CHUBB®

Week 7 - Project Report
By : Kanchan Rai

Topic : Spring Security & Dockerizing Flight Booking Microservices

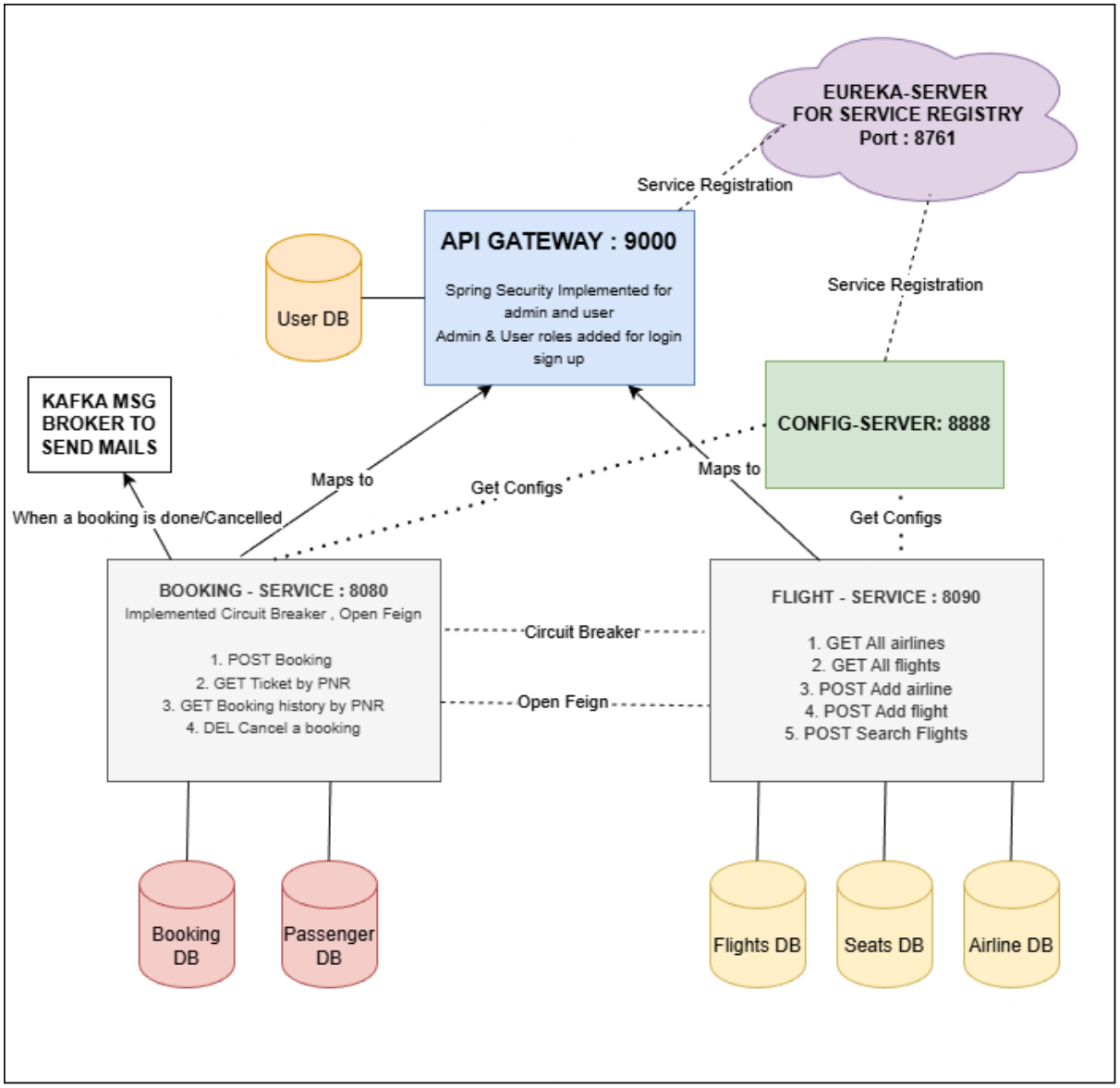
The following document contains the description and explanation of the project **Spring Security & Dockerizing Flight Booking Microservices.** This project has all the required requirements of the Week 6 project. As well as the spring security and dockerization of each microservice and running them as containers.

