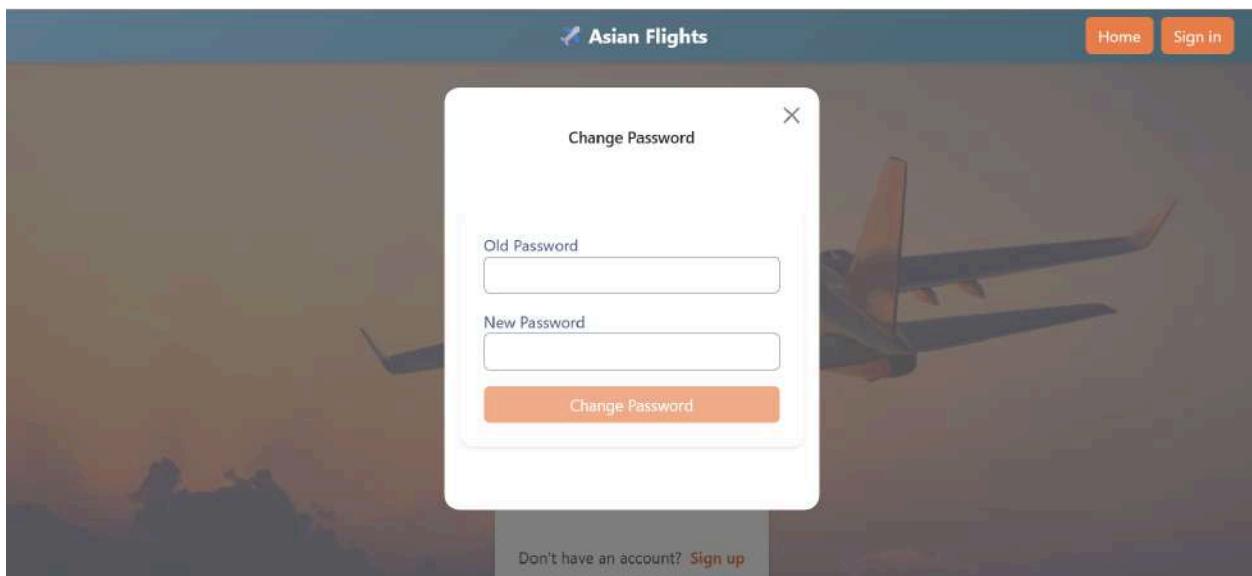


## Repositories links:

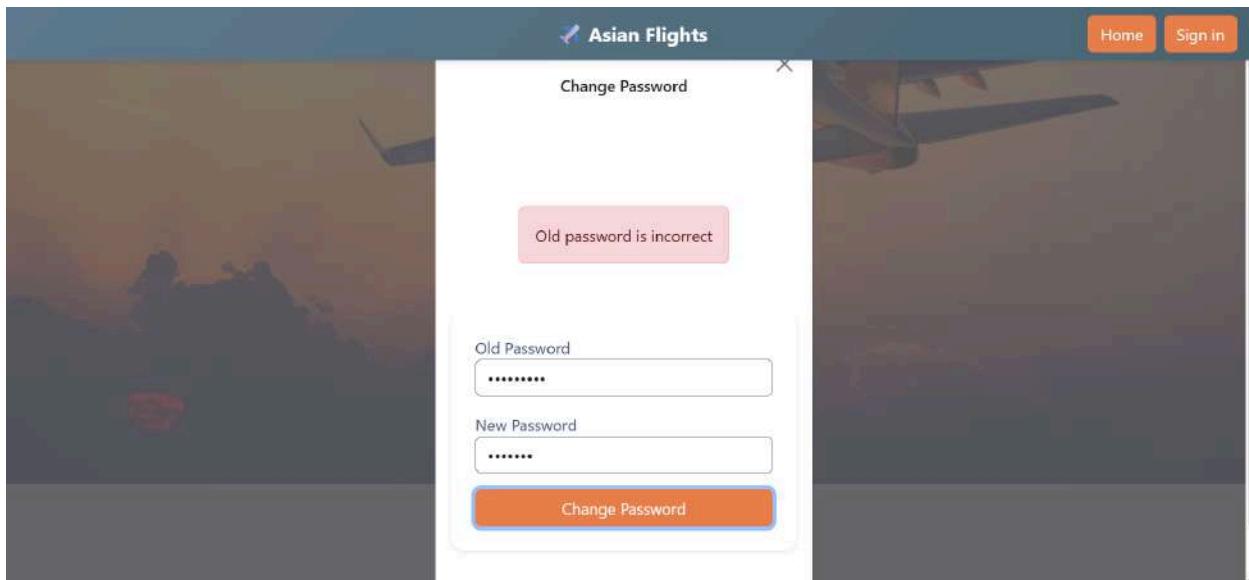
- Frontend:  
<https://github.com/asritha26k/frontend-for-flight-booking-app>
- Backend:  
<https://github.com/asritha26k/backend-flight-booking-app>

