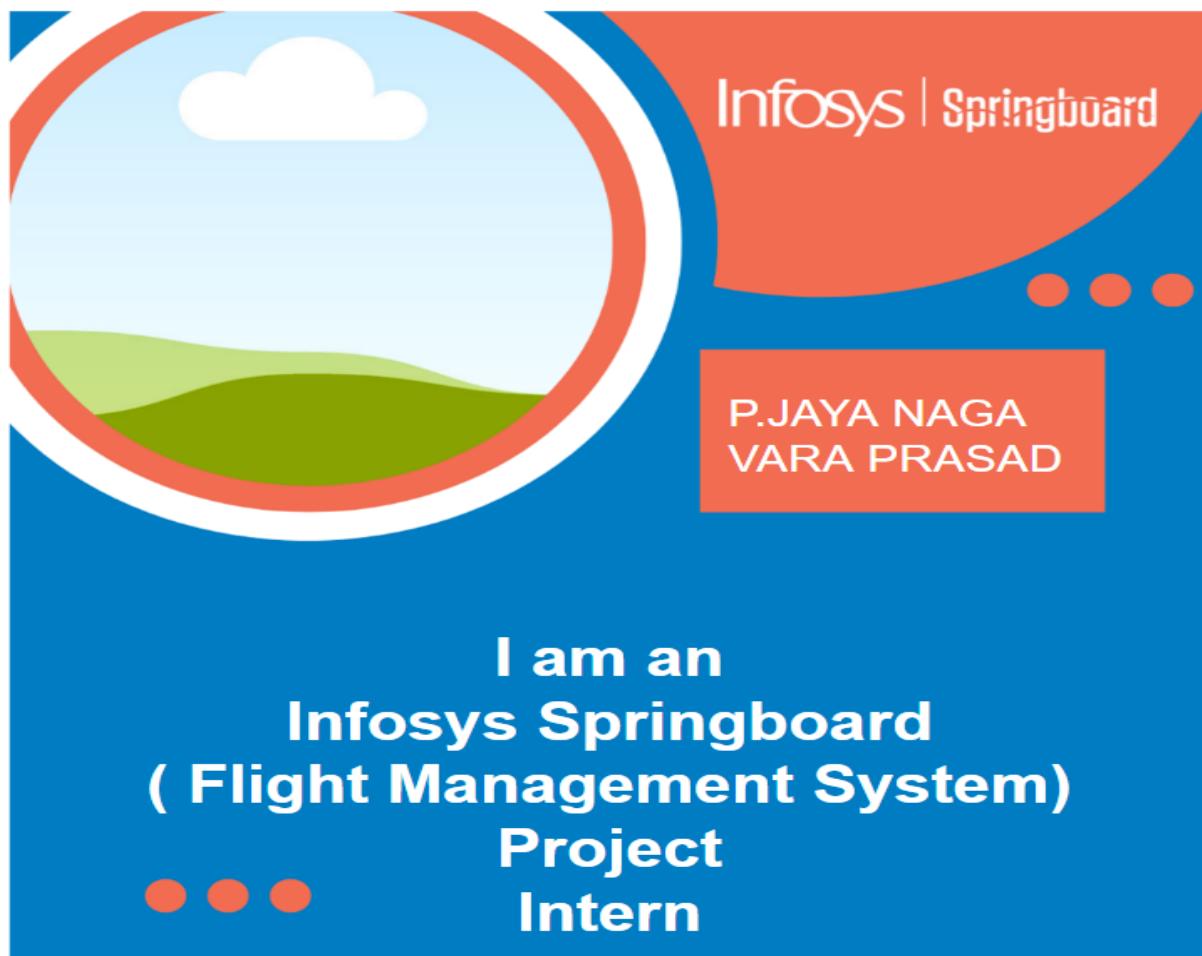


INFOSYS
SPRINGBOARD
COMPUTER SCIENCE AND SYSTEMS ENGINEERING



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.

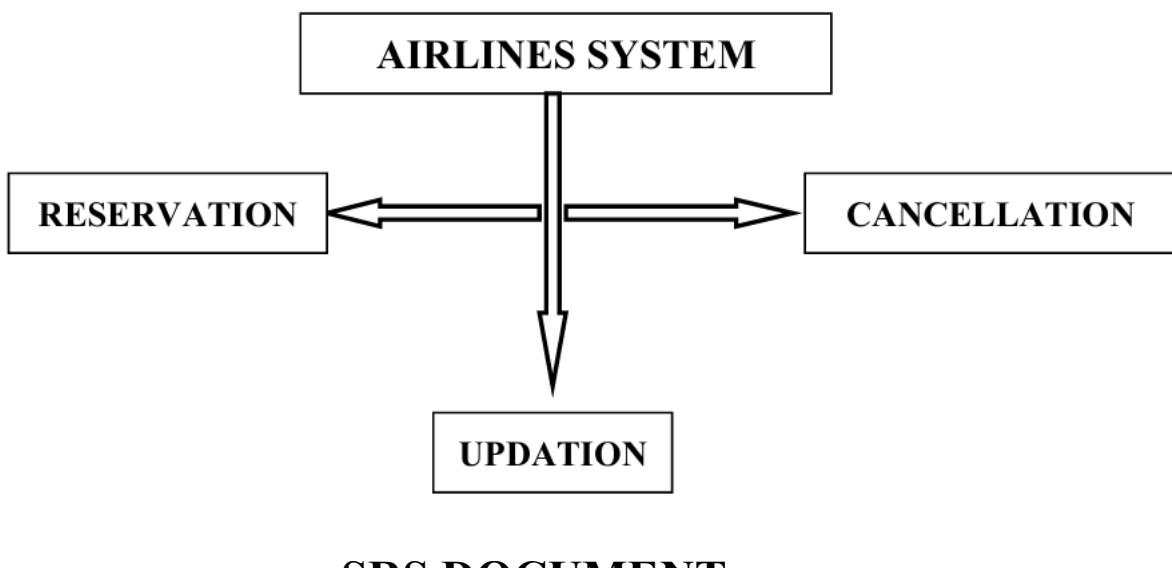
This project outlines the development of a sophisticated, Java-based FMS designed to revolutionize flight booking and management. This system offers a significant advantage over older methods by providing a user-centric interface, real-time data access, and comprehensive tools for both travelers and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

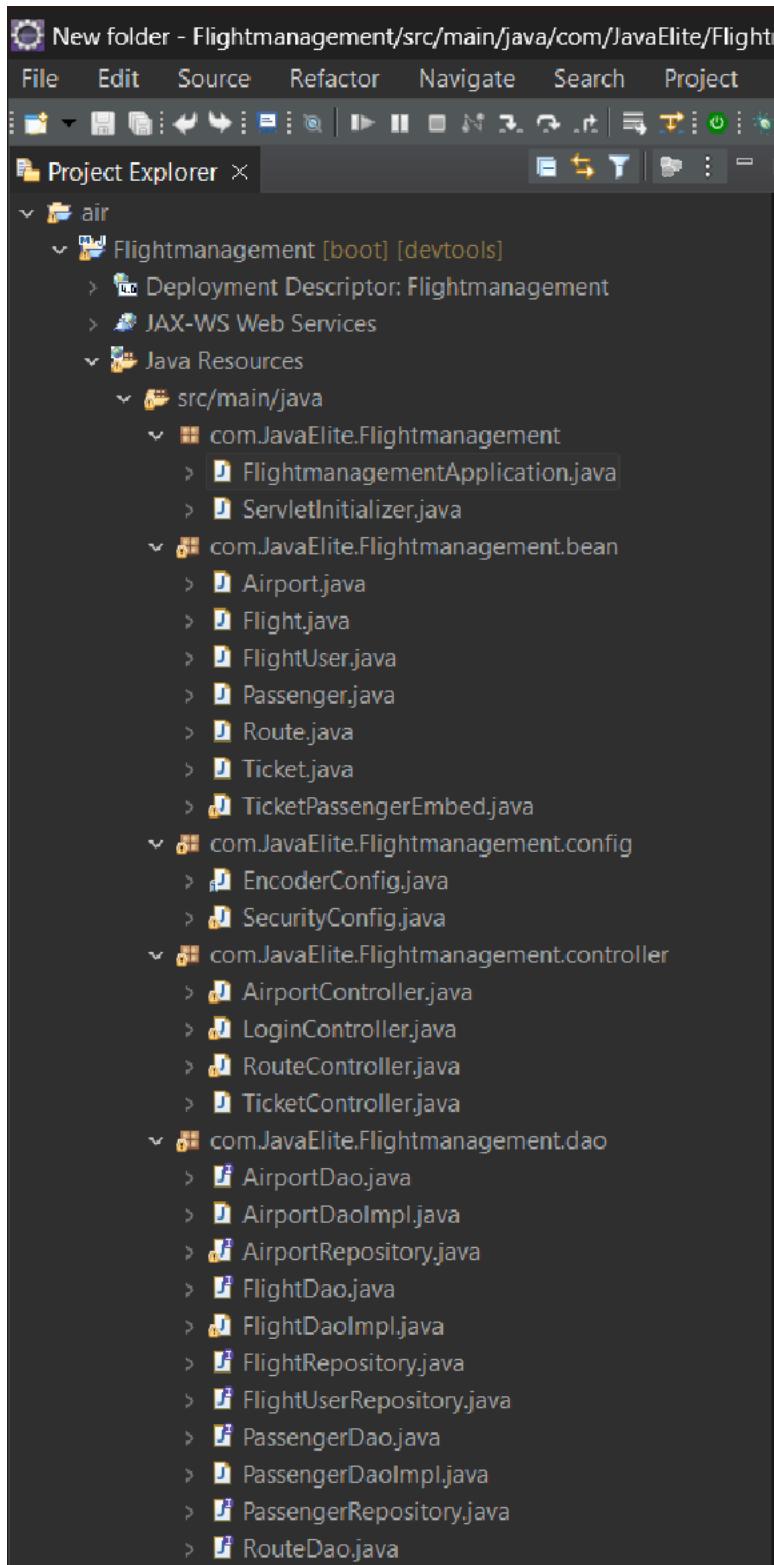
Technologies Used:

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.

```

1 package com.JavaElite.Flightmanagement;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class FlightmanagementApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(FlightmanagementApplication.class, args);
10    }
11 }
12
13 }
14

```

- **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.

```

1 package com.JavaElite.Flightmanagement;
2
3 import org.springframework.boot.builder.SpringApplicationBuilder;
4
5 public class ServletInitializer extends SpringBootServletInitializer {
6
7     @Override
8     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
9         return application.sources(FlightmanagementApplication.class);
10    }
11 }
12
13 }
14

```

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.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Airport {
7     public Airport() {
8         super();
9     }
10    public Airport(String airportCode, String airportlocation) {
11        super();
12        this.airportCode = airportCode;
13        this.airportlocation = airportlocation;
14    }
15    @Id
16    private String airportCode;
17    private String airportlocation;
18}

```

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

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Flight {
7     @Id
8     private Long flightNumber;
9     private String carrierName;
10    private Long routeld;
11    private Integer seatCapacity;
12    private String departure ;
13    private String arrival;
14    private Integer seatBooked;
15    public Flight() {
16        super();
17    }
18}

```

- **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.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import java.util.ArrayList;
4
5 @Entity
6 public class FlightUser extends User {
7     private static final long serialVersionUID = 1L;
8
9     @Id
10    private String username;
11    private String password;
12    private String type;
13    public FlightUser() {
14        super("abc", "pqr", new ArrayList<>());
15    }
16}

```

- **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.

The screenshot shows the Eclipse IDE interface with the 'Passenger.java' file open in the editor. The code defines a Java entity named 'Passenger' with fields for embedded ID, passenger name, DOB, and fare.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import javax.persistence.EmbeddedId;
4
5
6 @Entity
7 public class Passenger {
8
9     @EmbeddedId
10    private TicketPassengerEmbed embeddedId;
11    private String passengerName;
12    private String passengerDOB;
13    private Double fare;
14
15    public Passenger() {
16        super();
17    }
18

```

- **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.