INDEX

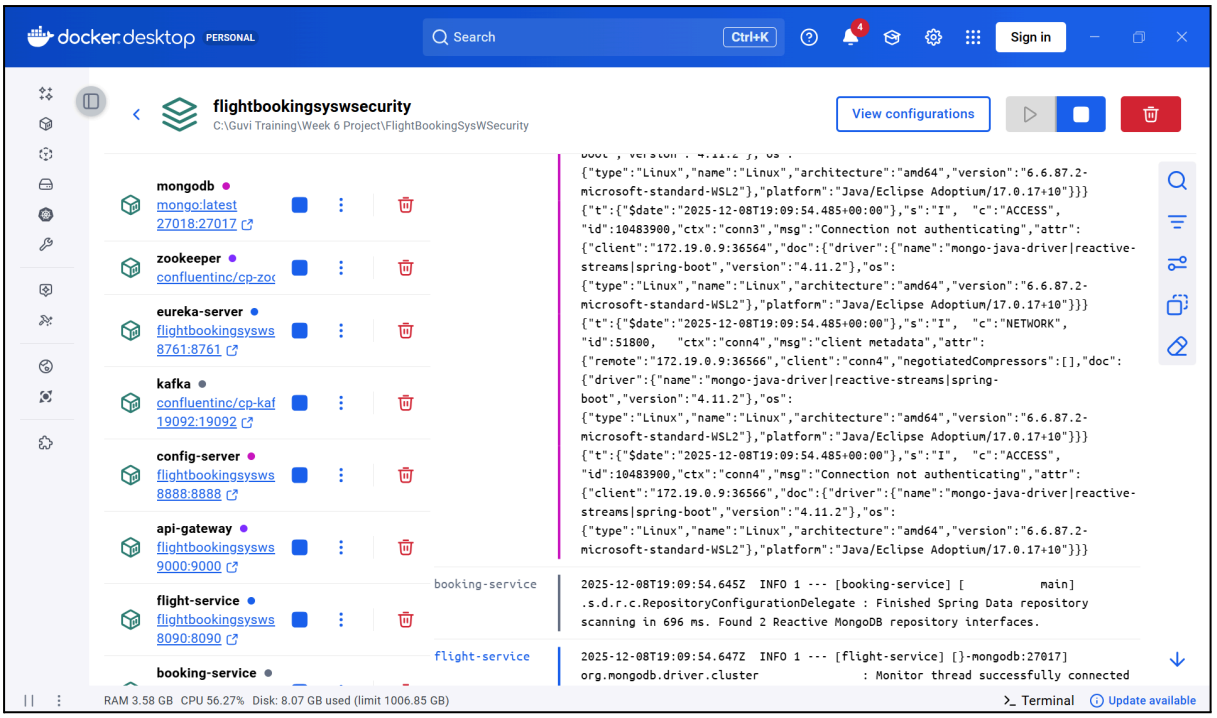
1. Project Overview

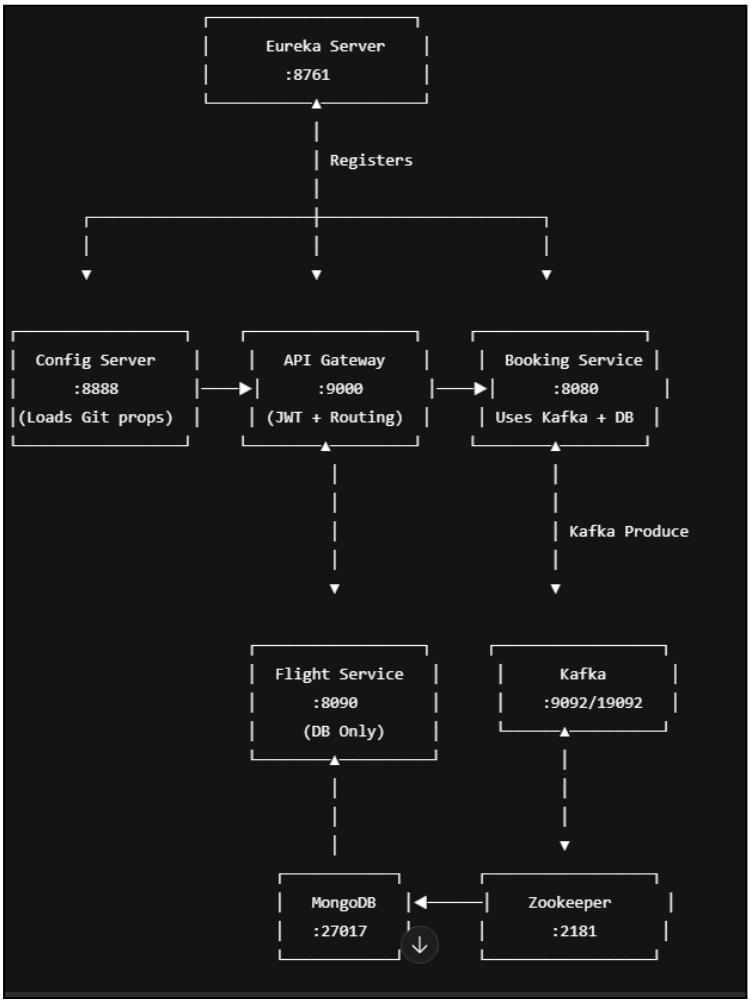
This project was a monolithic application with one single db called the "flightBookingSystem" in MongoDB, based on the reactive web flux in spring boot. The project has now been converted to a microservices architecture based project. There are two microservices namely "booking-service" and "flight-service". 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

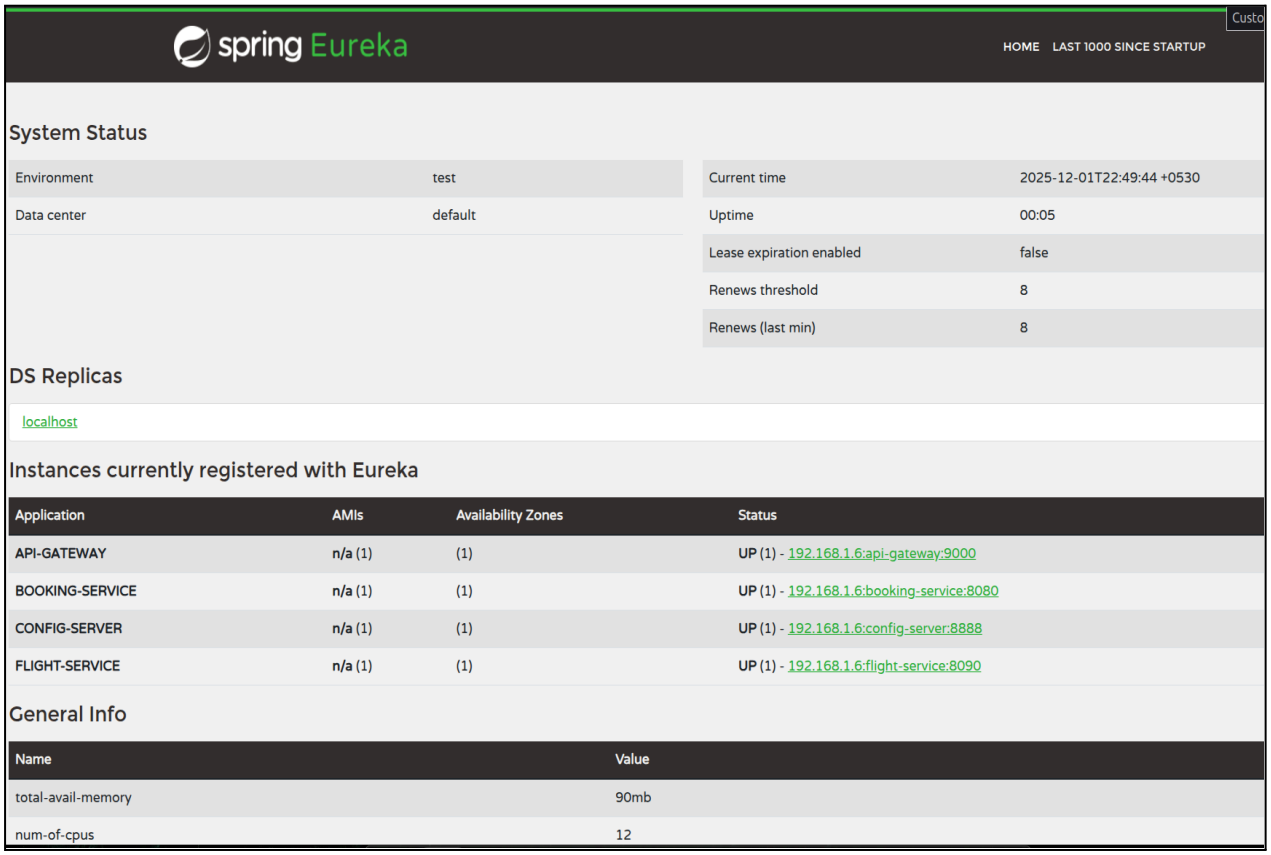


1.2 Dockerization Architecture Diagram





1.3 Eureka Registering

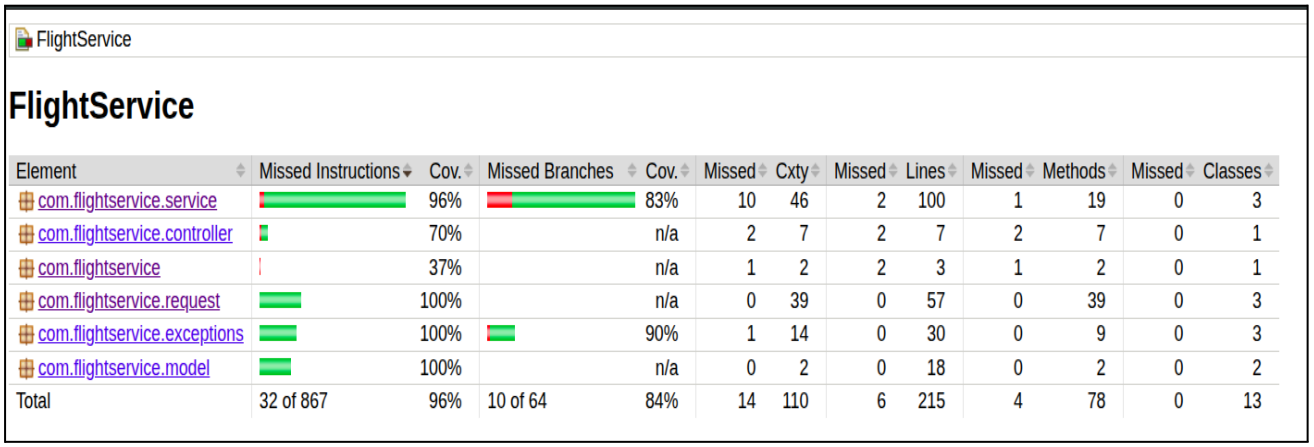


This is the Eureka Server where all four services are successfully registered, namely flight-service, booking-service, config-server,and api-gateway.

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.

