

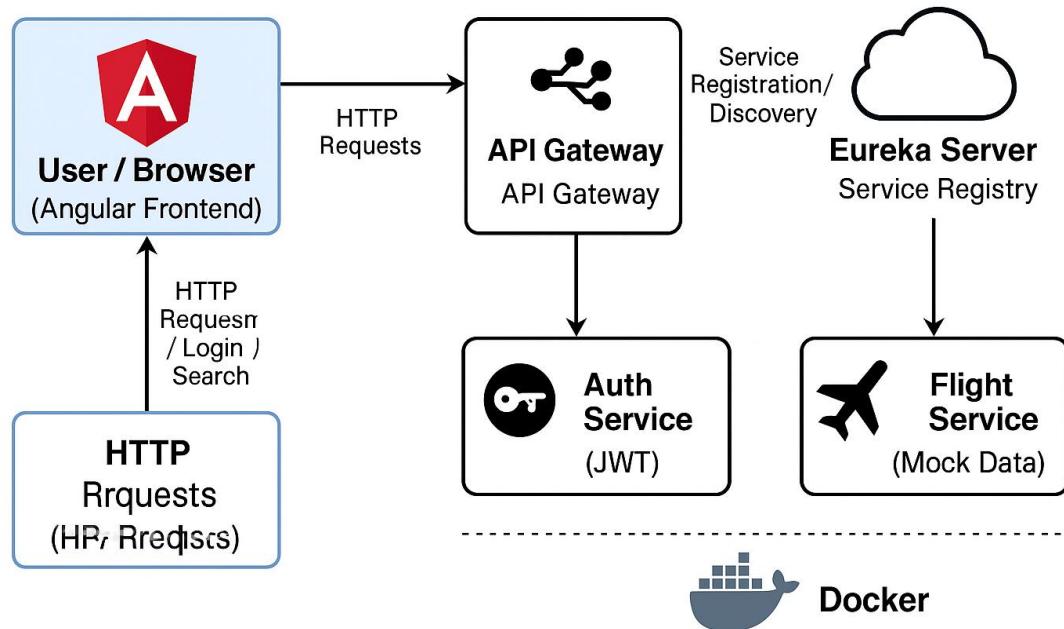
Important Screenshots

Flight Booking Application – Frontend using Angular:

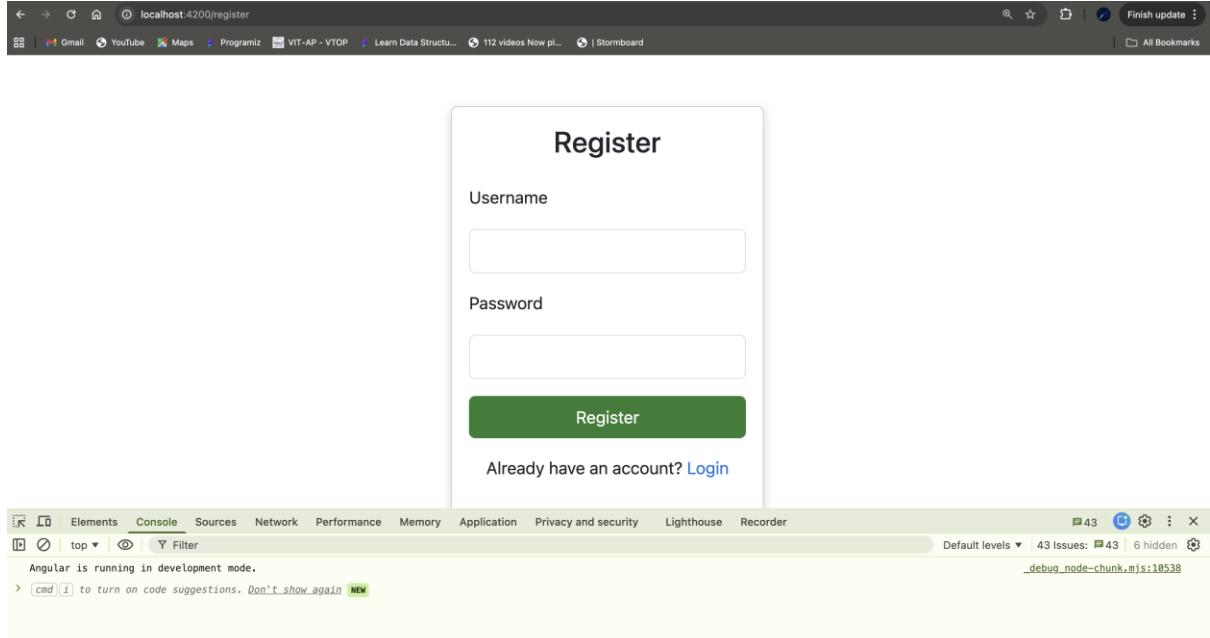
The application provides three main functionalities: User Registration, User Login, and Flight Search. Angular routing is used for navigation between components, and user-friendly forms with validations are implemented.