The screenshot shows the Eclipse IDE interface with the 'Route.java' file open in the editor. The code defines a Java entity named 'Route' with fields for route ID, source and destination airport codes, and fare.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import javax.persistence.Entity;
4
5
6 @Entity
7 public class Route {
8     @Override
9     public String toString() {
10         return "Route [routeId=" + routeId + ", sourceAirportCode=" + sourceAirportCode + ", destinationAirportCode=" +
11                + destinationAirportCode + ", fare=" + fare + "]";
12     }
13
14     @Id
15     private Long routeId;
16     private String sourceAirportCode;
17     private String destinationAirportCode;
18     private Double fare;
19
20     public Route(Long routeId, String sourceAirportCode, String destinationAirportCode, Double fare) {
21         super();
22         this.routeId = routeId;
23         this.sourceAirportCode = sourceAirportCode;
24         this.destinationAirportCode = destinationAirportCode;
25         this.fare = fare;
26     }
27

```

- **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.

The screenshot shows the Eclipse IDE interface with the 'Ticket.java' file open in the editor. The code defines a Java entity named 'Ticket' with fields for ticket number, route ID, flight ID, carrier name, and total amount.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import javax.persistence.Entity;
4
5
6 @Entity
7 public class Ticket {
8
9     @Id
10    private Long ticketNumber;
11    private Long routId;
12    private Long flightNumber;
13    private String carrierName;
14    private Double totalAmount;
15
16    public Ticket() {
17        super();
18    }
19
20

```

- **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.

```

1 package com.JavaElite.Flightmanagement.bean;
2
3 import java.io.Serializable;
4
5 @Embeddable
6 public class TicketPassengerEmbed implements Serializable{
7     @NotNull
8     private Long ticketNumber;
9     @NotNull
10    private Integer serialNumber;
11    public TicketPassengerEmbed() {
12        super();
13    }
14    public TicketPassengerEmbed(Long ticketNumber, Integer serialNumber) {
15        super();
16        this.ticketNumber = ticketNumber;
17        this.serialNumber = serialNumber;
18    }
19}

```

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.

```

1 package com.JavaElite.Flightmanagement.config;
2
3 import org.springframework.context.annotation.Bean;
4
5 @Configuration
6 public class EncoderConfig {
7     @Bean
8     public PasswordEncoder passwordEncoder() {
9         return new BCryptPasswordEncoder();
10    }
11}

```

- **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.

The screenshot shows the Eclipse IDE interface. The Project Explorer view on the left lists several Java packages and their files, including Flightmanagement, com.JavaElite.Flightmanagement, com.JavaElite.Flightmanagement.config, com.JavaElite.Flightmanagement.controller, and com.JavaElite.Flightmanagement.dao. The code editor window on the right displays the `SecurityConfig.java` file, which contains Spring Security configuration code. The status bar at the bottom shows the date and time as 24-07-2024.

```
1 package com.JavaElite.Flightmanagement.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @SuppressWarnings("deprecation")
6 @Configuration
7 @EnableWebSecurity
8 public class SecurityConfig extends WebSecurityConfigurerAdapter {
9
10     @Autowired
11     private FlightUserService service;
12
13     @Autowired
14     private EncoderConfig config;
15
16     @Override
17     protected void configure(AuthenticationManagerBuilder authority) throws Exception {
18         authority.userDetailsService(service).passwordEncoder(config.passwordEncoder());
19     }
20
21     @Override
22     public void configure(HttpSecurity http) throws Exception{
23         http.authorizeRequests().antMatchers("/register").permitAll().anyRequest().authenticated().and().formLogin().loginPage(
24             "/login").failureUrl("/loginerror").loginProcessingUrl("/login");
25         .permitAll().and().logout().logoutSuccessUrl("/");
26     }
27
28     @Override
29     public void configure(WebSecurity web) throws Exception {
30         web.ignoring().antMatchers("/css/**");
31     }
32
33     @Override
34     protected void configure(HttpSecurity http) throws Exception {
35         http.csrf().disable();
36     }
37
38 }
39
40
41
42
43
44
45
46
47 }
```

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 X
src/main/java
  com.JavaElite.Flightmanagement
    FlightmanagementApplication.java
    ServletInitializer.java
  com.JavaElite.Flightmanagement.bean
    Airport.java
    Flight.java
    FlightUser.java
    Passenger.java
    Route.java
    Ticket.java
    TicketPassengerEmbed.java
  com.JavaElite.Flightmanagement.config
    EncoderConfig.java
    SecurityConfig.java
  com.JavaElite.Flightmanagement.controller
    AirportController.java
    LoginController.java
    RouteController.java
    TicketController.java
  com.JavaElite.Flightmanagement.dao
    AirportDao.java
    AirportDaoImpl.java
    AirportRepository.java
    FlightDao.java
    FlightDaoImpl.java
    FlightRepository.java
    FlightUserRepository.java
    PassengerDao.java
    PassengerDaoImpl.java
    PassengerRepository.java
    RouteDao.java
AirportController.java X
1 package com.JavaElite.Flightmanagement.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 public class AirportController {
7
8     @Autowired
9     private AirportDao airportDao;
10
11
12     @GetMapping("/airport")
13     public ModelAndView showAirportEntryPage() {
14         Airport airport = new Airport();
15         ModelAndView mv = new ModelAndView("airportEntryPage");
16         mv.addObject("airportRecord", airport);
17         return mv;
18     }
19
20     @PostMapping("/airport")
21     public ModelAndView saveAirport(@ModelAttribute("airportRecord") Airport airport) {
22         String str = airport.getAirportCode().toUpperCase();
23         airport.setAirportCode(str);
24         String stg=airport.getAirportLocation().toUpperCase();
25         airport.setAirportLocation(stg);
26         airportDao.addAirport(airport);
27         return new ModelAndView("redirect:/index");
28     }
29
30     @GetMapping("/airport/{id}")
31     public ModelAndView showAirportShowPage(@PathVariable ("id") String id) {
32         Airport airport = airportDao.findAirportById(id);
33         ModelAndView mv = new ModelAndView("airportShowPage");
34         mv.addObject("airport", airport);
35         return mv;
36     }
37
38     @GetMapping("/airports")
39     public ModelAndView showAirportRecordPage() {
40         List<Airport> airportList=airportDao.findAllAirports();
41         ModelAndView mv = new ModelAndView("airportRecordPage");
42         mv.addObject("abc",airportList);
43         return mv;
44     }
45
46     @GetMapping("/airports")
47     public ModelAndView showAirportRecordPage() {
48         List<Airport> airportList=airportDao.findAllAirports();
49         ModelAndView mv = new ModelAndView("airportRecordPage");
50         mv.addObject("abc",airportList);
51         return mv;
52     }
53
54     @GetMapping("/airports")
55     public ModelAndView showAirportRecordPage() {
56         List<Airport> airportList=airportDao.findAllAirports();
57         ModelAndView mv = new ModelAndView("airportRecordPage");
58         mv.addObject("abc",airportList);
59         return mv;
60     }
61
62
63 }
```

Problems Servers Terminal Data Source Explorer Properties Console X

No consoles to display at this time.

Writable Smart Insert 34:88:1245

ENG IN 1641 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 X
src/main/java
  com.JavaElite.Flightmanagement
    FlightmanagementApplication.java
    ServletInitializer.java
  com.JavaElite.Flightmanagement.bean
    Airport.java
    Flight.java
    FlightUser.java
    Passenger.java
    Route.java
    Ticket.java
    TicketPassengerEmbed.java
  com.JavaElite.Flightmanagement.config
    EncoderConfig.java
    SecurityConfig.java
  com.JavaElite.Flightmanagement.controller
    AirportController.java
    LoginController.java
    RouteController.java
    TicketController.java
  com.JavaElite.Flightmanagement.dao
    AirportDao.java
    AirportDaoImpl.java
    AirportRepository.java
    FlightDao.java
    FlightDaoImpl.java
    FlightRepository.java
    FlightUserRepository.java
    PassengerDao.java
    PassengerDaoImpl.java
    PassengerRepository.java
    RouteDao.java
AirportController.java X
1 package com.JavaElite.Flightmanagement.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 public class AirportController {
7
8     @Autowired
9     private AirportDao airportDao;
10
11
12     @GetMapping("/airport")
13     public ModelAndView showAirportEntryPage() {
14         Airport airport = new Airport();
15         ModelAndView mv = new ModelAndView("airportEntryPage");
16         mv.addObject("airportRecord", airport);
17         return mv;
18     }
19
20     @PostMapping("/airport")
21     public ModelAndView saveAirport(@ModelAttribute("airportRecord") Airport airport) {
22         String str = airport.getAirportCode().toUpperCase();
23         airport.setAirportCode(str);
24         String stg=airport.getAirportLocation().toUpperCase();
25         airport.setAirportLocation(stg);
26         airportDao.addAirport(airport);
27         return new ModelAndView("redirect:/index");
28     }
29
30     @GetMapping("/airport/{id}")
31     public ModelAndView showAirportShowPage(@PathVariable ("id") String id) {
32         Airport airport = airportDao.findAirportById(id);
33         ModelAndView mv = new ModelAndView("airportShowPage");
34         mv.addObject("airport", airport);
35         return mv;
36     }
37
38     @GetMapping("/airports")
39     public ModelAndView showAirportRecordPage() {
40         List<Airport> airportList=airportDao.findAllAirports();
41         ModelAndView mv = new ModelAndView("airportRecordPage");
42         mv.addObject("abc",airportList);
43         return mv;
44     }
45
46     @GetMapping("/airports")
47     public ModelAndView showAirportRecordPage() {
48         List<Airport> airportList=airportDao.findAllAirports();
49         ModelAndView mv = new ModelAndView("airportRecordPage");
50         mv.addObject("abc",airportList);
51         return mv;
52     }
53
54     @GetMapping("/airports")
55     public ModelAndView showAirportRecordPage() {
56         List<Airport> airportList=airportDao.findAllAirports();
57         ModelAndView mv = new ModelAndView("airportRecordPage");
58         mv.addObject("abc",airportList);
59         return mv;
60     }
61
62
63 }
```