Change password initiated after every 90 days:

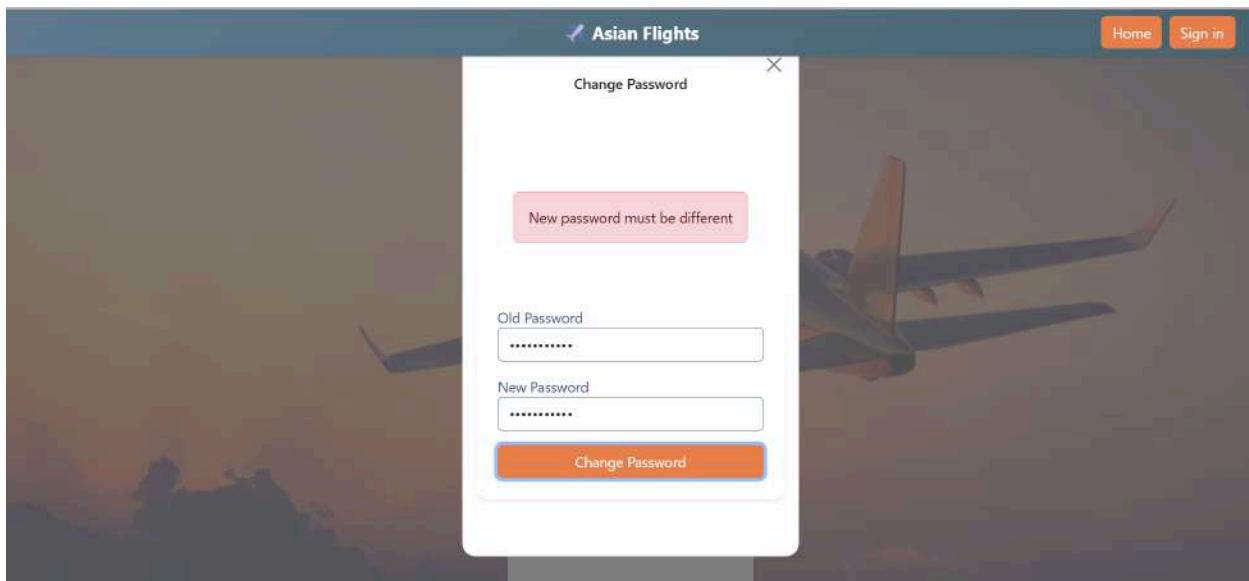
(Tested with 15min change)