The application communicates with a backend API for authentication using **JWT (JSON Web Token)**.

Flight Booking Application Architecture:

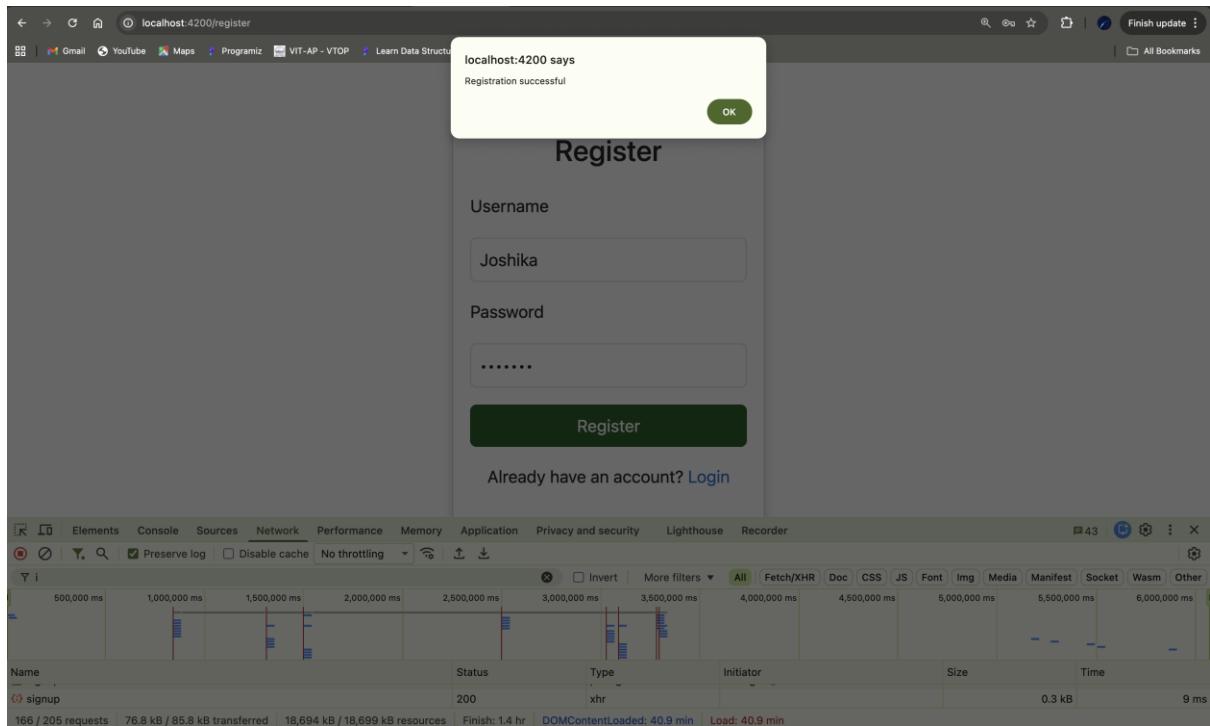


Register Page:



The screenshot shows a web browser window with the URL `localhost:4200/register`. The main content is a registration form titled "Register". It contains two input fields: "Username" and "Password", and a green "Register" button. Below the button is a link to "Login". At the bottom of the browser window, the developer tools are open, specifically the Network tab. It shows a single entry for a POST request to `/register` with a status of "OK". The browser's address bar also shows the URL `localhost:4200/register`.

This is the register page. Now we'll see response after adding username and password.

