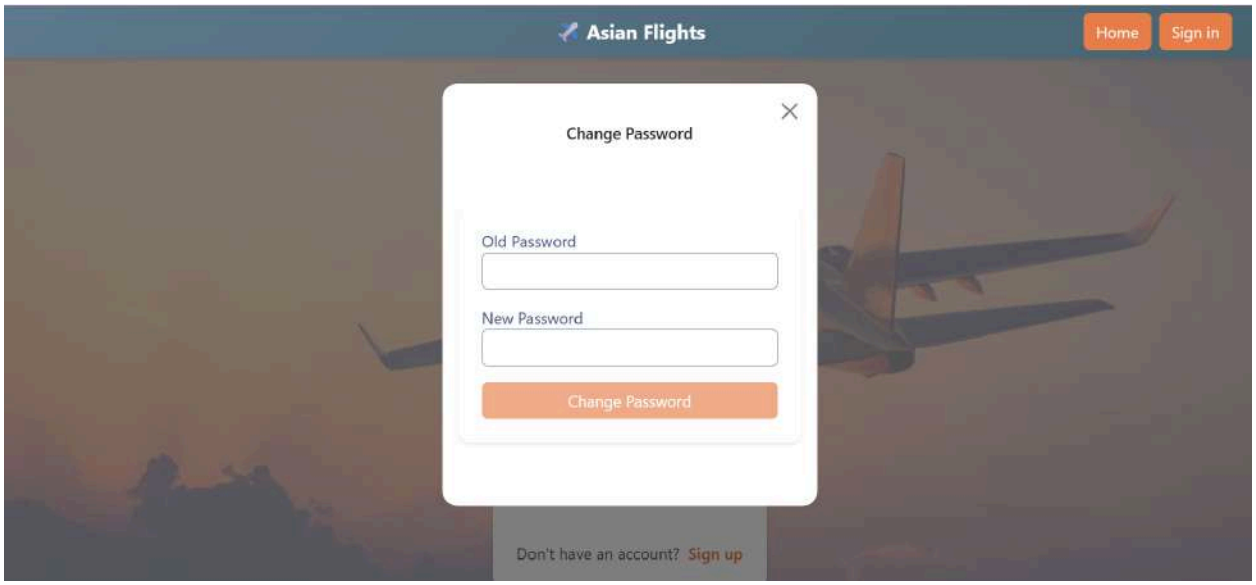
Repositories links:

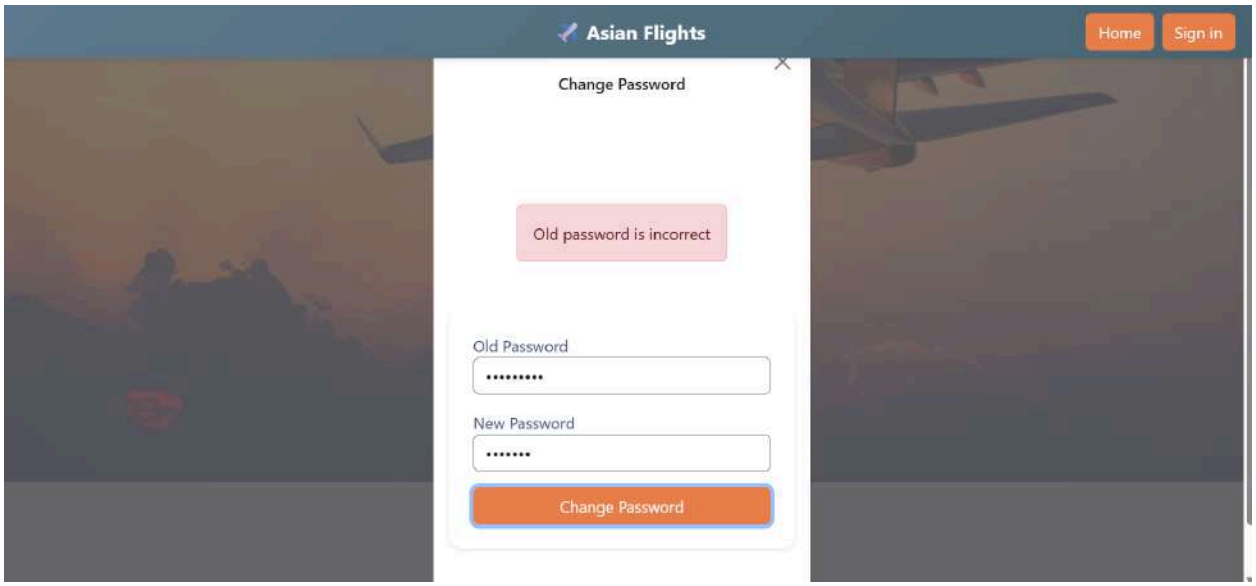
- Frontend: https://github.com/asritha26k/frontend-for-flight-booking-app
- Backend: https://github.com/asritha26k/backend-flight-booking-app

Change password initiated after every 90 days:

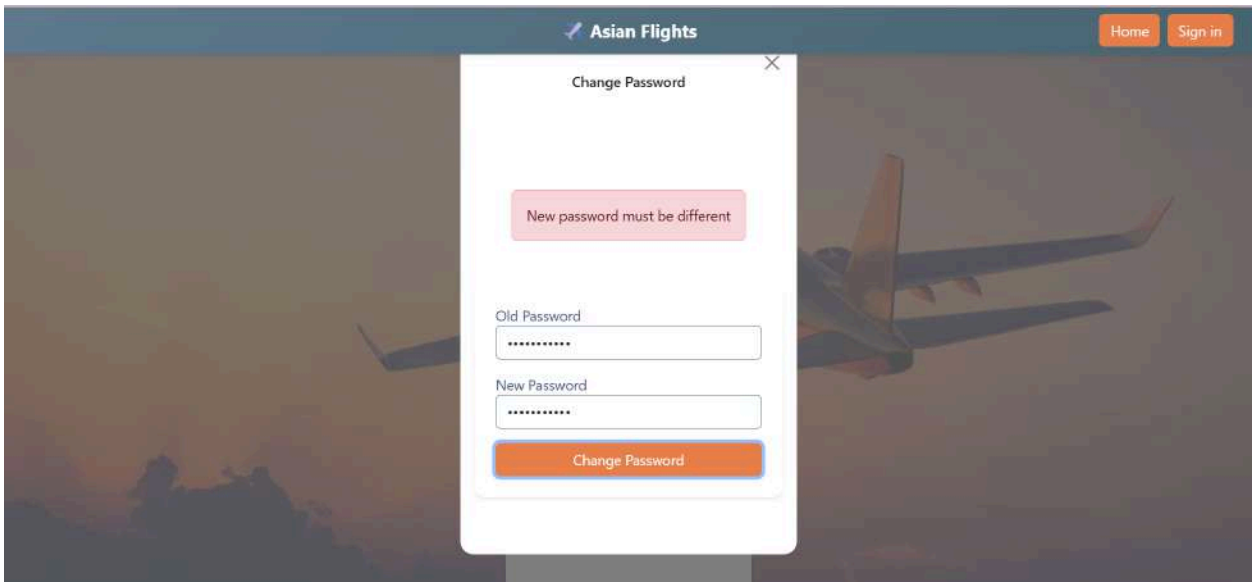
(Tested with 15min change)



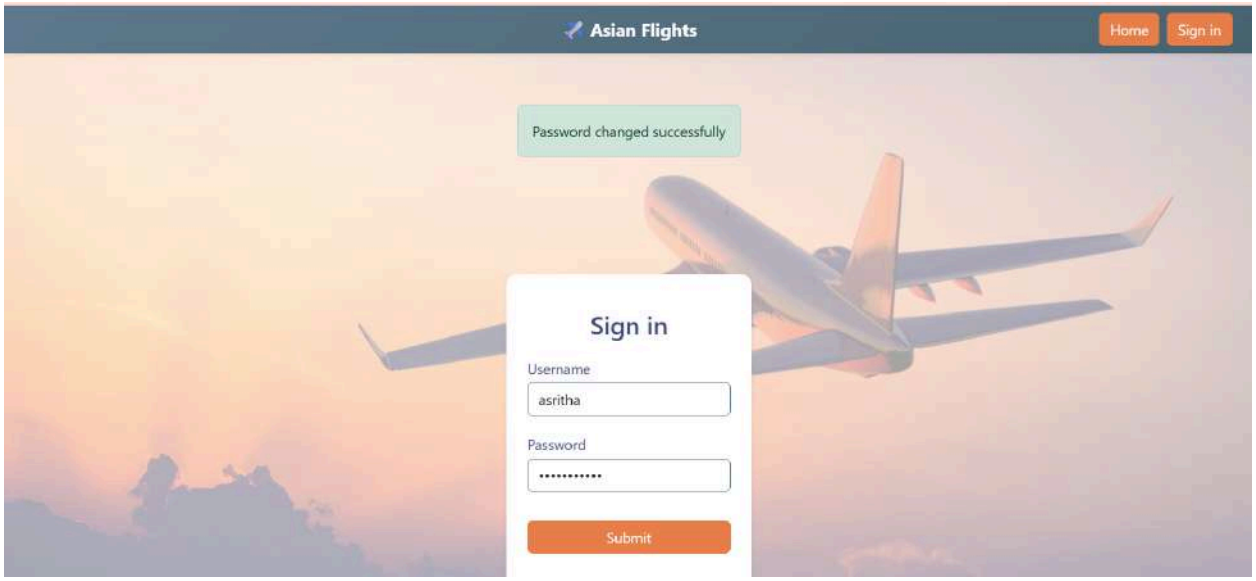
Case when Old password is incorrect:



When Old password and New password are similar:



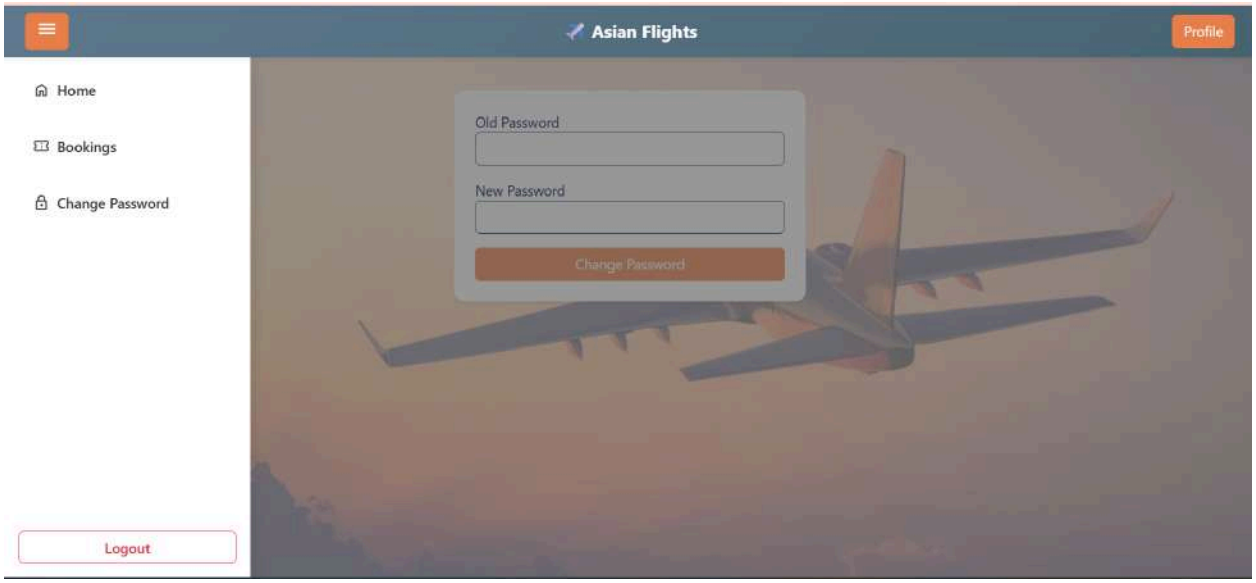
After successful change of password: redirects to sign in again



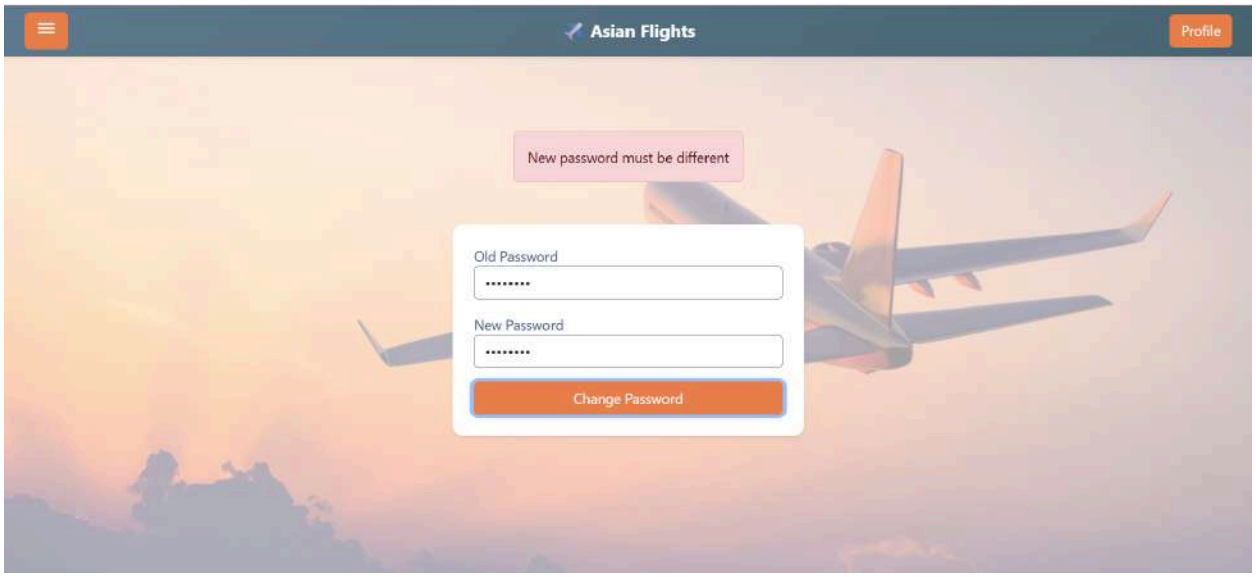
Change password feature if user wishes to change:

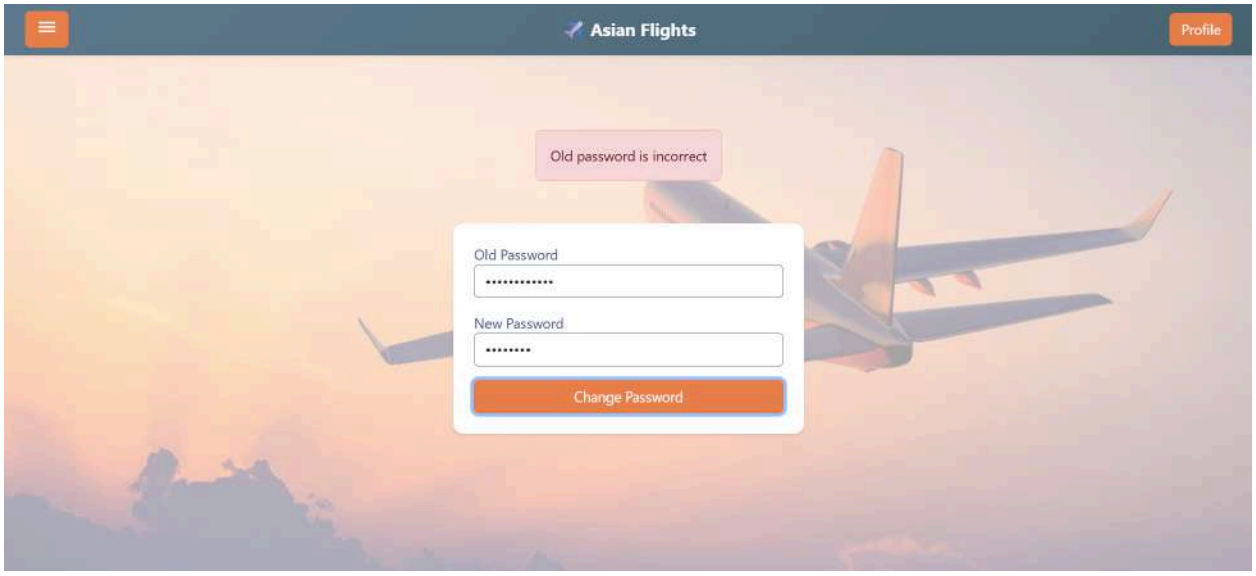
This is for admin role and user role both

After sign in



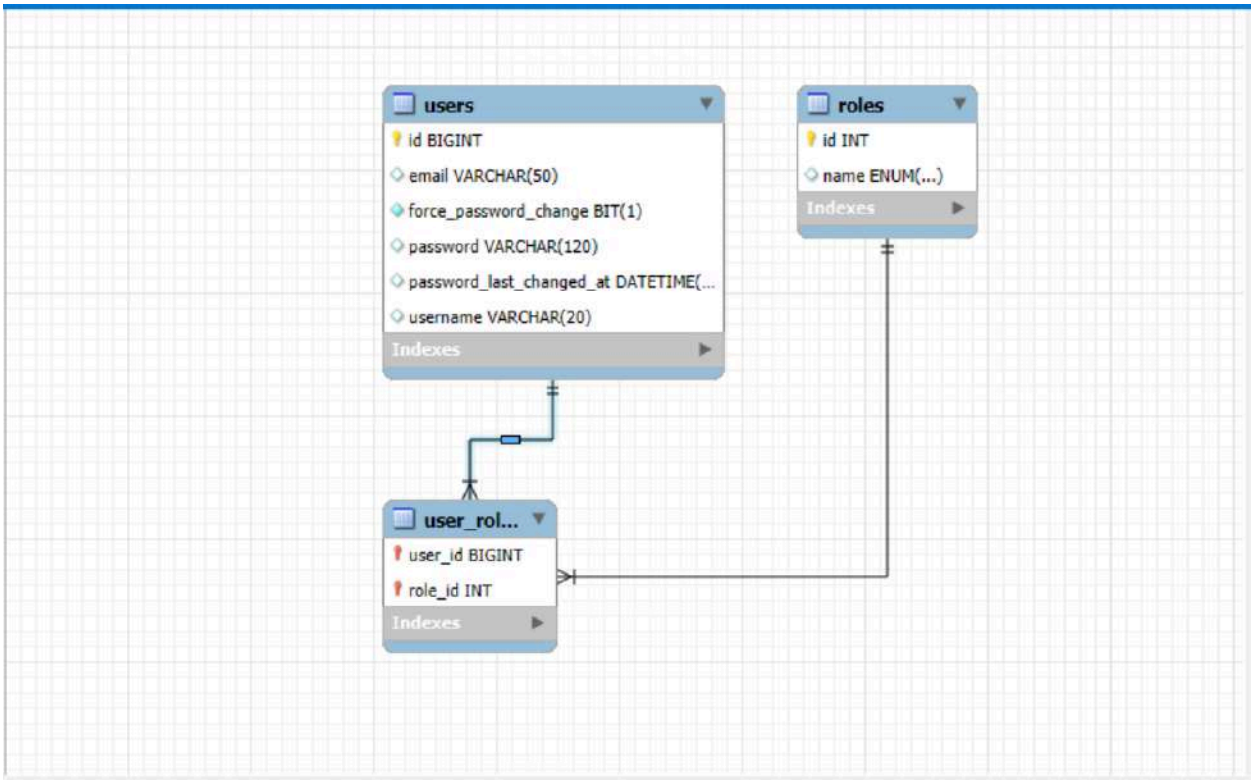
If user wishes to change:





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

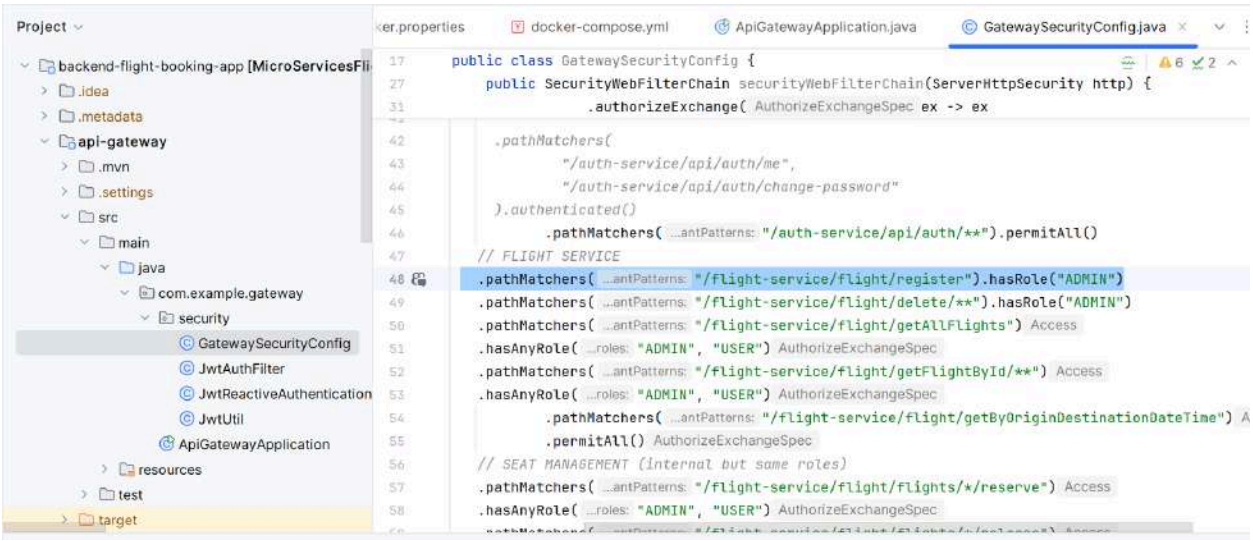
```
{
   "status": "PASSWORD_EXPIRED",
   "message": "Please change your password",
   "forcePasswordChange": true
}
```

Admin:

Role based access control

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

In API GATEWAY
Backend handling:



Frontend handling:

With the api:

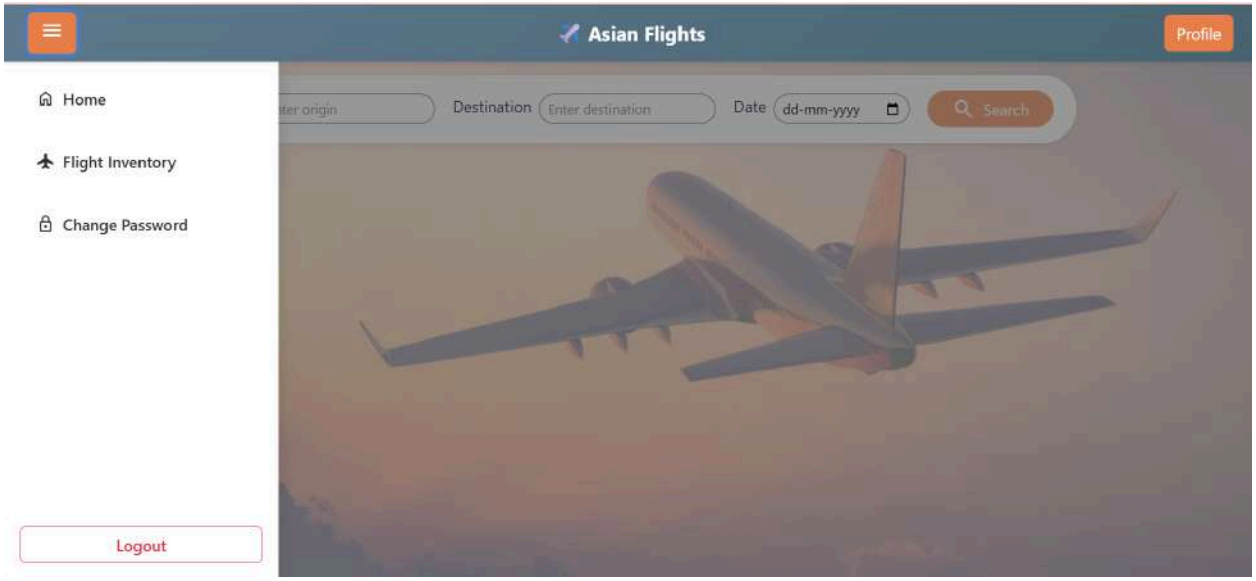
/auth-service/api/auth/me

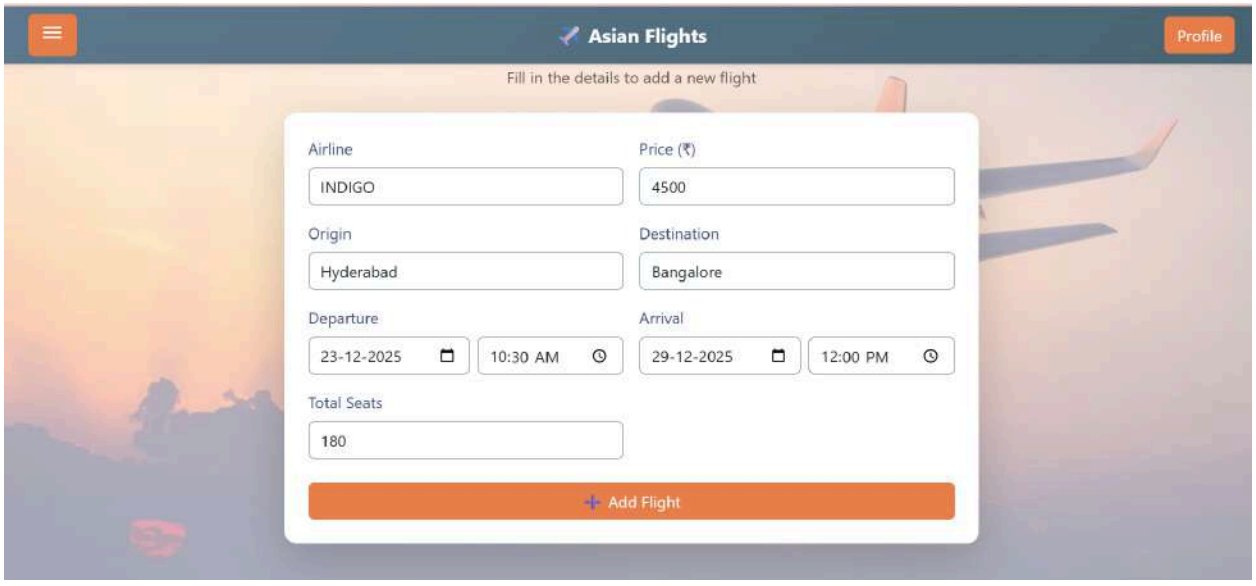
Response:


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


```
@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>
}
```

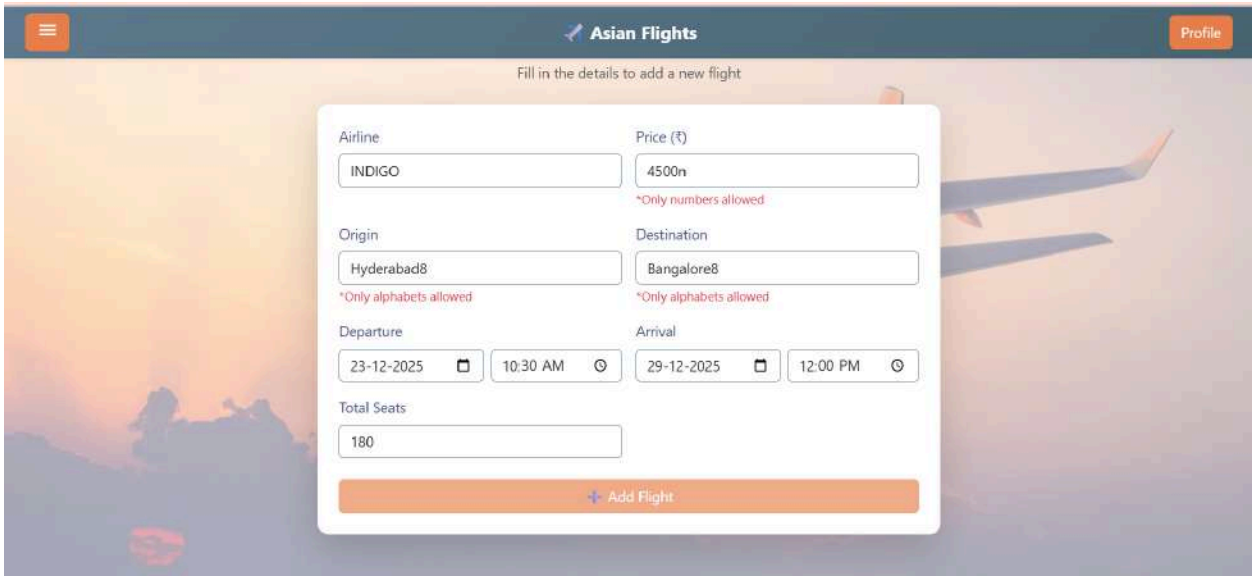
```
ngOnInit(): void {
  this.user$ = this.currentUser$.pipe(
    map(user =>{
      console.log(user);
return user?.roles.includes(UserRole.ROLE_ADMIN) ? 'admin' : 'user';
    }

    )
  );
}
```



