation application demonstrates a successful effort to create a user-friendly and engaging platform that combines Java programming with MySQL database integration. This project goes beyond traditional reservation methods by offering real-time feedback, loyalty rewards, and personalized booking history. The project effectively showcases Java and SQL skills while providing practical functionality and user engagement. This application enables users to book flights, manage reservations, and earn rewards efficiently. Future enhancements, such as mobile compatibility, dynamic pricing, and additional travel services, would make it a more versatile and scalable platform for the travel industry.

SAMPLE CODING

Connector Code

```
package airlinemanagementsystem;
import java.sql.*;
public class Conn {
    Connection c;
    Statement s;
    public Conn() {
```

```
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            c =
DriverManager.getConnection("jdbc:mysql://localhost:3306/airlinemanagementsystem
", "root", "root");
            s = c.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Main Code

HOME PAGE

```
package airlinemanagementsystem;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Home extends JFrame implements ActionListener{
    public Home() {
        setLayout(null);
        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("airlinemanagementsystem/icons/front.j
pg"));
        JLabel image = new JLabel(i1);
        image.setBounds(0, 0, 1600, 800);
        add(image);
        JLabel heading = new JLabel("AIR INDIA WELCOMES YOU");
        heading.setBounds(500, 40, 1000, 40);
```

```
heading.setForeground(Color.BLUE);
heading.setFont(new Font("Tahoma", Font.PLAIN, 36));
image.add(heading);
JMenuBar menubar = new JMenuBar();
setJMenuBar(menubar);
JMenu details = new JMenu("Details");
menubar.add(details);
JMenuItem flightDetails = new JMenuItem("Flight Details");
flightDetails.addActionListener(this);
details.add(flightDetails);
JMenuItem customerDetails = new JMenuItem("Add Customer Details");
customerDetails.addActionListener(this);
details.add(customerDetails);
JMenuItem bookFlight = new JMenuItem("Book Flight");
bookFlight.addActionListener(this);
details.add(bookFlight);
JMenuItem journeyDetails = new JMenuItem("Journey Details");
journeyDetails.addActionListener(this);
details.add(journeyDetails);
JMenuItem ticketCancellation = new JMenuItem("Cancel Ticket");
ticketCancellation.addActionListener(this);
details.add(ticketCancellation);
JMenu ticket = new JMenu("Ticket");
menubar.add(ticket);
JMenuItem boardingPass = new JMenuItem("Boarding Pass");
boardingPass.addActionListener(this);
ticket.add(boardingPass);
setExtendedState(JFrame.MAXIMIZED_BOTH);
setVisible(true);
```

```
    }
    public void actionPerformed(ActionEvent ae) {
        String text = ae.getActionCommand();
        if (text.equals("Add Customer Details")) {
            new AddCustomer();
        } else if (text.equals("Flight Details")) {
            new FlightInfo();
        } else if (text.equals("Book Flight")) {
            new BookFlight();
        } else if (text.equals("Journey Details")) {
            new JourneyDetails();
        } else if (text.equals("Cancel Ticket")) {
            new Cancel();
        }
        else if (text.equals("Boarding Pass")) {
            new BoardingPass();
        }
    }
    public static void main(String[] args) {
        new Home();
    }
}
```

FLIGHT INFO

```
package airlinemanagementsystem;
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import net.proteanit.sql.DbUtils;
```

```
public class FlightInfo extends JFrame{
    public FlightInfo() {
        getContentPane().setBackground(Color.WHITE);
        setLayout(null);
        JTable table = new JTable();
        try {
            Conn conn = new Conn();

            ResultSet rs = conn.s.executeQuery("select * from flight");
            table.setModel(DbUtils.resultSetToTableModel(rs));
        } catch(Exception e) {
            e.printStackTrace();
        }
        JScrollPane jsp = new JScrollPane(table);
        jsp.setBounds(0, 0, 800, 500);
        add(jsp);
        setSize(800, 500);
        setLocation(400, 200);
        setVisible(true);
    }
    public static void main(String[] args) {
        new FlightInfo();
    }
}
```

COUSTOMER INFO

