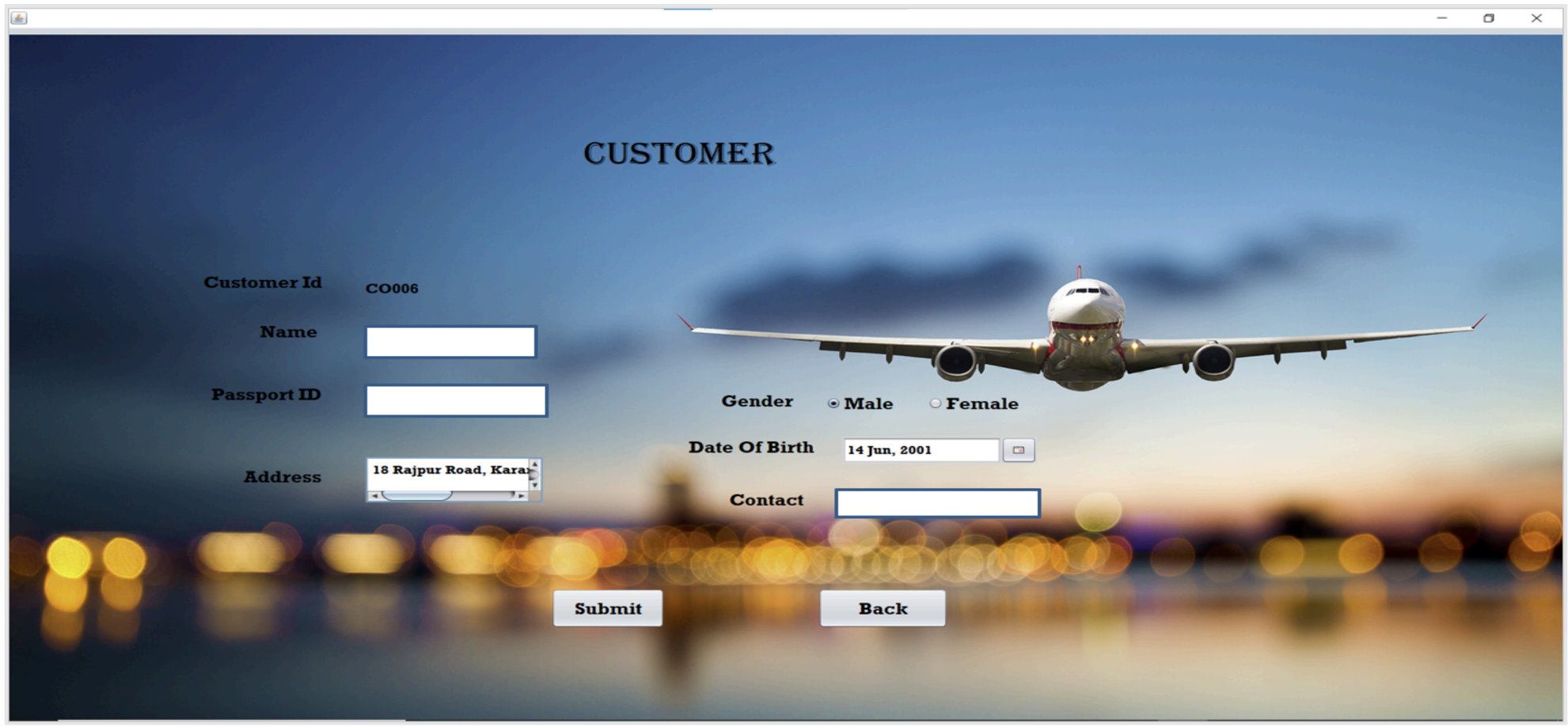
r



AIRLINE MANAGEMENT SYSTEM

DATABASE

SUBMITTED BY -

Lallesh rajak

lallesh.24scse11180169@galgotiasuniversity.ac.in

Dhruv Rawat

dhruv.24scse11180547@galgotiasuniversity.ac.in

Shlok Mishra

shlok.24scse11180214@galgotiasuniversity.ac.in

REVIEW ON (2)

JAVA PROJECT

TOPIC :-

1 Core Feature Implementation

2 Error Handling & Robustness

3 Integration of Components

4 Event Handling & Processing

5 Data Validation

6 Code Quality & Innovation

7 Project Documentation

Core Feature Implementation

1. ✅ User Management Module

Purpose:

To manage all user-related functionalities like registration, login, profile management, and role-based access (e.g., admin vs. customer).

Core Functionalities:

- **User registration & authentication**
- **Password encryption (e.g., BCrypt)**
- **Role-based access control (RBAC)**

Design Considerations:

- **User class/entity** with fields like id, name, email, password, role
- Use **Java Bean Validation** for input validation
- Store hashed passwords, not plain text
- Use **JWT or Spring Security** (in Spring Boot projects) for login sessions

Database Tables:

- Users (user_id, name, email, password_hash, role)

2. 🛫 Flight Management Module

Purpose:

To manage all operations related to flight schedules, such as creation, update, deletion, and querying flights.

Core Functionalities:

- Add/update/delete flight schedules (admin)
- Assign aircraft and crew
- Search flights by route and date (customers)

Design Considerations:

- Flight class/entity with fields like flight_id, origin, destination, departure_time, arrival_time, aircraft_id, available_seats
- Use DAO/Repository pattern for DB access
- Ensure no overlapping flights for the same aircraft

Database Tables:

- Flights (flight_id, flight_number, origin, destination, departure_time, arrival_time, aircraft_id)
- Aircrafts (aircraft_id, model, capacity, maintenance_status)

3. 🎟️ Booking and Reservation Module

Purpose:

To handle the reservation process including ticket booking, seat availability check, and PNR generation.

