



## **Week 6 - Project Report**

### **By : Kanchan Rai**

## **Topic : Microservices Flight Booking System w/ We Flux**

The following document contains the description and explanation of the project **Microservices Based FlightBookingSystem** made with **WebFlux using MongoDB**. This project has all the required validations for all end points, exception handling, unit test coverage of. Load testing carried out as well for all endpoints. The database used is MongoDB. And Reactive Programming has been carried out as well.

# **INDEX**

## **1. Project Over View**

1.1 System Architecture

1.2 Eureka Registering

## **2. JCoco Coverage Report**

## **3. SonarQube Report**

## **4. JMeter Load Testing**

4.1 With 20 Threads

4.2 With 50 Threads

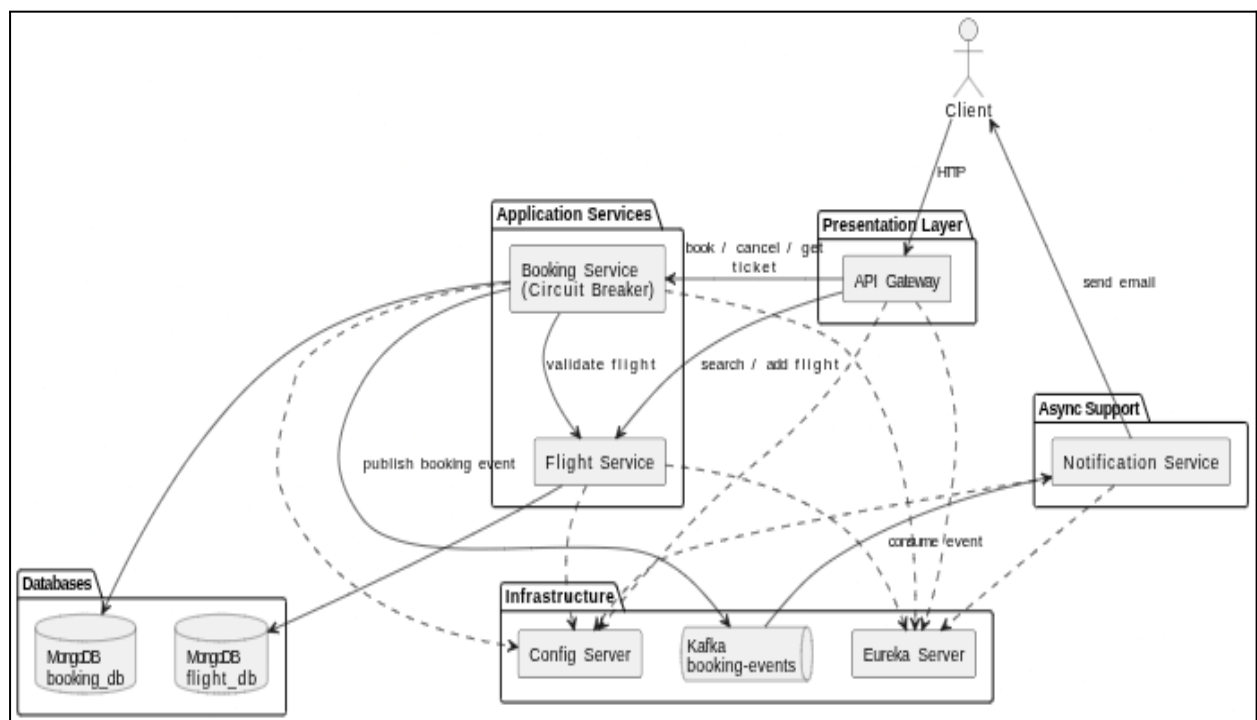
4.3 With 100 Threads

## **5. All API Endpoints Testing & Results**

# 1. Project Overview

This project was a monolithic application with one single db called the “flightBookingSystem” in MongoDB, based on the reactive web flux in spring boot. The project has now been converted to a microservices architecture based project. There are two microservices namely “booking-service” and “flight-service”. These services are connected by a common api-gateway and a configuration server. In between these services proper load balancing has been applied along with the circuit breaker in between to ensure that all services stay up and running. These services have been registered on the netflix eureka server. Along with this apache-kafka message broker technique has also been applied to send notifications whenever a flight is booked or cancelled. Proper load testing using Apache JMeter and JUnit testing has been done.