```
package airlinemanagementsystem;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class AddCustomer extends JFrame implements ActionListener{
    JTextField tfname, tfphone, tfaadhar, tfnationality, tfaddress;
    JRadioButton rbmale, rbfemale;
    public AddCustomer() {
        getContentPane().setBackground(Color.WHITE);
        setLayout(null);
        JLabel heading = new JLabel("ADD CUSTOMER DETAILS");
        heading.setBounds(220, 20, 500, 35);
        heading.setFont(new Font("Tahoma", Font.PLAIN, 32));
        heading.setForeground(Color.BLUE);
        add(heading);
        JLabel lblname = new JLabel("Name");
        lblname.setBounds(60, 80, 150, 25);
        lblname.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lblname);
        tfname = new JTextField();
        tfname.setBounds(220, 80, 150, 25);
        add(tfname);
        JLabel lblnationality = new JLabel("Nationality");
        lblnationality.setBounds(60, 130, 150, 25);
        lblnationality.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lblnationality);
        tfnationality = new JTextField();
        tfnationality.setBounds(220, 130, 150, 25);
        add(tfnationality);
        JLabel lblaadhar = new JLabel("Aadhar Number");
        lblaadhar.setBounds(60, 180, 150, 25);
        lblaadhar.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lblaadhar);
```

```
tfaadhar = new JTextField();
tfaadhar.setBounds(220, 180, 150, 25);
add(tfaadhar);
JLabel lbladdress = new JLabel("Address");
lbladdress.setBounds(60, 230, 150, 25);
lbladdress.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbladdress);
tfaddress = new JTextField();
tfaddress.setBounds(220, 230, 150, 25);
add(tfaddress);
JLabel lblgender = new JLabel("Gender");
lblgender.setBounds(60, 280, 150, 25);
lblgender.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblgender);
ButtonGroup gendergroup = new ButtonGroup();
rbmale = new JRadioButton("Male");
rbmale.setBounds(220, 280, 70, 25);
rbmale.setBackground(Color.WHITE);
add(rbmale);
rbfemale = new JRadioButton("Female");
rbfemale.setBounds(300, 280, 70, 25);
rbfemale.setBackground(Color.WHITE);
add(rbfemale);
gendergroup.add(rbmale);
gendergroup.add(rbfemale);
JLabel lblphone = new JLabel("Phone");
lblphone.setBounds(60, 330, 150, 25);
lblphone.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblphone);
```

```
        tfphone = new JTextField();
        tfphone.setBounds(220, 330, 150, 25);
        add(tfphone);
        JButton save = new JButton("SAVE");
        save.setBackground(Color.BLACK);
        save.setForeground(Color.WHITE);
        save.setBounds(220, 380, 150, 30);
        save.addActionListener(this);
        add(save);
        ImageIcon image = new
ImageIcon(ClassLoader.getSystemResource("airlinemanagementsystem/icons/emp.p
ng"));
        JLabel lblimage = new JLabel(image);
        lblimage.setBounds(450, 80, 280, 400);
        add(lblimage);
        setSize(900, 600);
        setLocation(300, 150);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        String name = tfname.getText();
        String nationality = tfnationality.getText();
        String phone = tfphone.getText();
        String address = tfaddress.getText();
        String aadhar = tfaadhar.getText();
        String gender = null;
        if (rbmale.isSelected()) {
            gender = "Male";
        } else {
```

```
            gender = "Female";
        }
        try {
            Conn conn = new Conn();
            String query = "insert into passenger values('"+name+"', '"+nationality+"',
'"+phone+"', '"+address+"', '"+aadhar+"', '"+gender+"')";
            conn.s.executeUpdate(query);
            JOptionPane.showMessageDialog(null, "Customer Details Added
Successfully");
            setVisible(false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new AddCustomer();
    }
}
```

BOOK FLIGHT

```
package airlinemanagementsystem;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
public class BoardingPass extends JFrame implements ActionListener{
    JTextField tfpnr;
    JLabel tfname, tfnationality, lblsrc, lbldest, labelfname, labelfcode, labeldate;
    JButton fetchButton;
    public BoardingPass() {
```

```
getContentPane().setBackground(Color.WHITE);
setLayout(null);
JLabel heading = new JLabel("AIR INDIA");
heading.setBounds(380, 10, 450, 35);
heading.setFont(new Font("Tahoma", Font.PLAIN, 32));
add(heading);
JLabel subheading = new JLabel("Boarding Pass");
subheading.setBounds(360, 50, 300, 30);
subheading.setFont(new Font("Tahoma", Font.PLAIN, 24));
subheading.setForeground(Color.BLUE);
add(subheading);
JLabel lblaadhar = new JLabel("PNR DETAILS");
lblaadhar.setBounds(60, 100, 150, 25);
lblaadhar.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblaadhar);
tfpnr = new JTextField();
tfpnr.setBounds(220, 100, 150, 25);
add(tfpnr);
fetchButton = new JButton("Enter");
fetchButton.setBackground(Color.BLACK);
fetchButton.setForeground(Color.WHITE);
fetchButton.setBounds(380, 100, 120, 25);
fetchButton.addActionListener(this);
add(fetchButton);
JLabel lblname = new JLabel("NAME");
lblname.setBounds(60, 140, 150, 25);
lblname.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblname);
tfname = new JLabel();
```

