CHUBB®

Week 5 - Project Report
By : Kanchan Rai

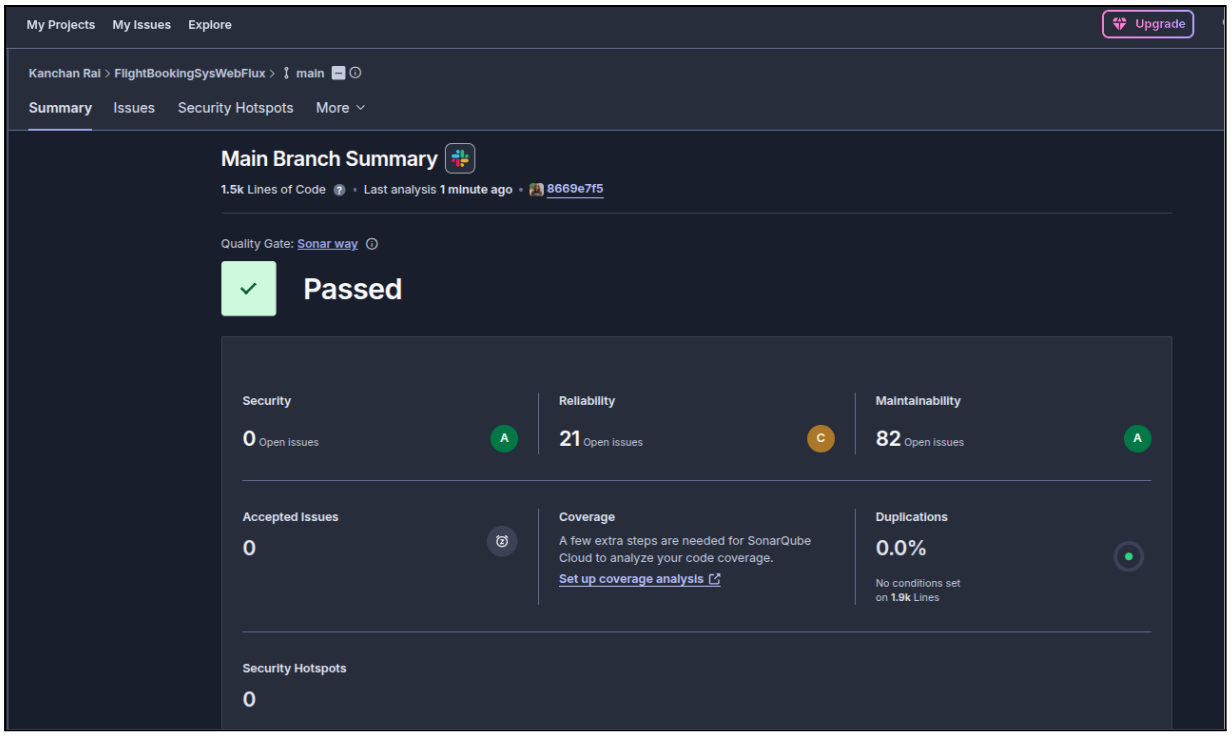
Topic : FlightBookingSystem w/ WebFlux

The following document contains the description and explanation of the project FlightBookingSystem made with WebFlux. This project has all the required validations for all end points, exception handling, unit test coverage of 90%. 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. Sonar Cube Report

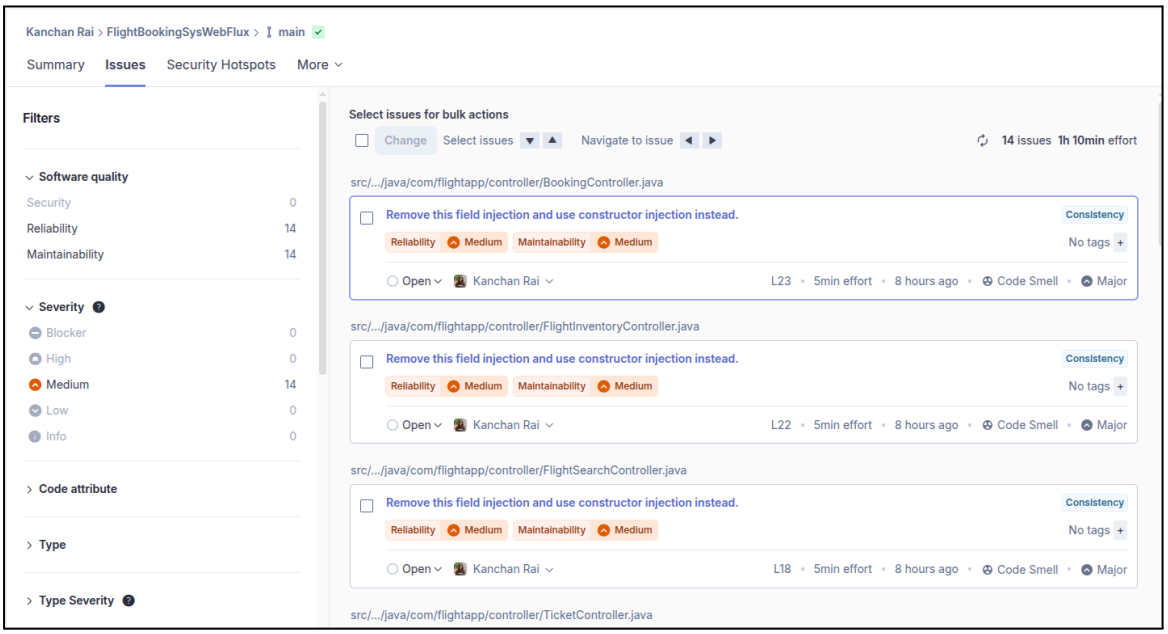
1.1 Before Fixing



21 - Reliability Issues, 82 - Maintainability Issues, 0 - Security Issues, 0.0 % Duplications, 0 Accepted Issues.

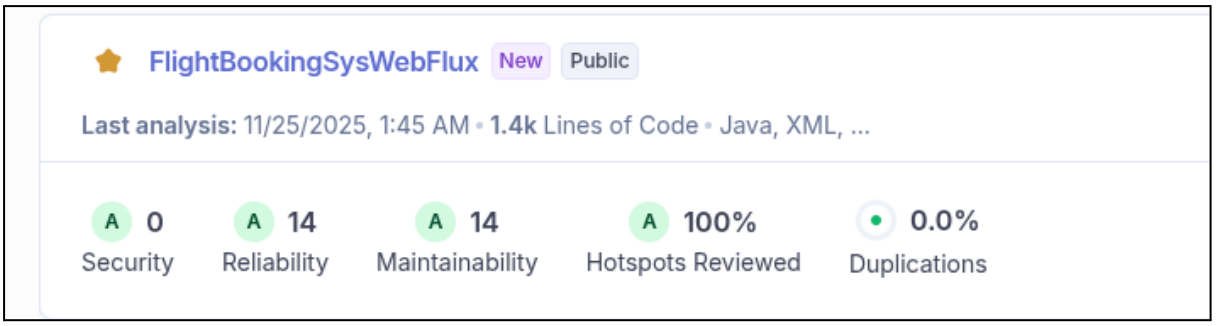
I fixed the issues in reliability and maintainability as suggested by Sonar Qube and reduced them from 87 to 14 issues. These 16 issues are related to changing the @Autowired annotation to constructor injection which I have not changed as of now.

