

## Eureka Server (service-registry):

### Instances currently registered with Eureka

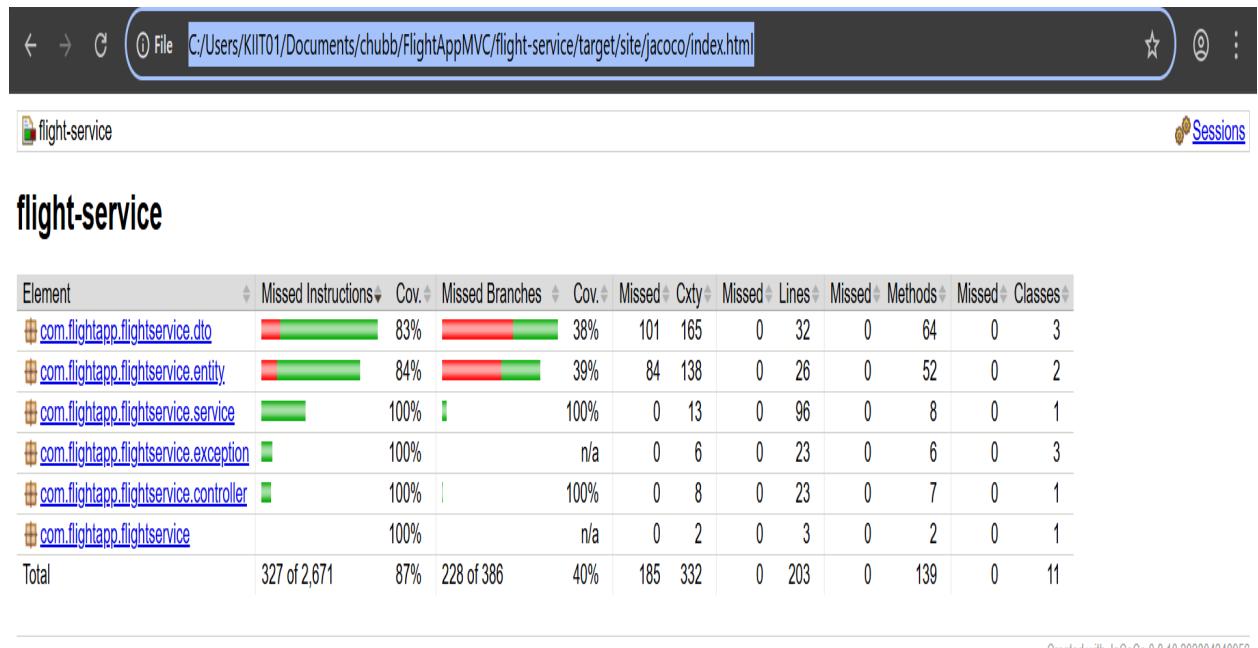
Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">KIIT01.mshome.net:api-gateway:8081</a>
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">KIIT01.mshome.net:booking-service:8082</a>
FLIGHT-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">KIIT01.mshome.net:flight-service:8083</a>

### General Info

Name	Value
total-avail-memory	90mb
num-of-cpus	8
current-memory-usage	53mb (58%)
server-upptime	01:07
registered-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
unavailable-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a> ,

## JACOCO-REPORTS:

### Flight-service(87%)



## Booking-service:

---

booking-service

Sessions

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Ctxy	Missed	Lines	Missed	Methods	Missed	Classes
com.flighthapp.bookingservice.exception	█████	31%	████	0%	41	51	7	26	17	27	0	5
com.flighthapp.bookingservice.dto	██████	80%	██████	41%	106	165	0	32	11	64	0	3
com.flighthapp.bookingservice.entity	████	82%	████	41%	74	120	1	23	4	46	0	2
com.flighthapp.bookingservice.feign	████	78%	████	40%	48	69	0	13	7	26	0	1
com.flighthapp.bookingservice.messaging	████	80%	████	43%	35	64	2	27	2	28	0	3
com.flighthapp.bookingservice.service	████	94%	████	78%	4	21	4	108	1	14	0	2
com.flighthapp.bookingservice.controller	████	88%	████	100%	0	8	3	25	0	6	0	1
com.flighthapp.bookingservice.config	████	100%		n/a	0	7	0	7	0	7	0	2
com.flighthapp.bookingservice		100%		n/a	0	2	0	3	0	2	0	1
Total	838 of 3,796	77%	348 of 574	39%	308	507	17	264	42	220	0	20

