INFOSYS

SPRINGBOARD

COMPUTER SCIENCE AND SYSTEMS ENGINEERING

FLIGHT MANAGEMENT SYSTEM

CRAFTED BY

D.LALLITESH
K. DINESH REDDY
P. JAYA NAGA VARA PRASAD
G. ADISESHU

Mentor: Suramya Biswas

INFOSYS SPRINGBOARD INTERNSHIP

Project Documentation for Flight Management System

.

ABSTRACT

1. This document outlines a Java-based flight management system designed for booking and managing flight reservations. Users can create accounts, search for flights, book and manage their itineraries, including cancellations and modifications. Administrators have access to a comprehensive control panel for managing flight details, schedules, and adding new flights.

2. The system prioritizes user experience by offering functionalities like searching for available flights, selecting preferred travel dates, and managing bookings. Robust validation ensures data accuracy during the booking process. For administrators, the system provides functionalities to manage the entire flight lifecycle. They can add flight details, set schedules, and control availability. The system enforces validations to ensure schedule conflicts are avoided and maintains passenger capacity limits.

3. While the core functionalities revolve around flight management, some aspects are left for future development. These include boarding pass generation, seat selection, and integration with third-party services for email and SMS notifications, as well as payment processing.

INTRODUCTION

A Win-Win Solution - The Modern Flight Management System (FMS)

The aviation industry is constantly evolving, striving to bridge the gap between passenger convenience and airline efficiency. Traditional flight booking systems, reliant on manual processes and limited information, often created a frustrating experience for both travelers and airlines. To address these challenges, a new generation of Flight Management Systems (FMS) has emerged, offering a more streamlined and user-friendly approach.

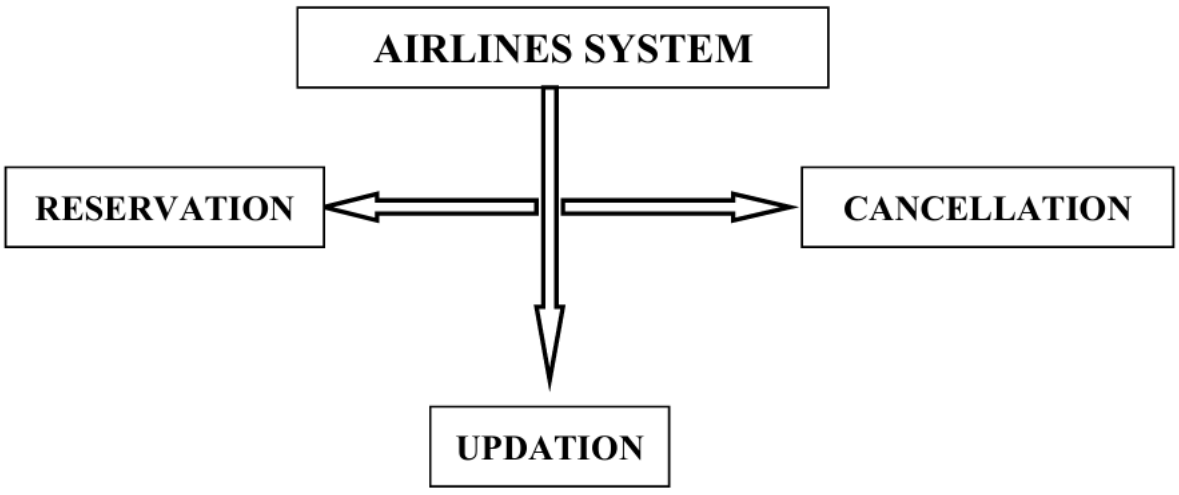
This project outlines the development of a sophisticated, Java-based FMS designed to revolutionize flight booking and management. This system offers a significant advantage over older methods by providing a user-centric interface, real-time data access, and comprehensive tools for both travelers and airlines.

- **Empowering the Customer:** A Smoother Journey from Search to Booking. For travelers, the new FMS translates to a more

convenient and empowering booking experience. Here's how it streamlines the process:

- Effortless Search and Comparison: Real-time access to flight information allows travelers to search for flights across airlines, compare prices and schedules, and find the best deals in a matter of minutes. Gone are the days of relying on travel agents or limited brochures.
- Simplified Booking Process: The user-friendly interface eliminates the need for phone calls or multiple website visits. Travelers can book flights directly through the platform, managing their entire itinerary with ease.
- Increased Transparency: Real-time availability and pricing data empower travelers to make informed decisions. They can see the latest information on flight options, ensuring they get the best value for their trip.
- Enhanced Control: The system allows travelers to manage their bookings seamlessly. They can easily modify travel details, check-in online, and access trip information readily.

Increased Revenue Potential: The user-friendly interface and real-time information encourage travelers to explore more options and potentially book additional services like baggage allowance or seat selection, leading to increased revenue streams for airlines.



Software Requirements Specifications (SRS) is a complete specification and description of software requirements that must be fulfilled to develop the software system successful. It comprises user requirements for a system and detailed specifications of the system requirements. This report lays a foundation for software engineering activities and is constructed when requirements are elicited and analyzed.

REQUIREMENTS SPECIFICATION

Functional Requirements:

- User Authentication and Registration: This functionality establishes a secure system by allowing users to create accounts and login with usernames and passwords (or other credentials).

- Booking: This function allows users to search for available flights based on their desired travel dates, origin and destination airports, and potentially other criteria like preferred airlines or travel times. Once a suitable flight is found, users can proceed to book the flight, specifying the number of passengers and potentially selecting additional services.

- Passenger: This functionality manages passenger information associated with a booking. It allows users to enter passenger details like names, contact information, and potentially Unique Identification Numbers (depending on the system's requirements).

- Flight: This functionality deals with managing flight information within the system. It allows administrators to define flight details such as flight numbers, airlines, departure and arrival airports, flight times, aircraft types, seat capacities, and potentially pricing information.

- Schedule Flight: This functionality allows administrators to define the schedule for a particular flight. This includes specifying the date and time of departure and arrival for each flight leg. The system might also consider integrating this functionality with managing available seats to avoid overbooking.

- Airport / Airport Schedule: This functionality manages information about airports within the system. It allows administrators to define airport details such as airport codes, names, locations, and potentially terminal information. The "Airport Schedule" aspect might refer to managing the arrival and departure schedules of various flights for each airport.

Non-Functional Requirements:

- Performance: The system should be very highly performable so that number of users

can access it any time without any lag.

- ❖ Security: The system should be very secure as it contains many important documents, and it should be secure against attacks

- ❖ Usability: The system should be user-friendly with simple user interface so that many people can access it easily.

- ❖ Reliability: The system should be reliable, that is it should work more with less maintenance or service and take less time to recover.

- ❖ Compatibility: As there are number of operating systems and web browsers the system should handle and support in all of them.

- ❖ Maintainability: The system should be easy to maintain, with easy code and possible upgrades

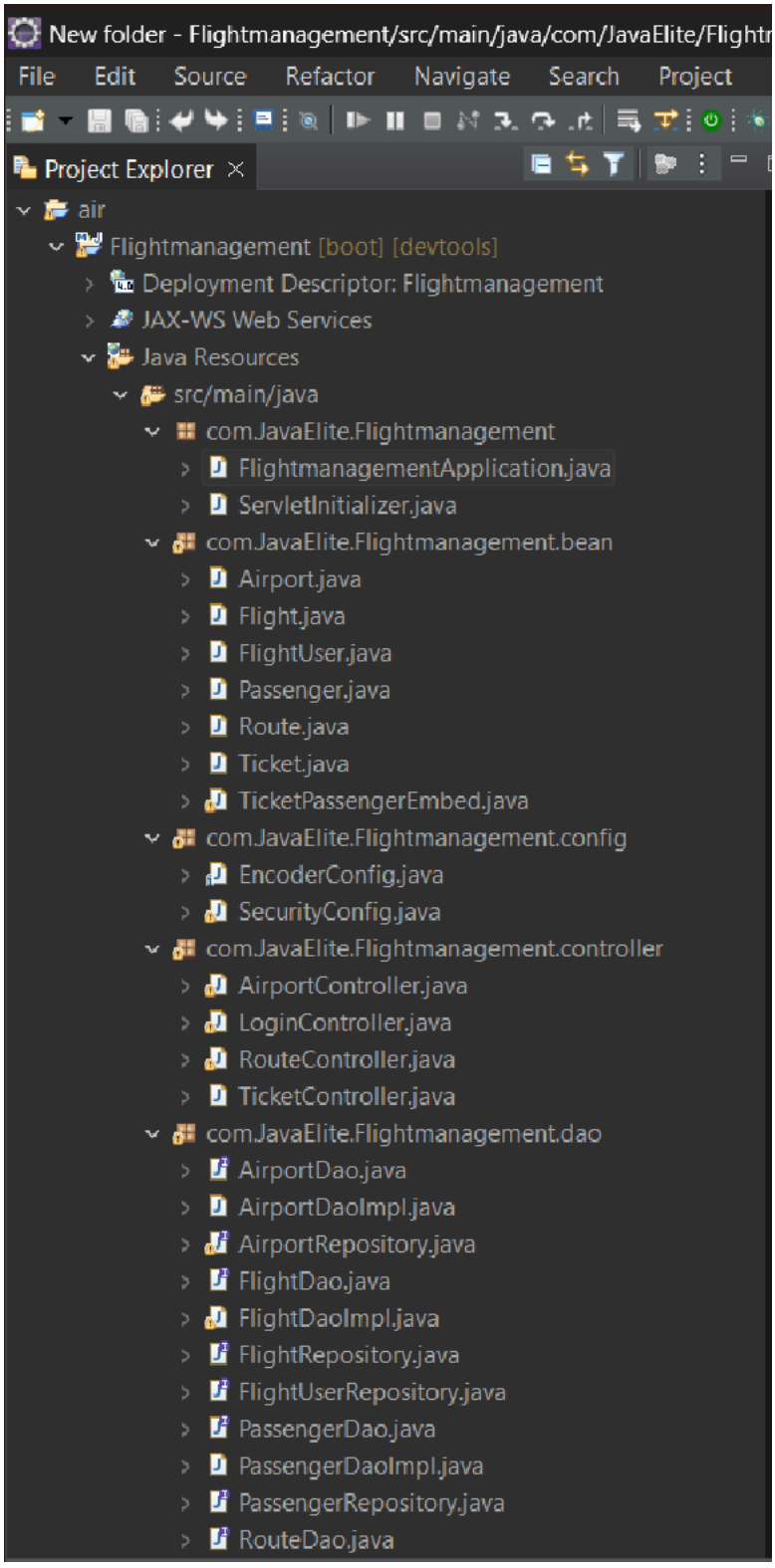
Technologies Used:

1. Backend: Spring Boot, Java, Maven
2. Frontend: HTML,CSS
3. Database: SQL (e.g., MySQL, PostgreSQL)
4. Dependencies: Spring Data JPA, SQL Driver, etc.

2. Project Setup (Prerequisites & Tools):

1. Java Development Kit (JDK)-
2. Maven
3. An SQL database (MySQL, PostgreSQL, etc.)
4. IDE (IntelliJ, Eclipse, Spring STS, etc.)
5. Spring STS 4(Version) - 4.22.

Code Implementation: -

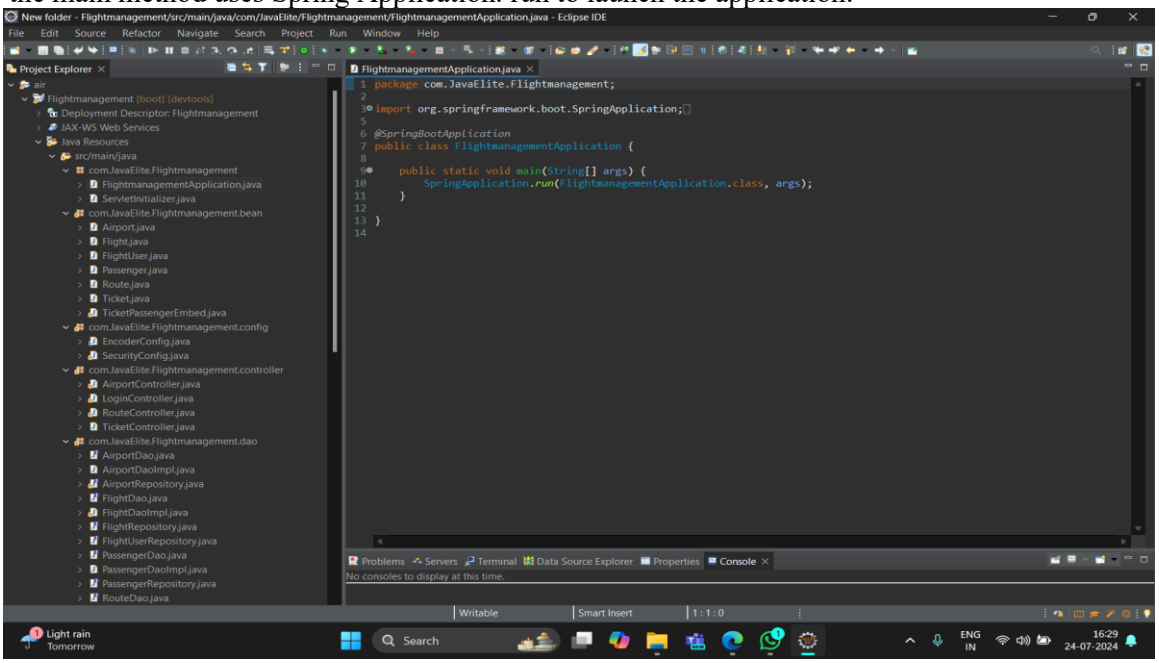


This Picture depicts the whole project directory (Which Shows the full Folder/Files).

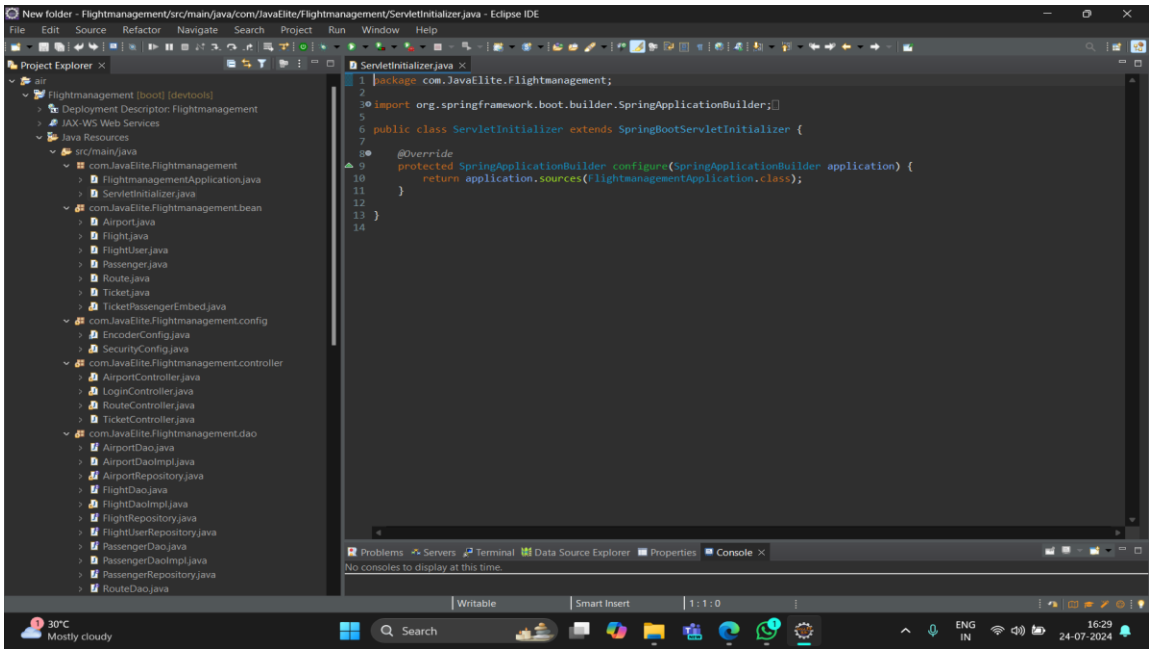
Detailed Explanation of Key Files and Folders:

1)Java Elite/ Flight management package:

- **FlightmanagementApplication.java:** the code snippet appears to be the starting point of a Spring Boot application named Flight management. The Flight management Application class is annotated with @SpringBootApplication, indicating it's the main class, and the main method uses Spring Application. run to launch the application.