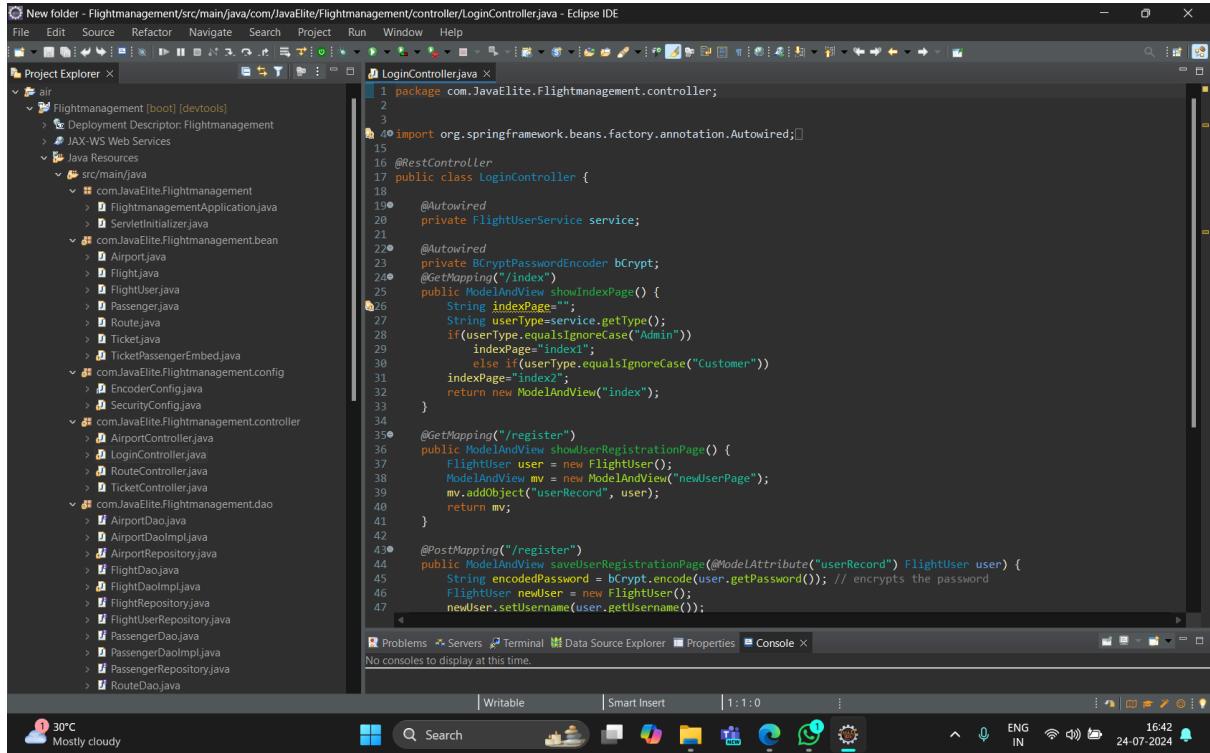
Problems Servers Terminal Data Source Explorer Properties Console X

No consoles to display at this time.

Writable Smart Insert 34:88:1245

ENG IN 1642 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.



The screenshot shows the Eclipse IDE interface with the LoginController.java file open in the editor. The code implements a REST controller for user management. It includes methods for showing the index page, handling user registration, and saving user registration data. The code uses Spring annotations like @RestController, @Autowired, and @GetMapping to map URLs to methods. It also uses BCryptPasswordEncoder for password encryption.

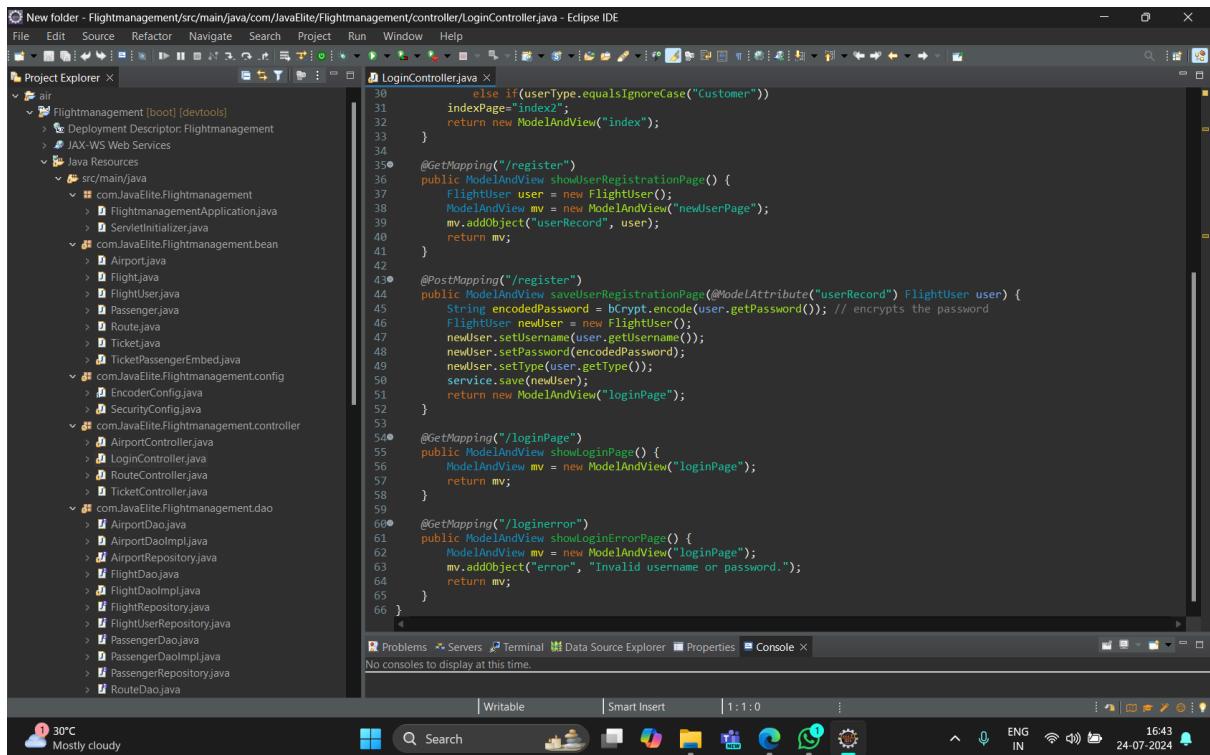
```

package com.JavaElite.Flightmanagement.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelAndView;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.JavaElite.Flightmanagement.model.UserRecord;
import com.JavaElite.Flightmanagement.service.FlightUserService;
import com.JavaElite.Flightmanagement.util.BCryptPasswordEncoder;

```



The screenshot shows the same Eclipse IDE interface with the LoginController.java file open. The code has been modified to include additional logic for handling customer registration. It checks if the user type is 'Customer' and sets the index page accordingly. It also adds a new method for handling login errors.

```

else if(userType.equalsIgnoreCase("Customer"))
    indexPage="index2";
return new ModelAndView("index");
}

@GetMapping("/register")
public ModelAndView showUserRegistrationPage() {
    FlightUser user = new FlightUser();
    ModelAndView mv = new ModelAndView("newUserPage");
    mv.addObject("userRecord", user);
    return mv;
}

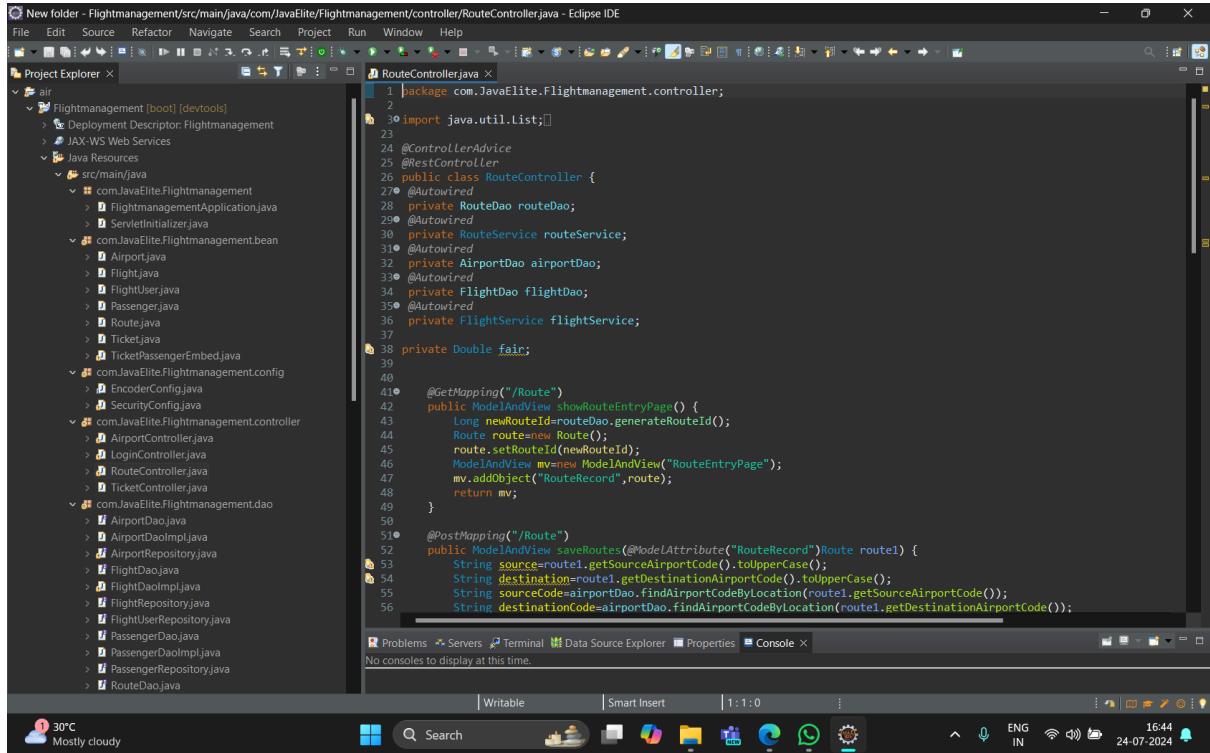
@PostMapping("/register")
public ModelAndView saveUserRegistrationPage(@ModelAttribute("userRecord") FlightUser user) {
    String encodedPassword = bCrypt.encode(user.getPassword()); // encrypts the password
    FlightUser newUser = new FlightUser();
    newUser.setUsername(user.getUsername());
    newUser.setPassword(encodedPassword);
    newUser.setType(user.getType());
    service.save(newUser);
    return new ModelAndView("loginPage");
}

@GetMapping("/loginPage")
public ModelAndView showLoginPage() {
    ModelAndView mv = new ModelAndView("loginPage");
    return mv;
}

@GetMapping("/loginerror")
public ModelAndView showLoginPage() {
    ModelAndView mv = new ModelAndView("loginPage");
    mv.addObject("error", "Invalid username or password.");
    return mv;
}

