



## **Week 5 - Project Report**

**By : Kanchan Rai**

**Topic : FlightBookingSystem w/ WebFlux**

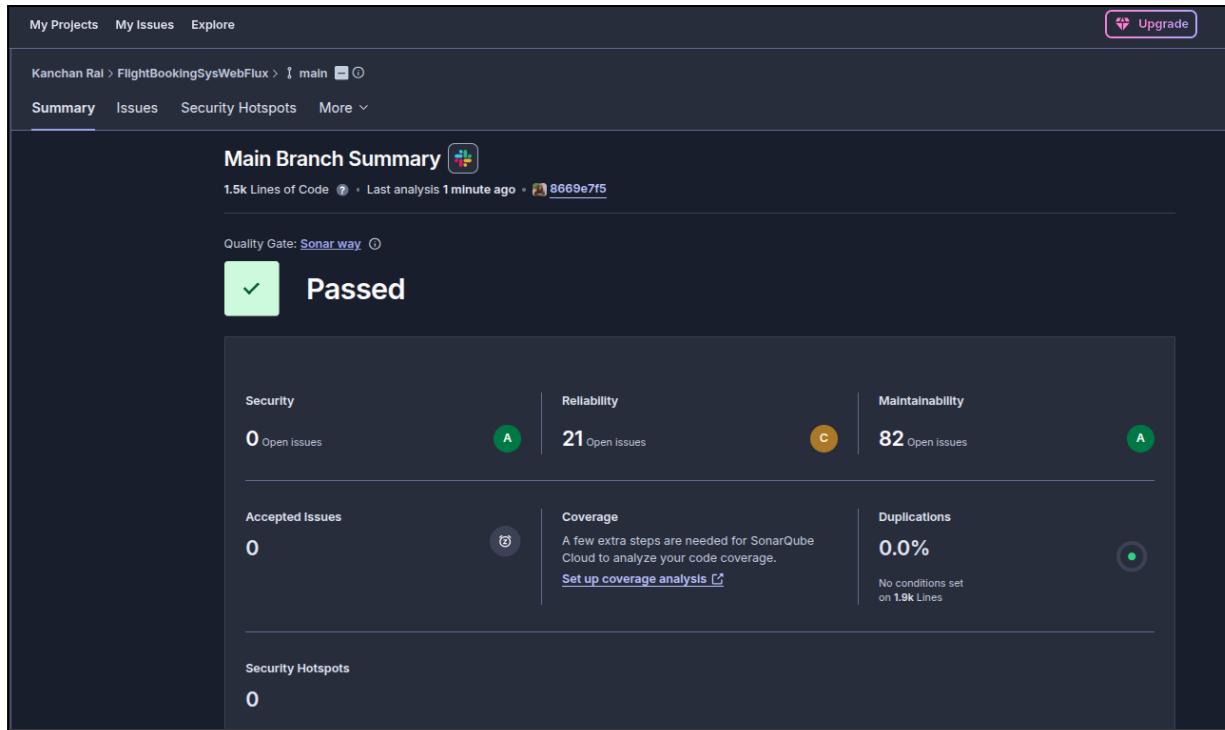
The following document contains the description and explanation of the project FlightBookingSystem made with WebFlux. This project has all the required validations for all end points, exception handling, unit test coverage of 90%. Load testing carried out as well for all endpoints. The database used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