2. JCoco Coverage Report -

Coverage Report for FlightService - 96%

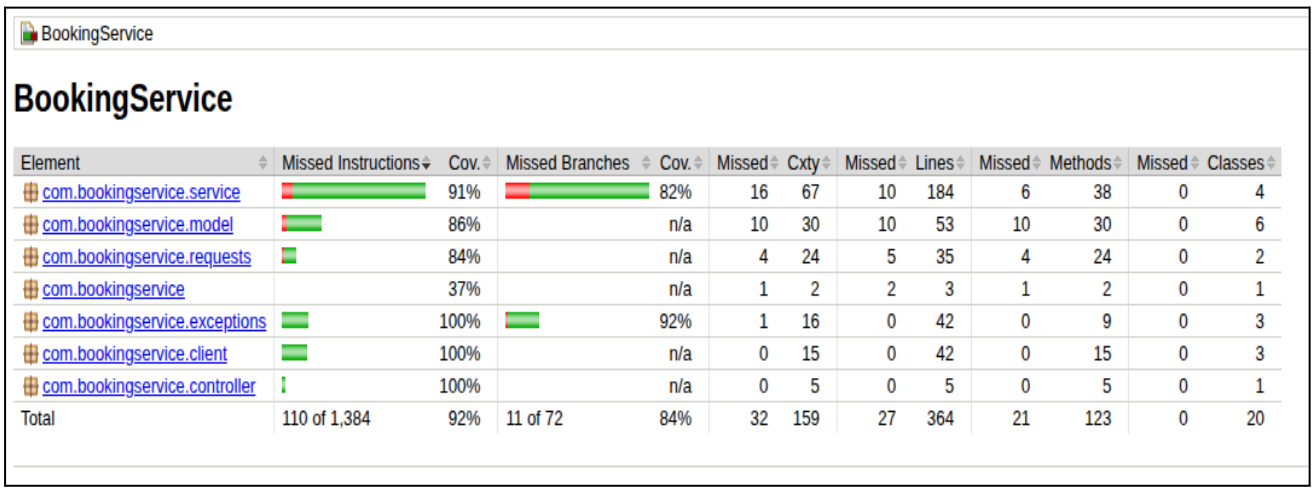


FlightService

FlightService

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%

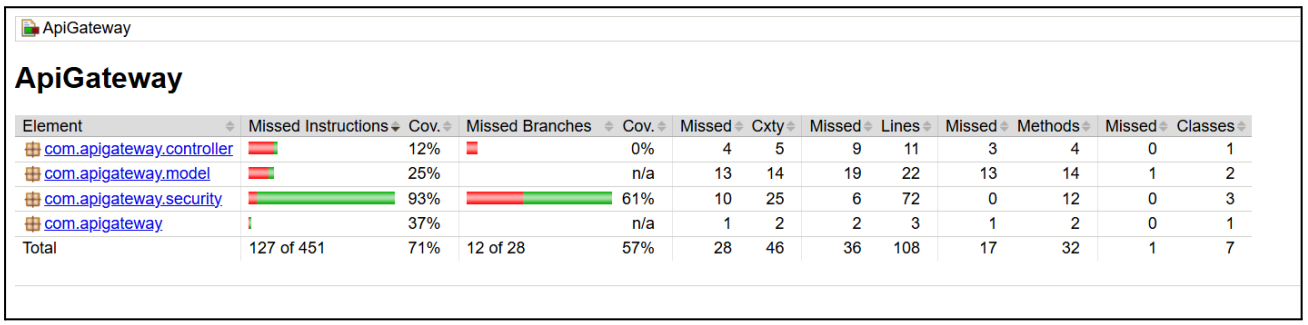


BookingService

BookingService

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

Coverage Report for Api Gateway -



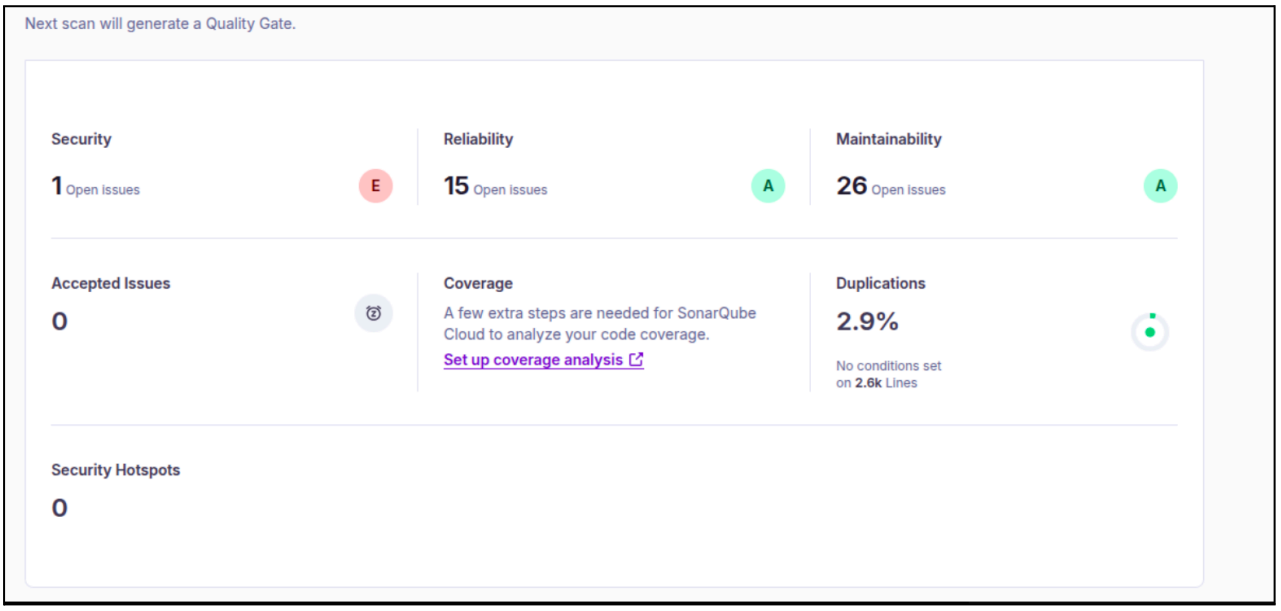
ApiGateway

ApiGateway

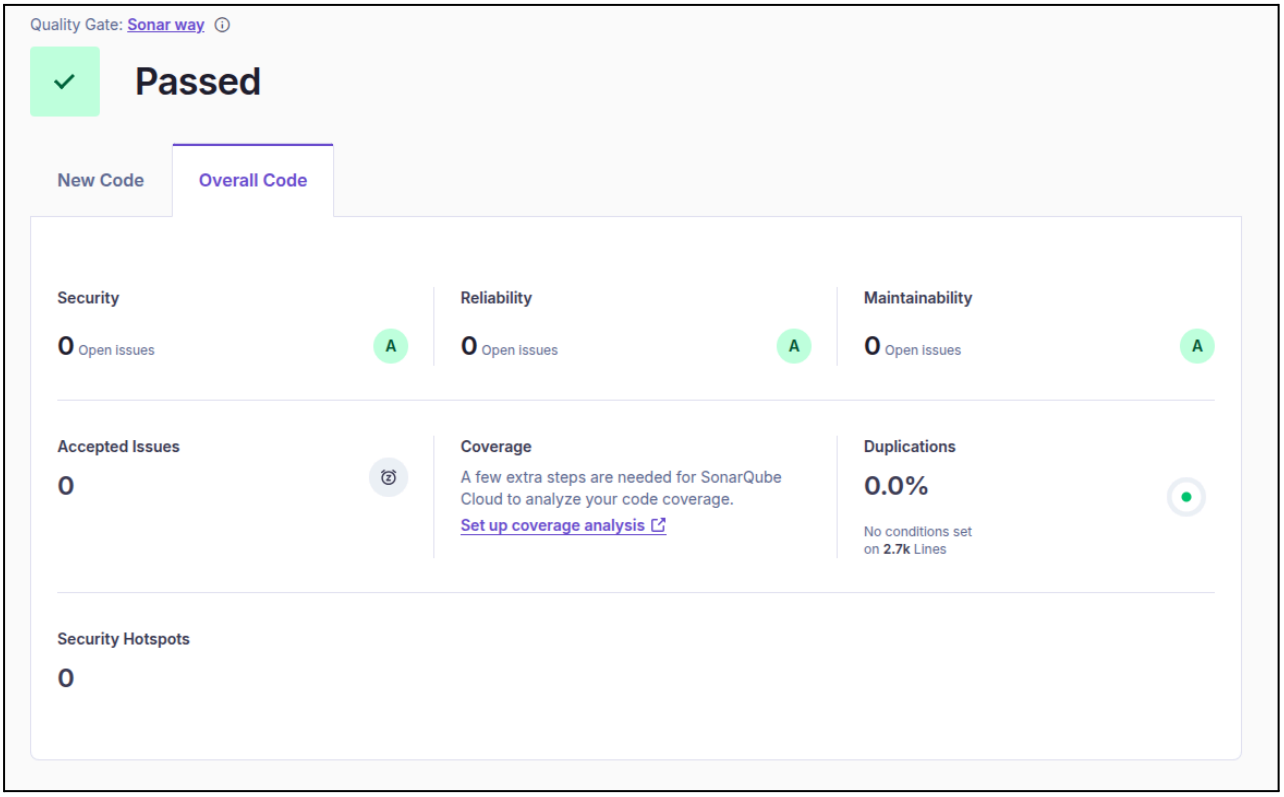
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.apigateway.controller		12%		0%	4	5	9	11	3	4	0	1
com.apigateway.model		25%		n/a	13	14	19	22	13	14	1	2
com.apigateway.security		93%		61%	10	25	6	72	0	12	0	3
com.apigateway		37%		n/a	1	2	2	3	1	2	0	1
Total	127 of 451	71%	12 of 28	57%	28	46	36	108	17	32	1	7