```

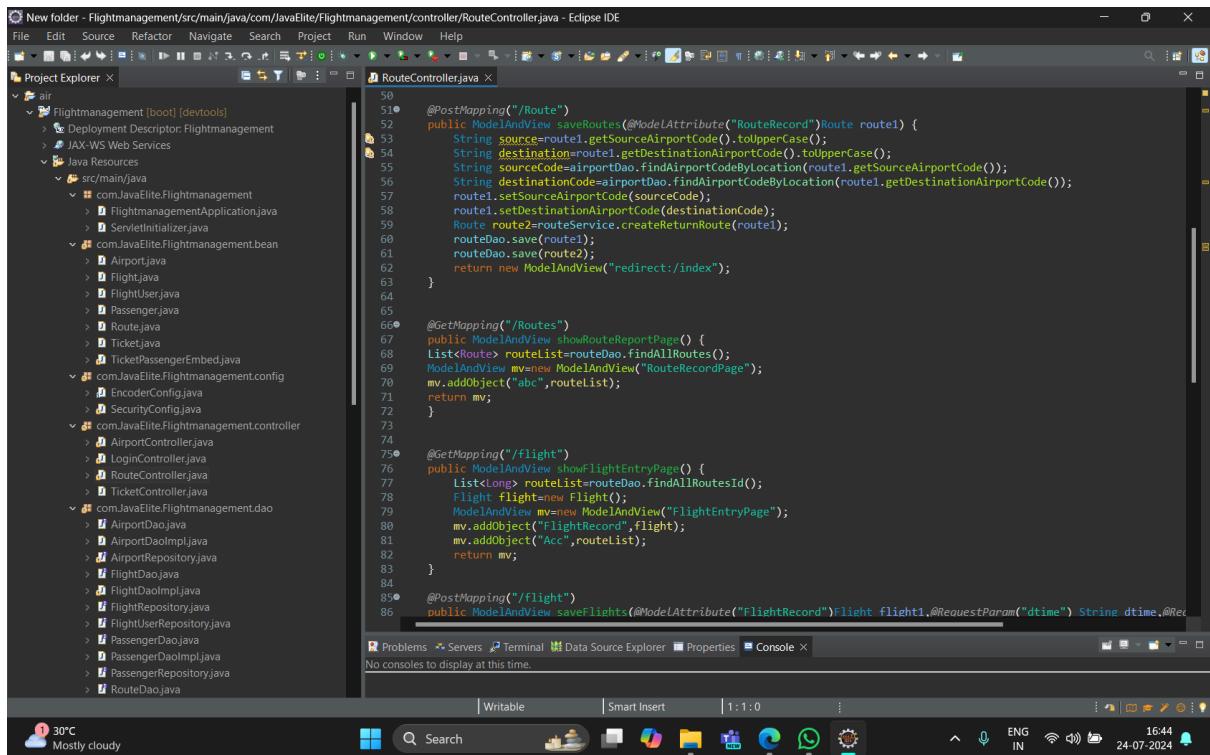
- **Route controller:** The Route Controller class is used to manage routes and flights within the Flight Management System. It provides functionality for creating, viewing, and searching routes and flights, as well as handling exceptions related to routing.



```

1 package com.Javaelite.Flightmanagement.controller;
2
3 import java.util.List;
4
5 @ControllerAdvice
6 @RestController
7 public class RouteController {
8     @Autowired
9     private RouteDao routeDao;
10    @Autowired
11    private RouteService routeService;
12    @Autowired
13    private AirportDao airportDao;
14    @Autowired
15    private FlightDao flightDao;
16    @Autowired
17    private PassengerDao passengerDao;
18
19    private Double fair;
20
21    @GetMapping("/Route")
22    public ModelAndView showRouteEntryPage() {
23        Long newRouteId=routeDao.generateRouteId();
24        Route route=new Route();
25        route.setRouteId(newRouteId);
26        ModelAndView mv=new ModelAndView("RouteEntryPage");
27        mv.addObject("RouteRecord",route);
28        return mv;
29    }
30
31    @PostMapping("/Route")
32    public ModelAndView saveRoutes(@ModelAttribute("RouteRecord")Route route1) {
33        String source=route1.getSourceAirportCode().toUpperCase();
34        String destination=route1.getDestinationAirportCode().toUpperCase();
35        String sourceCode=airportDao.findAirportCodeByLocation(route1.getSourceAirportCode());
36        String destinationCode=airportDao.findAirportCodeByLocation(route1.getDestinationAirportCode());
37
38        route1.setSourceAirportCode(sourceCode);
39        route1.setDestinationAirportCode(destinationCode);
40        Route route=routeService.createReturnRoute(route1);
41        routeDao.save(route1);
42        routeDao.save(route2);
43        return new ModelAndView("redirect:/index");
44    }
45
46    @GetMapping("/Routes")
47    public ModelAndView showRouteReportPage() {
48        List<Route> routelist=routeDao.findAllRoutes();
49        ModelAndView mv=new ModelAndView("RouteRecordPage");
50        mv.addObject("abc",routelist);
51        return mv;
52    }
53
54    @GetMapping("/Flight")
55    public ModelAndView showFlightEntryPage() {
56        List<Long> routelist=routeDao.findAllRoutesId();
57        Flight flight=new Flight();
58        ModelAndView mv=new ModelAndView("FlightEntryPage");
59        mv.addObject("FlightRecord",flight);
60        mv.addObject("Acc",routelist);
61        return mv;
62    }
63
64    @PostMapping("/Flight")
65    public ModelAndView saveFlights(@ModelAttribute("FlightRecord")Flight flight1,@RequestParam("dtime") String dtime,@Request

```



```

1 package com.Javaelite.Flightmanagement.controller;
2
3 import java.util.List;
4
5 @ControllerAdvice
6 @RestController
7 public class RouteController {
8     @Autowired
9     private RouteDao routeDao;
10    @Autowired
11    private RouteService routeService;
12    @Autowired
13    private AirportDao airportDao;
14    @Autowired
15    private FlightDao flightDao;
16    @Autowired
17    private PassengerDao passengerDao;
18
19    private Double fair;
20
21    @GetMapping("/Route")
22    public ModelAndView showRouteEntryPage() {
23        Long newRouteId=routeDao.generateRouteId();
24        Route route=new Route();
25        route.setRouteId(newRouteId);
26        ModelAndView mv=new ModelAndView("RouteEntryPage");
27        mv.addObject("RouteRecord",route);
28        return mv;
29    }
30
31    @PostMapping("/Route")
32    public ModelAndView saveRoutes(@ModelAttribute("RouteRecord")Route route1) {
33        String source=route1.getSourceAirportCode().toUpperCase();
34        String destination=route1.getDestinationAirportCode().toUpperCase();
35        String sourceCode=airportDao.findAirportCodeByLocation(route1.getSourceAirportCode());
36        String destinationCode=airportDao.findAirportCodeByLocation(route1.getDestinationAirportCode());
37
38        route1.setSourceAirportCode(sourceCode);
39        route1.setDestinationAirportCode(destinationCode);
40        Route route=routeService.createReturnRoute(route1);
41        routeDao.save(route1);
42        routeDao.save(route2);
43        return new ModelAndView("redirect:/index");
44    }
45
46    @GetMapping("/Routes")
47    public ModelAndView showRouteReportPage() {
48        List<Route> routelist=routeDao.findAllRoutes();
49        ModelAndView mv=new ModelAndView("RouteRecordPage");
50        mv.addObject("abc",routelist);
51        return mv;
52    }
53
54    @GetMapping("/Flight")
55    public ModelAndView showFlightEntryPage() {
56        List<Long> routelist=routeDao.findAllRoutesId();
57        Flight flight=new Flight();
58        ModelAndView mv=new ModelAndView("FlightEntryPage");
59        mv.addObject("FlightRecord",flight);
60        mv.addObject("Acc",routelist);
61        return mv;
62    }
63
64    @PostMapping("/Flight")
65    public ModelAndView saveFlights(@ModelAttribute("FlightRecord")Flight flight1,@RequestParam("dtime") String dtime,@Request

```

New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/RouteController.java - Eclipse IDE

```
85     @PostMapping("/flight")
86     public ModelAndView saveFlights(@ModelAttribute("FlightRecord")Flight flight1,@RequestParam("dtime") String dtime,@RequestParam("atime") String atime) {
87         Flight flight2=flightService.createReturnFlight(flight1,dtime,atime);
88         flightDao.save(flight1);
89         flightDao.save(flight2);
90         return new ModelAndView("redirect:/index");
91     }
92     @GetMapping("/flights")
93     public ModelAndView showFlightRecordPage() {
94         List<Flight> flightList=flightDao.findAllFlights();
95         ModelAndView mv=new ModelAndView("flightRecordPage");
96         mv.addObject("flight",flightList);
97         return mv;
98     }
99
100    @GetMapping("/flight-search")
101    public ModelAndView showRouteSelectPage() {
102        List<String> airportList=airportDao.findallAirportLocations();
103        ModelAndView mv=new ModelAndView("FlightSelectPage");
104        mv.addObject("abc",airportList);
105        return mv;
106    }
107
108    @PostMapping("/flight-search")
109    public ModelAndView showRouteFlightsPage(@RequestParam("fromCity")String fromCity,@RequestParam("toCity")String toCity) {
110        String fromAirport=airportDao.findAirportCodeByLocation(fromCity);
111        String toAirport=airportDao.findAirportCodeByLocation(toCity);
112        if(fromAirport.equalsIgnoreCase(toAirport))
113            throw new RouteNotFoundException();
114        Route route=routeDao.findRouteBySourceAndDestination(fromAirport, toAirport);
115        List<Flight> flightList=flightDao.findFlightsByRouteId(route.getRouteId());
116        ModelAndView mv=new ModelAndView("routeFlightShowPage");
117        mv.addObject("flightList",flightList);
118        mv.addObject("fromAirport",fromCity);
119        mv.addObject("toAirport",toCity);
120        mv.addObject("fair",route.getFare());
121        mv.addObject("fare",route.getFare());
122    }

```

30°C Mostly cloudy

16:45 24-07-2024

New folder - Flightmanagement/src/main/java/com/JavaElite/Flightmanagement/controller/RouteController.java - Eclipse IDE

```
100    @GetMapping("/flight-search")
101    public ModelAndView showRouteSelectPage() {
102        List<String> airportList=airportDao.findallAirportLocations();
103        ModelAndView mv=new ModelAndView("FlightSelectPage");
104        mv.addObject("abc",airportList);
105        return mv;
106    }
107
108    @PostMapping("/flight-search")
109    public ModelAndView showRouteFlightsPage(@RequestParam("fromCity")String fromCity,@RequestParam("toCity")String toCity) {
110        String fromAirport=airportDao.findAirportCodeByLocation(fromCity);
111        String toAirport=airportDao.findAirportCodeByLocation(toCity);
112        if(fromAirport.equalsIgnoreCase(toAirport))
113            throw new RouteNotFoundException();
114        Route route=routeDao.findRouteBySourceAndDestination(fromAirport, toAirport);
115        List<Flight> flightList=flightDao.findFlightsByRouteId(route.getRouteId());
116        ModelAndView mv=new ModelAndView("routeFlightShowPage");
117        mv.addObject("flightList",flightList);
118        mv.addObject("fromAirport",fromCity);
119        mv.addObject("toAirport",toCity);
120        mv.addObject("fair",route.getFare());
121        mv.addObject("fare",route.getFare());
122        return mv;
123    }
124
125    @ExceptionHandler(value=RouteNotFoundException.class)
126    public ModelAndView handlingRouteException(RouteNotFoundException exception) {
127        String message="From City & To City cannot be the same.....";
128        ModelAndView mv=new ModelAndView("routeErrorPage");
129        mv.addObject("errorMessage",message);
130        return mv;
131    }
132
133    }
134 }
```

30°C Mostly cloudy

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.