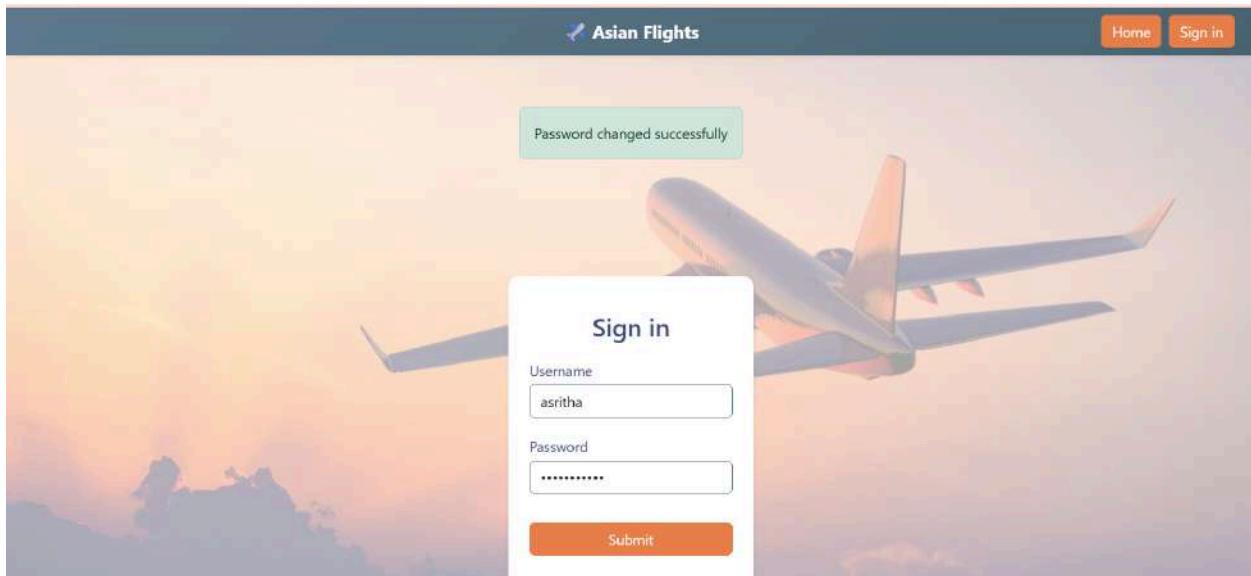
Case when Old password is incorrect:



When Old password and New password are similar:



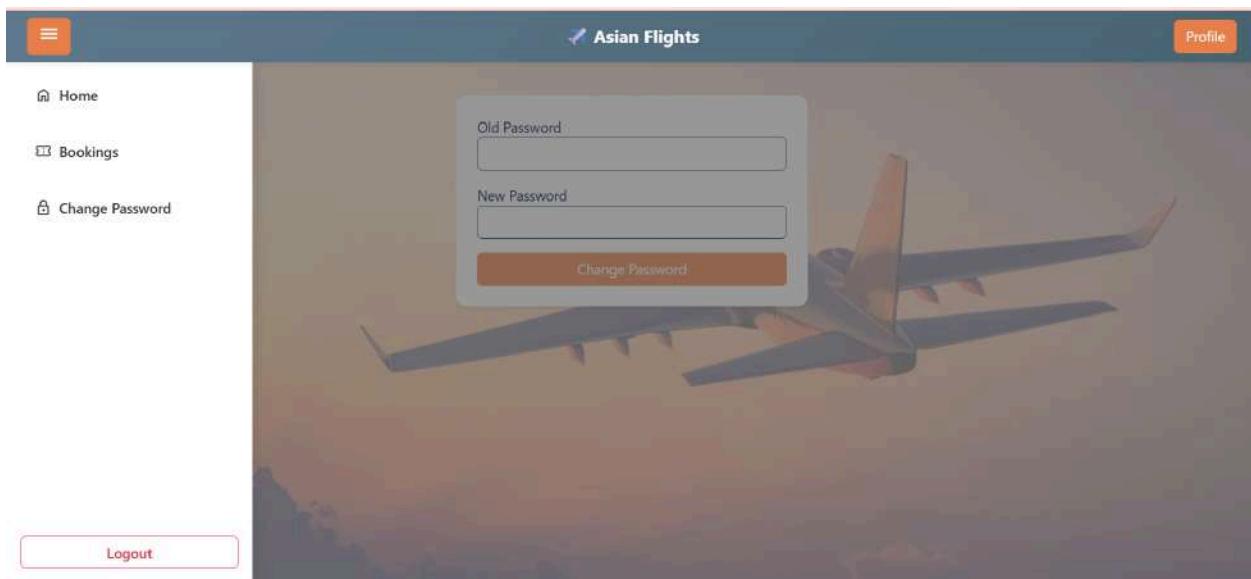
After successful change of password: redirects to sign in again



