

Week 5 - Project Report

By : Kanchan Rai

Topic : FlightBookingSystem w/ WebFlux

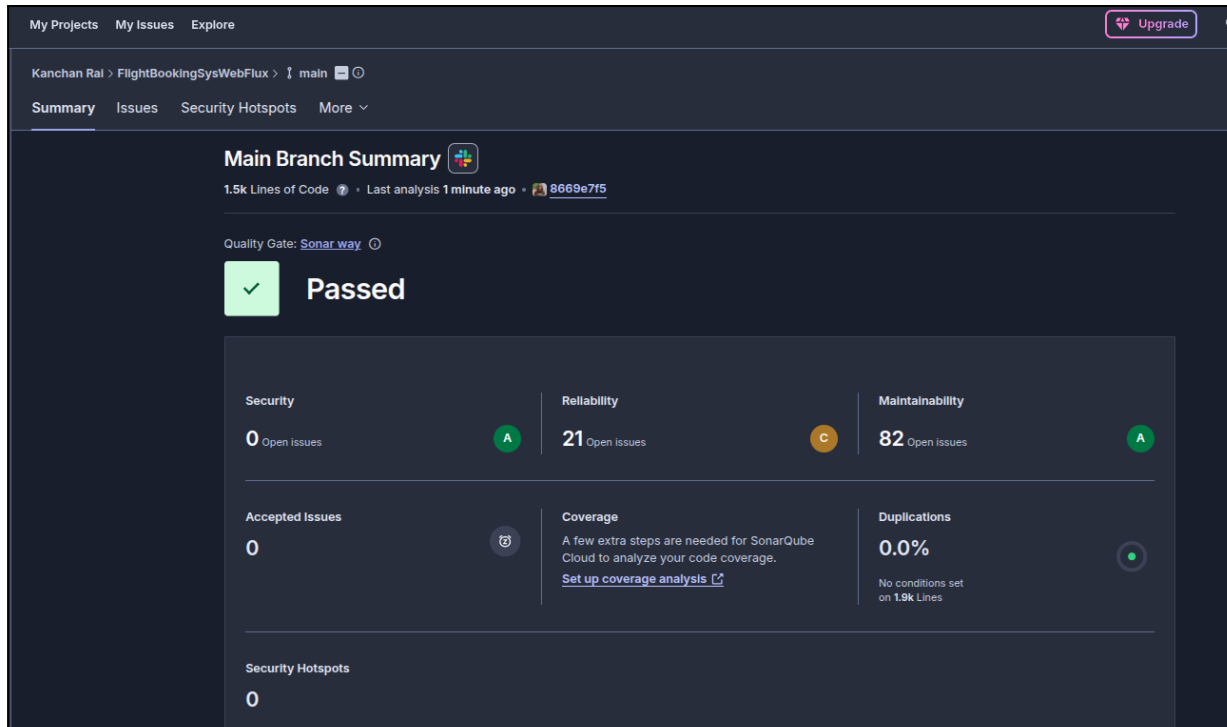
The following document contains the description and explanation of the project FlightBookingSystem made with WebFlux. This project has all the required validations for all end points, exception handling, unit test coverage of 90%. Load testing carried out as well for all endpoints. The database used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