## 1.1. System Architecture



## 1.2 Eureka Registering

spring Eureka

HOME LAST 1000 SINCE STARTUP

Custo

System Status

Environmenttest

Data centerdefault

Current time2025-12-01T22:49:44 +0530

Uptime00:05

Lease expiration enabledfalse

Renews threshold8

Renews (last min)8

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.1.6:api-gateway:9000
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.6:booking-service:8080
CONFIG-SERVER	n/a (1)	(1)	UP (1) - 192.168.1.6:config-server:8888
FLIGHT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.6:flight-service:8090

General Info

Name	Value
total-avail-memory	90mb
num-of-cpus	12

This is the Eureka Server where all four services are successfully registered, namely flight-service, booking-service, config-server, and api-gateway.









This project also implements Message Broker using Apache Kafka which is used whenever a flight is booked or cancelled in the booking service and a mail is sent to the passenger.

Kafka was used with docker the commands are as follows -









```
docker run -d \
  --name kafka \
  -p 9092:9092 \
  -p 9093:9093 \
  -e CLUSTER_ID=$CLUSTER_ID \
  -e KAFKA_NODE_ID=1 \
  -e KAFKA_PROCESS_ROLES=broker,controller \
  -e KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092,CONTROLLER://0.0.0.0:9093 \
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092 \
  -e KAFKA_CONTROLLER_LISTENER_NAMES=CONTROLLER \
  -e KAFKA_CONTROLLER_QUORUM_VOTERS=1@localhost:9093 \
  -e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 \
  -e KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR=1 \
  -e KAFKA_TRANSACTION_STATE_LOG_MIN_ISR=1 \
  confluentinc/cp-kafka:7.5.0
```

## 2. JCoco Coverage Report -

Coverage Report for FlightService - 96%

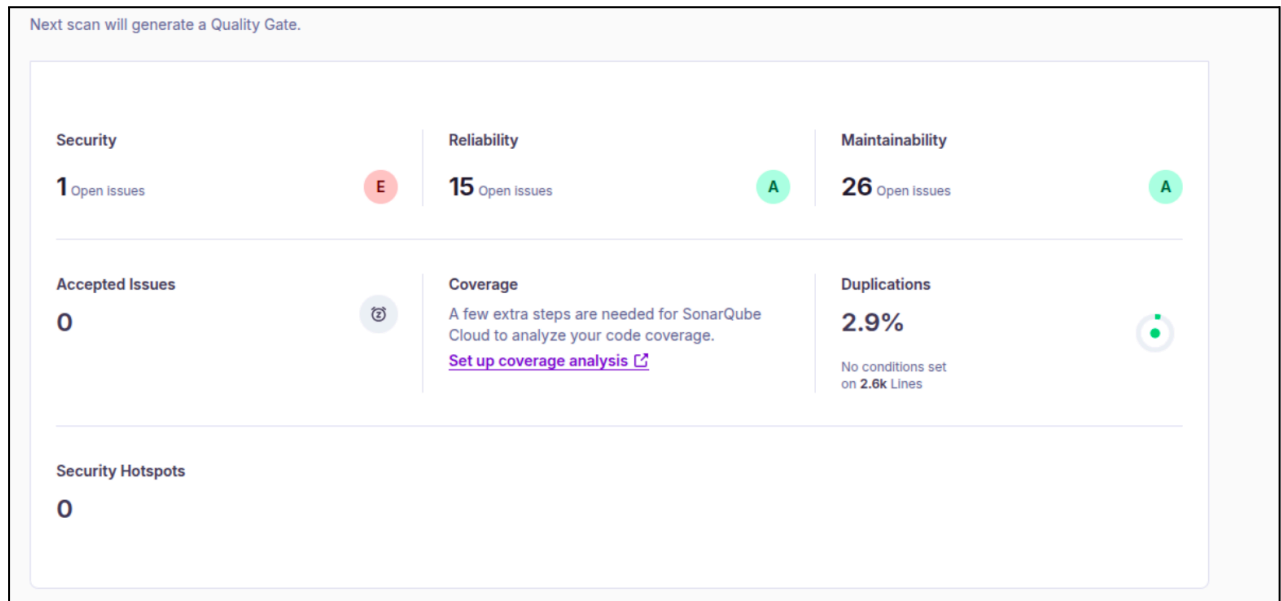
FlightService												
<b>FlightService</b>												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">com.flightservice.service</a>		96%		83%	10	46	2	100	1	19	0	3
<a href="#">com.flightservice.controller</a>		70%		n/a	2	7	2	7	2	7	0	1
<a href="#">com.flightservice</a>		37%		n/a	1	2	2	3	1	2	0	1
<a href="#">com.flightservice.request</a>		100%		n/a	0	39	0	57	0	39	0	3
<a href="#">com.flightservice.exceptions</a>		100%		90%	1	14	0	30	0	9	0	3
<a href="#">com.flightservice.model</a>		100%		n/a	0	2	0	18	0	2	0	2
Total	32 of 867	96%	10 of 64	84%	14	110	6	215	4	78	0	13