Core Functionalities:

- Search for available flights
- Select a flight and confirm booking
- Generate and return a **unique PNR**
- View booking history
- Cancel bookings and manage refund logic

Design Considerations:

- Check real-time seat availability

- Prevent race conditions using transactional control
- Integrate with a payment module
- Use **Builder Pattern** or Service Layer pattern for creating bookings

Database Tables:

- Reservations (reservation_id, user_id, flight_id, seat_number, status, booking_time)
- Payments (payment_id, reservation_id, amount, payment_status, payment_method)

4. 💺 Seat Management

Purpose:

To manage seat assignments and prevent double booking.

Core Functionalities:

- Maintain list of booked and available seats
- Allow customers to choose seats during booking
- Assign automatically if not chosen

Design Considerations:

- Implement logic to check seat availability
- Store seat maps in a normalized table or as JSON in modern DBs

Database Tables:

- Seats (seat_id, flight_id, seat_number, is_available, reservation_id)

5. 💳 Payment Integration

Purpose:

To enable users to pay for bookings using credit cards, net banking, or wallets.

Core Functionalities:

- Integrate with payment gateways (e.g., Stripe, Razorpay)
- Record transaction details

- Handle payment status (success/failure/refund)

Design Considerations:

- Payments must be atomic and transactional with bookings
- Store status for reconciliation
- Use secure handling for card details (or tokenized payments)

Database Tables:

- Payments (payment_id, reservation_id, amount, method, status, timestamp)

6. 🧑 ✈ Crew and Aircraft Management

Purpose:

To manage airline staff and aircraft resources.

Core Functionalities:

- Assign pilots and crew to flights
- Prevent overbooking crew or using unavailable aircraft
- Track maintenance logs

Design Considerations:

- Avoid conflicts in schedule
- Notify maintenance periods
- Each crew member must be linked to a valid role and license

Database Tables:

- Crew (crew_id, name, role, availability_status)
- AircraftLogs (log_id, aircraft_id, check_type, date, status)

7. 🧾 Admin Dashboard and Reporting

Purpose:

To allow admins to monitor system health, bookings, revenue, and user activity.

Core Functionalities:

- View flight statistics (occupancy rate, delay history)
- Manage users and bookings
- Generate reports (PDF/Excel)

Design Considerations:

- Use Java with libraries like Apache POI or JasperReports for generating reports
- Role-based views for staff/admin

Data Analytics:

- Daily bookings
- Revenue by route
- Cancellations and refunds

8. 🔐 Security and Authentication

Purpose:

To ensure secure access and prevent data breaches.

Core Features:

- JWT-based token authentication (or Spring Security sessions)
- Password hashing (e.g., BCrypt)
- Role-based access restrictions

Design Considerations:

- Input sanitization to prevent SQL injection/XSS
- HTTPS everywhere
- Log out/invalidate tokens after session expiry

🧠 Integration Summary (Java Concepts Used)

Feature	Java Concepts Used
Modularity	OOP, Interfaces, Service Layer

Feature	Java Concepts Used
Persistence	JDBC / Hibernate / JPA
Web API (if extended)	Spring Boot, RESTful Controllers
Transactions	Spring @Transactional, JDBC transactions
Security	JWT, BCrypt, Spring Security
Scheduling	Java Timer / Quartz Scheduler
Testing	JUnit, Mockito

1. Error Handling in Airline Management System

A. Types of Errors to Handle

Error Type	Examples
Input validation errors	Invalid email, missing passenger details
Business logic errors	Booking a full flight, cancelling a non-existent booking
Database errors	Duplicate entries, constraints violations
Network/API errors	Payment gateway timeout, external flight API unavailable
Security errors	Unauthorized access, invalid tokens
System-level exceptions	NullPointerException, IOExceptions

B. Error Handling Strategies (Java)

1. Use Custom Exception Classes

Create domain-specific exceptions for clear control over flow:

java

CopyEdit

```
public class FlightNotFoundException extends RuntimeException {
    public FlightNotFoundException(String message) {
        super(message);
    }
}
```

2. Centralized Exception Handling

If using Spring Boot:

java

CopyEdit

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(FlightNotFoundException.class)
    public ResponseEntity<String> handleFlightNotFound(FlightNotFoundException
ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleGenericError(Exception ex) {
        return new ResponseEntity<>("Internal Server Error",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

3. Validation with Java Bean Validation

Use @Valid, @NotNull, @Size, etc., to auto-check request data.