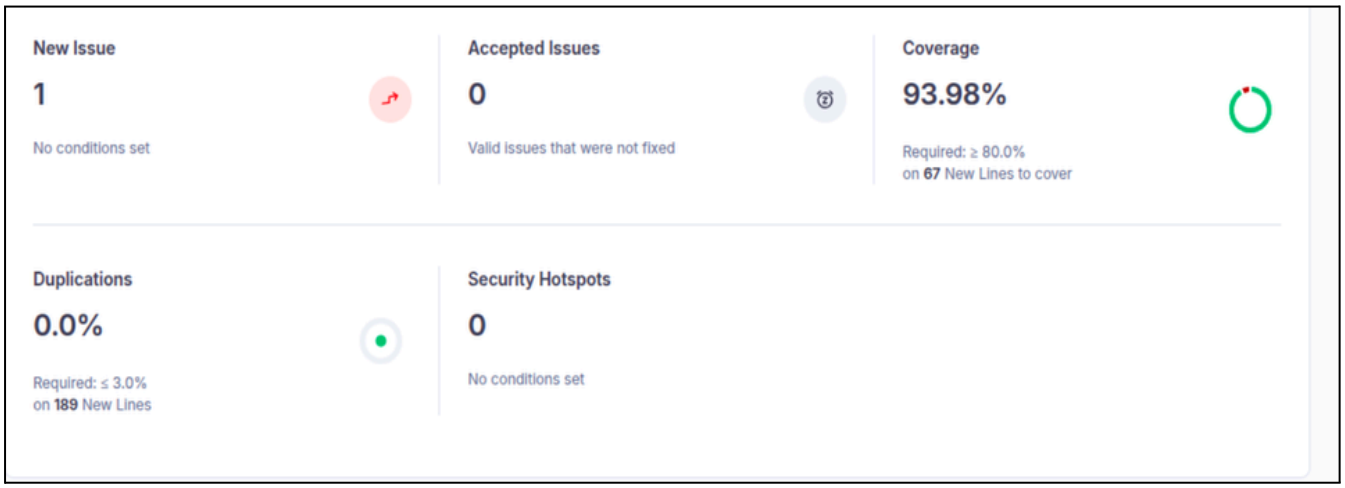
3. SonarQube Report

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.




New Issue
1
No conditions set
Accepted Issues
0
Valid issues that were not fixed
Coverage
93.98%
Required: ≥ 80.0%
on 67 New Lines to cover
Duplications
0.0%
Required: ≤ 3.0%
on 189 New Lines
Security Hotspots
0
No conditions set