Coverage Report for BookingService - 92%

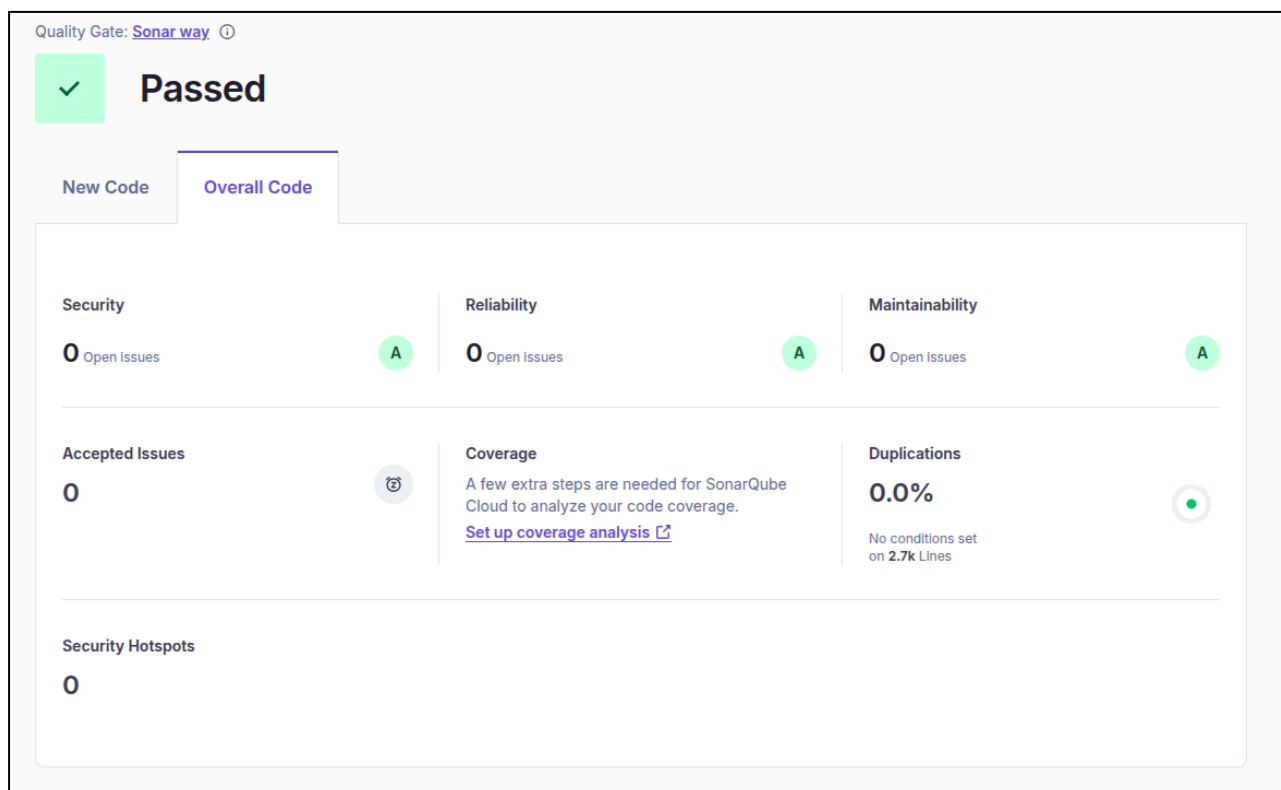
BookingService												
<b>BookingService</b>												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">com.booking.service.service</a>		91%		82%	16	67	10	184	6	38	0	4
<a href="#">com.booking.service.model</a>		86%		n/a	10	30	10	53	10	30	0	6
<a href="#">com.booking.service.requests</a>		84%		n/a	4	24	5	35	4	24	0	2
<a href="#">com.booking.service</a>		37%		n/a	1	2	2	3	1	2	0	1
<a href="#">com.booking.service.exceptions</a>		100%		92%	1	16	0	42	0	9	0	3
<a href="#">com.booking.service.client</a>		100%		n/a	0	15	0	42	0	15	0	3
<a href="#">com.booking.service.controller</a>		100%		n/a	0	5	0	5	0	5	0	1
Total	110 of 1,384	92%	11 of 72	84%	32	159	27	364	21	123	0	20