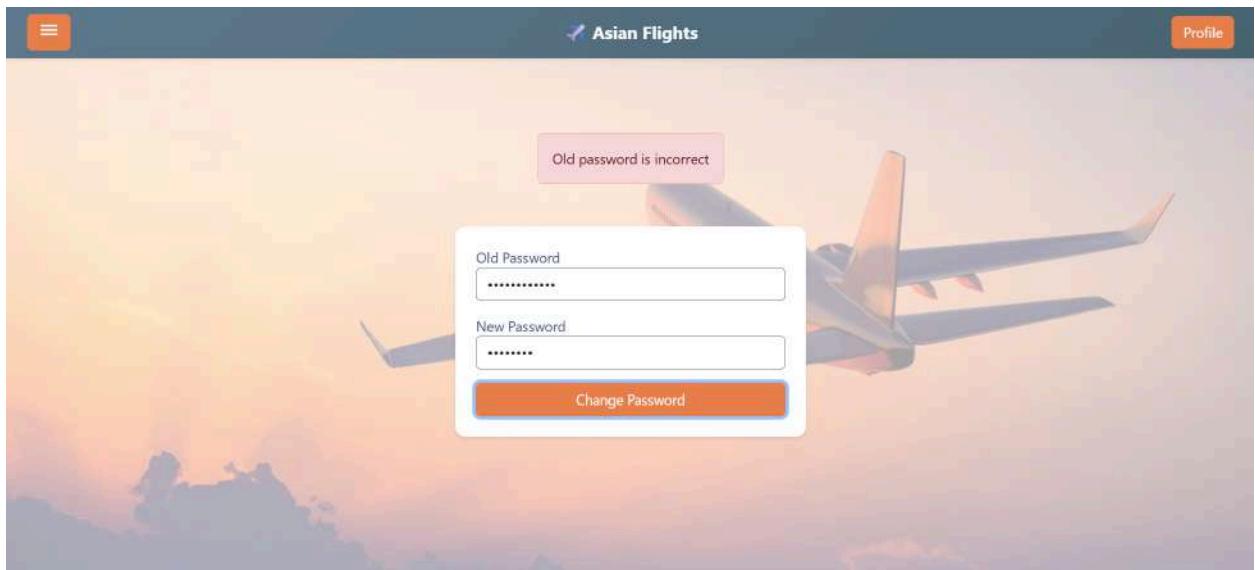
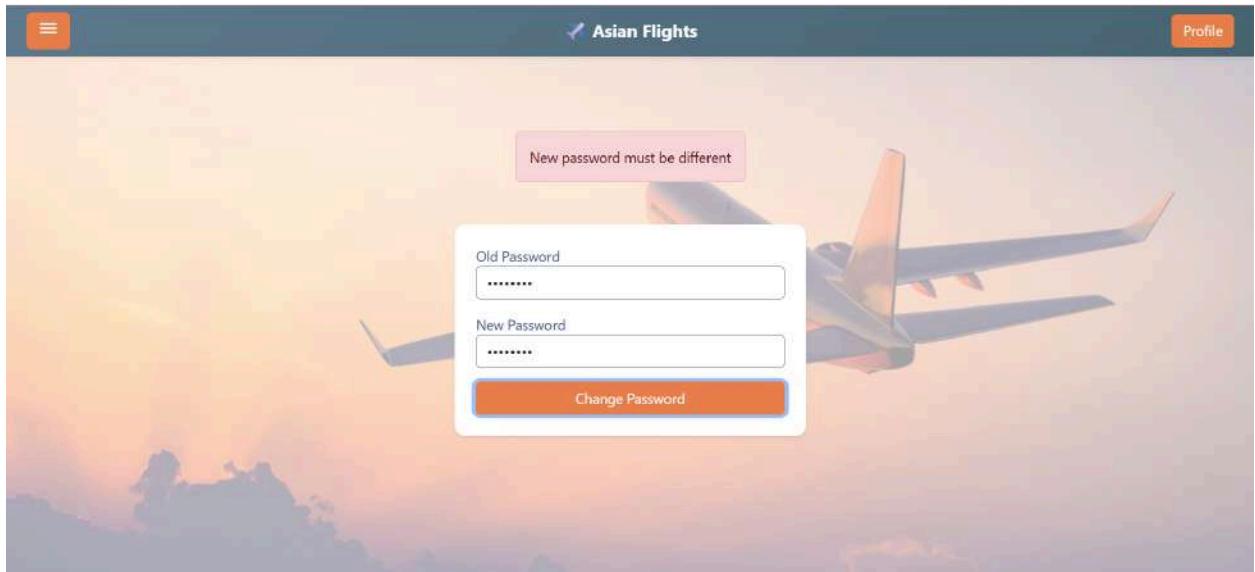
Change password feature if user wishes to change:

This is for admin role and user role both

After sign in

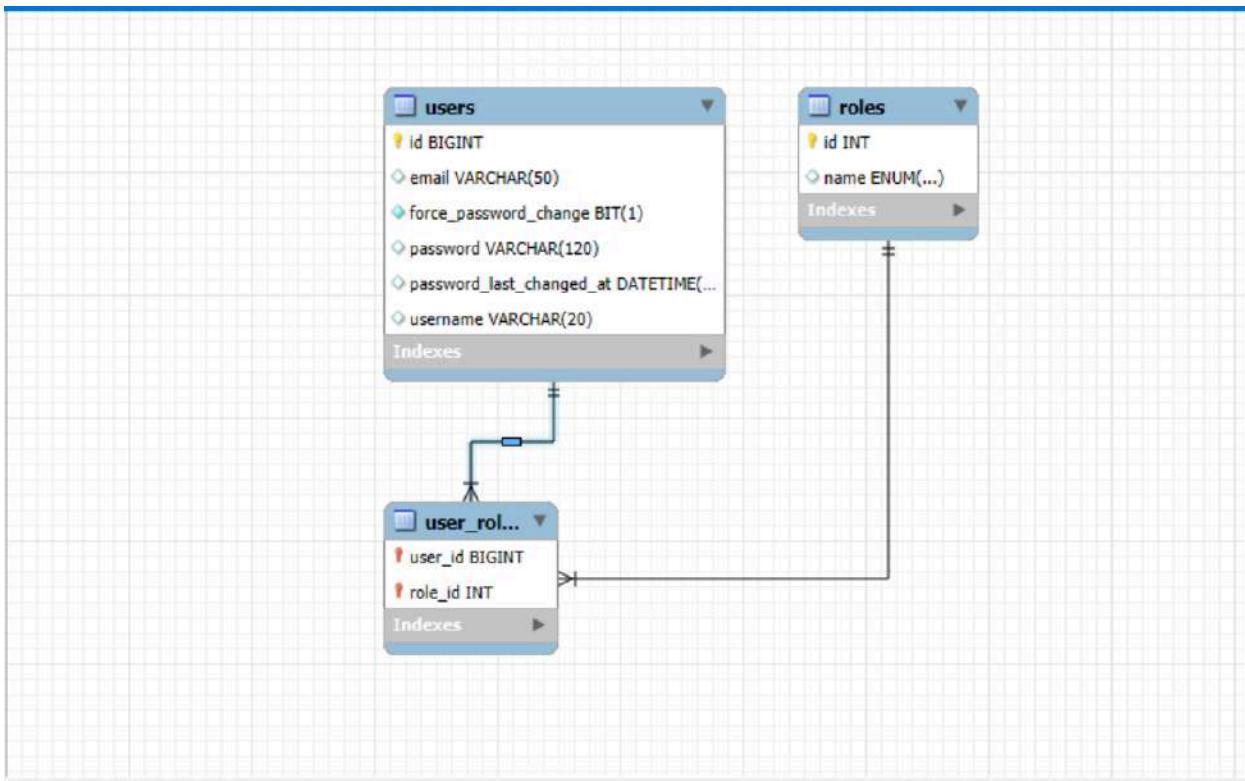


If user wishes to change:



## Schema changes:

User class features added: password\_last\_changed -> LocalDateTime  
And force\_password\_change -> boolean



New api:

<http://localhost:8765/auth-service/api/auth/change-password>  
 Password changed successfully (200 ok)  
 Error: Old password is incorrect (400 Bad Request)

<http://localhost:8765/auth-service/api/auth/signin>  
 Sign in api:  
 After 90 days response  
 {  
 "status": "PASSWORD\_EXPIRED",  
 "message": "Please change your password",  
 "forcePasswordChange": true  
 }

Admin:

Role based access control

.pathMatchers("/flight-service/flight/register").hasRole("ADMIN")