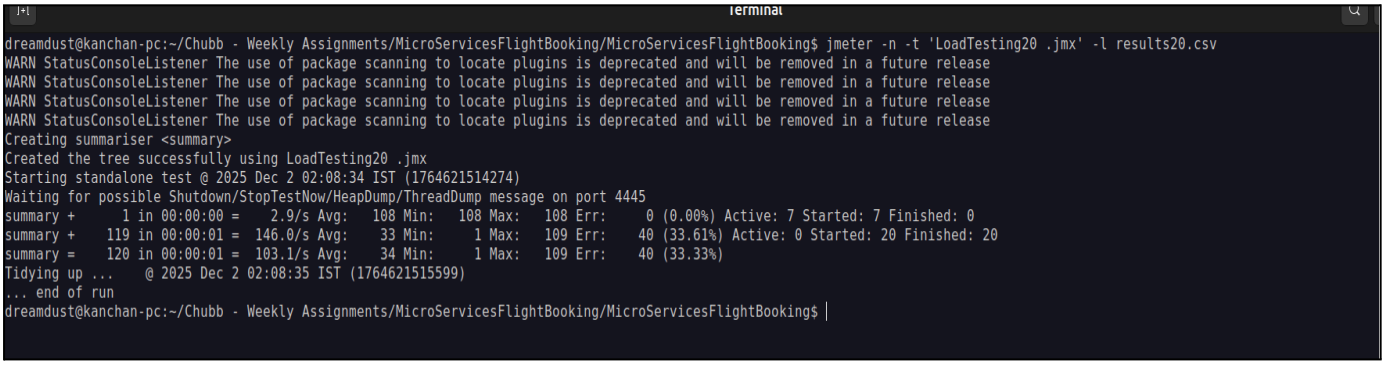
4. JMeter Load Testing

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
```



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

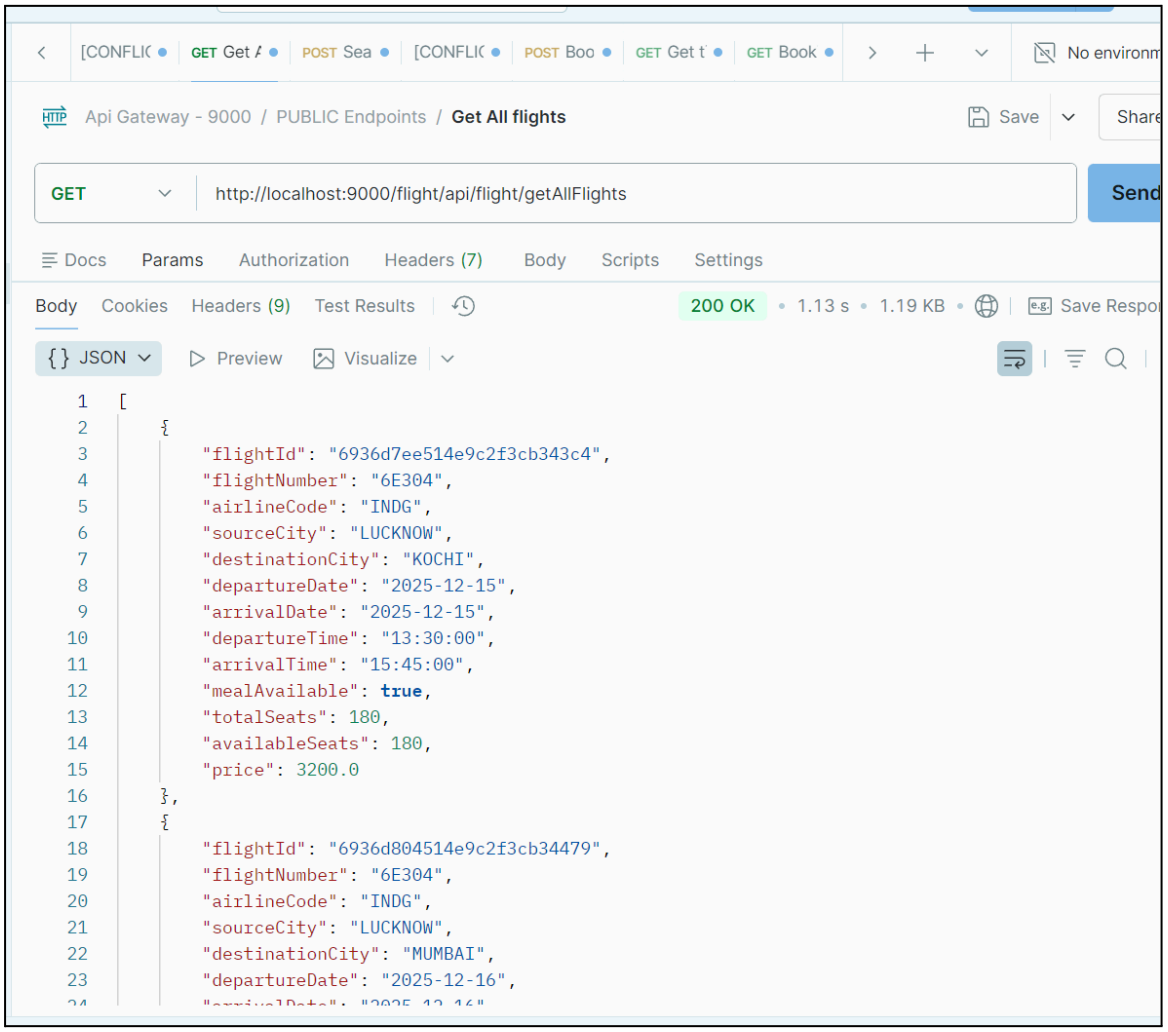
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get All Flights	200	553	203	806	139.40	0.00%	151.2/sec	391.82	30.71	2654.1