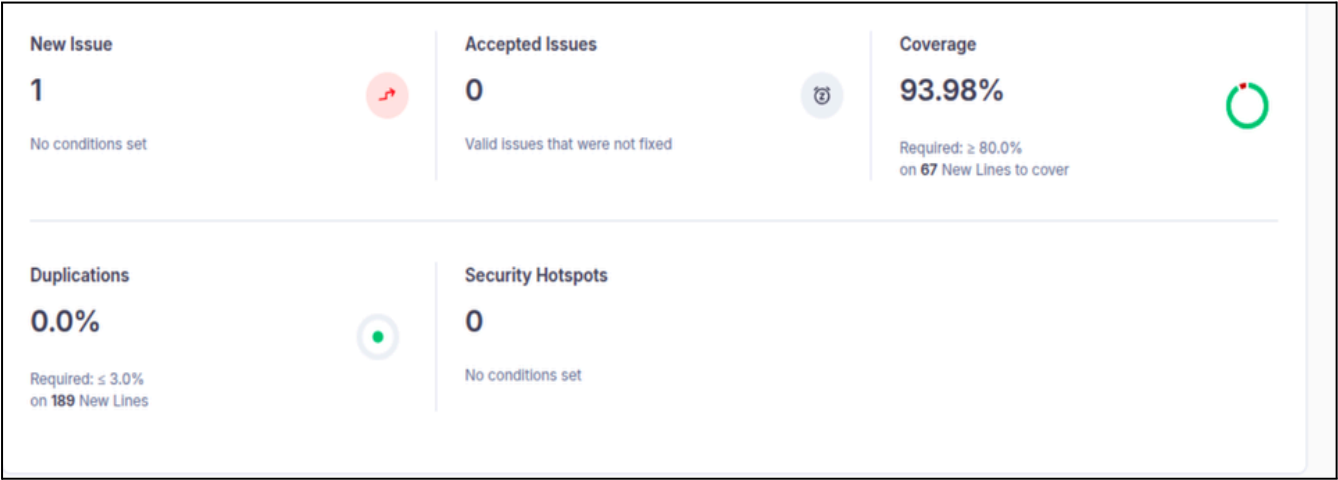
### 3. SonarQube Report

#### 1. Before Fixing



Before fixing the project had 1 security issue, 15 reliability, 26 open issues and 2.9% code duplication. After fixing everything was reduced to 0.





## 4. JMeter Load Testing

### 4.1 With 20 Threads

The JMeter testing for 20 threads across the get,post, and delete requests have been carried out successfully through CLI as well as GUI.

Command used and result stored in a csv file -

```
jmeter -n -t 'LoadTesting20 .jmx' -l result20.csv
```

```
dreamdust@kanchan-pc:~/Chubb - Weekly Assignments/MicroServicesFlightBooking/MicroServicesFlightBooking$ jmeter -n -t 'LoadTesting20 .jmx' -l results20.csv
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using LoadTesting20 .jmx
Starting standalone test @ 2025 Dec 2 02:08:34 IST (1764621514274)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 1 in 00:00:00 = 2.9/s Avg: 108 Min: 108 Max: 108 Err: 0 (0.00%) Active: 7 Started: 7 Finished: 0
summary + 119 in 00:00:01 = 146.0/s Avg: 33 Min: 1 Max: 109 Err: 40 (33.61%) Active: 0 Started: 20 Finished: 20
summary = 120 in 00:00:01 = 103.1/s Avg: 34 Min: 1 Max: 109 Err: 40 (33.33%)
Tidying up ... @ 2025 Dec 2 02:08:35 IST (1764621515599)
... end of run
dreamdust@kanchan-pc:~/Chubb - Weekly Assignments/MicroServicesFlightBooking/MicroServicesFlightBooking$
```

Result CSV-

timeStamp	elapsed	label	responseCode	responseMessage
1764621514660	108	Get All Flights		200 OK
1764621514660	108	Get All Flights		200 OK
1764621514715	53	Get All Flights		200 OK
1764621514660	108	Get All Flights		200 OK
1764621514660	109	Get All Flights		200 OK
1764621514664	104	Get All Flights		200 OK
1764621514763	33	Get All Flights		200 OK
1764621514778	35	Get All Airlines		200 OK
1764621514778	36	Get All Airlines		200 OK
1764621514778	43	Get All Airlines		200 OK
1764621514778	44	Get All Airlines		200 OK
1764621514782	40	Get All Airlines		200 OK
1764621514778	43	Get All Airlines		200 OK