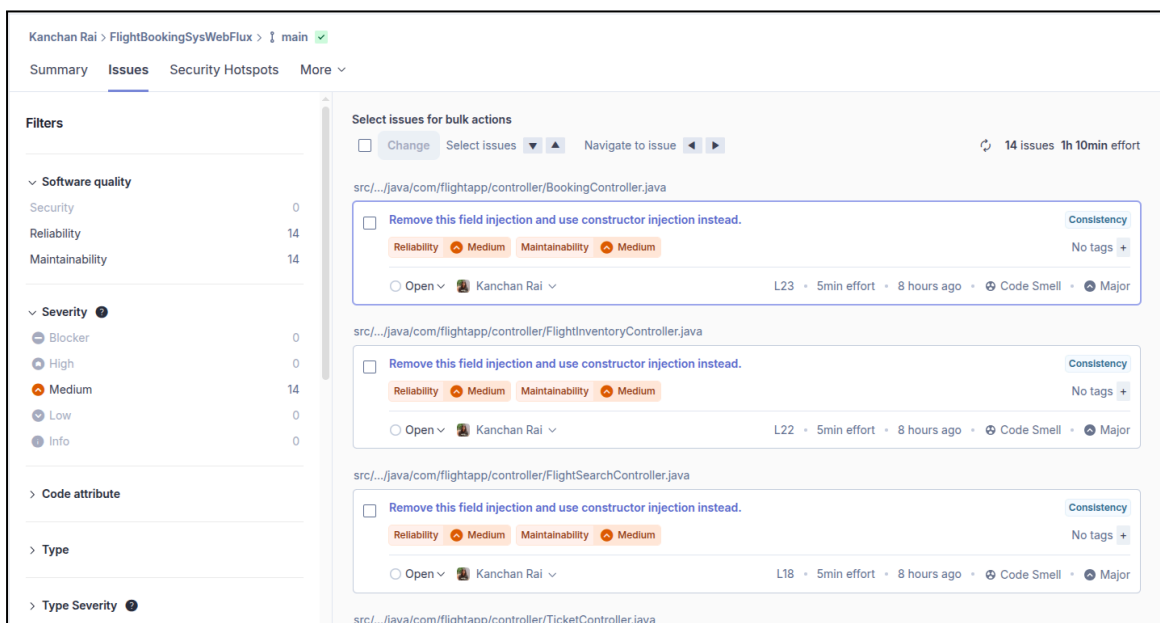
1.1 Before Fixing



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 -



1.2 After Fixing

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

 **FlightBookingSysWebFlux** New Public

Last analysis: 11/25/2025, 1:45 AM • 1.4k Lines of Code • Java, XML, ...


 **0**
Security

 **14**
Reliability

 **14**
Maintainability

 **100%**
Hotspots Reviewed


 **0.0%**
Duplications


 **FlightBookingSysWebFlux**


No tags
Last analysis Nov 25, 2025
1.4k Lines of Code

Java XML Unknown

Main Branch Status


Quality Gate 

 **Passed**

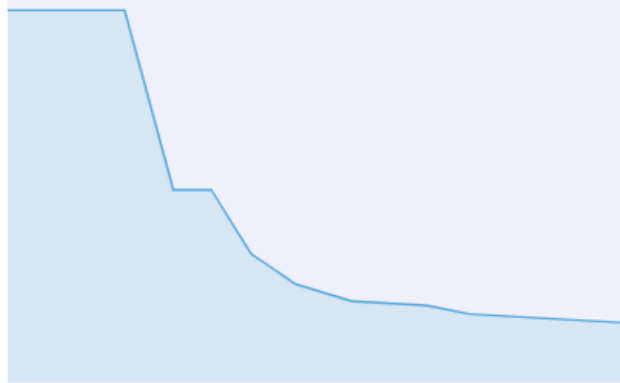

Enjoy your sparkling code!

[See Full Analysis](#)