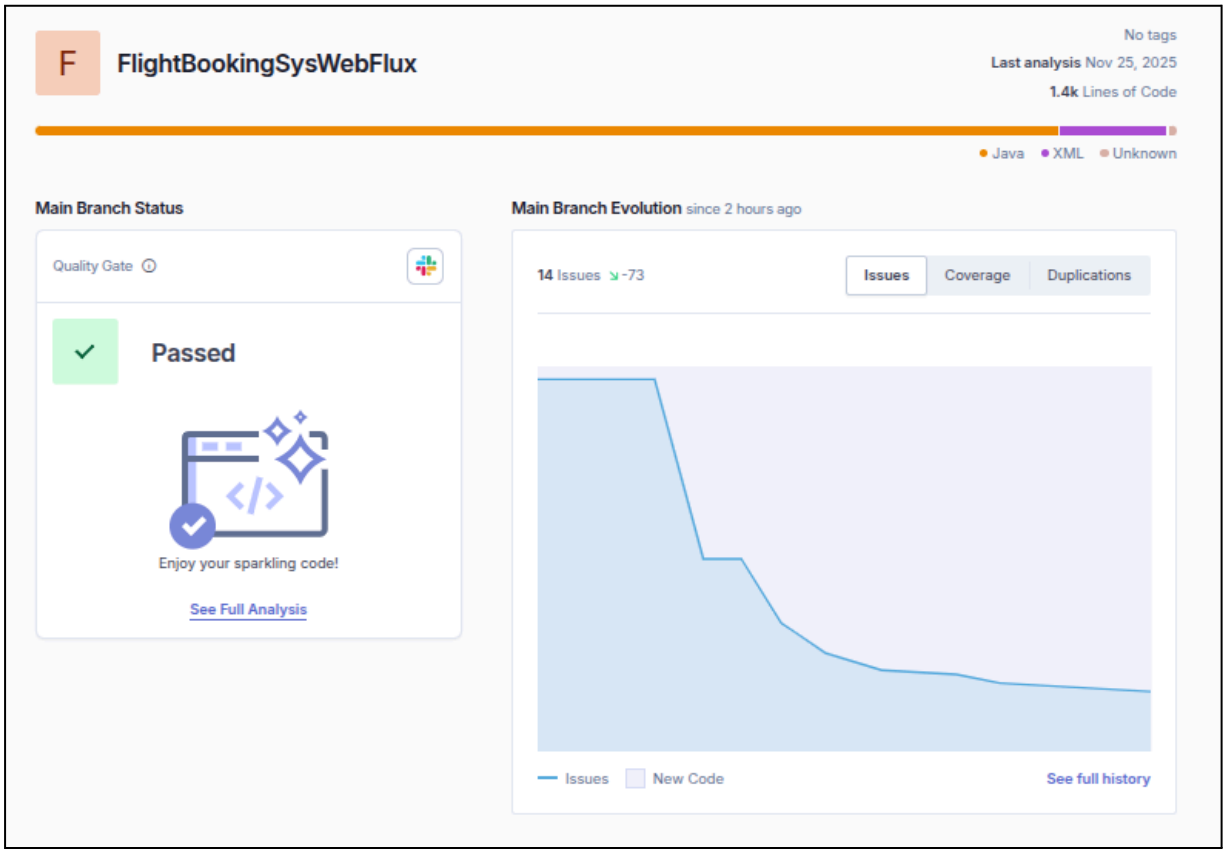
These issues present as follows -



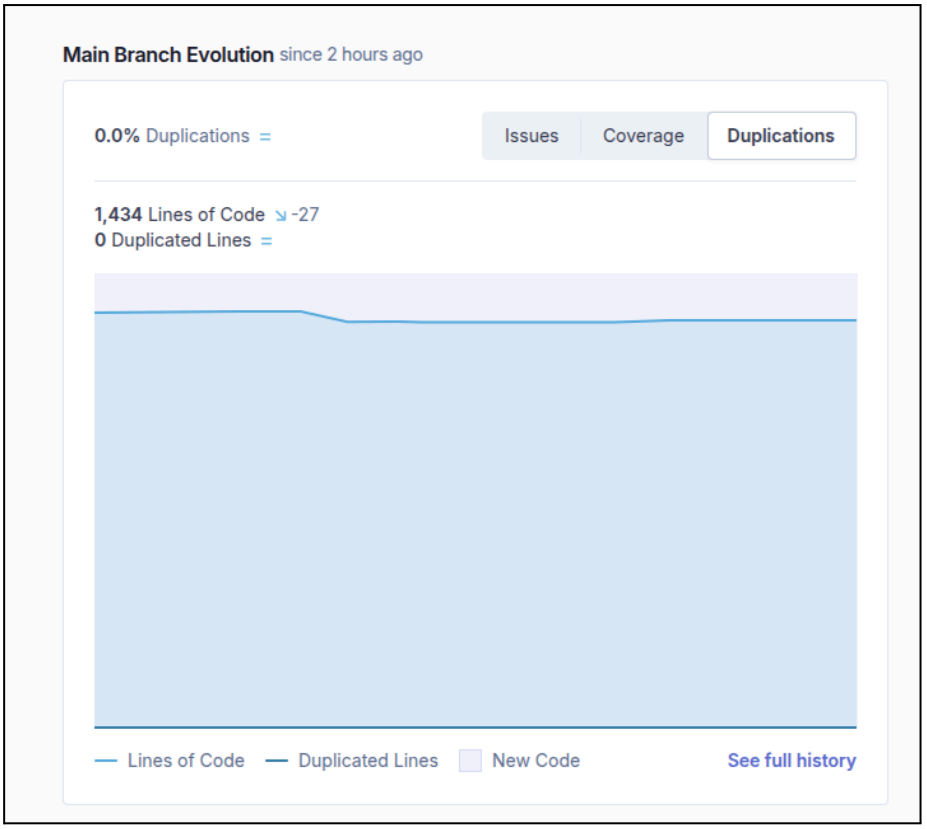
1.2 After Fixing

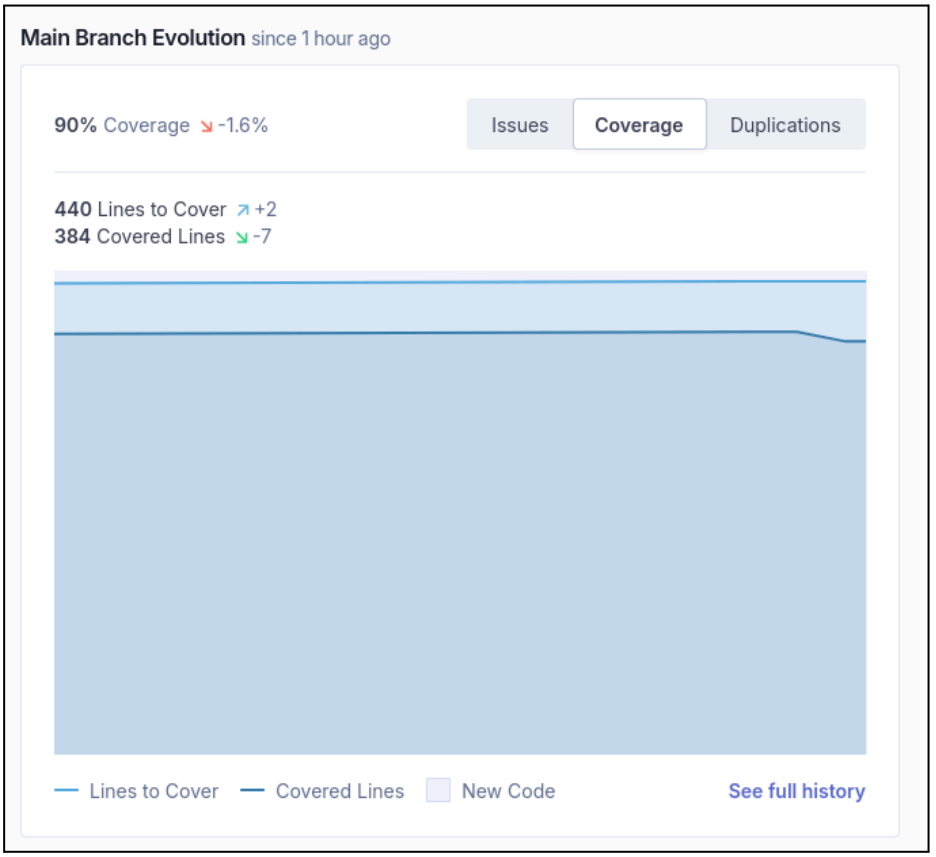
0.0% Duplications, 14 Issues, 0 Security Issues, 0 Reliability Issues





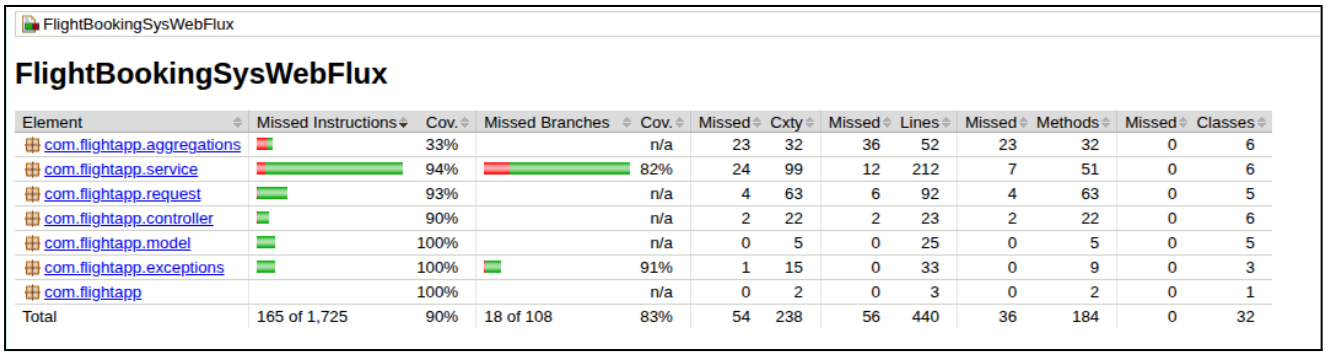
0.0% Duplications & 90 % Coverage





2. JCoco Coverage Report -

90 % Test Coverage



FlightBookingSysWebFlux

FlightBookingSysWebFlux

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.flightapp.aggregations		33%		n/a	23	32	36	52	23	32	0	6
com.flightapp.service		94%		82%	24	99	12	212	7	51	0	6
com.flightapp.request		93%		n/a	4	63	6	92	4	63	0	5
com.flightapp.controller		90%		n/a	2	22	2	23	2	22	0	6
com.flightapp.model		100%		n/a	0	5	0	25	0	5	0	5
com.flightapp.exceptions		100%		91%	1	15	0	33	0	9	0	3
com.flightapp		100%		n/a	0	2	0	3	0	2	0	1
Total	165 of 1,725	90%	18 of 108	83%	54	238	56	440	36	184	0	32

3. MongoDB Aggregations

I have also added the MongoDB Aggregations Pipeline and created GET routes for the same