# INDEX

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

# 1. Sonar Cube Report

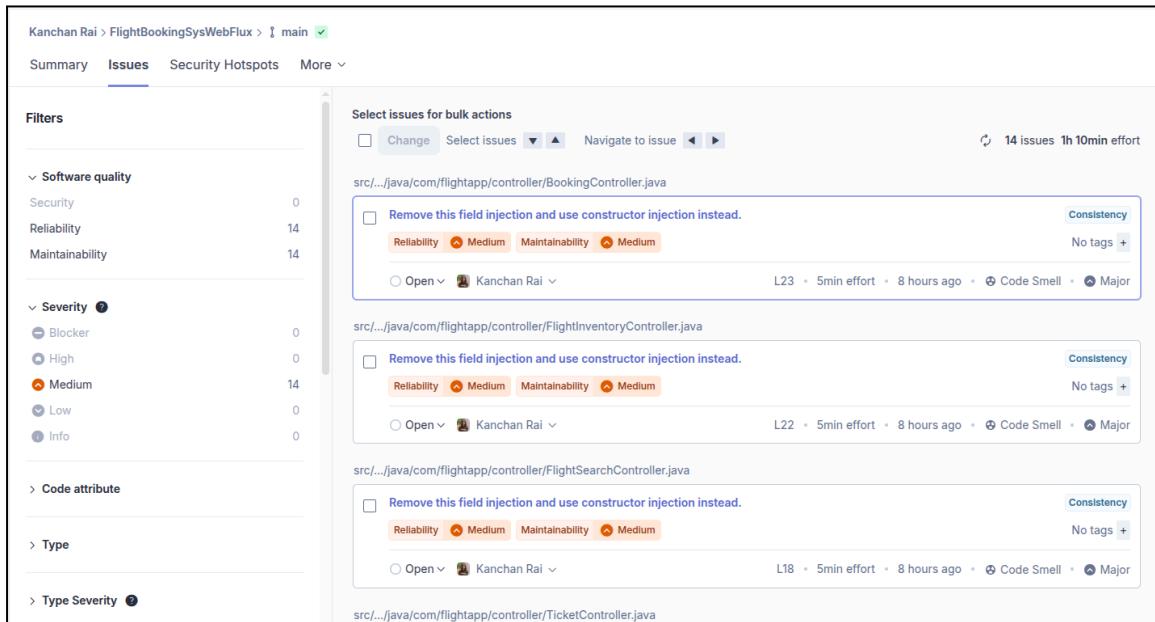
## 1.1 Before Fixing



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

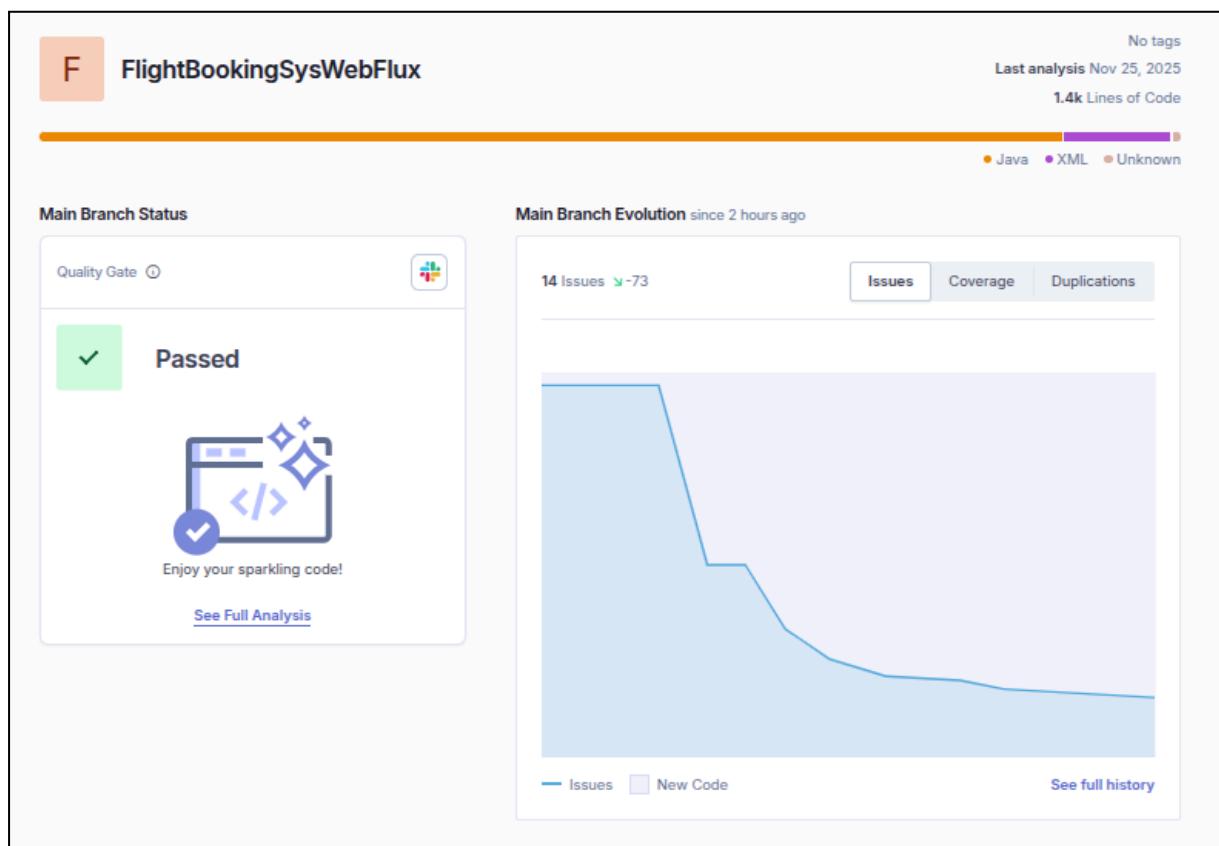
I fixed the issues in reliability and maintainability as suggested by Sonar Qube and reduced them from 87 to 14 issues. These 16 issues are related to changing the @Autowired annotation to constructor injection which I have not changed as of now.

