1. Introduction

I have attached a zip file that contains list of services. Below design can be verified with my services code.

This document details the backend design of given assignment that is ticket booking for movie or some concert. I have tried to keep the services as simple as possible. I have done rest calls instead of event driven architecture for simplicity purpose.

2. Architecture Overview

I have followed **Microservices Architecture**, ensuring modularity, scalability, and fault tolerance. Each service is responsible for a specific domain and communicates via **REST APIs** and **Eureka Service Discovery**.

Microservices and Dependencies

Order	Service Name	Key Dependencies	Port
1	Discovery Service	-	8761
2	Auth Service	Discovery Service	8080
3	Gateway Service	Discovery Service, Auth Service	8081
4	User Service	Discovery Service, Gateway Service, Auth Service	8082
5	Event Service	Discovery Service, Gateway Service, Auth Service	8083
6	Booking Service	Discovery Service, Gateway Service, Auth Service	8084
7	Payment Service	Discovery Service, Gateway Service, Auth Service, Booking Service	8085
8	Notification Service	Discovery Service, Gateway Service, Auth Service, Booking Service	8086

Technology Stack:

- **Spring Boot** (Microservices)
- **Spring Cloud Eureka** (Service Discovery)
- **Spring Security & JWT** (Authentication)
- **Spring Data JPA & Hibernate** (Database ORM)

- **MySQL** (Relational Database)
- **Redis** – It can be implemented in different part of application
- **Kafka or RabbitMQ** – can be used for notification
- **Docker & Kubernetes** (Containerization and Orchestration)- I have not covered this one for now, we can discuss later

3. Service Functionality

1️⃣ Discovery Service (Eureka Server)

- Provides service registry and discovery.
- Allows microservices to register and discover each other dynamically.

2️⃣ Auth Service

- Handles user authentication via **JWT tokens**.
- Provides login and token validation endpoints.
- Secure communication with other services using **Spring Security**.
- I have not used Oauth2 providers such key cloak or different clients. For simplicity I have used a HMACSHA256 secret key to generate JWT tokens and verfication.

3️⃣ API Gateway

- Centralized entry point for all services.
- Performs authentication and request routing.

4️⃣ User Service

- Manages user profiles and preferences.
- Provides endpoints to sign up a user, fetch user details, delete user and get all users. During signup activity I have kept few things such as unique username, encrypted password is being saved in table.

5️⃣ Event Service

- Stores details about movies, concerts, and events.
- Provides events (movie, concerts) search, creation and deletion functionalities.

- We can add, manipulated Venues and in those Venues different events can be organized.

6 Booking Service

- Handles booking requests.
- Validates seat availability and processes reservations.
- Calls **Payment Service** for transaction processing. (I'm using restTemplate)
- After successful calls notification service to send notifications to users. (This can be done via async way using queues as well). For simplicity (I'm using restTemplate)

7 Payment Service

- Processes payments and stores transaction details.
- Sents back response to booking service, so that booking service updates booking status to SUCCESS.
- In payment service different 3rd party vendors can be integrated such Razorpay, Jusypay etc. In future we can implement different type of wallet functionalities can be implemented such as user wallet, bonus wallet or reward wallet.

8 Notification Service

- Sends confirmation emails/SMS after successful booking.
- Uses **Spring Boot Mail** for email notifications.
- We can also implement different type of notification providers such WhatsApp, SMS etc.

Note –

I have not implemented few validations in code in order to save time. I have used JWT based authentication only in auth-service, gateway-service and user –service for demonstration purpose. Remaining other services, I have made all Apis without JWT in order to save time for this test.

To monitor services health, I have added actuator dependency that exposes health endpoints. Swagger API documentation.

I have not implemented different monitoring services such ELK, CloudWatch, Prometheus. I can implement later on.

4. API Design

1️⃣ Discovery Service API(default endpoint)

Endpoint: http://localhost:8761/

2️⃣ Gateway API (

Endpoint: http://localhost:8081

All request route through the gateway. All of my services can be accessed via gateway service as well.