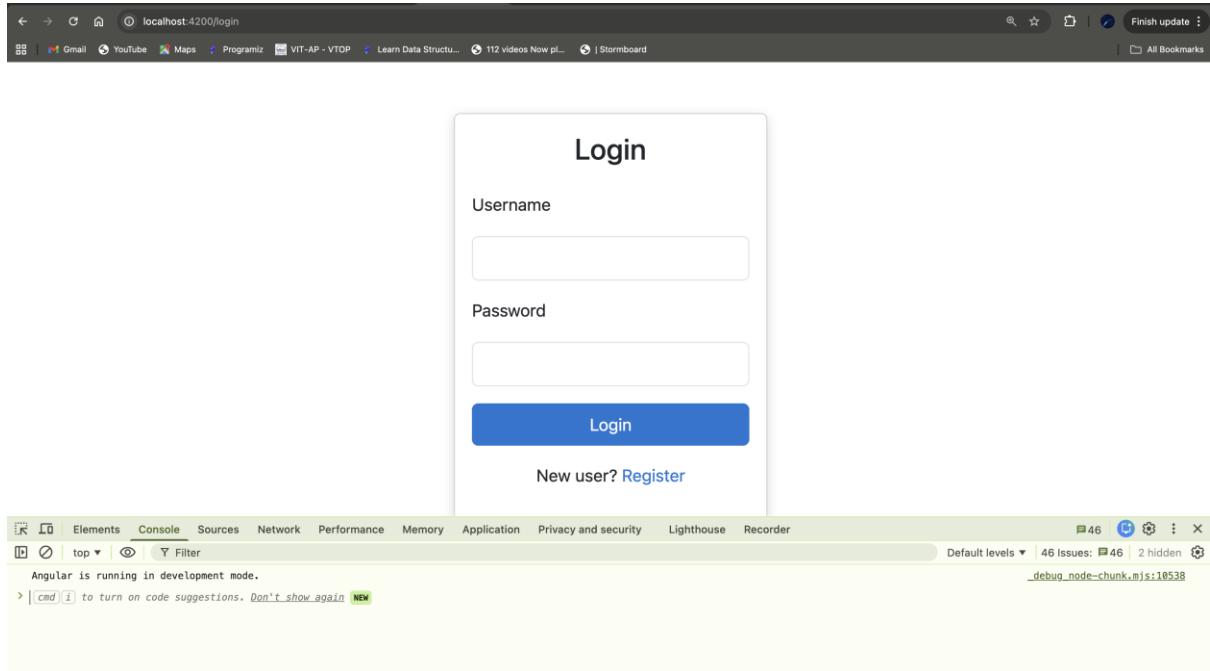


The screenshot shows the same registration form as before, but now with a modal overlay displaying the message "localhost:4200 says Registration successful". The developer tools Network tab shows a POST request to `/register` with a status of 200 OK. The browser's address bar still shows `localhost:4200/register`.

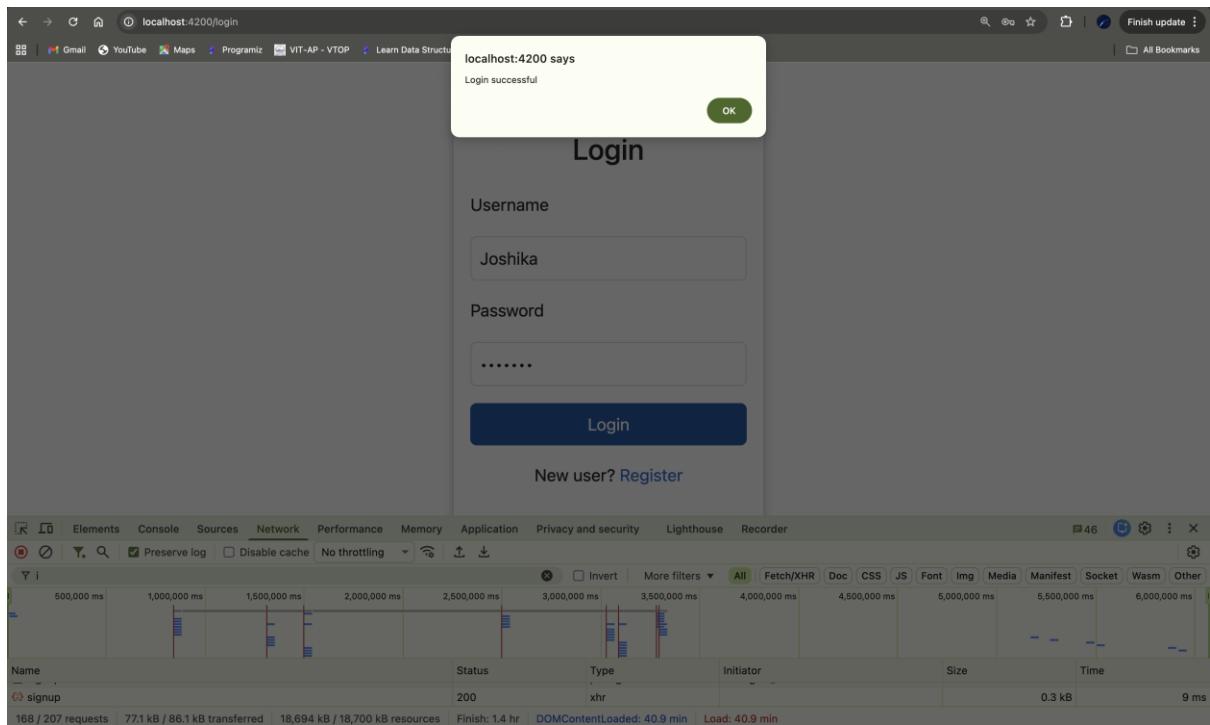
We can see the post status here. It shows that the backend is connected successfully.

The post request shows 200OK!