These issues present as follows -

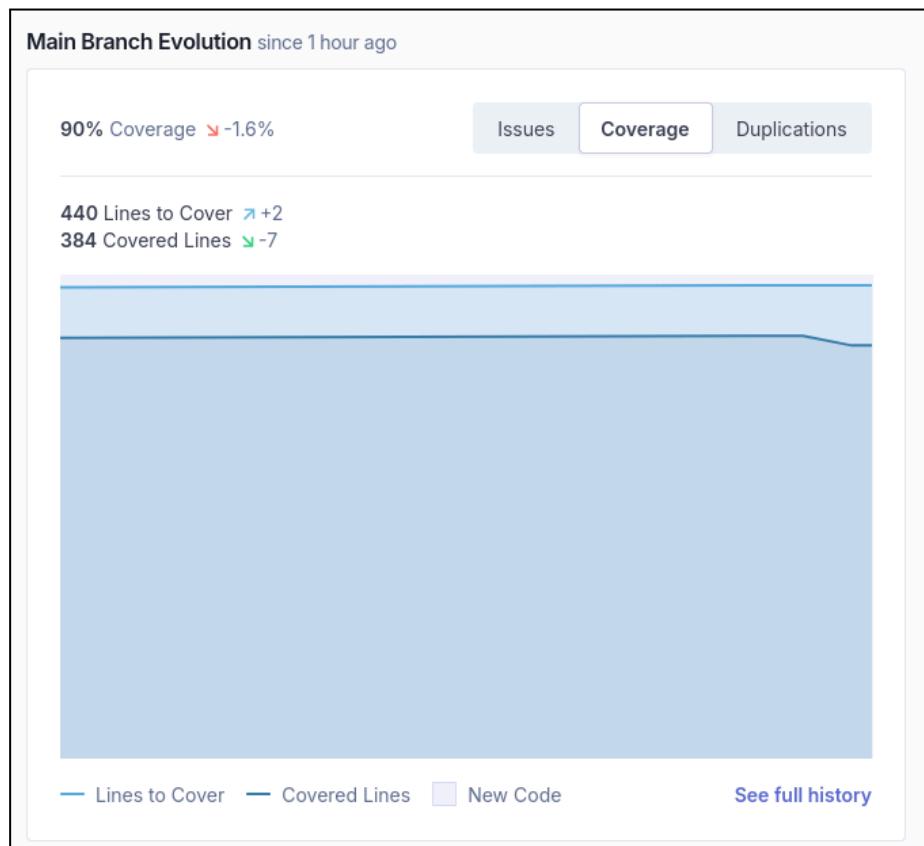
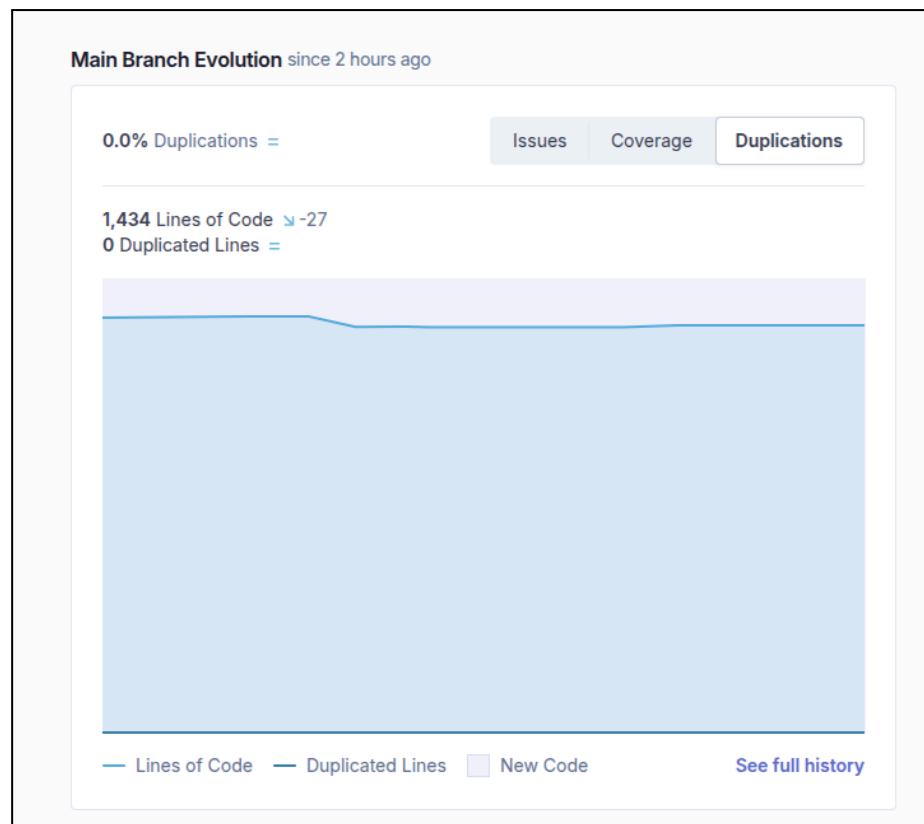


## 1.2 After Fixing

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



0.0% Duplications & 90 % Coverage



## 2. JCoco Coverage Report -

90 % Test Coverage

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 the Postman interface with a successful API call. The URL is `http://localhost:8080/api/flight/aggregations/flights-per-airline`. The response status is **200 OK** with a response time of 971 ms and a size of 160 B. The response body 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 the Postman interface with a successful API call. The URL is `http://localhost:8080/api/flight/aggregations/avg-price-route`. The response status is **200 OK** with a response time of 55 ms and a size of 30 B. The response body is a JSON array containing three elements, each representing a route with source and destination cities and their 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

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/flight/aggregations/top-destinations
- Headers:** (6)
- Body:** JSON (shown below)
- Test Results:** 200 OK (101 ms)

```
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

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/flight/aggregations/seats-per-airline
- Headers:** (6)
- Query Params:** (empty table)
- Body:** JSON (shown below)
- Test Results:** 200 OK (53 ms)

```
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:

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

```

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