```
tfname.setBounds(220, 140, 150, 25);
add(tfname);
JLabel lblnationality = new JLabel("NATIONALITY");
lblnationality.setBounds(60, 180, 150, 25);
lblnationality.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblnationality);
tfnationality = new JLabel();
tfnationality.setBounds(220, 180, 150, 25);
add(tfnationality);
JLabel lbladdress = new JLabel("SRC");
lbladdress.setBounds(60, 220, 150, 25);
lbladdress.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbladdress);
lblsrc = new JLabel();
lblsrc.setBounds(220, 220, 150, 25);
add(lblsrc);
JLabel lblgender = new JLabel("DEST");
lblgender.setBounds(380, 220, 150, 25);
lblgender.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblgender);
lbldest = new JLabel();
lbldest.setBounds(540, 220, 150, 25);
add(lbldest);
JLabel lblfname = new JLabel("Flight Name");
lblfname.setBounds(60, 260, 150, 25);
lblfname.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblfname);
labelfname = new JLabel();
labelfname.setBounds(220, 260, 150, 25);
```

```
        add(labelfname);
        JLabel lblfcode = new JLabel("Flight Code");
        lblfcode.setBounds(380, 260, 150, 25);
        lblfcode.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lblfcode);

        labelfcode = new JLabel();
        labelfcode.setBounds(540, 260, 150, 25);
        add(labelfcode);
        JLabel lbldate = new JLabel("Date");
        lbldate.setBounds(60, 300, 150, 25);
        lbldate.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lbldate);
        labeldate = new JLabel();
        labeldate.setBounds(220, 300, 150, 25);
        add(labeldate);
        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("airlinemanagementsystem/icons/airindi
a.png"));
        Image i2 = i1.getImage().getScaledInstance(300, 230,
Image.SCALE_DEFAULT);
        ImageIcon image = new ImageIcon(i2);
        JLabel lblimage = new JLabel(image);
        lblimage.setBounds(600, 0, 300, 300);
        add(lblimage);
        setSize(1000, 450);
        setLocation(300, 150);
        setVisible(true);
    }
```

```
    public void actionPerformed(ActionEvent ae) {
        String pnr = tfpnr.getText();
        try {
            Conn conn = new Conn();
            String query = "select * from reservation where PNR = '"+pnr+"'";
            ResultSet rs = conn.s.executeQuery(query);
            if (rs.next()) {
                tfname.setText(rs.getString("name"));
                tfnationality.setText(rs.getString("nationality"));
                lblsrc.setText(rs.getString("src"));
                lbldest.setText(rs.getString("des"));
                labelfname.setText(rs.getString("flightname"));
                labelfcode.setText(rs.getString("flightcode"));
                labeldate.setText(rs.getString("ddate"));
            } else {
                JOptionPane.showMessageDialog(null, "Please enter correct
PNR");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new BoardingPass();
    }
}
```

JOURNEY DETAILS

```
package airlinemanagementsystem;
```

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import java.awt.event.*;
import net.proteanit.sql.DbUtils;
public class JourneyDetails extends JFrame implements ActionListener{
    JTable table;
    JTextField pnr;
    JButton show;
    public JourneyDetails() {
        getContentPane().setBackground(Color.WHITE);
        setLayout(null);
        JLabel lblpnr = new JLabel("PNR Details");
        lblpnr.setFont(new Font("Tahoma", Font.PLAIN, 16));
        lblpnr.setBounds(50, 50, 100, 25);
        add(lblpnr);
        pnr = new JTextField();
        pnr.setBounds(160, 50, 120, 25);
        add(pnr);
        show = new JButton("Show Details");
        show.setBackground(Color.BLACK);
        show.setForeground(Color.WHITE);
        show.setBounds(290, 50, 120, 25);
        show.addActionListener(this);
        add(show);
        table = new JTable();
        JScrollPane jsp = new JScrollPane(table);
        jsp.setBounds(0, 100, 800, 150);
        jsp.setBackground(Color.WHITE);
```

```
        add(jsp);
        setSize(800, 600);
        setLocation(400, 150);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        try {
            Conn conn = new Conn();
            ResultSet rs = conn.s.executeQuery("select * from reservation where PNR =
'"+pnr.getText()+"'");
            if (!rs.isBeforeFirst()) {
                JOptionPane.showMessageDialog(null, "No Information Found");
                return;
            }
            table.setModel(DbUtils.resultSetToTableModel(rs));
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new JourneyDetails();
    }
}
```

Faculty Incharge **Academic Coordinator**

HOD