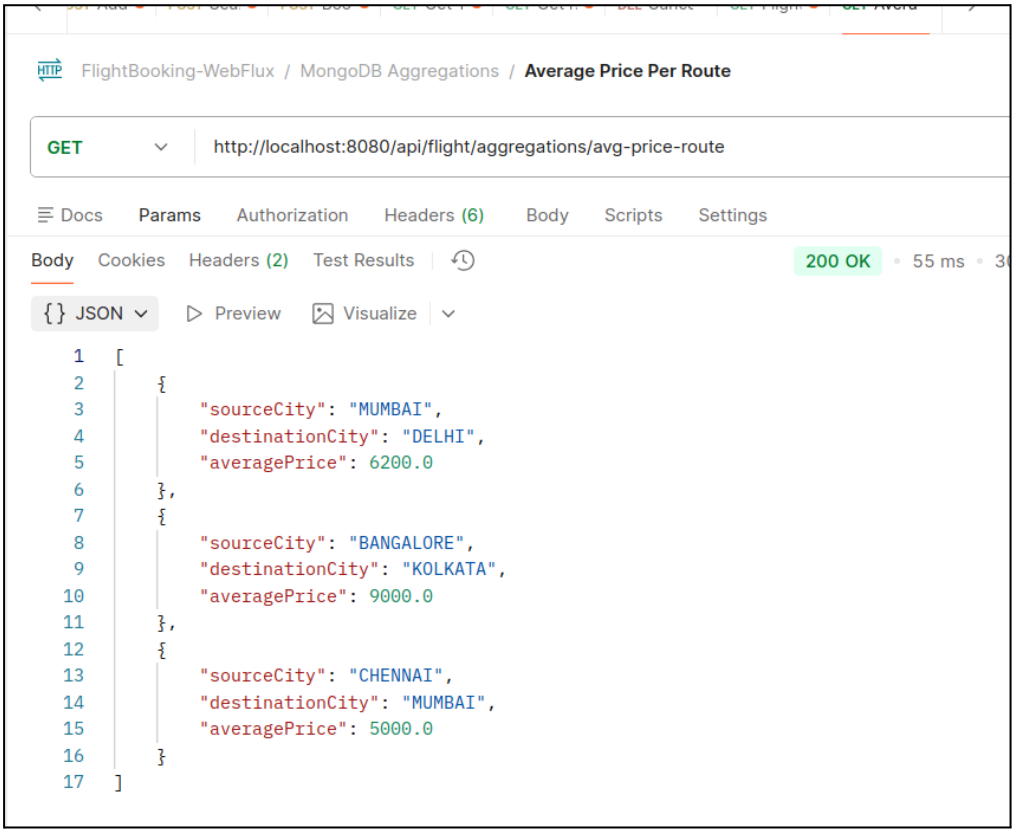
These aggregations have a separate package called `com.flightapp.aggregations`. Then a controller for the same has been made in the `com.flightapp.controller package`. The `FlightInventoryRepository` has these aggregations, like this -

```
@Aggregation(pipeline = {
    "{ $group: { _id: '$airlineCode', totalFlights: { $sum: 1 } }}",
    "{ $project: { airlineCode: '$_id', totalFlights: 1, _id: 0 }}"
})
Flux<AirlineFlightCount> getFlightsPerAirline();


@Aggregation(pipeline = {
    "{ $group: { _id: '$airlineCode', totalAvailableSeats: { $sum: '$availableSeats' } }}",
    "{ $project: { airlineCode: '$_id', totalAvailableSeats: 1, _id: 0 }}"
})
Flux<AirlineSeats> getSeatStatsPerAirline();


@Aggregation(pipeline = {
    "{ $sort: { price: -1 }}",
    "{ $limit: 5 }",
    "{ $project: { flightNumber: 1, price: 1, _id: 0 }}"
})
Flux<HighestPriceFlights> getTopExpensiveFlights();


@Aggregation(pipeline = {
    "{ $group: { _id: { src: '$sourceCity', dest: '$destinationCity' }, avgPrice: { $avg: '$price' } }}",
    "{ $project: { sourceCity: '$_id.src', destinationCity: '$_id.dest', averagePrice: '$avgPrice', _id: 0 }}"
})
Flux<RoutePrices> getAveragePricePerRoute();
```