4. Transaction Handling

Wrap critical sections (like booking and payment) in atomic transactions to roll back on failure:

java

CopyEdit

```
@Transactional
public void bookFlight(...) {
    // if anything fails here, the whole transaction is rolled back
}
```

5. Logging Errors

Use logging frameworks:

- SLF4J + Logback / Log4j
- Log error type, timestamp, and request context

🧱 2. Robustness in Airline Management System

A. Definition of Robustness

Robustness means the system continues to operate under:

- Unexpected inputs
- Partial failures (e.g., DB down, service unavailable)
- Concurrent usage

B. Key Design Principles for Robustness

1. Fail-Safe Design

- Gracefully degrade services if something fails (e.g., fallback flights list)
- Retry policies for failed remote calls (e.g., to payment gateway)

2. Input Validation & Sanitization

- Validate all user input on both client and server side
- Prevent injection attacks and logic errors

3. Concurrency Handling

- Use thread-safe structures (ConcurrentHashMap) and locks
- Avoid race conditions in booking logic
- Apply optimistic/pessimistic locking at the database level

4. Timeouts & Circuit Breakers

For remote service calls (e.g., payment gateway), use:

- Timeouts
- Retry with exponential backoff
- Circuit breakers (e.g., using Resilience4j or Hystrix)

5. Data Integrity Checks

- Foreign key constraints
- Unique constraints (e.g., one booking per seat per flight)
- Referential integrity between flights, users, reservations

6. Backup & Recovery

- Regular database backups
- Error logs stored for recovery
- Implement dead-letter queues (if using messaging services)

C. Testing for Robustness

1. Unit Testing

- Test all services, especially booking, flight management
- Use frameworks like JUnit and Mockito

2. Integration Testing

- Test complete workflows (e.g., search → book → pay → cancel)

3. Load & Stress Testing

- Simulate concurrent users booking the same flight
- Use tools like JMeter or Gatling

4. Failure Injection

- Simulate failure in payment gateway
- Simulate database unavailability

D. Monitoring and Alerts

- Implement health checks for services (e.g., /health endpoint)
- Use tools like:
    - Prometheus + Grafana for metrics
    - ELK stack or Graylog for logs
    - Alerting on booking failures or high error rates

🔗 1. Integration Overview

In a modular Java application (especially if using Spring Boot), integration happens via:

- **Service-to-Service communication** (within monolith or via microservices)
- **Shared models/entities**
- **RESTful APIs or internal interfaces**
- **Database relations and transactional operations**
- **Event handling (optional, for advanced systems)**

🧩 2. Key Component Integrations

A. User Management ↔ Reservation System

Integration Goal:

Authenticate users and associate reservations with specific user accounts.

How:

- After user logs in, their user ID/token is passed with each booking request.
- The reservation module uses this ID to record the booking under the right user.

Integration Flow:

plaintext

CopyEdit

User logs in → Receives token → Calls /book-flight API → ReservationService associates booking with user ID

Technologies:

- Spring Security or JWT for authentication
- Service class integration (e.g., UserService.getUserById() in ReservationService)

B. Flight Management ↔ Booking System

Integration Goal:

Ensure that seat availability, flight details, and aircraft status are synchronized during the booking.

How:

- Booking service queries the flight schedule and seat status before creating a reservation.
- Upon successful booking, available seats are decremented.

Integration Flow:

plaintext

CopyEdit

BookingService → FlightService.getFlightById()

→ check availableSeats

→ reserve seat → update flight availability

Key Integration Points:

- Shared Flight entity
- Transactions to ensure seat counts are updated atomically

C. Booking System ↔ Payment System

Integration Goal:

Ensure that bookings are only confirmed if payment is successful.

How:

- Booking is created in a **pending** state
- Payment is initiated
- On success: status updated to **confirmed**
- On failure: booking is cancelled or rolled back

Integration Flow:

plaintext

CopyEdit

1. BookFlightRequest → BookingService.createReservation() → set status: PENDING

2. Call PaymentService → confirm payment

3. On success → update status to CONFIRMED

4. On failure → cancel reservation

Best Practices:

- Use transactional methods or saga pattern (for microservices)
- External payment gateway integration (via REST or SDK)

📦 3. Realistic Integration Architecture (Layered or Microservice)

A. Layered Monolithic Architecture (Spring Boot Example)

plaintext

CopyEdit

[Controller Layer] (REST API)

↓

[Service Layer]

↓

[Repository Layer] ←→ [Database]

Example:

plaintext

CopyEdit

BookingController

→ BookingService

→ UserService (get user)

→ FlightService (validate flight & seats)

→ PaymentService (trigger payment)

→ ReservationRepository (save reservation)

B. Microservices Architecture (Advanced Systems)

Each module (user, flight, booking, payment) is its own service:

plaintext

CopyEdit

[User Service] ← REST API → [Booking Service] ← REST/API → [Payment Service]

↑

[Flight Service]

Integration:

- Services communicate via REST or messaging (Kafka, RabbitMQ)
- Each has its own database
- Use API gateways and service discovery tools (e.g., Eureka, Zuul)

🧠 Integration Design Considerations

Concern	Solution
Data consistency	Use transactions in monolith, or eventual consistency in microservices
Error propagation	Return proper HTTP codes and messages
API coupling	Use interface abstraction; avoid tight service dependencies
Security	Authenticate every inter-service API call (JWT, OAuth2)
Performance	Use caching for frequent data (e.g., flight schedules

🛎️ Event Handling

Event handling is the mechanism of listening for and responding to system or user-generated events.
Event processing is how the system executes logic in response to those events.