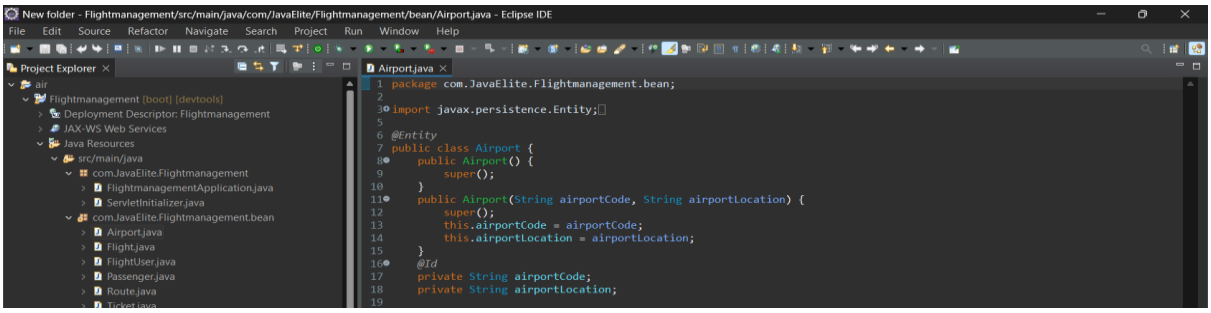


- **Servlet Initializer:** this class allows the Spring Boot application to be deployed and run in a servlet container, enabling the Spring Boot application to initialize correctly within that environment.

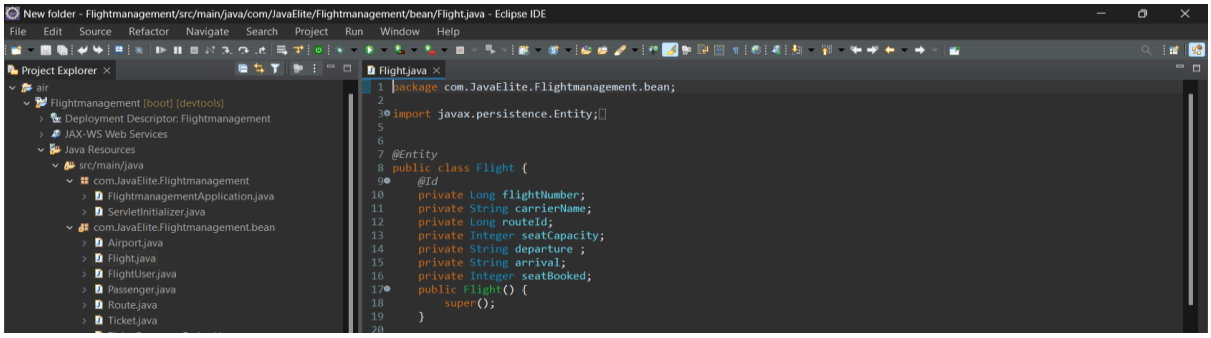


2)Java Elite. Flight management. Bean: -

- **Airport.java:** This class represents an entity in a flight management application, likely mapping to a database table for airports. The `airport` class is a JPA entity that maps to a database table representing airports.

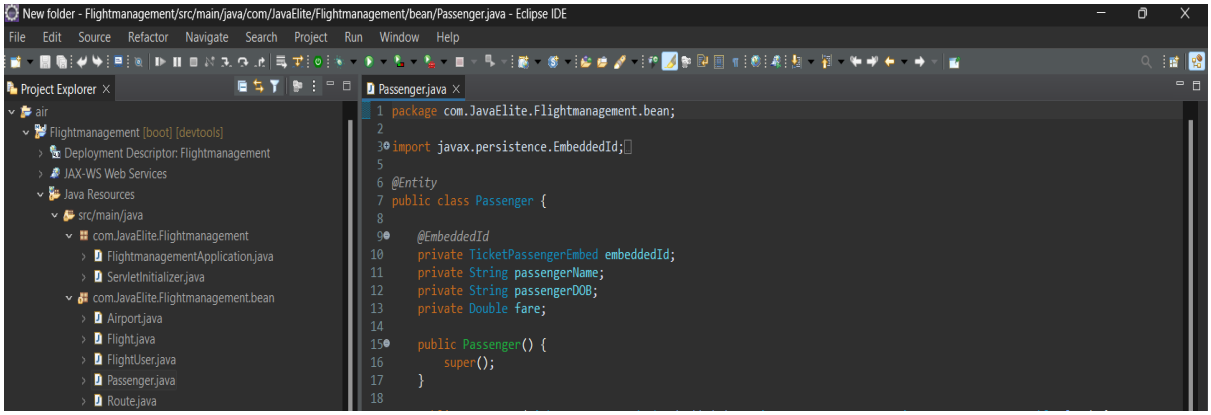


- **Flight.java:** This class represents an entity in a flight management application, likely mapping to a database table for flights. Fields for flight Number (primary key), carrier Name, route Id, seat Capacity, departure, arrival, and seat Booked.

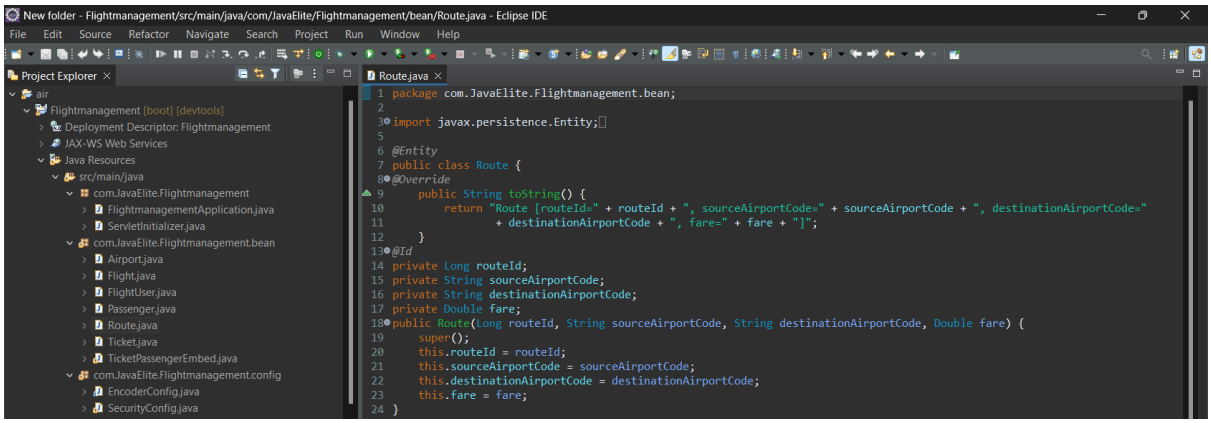


- **Flight User. java:** The class represent a user in a flight management system, with properties for username, password, and user type. The @Entity annotation suggests that instances of this class are mapped to a database table.

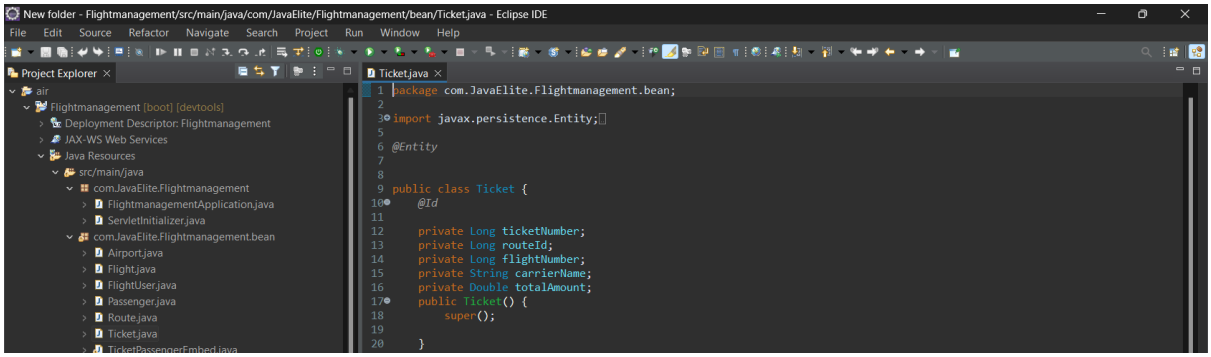
- **Passenger.java:** This class represents a passenger in the flight management system, with attributes such as an embedded ID (likely a composite key for database purposes), name, date of birth, and fare. The class includes constructors for creating Passenger objects and getter/setter methods for accessing and modifying the fields.



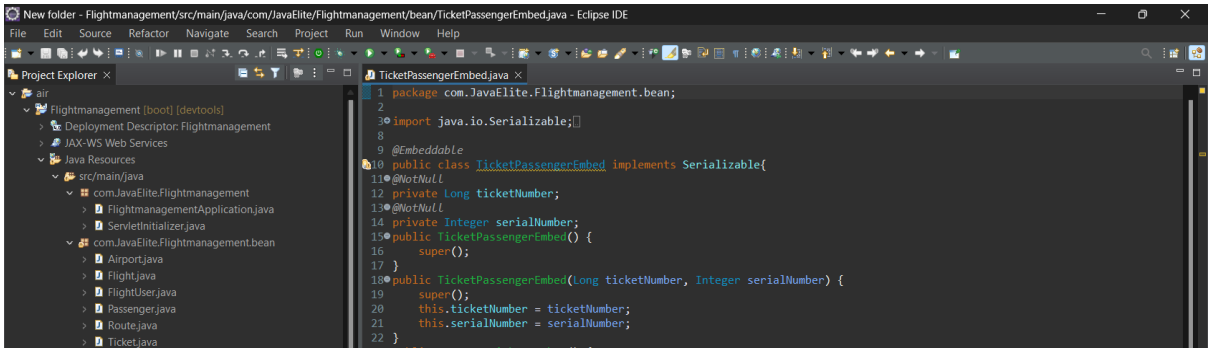
- **Route.java:** The Route class represents a flight route within the Flight Management System. It encapsulates the details of a route including the unique route identifier, source and destination airport codes, and the fare for the route.



- **Ticket.java:** This class is used to represent flight tickets in the Flight Management System. Instances of this class can be created, persisted, and retrieved from a database. The class provides encapsulated access to its fields through getters and setters, ensuring data integrity and allowing for easy manipulation.

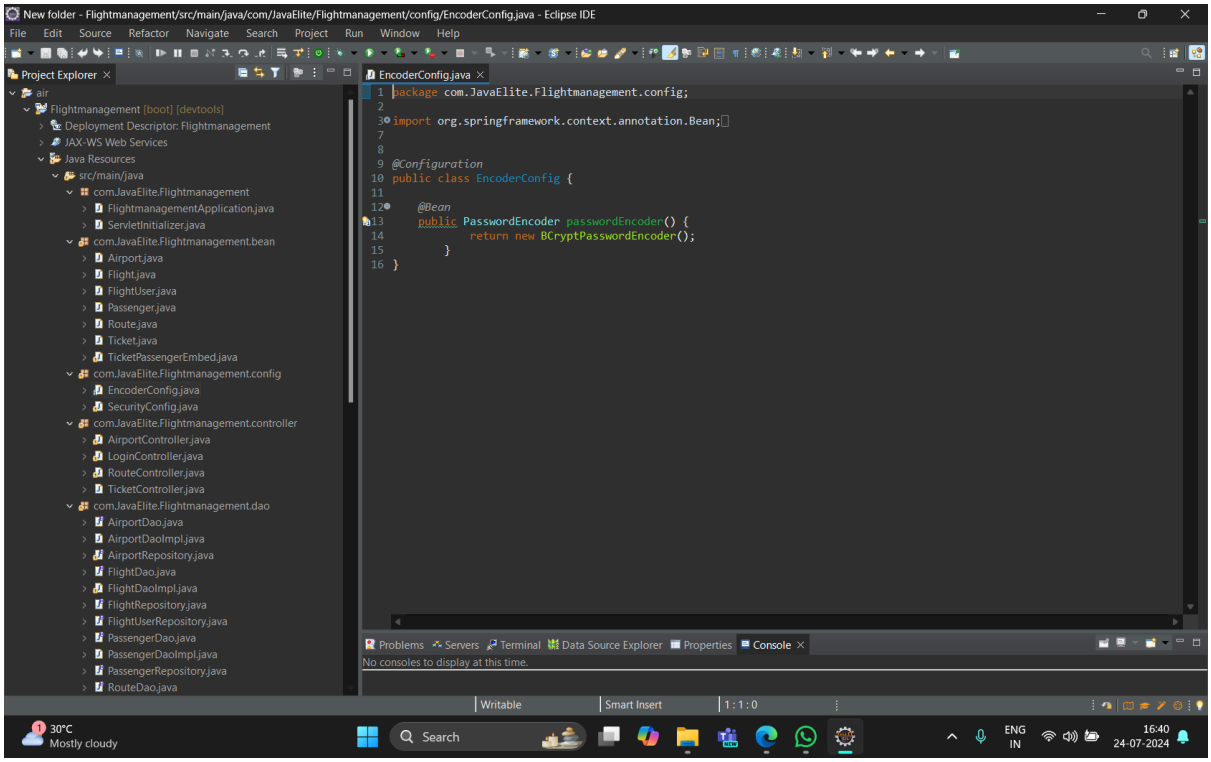


- **Ticket passenger embed.java**: The Ticket Passenger Embed class represents a composite primary key for the relationship between tickets and passengers within the Flight Management System. It encapsulates the ticket number and the serial number of the passenger on that ticket. This class is marked as @Embeddable, indicating that it can be embedded in another entity as a composite primary key**.**

