



Week 6 - Project Report

By : Kanchan Rai

Topic : Microservices Flight Booking System w/ Web 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

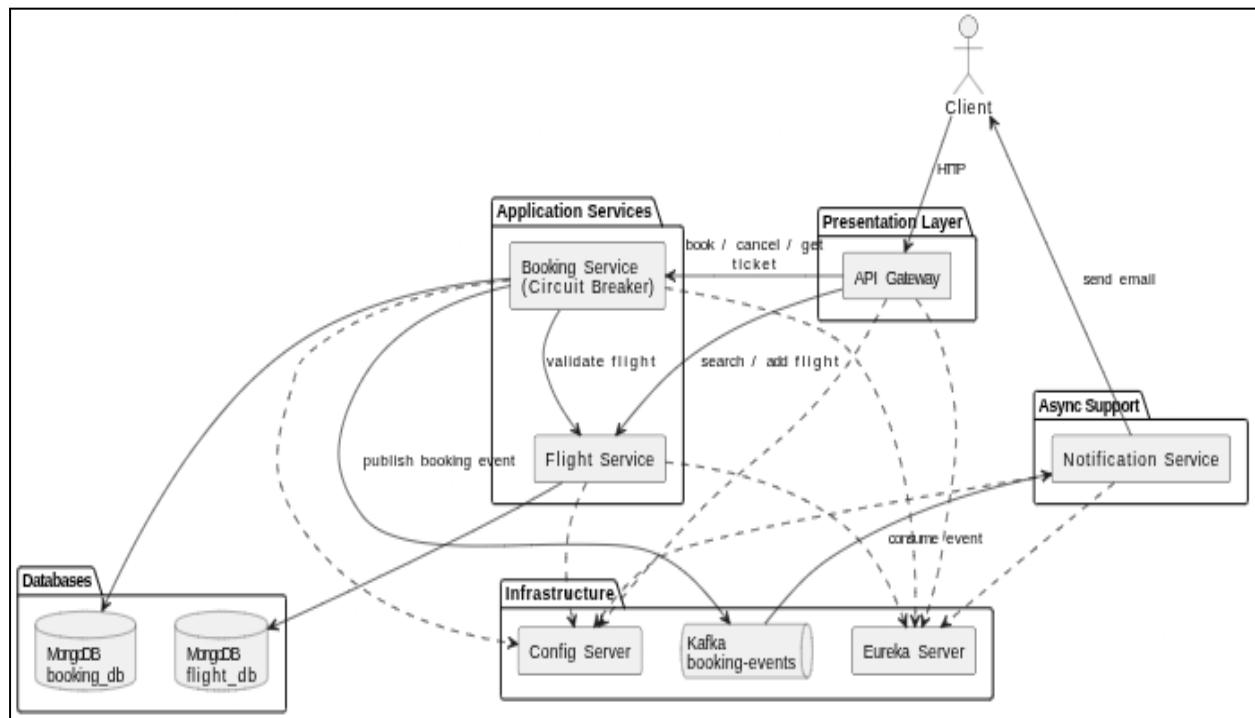
 4.3 With 100 Threads

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

The screenshot shows the Spring Eureka dashboard. At the top, it displays 'spring Eureka' and 'HOME LAST 1000 SINCE STARTUP'. Below this, the 'System Status' section shows environment 'test', data center 'default', current time '2025-12-01T22:49:44 +0530', uptime '00:05', lease expiration enabled 'false', renew threshold '8', and renew (last min) '8'. The 'DS Replicas' section lists a single instance 'localhost'. The 'Instances currently registered with Eureka' table shows four services: API-GATEWAY, BOOKING-SERVICE, CONFIG-SERVER, and FLIGHT-SERVICE, each with one instance and status 'UP'. The 'General Info' section shows two metrics: 'total-avail-memory' at 90mb and 'num-of-cpus' at 12.