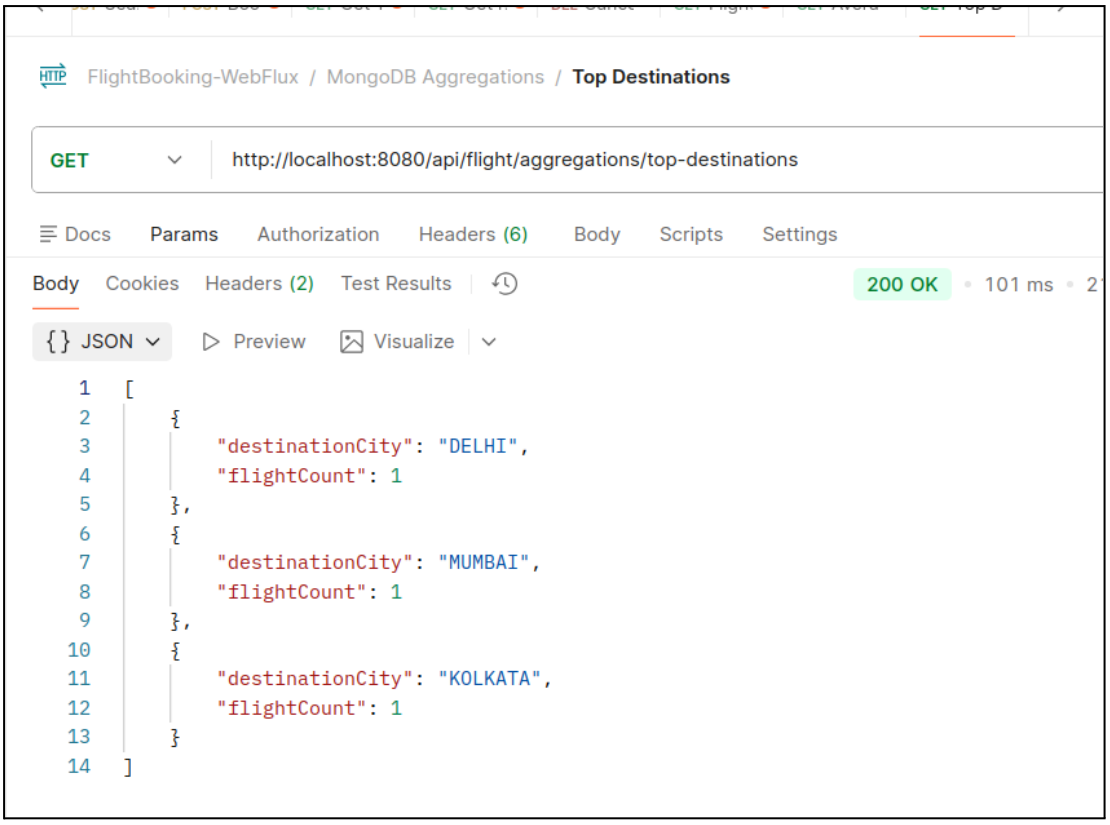
Each aggregation result is as follows -

1. GET flights per air line

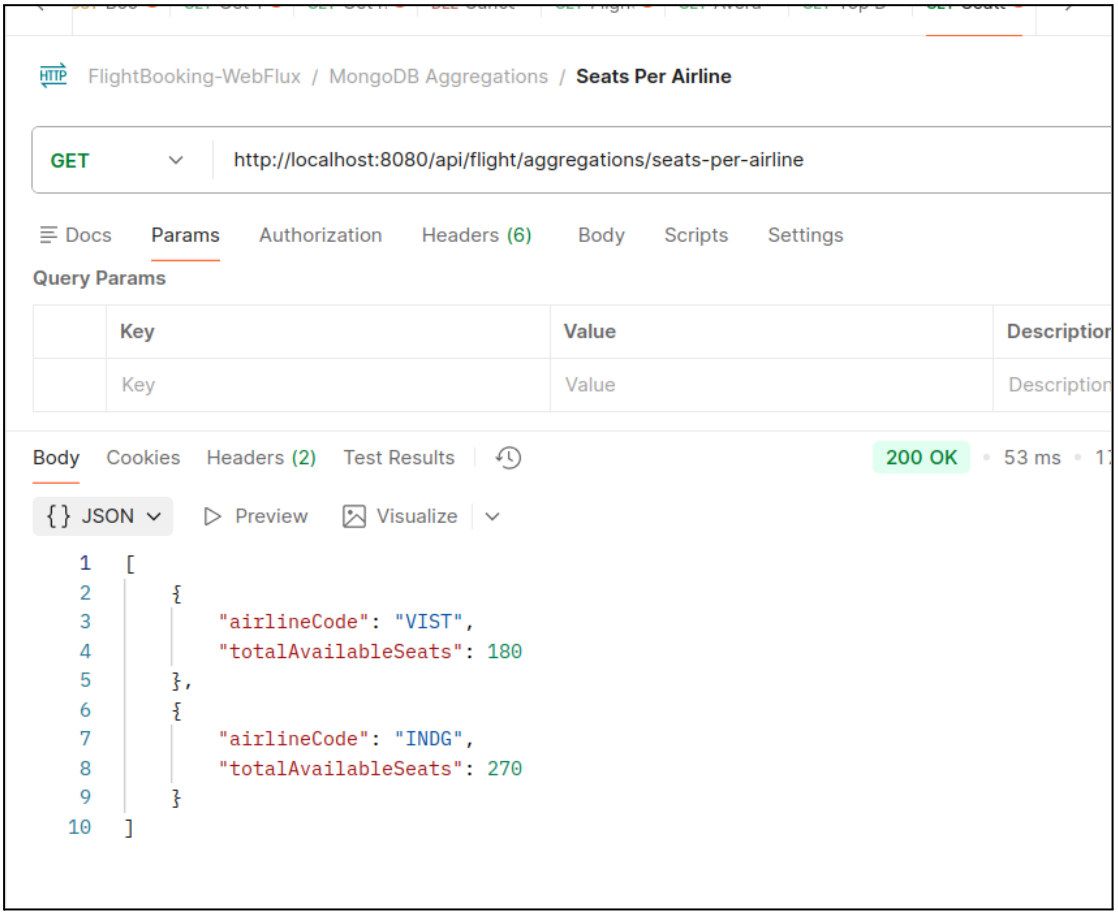
2. GET Average price per route



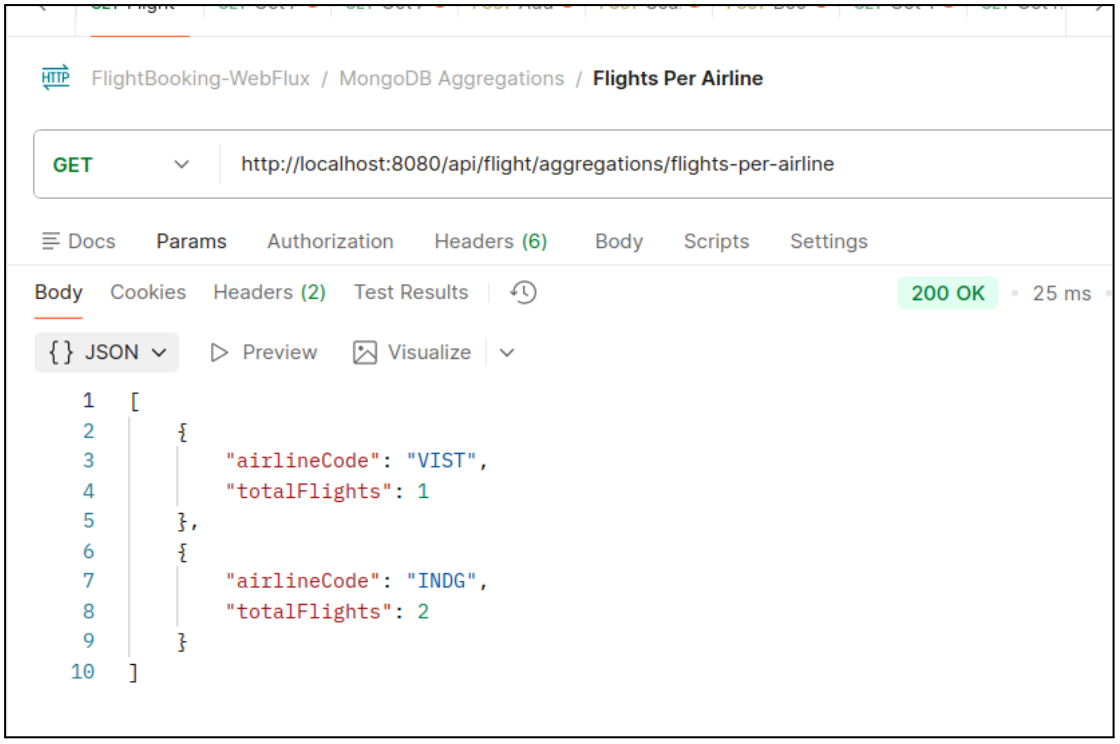
3. GET top destinations



4. GET Seats per airline



5. GET Flights per air line

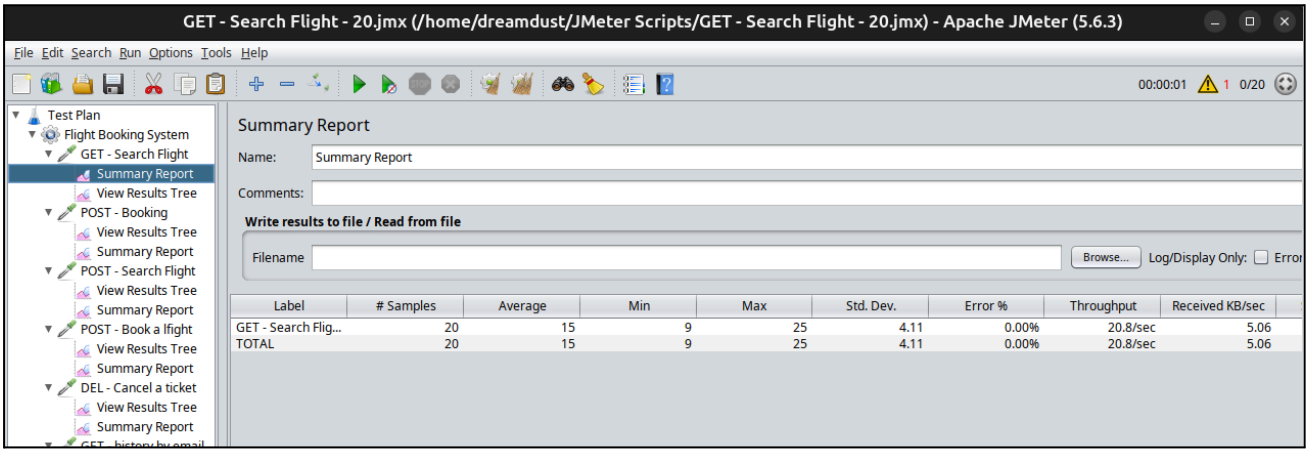


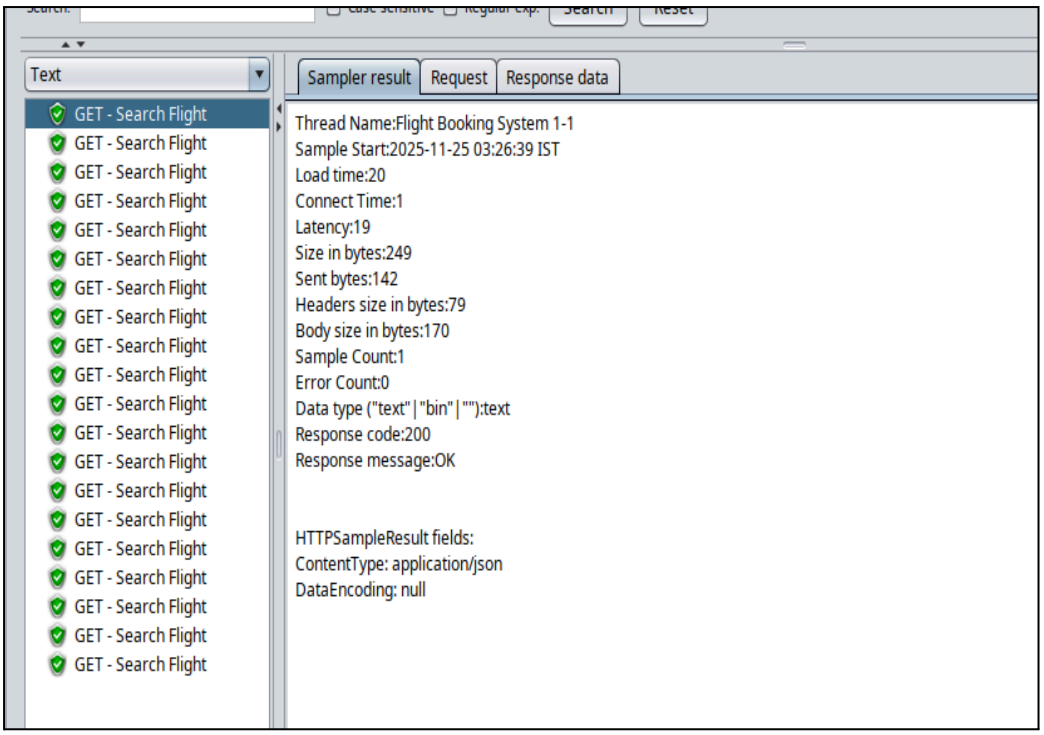
4. JMeter Load Testing

4.1 With 20 Threads

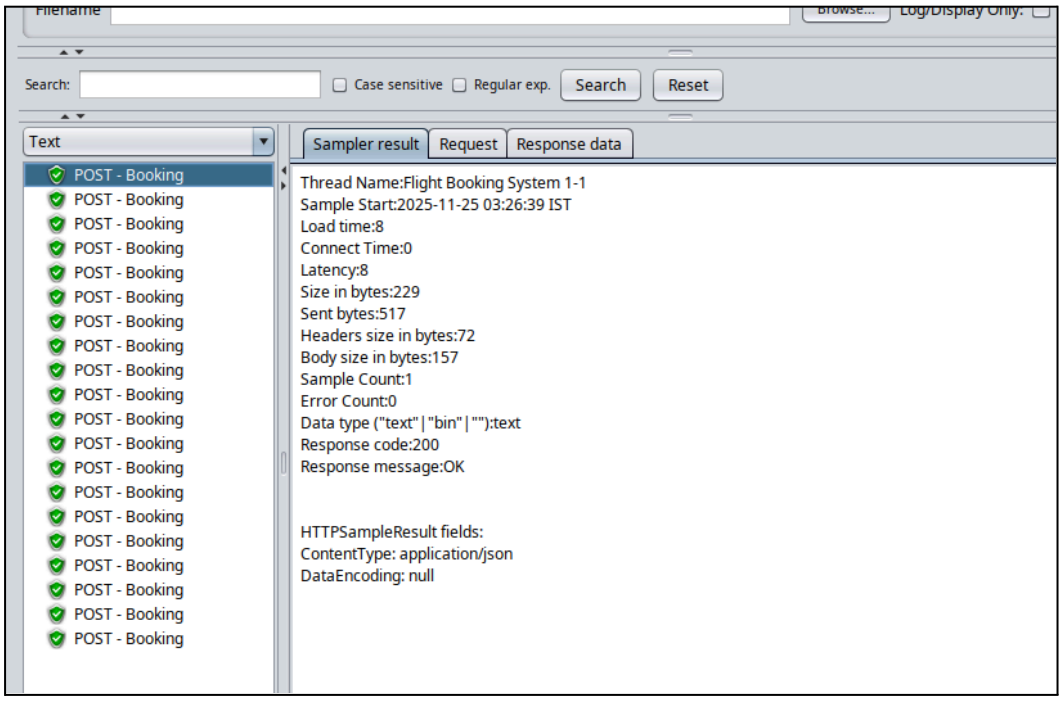
The load testing has been carried out for all endpoints. Each one of GET,POST,DELETE has been shown below.