Created with JaCoCo 0.8.10 20230424095

## Postman Screenshots:

### 1. Adding Inventory

The screenshot shows the Postman interface with the following details:

- Collection:** My Collection / FlightAppMVC
- Method:** POST
- URL:** {{base\_url}} /api/flight/airline/inventory/add
- Body:** raw JSON (selected)
- Request Body:**

```
1 {
2   "airlineCode": "IND",
3   "airlineName": "Indigo",
4   "source": "DEL",
5   "destination": "BOM",
6   "departureTime": "2025-12-25T10:00:00",
7   "arrivalTime": "2025-12-25T12:00:00",
8   "price": 5000.0,
9   "totalSeats": 100,
10  "aircraft": "A320"
11 }
```

- Response Status:** 201 Created
- Response Time:** 800 ms
- Response Size:** 378 B
- Response Headers:** (3) - includes Content-Type: application/json, Location: /api/flight/airline/inventory/1, Date: Mon, 25 Dec 2023 10:00:00 GMT
- Response Body:**

```
1 {
2   "flightId": "fc10c69e-09da-484b-be44-6e3669869ae0".
```

## 2. Searching Flights:

The screenshot shows the Postman application interface. On the left, the sidebar displays 'SONAL KUMARI\_3114's Workspace' with sections for Collections, Environments, History, Flows, and Files. Under 'Collections', 'My Collection' is expanded, showing 'FlightApp', 'SpringBootWebfluxExample', 'MongoSpring', 'SpringMongodbReactive', 'FlightAppWebflux', 'QuizAppMono', 'QuizAppMicro', and 'FlightAppMVC'. 'FlightAppMVC' is selected and expanded, showing 'POST Add Flight', 'POST Search Flights' (which is highlighted), 'POST Book tickets by flightID', 'GET Booking details based on pnr', 'GET Booking History based on emailID', and 'DEL Cancel tickets based on pnr'. The main workspace shows a POST request to '({base\_url}) /api/flight/search'. The request body is a JSON object:

```
1  {
2    "source": "DEL",
3    "destination": "BOM",
4    "departureDate": "2025-12-25",
5    "journeyType": "ONE WAY"
6 }
```

The response status is '200 OK' with 95 ms latency and 375 B size. The response body is also a JSON object:

```
2  {
3    "flightId": "fc10c69e-09da-484b-be44-6e3669869ae0",
4    "airlineCode": "IND",
5    "airlineName": "Indigo",
6    "source": "DEL",
7    "destination": "BOM",
8    "departureTime": "2025-12-25T10:00:00",
9    "arrivalTime": "2025-12-25T12:00:00",
10   "price": 5000.0,
11   "availableSeats": 100,
```

### 3. Book tickets by providing flight ID:

The screenshot shows the Postman application interface. On the left, there's a sidebar titled "My Collection" containing various project entries like FlightApp, SpringBootWebfluxExample, etc. The main area shows a POST request to "({base\_url}) /api/booking/flight/fc10c69e-09da-484b-be44-6e3669869ae0". The "Body" tab is selected, displaying the following JSON payload:

```
1 {
2   "userEmail": "john@test.com",
3   "userName": "John Miller",
4   "numberOfSeats": 2,
5   "passengers": [
6     {
7       "name": "John",
8       "gender": "Male",
9       "age": 30
10    },
11    {
12      "name": "Jane",
13      "gender": "Female",
14      "age": 28
15    }
]
```

At the bottom, the response status is "201 Created" with a timestamp of "29.09 s" and a size of "478 B".