### 4.2 With 50 Threads

The JMeter testing for 50 threads across the get,post, and delete requests have been carried out successfully through CLI as well as GUI.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get All Flights	50	568	245	1134	195.83	0.00%	26.8/sec	69.49	5.45	2654.3
Get All Airlines	50	622	248	1397	246.03	0.00%	31.0/sec	80.24	6.29	2653.8
Post Add a flight	50	10	1	41	10.26	100.00%	39.4/sec	84.92	0.00	2208.7
Post Search a ...	50	11	1	44	12.41	100.00%	40.7/sec	87.45	0.00	2203.0
Post Book a fl...	50	295	56	702	204.29	0.00%	38.7/sec	8.47	17.50	224.0
Del Cancel a b...	50	231	46	650	149.71	0.00%	43.3/sec	4.22	9.00	100.0
TOTAL	300	290	1	1397	291.80	33.33%	124.1/sec	202.91	22.06	1674.0

### 4.3 With 100 Threads

```
jmeter -n -t 'LoadTesting100.jmx' -l result100.csv
```

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get All Flights	200	553	203	806	139.40	0.00%	151.2/sec	391.82	30.71	2654.1
Get All Airlines	200	667	215	1081	213.94	0.00%	150.0/sec	389.13	30.48	2655.8
Post Book a fli...	200	283	20	738	190.77	0.00%	103.2/sec	22.57	46.66	224.0
Del Cancel a b...	200	157	47	668	127.68	0.00%	107.0/sec	10.45	22.26	100.0
TOTAL	800	415	20	1081	266.72	0.00%	249.4/sec	343.01	66.48	1408.5

## 5. API Gateway End Points

### API Gateway on port 9000

#### 1. POST Add an airline

The screenshot shows the API Gateway interface for the endpoint `http://localhost:9000/flight/api/flight/addAirline`. The method is **POST**. The **Body** tab is selected, showing a JSON payload:

```
1 {
2   "airlineCode": "XYZ",
3   "airlineName": "XYZ Airlines"
4 }
```

Below the body, the **JSON** tab is selected, displaying the same payload in a formatted view:

```
1 {
2   "airlineCode": "XYZ",
3   "airlineName": "XYZ Airlines"
4 }
```

#### 2. GET All Airlines

The screenshot shows the API Gateway interface for the endpoint `http://localhost:9000/flight/api/flight/getAllFlights`. The method is **GET**. The **Body** tab is selected, showing a JSON array of flight details:

```
1 [
2   {
3     "flightId": "692c3d95ccf32cc8590672cf",
4     "flightNumber": "AI101",
5     "airlineCode": "AI",
6     "sourceCity": "DELHI",
7     "destinationCity": "MUMBAI",
8     "departureDate": "2025-12-05",
9     "arrivalDate": "2025-12-05",
10    "departureTime": "10:30:00",
11    "arrivalTime": "12:45:00",
12    "mealAvailable": true,
13    "totalSeats": 180,
14    "availableSeats": 180,
15    "price": 5200.0
16  },
17  {
18    "flightId": "692c3db9ccf32cc859067384",
19    "flightNumber": "VIS101",
20    "airlineCode": "VIS",
21    "sourceCity": "DELHI",
22    "destinationCity": "KANPUR",
23    "departureDate": "2025-12-08",
```

### 3. POST Search a Flight

The screenshot shows an API client interface for a service named "Api Gateway - 9000". The endpoint is "http://localhost:9000/flight/api/flight/search" and the method is "POST". The request body is in raw JSON format, containing flight search criteria. The response body is also in raw JSON format, showing a single flight result with various details like flight ID, number, airline, cities, dates, times, seats, and price.

```
1 {
2   "sourceCity": "DELHI",
3   "destinationCity": "KOCHI",
4   "travelDate": "2025-12-06",
5   "tripType": "ONE_WAY"
6 }
```

```
1 [
2   {
3     "flightId": "692d75a8196ea5545b332d35",
4     "flightNumber": "VIS102",
5     "airlineCode": "VIS",
6     "sourceCity": "DELHI",
7     "destinationCity": "KOCHI",
8     "departureDate": "2025-12-06",
9     "arrivalDate": "2025-12-06",
10    "departureTime": "12:30:00",
11    "arrivalTime": "13:00:00",
12    "mealAvailable": true,
13    "totalSeats": 180,
14    "availableSeats": 180,
15    "price": 5200.0
16  }
17 ]
```