Get All Airlines	200	667	215	1081	213.94	0.00%	150.0/sec	389.13	30.48	2655.8
Post Book a fli...	200	283	20	738	190.77	0.00%	103.2/sec	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. Running the docker container

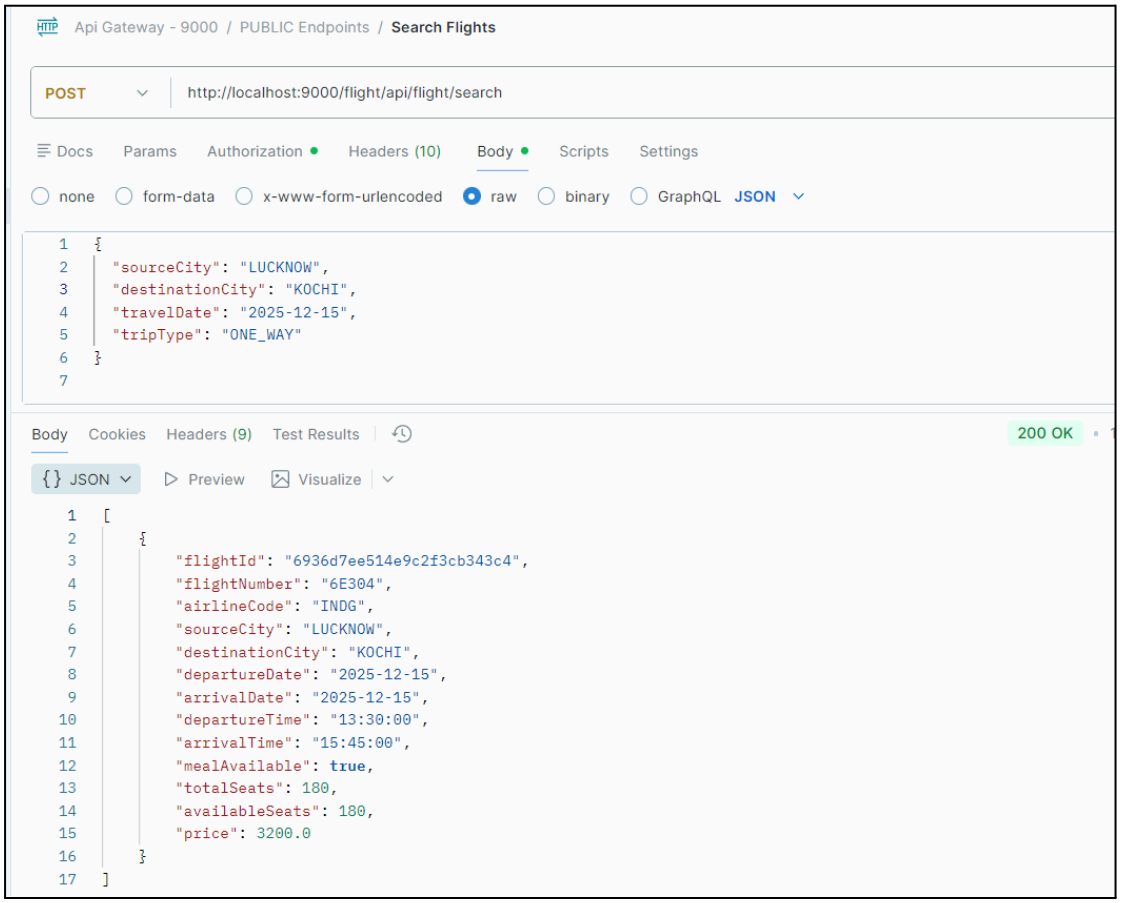
API Gateway on port 9000

5.1 Public Routes (No authorization required)

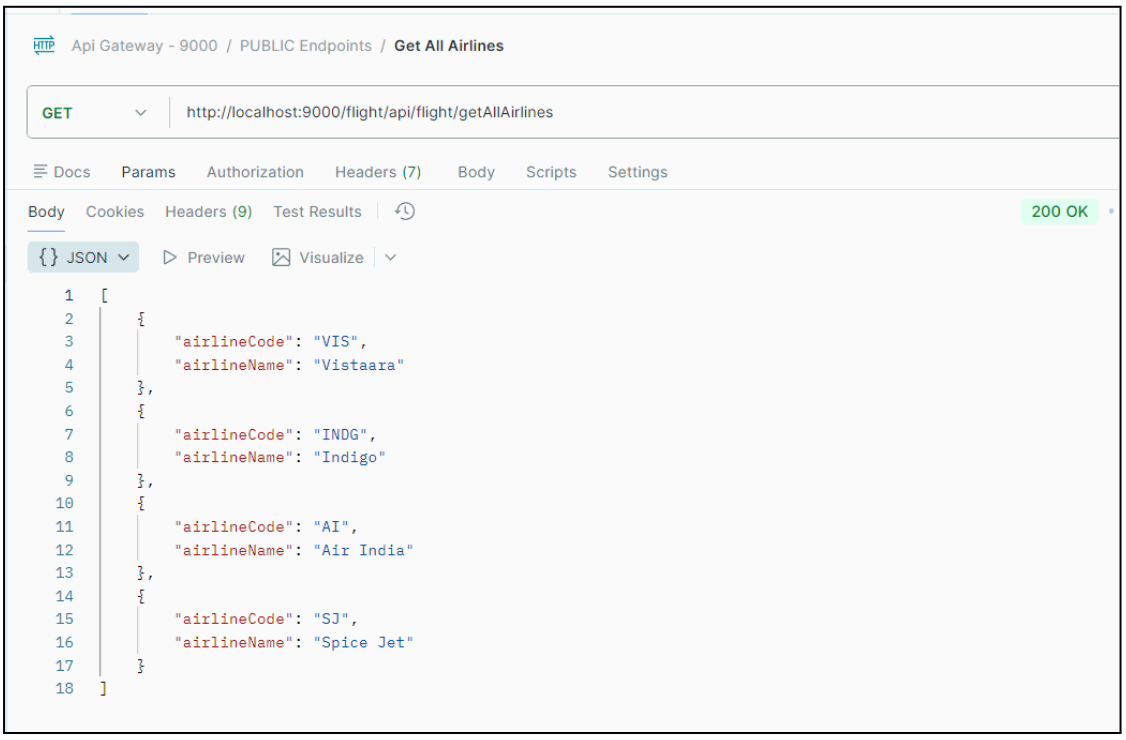
1. GET All flights