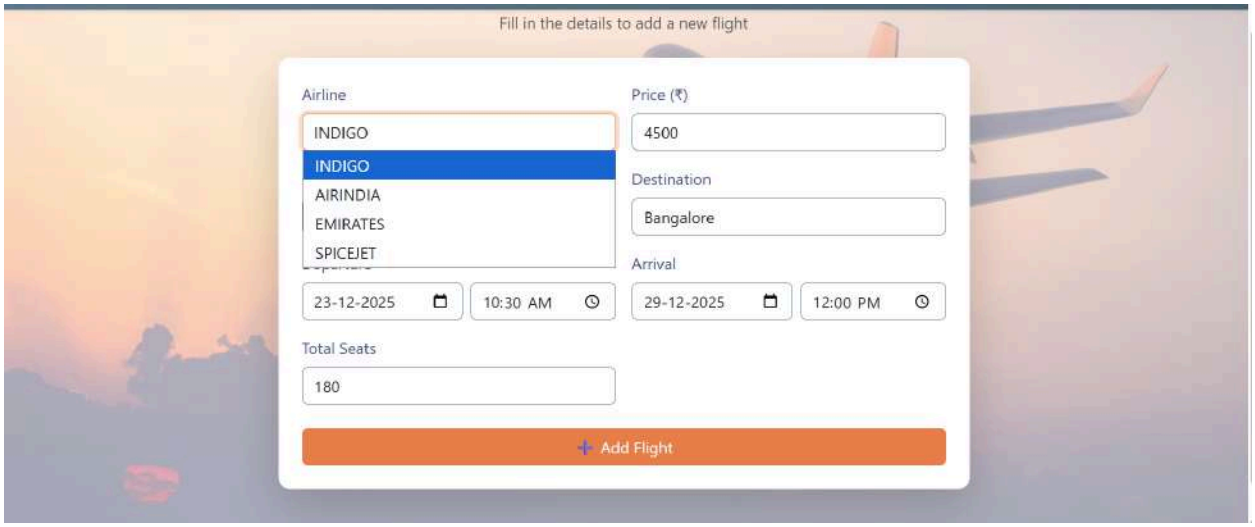


Validations:

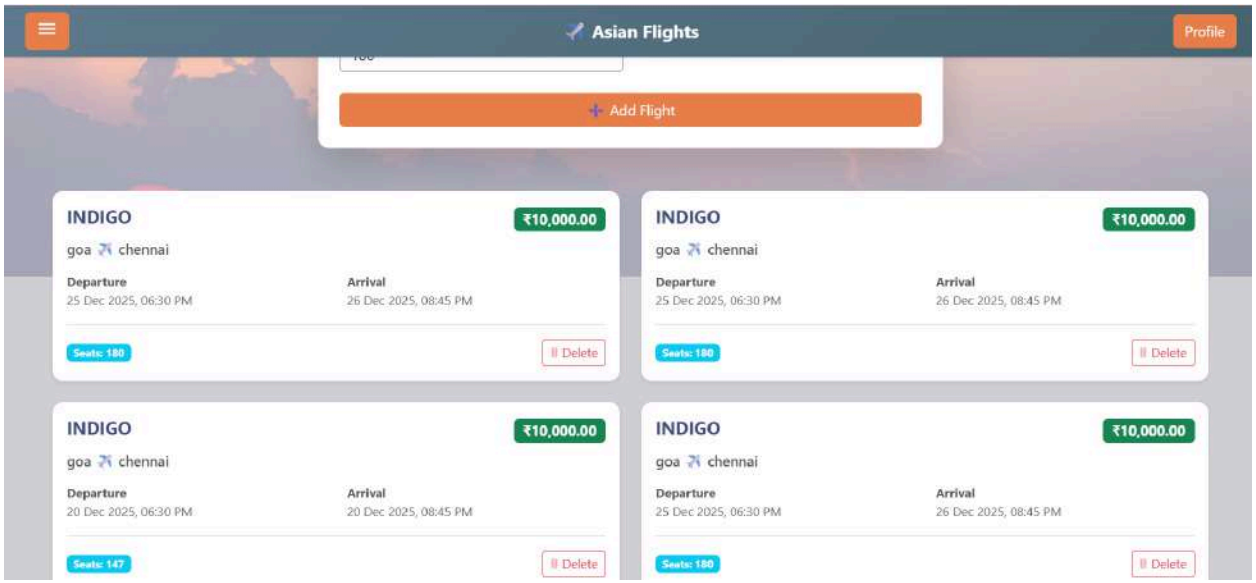
Field / Feature	Purpose	Validations Applied
Airline	Selects the airline for the flight	Required field
Origin	Departure city	Required, only alphabets and spaces allowed (`^[A-Za-z ]+$`)
Destination	Arrival city	Required, only alphabets and spaces allowed (`^[A-Za-z ]+$`)
Price	Ticket price of the flight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



