



Week 5 - Project Report

By : Kanchan Rai

Topic : FlightBookingSystem w/ WebFlux

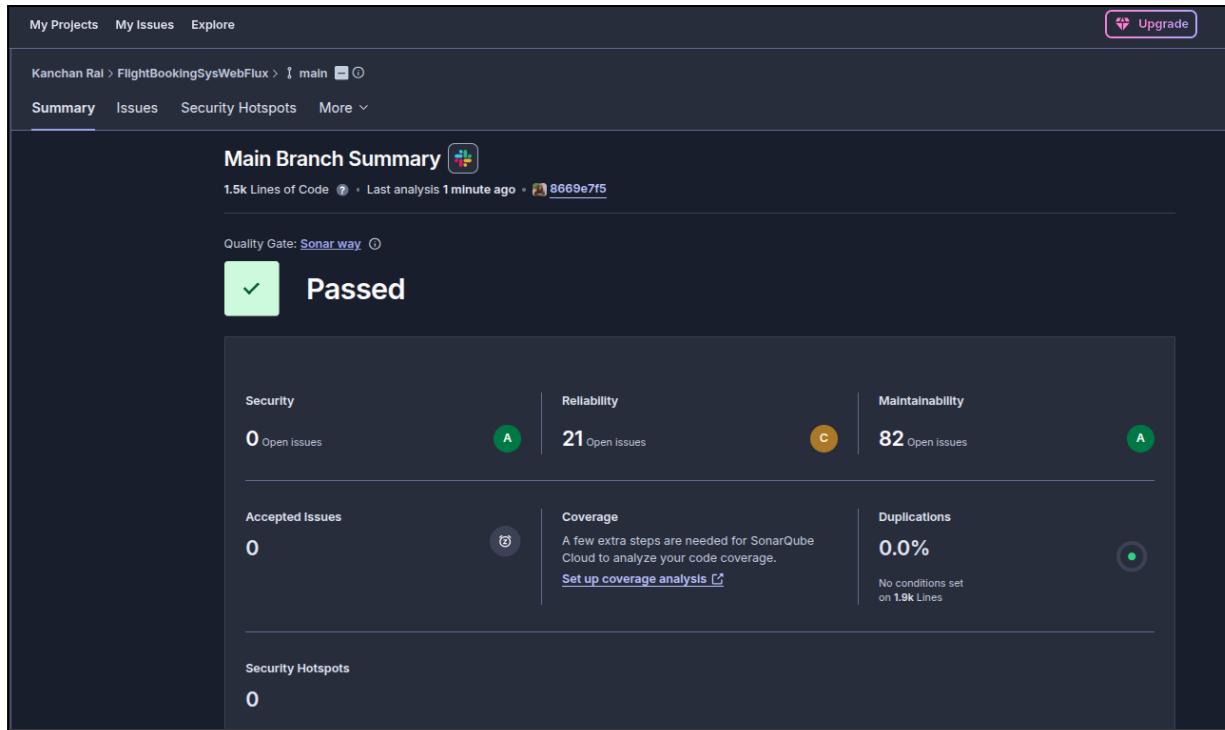
The following document contains the description and explanation of the project FlightBookingSystem made with WebFlux. This project has all the required validations for all end points, exception handling, unit test coverage of 90%. Load testing carried out as well for all endpoints. The database used is MongoDB. And Reactive Programming has been carried out as well.

INDEX

1. Sonar Cube Report	3
1.1 Before Fixing	3
1.2 After Fixing	4
2. JCoco Coverage Report	6
3. MongoDB Aggregations	6
4. JMeter Load Testing	9
4.1 With 20 Threads	9
4.1 With 50 Threads	12
4.1 With 100 Threads	13
5. All API Endpoints Testing & Results	15

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.