2. POST Search Flights

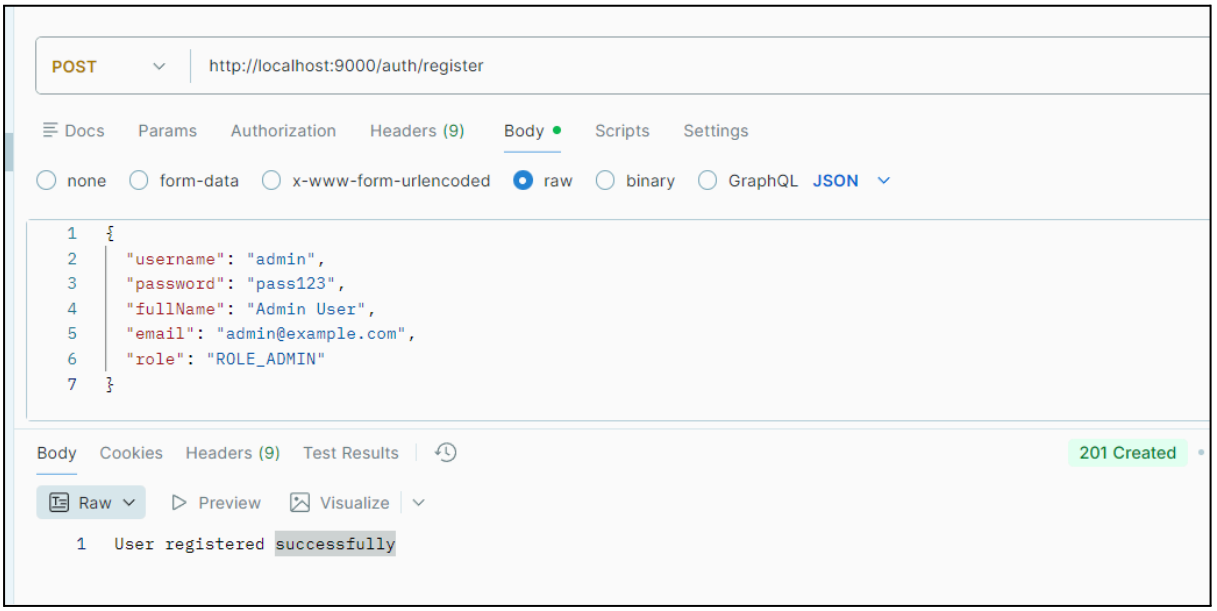


3. GET All Airlines

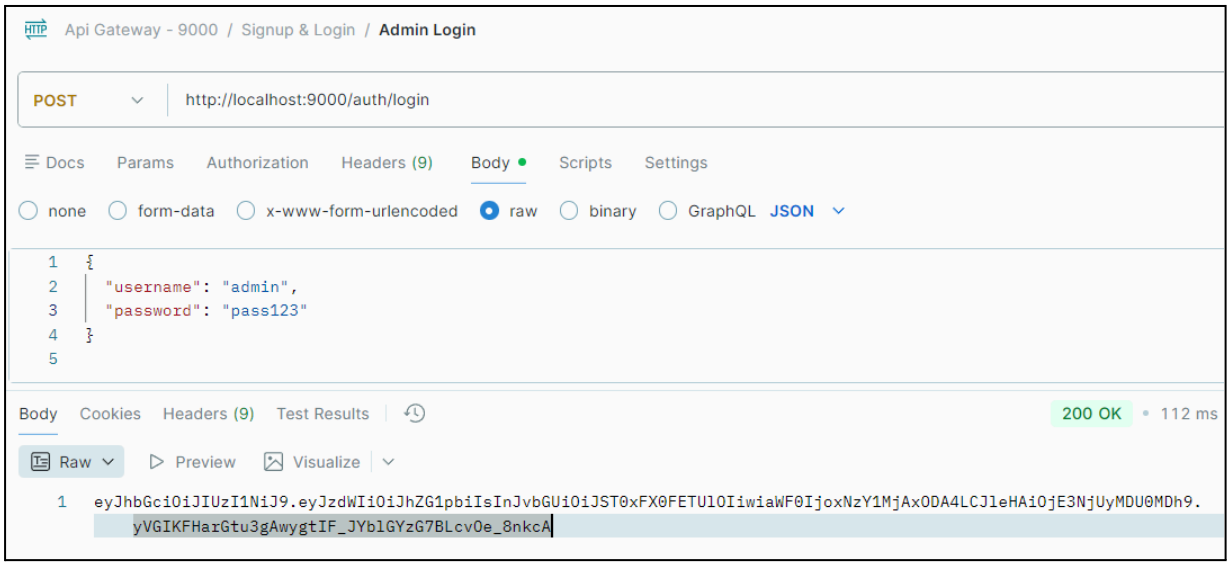


5.2 Sign Up & Login

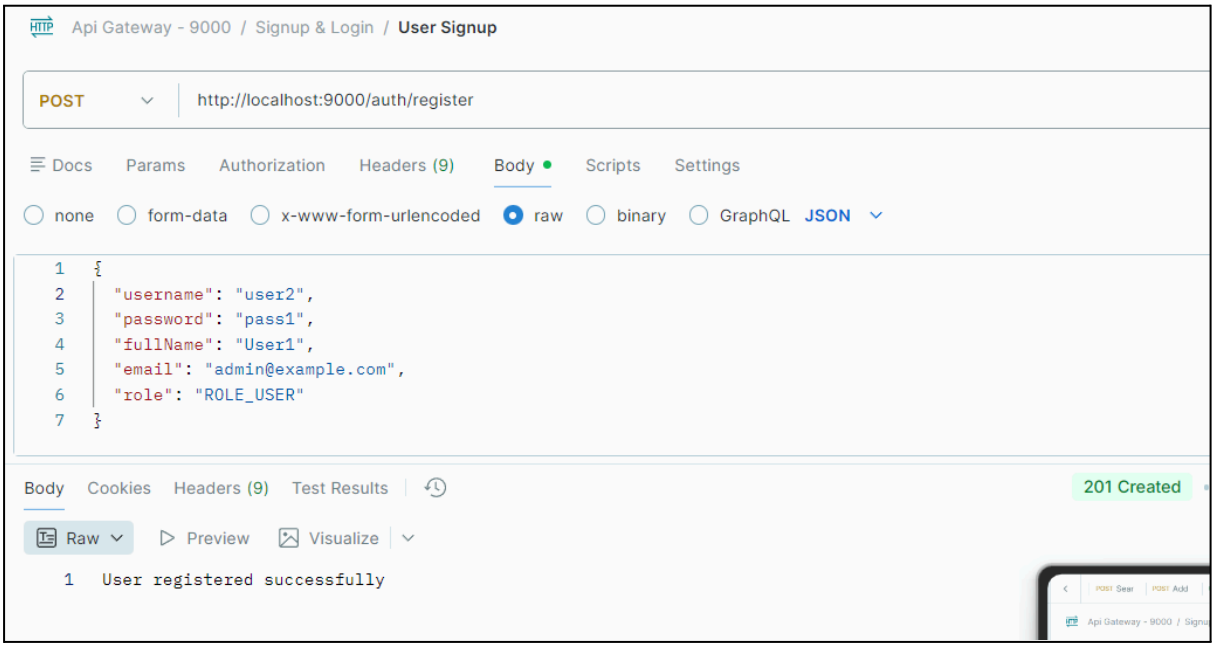
1. Admin Sign Up



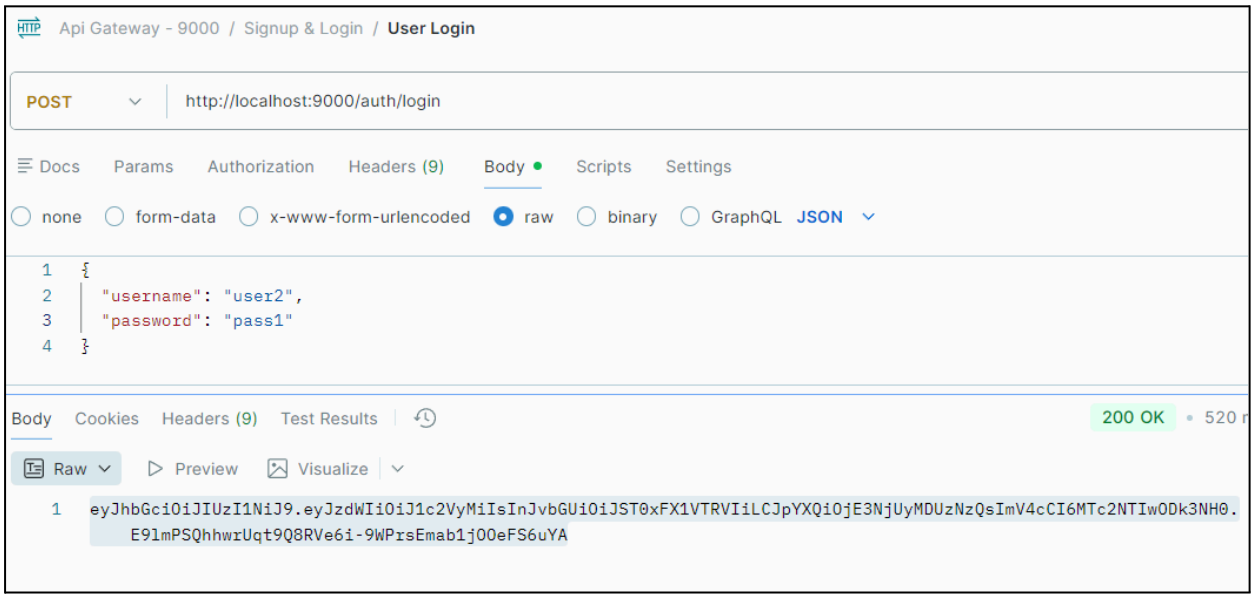
2. Admin Login



3. User Signup

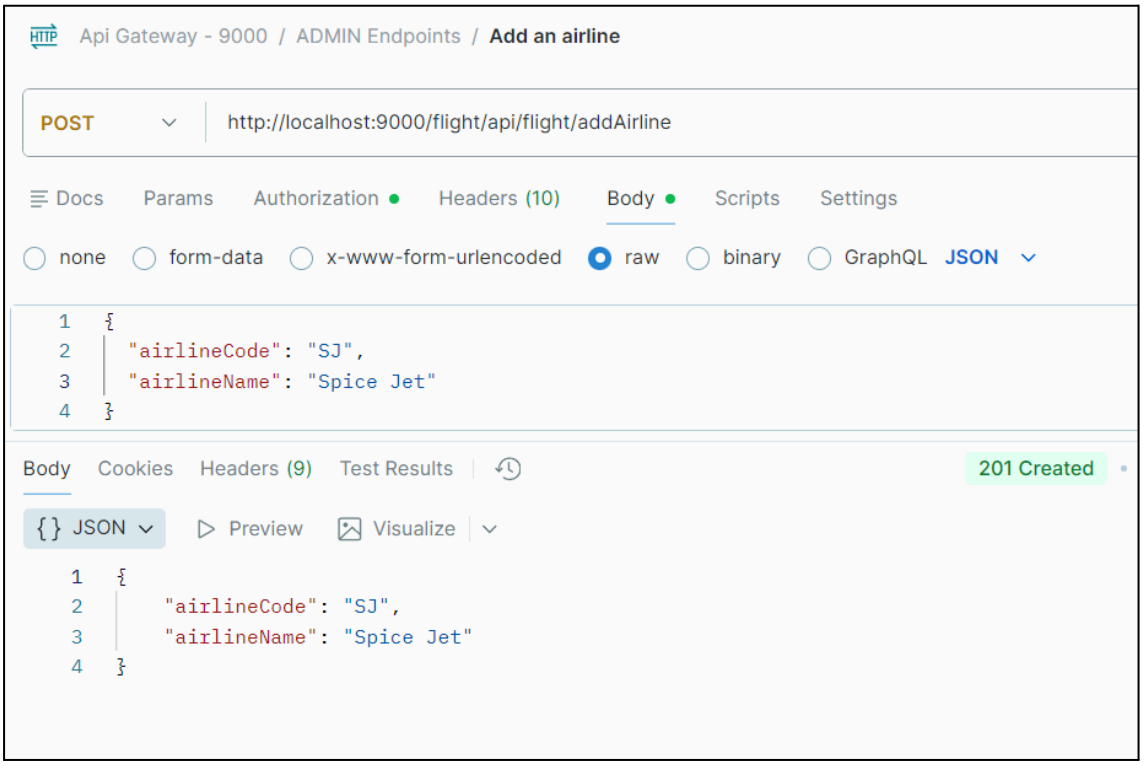


4. User Login

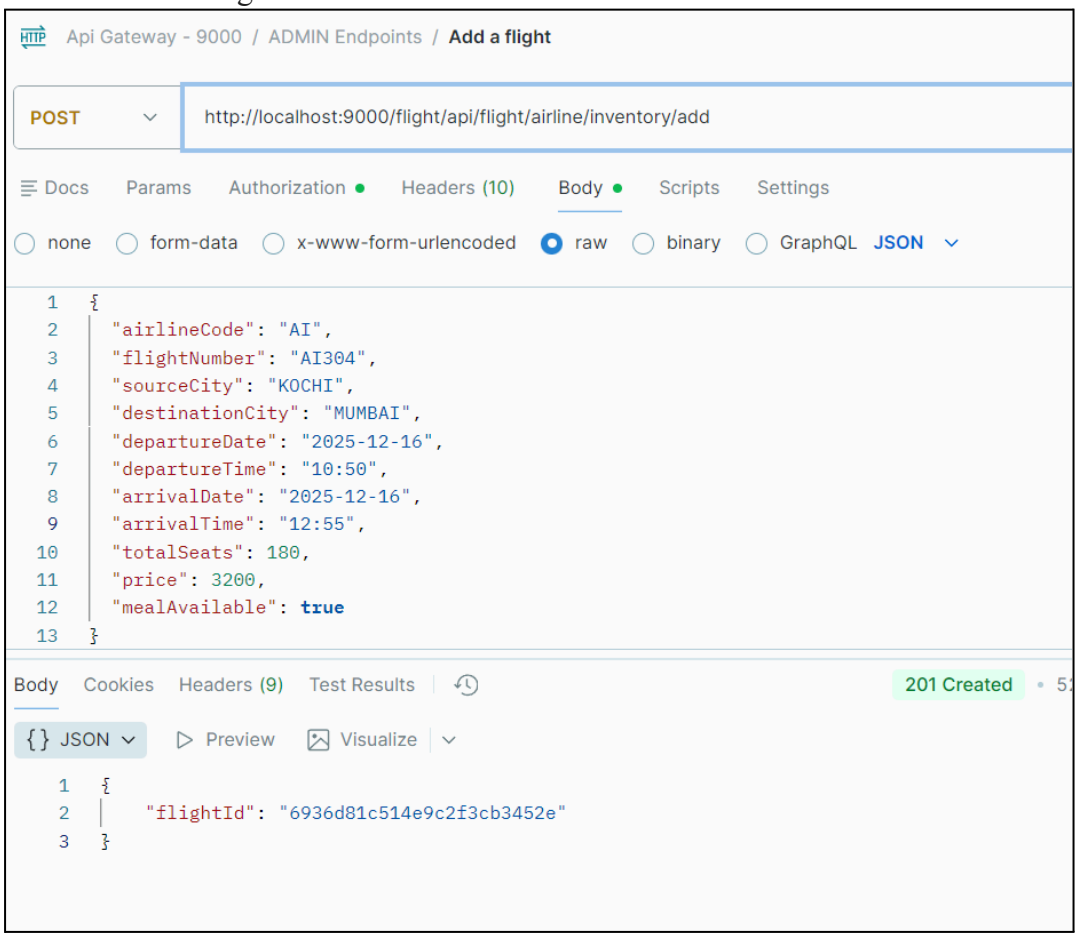