This screenshot shows another instance of the Postman interface with a similar setup. The "My Collection" sidebar is visible on the left. The main area shows a POST request to "({base\_url}) /api/booking/flight/fc10c69e-09da-484b-be44-6e3669869ae0". The "Body" tab is selected, displaying the following JSON payload:

```
1 {
2   "bookingId": "c40e0d50-646c-493a-b979-ec6e6f148f68",
3   "pnr": "PNR17646171725096a748e",
4   "flightId": "fc10c69e-09da-484b-be44-6e3669869ae0",
5   "userEmail": "john@test.com",
6   "userName": "John Miller",
7   "numberOfSeats": 2,
8   "selectedSeats": [
9     "1A",
10    "1B"
11  ],
12  "mealPreference": "VEG",
13  "totalPrice": 10000.0,
14  "bookingStatus": "CONFIRMED",
15  "journeyDate": "2025-12-25",
16  ...
17 }
```

At the bottom, the response status is "201 Created" with a timestamp of "29.09 s" and a size of "478 B".

## 4. GET Booking details based on PNR:

The screenshot shows the Postman application interface. On the left, there is a sidebar titled "My Collection" containing several project entries. The entry "Booking details based on pnr" is highlighted in green, indicating it is the selected API endpoint.

The main workspace displays the following details:

- Method:** GET
- URL:** {{base\_url}}/api/booking/ticket/PNR17646171725096a748e
- Body:** Raw JSON response (selected)
- Response Status:** 200 OK
- Response Time:** 5.61 s
- Response Size:** 473 B
- Response Content:** (JSON format, partially visible)

```
1 {  
2   "bookingId": "c40e0d50-646c-493a-b979-ec6e6f148f68",  
3   "pnr": "PNR17646171725096a748e",  
4   "flightId": "fc10c69e-09da-484b-be44-6e3669869ae0",  
5   "userEmail": "john@test.com",  
6   "userName": "John Miller",  
7   "numberOfSeats": 2,  
8   "selectedSeats": [  
9     "1A",  
10    "1B"  
11  ],  
12  "mealPreference": "VEG",  
13  "totalPrice": 10000.0,  
14  "bookingStatus": "CONFIRMED",  
15  "journeyDate": "2025-12-25",  
16  "otherDetails": "No additional details provided."}
```

## 5. Booking history based on emailID

The screenshot shows the MongoDB Compass interface with the following details:

- Collection:** My Collection
- API Call:** GET `{base_url}/api/booking/history/john@test.com`
- Status:** 200 OK (90 ms, 1.56 KB)
- Headers:** (6)
- Body:** JSON response (19 lines) containing flight booking details.

```
1 [ { "bookingId": "8bef14e7-e723-4b84-86dd-754357fb7813", "pnr": "PNR176461212744013dcac", "flightId": "fc10c69e-09da-484b-be44-6e3669869ae0", "userEmail": "john@test.com", "userName": "John Miller", "numberOfSeats": 2, "selectedSeats": [ "1A", "1B" ], "mealPreference": "VEG", "totalPrice": 10000.0, "bookingStatus": "CONFIRMED", "journeyDate": "2025-12-25", "createdAt": 1764612127464 } ]
```

## 6. Cancel ticket by giving pnr:

The screenshot shows the Postman interface with the following details:

- Collection:** My Collection
- Request Type:** DELETE
- URL:** {{base\_url}} /api/booking/cancel/PNR17646171725096a748e
- Headers:** (6 items)
- Body:** (Empty)
- Query Params:** (Empty)
- Response Status:** 200 OK
- Response Time:** 4.52 s
- Response Size:** 473 B
- Save Response:** Enabled
- Code Preview:** JSON
- Code:**

```
8   "selectedSeats": [
9     "1A",
10    "1B"
11  ],
12  "mealPreference": "VEG",
13  "totalPrice": 10000.0,
14  "bookingStatus": "CANCELLED",
15  "journeyDate": "2025-12-25",
16  "createdAt": 1764617172509
17 }
```