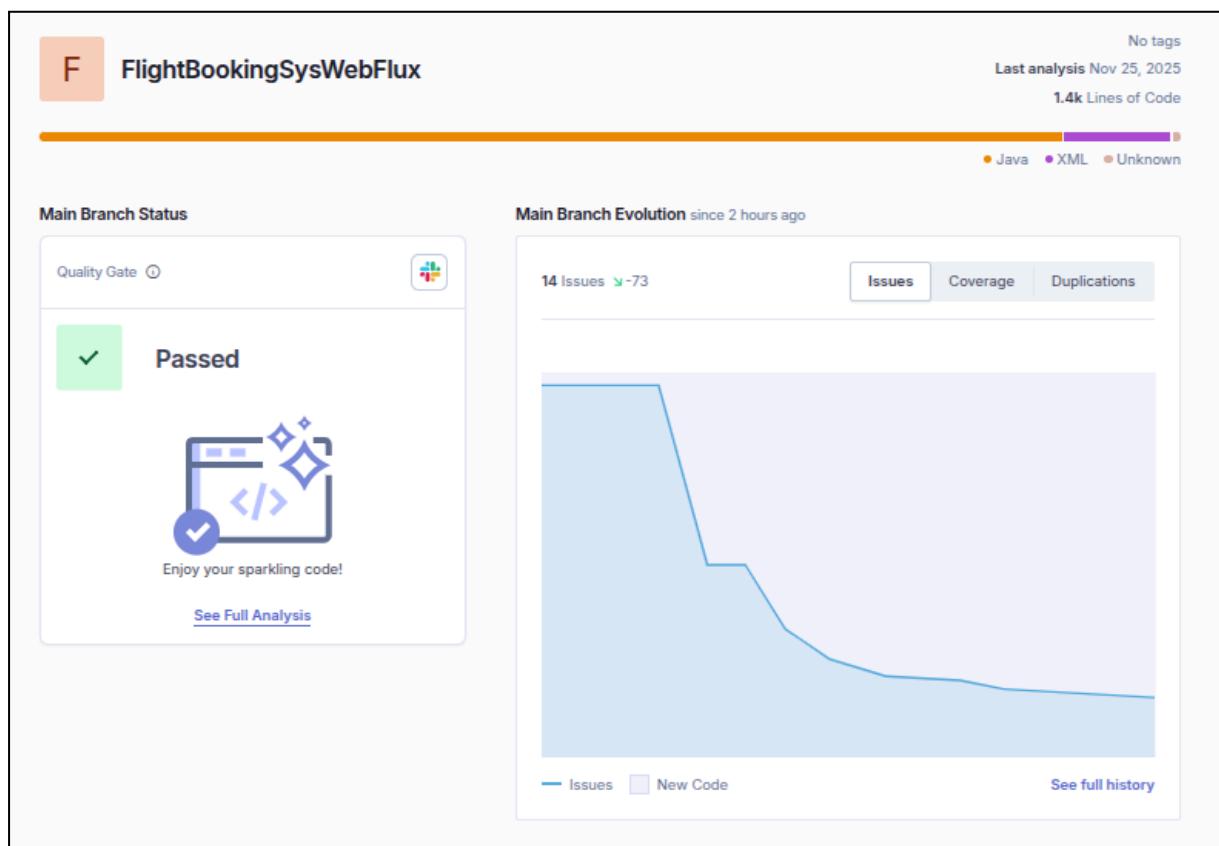
These issues present as follows -

The screenshot shows the SonarQube 'Issues' tab for the same project. The list of issues is as follows:

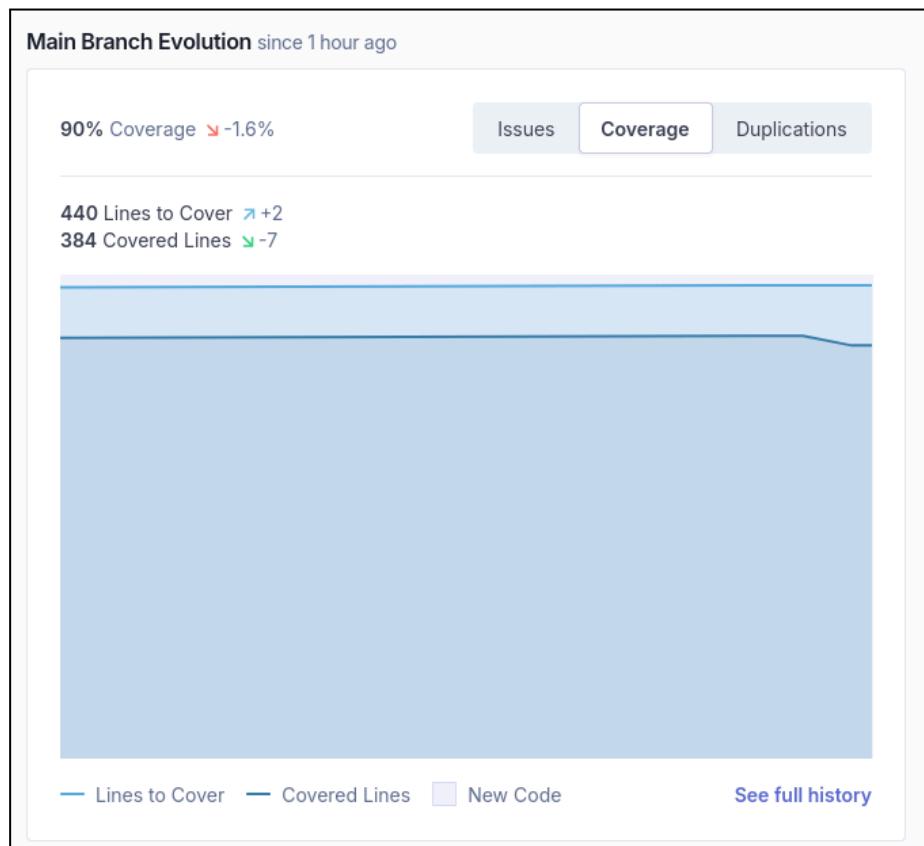
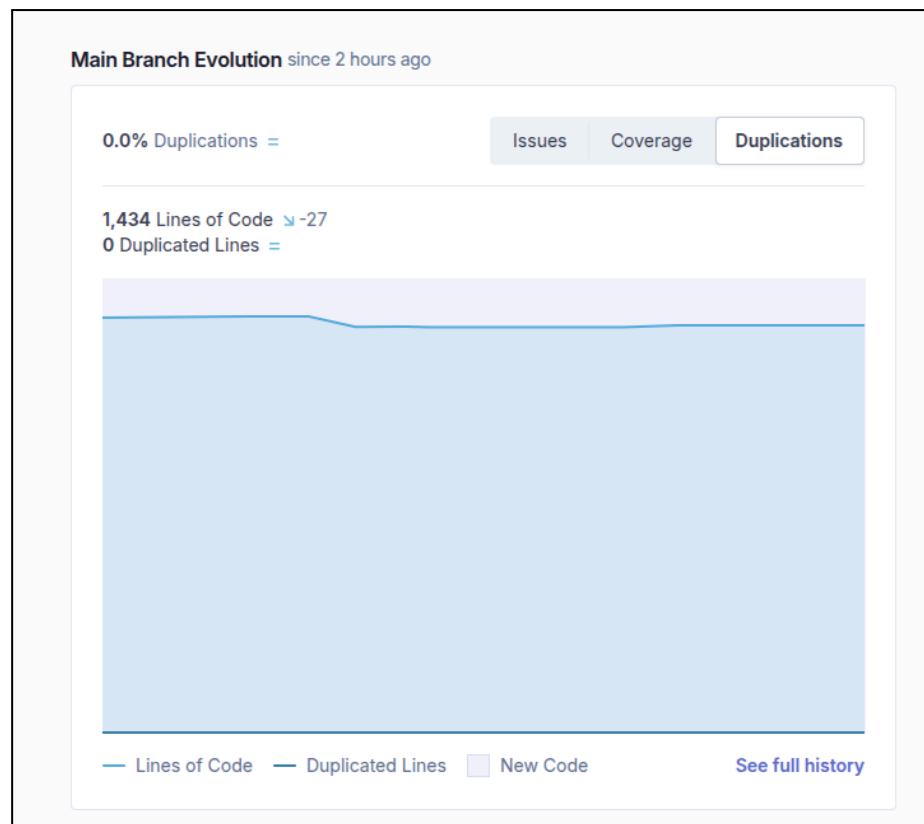
- src/.../java/com/flightapp/controller/BookingController.java:
 - Remove this field injection and use constructor injection instead. (Reliability Medium, Maintainability Medium, Consistency No tags) - L23, 5min effort, 8 hours ago, Code Smell, Major
- src/.../java/com/flightapp/controller/FlightInventoryController.java:
 - Remove this field injection and use constructor injection instead. (Reliability Medium, Maintainability Medium, Consistency No tags) - L22, 5min effort, 8 hours ago, Code Smell, Major
- src/.../java/com/flightapp/controller/FlightSearchController.java:
 - Remove this field injection and use constructor injection instead. (Reliability Medium, Maintainability Medium, Consistency No tags) - L18, 5min effort, 8 hours ago, Code Smell, Major
- src/.../java/com/flightapp/controller/TicketController.java: (No issues listed)