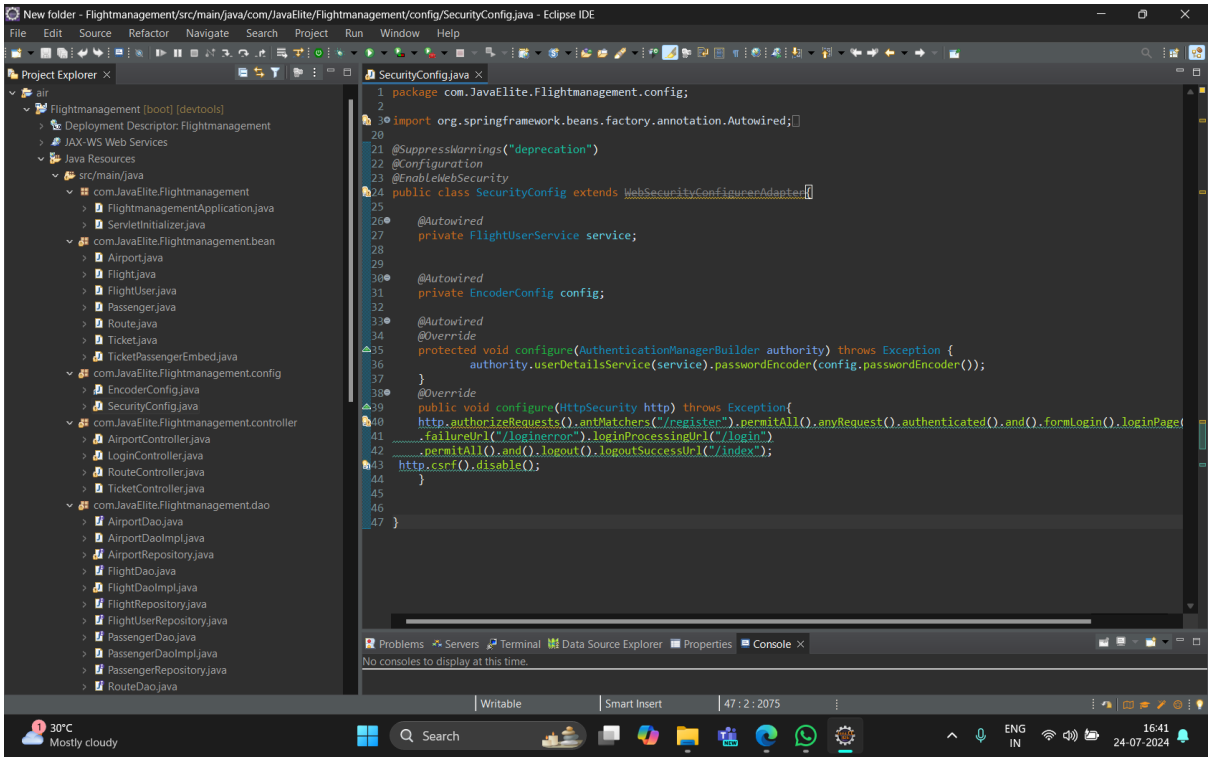


3)Flight Management. Config:-

- **Encoderconfig.java:** The Encoder Config class is a configuration class for setting up password encoding in the Flight Management System. It provides a Password Encoder bean that is used to encode passwords using the B Crypt hashing algorithm.

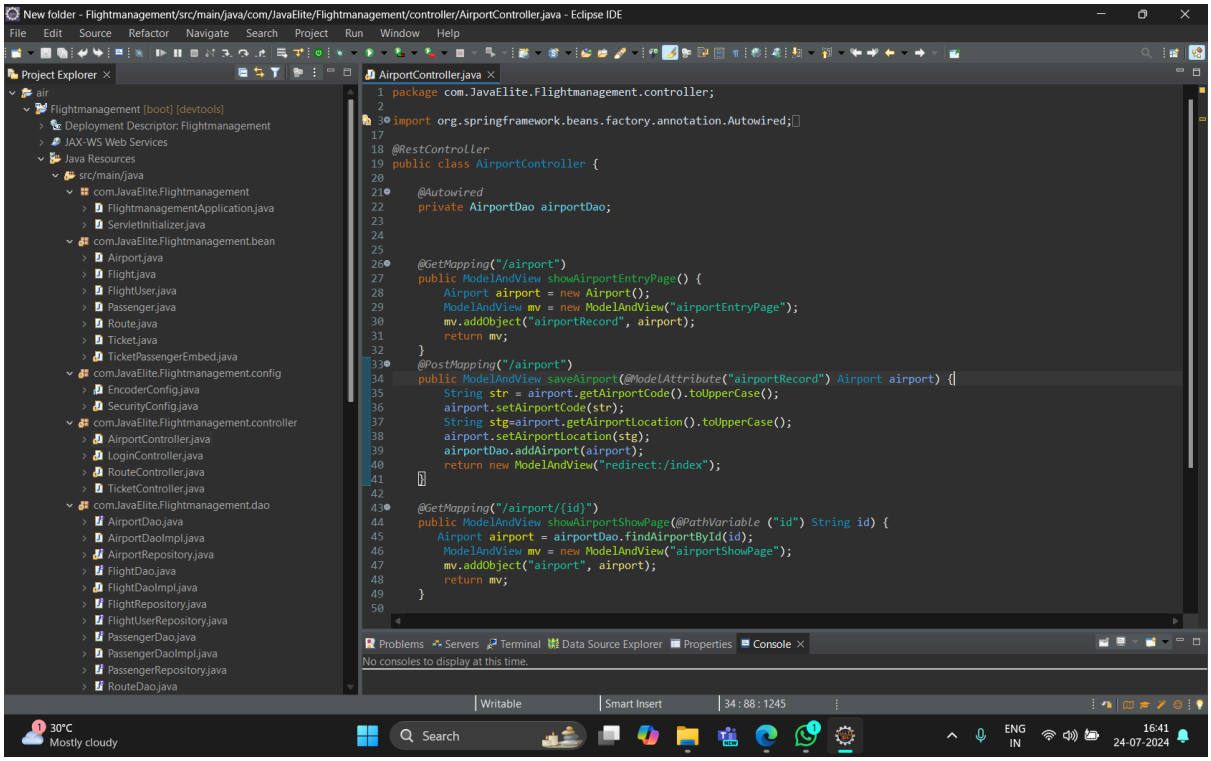


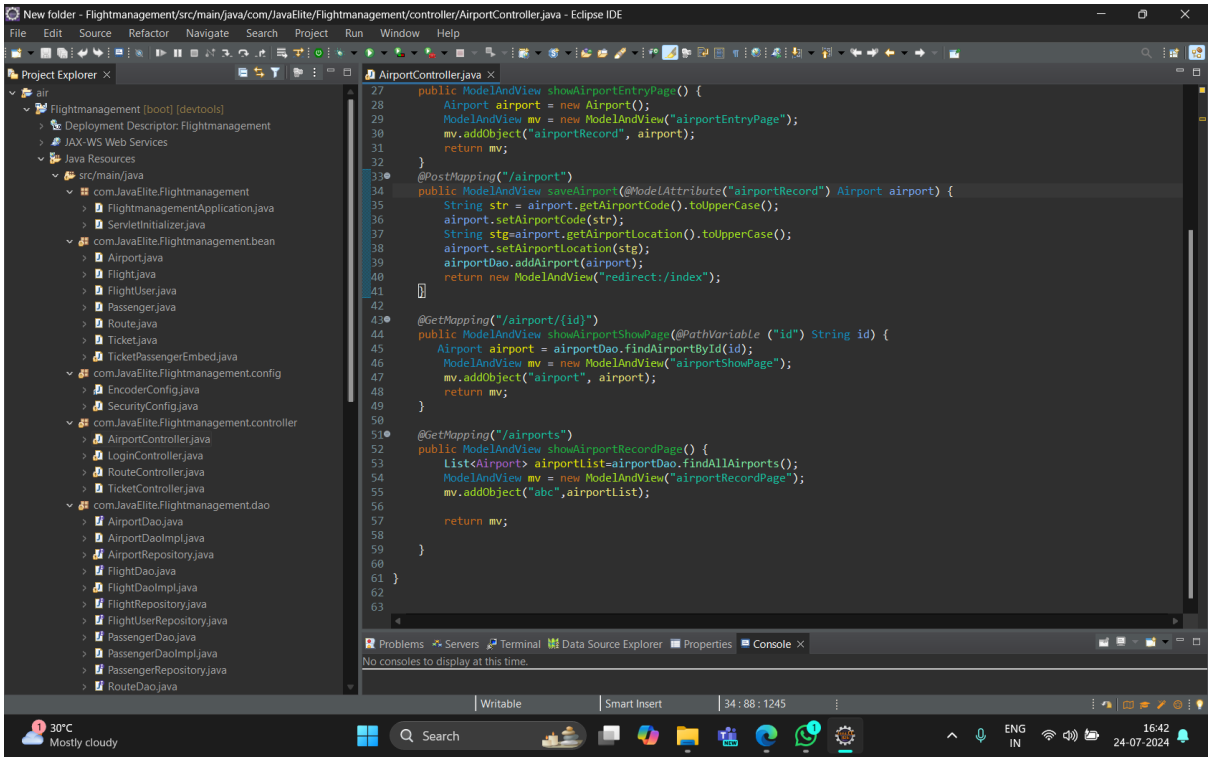
- **Security config:** The Security Config class is a configuration class for setting up security in the Flight Management System. It extends Web Security Configure Adapter and uses Spring Security to configure authentication and authorization mechanisms.



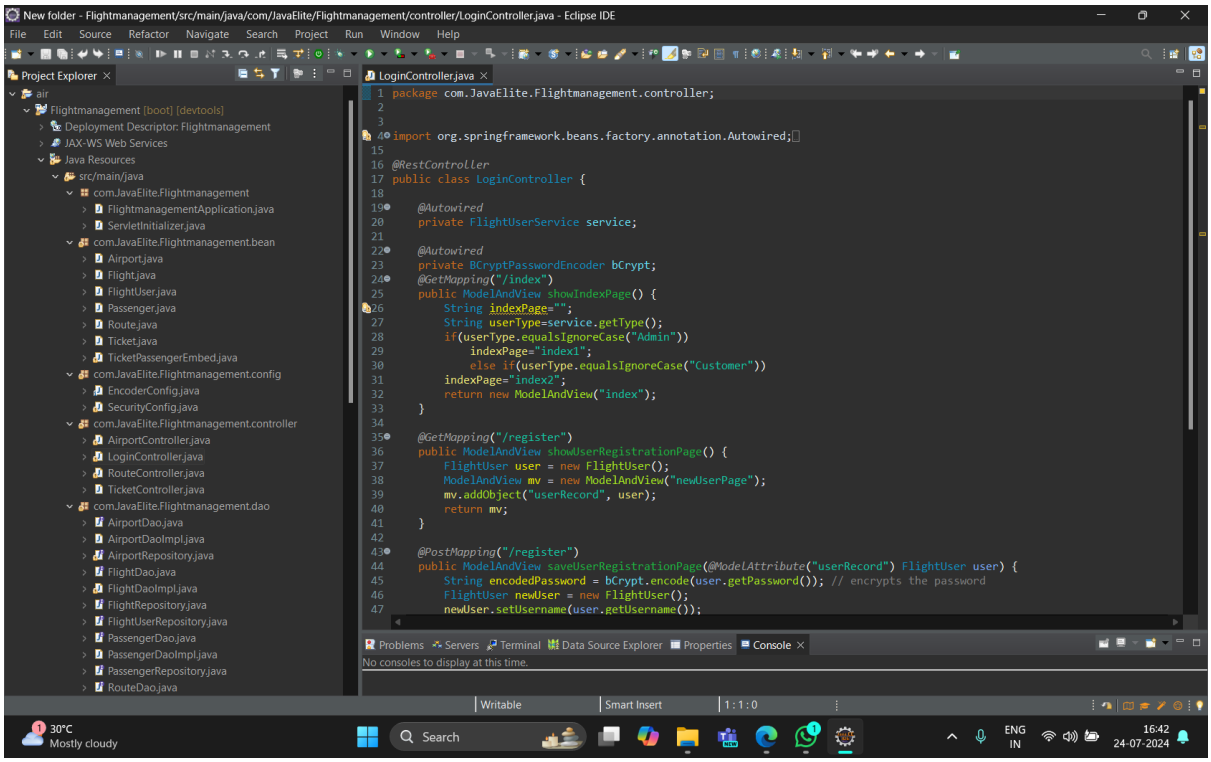
4)Flight Management. controller:

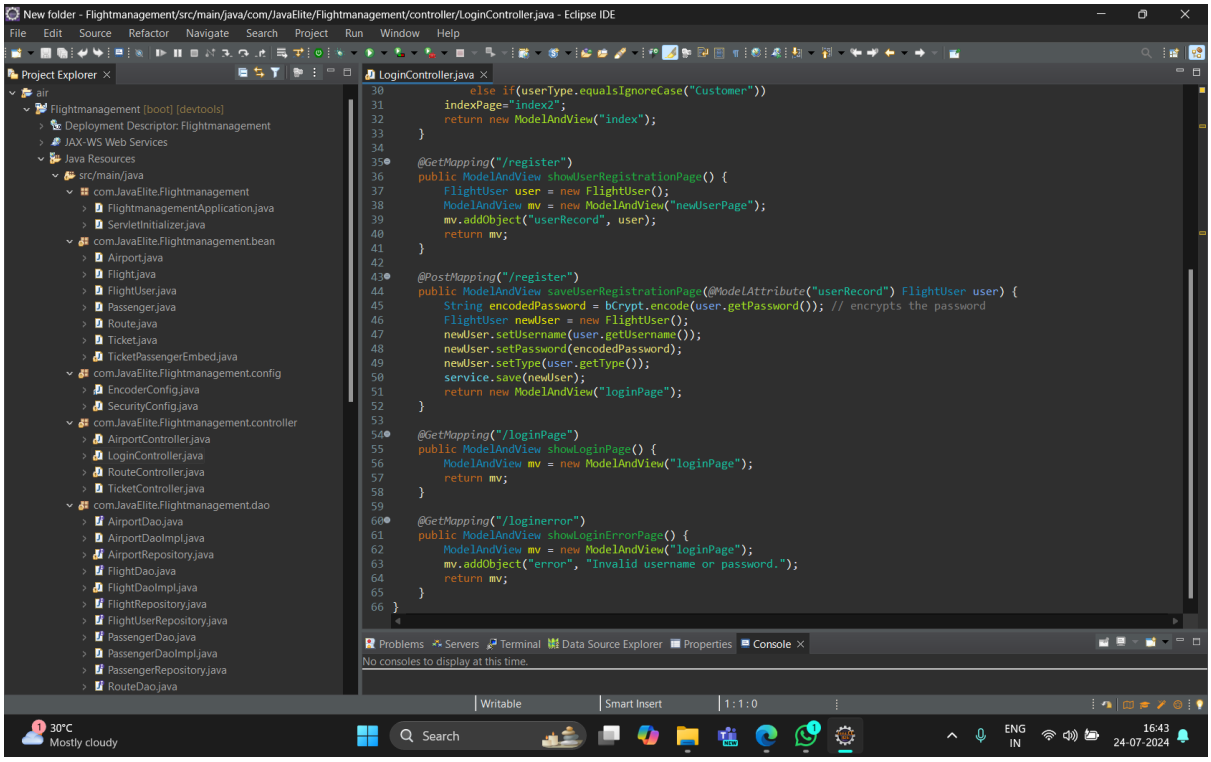
- **Airport controller:** The Airport Controller class is a REST controller that handles HTTP requests related to airport management in the Flight Management System. It provides endpoints to create, view, and list airports. This class uses Spring MVC annotations to map web requests to handler methods. This class is used to manage airport-related web requests in the Flight Management System. It provides endpoints for creating new airports, viewing airport details, and listing all airports.