Main Branch Evolution since 2 hours ago

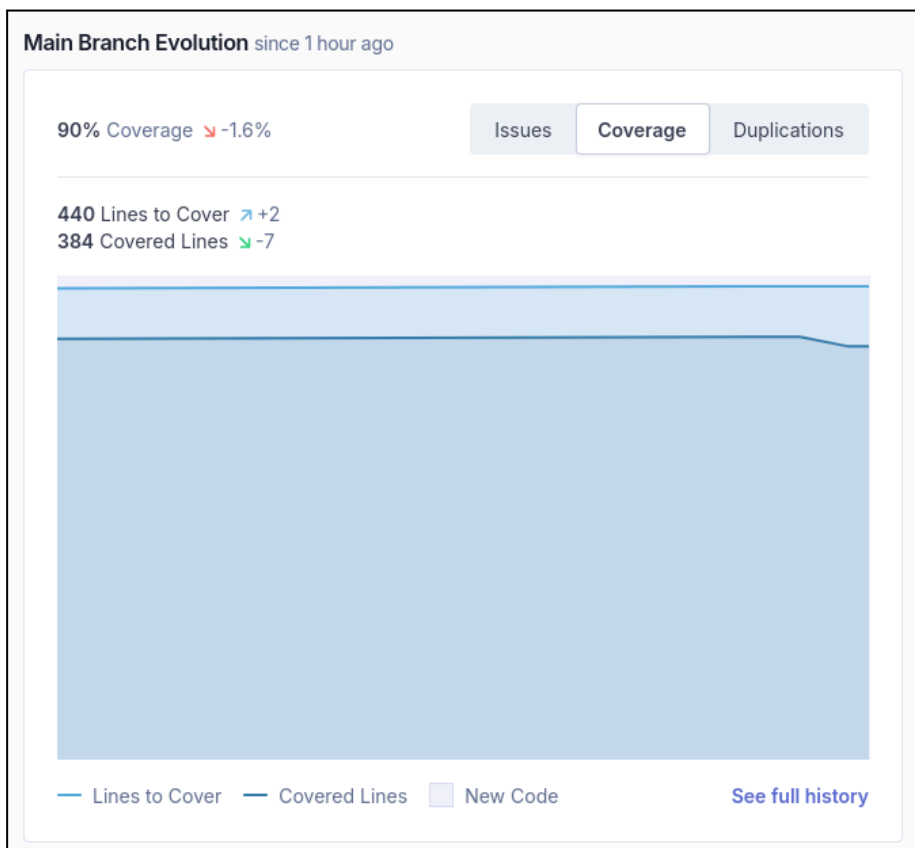
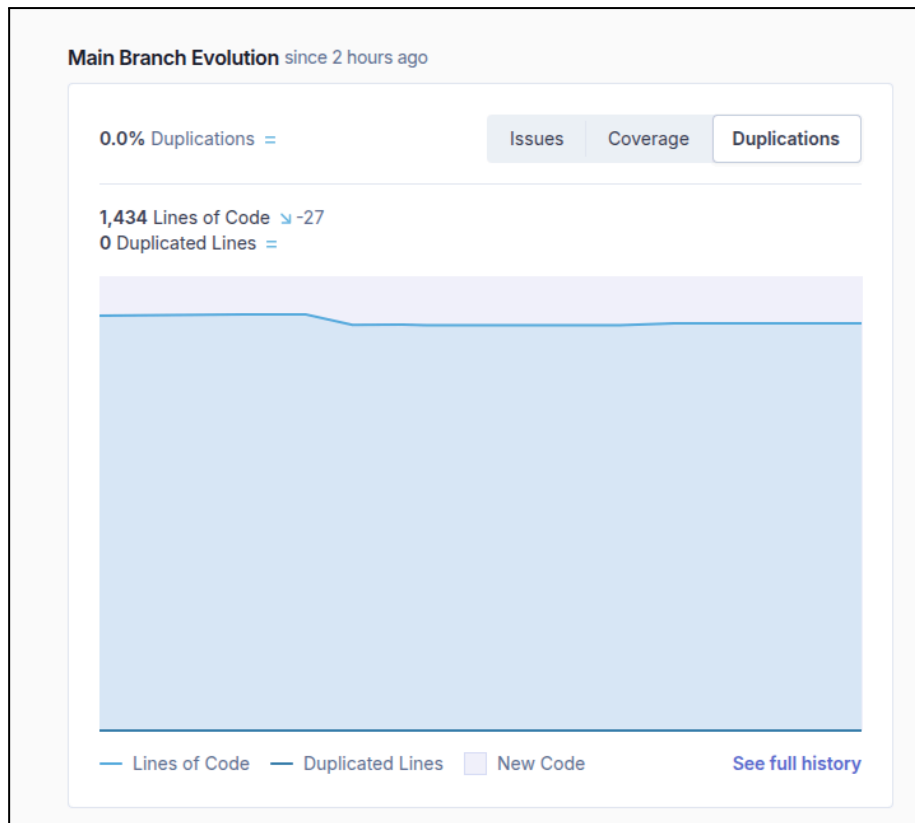
14 Issues  -73

Issues Coverage Duplications


Issues New Code

[See full history](#)

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();

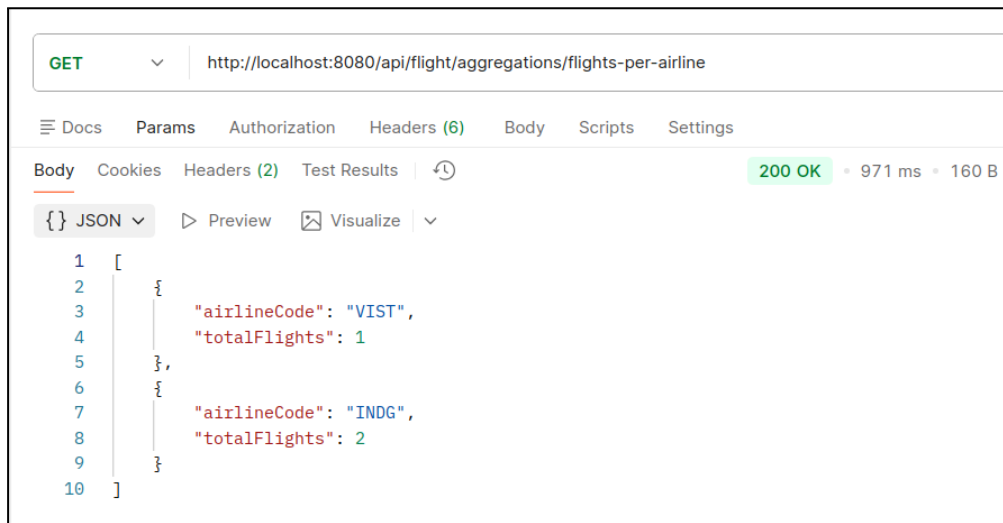
@Aggregation(pipeline = {
    "{ $group: { _id: '$airlineCode', totalAvailableSeats: { $sum: '$availableSeats' } } }",
    "{ $project: { airlineCode: '$_id', totalAvailableSeats: 1, _id: 0 } }"
})
Flux<AirlineSeats> getSeatStatsPerAirline();