```

## 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

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/...	Avg. Byt...
GET - Search Flight	20	11	9	13	1.22	0.00%	20.8/sec	5.06	2.88	249.0
POST - Booking	20	5	4	6	0.59	0.00%	21.0/sec	4.69	10.58	229.0
POST - Search Flight	20	5	4	6	0.44	0.00%	21.0/sec	4.57	6.51	223.0
POST - Book a flight	20	5	4	7	1.01	0.00%	21.0/sec	4.47	10.02	218.0
DEL - Cancel a ticket	20	15	12	19	1.66	0.00%	20.7/sec	2.21	4.76	109.0
GET - history by email	20	10	9	13	1.15	0.00%	20.8/sec	7.45	3.32	366.0
GET - Ticket by pnr	20	10	8	13	1.35	0.00%	20.9/sec	6.94	2.88	340.0
TOTAL	140	9	4	19	3.94	0.00%	137.8/sec	33.33	38.54	247.7

The screenshot shows the JMeter interface with the following details:

- Search:** Text input field containing "Text".
- Search Options:** Case Sensitive, Regular Exp., Search, Reset buttons.
- Samplers List:** A tree view on the left showing 15 instances of the sampler "GET - Search Flight".
- Result Tab:** Sampler result tab is selected, showing the following results for the first sampler:
  - 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
- Result Fields:** A section at the bottom lists the fields of the `HTTPSampleResult` object:
  - 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. On the right, a detailed view of one request is shown under the 'Sampler result' tab.

**Sampler result:**

- 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

**HTTPSampleResult fields:**

- ContentType: application/json
- DataEncoding: null

Comments:											
Write results to file / Read from file											
<input type="text" value="Filename"/> <input type="button" value="Browse..."/> Log/Display Only: <input type="checkbox"/> Errors <input type="checkbox"/> Successes <input type="button" value="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.

Text	Sampler result	Request	Response data
<input checked="" type="checkbox"/> DEL - Cancel a ticket	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

## 4.2 With 50 Threads

## GET - Search a flight

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB.../s	Avg. Byt...
GET - Search Flight	50	17	10	31	5.17	0.00%	49.4/sec	12.00	6.84	249.0
POST - Booking	50	7	4	17	2.93	0.00%	49.7/sec	11.10	25.07	229.0
POST - Search Flight	50	6	4	16	2.24	0.00%	49.7/sec	10.82	15.43	223.0
POST - Book a flight	50	6	4	14	2.12	0.00%	49.7/sec	10.58	23.73	218.0
DEL - Cancel a ticket	50	21	15	35	5.17	0.00%	49.1/sec	5.22	11.26	109.0
GET - history by email	50	14	8	25	3.64	0.00%	49.3/sec	17.61	7.84	366.0
GET - Ticket by pnr	50	13	8	23	2.89	0.00%	49.3/sec	16.36	6.78	340.0
TOTAL	350	12	4	35	6.65	0.00%	326.5/sec	78.98	91.32	247.7

The screenshot shows the JMeter Results Tree View. The left pane displays a list of 20 successful "GET - Search Flight" requests, each marked with a green checkmark. The right pane shows the details for the first request under the "Text" tab. The request information includes:

- 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 this, under "HTTPSampleResult fields:", the Content Type is listed as application/json and the Data Encoding is null.