In an airline management system, this can be implemented using:

- **Synchronous method calls** (in simple systems)
- **Asynchronous event dispatchers** (in larger systems)
- **Message queues or event buses** (in scalable architectures)

🔄 2. Types of Events in Airline Management System

Event Type	Example Events
User Events	User registration, login, password reset
Booking Events	Flight booked, booking cancelled, seat reserved
Flight Events	Flight delayed, rescheduled, or cancelled
Payment Events	Payment success/failure
Notification Events	Email/SMS sent for confirmations or alerts
System Events	Low seat alert, crew unavailability, system error

⚙️ 3. Event Handling Implementation (Java)

A. Synchronous Event Handling

In a simple Java system, this is handled by calling methods directly:

java

CopyEdit

```
public void bookFlight(User user, Flight flight) {
    reservationService.createReservation(...);
    notificationService.sendEmail(...);
}
```

Drawback: Tightly coupled; one failure can break the whole flow.

B. Asynchronous Event Handling (Observer Pattern)

In Java, you can use:

- **Observer pattern**
- **EventBus (e.g., Guava EventBus)**
- **Spring's ApplicationEventPublisher**

Example with Spring:

java

CopyEdit

```
public class BookingConfirmedEvent extends ApplicationEvent {
    private Reservation reservation;
    public BookingConfirmedEvent(Object source, Reservation reservation) {
        super(source);
        this.reservation = reservation;
    }
}
```

Publisher:

java

CopyEdit

```
eventPublisher.publishEvent(new BookingConfirmedEvent(this, reservation));
```

Listener:

java

CopyEdit

```
@Component
public class BookingEventListener {
    @EventListener
```

```
    public void handleBookingConfirmed(BookingConfirmedEvent event) {
        notificationService.sendEmail(event.getReservation());
    }
}
```

C. Message Queue-Based Event Processing (Advanced / Scalable)

Used in **microservices** or distributed systems:

- Tools: **Kafka, RabbitMQ, ActiveMQ**
- Events published to topics/queues and consumed asynchronously

Example Flow (Kafka):

plaintext

CopyEdit

Booking Service publishes → "booking.confirmed" → Kafka Topic

Notification Service listens → sends email confirmation

Analytics Service listens → logs for dashboard

Advantages:

- Loose coupling
- Retry mechanisms
- Scalable consumers

🧩 4. Event-Driven Scenarios in Airline System

1. Flight Booking Completed

- **Trigger:** User books a seat
- **Handlers:**
  - Reserve seat in flight
  - Send confirmation email
  - Trigger payment transaction

    - Update frequent flyer miles

2. Flight Cancellation

- **Trigger**: Admin cancels a flight
- **Handlers**:
    - Notify all passengers
    - Refund bookings
    - Update crew/aircraft schedule

3. Check-In Completed

- **Trigger**: User checks in
- **Handlers**:
    - Assign boarding pass
    - Update reservation status
    - Send boarding notification

🧠 5. Event Processing Considerations

Consideration	Strategy
Reliability	Use durable queues; retry logic for failed events
Ordering	Ensure event ordering in booking/flight cancellation flows
Error Handling	Log, retry or dead-letter failed events
Scalability	Use message brokers to scale consumers independently
Monitoring	Track event success/failure with tools like Prometheus + Grafana

✅ 5. Data Validation in an Airline Management System

Data validation ensures that the data entered into or processed by the system is correct, complete, and safe. In an **Airline Management System**, data validation is essential for maintaining **data integrity**, preventing **system errors**, and ensuring **security** and **regulatory compliance**.

🔍 Why Data Validation is Critical in an Airline System

Module	Validation Examples
User Registration	Valid email format, unique username, password strength
Flight Management	Valid dates, positive seat numbers, correct time ranges
Booking System	Valid flight ID, available seat check, valid user ID
Payment System	Non-negative amount, supported payment method
Check-in Module	Valid reservation ID, seat allocation confirmation

🧩 Levels of Data Validation

1. Client-side Validation *(Frontend)*

- Basic checks (e.g., email format, required fields)
- Implemented using JavaScript, HTML5 validation attributes

2. Server-side Validation *(Java Backend)*

- Must be enforced even if client-side fails
- Implemented using:
    - Java Validation API (javax.validation.constraints)
    - Custom logic in service or controller layers

3. Database-level Validation

- Use constraints (e.g., NOT NULL, UNIQUE, FOREIGN KEY, CHECK)

🧱 Java-based Data Validation Techniques

A. Using Java Bean Validation (JSR 380) + Hibernate Validator

Add annotations directly to data model classes:

Example: User Entity

java

CopyEdit

```
public class User {

    @NotNull
    @Size(min = 3, max = 50)
    private String name;

    @Email
    @NotBlank
    private String email;

    @Size(min = 6)
    private String password;
}
```

Then use:

java

CopyEdit

```
@PostMapping("/register")
public ResponseEntity<?> register(@Valid @RequestBody User user) {
    // automatically validates based on annotations
}
```

B. Custom Validator (Business Logic Layer)

Some validations cannot be handled by annotations.

Examples:

- Flight departure time must be before arrival time
- Cannot book a seat if already reserved

java

CopyEdit

```
if (flight.getAvailableSeats() <= 0) {
    throw new IllegalArgumentException("No seats available.");
}
```

C. Validation with Spring Boot

Spring Boot automatically supports validation if you use @Valid or @Validated in controller methods.