@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

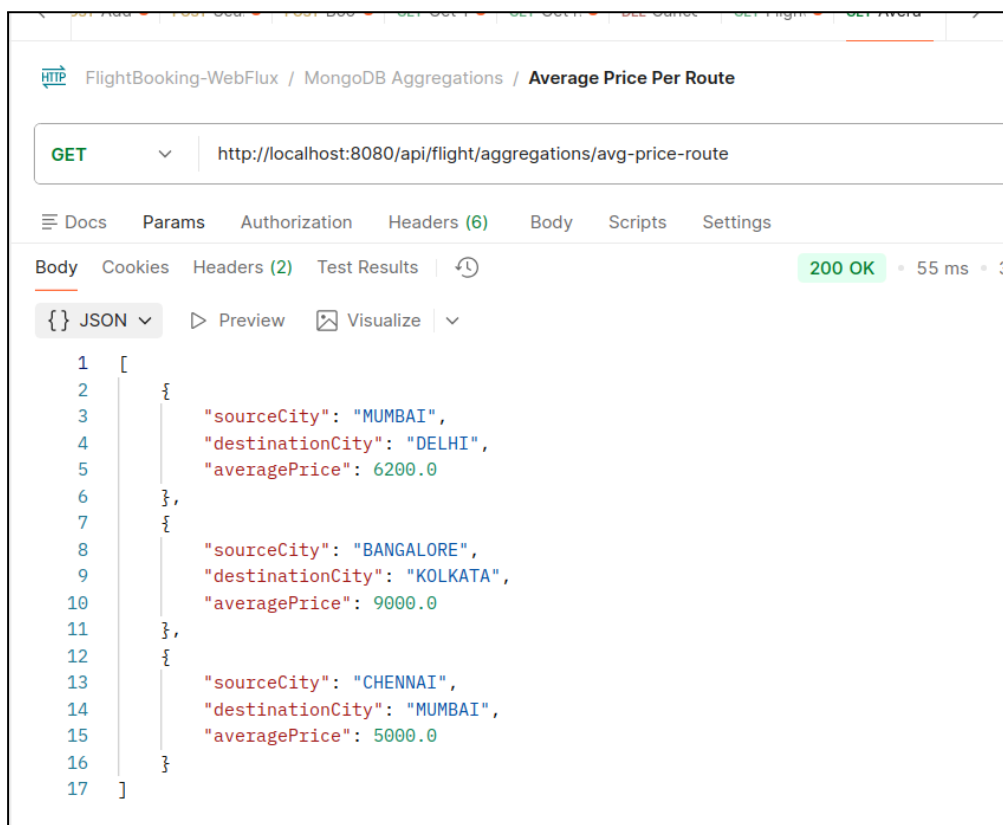


GET http://localhost:8080/api/flight/aggregations/flights-per-airline

200 OK • 971 ms • 160 B

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

2. GET Average price per route



FlightBooking-WebFlux / MongoDB Aggregations / Average Price Per Route

GET http://localhost:8080/api/flight/aggregations/avg-price-route

200 OK • 55 ms • 300 B

```
[
  {
    "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

FlightBooking-WebFlux / MongoDB Aggregations / **Top Destinations**

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

200 OK • 101 ms • 2

Body Cookies Headers (2) Test Results

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

FlightBooking-WebFlux / MongoDB Aggregations / **Seats Per Airline**

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

200 OK • 53 ms • 1

Body Cookies Headers (2) Test Results

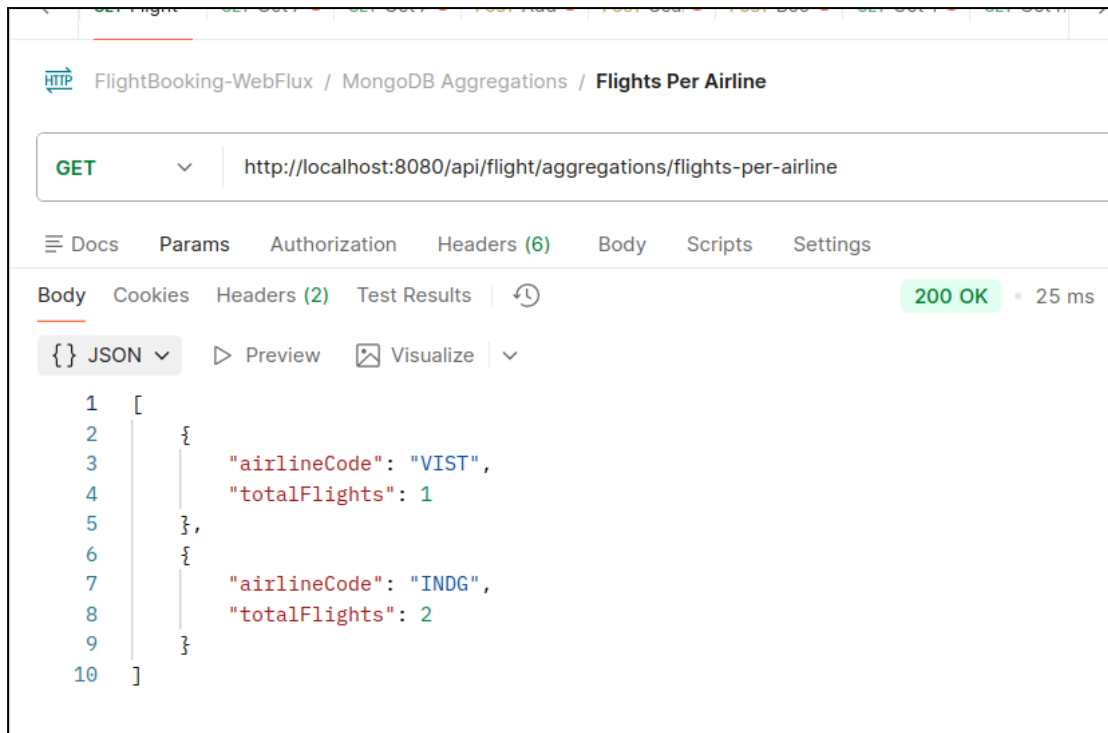
Query Params

Key	Value	Description
Key	Value	Description

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

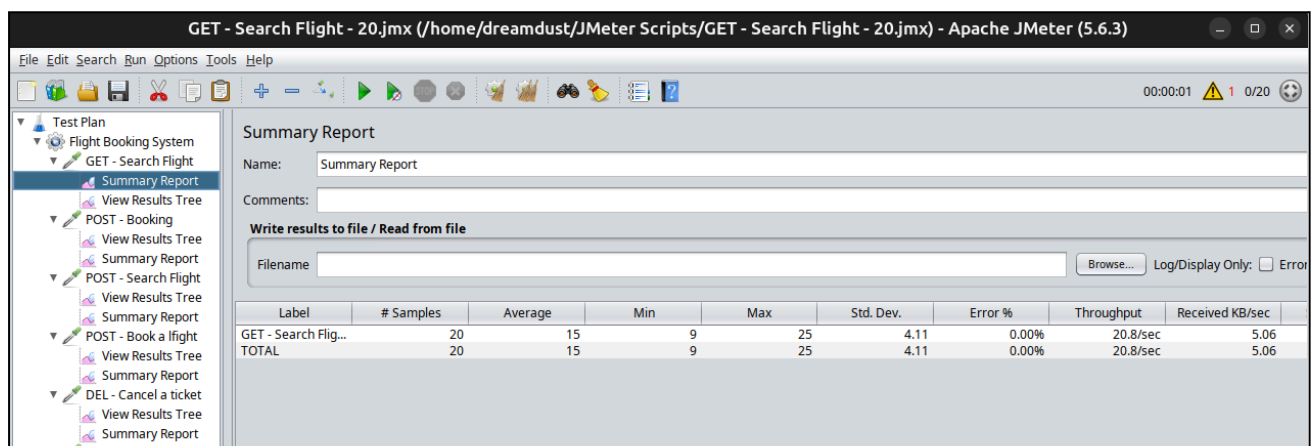


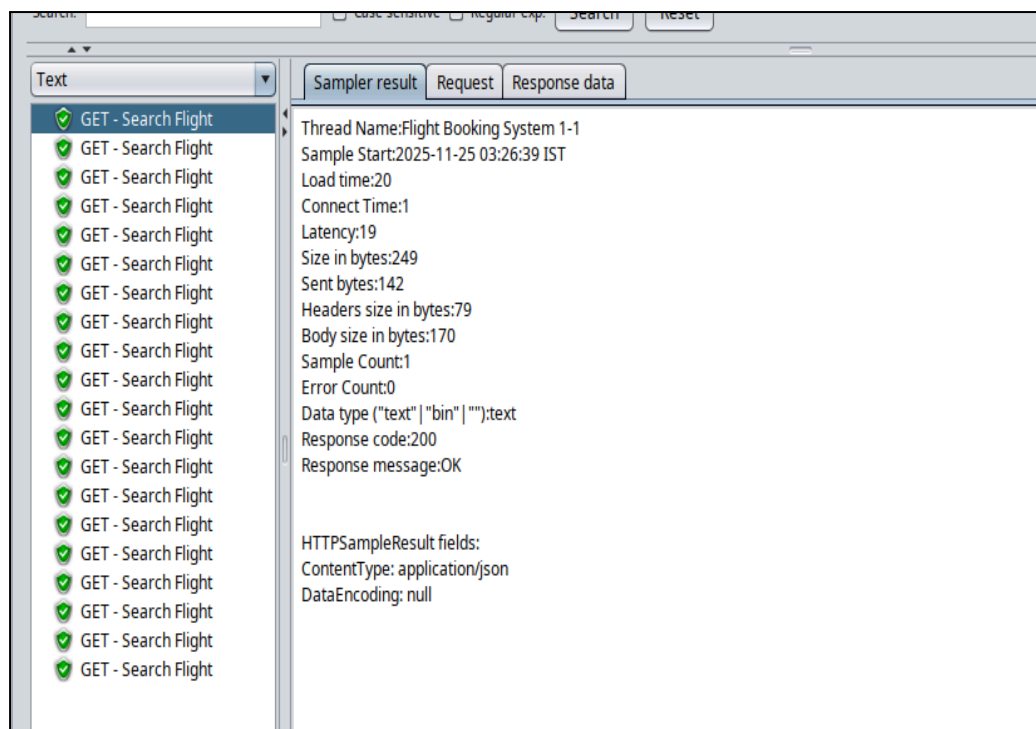