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

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.flighthapp.aggregations	33%	n/a	23	32	36	52	23	32	0	6	0	6
com.flighthapp.service	94%	82%	24	99	12	212	7	51	0	6	0	6
com.flighthapp.request	93%	n/a	4	63	6	92	4	63	0	5	0	5
com.flighthapp.controller	90%	n/a	2	22	2	23	2	22	0	6	0	6
com.flighthapp.model	100%	n/a	0	5	0	25	0	5	0	5	0	5
com.flighthapp.exceptions	100%	91%	1	15	0	33	0	9	0	3	0	1
com.flighthapp	100%	n/a	0	2	0	3	0	2	0	1	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.flighthapp.aggregations. Then a controller for the same has been made in the com.flighthapp.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();

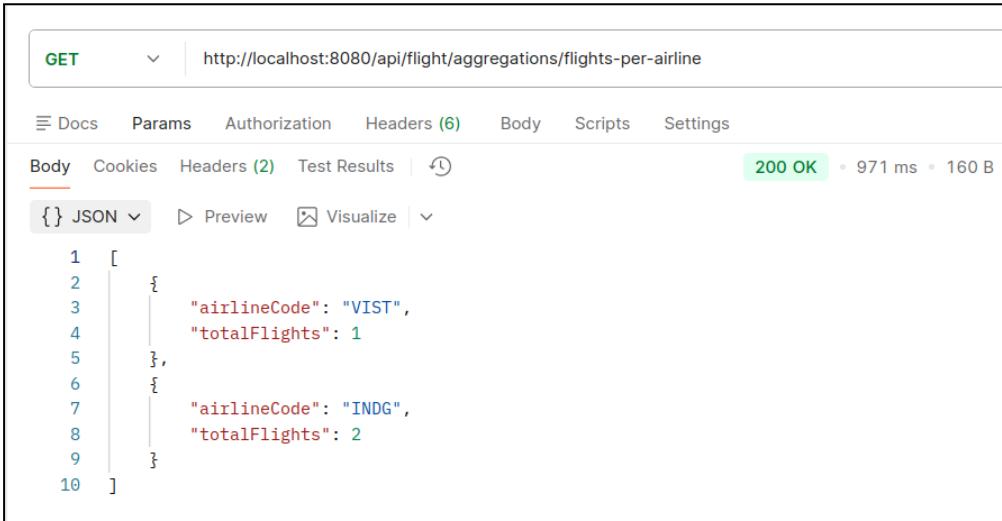
@Aggregation(pipeline = {
    "{$group: { _id: '$airlineCode', totalAvailableSeats: { $sum: '$availableSeats' } }}",
    "{$project: { airlineCode: '_id', totalAvailableSeats: 1, _id: 0 }}"
})
Flux<AirlineSeats> getSeatStatsPerAirline();

@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 -

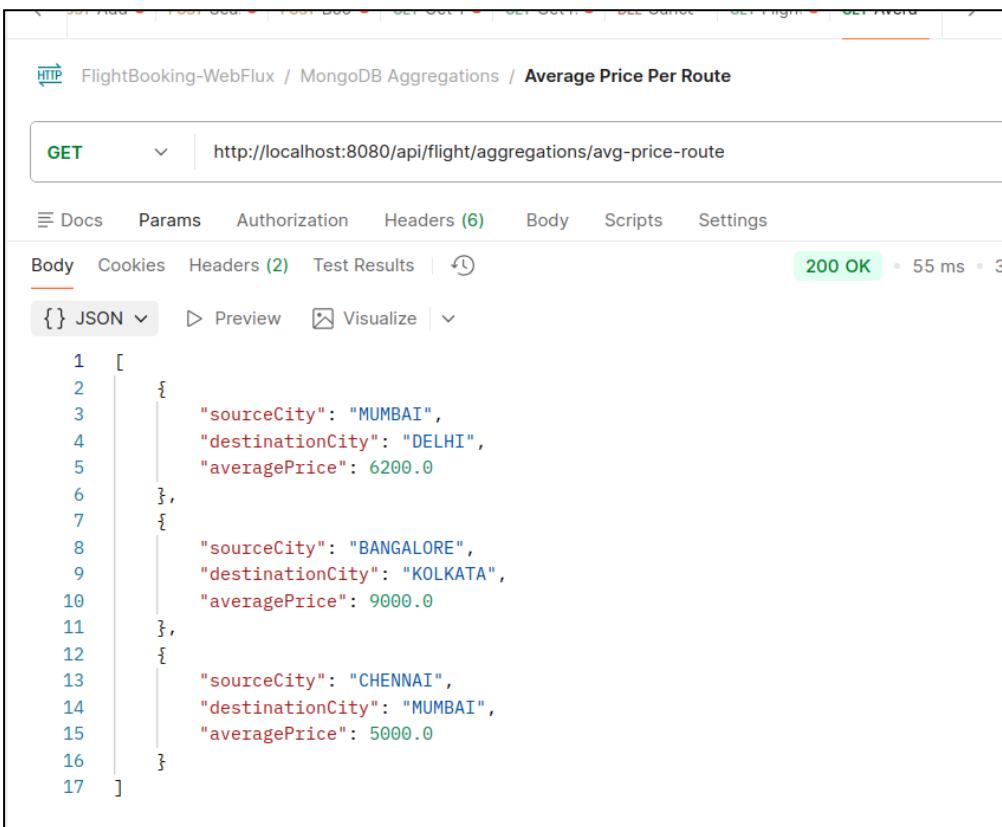
1. GET flights per air line



The screenshot shows a Postman collection named "FlightBooking-WebFlux / MongoDB Aggregations". A GET request is made to `http://localhost:8080/api/flight/aggregations/flights-per-airline`. The response status is 200 OK, with a response time of 971 ms and a body size of 160 B. The response is a JSON array containing two elements, each representing an airline code and its total flights.

```
[{"airlineCode": "VIST", "totalFlights": 1}, {"airlineCode": "INDG", "totalFlights": 2}]
```