Screenshot of Eclipse IDE showing the code for `TicketController.java`. The code implements a REST controller for ticket bookings, using annotations like `@GetMapping` and `@PostMapping` to handle HTTP requests. It interacts with various DAOs and services to manage flights, routes, and passengers.

```

1 package com.JavaElite.Flightmanagement.controller;
2
3 import java.util.ArrayList;
4
5 @ControllerAdvice
6 @RestController
7 public class TicketController {
8     @Autowired
9     private TicketDao ticketDao;
10
11     @Autowired
12     private FlightDao flightDao;
13
14     @Autowired
15     private RouteDao routeDao;
16
17     @Autowired
18     private TicketService ticketService;
19
20     @Autowired
21     private PassengerDao passengerDao;
22
23     @GetMapping("/ticket/{id}")
24     public ModelAndView showTicketBookingPage(@PathVariable Long id) {
25         Flight flight=flightDao.findFlightById(id);
26         Route route=routeDao.findRouteById(flight.getRouteId());
27         Long newTicketId=ticketDao.findLastTicketNumber();
28         System.out.println(newTicketId);
29         Ticket ticket=new Ticket();
30         ticket.setTicketNumber(newTicketId);
31         ticket.setFlightNumber(flight.getFlightNumber());
32         ticket.setCarrierName(flight.getCarrierName());
33         ticket.setRouteId(route.getId());
34         ticket.setFlightId(flight.getId());
35         ModelAndView mv=new ModelAndView("ticketBookingPage");
36         mv.addObject("ticketRecord",ticket);
37         mv.addObject("flight",flight);
38         mv.addObject("route",route);
39         return mv;
40     }
41
42     @PostMapping("/ticket")
43     public ModelAndView openShowTicketPage(@ModelAttribute("ticketRecord")Ticket ticket,HttpServletRequest request) {
44         List<Passenger> passengerList=new ArrayList<>();
45         String fromCity=request.getParameter("fromlocation");
46         String toCity=request.getParameter("tolocation");
47         Double fair=Double.parseDouble(request.getParameter("fair"));
48         String pname="";
49         String dob="";
50         for(int i=1;i<6;i++){
51             pname=request.getParameter("name"+i);
52             if(pname.equals("--")){
53                 dob=request.getParameter("dob"+i);
54                 TicketPassengerEmbed embed=new TicketPassengerEmbed(ticket.getTicketNumber(),i);
55                 Passenger passenger=new Passenger(embed,pname,dob,Fair);
56                 passenger.setFair(fair);
57                 passengerList.add(passenger);
58             }
59             else {
60                 break;
61             }
62         }
63         //end of loop
64         int size=passengerList.size();
65         if(ticketService.capacityCalculation(size,ticket.getFlightNumber())){
66             ticketDao.save(ticket);
67             for(Passenger passenger:passengerList) {
68                 passengerDao.save(passenger);
69             }
70         }
71         else {
72             throw new SeatNotFoundException();
73         }
74         Double totalAmount=ticketService.totalBillCalculation(passengerList);
75     }
76 }

```

Screenshot of Eclipse IDE showing the continuation of the `TicketController.java` code. This part handles the submission of ticket details, including fare calculations and seat availability checks. It uses the `TicketService` and `PassengerDao` to save the ticket and passengers respectively.

```

1 package com.JavaElite.Flightmanagement.controller;
2
3 import java.util.ArrayList;
4
5 @ControllerAdvice
6 @RestController
7 public class TicketController {
8     @Autowired
9     private TicketDao ticketDao;
10
11     @Autowired
12     private FlightDao flightDao;
13
14     @Autowired
15     private RouteDao routeDao;
16
17     @Autowired
18     private TicketService ticketService;
19
20     @Autowired
21     private PassengerDao passengerDao;
22
23     @PostMapping("/ticket")
24     public ModelAndView openShowTicketPage(@ModelAttribute("ticketRecord")Ticket ticket,HttpServletRequest request) {
25         List<Passenger> passengerList=new ArrayList<>();
26         String fromCity=request.getParameter("fromlocation");
27         String toCity=request.getParameter("tolocation");
28         Double fair=Double.parseDouble(request.getParameter("fair"));
29         String pname="";
30         String dob="";
31         for(int i=1;i<6;i++){
32             pname=request.getParameter("name"+i);
33             if(pname.equals("--")){
34                 dob=request.getParameter("dob"+i);
35                 TicketPassengerEmbed embed=new TicketPassengerEmbed(ticket.getTicketNumber(),i);
36                 Passenger passenger=new Passenger(embed,pname,dob,Fair);
37                 passenger.setFair(fair);
38                 passengerList.add(passenger);
39             }
40             else {
41                 break;
42             }
43         }
44         //end of loop
45         int size=passengerList.size();
46         if(ticketService.capacityCalculation(size,ticket.getFlightNumber())){
47             ticketDao.save(ticket);
48             for(Passenger passenger:passengerList) {
49                 passengerDao.save(passenger);
50             }
51         }
52         else {
53             throw new SeatNotFoundException();
54         }
55         Double totalAmount=ticketService.totalBillCalculation(passengerList);
56     }
57 }

```

```

1 package com.JavaElite.Flightmanagement.controller;
2
3 import java.util.List;
4 import java.util.Map;
5 import java.util.Optional;
6
7 import javax.persistence.EntityManager;
8 import javax.persistence.EntityTransaction;
9 import javax.persistence.Query;
10
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.http.ResponseEntity;
13 import org.springframework.web.bind.annotation.*;
14
15 import com.JavaElite.Flightmanagement.model.Ticket;
16 import com.JavaElite.Flightmanagement.model.TicketPassenger;
17 import com.JavaElite.Flightmanagement.model.TicketPassengerEmbed;
18 import com.JavaElite.Flightmanagement.model.TicketUser;
19 import com.JavaElite.Flightmanagement.model.Route;
20 import com.JavaElite.Flightmanagement.model.Airport;
21 import com.JavaElite.Flightmanagement.model.Flight;
22 import com.JavaElite.Flightmanagement.model.Passenger;
23 import com.JavaElite.Flightmanagement.model.Fare;
24 import com.JavaElite.Flightmanagement.model.Ticket;
25 import com.JavaElite.Flightmanagement.model.Ticket;
26 import com.JavaElite.Flightmanagement.model.Ticket;
27 import com.JavaElite.Flightmanagement.model.Ticket;
28 import com.JavaElite.Flightmanagement.model.Ticket;
29 import com.JavaElite.Flightmanagement.model.Ticket;
30 import com.JavaElite.Flightmanagement.model.Ticket;
31 import com.JavaElite.Flightmanagement.model.Ticket;
32 import com.JavaElite.Flightmanagement.model.Ticket;
33 import com.JavaElite.Flightmanagement.model.Ticket;
34 import com.JavaElite.Flightmanagement.model.Ticket;
35 import com.JavaElite.Flightmanagement.model.Ticket;
36 import com.JavaElite.Flightmanagement.model.Ticket;
37 import com.JavaElite.Flightmanagement.model.Ticket;
38 import com.JavaElite.Flightmanagement.model.Ticket;
39 import com.JavaElite.Flightmanagement.model.Ticket;
40 import com.JavaElite.Flightmanagement.model.Ticket;
41 import com.JavaElite.Flightmanagement.model.Ticket;
42 import com.JavaElite.Flightmanagement.model.Ticket;
43 import com.JavaElite.Flightmanagement.model.Ticket;
44 import com.JavaElite.Flightmanagement.model.Ticket;
45 import com.JavaElite.Flightmanagement.model.Ticket;
46 import com.JavaElite.Flightmanagement.model.Ticket;
47 import com.JavaElite.Flightmanagement.model.Ticket;
48 import com.JavaElite.Flightmanagement.model.Ticket;
49 import com.JavaElite.Flightmanagement.model.Ticket;
50 import com.JavaElite.Flightmanagement.model.Ticket;
51 import com.JavaElite.Flightmanagement.model.Ticket;
52 import com.JavaElite.Flightmanagement.model.Ticket;
53 import com.JavaElite.Flightmanagement.model.Ticket;
54 import com.JavaElite.Flightmanagement.model.Ticket;
55 import com.JavaElite.Flightmanagement.model.Ticket;
56 import com.JavaElite.Flightmanagement.model.Ticket;
57 import com.JavaElite.Flightmanagement.model.Ticket;
58 import com.JavaElite.Flightmanagement.model.Ticket;
59 import com.JavaElite.Flightmanagement.model.Ticket;
60 import com.JavaElite.Flightmanagement.model.Ticket;
61 import com.JavaElite.Flightmanagement.model.Ticket;
62 import com.JavaElite.Flightmanagement.model.Ticket;
63 import com.JavaElite.Flightmanagement.model.Ticket;
64 import com.JavaElite.Flightmanagement.model.Ticket;
65 import com.JavaElite.Flightmanagement.model.Ticket;
66 import com.JavaElite.Flightmanagement.model.Ticket;
67 import com.JavaElite.Flightmanagement.model.Ticket;
68 import com.JavaElite.Flightmanagement.model.Ticket;
69 import com.JavaElite.Flightmanagement.model.Ticket;
70 import com.JavaElite.Flightmanagement.model.Ticket;
71 import com.JavaElite.Flightmanagement.model.Ticket;
72 import com.JavaElite.Flightmanagement.model.Ticket;
73 import com.JavaElite.Flightmanagement.model.Ticket;
74 import com.JavaElite.Flightmanagement.model.Ticket;
75 import com.JavaElite.Flightmanagement.model.Ticket;
76 import com.JavaElite.Flightmanagement.model.Ticket;
77 import com.JavaElite.Flightmanagement.model.Ticket;
78 import com.JavaElite.Flightmanagement.model.Ticket;
79 import com.JavaElite.Flightmanagement.model.Ticket;
80 import com.JavaElite.Flightmanagement.model.Ticket;
81 import com.JavaElite.Flightmanagement.model.Ticket;
82 import com.JavaElite.Flightmanagement.model.Ticket;
83 import com.JavaElite.Flightmanagement.model.Ticket;
84 import com.JavaElite.Flightmanagement.model.Ticket;
85 import com.JavaElite.Flightmanagement.model.Ticket;
86 import com.JavaElite.Flightmanagement.model.Ticket;
87 import com.JavaElite.Flightmanagement.model.Ticket;
88 import com.JavaElite.Flightmanagement.model.Ticket;
89 import com.JavaElite.Flightmanagement.model.Ticket;
90 import com.JavaElite.Flightmanagement.model.Ticket;
91 import com.JavaElite.Flightmanagement.model.Ticket;
92 import com.JavaElite.Flightmanagement.model.Ticket;
93 import com.JavaElite.Flightmanagement.model.Ticket;
94 import com.JavaElite.Flightmanagement.model.Ticket;
95 import com.JavaElite.Flightmanagement.model.Ticket;
96 import com.JavaElite.Flightmanagement.model.Ticket;
97 import com.JavaElite.Flightmanagement.model.Ticket;
98 import com.JavaElite.Flightmanagement.model.Ticket;
99 import com.JavaElite.Flightmanagement.model.Ticket;
100 import com.JavaElite.Flightmanagement.model.Ticket;
101 import com.JavaElite.Flightmanagement.model.Ticket;
102 import com.JavaElite.Flightmanagement.model.Ticket;
103 import com.JavaElite.Flightmanagement.model.Ticket;
104 import com.JavaElite.Flightmanagement.model.Ticket;
105 import com.JavaElite.Flightmanagement.model.Ticket;
106 import com.JavaElite.Flightmanagement.model.Ticket;
107 import com.JavaElite.Flightmanagement.model.Ticket;
108 import com.JavaElite.Flightmanagement.model.Ticket;
109 import com.JavaElite.Flightmanagement.model.Ticket;
110 import com.JavaElite.Flightmanagement.model.Ticket;
111 }

```

5)Flight Management. Dao:

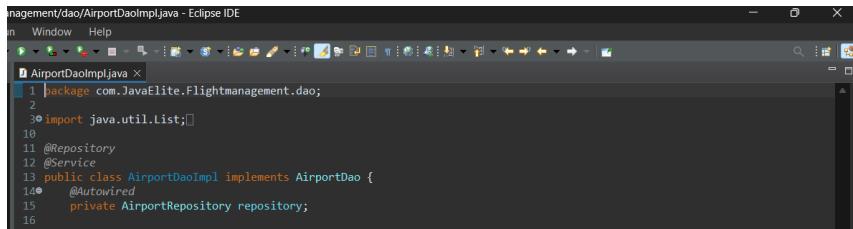
- **AirportDao.java:** The Airport Dao interface provides methods for managing airport data in the Flight Management System. It includes operations for adding new airports, retrieving airports by ID, and listing all airports. It also offers methods to get airport locations and find airport codes based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

```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 public interface AirportDao {
6     public void addAirport(Airport airport);
7     public Airport findAirportById(String id);
8     public List<Airport> findAllAirports();
9     public List<String> findAllAirportLocations();
10    public String findAirportCodeByLocation(String location);
11 }
12
13
14
15

```

- **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.