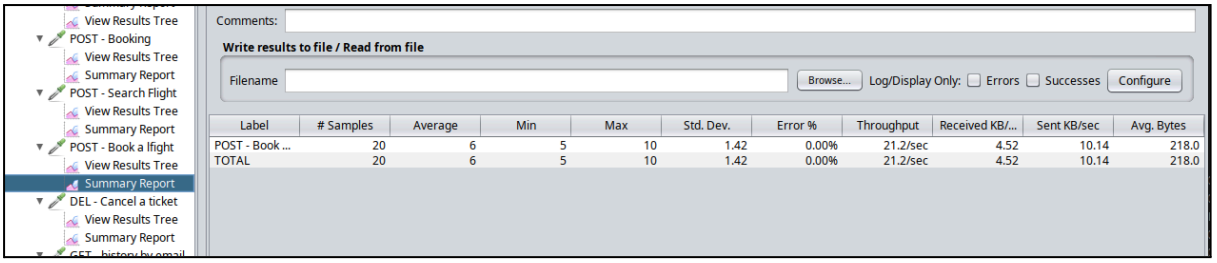
GET - Search a flight



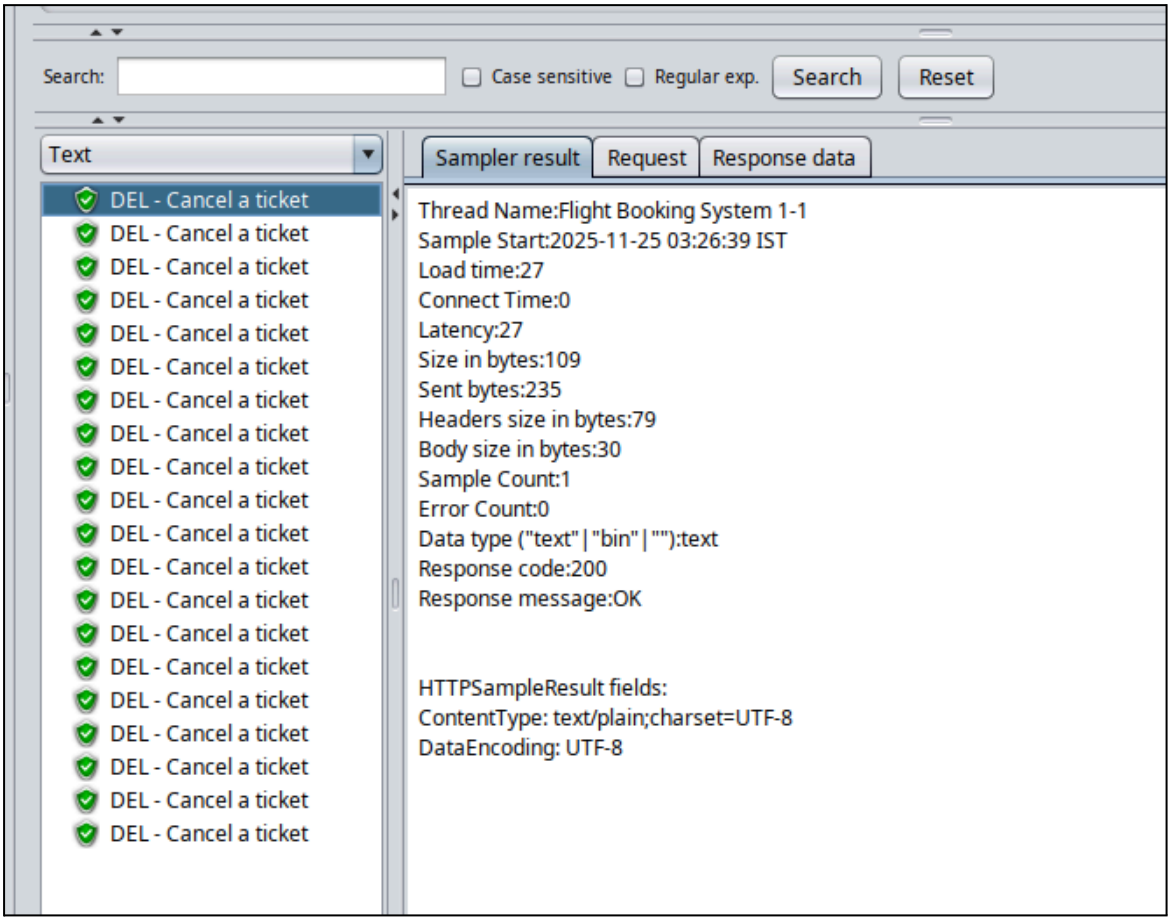


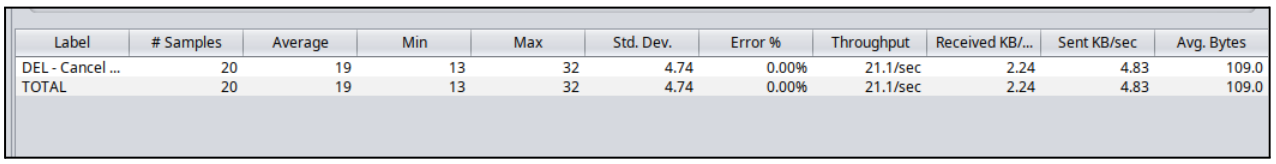
POST - Book a flight





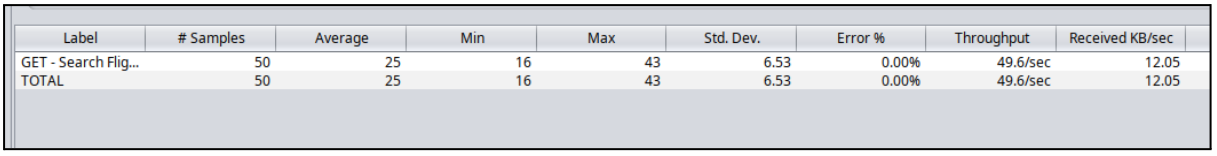
DELETE - Cancel a Booking



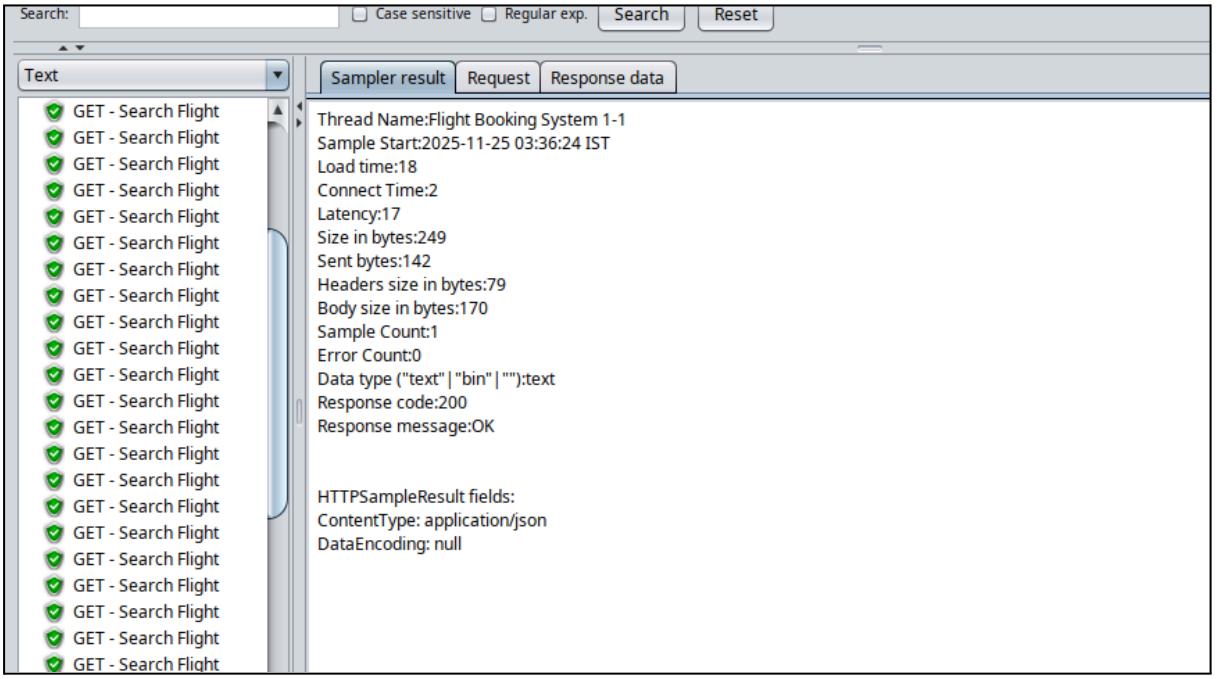


4.2 With 50 Threads

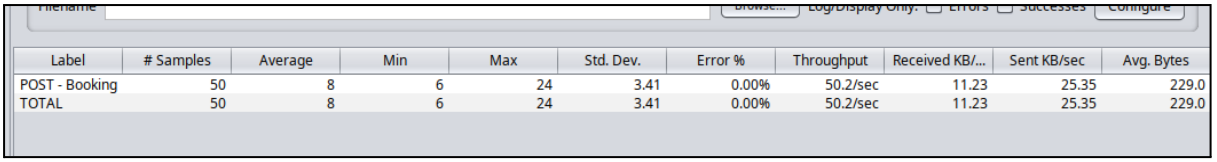
GET - Search a flight



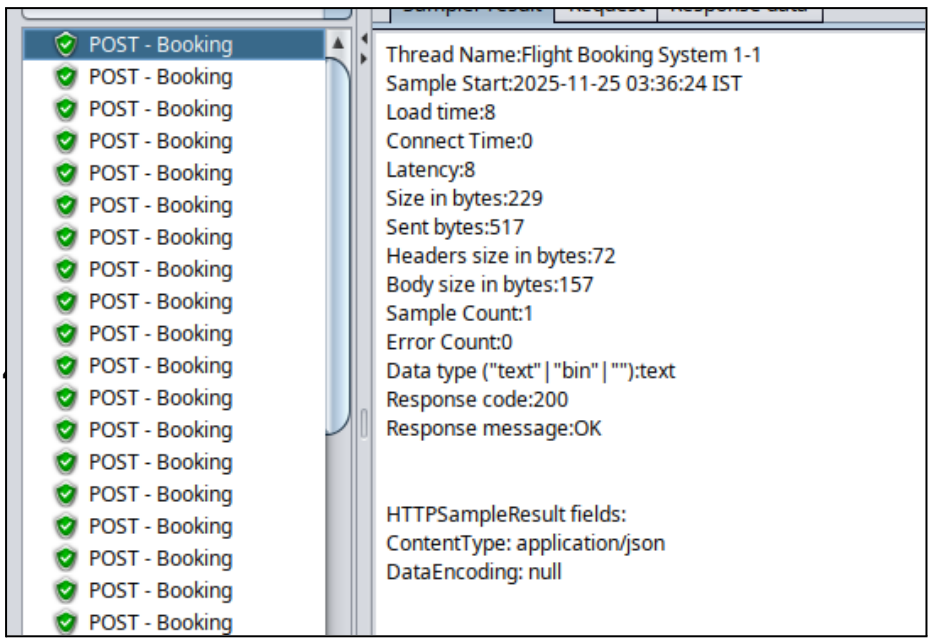
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec
GET - Search Flig...	50	25	16	43	6.53	0.00%	49.6/sec	12.05
TOTAL	50	25	16	43	6.53	0.00%	49.6/sec	12.05



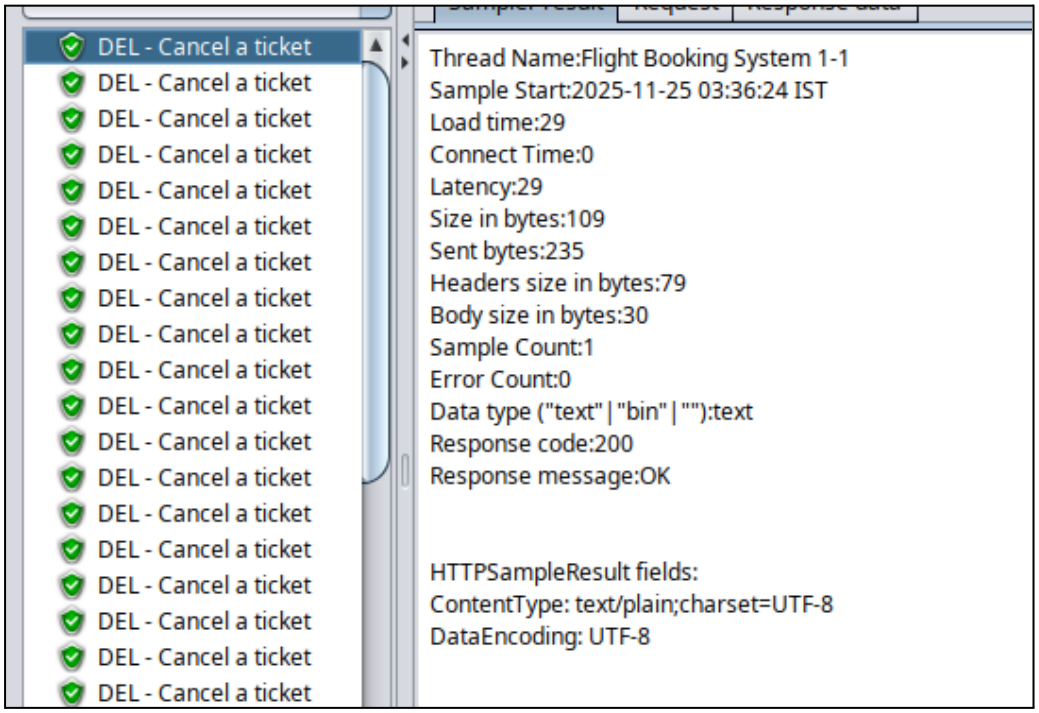
POST - Book a flight