New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/AirportController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
AirportController.java
1 package com.JavaElite.Flightmanagement.controller;
3 import org.springframework.beans.factory.annotation.Autowired;
18 @RestController
19 public class AirportController {
21 @Autowired
22 private AirportDao airportDao;
26 @GetMapping("/airport")
27 public ModelAndView showAirportEntryPage() {
28 Airport airport = new Airport();
29 ModelAndView mv = new ModelAndView("airportEntryPage");
30 mv.addObject("airportRecord", airport);
31 return mv;
32 }
33 @PostMapping("/airport")
34 public ModelAndView saveAirport(@ModelAttribute("airportRecord") Airport airport) {
35 String str = airport.getAirportCode().toUpperCase();
36 airport.setAirportCode(str);
37 String stg=airport.getAirportLocation().toUpperCase();
38 airport.setAirportLocation(stg);
39 airportDao.addAirport(airport);
40 return new ModelAndView("redirect:/index");
41 }
43 @GetMapping("/airport/{id}")
44 public ModelAndView showAirportShowPage(@PathVariable ("id") String id) {
45 Airport airport = airportDao.findAirportById(id);
46 ModelAndView mv = new ModelAndView("airportShowPage");
47 mv.addObject("airport", airport);
48 return mv;
49 }
Problems Servers Terminal Data Source Explorer Properties Console
No consoles to display at this time.
Writable Smart Insert 34 : 88 : 1245
30°C Mostly cloudy
Search
ENG IN
16:41 24-07-2024

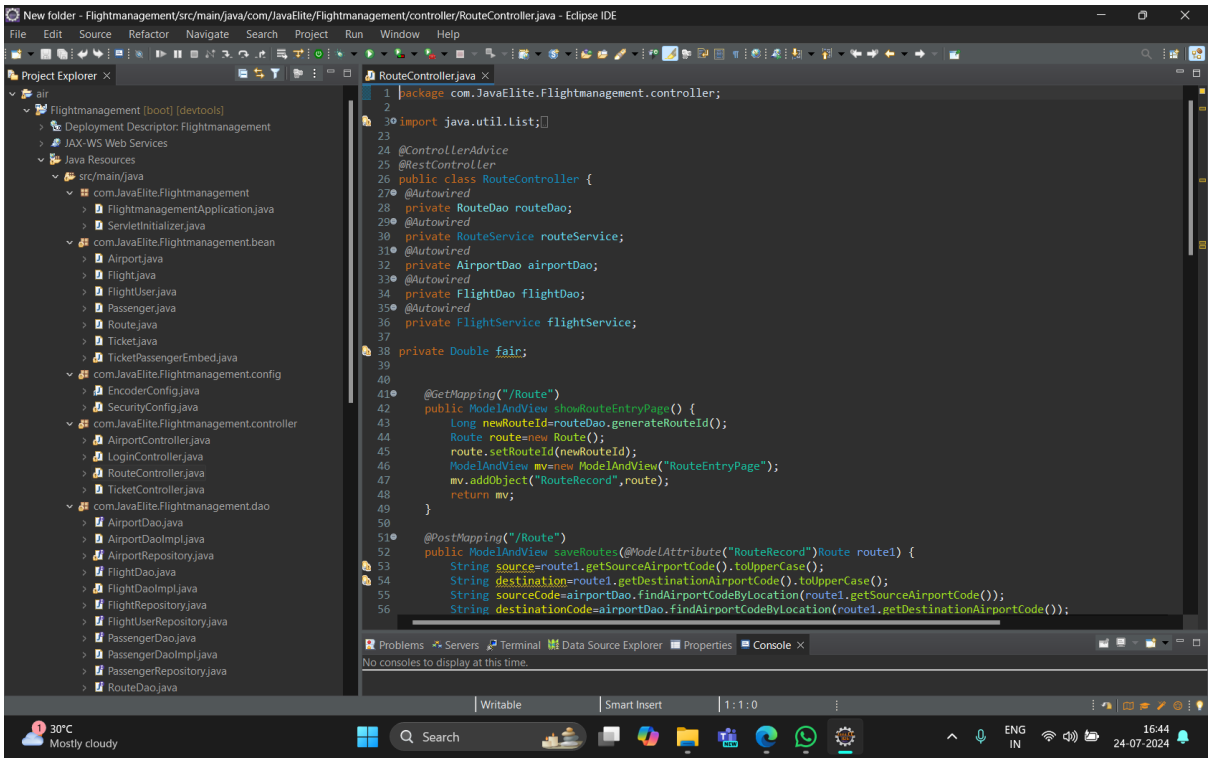

New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/AirportController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
AirportController.java
27 public ModelAndView showAirportEntryPage() {
28 Airport airport = new Airport();
29 ModelAndView mv = new ModelAndView("airportEntryPage");
30 mv.addObject("airportRecord", airport);
31 return mv;
32 }
33 @PostMapping("/airport")
34 public ModelAndView saveAirport(@ModelAttribute("airportRecord") Airport airport) {
35 String str = airport.getAirportCode().toUpperCase();
36 airport.setAirportCode(str);
37 String stg=airport.getAirportLocation().toUpperCase();
38 airport.setAirportLocation(stg);
39 airportDao.addAirport(airport);
40 return new ModelAndView("redirect:/index");
41 }
43 @GetMapping("/airport/{id}")
44 public ModelAndView showAirportShowPage(@PathVariable ("id") String id) {
45 Airport airport = airportDao.findAirportById(id);
46 ModelAndView mv = new ModelAndView("airportShowPage");
47 mv.addObject("airport", airport);
48 return mv;
49 }
51 @GetMapping("/airports")
52 public ModelAndView showAirportRecordPage() {
53 List<Airport> airportList=airportDao.findAllAirports();
54 ModelAndView mv = new ModelAndView("airportRecordPage");
55 mv.addObject("abc",airportList);
57 return mv;
59 }
61 }
Problems Servers Terminal Data Source Explorer Properties Console
No consoles to display at this time.
Writable Smart Insert 34 : 88 : 1245
30°C Mostly cloudy
Search
ENG IN
16:42 24-07-2024

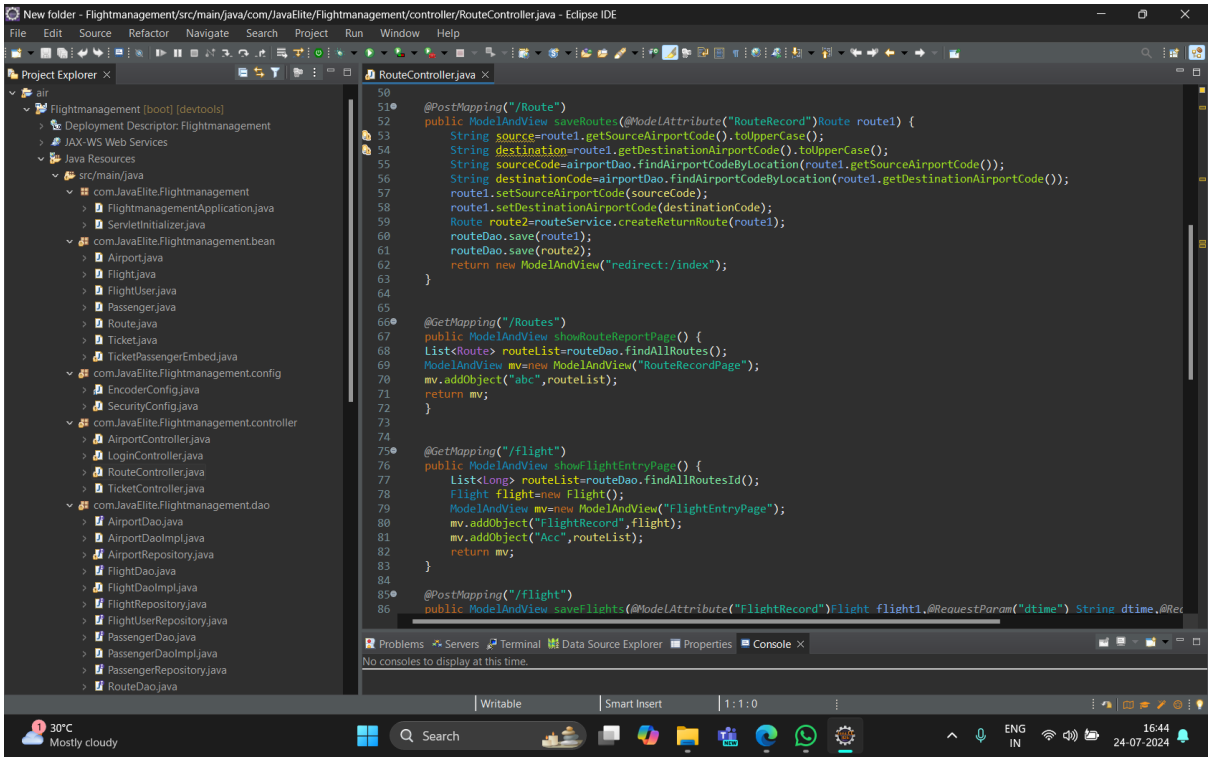
- **Login controller:** The Login Controller class handles user authentication and registration-related HTTP requests in the Flight Management System. It provides endpoints for user registration, login, and displaying relevant pages.

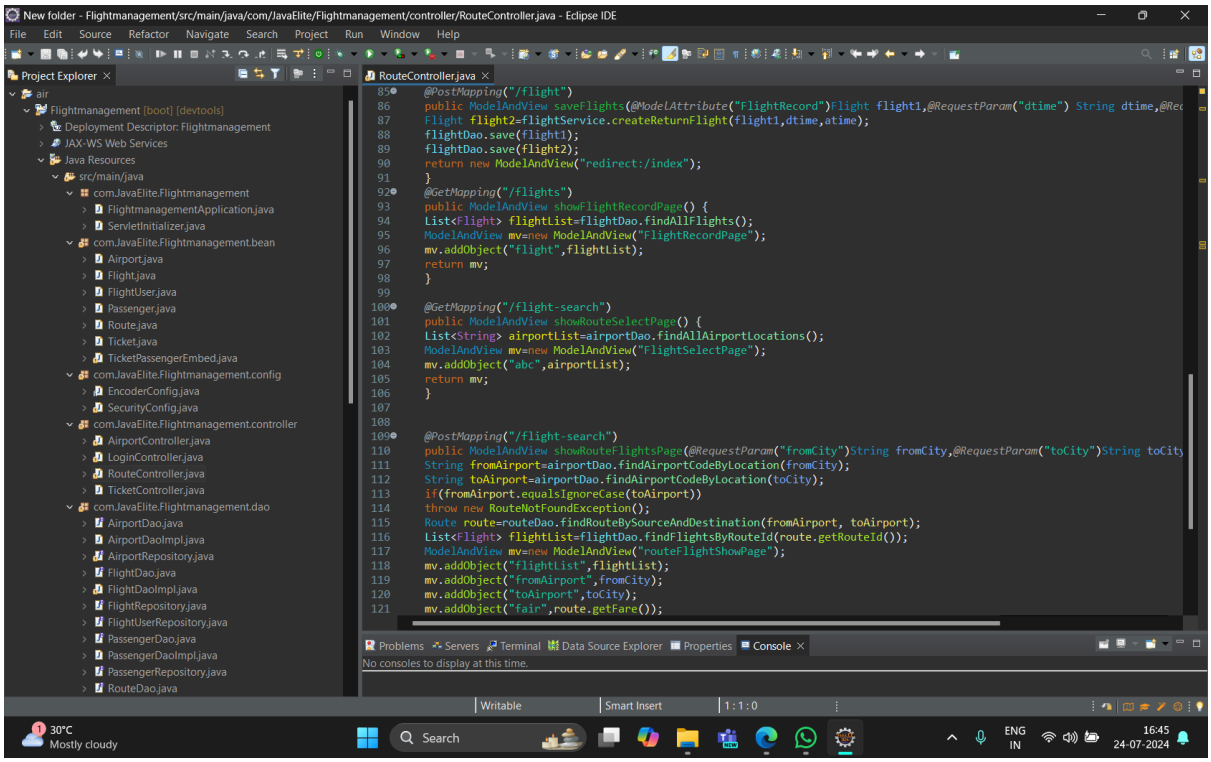


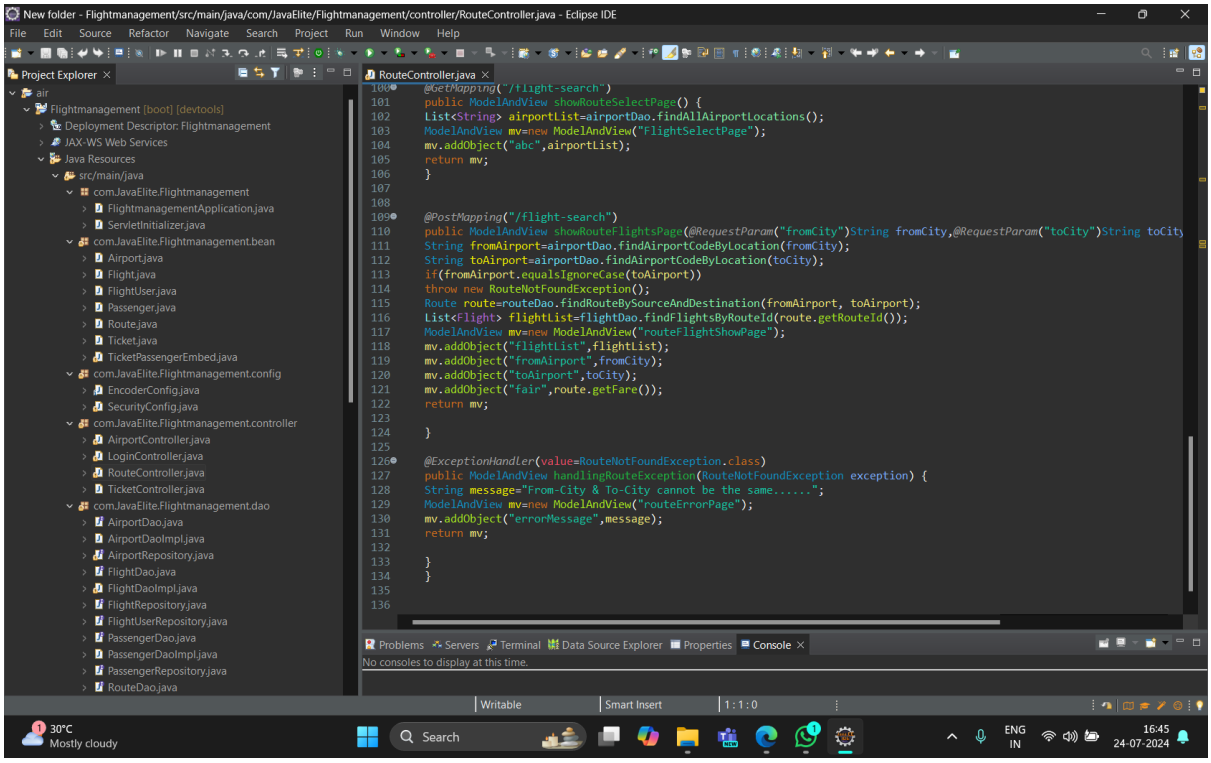


- **Route controller:** The Route Controller class is used to manage routes and flights within the Flight Management System. It provides functionality for creating, viewing, and searching routes and flights, as well as handling exceptions related to routing.

