Air-go Flight Reservation System – Functional Document

# Air-go is a full-stack airline reservation system designed to mimic the core functionality of real-world airline booking platforms. The system is built using Java Servlets, JSP, JDBC, and Hibernate, and follows a monolithic MVC architecture. Users can search for flights, make bookings, pay for reservations, and download booking confirmations. Role-based access ensures that managers and admins can maintain flight data, while regular users can perform flight reservations.

Air-go is a monolithic flight reservation system built using core Java technologies — Servlets, JSP, JDBC, and Hibernate. It simulates real-world airline booking with role-based access for users, managers, and admins.

# 👥 User Roles & Responsibilities

|  |  |
| --- | --- |
| Role | Responsibilities |
| User | Register, Login, Search Flights, Book Flights, Make Payment, Download Booking Confirmation |
| Manager | View/Edit Flights, Limited Admin Operations |
| Admin | Add/Edit/Delete Flights, View All Bookings, Manage Users |

# 🗂️ Functional Modules

## User Authentication

Input: username, password  
Output: Role-based dashboard (user.jsp, manager.jsp, admin.jsp)  
Flow: LoginServlet → UserServiceImpl → UserDao → OracleDB

## Flight Search

Input: source, destination, travel date  
Output: List of available flights (show-aval-flights.jsp)  
Flow: FlightSearchServlet → FlightService → FlightDao

## Booking Flow

Input: flight, user details, seat count  
Steps:  
 1. User submits form (booking-form.jsp)  
 2. BookingServlet creates BookingDetails  
 3. Data saved via Hibernate ORM  
 4. Session tracking  
Output: confirm.jsp + downloadable DOCX (via Apache POI)

## Payment Processing

Input: Payment details (dummy input)  
Output: Transaction record in DB  
Flow: PaymentServlet → TransactionService → TransactionDAO

## Flight Management (Admin/Manager)

Input: Flight ID, route, date, time, price  
Output: DB updates + view refreshed  
Screens: add-flight.jsp, edit-flight.jsp, admin.jsp, manager.jsp

# 📦 Entities and Relationships

|  |  |
| --- | --- |
| Entity | Description |
| Users | id, name, password, role |
| Flight | id, source, destination, scheduleTime, price |
| BookingDetails | id, user, flight, seat count, travel date |
| Transactions | id, booking, amount, timestamp |

Relationships:  
- Users → BookingDetails: 1-to-Many  
- BookingDetails → Transactions: 1-to-1

# 🧩 Architecture Overview

• Presentation: JSP Pages  
• Controller: Java Servlets  
• Business Logic: Service Layer (interfaces + impl)  
• Persistence: JDBC (for login, user, flights), Hibernate (for booking, transactions)

# 🧠 Session Handling

• Logged-in users are tracked using HTTP session.  
• Session stores role and user ID.  
• No timeout or expiration yet (planned improvement).

The Air-go project is structured into clear layers for separation of concerns:  
  
• JSP (View Layer): Presents HTML forms to users, collects data, and displays results.  
• Servlets (Controller Layer): Acts as the entry point for all user actions. Validates input, manages sessions, and delegates business logic.  
• Service Layer: Contains business logic and uses interfaces to decouple from DAO layer.  
• DAO Layer: Performs database operations using JDBC and Hibernate.  
• Hibernate ORM: Handles object-relational mapping for BookingDetails and Transactions.

# 🔁 Data Flow (Layer-wise)

# 🔐 Known Limitations

• Passwords stored in plaintext (need hashing)

• No form validation or CSRF/XSS protection

• Mixed use of JDBC and Hibernate — needs consistency

• No unit testing or logging implemented

• No REST API layer for frontend independence

• Session timeout not implemented

# 🔧 Tools & Stack

• Java 1.8  
• Servlets + JSP  
• Oracle DB  
• Hibernate ORM  
• Apache POI (Docx generation)  
• JUnit (planned)  
• Tomcat 9, Eclipse IDE

# 🚀 How to Run

1. Clone the repo: git clone https://github.com/GoluCode/Air-go-Backend-Project  
2. Open in Eclipse as a Dynamic Web Project  
3. Configure Tomcat 9  
4. Edit DB config in hibernate.cfg.xml  
5. Start from index.jsp

# 📊 Future Work

• REST APIs using Spring Boot

• Password encryption & login security

• Admin dashboard analytics

• Booking cancellation feature

• Seat availability management

• JUnit + Mockito test coverage

# 🧑‍💻 Author

Shyam Sunder Roy  
Backend enthusiast passionate about building real systems from scratch.

# 🧠 Developer's Learning Journey

This project was built from scratch without initial exposure to formal design patterns. However, through intuition and continuous iteration, it evolved into a clean monolithic MVC system. Each feature was developed to solve a real-world problem in flight booking, such as seat selection, payment tracking, and role-based permissions. Over time, the developer identified architectural limitations and started planning for Spring Boot migration and modularization.