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
POST - Booking	50	8	6	24	3.41	0.00%	50.2/sec	11.23	25.35	229.0
TOTAL	50	8	6	24	3.41	0.00%	50.2/sec	11.23	25.35	229.0



DELETE - Cancel a booking

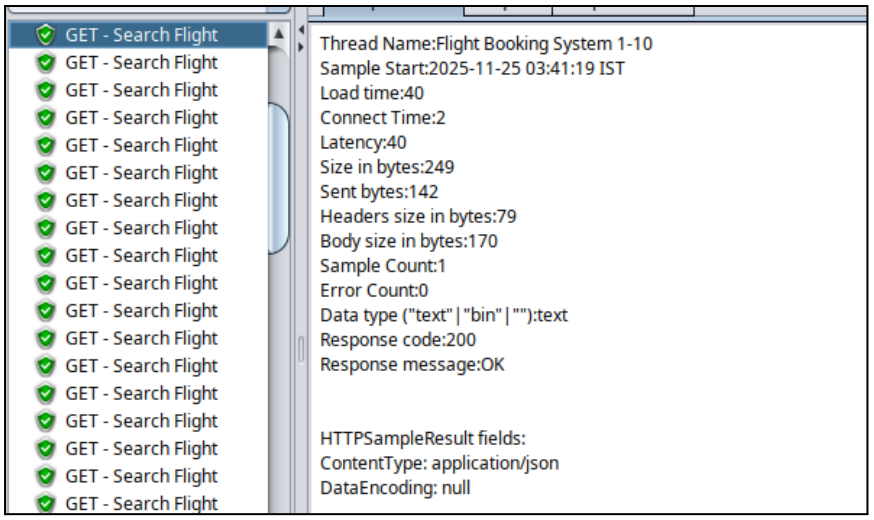


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
DEL - Cancel ...	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0
TOTAL	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0

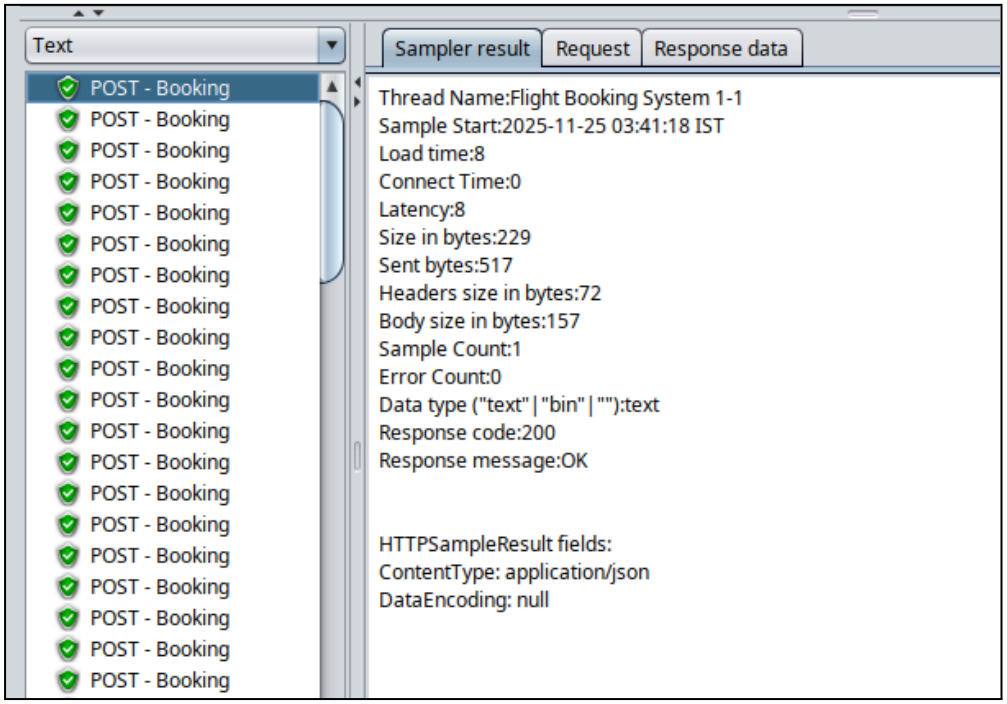
4.3 With <u>100</u> Threads

GET - Search a flight

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec
GET - Search Flig...	100	171	17	432	106.10	0.00%	74.3/sec	18.07
TOTAL	100	171	17	432	106.10	0.00%	74.3/sec	18.07