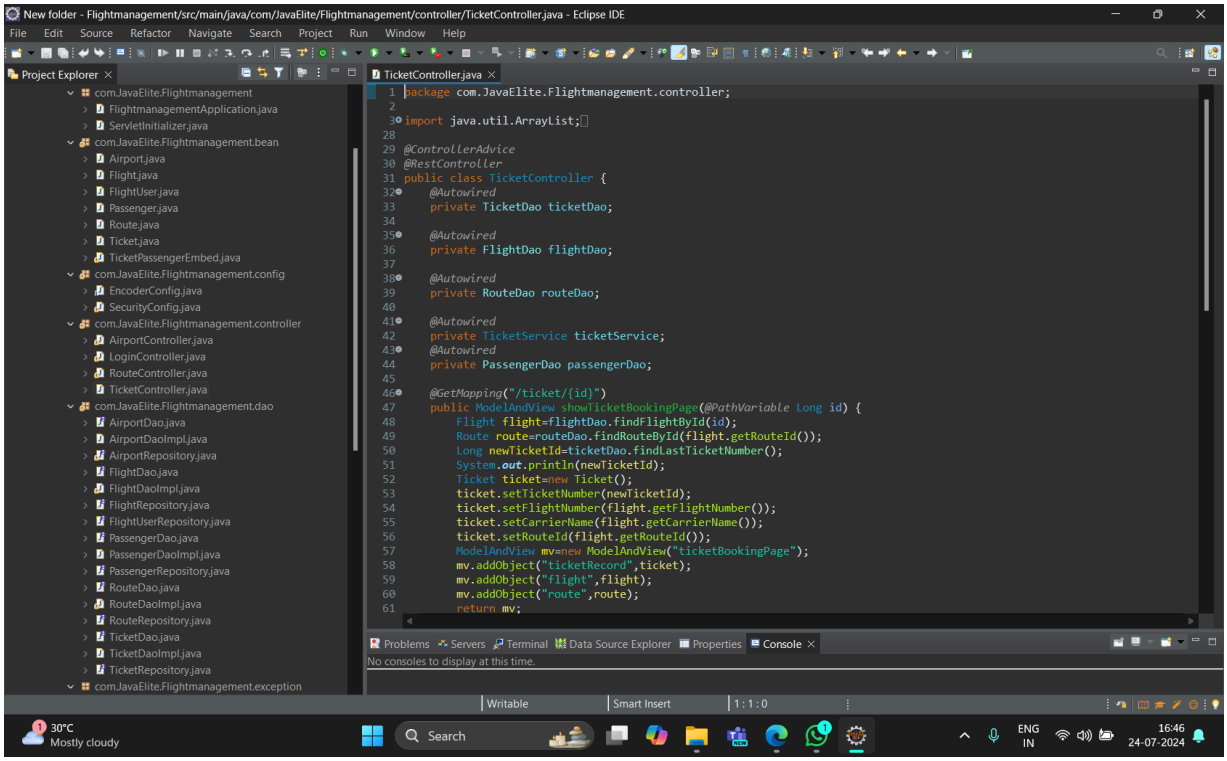


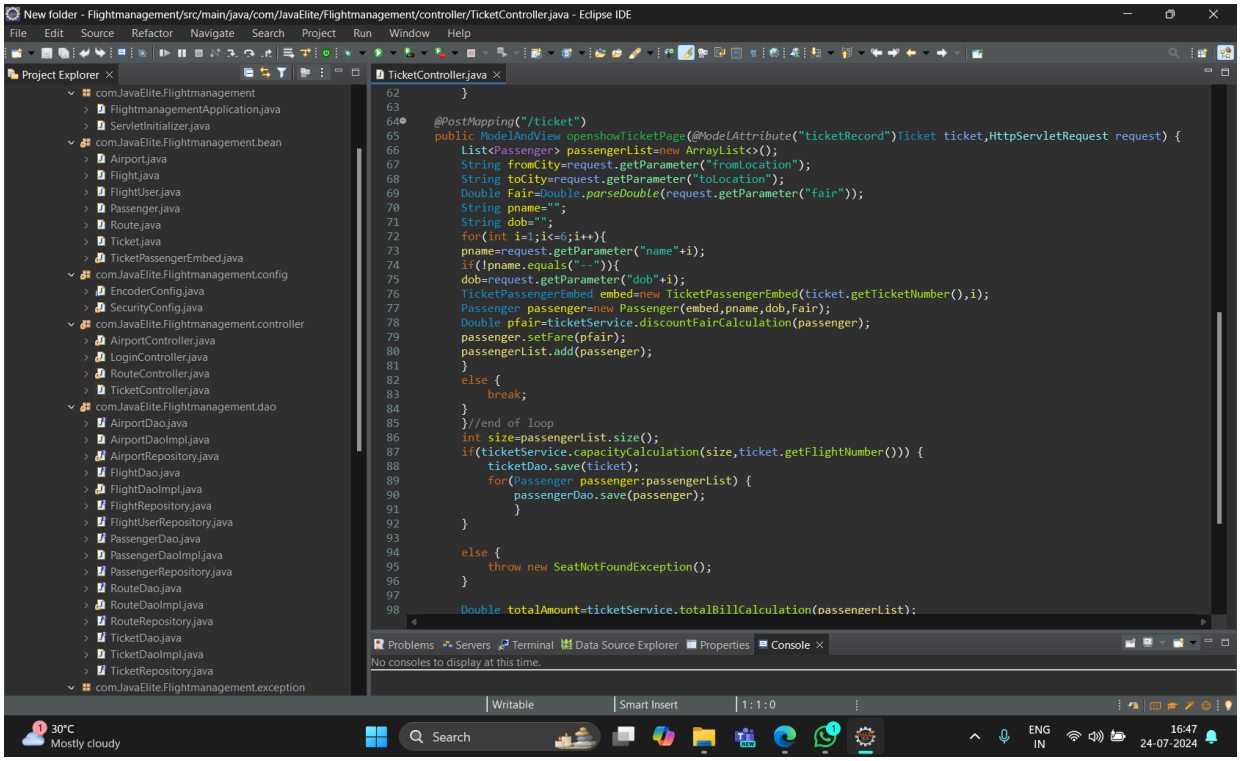


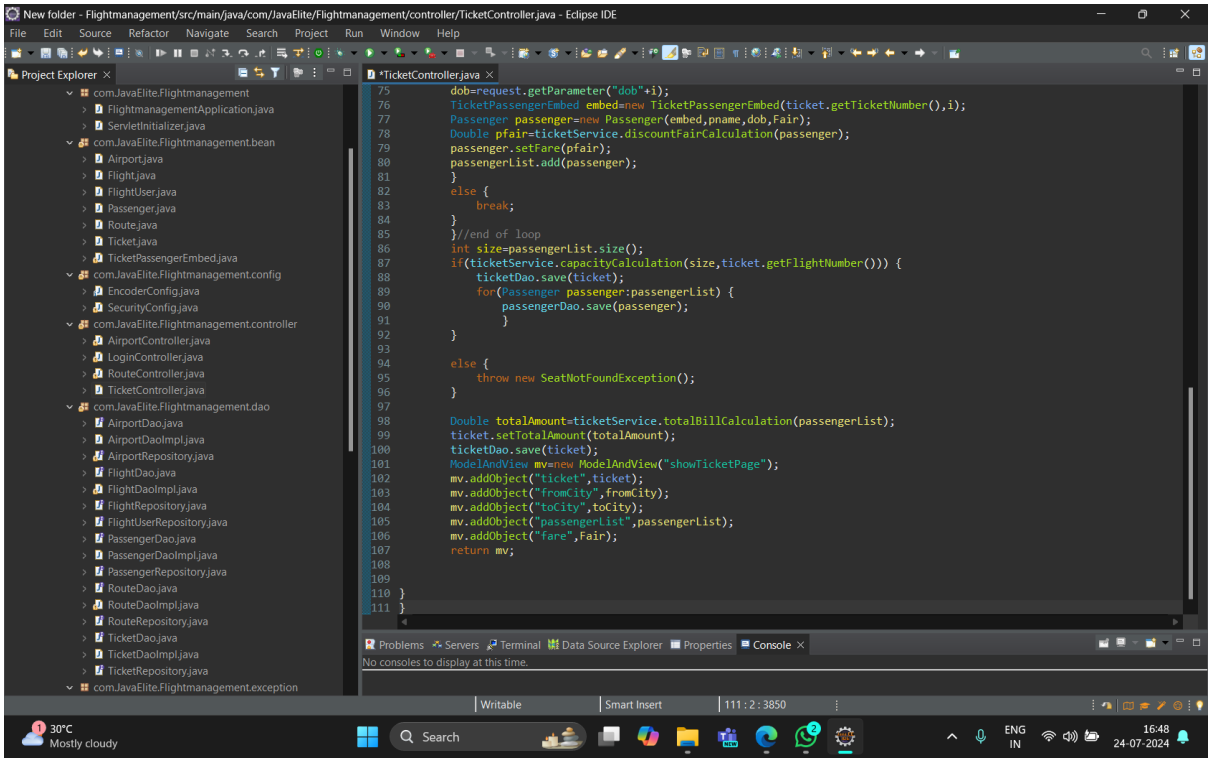

New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/RouteController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
RouteController.java
@PostMapping("/flight")
public ModelAndView saveFlights(@ModelAttribute("FlightRecord")Flight flight1,@RequestParam("dtime") String dtime,@Re
Flight flight2=flightService.createReturnFlight(flight1,dtime,atime);
flightDao.save(flight1);
flightDao.save(flight2);
return new ModelAndView("redirect:/index");
}
@GetMapping("/flights")
public ModelAndView showFlightRecordPage() {
List<Flight> flightList=flightDao.findAllFlights();
ModelAndView mv=new ModelAndView("FlightRecordPage");
mv.addObject("flight",flightList);
return mv;
}
@GetMapping("/flight-search")
public ModelAndView showRouteSelectPage() {
List<String> airportList=airportDao.findAllAirportLocations();
ModelAndView mv=new ModelAndView("FlightSelectPage");
mv.addObject("abc",airportList);
return mv;
}
@PostMapping("/flight-search")
public ModelAndView showRouteFlightsPage(@RequestParam("fromCity")String fromCity,@RequestParam("toCity")String toCity
String fromAirport=airportDao.findAirportCodeByLocation(fromCity);
String toAirport=airportDao.findAirportCodeByLocation(toCity);
if(fromAirport.equalsIgnoreCase(toAirport))
throw new RouteNotFoundException();
Route route=routeDao.findRouteBySourceAndDestination(fromAirport, toAirport);
List<Flight> flightList=flightDao.findFlightsByRouteId(route.getRouteId());
ModelAndView mv=new ModelAndView("routeFlightShowPage");
mv.addObject("flightList",flightList);
mv.addObject("fromAirport",fromCity);
mv.addObject("toAirport",toCity);
mv.addObject("fair",route.getFare());
Problems Servers Terminal Data Source Explorer Properties Console
No consoles to display at this time.
Writable Smart Insert 1 : 1 : 0
30°C Mostly cloudy
Search
ENG IN
16:45 24-07-2024


New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/RouteController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
RouteController.java
@GetMapping("/flight-search")
public ModelAndView showRouteSelectPage() {
List<String> airportList=airportDao.findAllAirportLocations();
ModelAndView mv=new ModelAndView("FlightSelectPage");
mv.addObject("abc",airportList);
return mv;
}
@PostMapping("/flight-search")
public ModelAndView showRouteFlightsPage(@RequestParam("fromCity")String fromCity,@RequestParam("toCity")String toCity
String fromAirport=airportDao.findAirportCodeByLocation(fromCity);
String toAirport=airportDao.findAirportCodeByLocation(toCity);
if(fromAirport.equalsIgnoreCase(toAirport))
throw new RouteNotFoundException();
Route route=routeDao.findRouteBySourceAndDestination(fromAirport, toAirport);
List<Flight> flightList=flightDao.findFlightsByRouteId(route.getRouteId());
ModelAndView mv=new ModelAndView("routeFlightShowPage");
mv.addObject("flightList",flightList);
mv.addObject("fromAirport",fromCity);
mv.addObject("toAirport",toCity);
mv.addObject("fair",route.getFare());
return mv;
}
@ExceptionHandler(value=RouteNotFoundException.class)
public ModelAndView handlingRouteException(RouteNotFoundException exception) {
String message="From-City & To-City cannot be the same.....";
ModelAndView mv=new ModelAndView("routeErrorPage");
mv.addObject("errorMessage",message);
return mv;
}
}
Problems Servers Terminal Data Source Explorer Properties Console
No consoles to display at this time.
Writable Smart Insert 1 : 1 : 0
30°C Mostly cloudy
Search
ENG IN
16:45 24-07-2024

- **Ticket controller:** The Ticket Controller manages ticket bookings and passenger processing in the Flight Management System. It displays booking forms for flights, handles the submission of ticket details, and processes passenger information. The controller performs fare calculations, checks seat availability, and manages exceptions for unavailable seats.

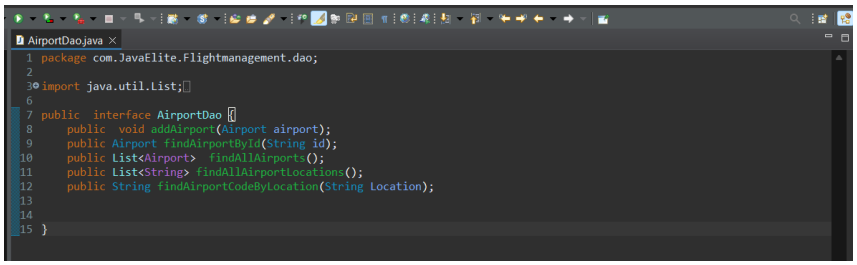




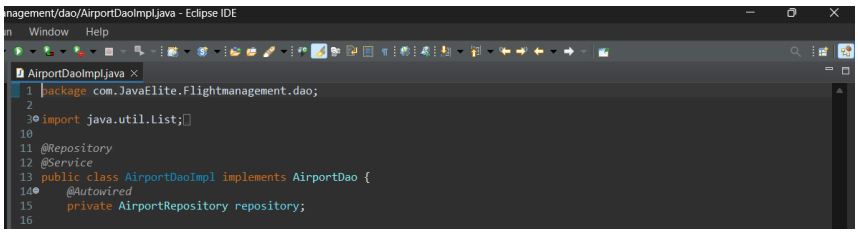


5)Flight Management. Dao:

- **AirportDao.java:** The Airport Dao interface provides methods for managing airport data in the Flight Management System. It includes operations for adding new airports, retrieving airports by ID, and listing all airports. It also offers methods to get airport locations and find airport codes based on locations.