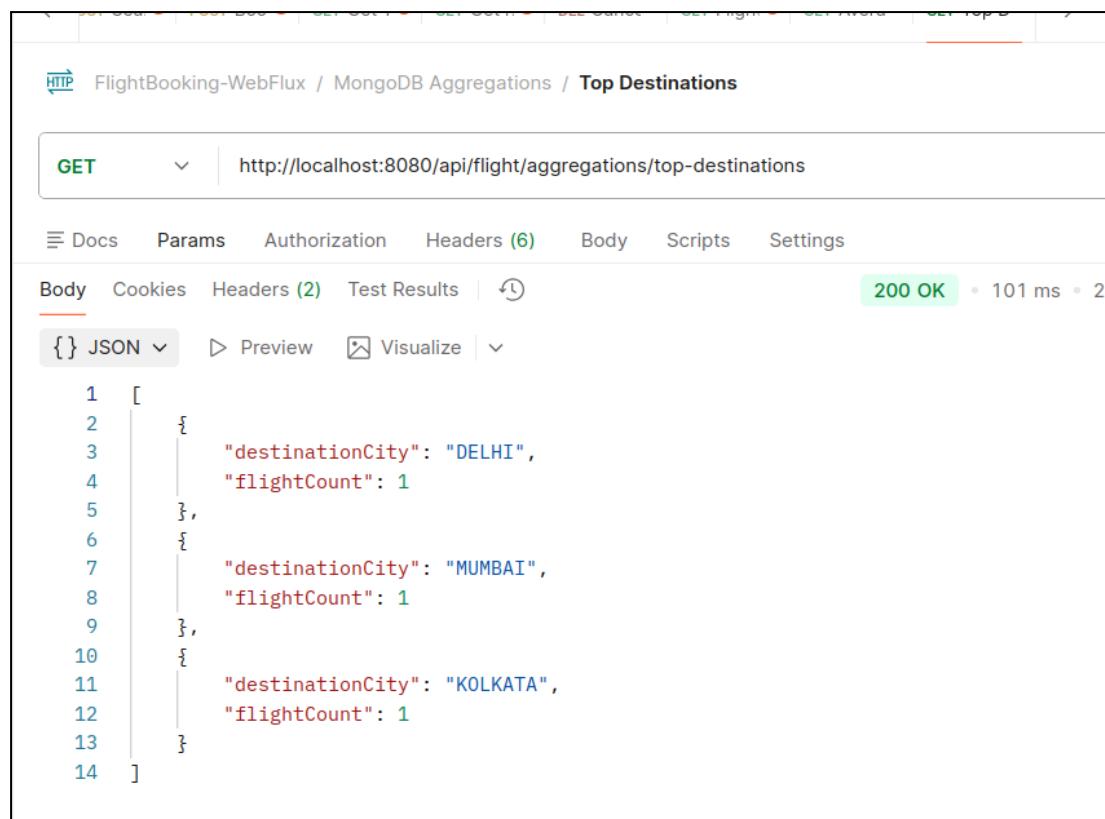
2. GET Average price per route



The screenshot shows a Postman collection named "FlightBooking-WebFlux / MongoDB Aggregations / Average Price Per Route". A GET request is made to `http://localhost:8080/api/flight/aggregations/avg-price-route`. The response status is 200 OK, with a response time of 55 ms and a body size of 30 B. The response is a JSON array containing three elements, each representing a route with its average price.

```
[{"sourceCity": "MUMBAI", "destinationCity": "DELHI", "averagePrice": 6200.0}, {"sourceCity": "BANGALORE", "destinationCity": "KOLKATA", "averagePrice": 9000.0}, {"sourceCity": "CHENNAI", "destinationCity": "MUMBAI", "averagePrice": 5000.0}]
```