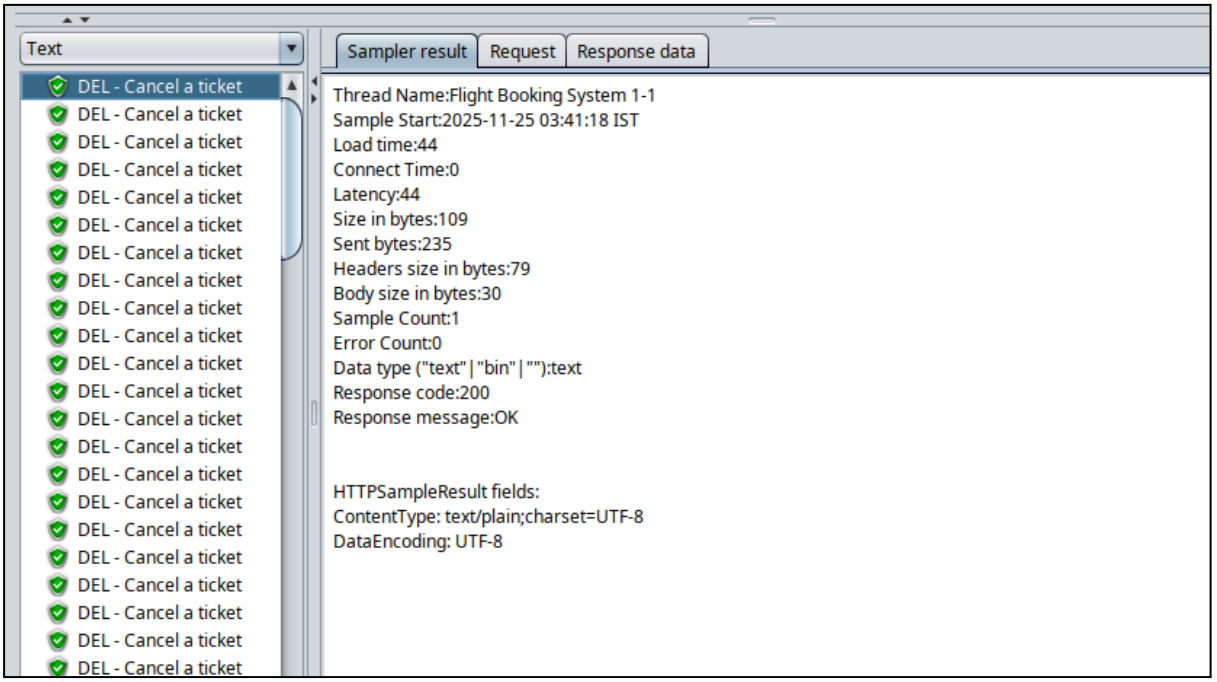


POST - Book a flight



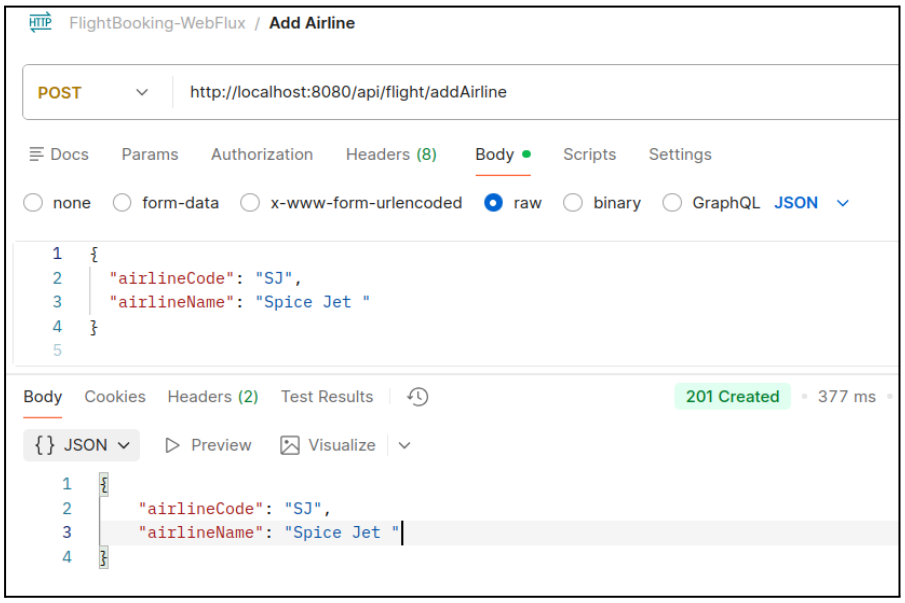
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
POST - Booking	100	25	5	91	17.71	0.00%	73.3/sec	16.40	37.01	229.0
TOTAL	100	25	5	91	17.71	0.00%	73.3/sec	16.40	37.01	229.0

DELETE - Cancel a Booking

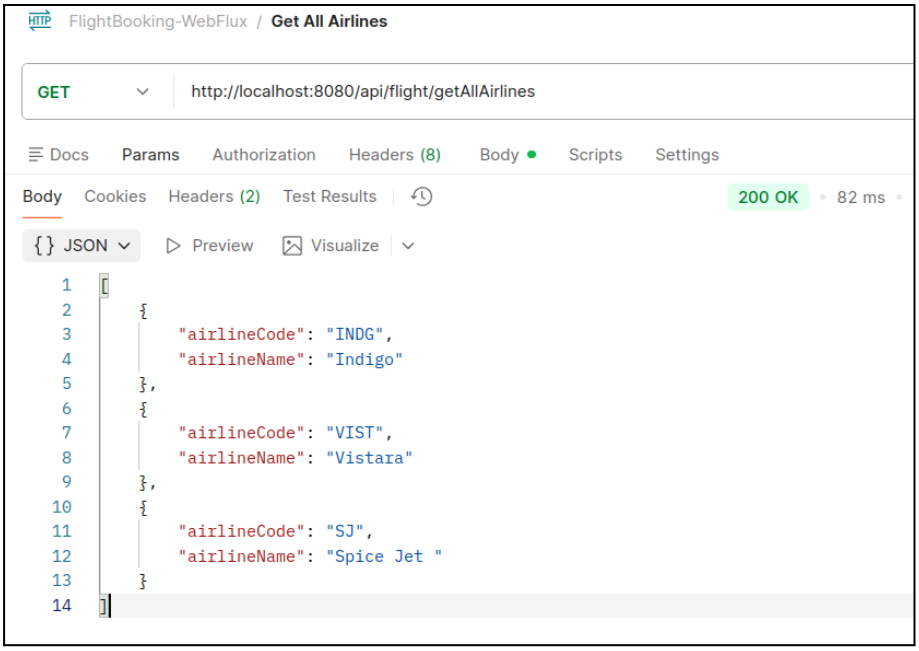


5. All API Endpoints Testing & Results

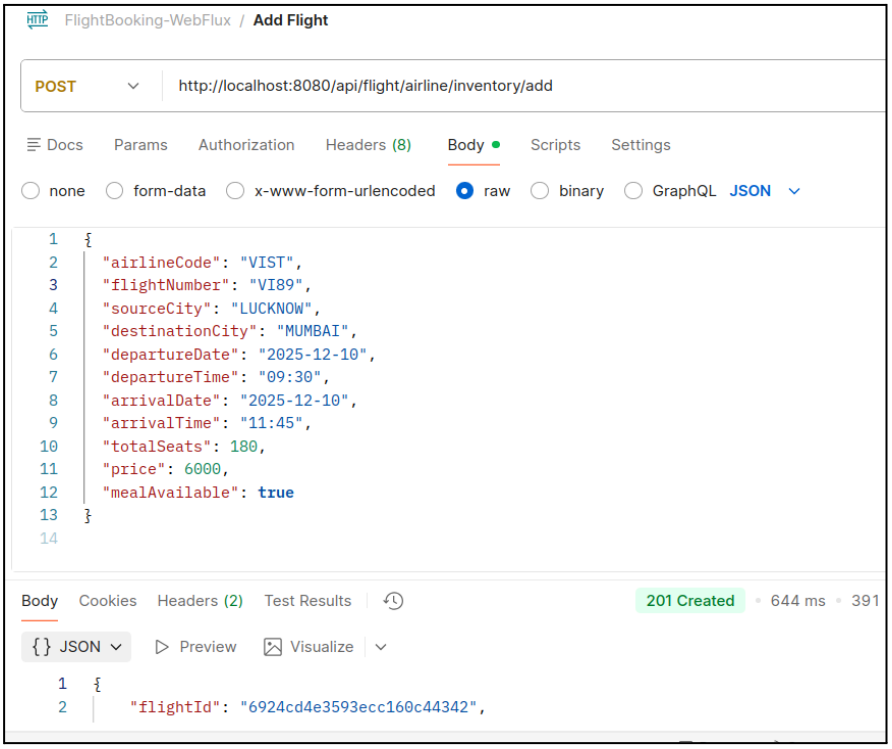
1. POST http://localhost:8080/api/flight/addAirline - Add Airline