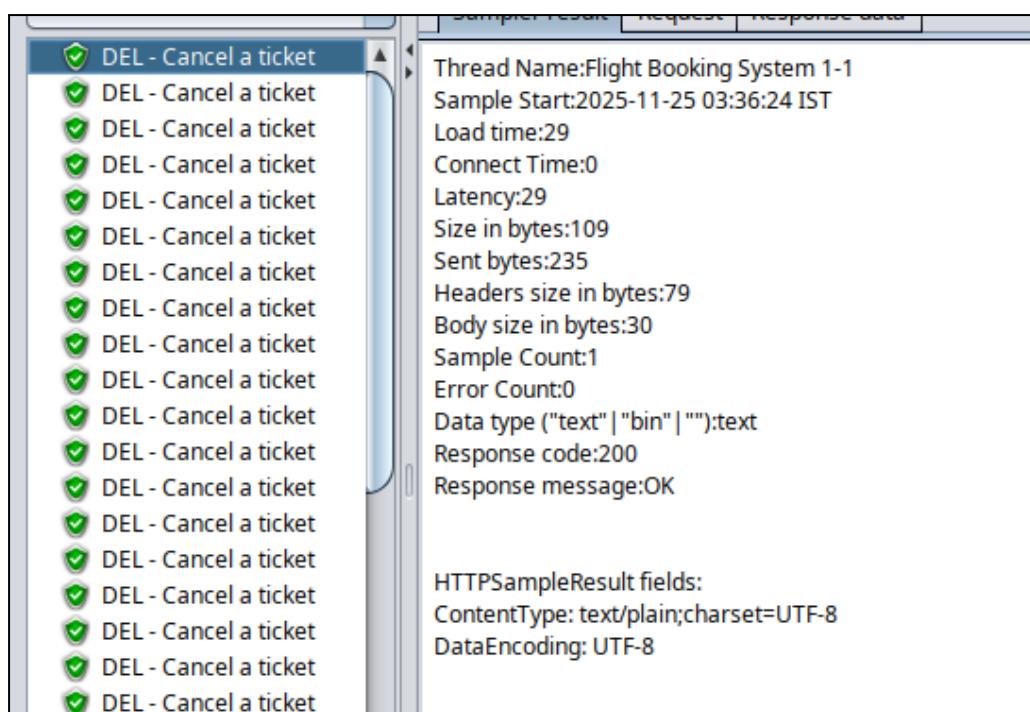
## POST - Book a flight

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

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
HTTPSampleResult fields:	
Content-Type:	application/json