Global error handling:

java

CopyEdit

```
@ControllerAdvice
public class ValidationExceptionHandler {


    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<?>
handleValidationErrors(MethodArgumentNotValidException ex) {
        List<String> errors = ex.getBindingResult()
                                .getFieldErrors()
                                .stream()
                                .map(err -> err.getField() + ": " + err.getDefaultMessage())
                                .toList();
        return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
    }
}
```

⚠ Common Validation Scenarios in Airline Systems

Area	What to Validate	How to Validate
User Registration	Email format, uniqueness, password strength	Annotations + service logic
Flight Scheduling	Future dates, valid times, positive seat numbers	Manual checks + JSR annotations
Booking	Seat availability, valid flight ID and user ID	Transactional logic in booking service
Payment	Non-negative amount, valid methods, unique ref ID	Enum checks + regex validation
Check-in	Valid booking, no duplicate check-in	Reservation status validation

🧠 Best Practices for Data Validation

Practice	Benefit
Always validate server-side	Prevents bypassing client-side rules
Use centralized error handling	Cleaner code and consistent error messages
Use DTOs for validation	Avoid polluting domain models
Fail early, fail fast	Catch errors as soon as possible
Return descriptive errors	Improves user experience

🧼 A. Code Quality in Airline Management System

1. ✅ Code Structure and Organization

Best Practice	Description
Layered Architecture	Separate concerns into controller, service, repository, and model layers
Modular Design	Each feature (booking, flights, users) in its own package/module
Single Responsibility	Every class/method should have one job only

2. ✅ Naming Conventions

- Use meaningful names for classes, methods, and variables:
    - FlightController, not MainController
    - getAvailableFlights(), not function1()

3. ✅ Code Reusability

- Reuse components like validation logic, error handlers, and utility methods
- Use base classes or interfaces for shared behavior (e.g., BaseEntity, Auditable)

4. ✅ Code Readability

- Keep methods short and readable
- Comment only where necessary (when the logic isn't obvious)
- Follow Java coding standards (camelCase, PascalCase for classes)

5. ✅ Testing and Quality Checks

Type of Test	Purpose	Tools
Unit Testing	Test each class in isolation	JUnit, Mockito
Integration	Test interactions (e.g., booking +	Spring Test, TestContainers

Type of Test	Purpose	Tools
	payment)	
Code Coverage	Ensure all logic paths are tested	JaCoCo
Static Analysis	Detect bugs, smells, and security issues	SonarQube, PMD, Checkstyle

💡 B. Innovation in Airline Management System

Innovation can make your system stand out — both functionally and technically. Here are ways to innovate:

1. 🤖 Smart Booking Assistant

- Use **AI algorithms** or rules engines to recommend the cheapest or fastest route
- Implement auto-rebooking logic during flight cancellations or delays

Example:
"If Flight A is delayed, system automatically checks and offers alternative flights to the user."

2. 📈 Real-Time Flight Analytics Dashboard

- Integrate a **live dashboard** (using WebSocket or polling)
- Show stats like:
    - Most booked routes
    - Flight occupancy rate
    - Delayed flights

Tools: Spring Boot + WebSocket + Chart.js or React.js

3. 🧠 Predictive Maintenance Integration

- Use **sensor data or logs** (if integrating with aircraft systems) to predict failures
- Mark aircraft as unavailable in the system when issues are predicted

Innovation: Integrates IoT and predictive analytics

4. 🔔 Event-Driven Architecture with Notifications

- Use event handling (as discussed before) + notification service:
    - Email, SMS, push notifications
- Notify on:
    - Flight delays
    - Gate changes
    - Special offers

Tools: RabbitMQ or Kafka + Spring Cloud Stream

5. 🔐 Security Innovations

- **Role-based access control (RBAC)**
- Use **OAuth2 + JWT** for token-based auth
- Implement **audit logs** to track every critical action:
    - Who booked what?
    - Who cancelled a flight?

6. 📱 Progressive Web App (PWA) or Mobile-first Design

- Extend Java backend with a **React/Vue frontend or mobile app**
- Use RESTful APIs to serve mobile clients
- Enable offline ticket viewing or check-in

7. 🌍 Localization and Multi-Currency Support

- Add support for **multiple languages** and **currencies**
- Auto-detect user locale and adjust display accordingly

🧠 Combining Code Quality + Innovation

Objective	Code Practice	Innovation Opportunity
Maintainability	Modular, layered architecture	Plug-and-play modules (e.g., add new airline easily)
Scalability	REST APIs, microservices ready	Integrate cloud auto-scaling (AWS/GCP)
Performance	Efficient DB queries, caching	Use Redis for frequent data (e.g., flight lookup)
UX/User Delight	Fast APIs, proper validation	Smart assistant, voice support, dark mode