## JMETER TESTING:

### 20 requests:

The screenshot shows the JMeter Test Plan interface. On the left, the tree view displays the test plan structure:

- Test Plan
  - Thread Group (20 requests)
    - Search ticket based on pnr
    - Summary Report
    - View Results Tree** (selected)
  - Thread Group (50 requests)
    - Get booking history by emailID
    - View Results Tree
    - Summary Report
  - Thread Group (100 requests)
    - Search flights
    - HTTP Header Manager
    - Summary Report
    - View Results Tree

## 50 requests:

The screenshot shows the JMeter interface with a test plan containing several thread groups and a summary report.

**Test Plan Structure:**

- Test Plan
- Thread Group (20 requests)
  - Search ticket based on pnr
  - Summary Report
  - View Results Tree
- Thread Group (50 requests)
  - Get booking history by emailID
  - View Results Tree
- Summary Report** (highlighted in blue)
- Thread Group (100 requests)
  - Search flights
  - HTTP Header Manager
  - Summary Report
  - View Results Tree

**Summary Report Configuration:**

Name: Summary Report

Comments:

Write results to file / Read from file

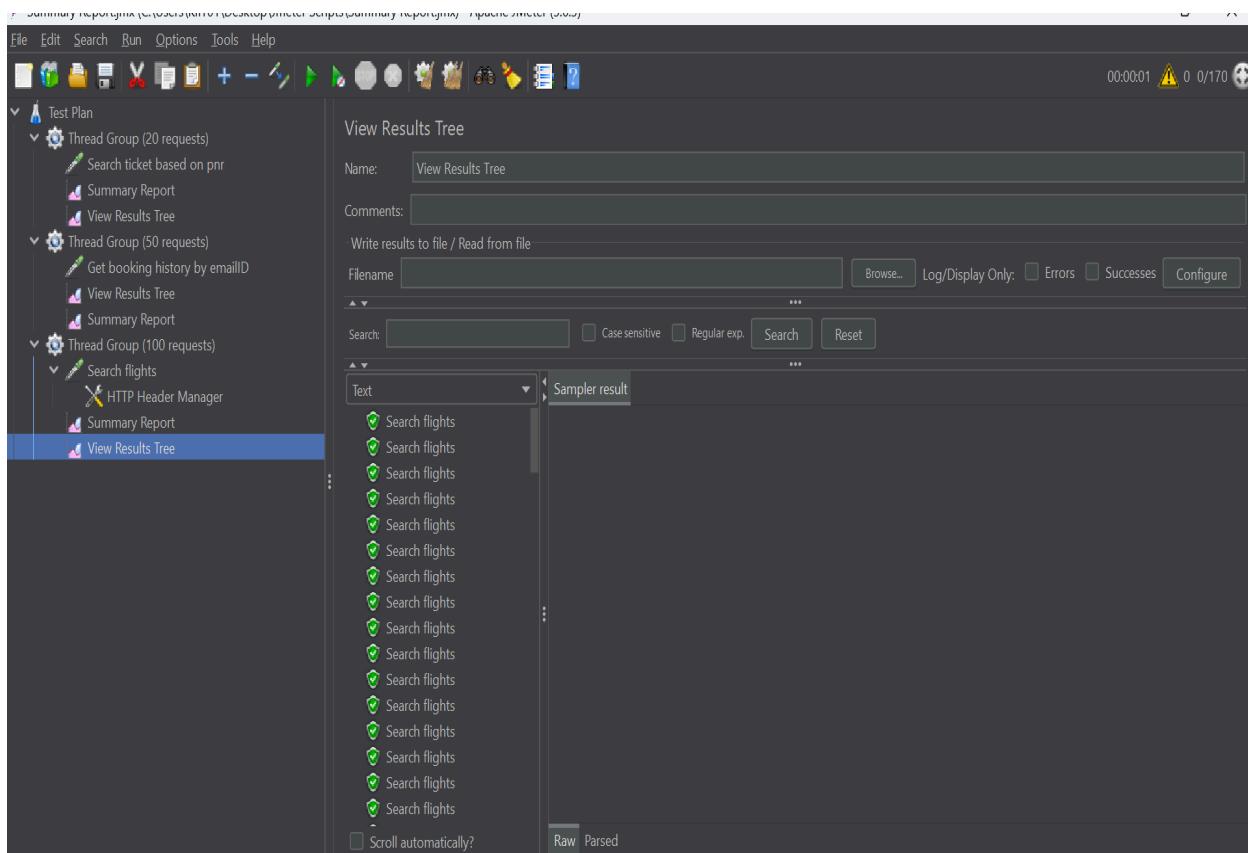
Filename:

Log/Display Only:  Errors  Successes  Configure

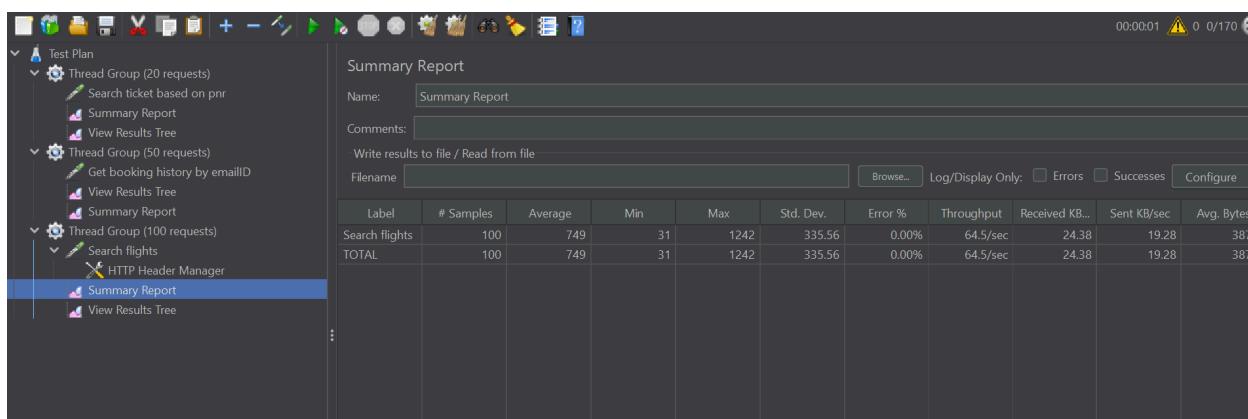
**Summary Report Data Table:**

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get booking...	50	277	20	711	205.87	0.00%	38.9/sec	60.97	5.66	1605.8
TOTAL	50	277	20	711	205.87	0.00%	38.9/sec	60.97	5.66	1605.8

## 100 requests:



The screenshot shows the JMeter interface with the 'View Results Tree' listener selected in the Test Plan tree on the left. The main panel displays a list of 100 sampler results, all labeled 'Search flights'. A dropdown menu above the list allows switching between 'Text' and 'Sampler result' views. At the bottom, there are tabs for 'Raw' and 'Parsed' data.



The screenshot shows the JMeter interface with the 'Summary Report' listener selected in the Test Plan tree on the left. The main panel displays a summary table for the 'Search flights' sampler. The table includes columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. The data shows 100 samples with an average response time of 749 ms, a minimum of 31 ms, and a maximum of 1242 ms. The error percentage is 0.00%, and the throughput is 64.5/sec.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Search flights	100	749	31	1242	335.56	0.00%	64.5/sec	24.38	19.28	387
TOTAL	100	749	31	1242	335.56	0.00%	64.5/sec	24.38	19.28	387

**Postman Collections and the log files for the dockerized microservices have been added to the Repository.**