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

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%

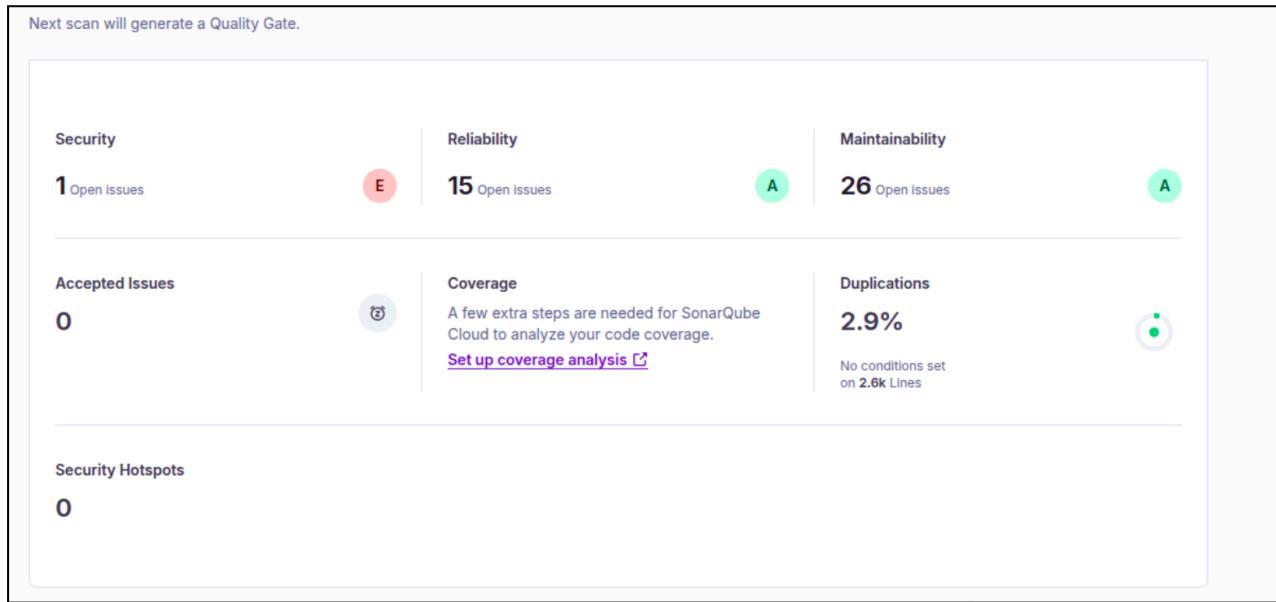
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.flightservice.service	96%	83%	10	46	2	100	1	19	0	3		
com.flightservice.controller	70%	n/a	2	7	2	7	2	7	0	1		
com.flightservice	37%	n/a	1	2	2	3	1	2	0	1		
com.flightservice.request	100%	n/a	0	39	0	57	0	39	0	3		
com.flightservice.exceptions	100%	90%	1	14	0	30	0	9	0	3		
com.flightservice.model	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%

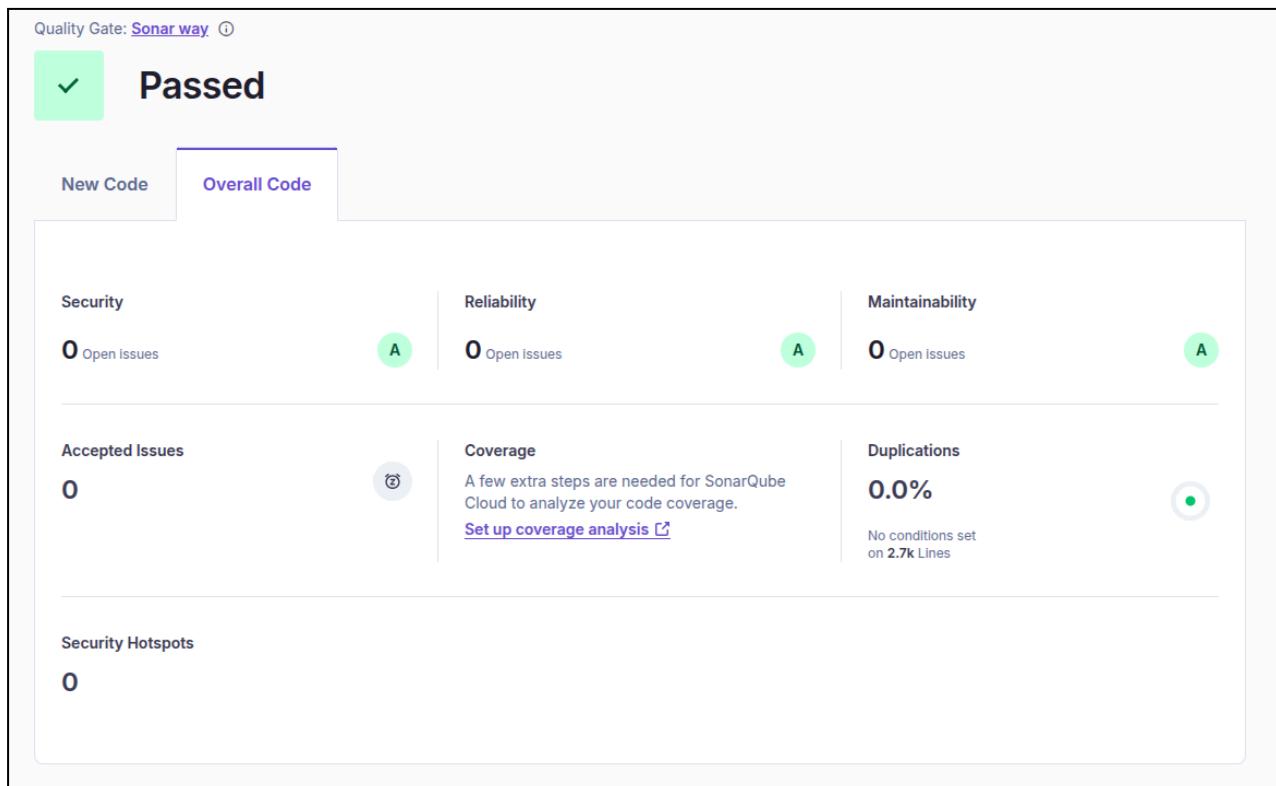
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.bookingservice.service	91%	82%	16	67	10	184	6	38	0	4		
com.bookingservice.model	86%	n/a	10	30	10	53	10	30	0	6		
com.bookingservice.requests	84%	n/a	4	24	5	35	4	24	0	2		
com.bookingservice	37%	n/a	1	2	2	3	1	2	0	1		
com.bookingservice.exceptions	100%	92%	1	16	0	42	0	9	0	3		
com.bookingservice.client	100%	n/a	0	15	0	42	0	15	0	3		
com.bookingservice.controller	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