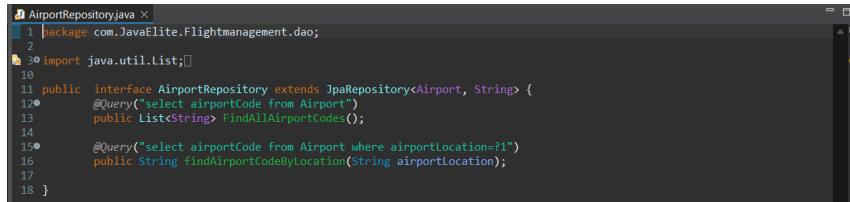


```

management/dao/AirportDaoImpl.java - Eclipse IDE
File Window Help
AirportDaoImpl.java ×
1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 @Repository
6 @Service
7 public class AirportDaoImpl implements AirportDao {
8
9     @Autowired
10    private AirportRepository repository;
11
12 }

```

- **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.



```

AirportRepository.java ×
1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 public interface AirportRepository extends JpaRepository<Airport, String> {
6
7     @Query("select airportCode from Airport")
8     public List<String> FindAllAirportCodes();
9
10    @Query("select airportCode from Airport where airportlocation=?1")
11    public String findAirportCodeByLocation(String airportlocation);
12
13 }

```

- **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.

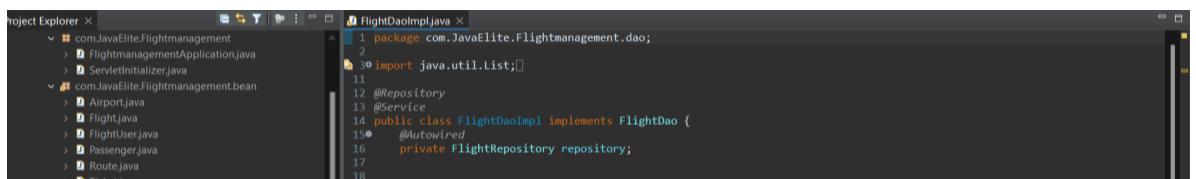


```

Project Explorer ×
com.JavaElite.Flightmanagement
  > FlightmanagementApplication.java
  > ServletInitializer.java
  > com.JavaElite.Flightmanagement.bean
    > Airport.java
    > Flight.java
    > FlightUser.java
    > Passenger.java
    > Route.java
    > Ticket.java
    > TicketPassengerEmbed.java
FlightDao.java ×
1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 public interface FlightDao {
6
7     public void save(Flight flight);
8
9     public List<Flight> findAllFlights();
10    public List<Flight> findFlightsByRouteId(Long routeId);
11    public Flight findFlightById(Long id);
12
13 }

```

- **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.

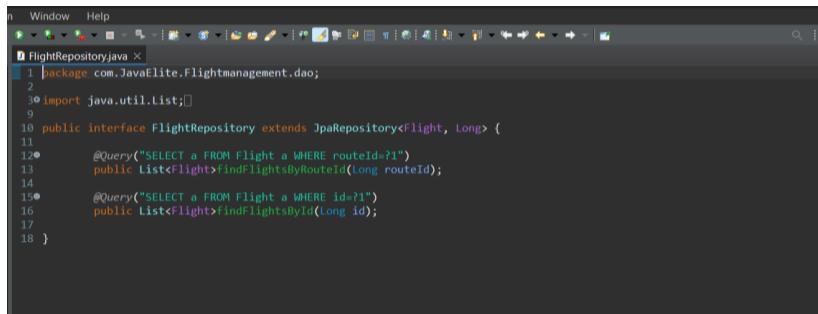


```

Project Explorer ×
com.JavaElite.Flightmanagement
  > FlightmanagementApplication.java
  > ServletInitializer.java
  > com.JavaElite.Flightmanagement.bean
    > Airport.java
    > Flight.java
    > FlightUser.java
    > Passenger.java
    > Route.java
    > Ticket.java
    > TicketPassengerEmbed.java
FlightDaoImpl.java ×
1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 @Repository
6 @Service
7 public class FlightDaoImpl implements FlightDao {
8
9     @Autowired
10    private FlightRepository repository;
11
12 }

```

- **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.

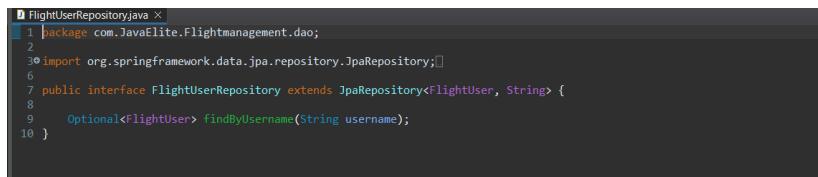


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 public interface FlightRepository extends JpaRepository<Flight, Long> {
6
7     @Query("SELECT a FROM Flight a WHERE routeId=?1")
8     public List<Flight> findFlightsByRouteId(Long routeId);
9
10    @Query("SELECT a FROM Flight a WHERE id=?1")
11    public List<Flight> findFlightsById(Long id);
12
13 }

```

- **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.

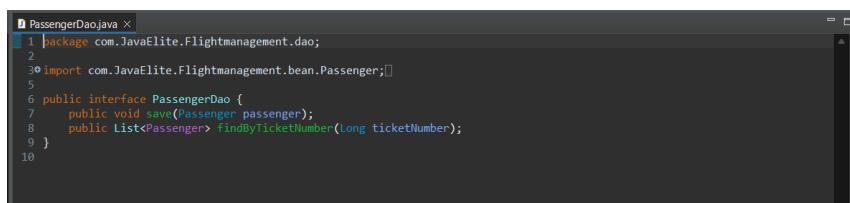


```

1 FlightUserRepository.java x
2 package com.JavaElite.Flightmanagement.dao;
3
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface FlightUserRepository extends JpaRepository<FlightUser, String> {
7
8     Optional<FlightUser> findByUsername(String username);
9
10 }

```

- **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.

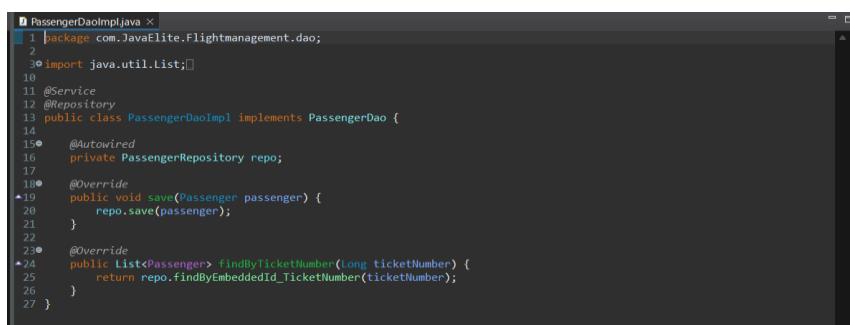


```

1 PassengerDao.java x
2 package com.JavaElite.Flightmanagement.dao;
3
4 import com.JavaElite.Flightmanagement.bean.Passenger;
5
6 public interface PassengerDao extends JpaRepository<Passenger, Long> {
7
8     public void save(Passenger passenger);
9     public List<Passenger> findByTicketNumber(Long ticketNumber);
10 }

```

- **PassengerDaoImpl.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.

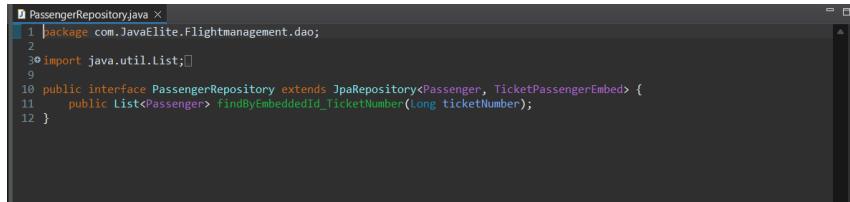


```

1 PassengerDaoImpl.java x
2 package com.JavaElite.Flightmanagement.dao;
3
4 import java.util.List;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8 import org.springframework.stereotype.Repository;
9 import org.springframework.data.jpa.repository.JpaRepository;
10
11 @Service
12 @Repository
13 public class PassengerDaoImpl implements PassengerDao {
14
15     @Autowired
16     private PassengerRepository repo;
17
18     @Override
19     public void save(Passenger passenger) {
20         repo.save(passenger);
21     }
22
23     @Override
24     public List<Passenger> findByTicketNumber(Long ticketNumber) {
25         return repo.findByEmbeddedId_TicketNumber(ticketNumber);
26     }
27 }

```

- **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.

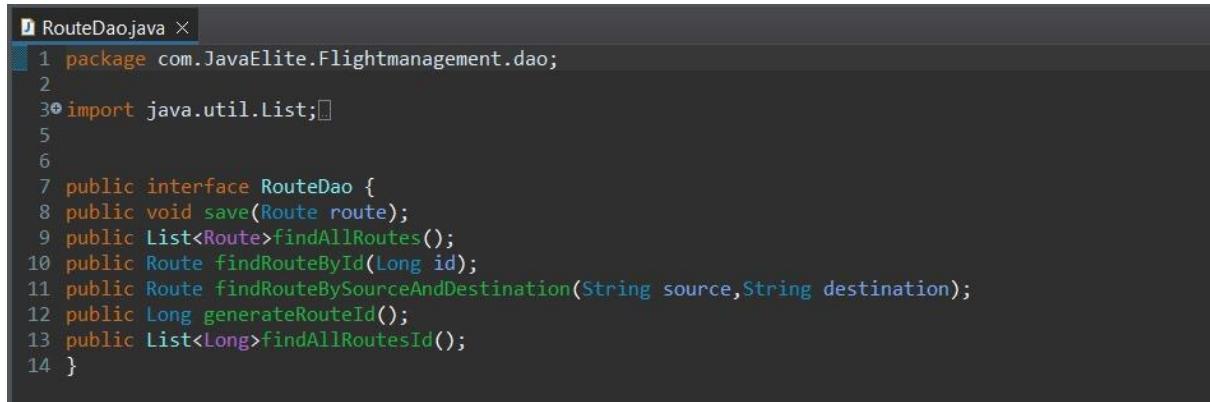


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5 public interface PassengerRepository extends JpaRepository<Passenger, TicketPassengerEmbed> {
6     public List<Passenger> findByEmbeddedId_TicketNumber(Long ticketNumber);
7 }

```

- **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.