3️⃣ Auth API

Endpoint: http://localhost:8080/auth

a) POST http://localhost:8080/auth /login

{ "username": "kisan", "password": "firstpassword" }

- User can get the valid token.

b) GET http://localhost:8080/auth /validate-token

Authorization:Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJraXNhbiIsImlhdCI6MTczODUyMDYwOCwiZXhwIjoxNzM4NTU2NjA4fQ.9YPCBvVrDrA8zwb1OGPfzTnG6Xy605Z-9CmmYw__LkdLaU62YfrMZp4cmT1_kOtk_n910Cgvw9O2FX33or93Aw

– User can validate their token whether it is valid or not.

4 User Service API

Endpoint: http://localhost:8082/users/

a) POST http://localhost:8082/users//signup

{ "username": "kisan1", "email": "kisankumarvimal185.com", "password": "firstpassword", "role": "ADMIN" }

Create a new user using this endpoint. It is an open API. Assigning role can be done in other way as well.

b)GET http://localhost:8082/users/{userid} - Fetch given user details.

c) DELETE http://localhost:8082/users/{userid} - Delete given user details.

d)GET http://localhost:8082/users - Fetch all user details.

One more PUT endpoint should be added in order to modify the details but I have skipped it.

5 Event Service API – (movies, venue, concert management)

Endpoint:

http://localhost:8083/events (To register events such movie concert)

http://localhost:8083/venues (To register different venues)

a) POST http://localhost:8083/events

{ "name": "Arijit Singh Fever", "description": "A singing event", "eventDate": "2025-03-11T18:30:00", "venueId": 4, "availableSeats": 1000 }

A type of event can be created.

b)GET http://localhost:8083/events{id} - Fetch given event details.

c) DELETE http://localhost:8083/events/{id} - Delete event details.

d)GET http://localhost:8083/events - Get all events details.

a) POST http://localhost:8083/venues

{ "name": "PVR", "location": "Select City Walk Saket Delhi" }

Venue registration Api.

b)GET http://localhost:8083/venues /{id} - Fetch given venue details.

c) DELETE http://localhost:8083/venues/{id} - Delete venue details.

d)GET http://localhost:8083/venues - Get all venue details.

e) PUT will implement later

6 Booking API

Endpoint: http://localhost:8084/bookings

a) POST http://localhost:8084/bookings

{ "userId": 5, "eventId": 4, "venueId": 4, "numberOfSeats": 10, "amount": 1000.0 }

A user can do booking for a particular event.

b)GET http://localhost:8084/bookings {id} - Fetch given booking details.

c) DELETE http://localhost:8084/bookings {id} - Delete given booking details.

d)GET http://localhost:8084/bookings /users/{user_id} - Fetch all booking details for a given user.

7️⃣ Payment API

Endpoint: http://localhost:8085/payments

a) POST http://localhost:8085/payments/process

{ "bookingId": 123456, "amount": 250.75, "status": "SUCCESS", "transactionId": "TX1234567890", "paymentDate": "2025-02-03T10:30:00" }

Process payment for a booking.

b) GET http://localhost:8085/payments/{transaction_id}

For a given transaction payment details can be fetched.

8️⃣ Notification API

Endpoint: http://localhost:8086/notifications

POST http://localhost:8086/notifications/send

{ "recipientEmail": "infoexpertt111@gmail.com", "subject": "Notification Subject", "message": "This is the body of the notification message", "timestamp": "2025-02-03T12:00:00" }

Send booking confirmation.

5. Database Design

I have use MySql database for now. NoSQL databases can also be used for storing different aspect of applications because our entities may evolve.