3. GET top destinations



HTTP FlightBooking-WebFlux / MongoDB Aggregations / Top Destinations

GET http://localhost:8080/api/flight/aggregations/top-destinations

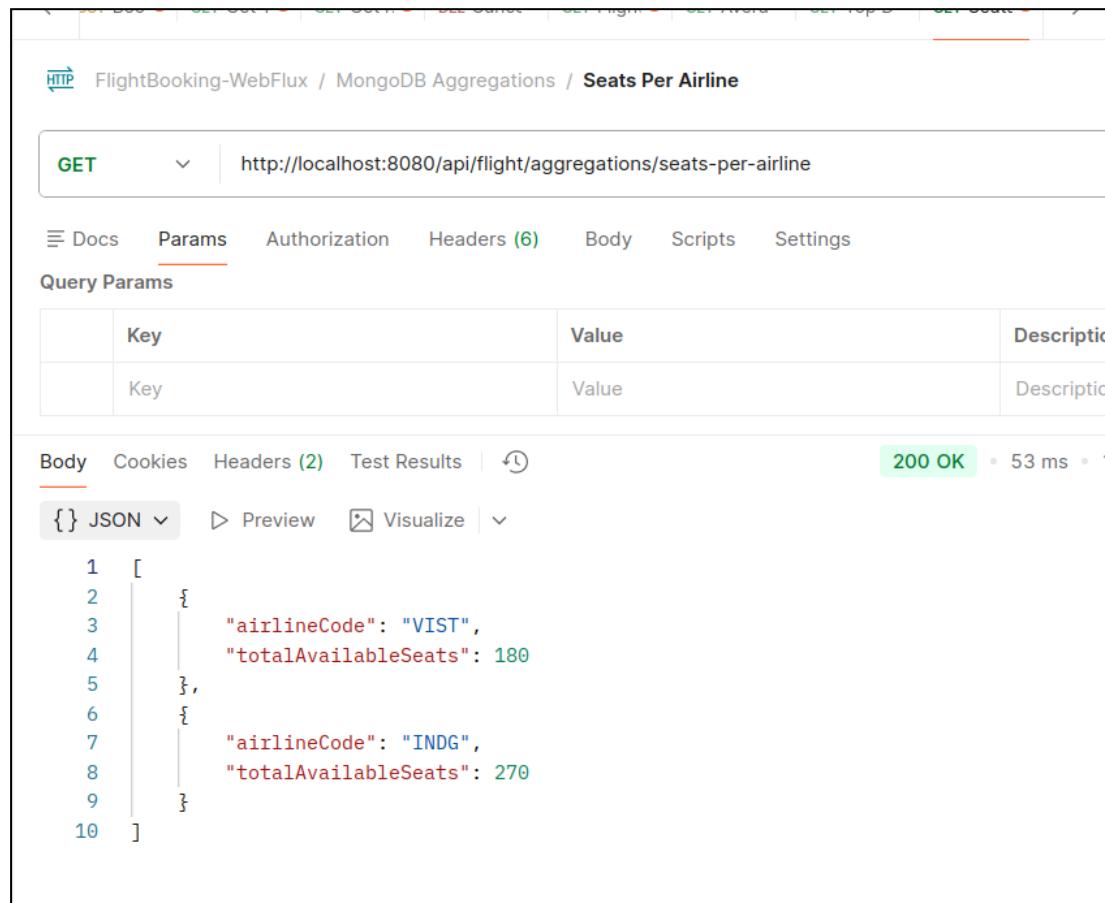
Headers (6) Body Scripts Settings

Body Cookies Headers (2) Test Results 200 OK • 101 ms • 2

{ } JSON ▾ Preview Visualize

```
1 [  
2 {  
3   "destinationCity": "DELHI",  
4   "flightCount": 1  
5 },  
6 {  
7   "destinationCity": "MUMBAI",  
8   "flightCount": 1  
9 },  
10 {  
11   "destinationCity": "KOLKATA",  
12   "flightCount": 1  
13 }  
14 ]
```

4. GET Seats per airline



HTTP FlightBooking-WebFlux / MongoDB Aggregations / Seats Per Airline

GET http://localhost:8080/api/flight/aggregations/seats-per-airline

Docs Params Authorization Headers (6) Body Scripts Settings

Query Params

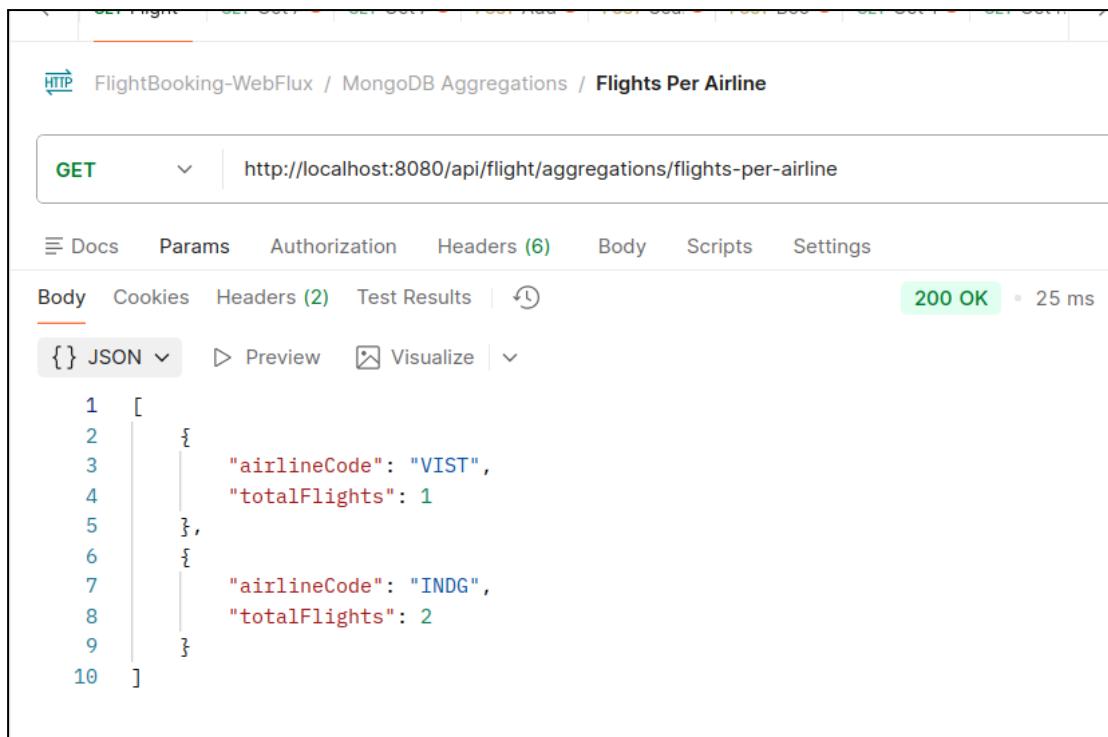
Key	Value	Description
Key	Value	Description

Body Cookies Headers (2) Test Results 200 OK • 53 ms • 1

{ } JSON ▾ Preview Visualize

```
1 [  
2 {  
3   "airlineCode": "VIST",  
4   "totalAvailableSeats": 180  
5 },  
6 {  
7   "airlineCode": "INDG",  
8   "totalAvailableSeats": 270  
9 }  
10 ]
```

5. GET Flights per air line



The screenshot shows the Postman interface with the following details:

- HTTP Method:** GET
- URL:** http://localhost:8080/api/flight/aggregations/flights-per-airline
- Status:** 200 OK (25 ms)
- Body (JSON):**