In API GATEWAY

Backend handling:

The screenshot shows a Java-based microservices architecture. On the left, the project structure for 'backend-flight-booking-app [MicroServicesFl]' is displayed, including sub-modules like 'api-gateway' and 'flight-service'. The code editor on the right contains the 'GatewaySecurityConfig.java' file, which defines security configurations for various API endpoints across different services.

```

Project ▾
  backend-flight-booking-app [MicroServicesFl]
    > .idea
    > .metadata
    > api-gateway
      > .mvn
      > .settings
      > src
        > main
          > java
            > com.example.gateway
              > security
                <-- GatewaySecurityConfig (selected)
                <-- JwtAuthFilter
                <-- JwtReactiveAuthentication
                <-- JwtUtil
                <-- ApiGatewayApplication
            > resources
        > test
    > target

src/main/java/com/example/gateway/security/GatewaySecurityConfig.java
  public class GatewaySecurityConfig {
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity http) {
      .authorizeExchange( AuthorizeExchangeSpec ex -> ex
        .pathMatchers(
          "/auth-service/api/auth/me",
          "/auth-service/api/auth/change-password"
        ).authenticated()
        .pathMatchers( ...antPatterns: "/auth-service/api/auth/**").permitAll()
      // FLIGHT SERVICE
      .pathMatchers( ...antPatterns: "/flight-service/flight/register").hasRole("ADMIN")
      .pathMatchers( ...antPatterns: "/flight-service/flight/delete/**").hasRole("ADMIN")
      .pathMatchers( ...antPatterns: "/flight-service/flight/getAllFlights") Access
      .hasAnyRole( ...roles: "ADMIN", "USER" ) AuthorizeExchangeSpec
      .pathMatchers( ...antPatterns: "/flight-service/flight/getFlightById/**") Access
      .hasAnyRole( ...roles: "ADMIN", "USER" ) AuthorizeExchangeSpec
      .pathMatchers( ...antPatterns: "/flight-service/flight/getByOriginDestinationDateTime") Access
      .permitAll() AuthorizeExchangeSpec
      // SEAT MANAGEMENT (Internal but same roles)
      .pathMatchers( ...antPatterns: "/flight-service/flight/flights/*/reserve") Access
      .hasAnyRole( ...roles: "ADMIN", "USER" ) AuthorizeExchangeSpec
    }
  }

```

## Frontend handling:

With the api:

/auth-service/api/auth/me

Response:

```
{
  "id": 5,
  "username": "admin",
  "email": "admin@gmail.com",
  "roles": [
    "ROLE_ADMIN"
  ]
}
```

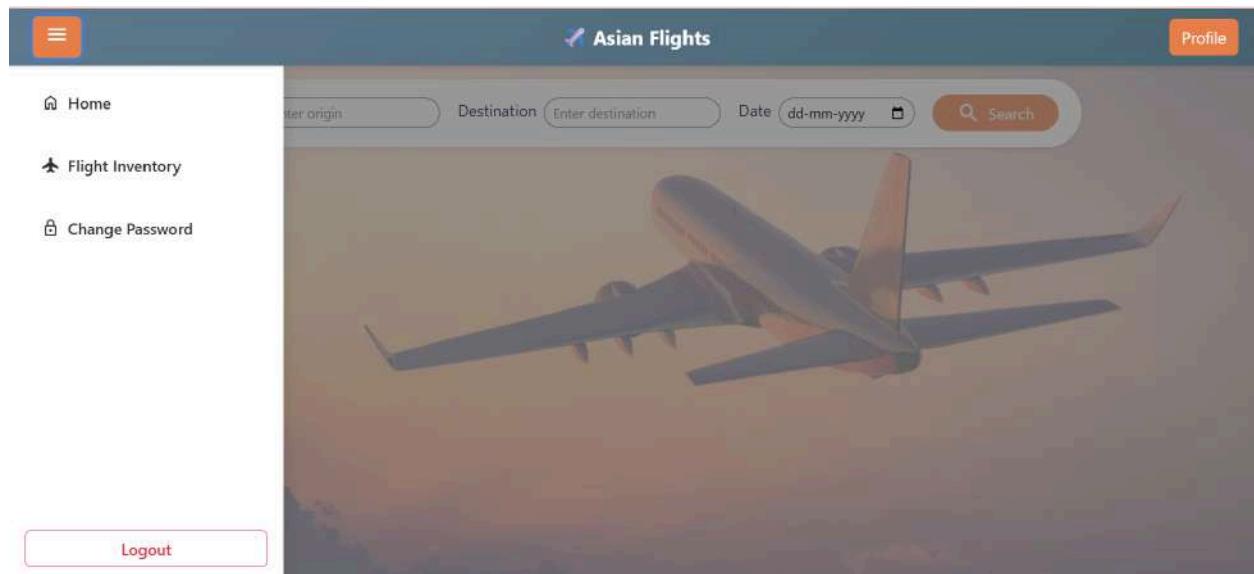
The screenshot shows a portion of a web application's UI. It features a navigation menu with items such as 'Dashboard', 'Add Flight', 'Flight Inventory', and 'Logout'. The 'Flight Inventory' item is currently selected, indicated by a blue background.

```

@if ((user$ | async) === 'admin') {
  <li>
    <a class="nav-link d-flex align-items-center gap-2" routerLink="/addFlights">
      <span class="material-symbols-outlined">flight</span>
      Flight Inventory
    </a>
  </li>
}