🧠CODE:

```
package airlinemanagementsystem;

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.sql.*;


public class Login extends JFrame implements ActionListener{

    JButton submit, reset, close;

    JTextField tfusername;

    JPasswordField tfpassword;


public Login() {

    getContentPane().setBackground(Color.WHITE);

    setLayout(null);
```

```
JLabel lblusername = new JLabel("Username");
lblusername.setBounds(20, 20, 100, 20);
add(lblusername);

tfusername = new JTextField();
tfusername.setBounds(130, 20, 200, 20);
add(tfusername);

JLabel lblpassword = new JLabel("Password");
lblpassword.setBounds(20, 60, 100, 20);
add(lblpassword);

tfpassword = new JPasswordField();
tfpassword.setBounds(130, 60, 200, 20);
add(tfpassword);

reset = new JButton("Reset");
reset.setBounds(40, 120, 120, 20);
reset.addActionListener(this);
add(reset);

submit = new JButton("Submit");
submit.setBounds(190, 120, 120, 20);
submit.addActionListener(this);
add(submit);
```

```
close = new JButton("Close");
close.setBounds(120, 160, 120, 20);
close.addActionListener(this);
add(close);

setSize(400, 250);
    setLocation(600, 250);
    setVisible(true);
}

public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == submit) {
        String username = tfusername.getText();
        String password = tfpassword.getText();

        try {
            Conn c = new Conn();

            String query = "select * from login where username = '"+username+"' and
password = '"+password+"'";

            ResultSet rs = c.s.executeQuery(query);
            if (rs.next()) {
                new Home();
                setVisible(false);
            } else {
```

```
                JOptionPane.showMessageDialog(null, "Invalid Username or
Password");
                setVisible(false);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else if (ae.getSource() == close) {
        setVisible(false);
    } else if (ae.getSource() == reset) {
        tfusername.setText("");
        tfpassword.setText("");
    }
}

    public static void main(String[] args) {
        new Login();
    }
}

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:///airlinemanagementsystem", "root",
"lallesh#01");
            s = c.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
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/fr
ont.jpg"));
        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");
        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();
        }
    }

    public static void main(String[] args) {
        new Home();
    }
}
```

```
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();
    }
```

```
}


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();
    }
}
```

```
package airlinemanagementsystem;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;

public class Cancel extends JFrame implements ActionListener{

    JTextField tfpnr;
    JLabel tfname, cancellationno, lblfcode, lbldateoftravel;
    JButton fetchButton, flight;
```

```
public Cancel() {
    getContentPane().setBackground(Color.WHITE);
    setLayout(null);

    Random random = new Random();

        JLabel heading = new JLabel("CANCELLATION");
        heading.setBounds(180, 20, 250, 35);
        heading.setFont(new Font("Tahoma", Font.PLAIN, 32));
        add(heading);

        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("airlinemanagementsystem/icons/c
ancel.jpg"));
        Image i2 = i1.getImage().getScaledInstance(250, 250,
Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel image = new JLabel(i3);
        image.setBounds(470, 120, 250, 250);
        add(image);

        JLabel lblaadhar = new JLabel("PNR Number");
        lblaadhar.setBounds(60, 80, 150, 25);
        lblaadhar.setFont(new Font("Tahoma", Font.PLAIN, 16));
        add(lblaadhar);

        tfpnr = new JTextField();
```

```
tfpnr.setBounds(220, 80, 150, 25);
add(tfpnr);

fetchButton = new JButton("Show Details");
fetchButton.setBackground(Color.BLACK);
fetchButton.setForeground(Color.WHITE);
fetchButton.setBounds(380, 80, 120, 25);
fetchButton.addActionListener(this);
add(fetchButton);

JLabel lblname = new JLabel("Name");
lblname.setBounds(60, 130, 150, 25);
lblname.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblname);

tfname = new JLabel();
tfname.setBounds(220, 130, 150, 25);
add(tfname);

JLabel lblnationality = new JLabel("Cancellation No");
lblnationality.setBounds(60, 180, 150, 25);
lblnationality.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblnationality);

cancellationno = new JLabel("" + random.nextInt(1000000));
cancellationno.setBounds(220, 180, 150, 25);
add(cancellationno);
```

```
JLabel lbladdress = new JLabel("Flight Code");
lbladdress.setBounds(60, 230, 150, 25);
lbladdress.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbladdress);

lblfcode = new JLabel();
lblfcode.setBounds(220, 230, 150, 25);
add(lblfcode);

JLabel lblgender = new JLabel("Date");
lblgender.setBounds(60, 280, 150, 25);
lblgender.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblgender);

lbldateoftravel = new JLabel();
lbldateoftravel.setBounds(220, 280, 150, 25);
add(lbldateoftravel);

flight = new JButton("Cancel");
flight.setBackground(Color.BLACK);
flight.setForeground(Color.WHITE);
flight.setBounds(220, 330, 120, 25);
flight.addActionListener(this);
add(flight);

setSize(800, 450);
```

```
        setLocation(350, 150);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == fetchButton) {
            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"));
                    lblfcode.setText(rs.getString("flightcode"));
                    lbldateoftravel.setText(rs.getString("ddate"));
                } else {
                    JOptionPane.showMessageDialog(null, "Please enter correct PNR");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else if (ae.getSource() == flight) {
            String name = tfname.getText();
```

```
String pnr = tfpnr.getText();

String cancelno = cancellationno.getText();

String fcode = lblfcode.getText();

String date = lbldateoftravel.getText();


try {

        Conn conn = new Conn();


        String query = "insert into cancel values('"+pnr+"', '"+name+"',
'"+cancelno+"', '"+fcode+"', '"+date+"')";


        conn.s.executeUpdate(query);

        conn.s.executeUpdate("delete from reservation where PNR = '"+pnr+"'");


            JOptionPane.showMessageDialog(null, "Ticket Cancelled");

            setVisible(false);


        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}


    public static void main(String[] args) {

        new Cancel();

    }

}
```

```
package airlinemanagementsystem;


import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.sql.*;

import com.toedter.calendar.JDateChooser;

import java.util.*;


public class BookFlight extends JFrame implements ActionListener{


    JTextField tfaadhar;

    JLabel tfname, tfnationality, tfaddress, labelgender, labelfname, labelfcode;

    JButton bookflight, fetchButton, flight;

    Choice source, destination;

    JDateChooser dcdate;


    public BookFlight() {

        getContentPane().setBackground(Color.WHITE);

        setLayout(null);


        JLabel heading = new JLabel("Book Flight");

        heading.setBounds(420, 20, 500, 35);

        heading.setFont(new Font("Tahoma", Font.PLAIN, 32));

        heading.setForeground(Color.BLUE);

        add(heading);
```

```
JLabel lblaadhar = new JLabel("Aadhar");
lblaadhar.setBounds(60, 80, 150, 25);
lblaadhar.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblaadhar);

tfaadhar = new JTextField();
tfaadhar.setBounds(220, 80, 150, 25);
add(tfaadhar);

fetchButton = new JButton("Fetch User");
fetchButton.setBackground(Color.BLACK);
fetchButton.setForeground(Color.WHITE);
fetchButton.setBounds(380, 80, 120, 25);
fetchButton.addActionListener(this);
add(fetchButton);

JLabel lblname = new JLabel("Name");
lblname.setBounds(60, 130, 150, 25);
lblname.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblname);

tfname = new JLabel();
tfname.setBounds(220, 130, 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("Address");
lbladdress.setBounds(60, 230, 150, 25);
lbladdress.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbladdress);

tfaddress = new JLabel();
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);

labelgender = new JLabel("Gender");
labelgender.setBounds(220, 280, 150, 25);
add(labelgender);

JLabel lblsource = new JLabel("Source");
```

```
lblsource.setBounds(60, 330, 150, 25);
lblsource.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblsource);

source = new Choice();
source.setBounds(220, 330, 150, 25);
add(source);

JLabel lbldest = new JLabel("Destination");
lbldest.setBounds(60, 380, 150, 25);
lbldest.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbldest);

destination = new Choice();
destination.setBounds(220, 380, 150, 25);
add(destination);

try {
    Conn c = new Conn();
    String query = "select * from flight";
    ResultSet rs = c.s.executeQuery(query);

    while(rs.next()) {
        source.add(rs.getString("source"));
        destination.add(rs.getString("destination"));
    }
```