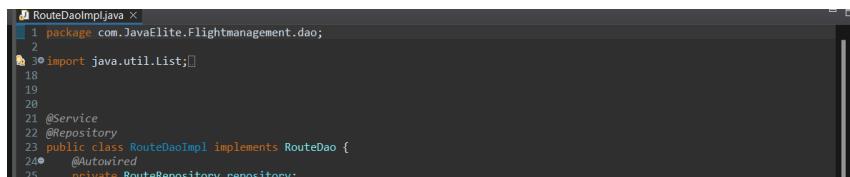


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5
6 public interface RouteDao {
7     public void save(Route route);
8     public List<Route> findAllRoutes();
9     public Route findRouteById(Long id);
10    public Route findRouteBySourceAndDestination(String source, String destination);
11    public Long generateRouteId();
12    public List<Long> findAllRoutesId();
13 }
14

```

- **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.

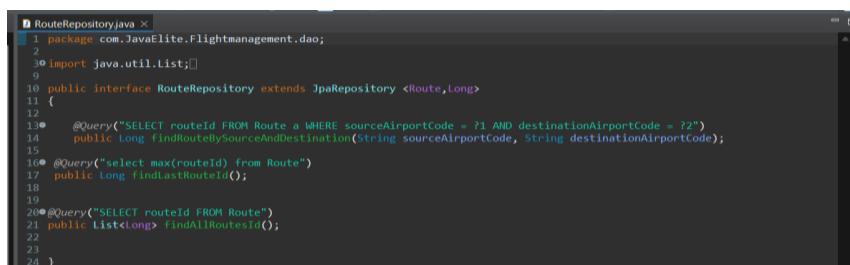


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5
6 @Service
7 @Repository
8 public class RouteDaoImpl implements RouteDao {
9     @Autowired
10    private RouteRepository repository;
11
12 }
13

```

- **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.

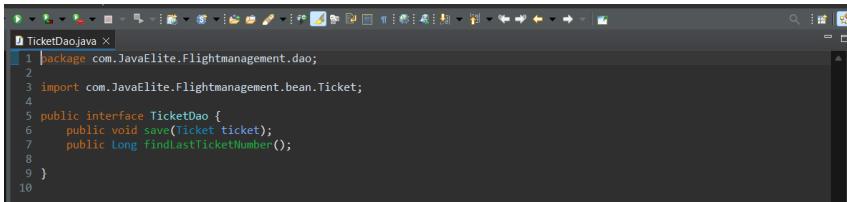


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import java.util.List;
4
5
6 public interface RouteRepository extends JpaRepository<Route, Long> {
7
8     @Query("SELECT routeId FROM Route r WHERE sourceAirportCode = ?1 AND destinationAirportCode = ?2")
9     public Long findRouteIdBySourceAndDestination(String sourceAirportCode, String destinationAirportCode);
10
11     @Query("Select max(routeId) From Route")
12     public Long findlastRouteId();
13
14     @Query("SELECT routId FROM Route")
15     public List<Long> findAllRoutesId();
16
17 }
18

```

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

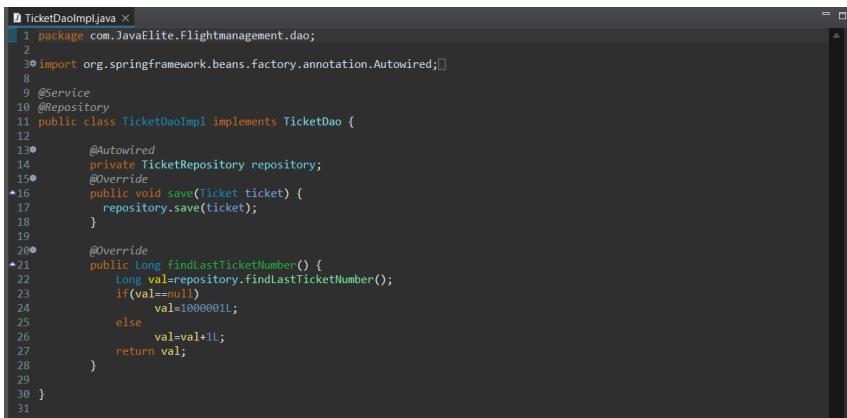


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import com.JavaElite.Flightmanagement.bean.Ticket;
4
5 public interface TicketDao {
6     public void save(Ticket ticket);
7     public Long findLastTicketNumber();
8 }
9
10

```

- **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.

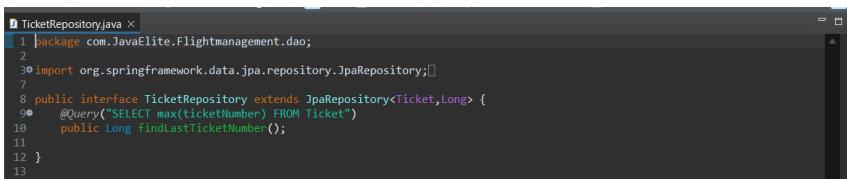


```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 @Repository
7 public class TicketDaoImpl implements TicketDao {
8
9     @Autowired
10     private TicketRepository repository;
11
12     @Override
13     public void save(Ticket ticket) {
14         repository.save(ticket);
15     }
16
17     @Override
18     public Long findLastTicketNumber() {
19         Long val=repository.findLastTicketNumber();
20         if(val==null)
21             val=1000001L;
22         else
23             val=val+1L;
24         return val;
25     }
26
27 }
28
29
30
31

```

- **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.



```

1 package com.JavaElite.Flightmanagement.dao;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface TicketRepository extends JpaRepository<Ticket,Long> {
6     @Query("SELECT max(ticketNumber) FROM Ticket")
7     public Long findLastTicketNumber();
8 }
9
10
11
12
13

```

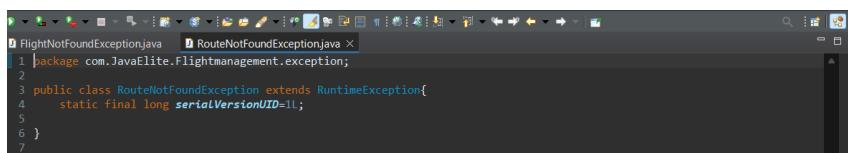
6)JavaElite.Flightmanagement.java

- **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.



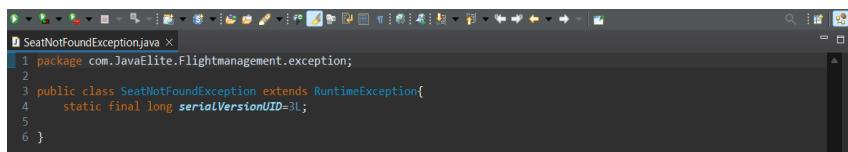
```
FlightNotFoundException.java ×
1 package com.JavaElite.Flightmanagement.exception;
2
3
4 public class FlightNotFoundException extends RuntimeException{
5     static final long serialVersionUID=2L;
6 }
7
```

- **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.



```
FlightNotFoundException.java × RouteNotFoundException.java ×
1 package com.JavaElite.Flightmanagement.exception;
2
3 public class RouteNotFoundException extends RuntimeException{
4     static final long serialVersionUID=1L;
5 }
6
7
```

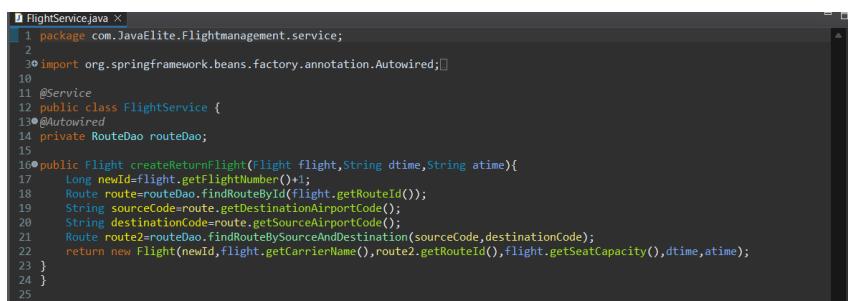
- **SeatNotFoundException.java:** Seat Not Found Exception is a custom runtime exception indicating that a requested seat is not available. It extends Runtime Exception and includes a serial version UID for serialization. This exception is used to handle scenarios where seat availability checks fail.



```
SeatNotFoundException.java ×
1 package com.JavaElite.Flightmanagement.exception;
2
3 public class SeatNotFoundException extends RuntimeException{
4     static final long serialVersionUID=3L;
5 }
6
7
```

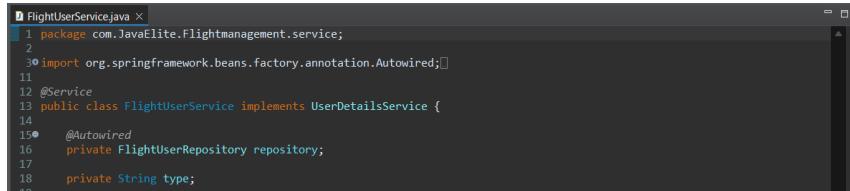
7)Java Elite. Flight management. service:

- **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.



```
FlightService.java ×
1 package com.JavaElite.Flightmanagement.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 public class FlightService {
7     @Autowired
8     private RouteDao routeDao;
9
10    public Flight createReturnFlight(Flight flight, String dtime, String atime){
11        Long newId=flight.getFlightNumber()+1;
12        Route route=routeDao.findRouteById(flight.getRouteId());
13        String sourceCode=route.getDestinationAirportCode();
14        String destinationCode=route.getSourceAirportCode();
15        Route route2=routeDao.findRouteBySourceAndDestination(sourceCode,destinationCode);
16        return new Flight(newId,flight.getCarrierName(),route2.getRouteId(),flight.getSeatCapacity(),dtime,atime);
17    }
18}
19
```

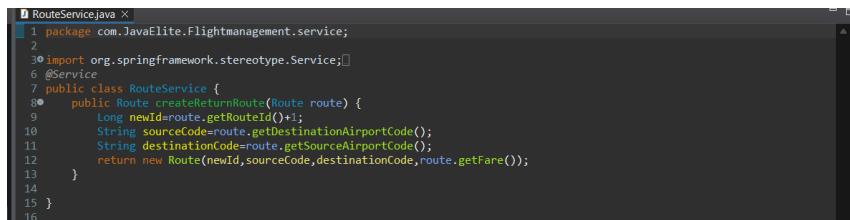
- **FlightUserService.java:** Flight Service is a Spring service component that provides flight-related operations. It has a dependency on Route Dao for accessing route information. The create Return Flight method generates a new return flight based on the provided flight details, including calculating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



```

1 package com.JavaElite.Flightmanagement.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 public class FlightUserService implements UserDetailsService {
7
8     @Autowired
9     private FlightUserRepository repository;
10
11     private String type;
12 }
```