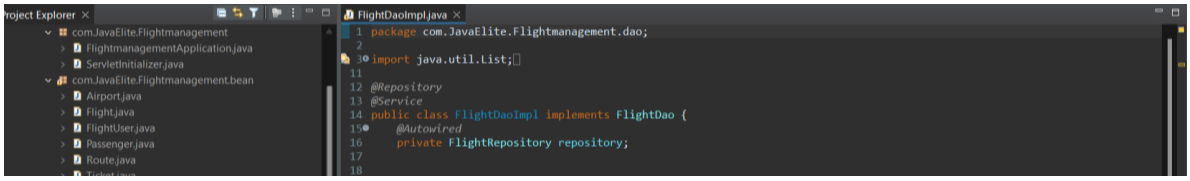
- **AirportDaoImpl.java:** The Airport Dao Impl class implements the Airport Dao interface, providing concrete methods for managing airport data. It uses Airport Repository for data access operations.



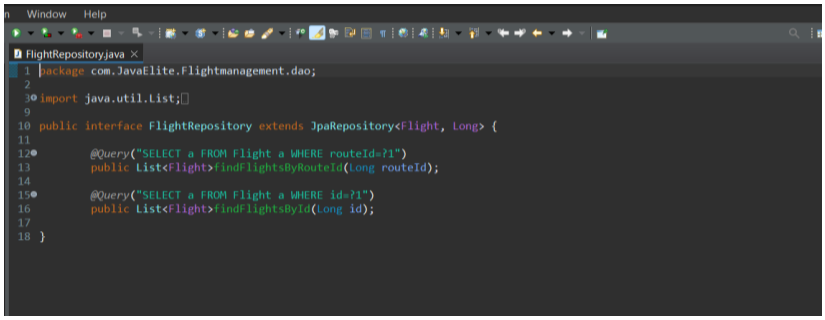
- **AirportRepository.java:** The Airport Repository interface extends JPA Repository to provide basic CRUD operations for the Airport entity. It includes custom query methods to retrieve all airport codes and find an airport code based on its location.

- **FlightDao.java:** The Flight Dao interface defines methods for managing flight data in the Flight Management System. It includes operations to save a flight, retrieve all flights, find flights by a specific route ID, and locate a flight by its unique ID. This interface provides a blueprint for interacting with flight data, enabling various data access operations.

- **FlightDaoImpl.java:** The Flight Dao Impl class implements the Flight Dao interface, offering methods for managing flight data. It uses Flight Repository to perform CRUD operations such as saving flights, retrieving all flights, and finding flights by route ID or by a specific ID. This class facilitates interaction with the database for flight-related operations.

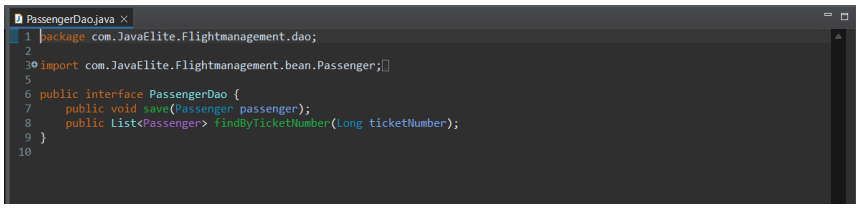


- **FlightRepository.java:** The Flight Repository interface extends Jpa Repository to manage Flight entities. It includes: Retrieves flights based on a given route ID, Retrieves flights with a specific ID.



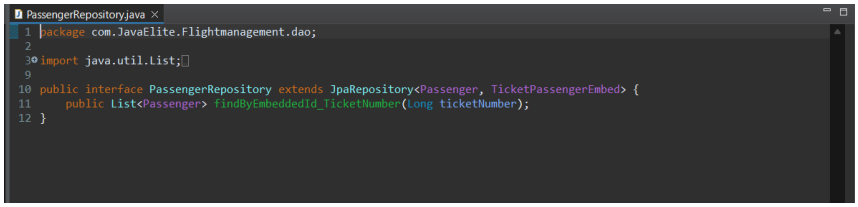
- **FlightUserRepository.java:** The Flight User Repository interface extends Jpa Repository for Flight User entities. It includes a method to find a user by their username, returning an Optional to handle possible absence of the user. This enables efficient user lookups based on the username.

- **PassengerDao.java:** The Passenger Dao interface offers methods for saving Passenger entities and retrieving passengers associated with a specific ticket number. It facilitates efficient management of passenger data.



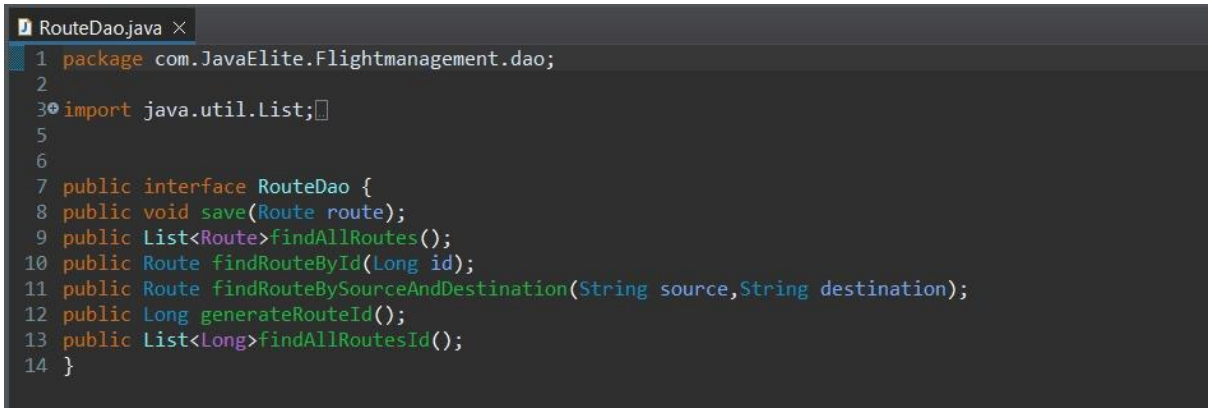
- **PassengeDaoImpl.java:** The Passenger Dao interface offers methods for saving Passenger entities and retrieving passengers associated with a specific ticket number. It facilitates efficient management of passenger data.

- **PassengerRepository.java:** The Passenger Repository interface extends Jpa Repository for Passenger entities with a composite key. It includes a method to find passengers by ticket number. This setup supports complex queries and data persistence operations.



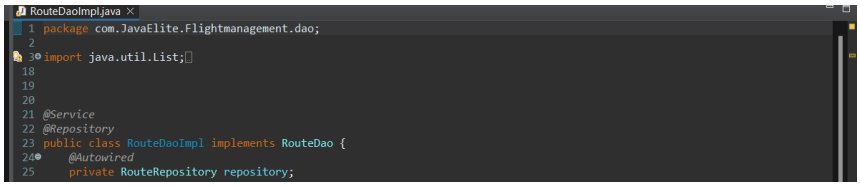
```java
package com.JavaElite.Flightmanagement.dao;

import java.util.List;

public interface PassengerRepository extends JpaRepository<Passenger, TicketPassengerEmbed> {
    public List<Passenger> findByEmbeddedId_TicketNumber(Long ticketNumber);
}
```

- **RouteDao.java:** The Route Dao interface manages Route entities, offering methods to save routes, retrieve them by ID or source/destination, and list all routes. It also includes functionality to generate new route IDs and list all existing ones.



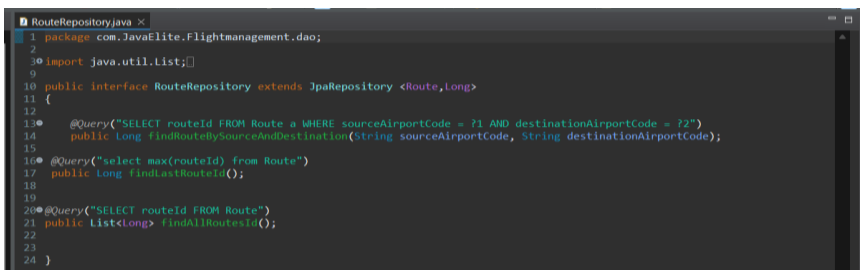
```java
package com.JavaElite.Flightmanagement.dao;

import java.util.List;


public interface RouteDao {
public void save(Route route);
public List<Route>findAllRoutes();
public Route findRouteById(Long id);
public Route findRouteBySourceAndDestination(String source,String destination);
public Long generateRouteId();
public List<Long>findAllRoutesId();
}
```

- **RouteDaoImpl.java:** The RouteDaoImpl class implements RouteDao, handling CRUD operations for Route entities. It uses RouteRepository to save routes, find routes by ID or by source and destination, and list all routes. It includes methods to generate new route IDs and retrieve all existing route IDs.



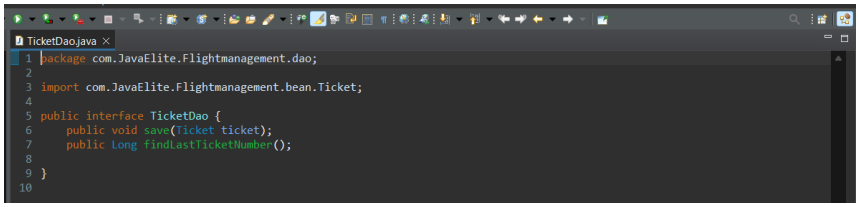
```java
package com.JavaElite.Flightmanagement.dao;

import java.util.List;



@Service
@Repository
public class RouteDaoImpl implements RouteDao {
    @Autowired
    private RouteRepository repository;
```

- **RouteRepository.java:** The Route Repository interface manages Route entities with custom queries for:
  Finding a route ID by source and destination airport codes.
  Retrieving the highest existing route ID.
  Listing all route IDs.



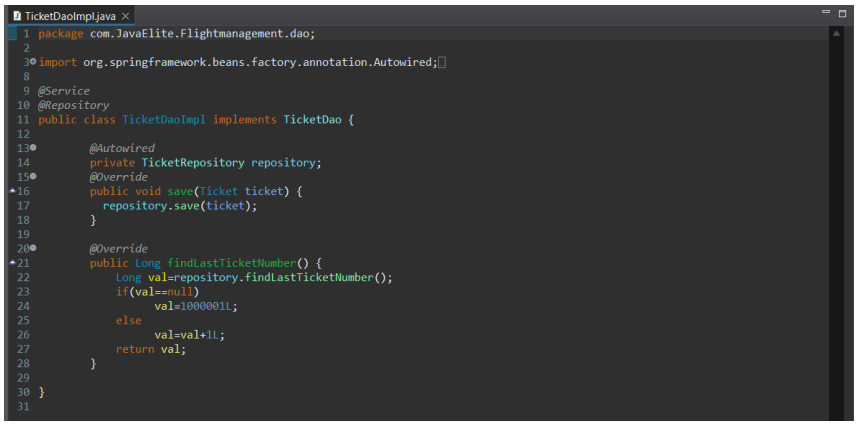
```java
package com.JavaElite.Flightmanagement.dao;

import java.util.List;

public interface RouteRepository extends JpaRepository <Route,Long>
{

    @Query("SELECT routeId FROM Route a WHERE sourceAirportCode = ?1 AND destinationAirportCode = ?2")
    public Long findRouteBySourceAndDestination(String sourceAirportCode, String destinationAirportCode);

@Query("select max(routeId) from Route")
 public Long findLastRouteId();


@Query("SELECT routeId FROM Route")
public List<Long> findAllRoutesId();


}
```

- **TicketDao.java:** The Ticket Dao interface provides methods to:
  Save a Ticket entity.
  Find the most recent ticket number for generating new tickets.



```java
package com.JavaElite.Flightmanagement.dao;

import com.JavaElite.Flightmanagement.bean.Ticket;

public interface TicketDao {
    public void save(Ticket ticket);
    public Long findLastTicketNumber();

}
```

- **TicketDaoImpl.java:** The Ticket Dao Impl class implements the Ticket Dao interface with the following functionalities:
  Save: Persists a Ticket entity to the database using TicketRepository.
  Find Last Ticket Number: Retrieves the highest ticket number from the database, increments it, and defaults to 1000001 if no tickets are found. This ensures a unique ticket number for new entries.
  Auto wired: Uses dependency injection to obtain an instance of TicketRepository.Service and Repository Annotations: Marks the class as a Spring service and repository, enabling transaction management and data access.



```java
package com.JavaElite.Flightmanagement.dao;

import org.springframework.beans.factory.annotation.Autowired;

@Service
@Repository
public class TicketDaoImpl implements TicketDao {

        @Autowired
        private TicketRepository repository;
        @Override
        public void save(Ticket ticket) {
          repository.save(ticket);
        }

        @Override
        public Long findLastTicketNumber() {
            Long val=repository.findLastTicketNumber();
            if(val==null)
                val=1000001L;
            else
                val=val+1L;
            return val;
        }

}
```

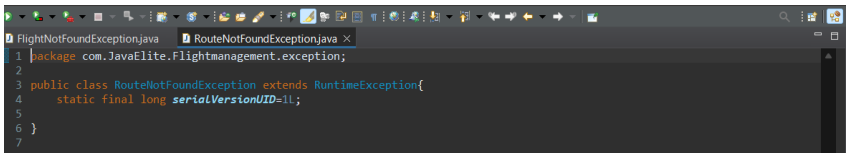
- **TicketRepository.java:** The Ticket Repository interface extends Jpa Repository, providing methods to manage Ticket entities. It includes a custom query, find Last Ticket Number (), to retrieve the highest ticket number for generating new tickets. Additionally, it inherits standard operations for Ticket management.

```java
package com.JavaElite.Flightmanagement.dao;

import org.springframework.data.jpa.repository.JpaRepository;

public interface TicketRepository extends JpaRepository<Ticket,Long> {
    @Query("SELECT max(ticketNumber) FROM Ticket")
    public Long findLastTicketNumber();

}
```

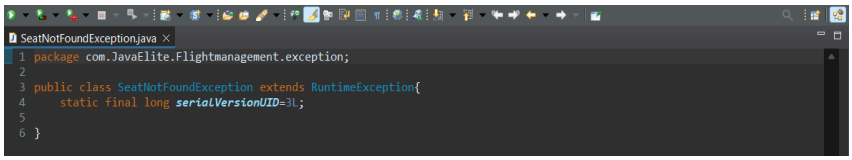
6)JavaElite.Flightmanagement.java

- **FlightNotFoundException.java:** The Flight Not Found Exception class extends Run time Exception and is used to indicate when a flight cannot be found. It includes a serial version UID for serialization purposes, ensuring consistency during deserialization.

- **RouteNotFoundException.java**: Route Not Found Exception is a custom runtime exception used to indicate that a requested route is not found. It extends RuntimeException and includes a serial version UID for serialization compatibility. This exception is used in scenarios where a route does not exist in the system.