### 4. POST Add a flight

The screenshot shows an API client interface for a service named "Api Gateway - 9000". The endpoint is "http://localhost:9000/flight/api/flight/airline/inventory/add" and the method is "POST". The request body is in raw JSON format, containing flight details for a new entry. The response body is in raw JSON format, showing a confirmation with a new flight ID. A green status bar at the bottom right of the response area indicates "201 Created".

```
1 {
2   "airlineCode": "MA",
3   "flightNumber": "AI178",
4   "sourceCity": "DELHI",
5   "destinationCity": "MUMBAI",
6   "departureDate": "2025-12-05",
7   "departureTime": "10:30",
8   "arrivalDate": "2025-12-05",
9   "arrivalTime": "12:45",
10  "totalSeats": 180,
11  "price": 5200,
12  "mealAvailable": true
13 }
```

```
1 {
2   "flightId": "692e0dcccdd57334064d81ab9"
3 }
```

201 Created

## 5. DEL Cancel a booking by pnr

The screenshot shows the API Gateway interface for the endpoint `http://localhost:9000/booking/api/booking/cancel/OEF4DE`. The method is `DELETE`. The response body is JSON, showing a successful cancellation with the message `"message": "Booking cancelled"`.

```
{
  "message": "Booking cancelled"
}
```

## 6. GET Ticket by pnr

The screenshot shows the API Gateway interface for the endpoint `http://localhost:9000/booking/api/booking/ticket/6ADB70`. The method is `GET`. The response body is JSON, showing a cancelled ticket with the following details:

```
{
  "bookingId": "692d80341a1d5941c03de245",
  "tripType": "ONE_WAY",
  "outboundFlightId": "692c3dd7ccf32cc859067439",
  "returnFlight": null,
  "pnrOutbound": "6ADB70",
  "pnrReturn": null,
  "contactName": "Kanchan",
  "contactEmail": "k@example.com",
  "totalPassengers": 1,
  "status": "CANCELLED"
}
```

## 7. GET Booking history by email

The screenshot shows the API Gateway interface for the endpoint `http://localhost:9000/booking/api/booking/history/k@example.com`. The method is `GET`. The response body is JSON, showing a list of booking history entries for the email `k@example.com`:

```
[
  {
    "bookingId": "692d7e0f2ede0e4dc798ffa2",
    "tripType": "ONE_WAY",
    "outboundFlightId": "692c3d95ccf32cc8590672cf",
    "returnFlight": null,
    "pnrOutbound": "FB7A59",
    "pnrReturn": null,
    "contactName": "Kanchan",
    "contactEmail": "k@example.com",
    "totalPassengers": 1,
    "status": "CANCELLED"
  },
  {
    "bookingId": "692d80341a1d5941c03de245",
    "tripType": "ONE_WAY",
    "outboundFlightId": "692c3dd7ccf32cc859067439",
    "returnFlight": null,
    "pnrOutbound": "6ADB70",
    "pnrReturn": null,
    "contactName": "Kanchan",
    "contactEmail": "k@example.com",
    "totalPassengers": 1,
    "status": "CANCELLED"
  }
]
```

## 8. Service Unavailable when Flight Service is down - Circuit Breaker

The screenshot shows an API client interface for a POST request to `http://localhost:9000/booking/api/booking/692d7540196ea5545b332bcb`. The request body is a JSON object with the following structure:

```
1 {
2   "tripType": "ONE_WAY",
3   "contactName": "Kanchan",
4   "contactEmail": "rajatrajputdev@gmail.com",
5   "passengers": [
6     {
7       "name": "Rajat",
8       "age": 26,
9       "gender": "MALE",
10      "seatOutbound": "S6"
11    }
12  ]
13 }
```

The response body shows a JSON object with an error message:

```
1 {
2   "error": "FlightService unavailable"
3 }
```

## 9. SEND Mail when a flight is booked/cancelled through message broker in Apache Kafka

The screenshot shows a WhatsApp message from `kanchanrai2307` at 12:50 AM. The message content is:

**Ticket cancelled - PNR 0EF4DE**

Your ticket is cancelled.  
PNR: 0EF4DE  
Return PNR: N/A  
Outbound flight: 692d7540196ea5545b332bcb  
Return flight: N/A  
Passengers: 1  
Status: CANCELLED