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

flight = new JButton("Fetch Flights");
flight.setBackground(Color.BLACK);
flight.setForeground(Color.WHITE);
flight.setBounds(380, 380, 120, 25);
flight.addActionListener(this);
add(flight);

JLabel lblfname = new JLabel("Flight Name");
lblfname.setBounds(60, 430, 150, 25);
lblfname.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblfname);

labelfname = new JLabel();
labelfname.setBounds(220, 430, 150, 25);
add(labelfname);

JLabel lblfcode = new JLabel("Flight Code");
lblfcode.setBounds(60, 480, 150, 25);
lblfcode.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lblfcode);

labelfcode = new JLabel();
labelfcode.setBounds(220, 480, 150, 25);
```

```
add(labelfcode);


JLabel lbldate = new JLabel("Date of Travel");
lbldate.setBounds(60, 530, 150, 25);
lbldate.setFont(new Font("Tahoma", Font.PLAIN, 16));
add(lbldate);

        dcdate = new JDateChooser();
        dcdate.setBounds(220, 530, 150, 25);
        add(dcdate);

        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("airlinemanagementsystem/icons/d
etails.jpg"));
        Image i2 = i1.getImage().getScaledInstance(450, 320,
Image.SCALE_DEFAULT);
        ImageIcon image = new ImageIcon(i2);
        JLabel lblimage = new JLabel(image);
        lblimage.setBounds(550, 80, 500, 410);
        add(lblimage);


        bookflight = new JButton("Book Flight");
        bookflight.setBackground(Color.BLACK);
        bookflight.setForeground(Color.WHITE);
        bookflight.setBounds(220, 580, 150, 25);
        bookflight.addActionListener(this);
        add(bookflight);
```

```
        setSize(1100, 700);
        setLocation(200, 50);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == fetchButton) {
            String aadhar = tfaadhar.getText();

            try {
                Conn conn = new Conn();

                String query = "select * from passenger where aadhar = '"+aadhar+"'";

                ResultSet rs = conn.s.executeQuery(query);

                if (rs.next()) {
                    tfname.setText(rs.getString("name"));
                    tfnationality.setText(rs.getString("nationality"));
                    tfaddress.setText(rs.getString("address"));
                    labelgender.setText(rs.getString("gender"));
                } else {
                    JOptionPane.showMessageDialog(null, "Please enter correct aadhar");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
```

```
} else if (ae.getSource() == flight) {
    String src = source.getSelectedItem();
    String dest = destination.getSelectedItem();
    try {
        Conn conn = new Conn();
            String query = "select * from flight where source = '"+src+"' and destination
= '"+dest+"'";

            ResultSet rs = conn.s.executeQuery(query);

            if (rs.next()) {
                labelfname.setText(rs.getString("f_name"));
                labelfcode.setText(rs.getString("f_code"));
            } else {
                JOptionPane.showMessageDialog(null, "No Flights Found");
            }
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    Random random = new Random();
    String aadhar = tfaadhar.getText();
    String name = tfname.getText();
    String nationality = tfnationality.getText();
    String flightname = labelfname.getText();
```

```
            String flightcode = labelfcode.getText();

            String src = source.getSelectedItem();

            String des = destination.getSelectedItem();

            String ddate = ((JTextField)
dcdate.getDateEditor().getUiComponent()).getText();

            try {

                Conn conn = new Conn();