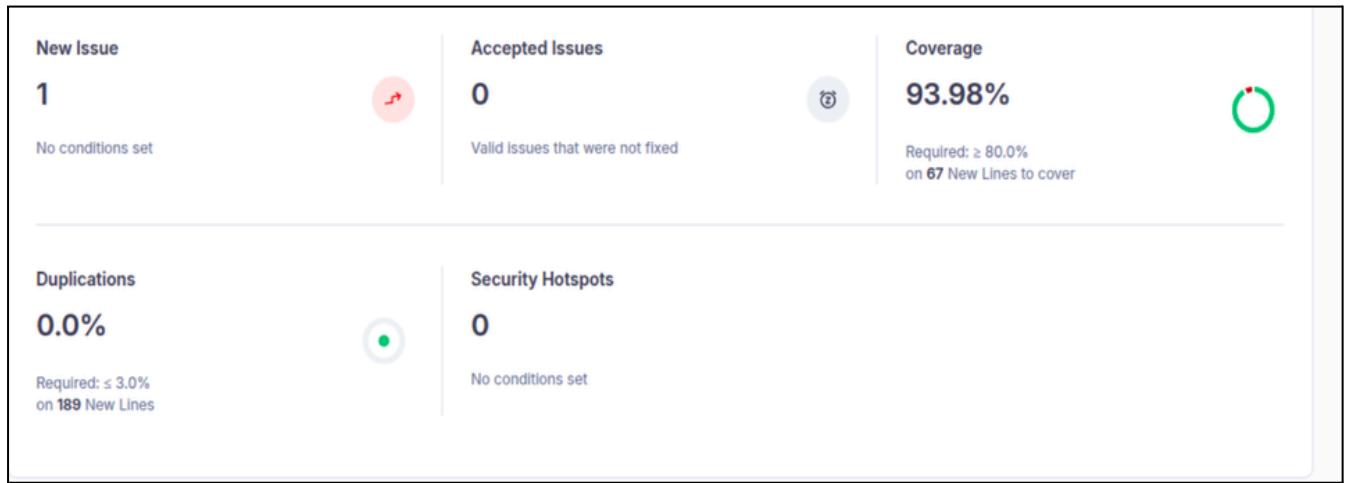
3. SonarQube Report

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.