Login Page:



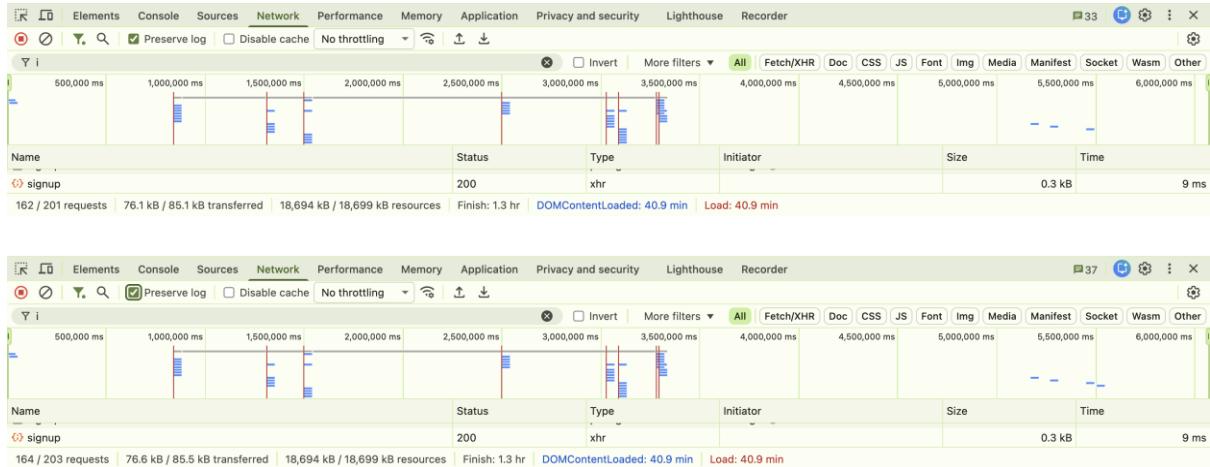
This is the Login page. Now we'll see response after adding username and password.



We can see the post status here. It shows the backend is connected successfully.

The post request shows 200OK!

Backend Connection Proof:



The requests used in backend.

The figure is a screenshot of the Docker Desktop interface. It shows a sidebar with various options like Ask Gordon, Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit, Docker Hub, and Docker Scout. The main area is titled "Containers" and shows a table of running containers. The table includes columns for Name, Container ID, Image, Port(s), CPU (%), Last started, and Actions. The containers listed are: sqlserver (Container ID: 703dd0db363f, Image: mssql/server_2022-latest, Last started: 2 months ago), azureqledge (Container ID: 50f772900a27, Image: azure-sole-edge, Last started: 2 months ago), welcome-to-docker (Container ID: 283962f26e6e, Image: docker/welcome-to-docker, Last started: 2 months ago), flightbookingapplication (Container ID: -, Image: flightbookingapplication, Last started: 4 hours ago), eureka-server (Container ID: c586ac0cd304, Image: flightbookingapplication-eureka-ser, Last started: 4 hours ago), config-server (Container ID: c999b8093232, Image: flightbookingapplication-config-ser, Last started: 4 hours ago), notification-service (Container ID: 19145ddacc54, Image: flightbookingapplication-notification, Last started: 4 hours ago), flight-service (Container ID: 4ab573d869b5, Image: flightbookingapplication-flight-seri, Last started: 4 hours ago), booking-service (Container ID: 162743d1315a, Image: flightbookingapplication-booking-se, Last started: 4 hours ago), and api-gateway (Container ID: 5bb8e3316af8, Image: flightbookingapplication-api-gatewe, Last started: 4 hours ago).

Docker running all the services

The figure is a screenshot of the Chrome DevTools Application tab. The left sidebar shows Manifest, Service workers, Storage, Local storage (with http://localhost:4200 selected), Session storage, Extension storage, and IndexedDB. The main area shows a table with a single entry: Key (token) and Value (a long, encoded string: eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJ0ZXN0dXNlclslmhdCl6MTc2NTczMzI4MSwiZXhwIjoxNzY1NzUxMj...). Below the table, it says "No value selected" and "Select a value to preview".

Token generated after Login.

Flight Search Page:

The screenshot shows a web browser window with the URL `localhost:4200/search`. The main content area is titled "Flight Search". It has two input fields: "Source" containing "Enter source" and "Destination" containing "Enter destination". A blue "Search" button is to the right of the destination field. Below the inputs, the message "No flights found" is displayed.

The screenshot shows the browser's developer tools open to the "Console" tab. The message "Angular is running in development mode." is visible, along with a note about code suggestions and a "Don't show again" link.

This is the Flight search page. Now we'll see response after adding src and dest.

The screenshot shows the same web browser window after entering source and destination. The "Source" field now contains "VJA" and the "Destination" field contains "BLR". The "Search" button is still present. Below the inputs, a table displays the available flights:

Flight No	Source	Destination	Price (₹)
AI101	VJA	BLR	4500
GO404	VJA	BLR	4800

The developer tools console message remains the same as in the previous screenshot.