                String query = "insert into reservation
values('PNR-"+random.nextInt(1000000)+"', 'TIC-"+random.nextInt(10000)+"',
'"+aadhar+"', '"+name+"', '"+nationality+"', '"+flightname+"', '"+flightcode+"',
'"+src+"', '"+des+"', '"+ddate+"')";


                conn.s.executeUpdate(query);

                JOptionPane.showMessageDialog(null, "Ticket Booked Successfully");


                setVisible(false);

            } catch (Exception e) {

                e.printStackTrace();

            }

    }


    public static void main(String[] args) {

        new BookFlight();

    }
```

```
}

package airlinemanagementsystem;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;

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/a
irindia.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();
    }
}


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/e
mp.png"));
        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();
}
}
```

✅ 7. Project Documentation for Airline Management System (Java-based)

Project documentation is essential to ensure that your **Airline Management System** is understandable, maintainable, and scalable. Whether for developers, QA teams, or stakeholders, documentation provides clarity, structure, and a reference point for further development or deployment.

Below is a complete structure and guide for documenting a Java-based airline management system.

📚 Documentation Structure Overview

Section	Purpose
1. **Project Overview**	High-level summary of the system
2. **System Requirements**	Software, hardware, and dependencies
3. **Architecture & Design**	System design, components, and flow diagrams
4. **Module Descriptions**	Detailed explanation of each functional component
5. **Database Design**	ER diagrams, schema details
6. **API Documentation**	REST API specs, input/output, authentication
7. **Error Handling Strategy**	List of error types and response formats
8. **Testing Documentation**	Test plan, test cases, tools used
9. **Security Measures**	Authentication, encryption, access control
10. **Deployment Instructions**	How to install, configure, and run the system
11. **Future Enhancements**	Roadmap for improvements or features to be added

Section	Purpose
12. **Appendices**	Glossary, references, changelog

📝 1. Project Overview

- **Title**: Airline Management System
- **Tech Stack**: Java, Spring Boot, Hibernate, MySQL, JWT, Maven
- **Goal**: To allow airline staff and customers to manage flights, bookings, payments, and check-ins efficiently.

Key Features:

- User registration and login
- Flight scheduling and search
- Ticket booking and payment
- Real-time seat management
- Admin and crew management

🧾 2. System Requirements

A. Software Requirements

- Java 17+
- Spring Boot 3.x
- MySQL 8.x
- Apache Maven or Gradle
- Postman (for API testing)
- Git (version control)

B. Hardware Requirements

- Minimum 4 GB RAM (dev machine)
- 100 GB HDD
- Cloud deployment (optional): AWS/GCP with Docker

🏗️ 3. Architecture & Design

A. Architectural Pattern

- **Layered (MVC) Architecture**: Controller → Service → Repository → DB
- **Modular**: Independent modules for users, flights, bookings, etc.

B. Diagrams

- **Component Diagram**: Shows main system parts
- **Sequence Diagram**: Flight booking flow
- **Deployment Diagram**: Web server, DB, external APIs

🧩 4. Module Descriptions

1. User Management

- Register, login (JWT), role-based access (user/admin)

2. Flight Management

- Add/update flights, schedule aircraft, manage delays

3. Booking Module

- Search flights, book tickets, reserve seats

4. Payment Module

- Simulated payment API, transaction logging, payment status

5. Check-In Module

- Passenger check-in, seat confirmation, boarding pass issue

6. Notification System

- Email/SMS alerts using event-driven design (optional)

🗄️ 5. Database Design

A. ER Diagram

- Visual schema showing Users, Flights, Reservations, Payments, etc.

B. Table Definitions

- Users (id, name, email, role, password_hash)
- Flights (id, origin, destination, departure_time, aircraft_id)
- Reservations (id, user_id, flight_id, seat_number, status)
- Payments (id, reservation_id, amount, status)

🌐 6. API Documentation (Swagger / Postman)

Each endpoint should be documented with:

- URL path
- HTTP method
- Description
- Request example (JSON)
- Response format
- HTTP status codes

Tools:

- Swagger UI (springdoc-openapi)
- Postman collection with examples

⚠ 7. Error Handling Strategy

Error Code	Message	Module
400	Invalid input format	User, Booking
401	Unauthorized access	Login, Admin APIs
404	Resource not found	Flight, User
500	Internal server error	General catch-all

🧪 8. Testing Documentation

A. Test Plan

- Unit tests for each service class

- Integration tests for API flows (booking → payment → confirmation)
- Load testing for concurrent bookings

B. Tools

- JUnit 5
- Mockito
- Postman / REST Assured
- JMeter (for load testing)

🔐 9. Security Measures

- JWT-based authentication
- Password hashing (BCrypt)
- Role-based access control
- Input validation against injection
- HTTPS (for deployment)

🚀 10. Deployment Instructions

A. Steps to Run Locally

1. Clone repo
2. Set up MySQL DB
3. Configure application.properties
4. Run using mvn spring-boot:run
5. Access APIs at http://localhost:8080/api

B. Docker Deployment (Optional)

- Dockerfile
- docker-compose.yml with MySQL and App container

📈 11. Future Enhancements

- Integration with real-world payment gateway (Stripe, Razorpay)

- Mobile app (Android/iOS)
- Live flight tracking via third-party APIs
- Frequent flyer program
- Internationalization (multi-language support)

📎 12. Appendices

- **Glossary**: PNR, JWT, REST, etc.
- **References**: API docs, Spring Boot docs, MySQL docs
- **Changelog**: Date-wise list of changes/updates