Schema Overview

User Table

Column Name	Type	Description
id	BIGINT	Primary Key
username	VARCHAR	Unique user
password	VARCHAR	Hashed password
email	VARCHAR	
role	VARCHAR	

Events Table

Column Name	Type	Description
id	BIGINT	Primary Key
name	VARCHAR(255)	Unique Event Name
description	TEXT	Detailed description of the event
event_date	TIMESTAMP	Date and time of the event
venue_id	BIGINT	Foreign Key referencing the venues table
available_seats	INT	Number of available seats for the event

Venues Table

Column Name	Type	Description
id	BIGINT	Primary Key
name	VARCHAR(255)	Unique Venue Name
location	VARCHAR(255)	Location of the Venue

Bookings Table

Column Name	Type	Description
id	BIGINT	Primary Key
user_id	BIGINT	Foreign Key referencing users(id)
event_id	BIGINT	Foreign Key referencing events(id)
venue_id	BIGINT	Foreign Key referencing venues(id)
number_of_seats	INT	Number of seats booked
status	VARCHAR(20)	Booking status (CONFIRMED, PENDING, CANCELLED)
booking_time	TIMESTAMP	Timestamp of booking
amount	DOUBLE	Total booking amount

Payments Table

Column Name	Type	Description
id	BIGINT	Primary Key
booking_id	BIGINT	Foreign Key referencing bookings(id)
amount	DOUBLE	Payment amount
status	VARCHAR(255)	Payment status
transaction_id	VARCHAR(255)	Unique transaction identifier
payment_date	DATETIME	Timestamp of payment

Notifications Table

Column Name	Type	Description
id	BIGINT	Primary Key
recipient_email	VARCHAR(255)	Email of the recipient
subject	VARCHAR(255)	Subject of the notification

message	TEXT	Notification message content
timestamp	DATETIME	Time when the notification was sent

6. Scalability & Performance

- **Load Balancing**: Using **NGINX/Kubernetes** to distribute load.
- **Caching**: Using **Redis** for frequently accessed data.
- **Database Indexing**: Optimized queries using proper indexes.
- **Asynchronous Processing**: Using **Kafka/RabbitMQ** for notifications.
- All of the above will be implemented in order to ensure seamless experience.

7. Authentication & Authorization

- **JWT-based authentication** for secure access.
- **API Gateway filters** to authenticate requests.

8. Real-Time Data Handling -

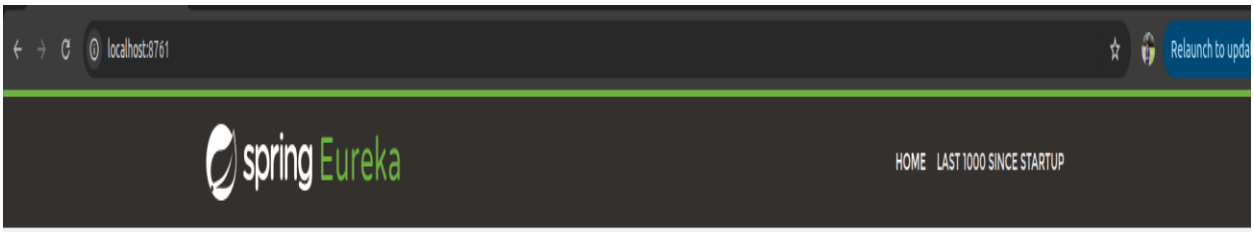
- **Event-driven updates** can be implemented using Kafka.
- Other strategies are there.

9. Conclusion

Overall, I have tried to keep it as simple as possible. This backend design efficiently handles event booking with secure authentication, optimized database queries, and scalable microservices. The system will ensure **fault tolerance**, **high availability**, and **efficient processing** for a seamless user experience when we implement distributed servers, databases and other policies.

Please refer -

Below are some screenshots for tables and services running on my local Linux machine.