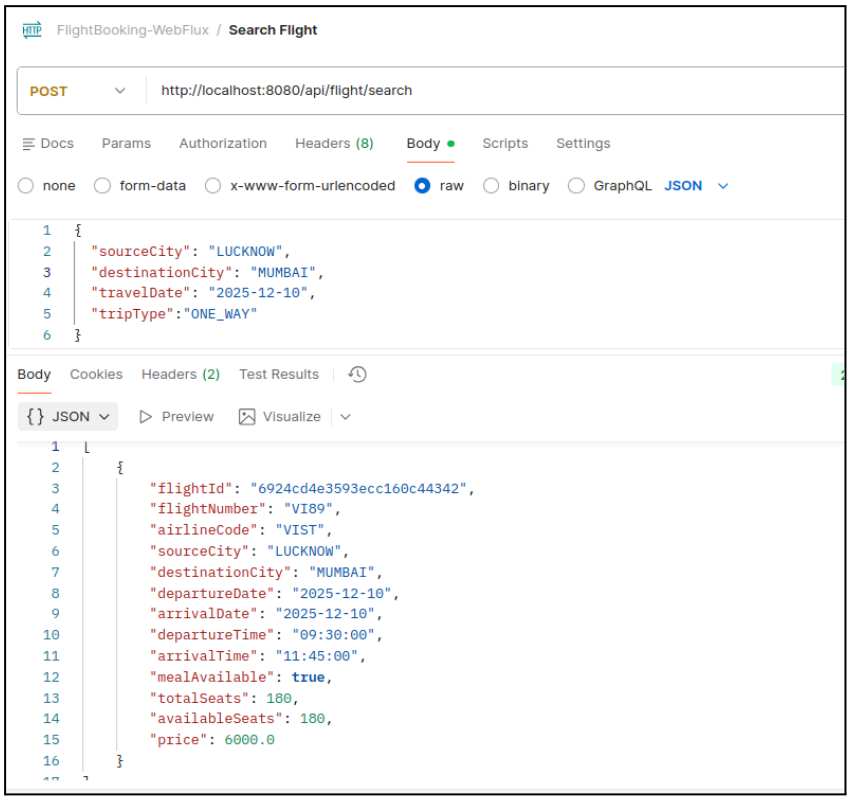
2. GET http://localhost:8080/api/flight/getAllAirlines - Get All Airlines



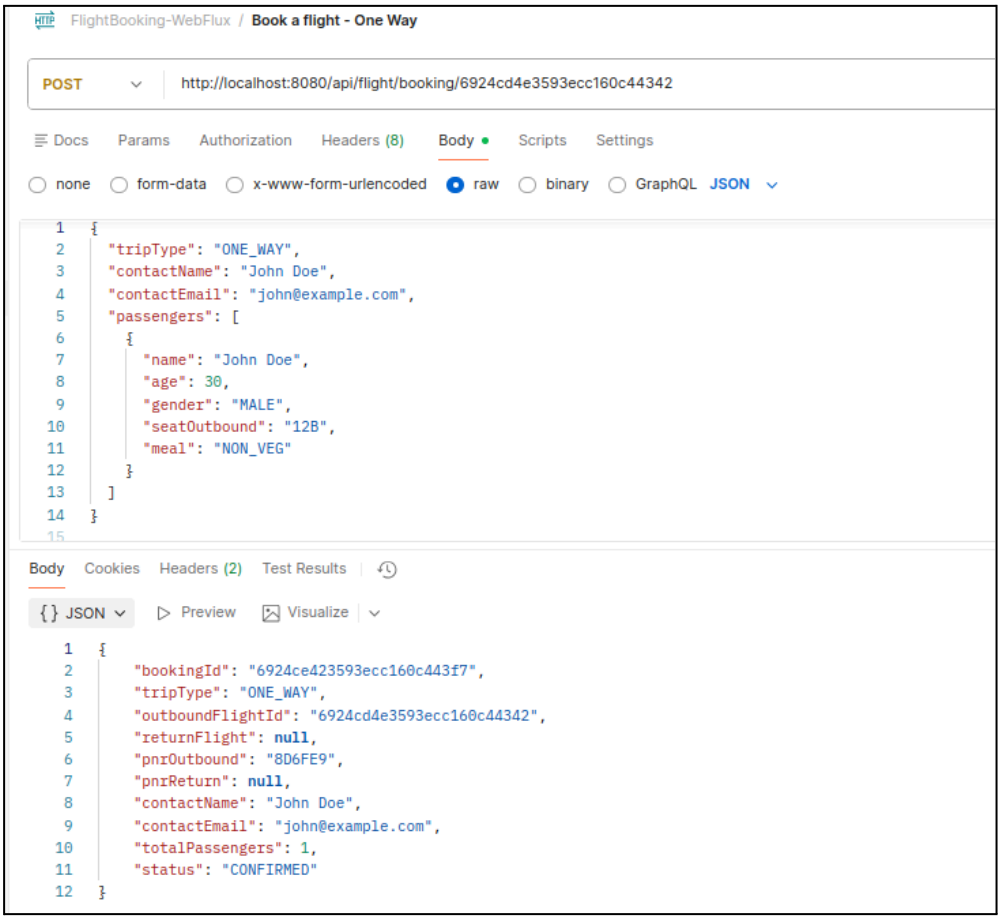
3. POST http://localhost:8080/api/flight/airline/inventory/add - Add Flight



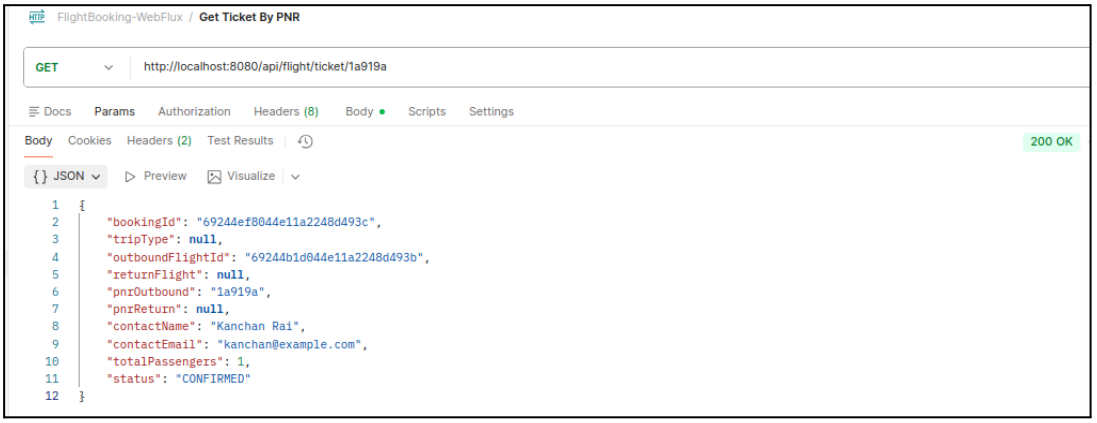
4. POST http://localhost:8080/api/flight/search - Search Flight



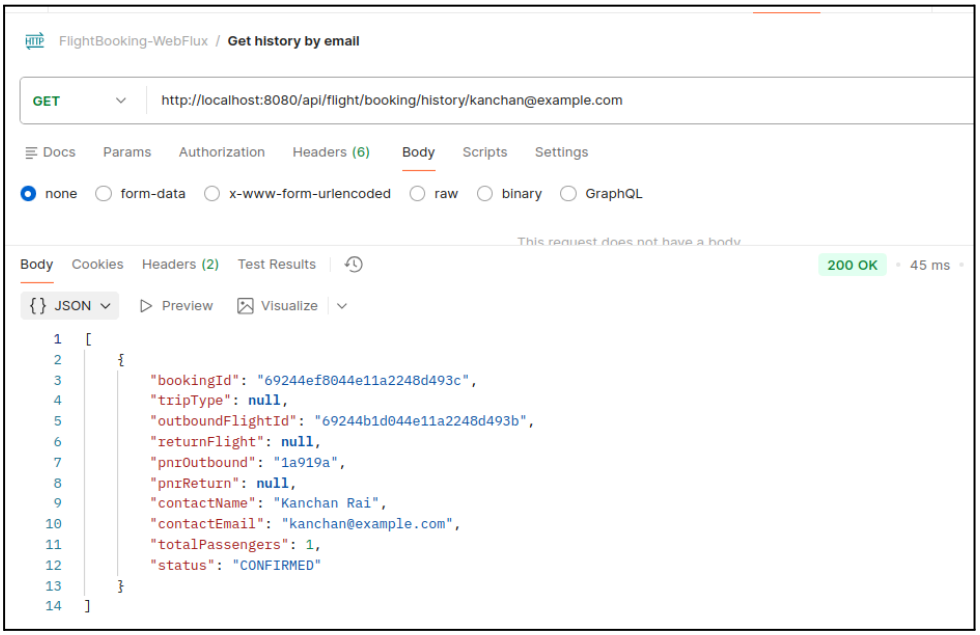
5. POST http://localhost:8080/api/flight/booking/6924cd4e3593ecc160c44342 - Book a flight



6. GET http://localhost:8080/api/flight/ticket/1a919a - Get Ticket by pnr



7. GET - History by email



8. DELETE - Cancel a ticket