```

```
ngOnInit(): void {
  this.user$ = this.currentUser$.pipe(
    map(user =>{
      console.log(user);
      return user?.roles.includes(UserRole.ROLE_ADMIN) ? 'admin' : 'user';
    })
  );
}
```



The screenshot shows a modal dialog box for adding a new flight. The title of the dialog is "Fill in the details to add a new flight". The form contains the following fields:

Airline	Price (₹)		
INDIGO	4500		
Origin	Destination		
Hyderabad	Bangalore		
Departure	Arrival		
23-12-2025	10:30 AM	29-12-2025	12:00 PM
Total Seats			
180			

At the bottom of the dialog is an orange "Add Flight" button with a plus sign icon.

## Validations:

Field / Feature	Purpose	Validations Applied
<b>Airline</b>	Selects the airline for the flight	Required field
<b>Origin</b>	Departure city	Required, only alphabets and spaces allowed (^[A-Za-z ]+\$)
<b>Destination</b>	Arrival city	Required, only alphabets and spaces allowed (^[A-Za-z ]+\$)
<b>Price</b>	Ticket price of the flight	Required, numeric values only (^[0-9]+\$)
<b>Departure Date</b>	Flight departure date	Required
<b>Departure Time</b>	Flight departure time	Required
<b>Arrival Date</b>	Flight arrival date	Required
<b>Arrival Time</b>	Flight arrival time	Required
<b>Total Seats</b>	Total seat capacity of the flight	Required
<b>Min Date Constraint</b>	Prevents selecting past dates	Minimum date set to current date
<b>Flight Registration</b>	Adds a new flight	Form must be valid before submission
<b>Flight Listing</b>	Displays all flights	Auto-refresh using <code>BehaviorSubject</code>
<b>Flight Deletion</b>	Deletes a selected flight	Requires valid flight ID & confirmation
<b>Auto Refresh</b>	Updates UI after add/delete	Uses <code>BehaviorSubject + switchMap</code>

The screenshot shows a flight addition form on the Asian Flights website. The form fields are as follows:

- Airline: INDIGO
- Price (₹): 4500n
- Origin: Hyderabad8
- Destination: Bangalore8
- Departure: 23-12-2025 at 10:30 AM
- Arrival: 29-12-2025 at 12:00 PM
- Total Seats: 180

The "Add Flight" button is visible at the bottom of the form.

Drop down for Different airlines selection

The screenshot shows a dropdown menu for selecting an airline. The options listed are:

- INDIGO
- INDIGO (selected)
- AIRINDIA
- EMIRATES
- SPICEJET

The rest of the flight details (Price, Destination, Arrival, Total Seats) and the "Add Flight" button are identical to the first screenshot.

Flights shown after adding.

The screenshot shows a list of four flight cards. Each card displays the airline (INDIGO), route (goa → chennai), price (₹10,000.00), departure date (25 Dec 2025, 06:30 PM), arrival date (26 Dec 2025, 08:45 PM), seat availability (Seats: 180), and a delete button. The cards are arranged in a grid-like layout.