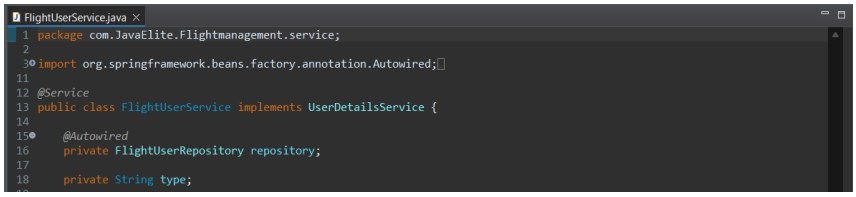
- **SeatNotFoundException.java**: Seat Not Found Exception is a custom runtime exception indicating that a requested seat is not available. It extends Runtime Exception and includes a serial version UID for serialization. This exception is used to handle scenarios where seat availability checks fail.



7)Java Elite. Flight management. service:

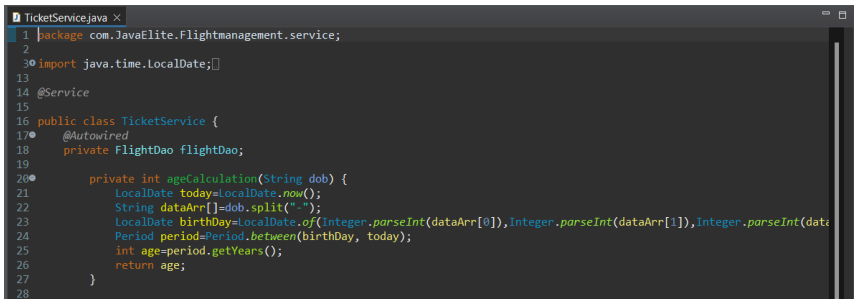
- **FlightService.java:** Flight Service is a Spring service component that provides flight-related operations. It has a dependency on Route Dao for accessing route information. The create Return Flight method generates a new return flight based on the provided flight details, including calculating a new flight ID and fetching route details for the return journey.

- **FlightUserService.java:** Flight Service is a Spring service component that provides flight-related operations. It has a dependency on Route Dao for accessing route information. The create Return Flight method generates a new return flight based on the provided flight details, including calculating a new flight ID and fetching route details for the return journey.

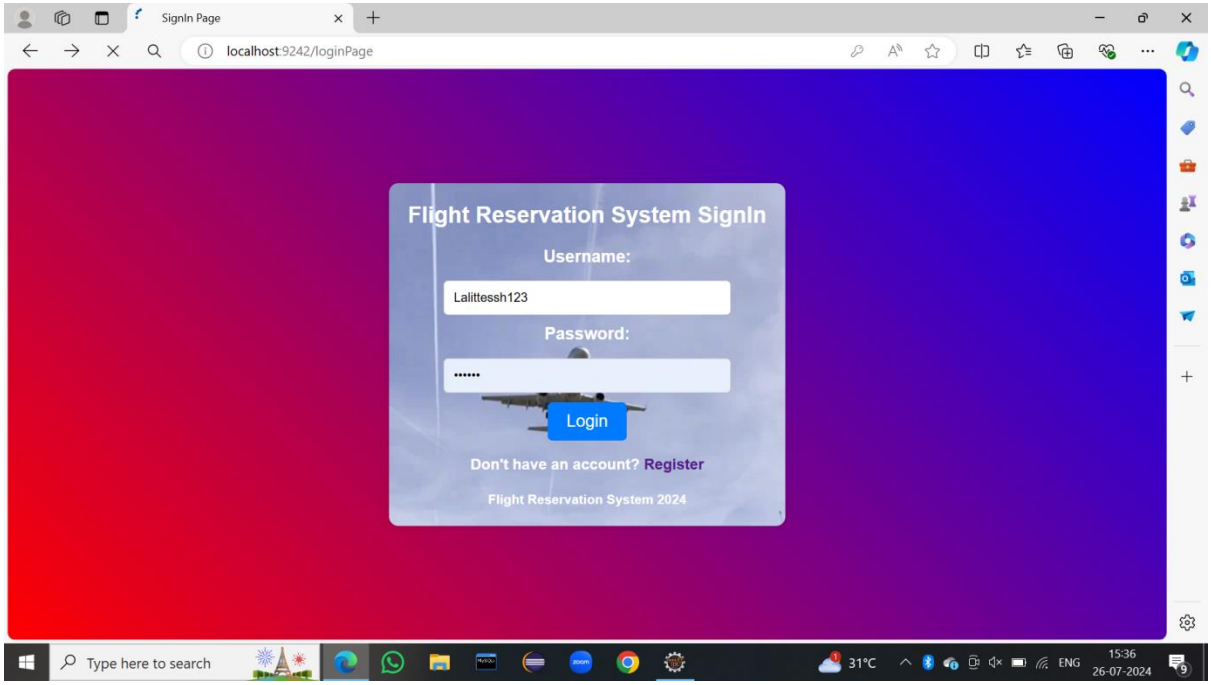


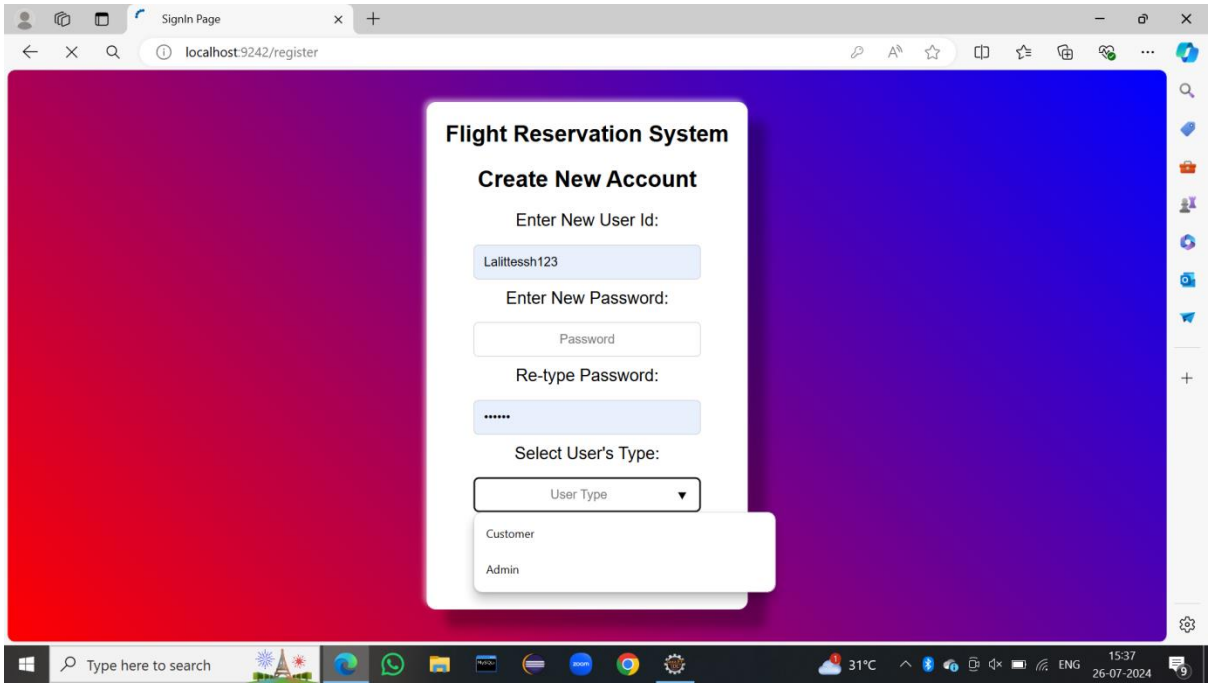
- **RouteService.java:** RouteService provides functionality for managing Route objects. The createReturnRoute method generates a new Route instance representing the return journey, by incrementing the route ID and swapping the source and destination airport codes from the given route.

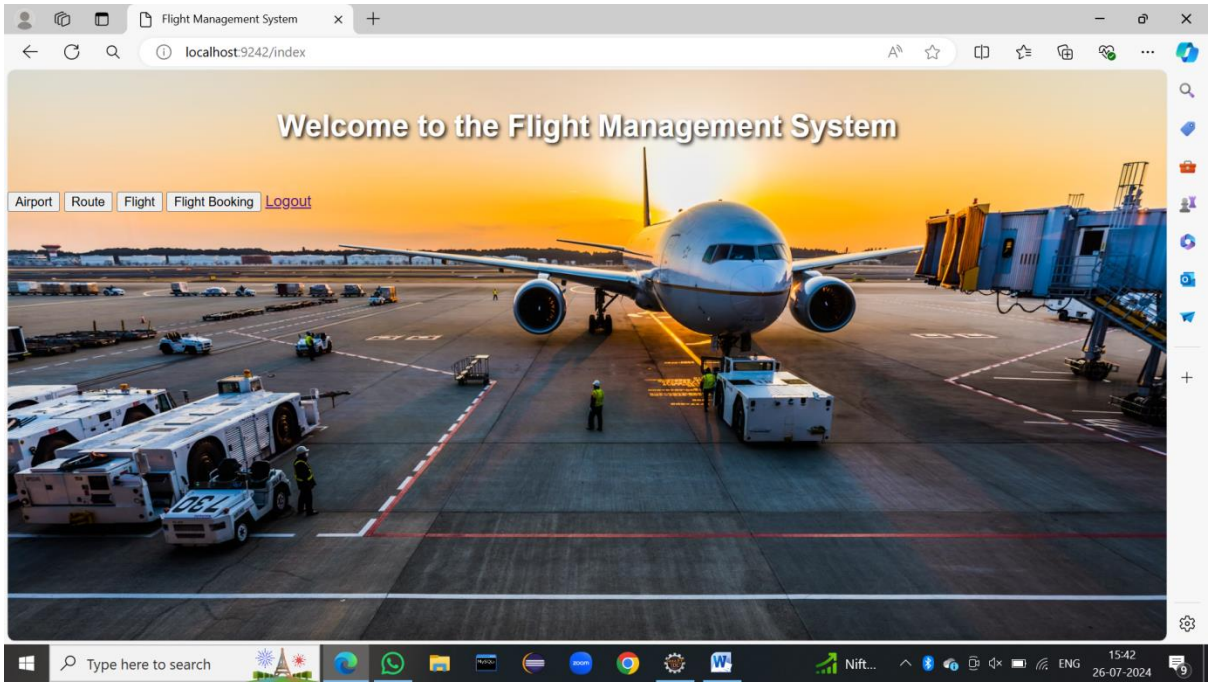
- **TicketService.java:** Ticket Service manages ticket-related operations, including calculating fare discounts based on passenger age, updating flight seat capacities, and summing up total fares for bookings. It ensures proper fare adjustments and seat availability for efficient ticket management.

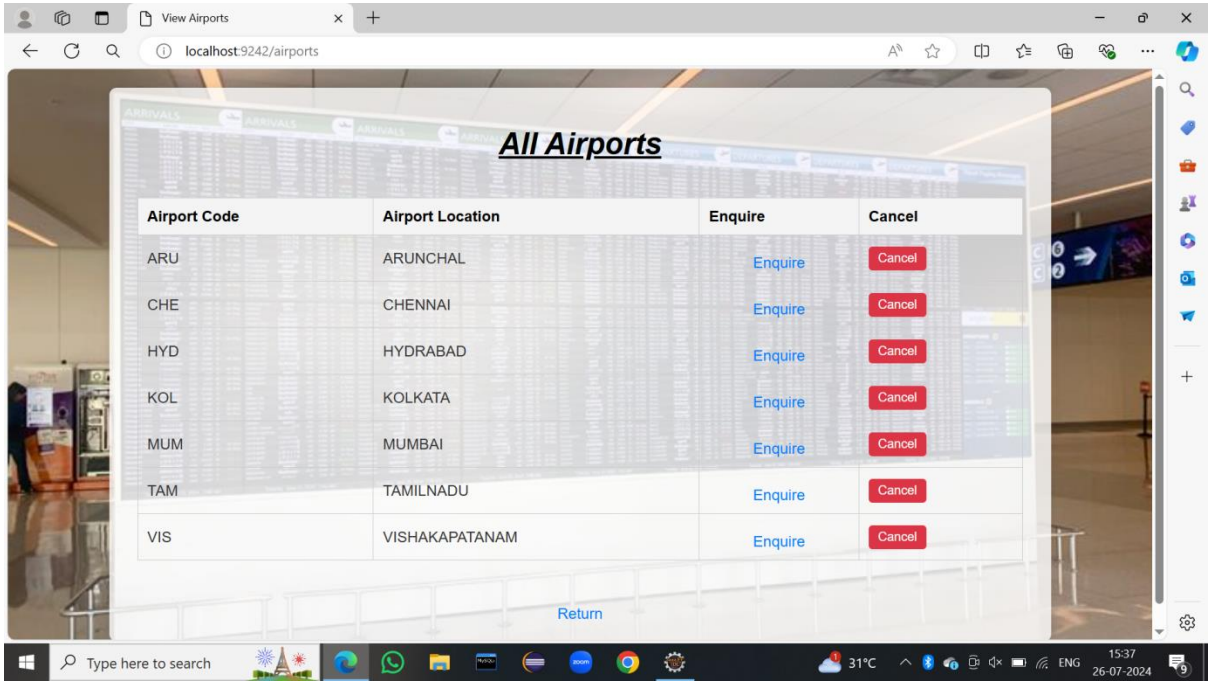


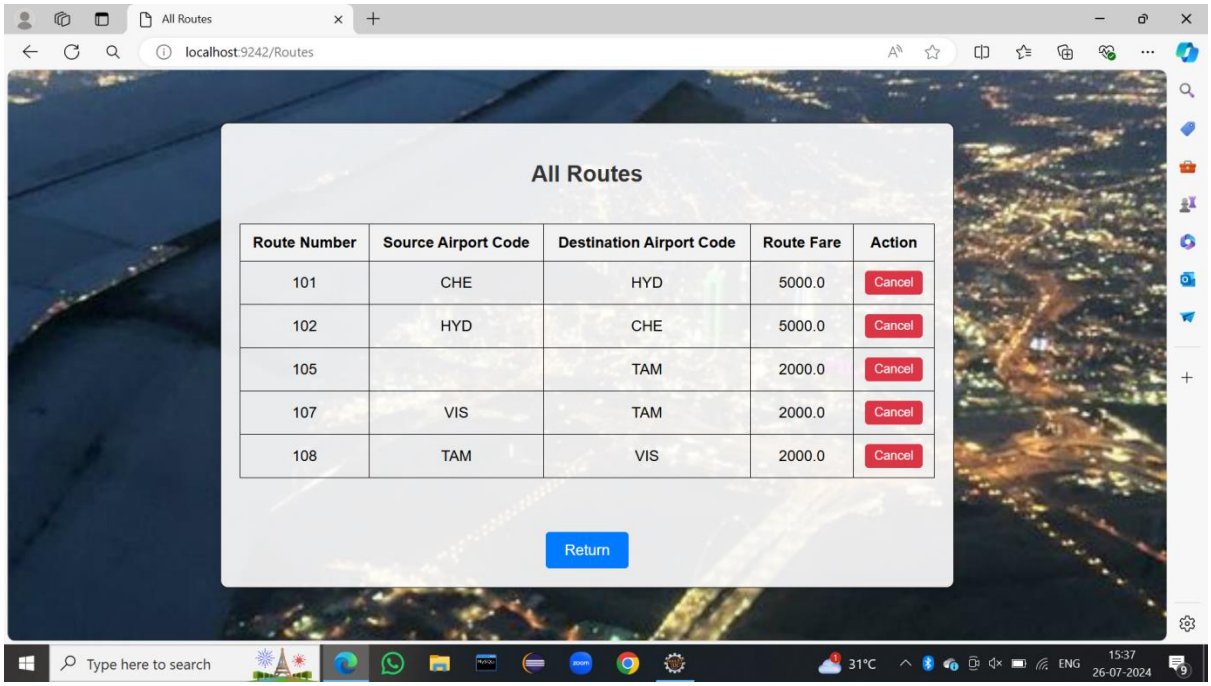
OUTPUTS:

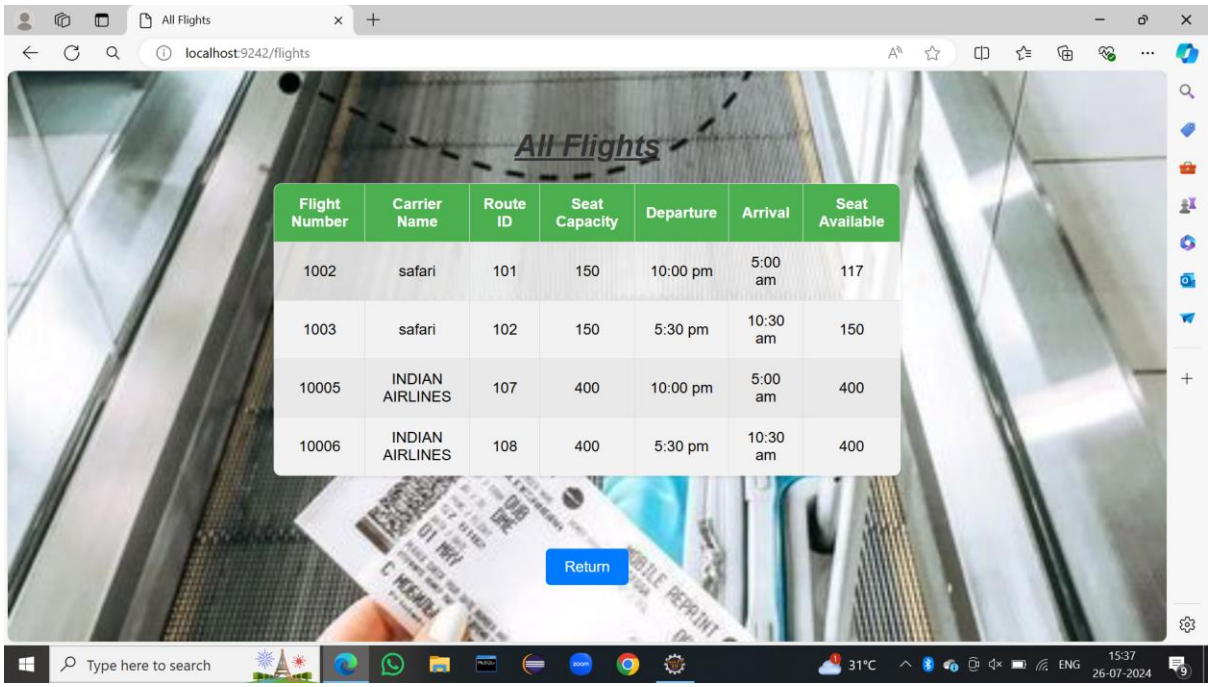

SignIn Page
localhost:9242/loginPage
Flight Reservation System SignIn
Username:
Lalittessh123
Password:
Login
Don't have an account? Register
Flight Reservation System 2024
Type here to search
31°C
ENG
15:36
26-07-2024

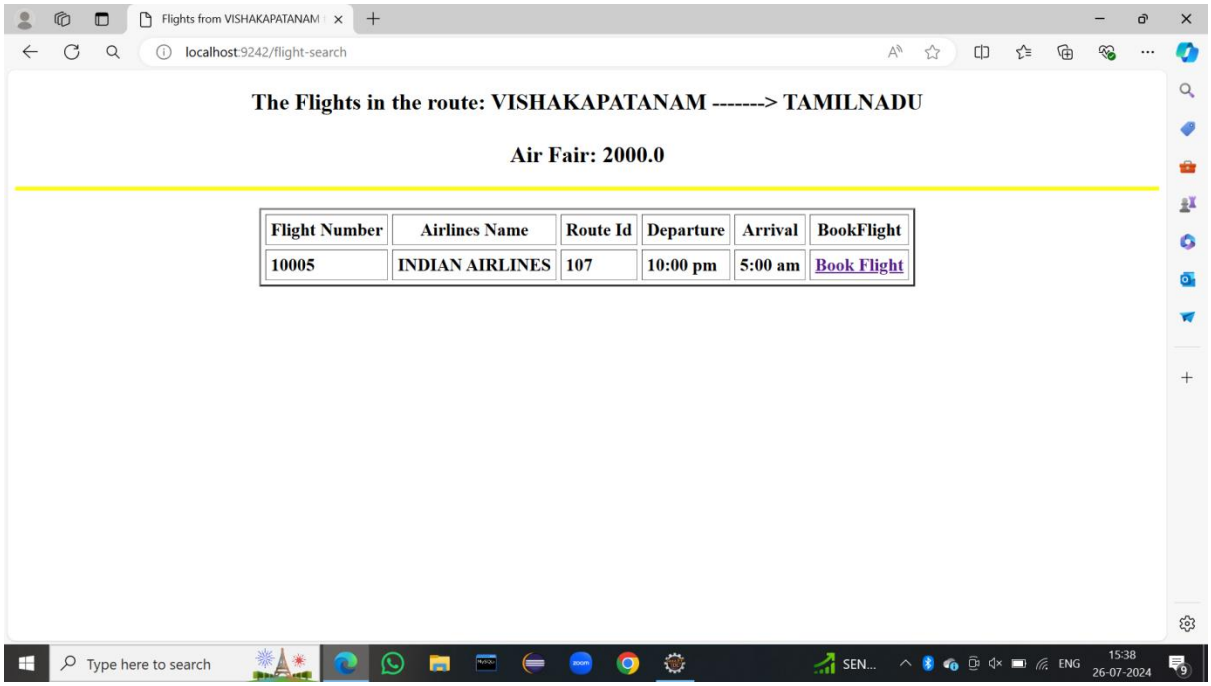

SignIn Page
localhost:9242/register
Flight Reservation System
Create New Account
Enter New User Id:
Lalittessh123
Enter New Password:
Password
Re-type Password:
Select User's Type:
User Type
Customer
Admin
Type here to search
31°C
ENG
15:37
26-07-2024

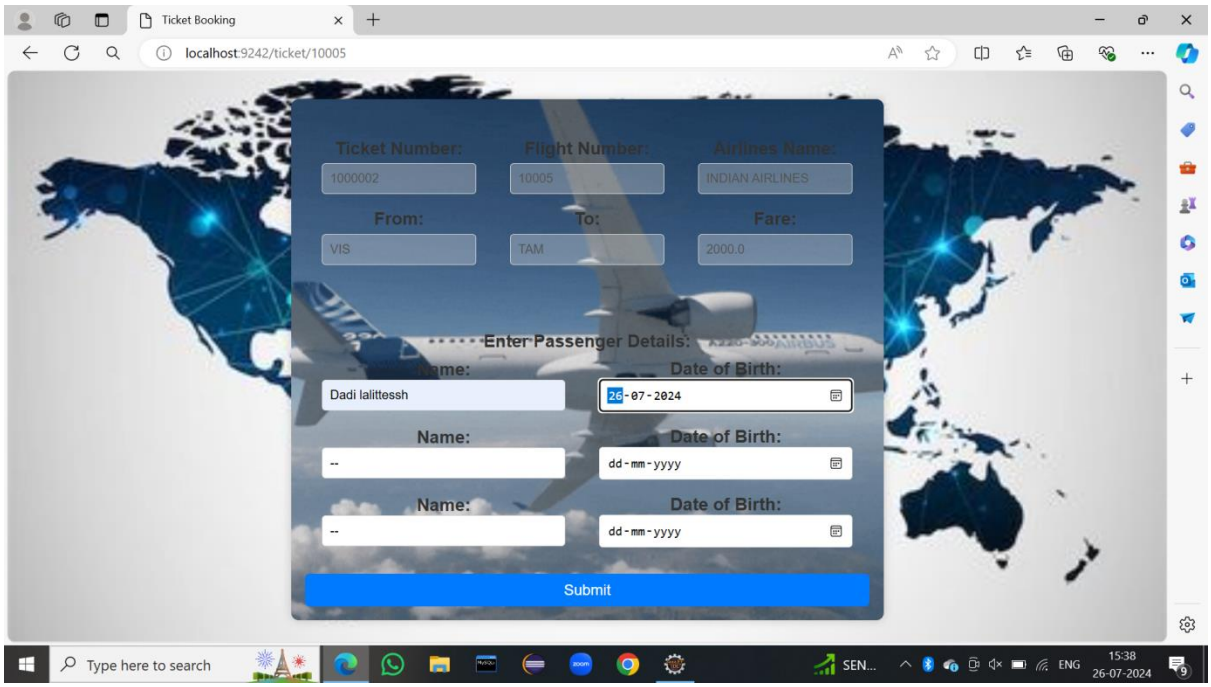

Flight Management System
localhost:9242/index
Welcome to the Flight Management System
Airport
Route
Flight
Flight Booking
Logout
Type here to search
Nift...
ENG
15:42
26-07-2024

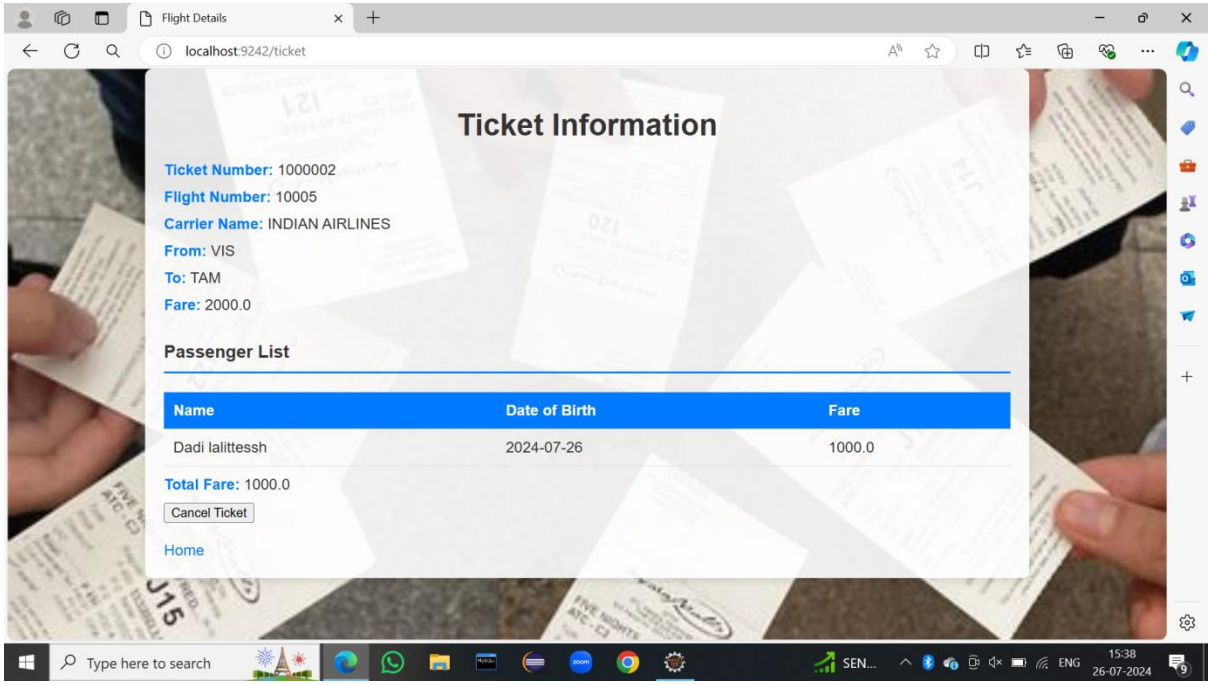

View Airports
localhost:9242/airports
All Airports
Airport Code
Airport Location
Enquire
Cancel
ARU
ARUNCHAL
CHE
CHENNAI
HYD
HYDRABAD
KOL
KOLKATA
MUM
MUMBAI
TAM
TAMILNADU
VIS
VISHAKAPATANAM
Enquire
Cancel
Return
Type here to search
31°C
ENG
15:37
26-07-2024


All Routes
localhost:9242/Routes
All Routes
Route Number
Source Airport Code
Destination Airport Code
Route Fare
Action
101
CHE
HYD
5000.0
Cancel
102
HYD
CHE
5000.0
Cancel
105
TAM
2000.0
Cancel
107
VIS
TAM
2000.0
Cancel
108
TAM
VIS
2000.0
Cancel
Return
Type here to search
31°C
ENG
15:37
26-07-2024


All Flights
localhost:9242/flights
All Flights
Flight Number
Carrier Name
Route ID
Seat Capacity
Departure
Arrival
Seat Available
1002
safari
101
150
10:00 pm
5:00 am
117
1003
safari
102
150
5:30 pm
10:30 am
150
10005
INDIAN AIRLINES
107
400
10:00 pm
5:00 am
400
10006
INDIAN AIRLINES
108
400
5:30 pm
10:30 am
400
Return
Type here to search
31°C
ENG
15:37
26-07-2024


Flights from VISHAKAPATANAM
localhost:9242/flight-search
The Flights in the route: VISHAKAPATANAM -------> TAMILNADU
Air Fair: 2000.0
Flight Number | Airlines Name | Route Id | Departure | Arrival | BookFlight
10005 | INDIAN AIRLINES | 107 | 10:00 pm | 5:00 am | Book Flight
Type here to search
ENG
15:38
26-07-2024


Ticket Booking
localhost:9242/ticket/10005
Ticket Number:
1000002
Flight Number:
10005
Airlines Name:
INDIAN AIRLINES
From:
VIS
To:
TAM
Fare:
2000.0
Enter Passenger Details:
Name:
Dadi lalittessh
Date of Birth:
26-07-2024
Name:
--
Date of Birth:
dd-mm-yyyy
Name:
--
Date of Birth:
dd-mm-yyyy
Submit
Type here to search
ENG
15:38
26-07-2024


Flight Details
localhost:9242/ticket
Ticket Information
Ticket Number: 1000002
Flight Number: 10005
Carrier Name: INDIAN AIRLINES
From: VIS
To: TAM
Fare: 2000.0
Passenger List
Name
Date of Birth
Fare
Dadi lalittessh
2024-07-26
1000.0
Total Fare: 1000.0
Cancel Ticket
Home
Type here to search
ENG
15:38
26-07-2024