System Status

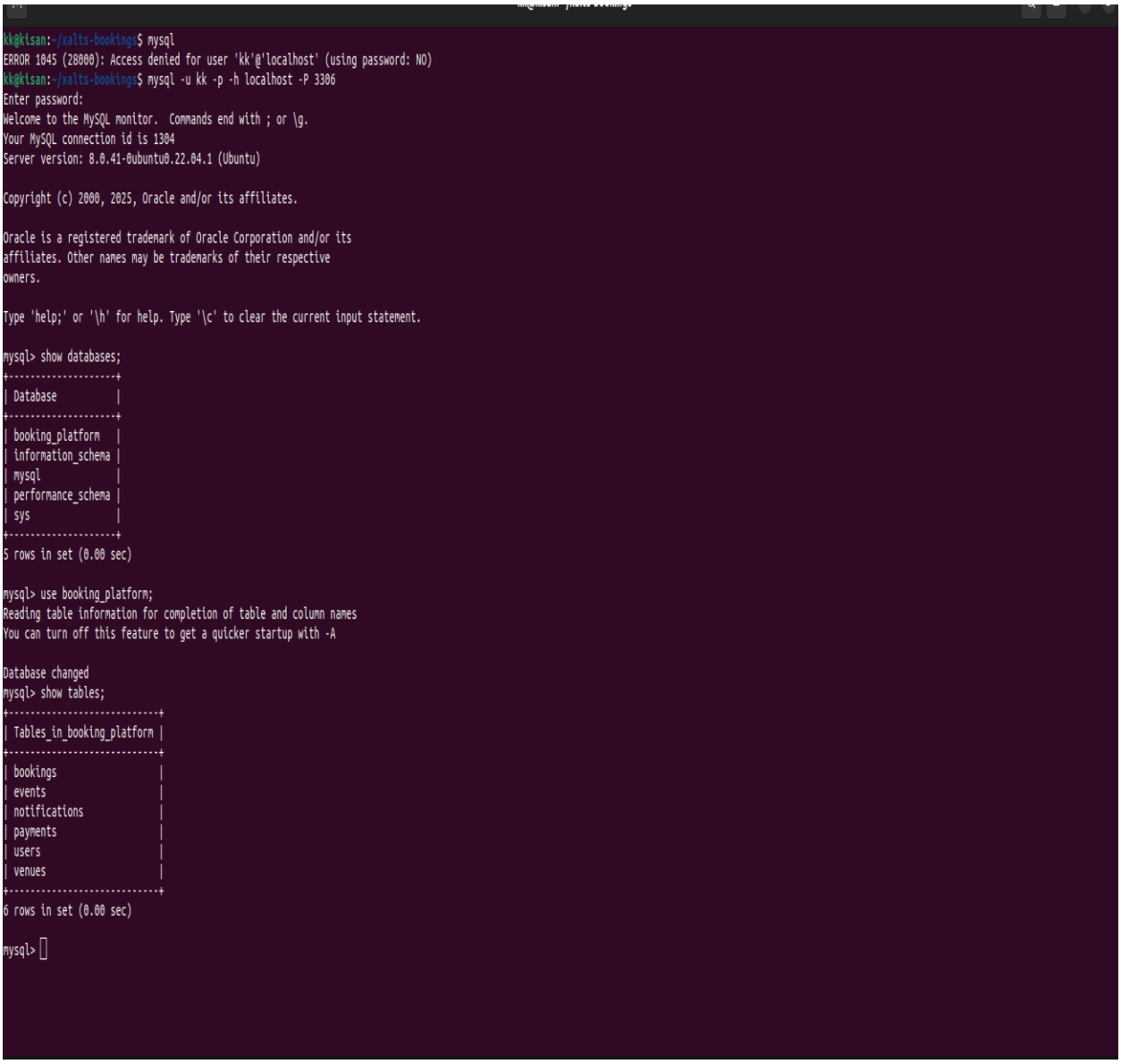
Environment	test	Current time	2025-02-04T03:06:27 +0530
Data center	default	Uptime	00:30
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	16

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AUTH-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.7:auth-service
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.7:booking-service:8084
EVENT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.7:event-service:8083
USER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.7:user-service:8082