4. JMeter Load Testing

4.1 With 20 Threads

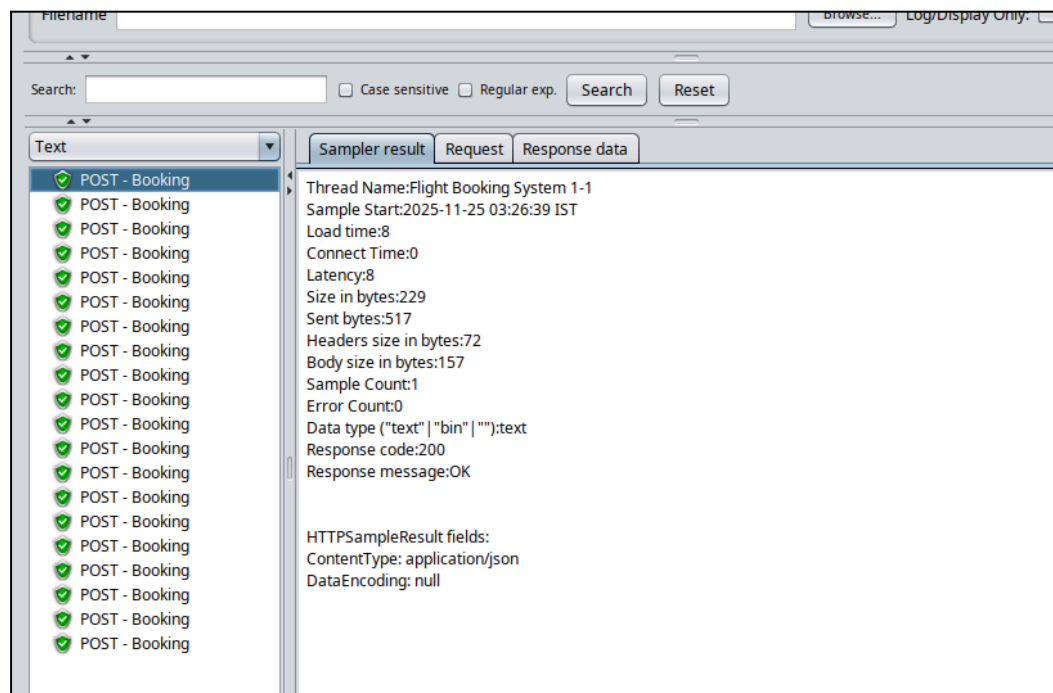
The load testing has been carried out for all endpoints. Each one of GET,POST,DELETE has been shown below.

GET - Search a flight





POST - Book a flight



View Results Tree

POST - Booking

View Results Tree

Summary Report

POST - Search Flight

View Results Tree

Summary Report

POST - Book a flight

View Results Tree

Summary Report

DEL - Cancel a ticket

View Results Tree

Summary Report

GET - history by email

Comments:

Write results to file / Read from file

Filename

Browse...

Log/Display Only:

Errors

Successes

Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
POST - Book ...	20	6	5	10	1.42	0.00%	21.2/sec	4.52	10.14	218.0
TOTAL	20	6	5	10	1.42	0.00%	21.2/sec	4.52	10.14	218.0

DELETE - Cancel a Booking

Search:

☐ Case sensitive
☐ Regular exp.

Search

Reset

Text

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

✓ DEL - Cancel a ticket

Sampler result

Request

Response data

Thread Name:Flight Booking System 1-1

Sample Start:2025-11-25 03:26:39 IST

Load time:27

Connect Time:0

Latency:27

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