It shows available flights from source to destination.

EDGE CASES:

If flight not found.

A screenshot of a web browser window titled "Flight Search". The page has two input fields: "Source" containing "ATL" and "Destination" containing "FRC". A blue "Search" button is to the right of the destination field. Below the search form, the text "No flights found" is displayed. The browser's address bar shows "localhost:4200/search".



CODE ADDED IN BACKEND FOR INTEGRATION:

```
package com.apigateway.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.reactive.CorsWebFilter;
import org.springframework.web.cors.reactive.UrlBasedCorsConfigurationSource;

@Configuration
public class CorsConfig {
    @Bean
    public CorsWebFilter corsWebFilter() {
        CorsConfiguration config = new CorsConfiguration();
        config.addAllowedOrigin("http://localhost:4200");
        config.addAllowedMethod("*");
        config.addAllowedHeader("*");
        config.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source =
            new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return new CorsWebFilter(source);
    }
}
```

Working Eureka:

The screenshot shows the Spring Eureka dashboard at localhost:8761. The top navigation bar includes links for Gmail, YouTube, Maps, Programiz, VIT-AP - VTOP, Learn Data Structu..., 112 videos Now pl..., and Stormboard, along with a 'Finish update' button and a 'All Bookmarks' link.

The main content area has a header 'spring Eureka' and a sub-header 'HOME LAST 1000 SINCE STARTUP'. It displays two sections: 'System Status' and 'DS Replicas'.

System Status table:

Environment	test	Current time	2025-12-14T16:57:33 +0000
Data center	default	Uptime	01:39
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	14

DS Replicas table:

localhost
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 5bb8e3316af8:api-gateway:8080
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 162743d1315a:booking-service:8082
CONFIG-SERVER	n/a (1)	(1)	UP (1) - ca99b8093232:config-server:8888
FLIGHT-SERVICE	n/a (1)	(1)	UP (1) - 4ab573d869b5:FLIGHT-SERVICE:8081

General Info table:

Name	Value
------	-------

KAFKA:

```
polakujoshikasree@POLAKUs-Laptop FlightBookingApplication % kafka-topics --bootstrap-server localhost:9092 --list

booking-cancelled
booking-created
bookings-topic
cancellations-topic
polakujoshikasree@POLAKUs-Laptop FlightBookingApplication %
```

Notification:

```
polakujoshikasree@POLAKUs-Laptop FlightBookingApplication % brew services list

Name      Status  User          File
kafka     started polakujoshikasree ~/Library/LaunchAgents/homebrew.mxcl.kafka.plist
mongodb-community started polakujoshikasree ~/Library/LaunchAgents/homebrew.mxcl.mongodb-community.plist
zookeeper started polakujoshikasree ~/Library/LaunchAgents/homebrew.mxcl.zookeeper.plist
polakujoshikasree@POLAKUs-Laptop FlightBookingApplication % kafka-console-consumer --bootstrap-server localhost:9092 --topic booking-events --from-beginning

Booking Successful - PNR: 38A6510D
```

POSTMAN:

SignUp:

The screenshot shows the Postman interface with a collection named "Joshika sree Polaku's Workspace". A POST request is being made to `http://localhost:8080/auth/signup`. The request body is set to raw JSON with the following content:

```
1 {
2   "username": "testuser",
3   "password": "test123"
4 }
```

The response status is 200 OK, with a response time of 41 ms and a size of 182 B. The response body contains the message "Signup success".

Login:

The screenshot shows the Postman interface with the same collection. A POST request is being made to `http://localhost:8080/auth/login`. The request body is set to raw JSON with the following content:

```
1 {
2   "username": "testuser",
3   "password": "test123"
4 }
```

The response status is 200 OK, with a response time of 31 ms and a size of 376 B. The response body is a JSON object containing user information and a token:

```
1 {
2   "username": "testuser",
3   "role": "CUSTOMER",
4   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1302XN0dXNiciIsImhdCi6MtC2NTczNDkzNCwiZXhwIjoxNzY1NzUyOTM0LC3yb2xlcI6Wj3DVVNUT01FU13JdfQ.9ERaIyjBELWLjAoByx975pYfV9U14TmCvcodsVjh-Dc"
```

Logout:

The screenshot shows the Postman interface with the following details:

- Collection:** Joshika sree Polaku's Workspace
- Request Type:** POST
- URL:** http://localhost:8080/auth/logout
- Headers:** (8)
 - Postman-Token: <calculated when request is sent>
 - Content-Length: 0
 - Host: <calculated when request is sent>
 - User-Agent: PostmanRuntime/7.49.1
 - Accept: */*
 - Accept-Encoding: gzip, deflate, br
 - Connection: keep-alive
 - Authorization: Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJhZ... (redacted)
- Body:** (Raw) 1. Logout success
- Status:** 200 OK
- Time:** 22 ms
- Size:** 182 B