kk@kisan:~/xalts-bookings$ mysql
ERROR 1045 (28000): Access denied for user 'kk'@'localhost' (using password: NO)
kk@kisan:~/xalts-bookings$ mysql -u kk -p -h localhost -P 3306
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1304
Server version: 8.0.41-0ubuntu0.22.04.1 (Ubuntu)
Copyright (c) 2000, 2025, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| booking_platform |
| information_schema |
| mysql |
| performance_schema |
| sys |
+--------------------+
5 rows in set (0.00 sec)
mysql> use booking_platform;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> show tables;
+---------------------------+
| Tables_in_booking_platform |
+---------------------------+
| bookings |
| events |
| notifications |
| payments |
| users |
| venues |
+---------------------------+
6 rows in set (0.00 sec)
mysql>

```
6 rows in set (0.00 sec)

mysql> select * from users;
+----+----------+--------------------------------------------------------------+---------------------+-------+
| id | username | password                                                     | email               | role  |
+----+----------+--------------------------------------------------------------+---------------------+-------+
|  4 | kisan    | $2a$10$gkT3ttxT7p5L93YF2ZFw9e0JU0TnT/eC/S6CWVLRzTPm3u0B6Sdsy | kisankumarvimal185.com | ADMIN |
|  5 | kisan1   | $2a$10$3CyFcT81Rg6pP4JCLvisXOgY1y7/mOALHB4Fj8WV4iFhJeIS3DYEa | kisankumarvimal185.com | ADMIN |
+----+----------+--------------------------------------------------------------+---------------------+-------+
2 rows in set (0.01 sec)

mysql> select * from venues;
+----+----------------+---------------------------+
| id | name           | location                  |
+----+----------------+---------------------------+
|  1 | Cinepolis      | V3S Nirman Vihar Delhi    |
|  2 | Cinepolis Vision | CP Delhi                |
|  3 | Inox           | Ambience Mall Gurgaon     |
|  4 | PVR            | Select City Walk Saket Delhi |
+----+----------------+---------------------------+
4 rows in set (0.00 sec)

mysql> select * from events;
+----+---------------------------+-------------------------+---------------------+----------+-----------------+
| id | name                      | description             | event_date          | venue_id | available_seats |
+----+---------------------------+-------------------------+---------------------+----------+-----------------+
|  1 | Interstellar Movie Night  | Sci-fi thriller screening | 2025-04-15 13:30:00 |        2 |             150 |
|  2 | Avengers: Doomsday        | Marvel movie releasing  | 2025-03-10 13:00:00 |        1 |             100 |
|  3 | Captain Ameria vs Red Hulk | Marvel movie releasing | 2025-02-14 13:00:00 |        3 |             100 |
|  4 | Arijit Singh Fever        | A singing event         | 2025-03-11 13:00:00 |        4 |            1000 |
+----+---------------------------+-------------------------+---------------------+----------+-----------------+
4 rows in set (0.00 sec)

mysql> select * from bookings;
+----+---------+----------+----------+-----------------+-----------+---------------------+--------+
| id | user_id | event_id | venue_id | number_of_seats | status    | booking_time        | amount |
+----+---------+----------+----------+-----------------+-----------+---------------------+--------+
|  3 |       5 |        4 |        4 |              10 | PENDING   | 2025-02-02 23:08:18 |   1000 |
|  4 |       5 |        4 |        4 |              10 | PENDING   | 2025-02-02 23:09:01 |   1000 |
|  5 |       5 |        4 |        4 |              10 | PENDING   | 2025-02-02 23:09:17 |   1000 |
|  6 |       5 |        4 |        4 |              10 | PENDING   | 2025-02-02 23:12:31 |   1000 |
|  7 |       5 |        4 |        4 |              10 | CONFIRMED | 2025-02-02 23:17:16 |   1000 |
+----+---------+----------+----------+-----------------+-----------+---------------------+--------+
5 rows in set (0.00 sec)

mysql> select * from payments;
+----+------------+--------+---------+--------------------------------------+---------------------+
| id | booking_id | amount | status  | transaction_id                       | payment_date        |
+----+------------+--------+---------+--------------------------------------+---------------------+
|  2 |          3 |   1000 | SUCCESS | 1098ee7e-957f-443f-949d-819e068b5b4b | 2025-02-02 23:08:18 |
|  3 |          4 |   1000 | SUCCESS | 334bb3ef-37e6-449b-9574-3aed3fbdc1f8 | 2025-02-02 23:09:01 |
|  4 |          5 |   1000 | SUCCESS | 52b160c0-5000-4eb0-a242-4f2e3702af7d | 2025-02-02 23:09:17 |
```