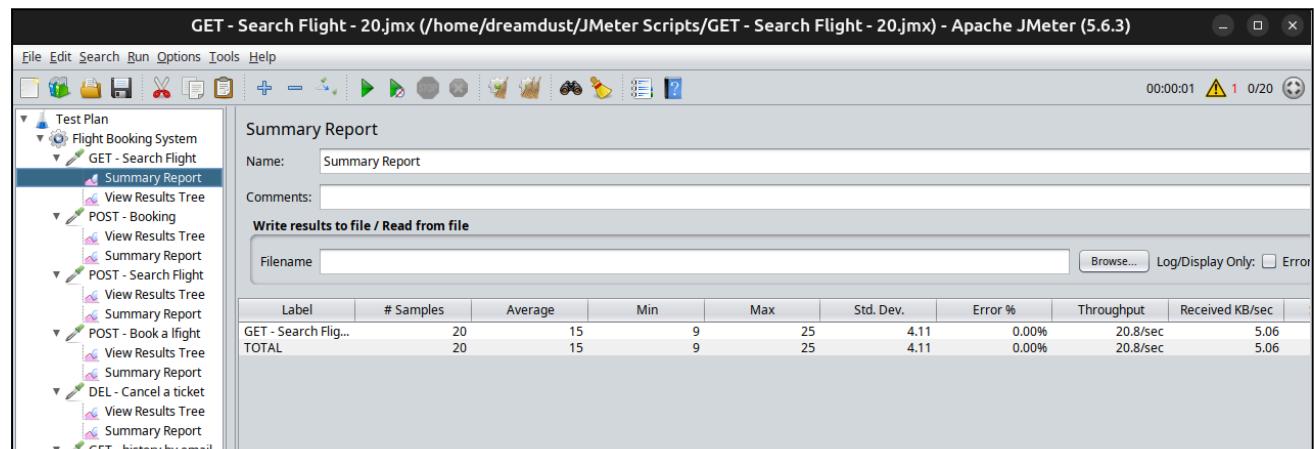
```
[{"airlineCode": "VIST", "totalFlights": 1}, {"airlineCode": "INDG", "totalFlights": 2}]
```

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



The screenshot shows the Apache JMeter interface with the following details:

- Test Plan:** GET - Search Flight - 20.jmx
- Summary Report:**
 - Name: Summary Report
 - Comments:
 - Write results to file / Read from file
 - Filename:
 - Browse... Log/Display Only: Error:
 - Table Data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec
GET - Search Flig...	20	15	9	25	4.11	0.00%	20.8/sec	5.06
TOTAL	20	15	9	25	4.11	0.00%	20.8/sec	5.06

The screenshot shows the JMeter Test Results window. The left pane displays a list of 15 test samples, all marked as successful (green checkmark) and titled "GET - Search Flight". The right pane shows the details for the first sample, which is a "Sampler result".

Sampler result Details:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:26:39 IST
- Load time: 20
- Connect Time: 1
- Latency: 19
- Size in bytes: 249
- Sent bytes: 142
- Headers size in bytes: 79
- Body size in bytes: 170
- Sample Count: 1
- Error Count: 0
- Data type ("text" | "bin" | ""): text
- Response code: 200
- Response message: OK

HTTPSampleResult fields:

- ContentType: application/json
- DataEncoding: null

POST - Book a flight

The screenshot shows the JMeter Test Results window. On the left, a tree view lists multiple 'POST - Booking' requests. The right panel displays the results for the first request, with tabs for 'Sampler result', 'Request', and 'Response data'. The 'Sampler result' tab contains the following details:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:26:39 IST
- Load time: 8
- Connect Time: 0
- Latency: 8
- Size in bytes: 229
- Sent bytes: 517
- Headers size in bytes: 72
- Body size in bytes: 157
- Sample Count: 1
- Error Count: 0
- Data type ("text" | "bin" | ""): text
- Response code: 200
- Response message: OK

Below these details, under 'HTTPSampleResult fields:', the following values are listed:

- ContentType: application/json
- DataEncoding: null

 View Results Tree	 POST - Booking	 View Results Tree	 Summary Report
 View Results Tree	 POST - Search Flight	 View Results Tree	 Summary Report
 View Results Tree	 POST - Book a flight	 View Results Tree	 Summary Report
 View Results Tree	 DEL - Cancel a ticket	 View Results Tree	 Summary Report
 GET - history by email			

DELETE - Cancel a Booking

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
DEL - Cancel ...	20	19	13	32	4.74	0.00%	21.1/sec	2.24	4.83	109.0
TOTAL	20	19	13	32	4.74	0.00%	21.1/sec	2.24	4.83	109.0

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

The screenshot shows the JMeter Test Results window. The left pane displays a list of 20 successful requests, each labeled "GET - Search Flight". The right pane provides detailed results for the first request:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:36:24 IST
- Load time: 18
- Connect Time: 2
- Latency: 17
- Size in bytes: 249
- Sent bytes: 142
- Headers size in bytes: 79
- Body size in bytes: 170
- Sample Count: 1
- Error Count: 0
- Data type ("text" | "bin" | ""): text
- Response code: 200
- Response message: OK