MongoDB:(2 databases)

booking_db	20.48 kB	1	1
config	12.29 kB	0	2
flightdb	20.48 kB	1	1

FlightDB:

The screenshot shows the Compass interface connected to localhost:27017, displaying the following details:

- Connections:** localhost:27017
 - admin
 - config
 - flight-db
 - flights
 - flight_booking
 - flightdb
 - flights
- Current Database:** flightdb
- Collection:** flights
- Documents:** 1
- Aggregations:** 0
- Schema:** 1
- Indexes:** 1
- Validation:** 0

The interface includes a search bar, a query builder, and a preview of the document structure:

```
_id: ObjectId("602b227d1a1aa2c6ca9c238")
fromCity: "HYD"
toCity: "DEL"
date: "2021-12-10"
totalSeats: 100
availableSeats: 100
class: "com.flightservice.model.Flight"
```

BookingDB:

The screenshot shows the Compass MongoDB interface. On the left, the 'Connections' sidebar lists several databases, with 'booking_db' selected. Under 'booking_db', the 'bookings' collection is highlighted. The main pane displays a document in JSON format:

```
_id: "ObjectID('692b827d1a16a2c6ca9e238')"
tripType: "ONE WAY"
flightIdForward: "692b827d1a16a2c6ca9e238"
seatNumbersForward: Array (2)
  0: "11B"
  1: "11C"
passenger: Array (2)
  0: Object
    name: "Kavita"
    age: 18
    gender: "Male"
  1: Object
    name: "Sita"
    age: 25
    gender: "Female"
meal: false
bookingDateTime: 2023-11-29T23:33:34.637+00:00
status: "BOOKED"
__class: "com.booking.service.model.Booking"
```

JMeter:

Post request (add inventory):

The screenshot shows the JMeter Test Plan interface. It includes a 'View Results Tree' listener and a 'Summary Report' listener.

View Results Tree:

- Test Plan
- Thread Group
 - HTTP Request
 - HTTP Header Manager
- Summary Report
- View Results Tree

The 'View Results Tree' panel shows a tree of 20 'HTTP Request' samplers. The expanded 'HTTP Request' node shows the following request details:

```
1. POST http://localhost:8080/flights/add
2. POST data
3. {
4.   "flightId": "692b827d1a16a2c6ca9e238",
5.   "seatNumbers": [
6.     "11B",
7.     "11C"
8.   ],
9.   "passenger": [
10.    {
11.      "name": "Kavita",
12.      "age": 18,
13.      "gender": "Male"
14.    },
15.    {
16.      "name": "Sita",
17.      "age": 25,
18.      "gender": "Female"
19.    }
20.  ],
21.   "meal": false,
22.   "bookingDateTime": "2023-11-29T23:33:34.637+00:00",
23.   "status": "BOOKED"
24. }
```

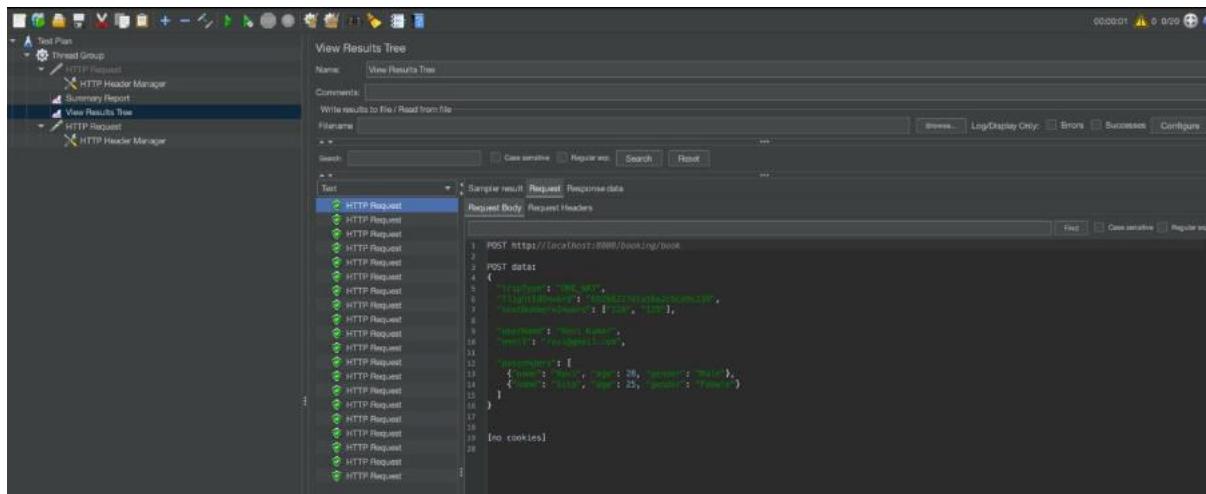
Summary Report:

- Test Plan
- Thread Group
 - HTTP Request
 - HTTP Header Manager
- Summary Report
- View Results Tree

The 'Summary Report' panel displays the following summary statistics:

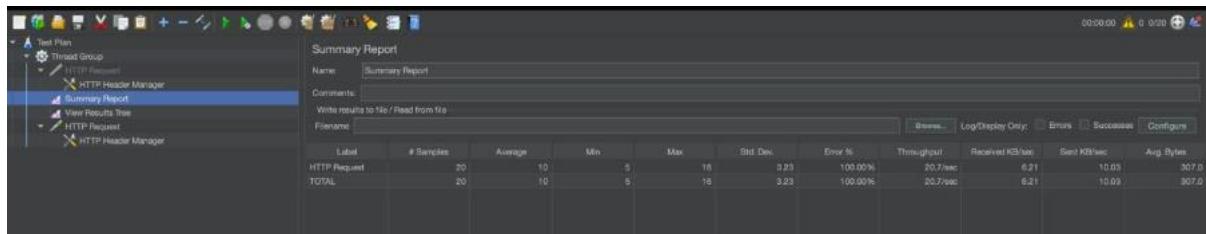
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	20	13	11	18	1.97	0.00%	20.7/sec	5.16	6.61	255.0
TOTAL	20	13	11	18	1.97	0.00%	20.7/sec	5.16	6.61	255.0

Add Booking :



The screenshot shows the JMeter interface with the 'View Results Tree' listener selected. The tree view displays a single POST request under the 'HTTP Request' node. The request details are as follows:

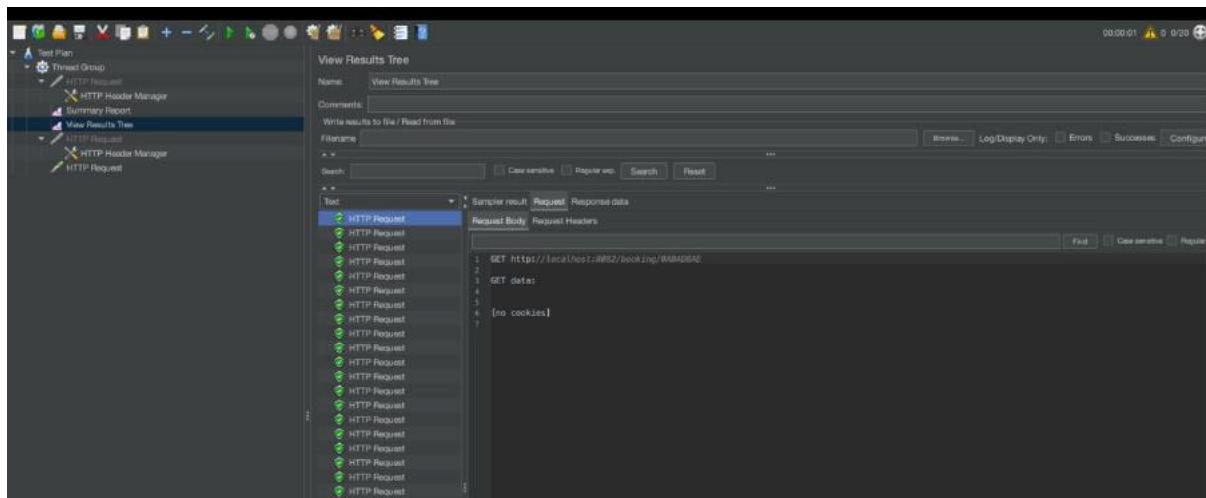
```
1 POST http://localhost:8080/booking/book
2 POST data:
3 {
4     "type": "req_type",
5     "id": "11111111111111111111111111111111",
6     "status": "status1",
7     "name": "Book Name",
8     "time": "2019-01-01T00:00:00Z",
9     "amount": 100.0,
10    "details": [
11        {"name": "Name", "age": 28, "gender": "Male"},
12        {"name": "Name", "age": 25, "gender": "Female"}
13    ],
14    "no_cookies": []
15 }
```



The screenshot shows the JMeter interface with the 'Summary Report' listener selected. The summary table provides an overview of the test execution:

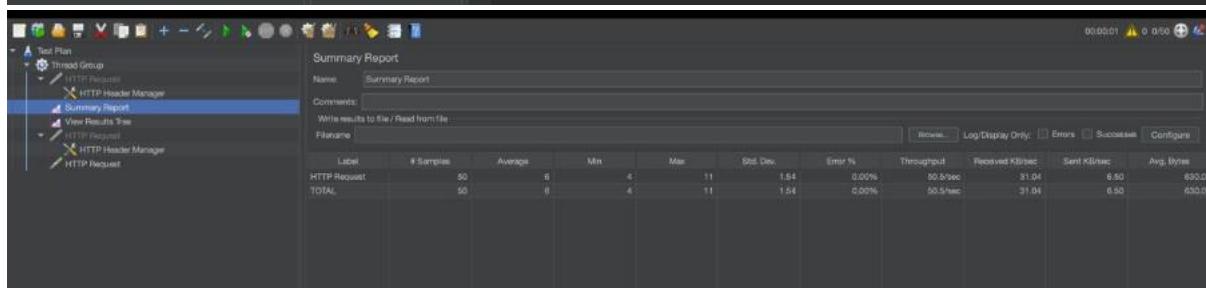
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	20	10	5	15	3.23	100.00%	20.7/sec	6.21	10.03	307.0
TOTAL	20	10	5	15	3.23	100.00%	20.7/sec	6.21	10.03	307.0

Get PNR:



The screenshot shows the JMeter interface with the 'View Results Tree' listener selected. The tree view displays a single GET request under the 'HTTP Request' node. The request details are as follows:

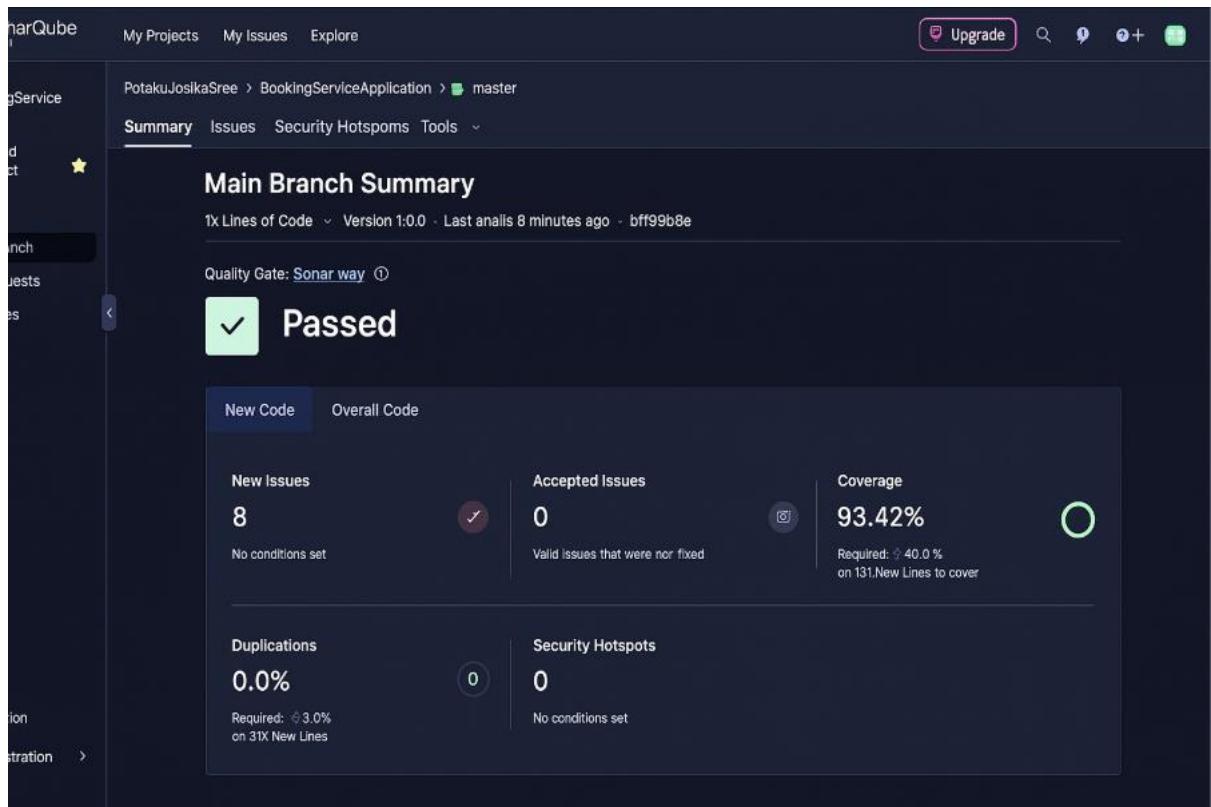
```
1 GET http://localhost:8080/booking/BA8M0B8E
2 GET data:
3
4
5
6 [no cookies]
7
```



The screenshot shows the JMeter interface with the 'Summary Report' listener selected. The summary table provides an overview of the test execution:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	50	0	0	11	1.54	0.00%	10.5/sec	31.04	8.50	650.0
TOTAL	50	0	0	11	1.54	0.00%	10.5/sec	31.04	8.50	650.0

Sonar Analysis:



CONCLUSION:

- Successfully implemented Angular frontend
- Authentication integrated with backend
- Clean UI and component-based design