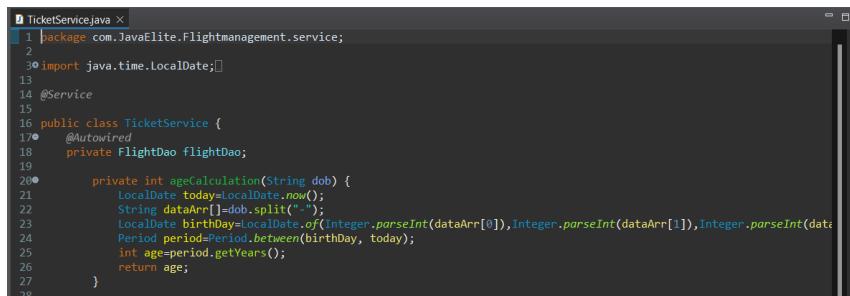
- **RouteService.java:** RouteService provides functionality for managing Route objects. The createReturnRoute method generates a new Route instance representing the return journey, by incrementing the route ID and swapping the source and destination airport codes from the given route.



```

1 package com.JavaElite.Flightmanagement.service;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class RouteService {
7
8     public Route createReturnRoute(Route route) {
9         Long newId=route.getRouteId()+1;
10        String sourceCode=route.getDestinationAirportCode();
11        String destinationCode=route.getSourceAirportCode();
12        return new Route(newId,sourceCode,destinationCode,route.getFare());
13    }
14
15 }
16 }
```

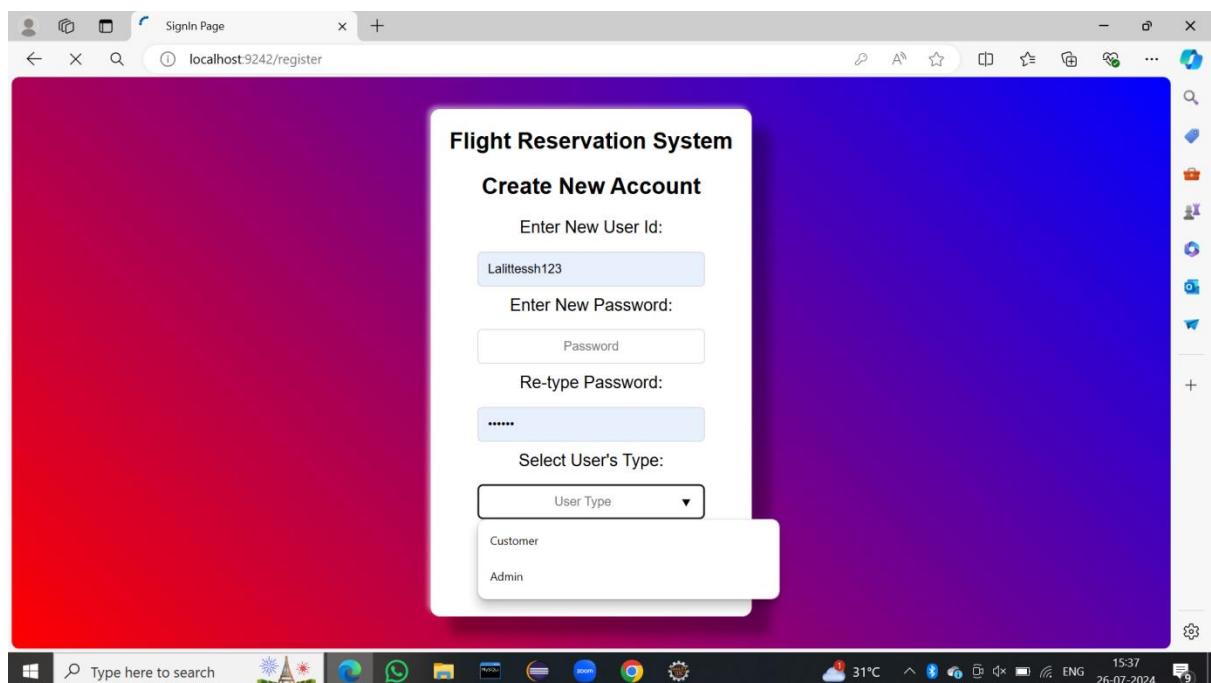
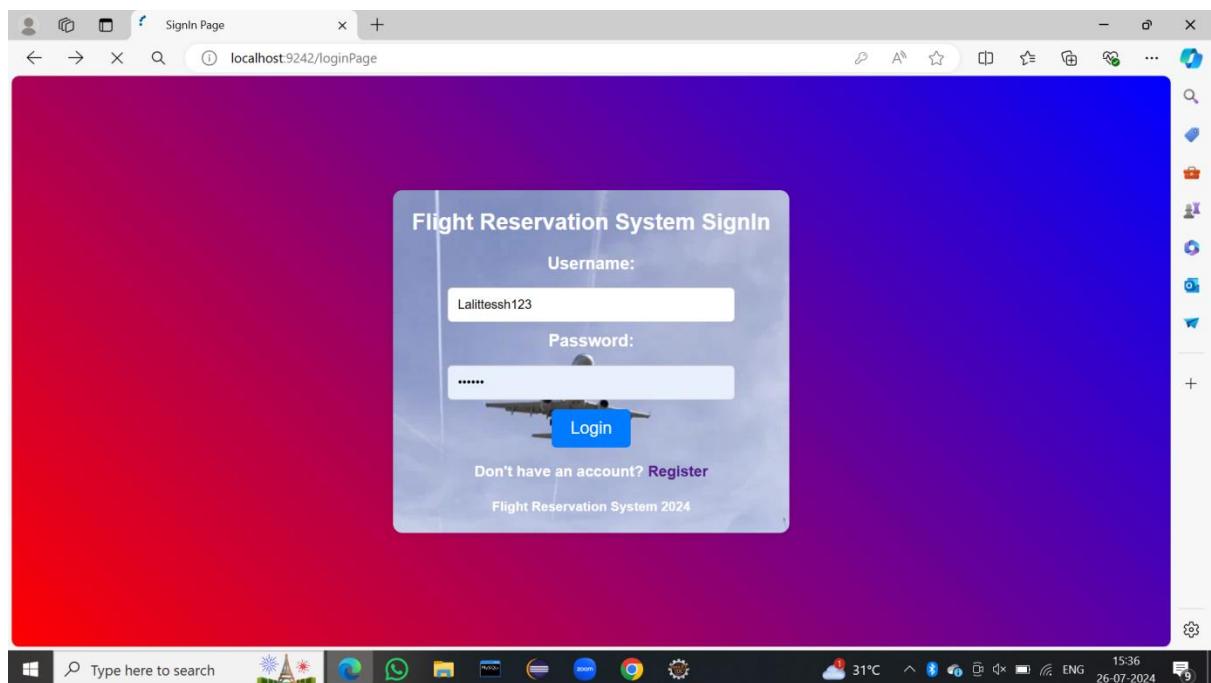
- **TicketService.java:** Ticket Service manages ticket-related operations, including calculating fare discounts based on passenger age, updating flight seat capacities, and summing up total fares for bookings. It ensures proper fare adjustments and seat availability for efficient ticket management.

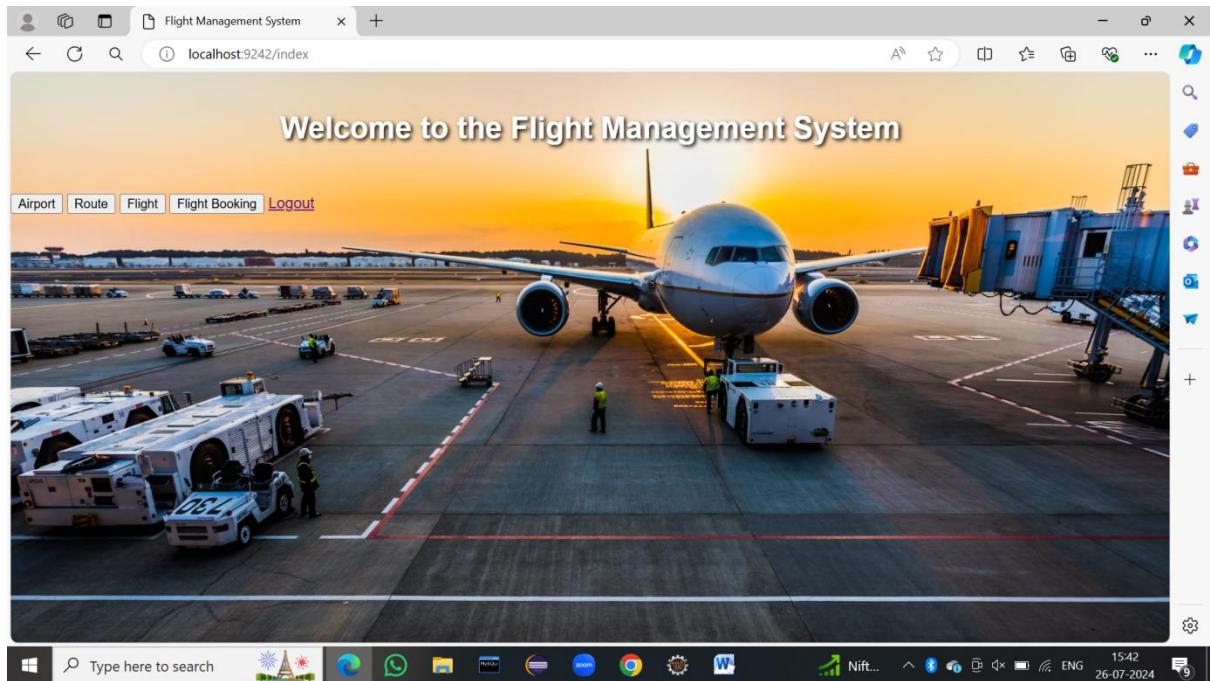


```

1 package com.JavaElite.Flightmanagement.service;
2
3 import java.time.LocalDate;
4
5 @Service
6 public class TicketService {
7
8     @Autowired
9     private FlightDao flightDao;
10
11     private int ageCalculation(String dob) {
12         LocalDate today=LocalDate.now();
13         String dataArr[]=dob.split("-");
14         LocalDate birthDay=LocalDate.of(Integer.parseInt(dataArr[0]),Integer.parseInt(dataArr[1]),Integer.parseInt(dataArr[2]));
15         Period period=Period.between(birthDay,today);
16         int age=period.getYears();
17         return age;
18     }
19 }
```

OUTPUTS:





The screenshot displays a list of airports under the heading "All Airports". The table includes columns for "Airport Code" and "Airport Location", along with "Enquire" and "Cancel" buttons for each entry. The entries are:

Airport Code	Airport Location	Enquire	Cancel
ARU	ARUNCHAL	Enquire	Cancel
CHE	CHENNAI	Enquire	Cancel
HYD	HYDRABAD	Enquire	Cancel
KOL	KOLKATA	Enquire	Cancel
MUM	MUMBAI	Enquire	Cancel
TAM	TAMILNADU	Enquire	Cancel
VIS	VISHAKAPATANAM	Enquire	Cancel

A "Return" button is located at the bottom of the table. The bottom of the screen shows a Windows taskbar with various icons and system status information.

All Routes

Route Number	Source Airport Code	Destination Airport Code	Route Fare	Action
101	CHE	HYD	5000.0	<button>Cancel</button>
102	HYD	CHE	5000.0	<button>Cancel</button>
105		TAM	2000.0	<button>Cancel</button>
107	VIS	TAM	2000.0	<button>Cancel</button>
108	TAM	VIS	2000.0	<button>Cancel</button>

Return

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

Flights from VISHAKAPATANAM x +

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

SEN... 15:38 26-07-2024

Ticket Booking x +

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:	Date of Birth:
Dadi Lalittesh	26-07-2024
Name:	Date of Birth:
--	dd-mm-yyyy
Name:	Date of Birth:
--	dd-mm-yyyy

Submit

Type here to search

SEN... 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](#)