## 4.1 With 100 Threads

DELETE - Cancel a booking

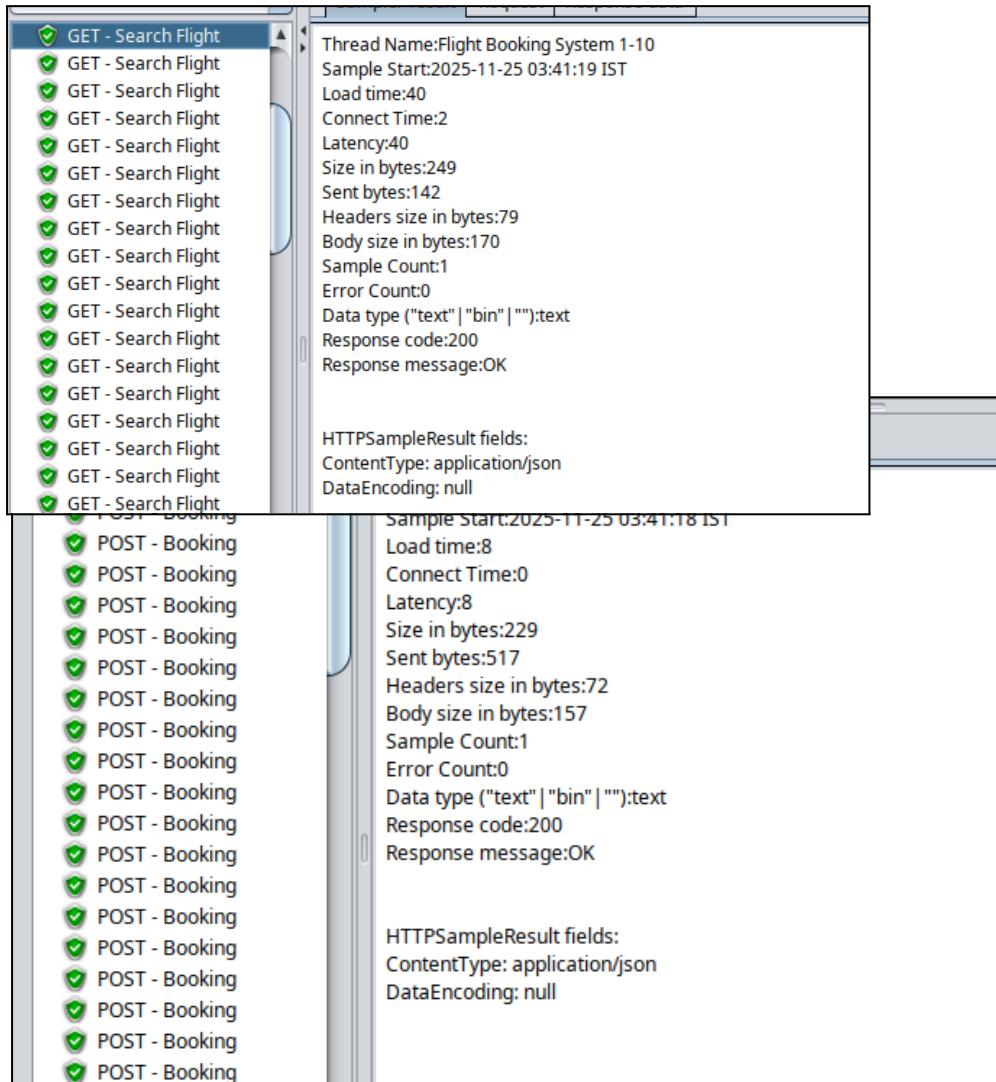


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
DEL - Cancel ...	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0
TOTAL	50	31	19	51	6.74	0.00%	49.4/sec	5.26	11.34	109.0

## 4.3 With 100 Threads

GET - Search a flight

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/sec	Avg. Byt...
GET - Search Flight	100	319	75	723	169.93	0.00%	71.2/sec	17.32	9.88	249.0
POST - Booking	100	72	9	254	47.64	0.00%	73.2/sec	16.36	36.93	229.0
POST - Search Flight	100	63	5	185	39.23	0.00%	75.3/sec	16.40	23.38	223.0
POST - Book a flight	100	65	6	194	45.54	0.00%	75.9/sec	16.15	36.23	218.0
DEL - Cancel a ticket	100	259	55	521	114.03	0.00%	73.0/sec	7.78	16.76	109.0
GET - history by email	100	185	16	517	119.04	0.00%	83.1/sec	29.71	13.23	366.0
GET - Ticket by pnr	100	130	14	414	93.96	0.00%	95.9/sec	31.83	13.20	340.0
TOTAL	700	156	5	723	138.11	0.00%	383.6/sec	92.79	107.29	247.7



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 JMeter interface with the 'Text' listener selected. The left pane lists 20 samples, all marked with a green checkmark, indicating successful execution. The right pane displays the 'Sampler result' tab with 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 these details, under 'HTTPSampleResult fields:', the following information is 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 a POST request to <http://localhost:8080/api/flight/addAirline>. The request body contains the following JSON payload:

```
1 {
2   "airlineCode": "SJ",
3   "airlineName": "Spice Jet "
4 }
```

The response status is 201 Created, and the response time is 377 ms. The response body is shown as an empty array: `[ ]`.

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

The screenshot shows the Postman interface with a GET request to <http://localhost:8080/api/flight/getAllAirlines>. The response status is 200 OK, and the response time is 82 ms. The response body is a JSON array containing two airline objects:

```
1 [
2   {
3     "airlineCode": "TNDG"
4   }
5 ]
```

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

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/api/flight/airline/inventory/add`. The request body contains the following JSON payload:

```
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 }
14
```

The response status is **201 Created** with a response time of 644 ms and a size of 391 bytes. The response body is shown as:

```
{ } JSON ▾
```

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

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

The screenshot shows the Postman application interface. At the top, it says "FlightBooking-WebFlux / Search Flight". Below that, a "POST" method is selected with the URL "http://localhost:8080/api/flight/search". The "Body" tab is active, showing a JSON payload:

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

Below the body, the "Body" tab is selected again, showing the raw JSON response:

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

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

HTTP FlightBooking-WebFlux / Book a flight - One Way

POST <http://localhost:8080/api/flight/booking/6924cd4e3593ecc160c44342>

Docs Params Authorization Headers (8) Body Scripts Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL [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

```

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": "BD6FE9",
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

HTTP FlightBooking-WebFlux / Get Ticket By PNR

GET <http://localhost:8080/api/flight/ticket/1a919a>

Docs Params Authorization Headers (8) Body Scripts Settings 200 OK

Body Cookies Headers (2) Test Results |

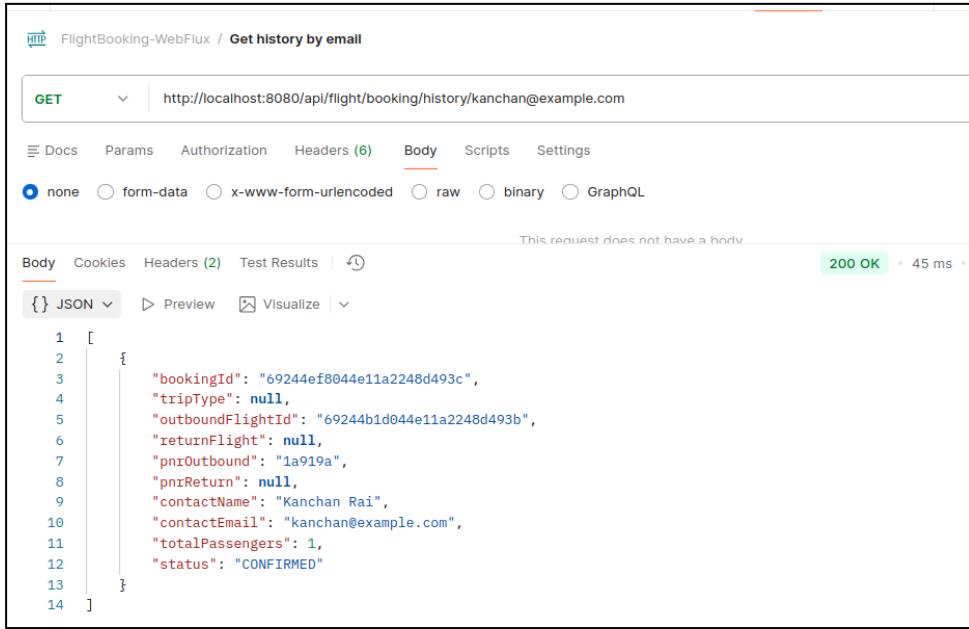
{ } JSON ▾ ▶ Preview Visualize ▾

```

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

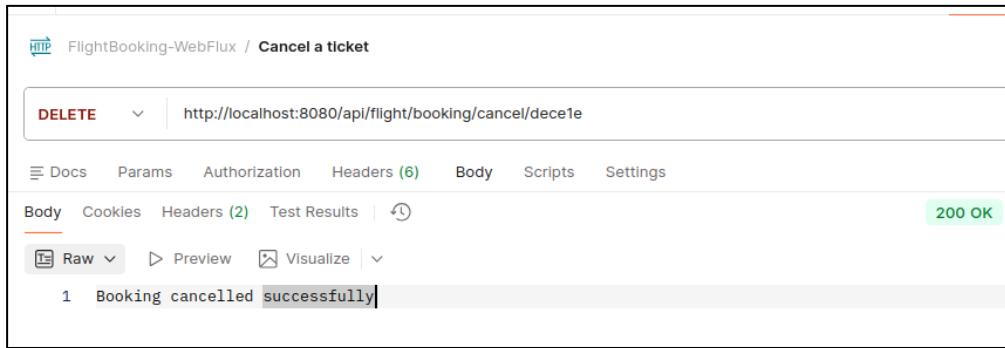


The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** <http://localhost:8080/api/flight/booking/history/kanchan@example.com>
- Headers:** (6) (Body is selected)
- Body:** none
- Test Results:** This request does not have a body.
- Status:** 200 OK (45 ms)
- Body (JSON):**

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

## 8. DELETE - Cancel a ticket



The screenshot shows the Postman interface with the following details:

- Method:** DELETE
- URL:** <http://localhost:8080/api/flight/booking/cancel/dece1e>
- Headers:** (6)
- Body:** Raw
- Test Results:** 200 OK
- Body (Raw):**

```
1 Booking cancelled successfully
```