Flight Details	Price
INDIGO goa → chennai Departure: 25 Dec 2025, 06:30 PM   Arrival: 26 Dec 2025, 08:45 PM Seats: 180	₹10,000.00
INDIGO goa → chennai Departure: 25 Dec 2025, 06:30 PM   Arrival: 26 Dec 2025, 08:45 PM Seats: 180	₹10,000.00
INDIGO goa → chennai Departure: 20 Dec 2025, 06:30 PM   Arrival: 20 Dec 2025, 08:45 PM Seats: 147	₹10,000.00
INDIGO goa → chennai Departure: 25 Dec 2025, 06:30 PM   Arrival: 26 Dec 2025, 08:45 PM Seats: 180	₹10,000.00

Modal card to ask for cancellation

The screenshot shows the same list of flights as above, but with a modal dialog box centered over the third flight card. The dialog is titled "Confirm Deletion" and contains the message "Do you really want to delete this flight? This action cannot be undone." It features two buttons: "Cancel" and "Delete". The background flights are partially visible behind the modal.

## Optimized Docker file:

Example for flight service:

```
FROM eclipse-temurin:17-jdk
WORKDIR /app
COPY target/*.jar flight-service.jar
EXPOSE 9002
ENTRYPOINT ["java", "-jar", "flight-service.jar"]
```

## Star-service.cmd file:

```
java -jar service-registry\target\service-registry-0.0.1-SNAPSHOT.jar
java -jar ConfigServer\target\ConfigServer-0.0.1-SNAPSHOT.jar
java -jar api-gateway\target\api-gateway-0.0.1-SNAPSHOT.jar
java -jar auth-service\target\spring-security-own-0.0.1-SNAPSHOT.jar
java -jar flight-service\target\flight-service-0.0.1-SNAPSHOT.jar
java -jar passenger-service\target\passenger-service-0.0.1-SNAPSHOT.jar
java -jar ticket-service\target\ticket-service-0.0.1-SNAPSHOT.jar
java -jar email-service\target\email-service-0.0.1-SNAPSHOT.jar
```

## 2 Property files:

1. For Docker
2. Local

```
application.properties
spring.application.name=flight-service
server.port = 9002
logging.level.root=INFO
logging.level.com.example=DEBUG
#these properties are moved to config server
#spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
#spring.datasource.username=postgres
#spring.datasource.password=password
#
#spring.jpa.hibernate.ddl-auto=update
#spring.jpa.hibernate.show-sql=true
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
eureka.instance.prefer-ip-address=true
spring.config.import=optional:configserver:http://configserver:8888/config/application
```

```
application-docker.properties
spring.application.name=flight-service
server.port=9002
spring.config.import=optional:configserver:http://configserver:8888/config/application
eureka.client.service-url.defaultZone=http://eureka:8761/eureka
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
```

application-docker.properties file for docker purpose

In docker-compose.yml file this is how we use the docker profile:

```
flight-service:
  build: ./flight-service
  container_name: flight-service
  ports:
    - "9002:9002"
  environment:
    SPRING_PROFILES_ACTIVE: docker
  depends_on:
    eureka-server:
      condition: service_healthy
    config-server:
      condition: service_healthy
    postgres-flight:
      condition: service_started
  networks:
    - app-net
```

# SWOT Analysis – Flight Ticket Booking Application

## Strengths

- Microservices-based architecture (Eureka, Config Server, API Gateway)
- Secure authentication using **Spring Security + JWT**
- Clear separation of services (Auth, Flight, Passenger, Ticket, Email)
- Admin features for adding and managing flights

## Weaknesses

- Limited frontend features and UI polish
- No real payment gateway integration
- Basic error handling and validation
- High dependency on multiple services running together

## Opportunities

- Integration of payment gateways (Razorpay/Stripe)
- Seat locking and dynamic pricing
- Real-time notifications using WebSockets
- Docker + Kubernetes deployment
- Advanced search, filters, and recommendations

## Threats

- System failure if a critical microservice goes down
- Security risks if JWT handling is misconfigured

- Performance issues under high traffic
- Data inconsistency in distributed services