# Java Backend – Advanced

## Air Ticket Reservation System – Case Study

The application should have the functionalities related to Air Ticket Reservation.

Broad Functionalities:

a. UserManagement – Two types of users exist in the system, Admin and Passenger
b. FlightManagement – For managing flights information

Passenger Related Functionalities:

1. Register – For new Passengers.
2. Login Functionality – For registered passengers
3. View Flight schedule
4. Book flight - A reservation form to be filled by passengers giving details like name, address, date of journey, destination and boarding airport, purpose of journey and mode of payment etc. Depending on the availability of the seats, booking will be confirmed. If seats are not available, the passenger is informed accordingly.
5. View Reservation status
6. Provision for the passenger to upgrade booking and cancel booking

Admin Related Functionalities:

1. Login Functionalities
2. Create Admin user – to create new admins
3. Adding flight details. Use form to fill flight details. On submitting the form, the flight details are to added to the database.
4. Showing availability of seats based on flight, date, source, destination
5. Generating Reports – based on date, destination

Functionalities mentioned above are just indicative. You can think about the functionalities of any real airline ticket booking system and try to make your solution as near as possible to the real system.

Design the application using Microservices architecture.

Apart from implementing the Microservices based on the functionalities discussed above,

a. Use Eureka as your Registry and Discovery service so that Microservices can find/discover each other and interact with each other.
b. Implement Client-side load balancer wherever applicable (Spring Cloud Load Balancer)

c. Identify some important methods that can lead to cascading failures and guard them by implementing Software Circuit Breaker/Fault Tolerance mechanism (Spring Cloud Circuit Breaker with Resilience4J and optionally can also setup a dashboard like Prometheus)
**d.** Use Thymeleaf based dynamic views wherever needed (do not use JSPs)
**e.** Create an external configuration server (file/github) and applications should get the configuration information from the configuration server.
f. Enable distributed log tracing with micrometer brave and configure the zipkin dashboard to monitor and trace the logs.

**Assumptions:**

a. Load the application with predefined set of admins and passengers.
b. The choice of how many number of Microservices you want to implement for the given scenario is left to you. Keep in mind – Nano services are anti-patterns.
c. You can use some predefined set of flights and also admin can add additional flights.
d. Implement authentication to secure the endpoints. Ensure appropriate validations and error messages are implemented to have better end user experience.
e. You can design and implement the user-interface as per your choice.

**Other Details:**

a. You can use any IDE of your choice (STS/IntelliJ/Eclipse) etc.,
b. You can use any in-memory database like H2/HSQL as your data store.
c. Try to use good programming constructs wherever applicable.