5.3 Admin Endpoints

1. POST Add an airline

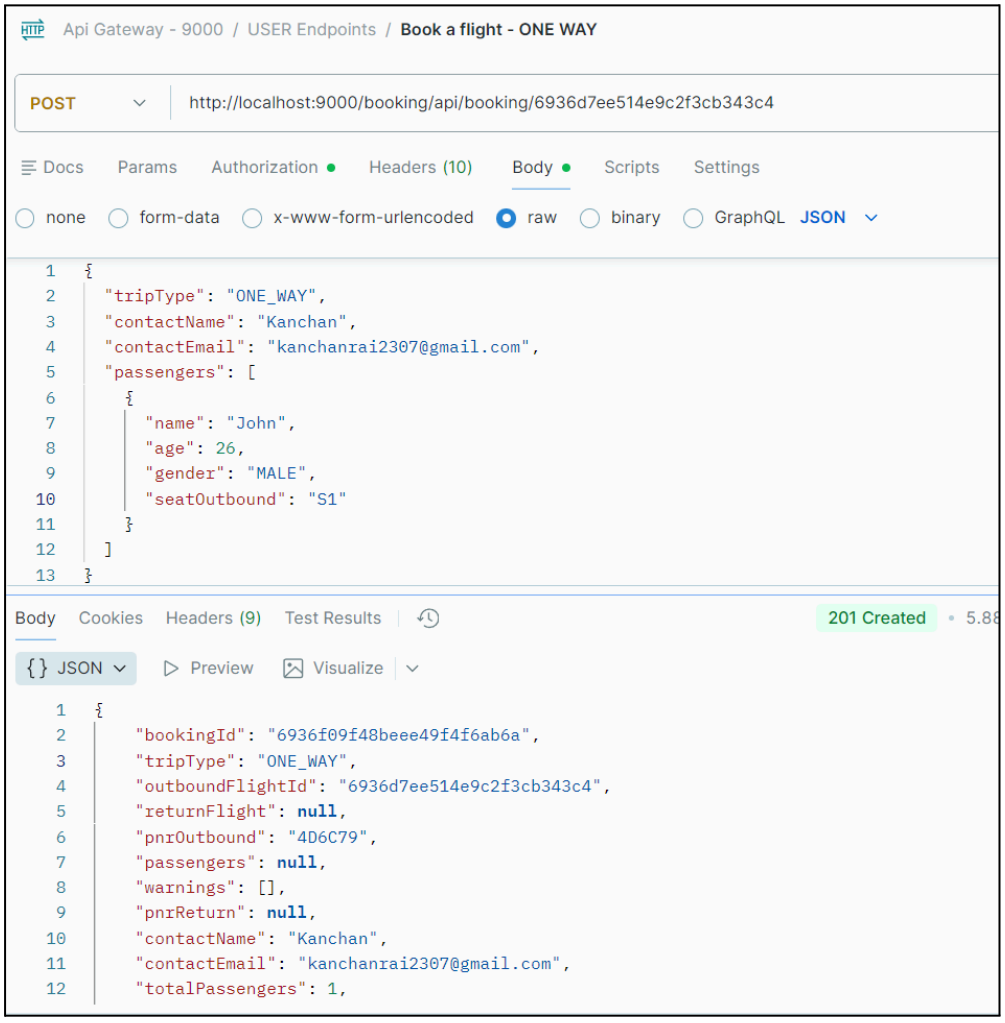


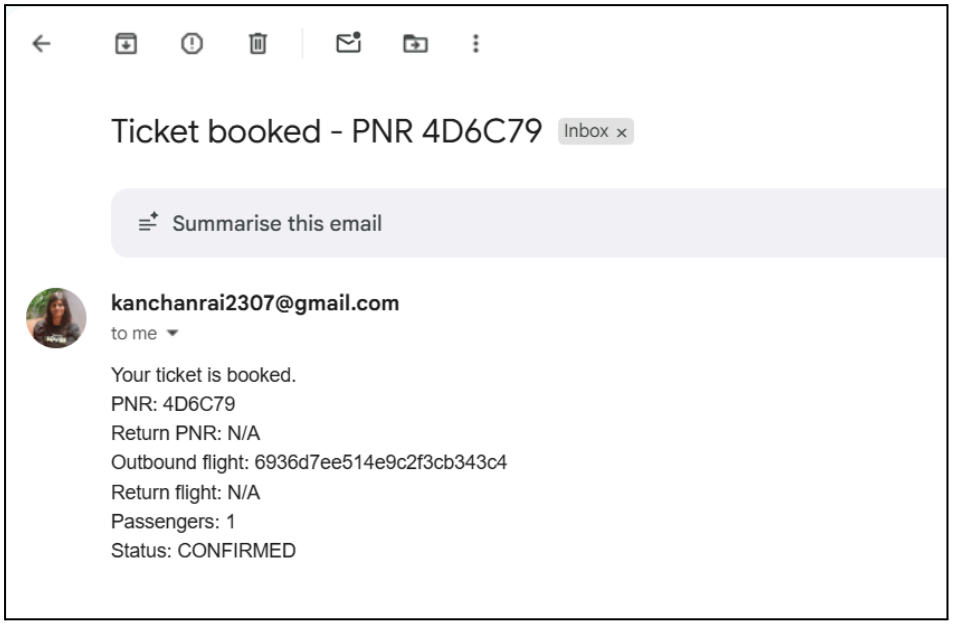
2. POST Add a flight



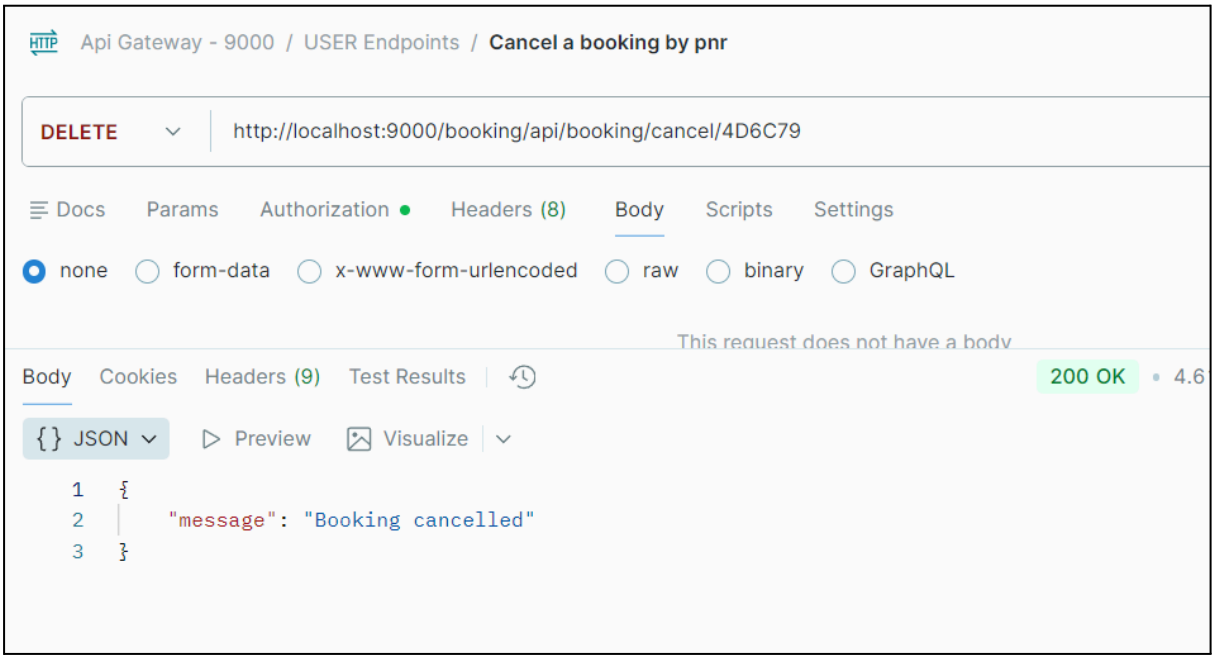
5.4 User Endpoints

1. POST Book a flight





2. DEL Cancel a flight by pnr



5.4 User + Admin points

1. GET Ticket by pnr with validation 2. GET Booking History by mail