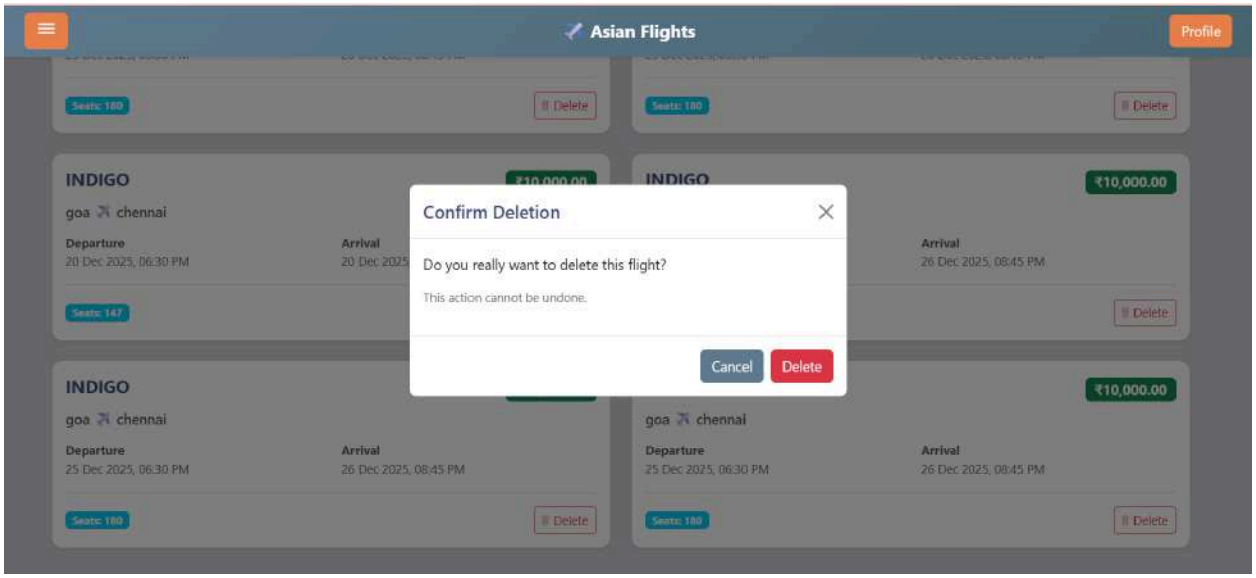
Drop down for Different airlines selection



Flights shown after adding.



Modal card to ask for cancellation



Optimized Docker file:

Example for flight service:

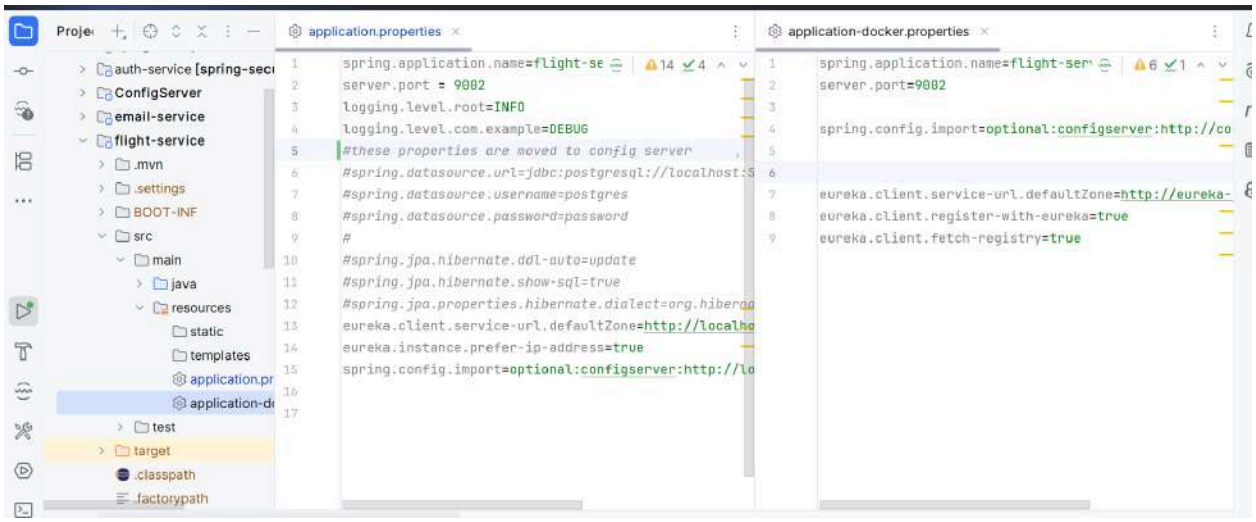
```
FROM eclipse-temurin:17-jdk

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-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