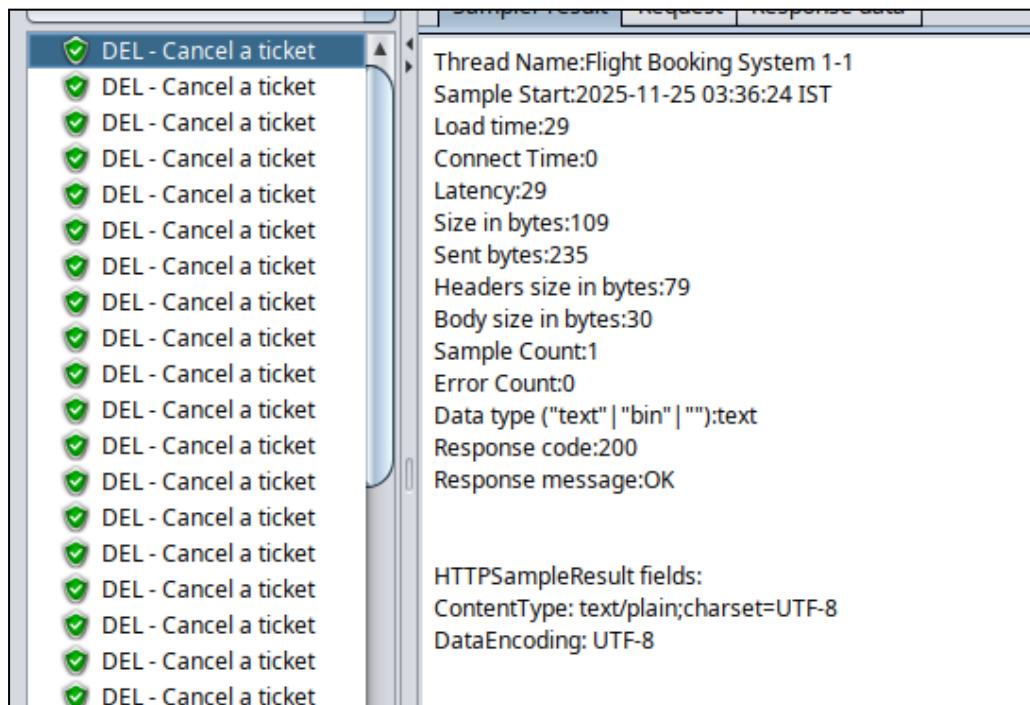
Below these details, under "HTTPSampleResult fields:", the following information is listed:

- ContentType: application/json
- DataEncoding: null

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

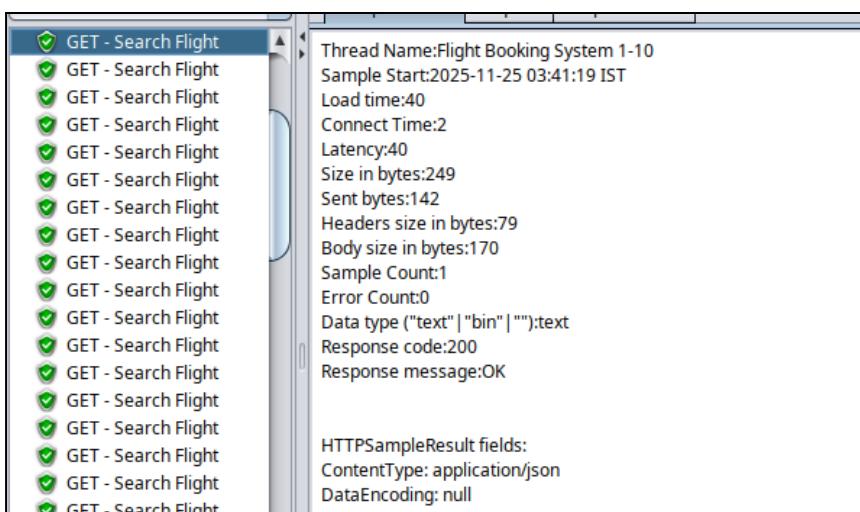


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
DEL - Cancel ...	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0
TOTAL	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0

4.3 With 100 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/sec	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

The screenshot shows the JMeter interface with a 'Text' listener selected in the left sidebar. The main area displays the 'Sampler result' tab, which contains the following details:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:41:18 IST
- Load time: 44
- Connect Time: 0
- Latency: 44
- Size in bytes: 109
- Sent bytes: 235
- Headers size in bytes: 79
- Body size in bytes: 30
- Sample Count: 1
- Error Count: 0
- Data type ("text" | "bin" | ""): text
- Response code: 200
- Response message: OK

Below the Sampler results, under the 'HTTPSampleResult fields:' section, the following values are listed:

- ContentType: text/plain; charset=UTF-8
- DataEncoding: UTF-8

5. All API Endpoints Testing & Results

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

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/flight/addAirline
- Body (JSON):**

```
1 {
2   "airlineCode": "SJ",
3   "airlineName": "Spice Jet "
4 }
```
- Status:** 201 Created
- Time:** 377 ms

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

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/flight/getAllAirlines
- Body (JSON):**

```
1 [
2   {
3     "airlineCode": "INDG",
4     "airlineName": "Indigo"
5   },
6   {
7     "airlineCode": "VIST",
8     "airlineName": "Vistara"
9   },
10  {
11    "airlineCode": "SJ",
12    "airlineName": "Spice Jet "
13  }
14 ]
```
- Status:** 200 OK
- Time:** 82 ms

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

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/flight/airline/inventory/add
- Body:** Raw JSON (selected)
- Request Body:**

```
1 {
2     "airlineCode": "VIST",
3     "flightNumber": "VI89",
4     "sourceCity": "LUCKNOW",
5     "destinationCity": "MUMBAI",
6     "departureDate": "2025-12-10",
7     "departureTime": "09:30",
8     "arrivalDate": "2025-12-10",
9     "arrivalTime": "11:45",
10    "totalSeats": 180,
11    "price": 6000,
12    "mealAvailable": true
13 }
```
- Response Status:** 201 Created
- Response Time:** 644 ms
- Response Body:**

```
1 {
2     "flightId": "6924cd4e3593ecc160c44342",
```

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

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/flight/search
- Body:** Raw JSON (selected)
- Request Body:**

```
1 {
2     "sourceCity": "LUCKNOW",
3     "destinationCity": "MUMBAI",
4     "travelDate": "2025-12-10",
5     "tripType": "ONE WAY"
6 }
```
- Response Status:** 201 Created
- Response Time:** 644 ms
- Response Body:**

```
1 [
2     {
3         "flightId": "6924cd4e3593ecc160c44342",
4         "flightNumber": "VI89",
5         "airlineCode": "VIST",
6         "sourceCity": "LUCKNOW",
7         "destinationCity": "MUMBAI",
8         "departureDate": "2025-12-10",
9         "arrivalDate": "2025-12-10",
10        "departureTime": "09:30:00",
11        "arrivalTime": "11:45:00",
12        "mealAvailable": true,
13        "totalSeats": 180,
14        "availableSeats": 180,
15        "price": 6000.0
16    }
17 ]
```