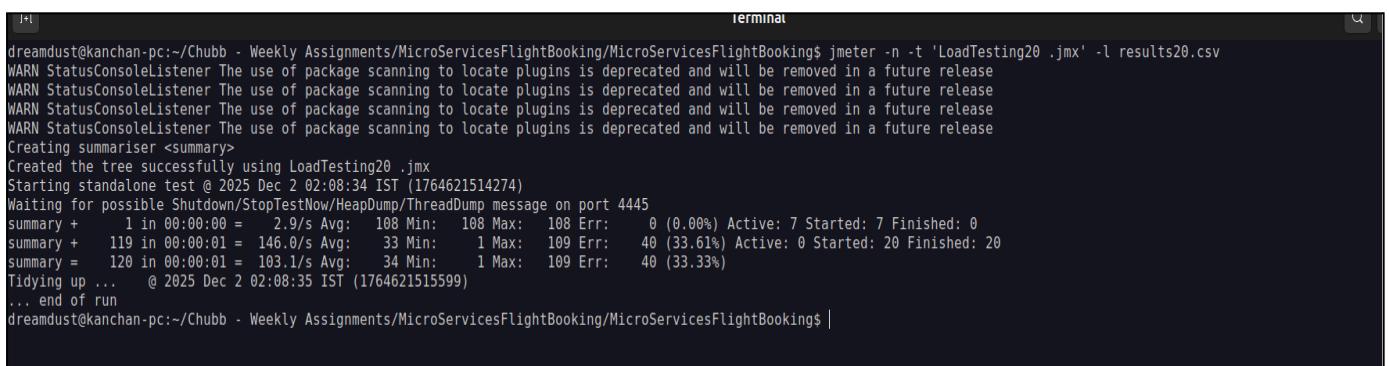
4. JMeter Load Testing

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: 20
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 fli...	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 Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:9000/flight/api/flight/addAirline
- Headers:** (8)
- Body:** raw JSON
- Request Body:**

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

- Response Body:**

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

2. GET All Airlines

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:9000/flight/api/flight/getAllFlights
- Headers:** (6)
- Body:** raw JSON
- Response Body:**

```
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 the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:9000/flight/api/flight/search
- Body:** JSON (selected)
- Request Body:**

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

- Response Body:**

```
1 [
2     {
3         "flightId": "692d75a8196ea5545b332d35",
4         "flightNumber": "VIS102",
5         "airlineCode": "VIS",
6         "sourceCity": "DELHI",
7         "destinationCity": "KOCHE",
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 the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:9000/flight/api/flight/airline/inventory/add
- Body:** JSON (selected)
- Request Body:**

```
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 }
```

- Response Status:** 201 Created
- Response Body:**

```
1 {
2     "flightId": "692e0dccdd57334064d81ab9"
3 }
```

5. DEL Cancel a booking by pnr

The screenshot shows the API Gateway interface with a DELETE request to cancel a booking. The URL is `http://localhost:9000/booking/api/booking/cancel/0EF4DE`. The response body is a JSON object with a single key "message": "Booking cancelled".

```
1 {  
2   "message": "Booking cancelled"  
3 }
```

6. GET Ticket by pnr

The screenshot shows the API Gateway interface with a GET request to get a ticket by pnr. The URL is `http://localhost:9000/booking/api/booking/ticket/6ADB70`. The response body is a JSON object representing a booking with various fields like bookingId, tripType, outboundFlightId, etc.

```
1 {  
2   "bookingId": "692d80341a1d5941c03de245",  
3   "tripType": "ONE WAY",  
4   "outboundFlightId": "692c3dd7ccf32cc859067439",  
5   "returnFlight": null,  
6   "pnrOutbound": "6ADB70",  
7   "pnrReturn": null,  
8   "contactName": "Kanchan",  
9   "contactEmail": "k@example.com",  
10  "totalPassengers": 1,  
11  "status": "CANCELLED"  
12 }
```

7. GET Booking history by email

The screenshot shows the API Gateway interface with a GET request to get booking history by email. The URL is `http://localhost:9000/booking/api/booking/history/k@example.com`. The response body is a JSON array containing two objects, each representing a booking with fields like bookingId, tripType, outboundFlightId, etc.

```
1 [  
2   {  
3     "bookingId": "692d7e0f2ede0e4dc798ffa2",  
4     "tripType": "ONE WAY",  
5     "outboundFlightId": "692c3d95ccf32cc8590672cf",  
6     "returnFlight": null,  
7     "pnrOutbound": "FB7A59",  
8     "pnrReturn": null,  
9     "contactName": "Kanchan",  
10    "contactEmail": "k@example.com",  
11    "totalPassengers": 1,  
12    "status": "CANCELLED"  
13  },  
14  {  
15    "bookingId": "692d80341a1d5941c03de245",  
16    "tripType": "ONE WAY",  
17    "outboundFlightId": "692c3dd7ccf32cc859067439",  
18    "returnFlight": null,  
19    "pnrOutbound": "6ADB70",  
20    "pnrReturn": null,  
21    "contactName": "Kanchan",  
22    "contactEmail": "k@example.com",  
23    "totalPassengers": 1,  
24    "status": "CANCELLED"  
25  }  
26 ]
```

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

The screenshot shows a Postman request to `http://localhost:9000/booking/api/booking/692d7540196ea5545b332bcb`. The request method is POST. The body contains a JSON object representing a flight booking:

```
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 an error message:

```
{ } JSON ▾ D Preview I Visualize ▾
1 {
2   "error": "FlightService unavailable"
3 }
```

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