HTTPSampleResult fields:

ContentType: text/plain;charset=UTF-8

DataEncoding: UTF-8

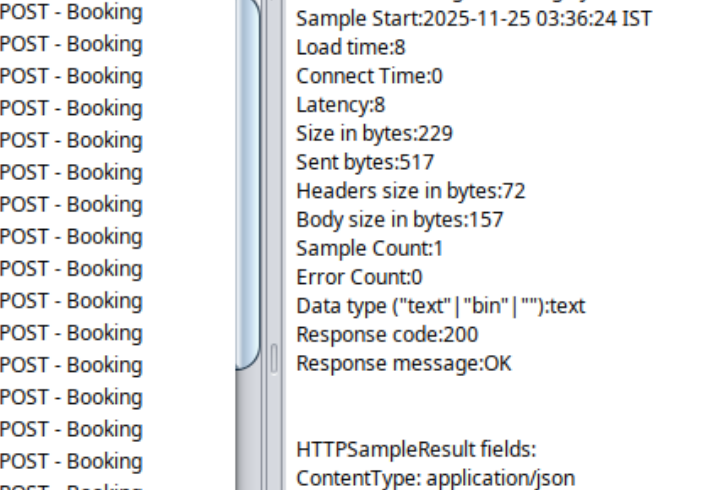
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
DEL - Cancel ...	20	19	13	32	4.74	0.00%	21.1/sec	2.24	4.83	109.0
TOTAL	20	19	13	32	4.74	0.00%	21.1/sec	2.24	4.83	109.0

11

GET - Search a flight

[illegible]

POST - Book a flight



The screenshot displays the JMeter GUI. On the left, a tree view shows 15 'POST - Booking' requests. The right pane shows the details for the selected request:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:36:24 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, the 'HTTPSampleResult fields:' section shows:

- ContentType: application/json
- DataEncoding: null

DELETE - Cancel a booking

Thread Name: Flight Booking System 1-1
Sample Start: 2025-11-25 03:36:24 IST
Load time: 29
Connect Time: 0
Latency: 29
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

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

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 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

Thread Name: Flight Booking System 1-10
Sample Start: 2025-11-25 03:41:19 IST
Load time: 40
Connect Time: 2
Latency: 40
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 'Text' window in JMeter. On the left, a list of 18 items, all labeled 'POST - Booking', are shown with green checkmarks. The right pane displays the 'Sampler result' for the selected item. The details are as follows:

- Thread Name: Flight Booking System 1-1
- Sample Start: 2025-11-25 03:41:18 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, the 'HTTPSampleResult fields' are listed:

- ContentType: application/json
- DataEncoding: null

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

The screenshot shows the 'Text' window in JMeter. On the left, a list of 18 items, all labeled 'DEL - Cancel a ticket', are shown with green checkmarks. The right pane displays the 'Sampler result' for the selected item. The details are as follows:

- 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 these details, the 'HTTPSampleResult fields' 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 a REST client interface for the endpoint `http://localhost:8080/api/flight/addAirline`. The request method is **POST**. The request body is raw JSON:

```
{  "airlineCode": "SJ",  "airlineName": "Spice Jet "}
```

. The response status is **201 Created** with a response time of 377 ms. The response body is also shown as raw JSON:

```
{  "airlineCode": "SJ",  "airlineName": "Spice Jet "}
```

.

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

The screenshot shows a REST client interface for the endpoint `http://localhost:8080/api/flight/getAllAirlines`. The request method is **GET**. The response status is **200 OK** with a response time of 82 ms. The response body is shown as raw JSON:

```
[  {    "airlineCode": "INDG",    "airlineName": "Indigo"  },  {    "airlineCode": "VIST",    "airlineName": "Vistara"  },  {    "airlineCode": "SJ",    "airlineName": "Spice Jet "  }]
```

.

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

The screenshot shows the Postman interface for the 'Add Flight' endpoint. The URL is `http://localhost:8080/api/flight/airline/inventory/add` and the method is **POST**. The 'Body' tab is selected, showing a raw JSON request:

```
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 }
```

The response status is **201 Created** with a response time of 644 ms and a size of 391 bytes. The response body is shown in the 'Body' tab as a JSON object:

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

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

The screenshot shows the Postman interface for the 'Search Flight' endpoint. The URL is `http://localhost:8080/api/flight/search` and the method is **POST**. The 'Body' tab is selected, showing a raw JSON request:

```
1 {
2   "sourceCity": "LUCKNOW",
3   "destinationCity": "MUMBAI",
4   "travelDate": "2025-12-10",
5   "tripType": "ONE_WAY"
6 }
```

The response status is **200 OK** with a response time of 20 ms and a size of 1024 bytes. The response body is shown in the 'Body' tab as a JSON array containing one flight object:

```
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 a REST client interface for a service named "FlightBooking-WebFlux". The active tab is "Book a flight - One Way". The request method is "POST" and the URL is "http://localhost:8080/api/flight/booking/6924cd4e3593ecc160c44342". The request body is in JSON format, containing details for a one-way flight for John Doe. The response is also in JSON format, indicating a successful booking with a confirmed status.

```
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
```

Body Cookies Headers (2) Test Results

{ } JSON Preview Visualize

```
1 {
2   "bookingId": "6924ce423593ecc160c443f7",
3   "tripType": "ONE_WAY",
4   "outboundFlightId": "6924cd4e3593ecc160c44342",
5   "returnFlight": null,
6   "pnrOutbound": "8D6FE9",
7   "pnrReturn": null,
8   "contactName": "John Doe",
9   "contactEmail": "john@example.com",
10  "totalPassengers": 1,
11  "status": "CONFIRMED"
12 }
```

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

The screenshot shows the same REST client interface, but for a "Get Ticket By PNR" endpoint. The request method is "GET" and the URL is "http://localhost:8080/api/flight/ticket/1a919a". The response is a 200 OK status with a JSON body containing the details of the flight ticket for PNR 1a919a.

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

7. GET - History by email

The screenshot shows a REST client interface for a project named "FlightBooking-WebFlux". The active tab is "Get history by email". The request method is "GET" and the URL is "http://localhost:8080/api/flight/booking/history/kanchan@example.com". The "Body" tab is selected, showing a JSON response. The response status is "200 OK" and the response time is "45 ms".

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

Response Body (JSON):

```
[
  {
    "bookingId": "69244ef8044e11a2248d493c",
    "tripType": null,
    "outboundFlightId": "69244bd044e11a2248d493b",
    "returnFlight": null,
    "pnrOutbound": "1a919a",
    "pnrReturn": null,
    "contactName": "Kanchan Rai",
    "contactEmail": "kanchan@example.com",
    "totalPassengers": 1,
    "status": "CONFIRMED"
  }
]
```

8. DELETE - Cancel a ticket

The screenshot shows a REST client interface for a project named "FlightBooking-WebFlux". The active tab is "Cancel a ticket". The request method is "DELETE" and the URL is "http://localhost:8080/api/flight/booking/cancel/delete". The "Body" tab is selected, showing a raw text response. The response status is "200 OK".

Request: DELETE http://localhost:8080/api/flight/booking/cancel/delete

Response Body (Raw):

```
1 Booking cancelled successfully
```