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

The screenshot shows the Postman interface for a POST request to <http://localhost:8080/api/flight/booking/6924cd4e3593ecc160c44342>. The request body contains the following JSON:

```

1 {
2   "tripType": "ONE WAY",
3   "contactName": "John Doe",
4   "contactEmail": "john@example.com",
5   "passengers": [
6     {
7       "name": "John Doe",
8       "age": 30,
9       "gender": "MALE",
10      "seatOutbound": "12B",
11      "meal": "NON_VEG"
12    }
13  ]
14 }
15

```

The response body is:

```

1 {
2   "bookingId": "6924ce423593ecc160c443f7",
3   "tripType": "ONE WAY",
4   "outboundFlightId": "6924cd4e3593ecc160c44342",
5   "returnFlight": null,
6   "pnzOutbound": "BD6FE9",
7   "pnzReturn": null,
8   "contactName": "John Doe",
9   "contactEmail": "john@example.com",
10  "totalPassengers": 1,
11  "status": "CONFIRMED"
12 }
13

```

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

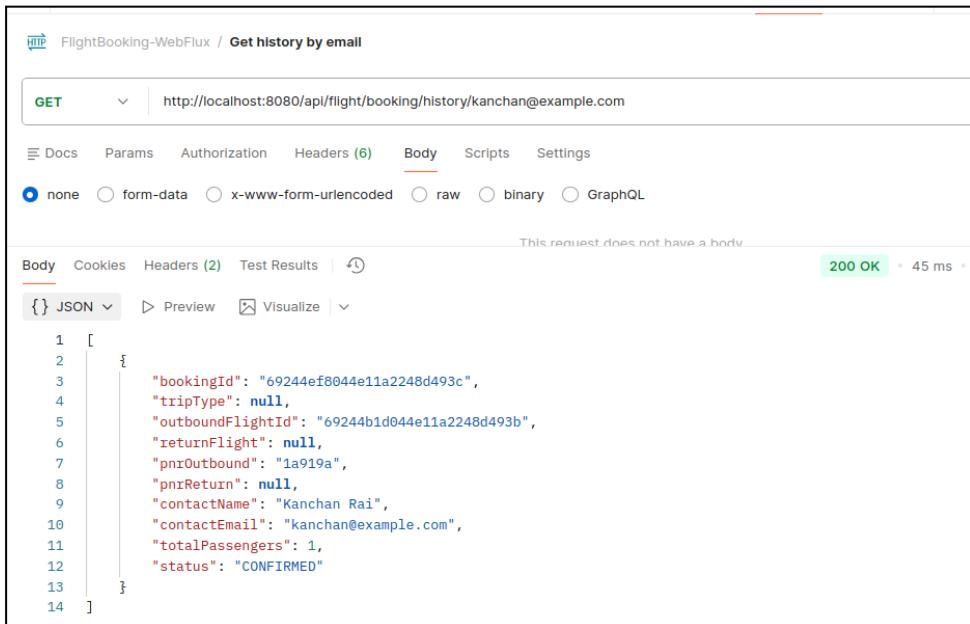
The screenshot shows the Postman interface for a GET request to <http://localhost:8080/api/flight/ticket/1a919a>. The response status is 200 OK. The response body is:

```

1 {
2   "bookingId": "69244ef8044e11a2248d493c",
3   "tripType": null,
4   "outboundFlightId": "69244b1d044e11a2248d493b",
5   "returnFlight": null,
6   "pnzOutbound": "1a919a",
7   "pnzReturn": null,
8   "contactName": "Kanchan Rai",
9   "contactEmail": "kanchan@example.com",
10  "totalPassengers": 1,
11  "status": "CONFIRMED"
12 }
13

```

7. GET - History by email



HTTP FlightBooking-WebFlux / Get history by email

GET http://localhost:8080/api/flight/booking/history/kanchan@example.com

Body

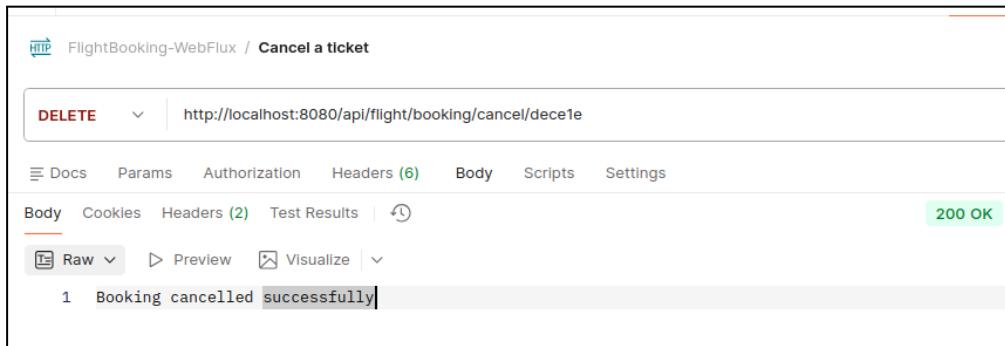
None form-data x-www-form-urlencoded raw binary GraphQL

200 OK 45 ms

{ } JSON ▶ Preview Visualize

```
1 [  
2 {  
3   "bookingId": "69244ef8044e11a2248d493c",  
4   "tripType": null,  
5   "outboundFlightId": "69244b1d044e11a2248d493b",  
6   "returnFlight": null,  
7   "pnrOutbound": "1a919a",  
8   "pnrReturn": null,  
9   "contactName": "Kanchan Rai",  
10  "contactEmail": "kanchan@example.com",  
11  "totalPassengers": 1,  
12  "status": "CONFIRMED"  
13 }]  
14 ]
```

8. DELETE - Cancel a ticket



HTTP FlightBooking-WebFlux / Cancel a ticket

DELETE http://localhost:8080/api/flight/booking/cancel/dece1e

Body

200 OK

Raw ▶ Preview Visualize

```
1 Booking cancelled successfully|
```