Online Ticket Booking System

Key goals it wants accomplished as part of its solution:

- Enable theatre partners to onboard their theatres over this platform and get access to a bigger customer base while going digital.
- Enable end customers to browse the platform to get access to movies across different cities, languages, and genres, as well as book tickets in advance with a seamless experience.

Functional Requirements

B2B

1.) A new theater can be onboarded to the ticket booking platform.
    a. Add/Update/Delete details of number of halls in theater.
    b. Add/Update/Delete sitting arrangement for different halls.
    c. Add/Update/Delete show details on ticket booking platform.
2.) Ticket booking platform should be able to get updates of shows/movies/bookings from Theateres having their own IT system.

B2C

1.) Booking platform should display list of cities to the user having theaters registered with platform. Use should be able to select the city.
2.) User should be able to browse the list of movies currently running in theaters in selected city.
3.) Use should be able to select the format (2D, 3D, Imax 4D etc.) and language for the movie.
4.) Once user selects movie details, he should be able to see the list of theaters running that movie along with show times. By default, current date should be selected if any show is available during that day and time. Otherwise, next day should be pre-selected.
5.) User should be able to select a particular show and number of seats he wants to book. User should be able to book multiple tickets at a time (at max 10)
6.) User should be displayed the seats layout.
7.) User should be able to see which seats are available to book and which are already booked.
8.) Once the user selects the seats and proceeds, seats should be blocked for 10 minutes before payment.
9.) User should be able to make the payment and finalize the booking.
10.) User should receive booking success on SMS, Email.
11.) User should be able to see the ticket details in platform.
12.) User should be able to see the booking history.

Non-Functional Requirements

1) Low latency – Browse, search features should be seamless without delays. Use should be able to book tickets without any delays.
2) High Availability – System should be highly available for overall customer experience.
3) High concurrency – Multiple users will try to book seat at the same, system should be able to handle this gracefully.
4) Fault tolerant
5) ACID complaint – As booking and payment transactions involved, system should comply with ACID properties for transactions.
6) Scalable – During high demand and popular releases system should be scalable

Out of Scope

1) No more than 10 seats can be booked at a time.
2) Either all seats are booked, or nothing gets booked in case transaction failure.
3) User authorization and authentication.

Storage Estimation

Assumptions

100 million visits per month
10 million tickets sold per month

Number of cities – 500
Number of theatres per city – 5
Number of seats per theatre – 1000
Number of shows per day - 4

Storage needed to store theatre and show details – 150 bytes
Storage needed to store movie details – 100 bytes
Storage needed per booking – 100 bytes

Storage for booking per day = 500 (cities) * 5 (theatres) * 1000 (seats) * 4 (shows) * 250 bytes = 2.5 GB per day

Storage for 10 years = 10 TB approx.

Storage for movies, user details etc is included in overall approximation.

Use Cases

Platform Admin

- Add/Modify/Delete show
- Add/Modify/Delete movie
- Search movie: To search for any particular movie based on the given criteria (title, language, genre, release date)

Customer

- Search movie: Search latest release movies based on language, genre etc.
- Search theatre and shows running in city for chosen date.
- Filter shows based on language, format etc.
- Create/Modify/View/Cancel booking: To create or cancel a booking for any show of a movie, and to view or modify the booking details for any show of a movie
- Reserve a seat: To reserve a seat from the available seats for any show of a movie in a particular city
- Pay using credit card/cash: To pay the movie ticket fee via credit card or cash

Theatre Owners

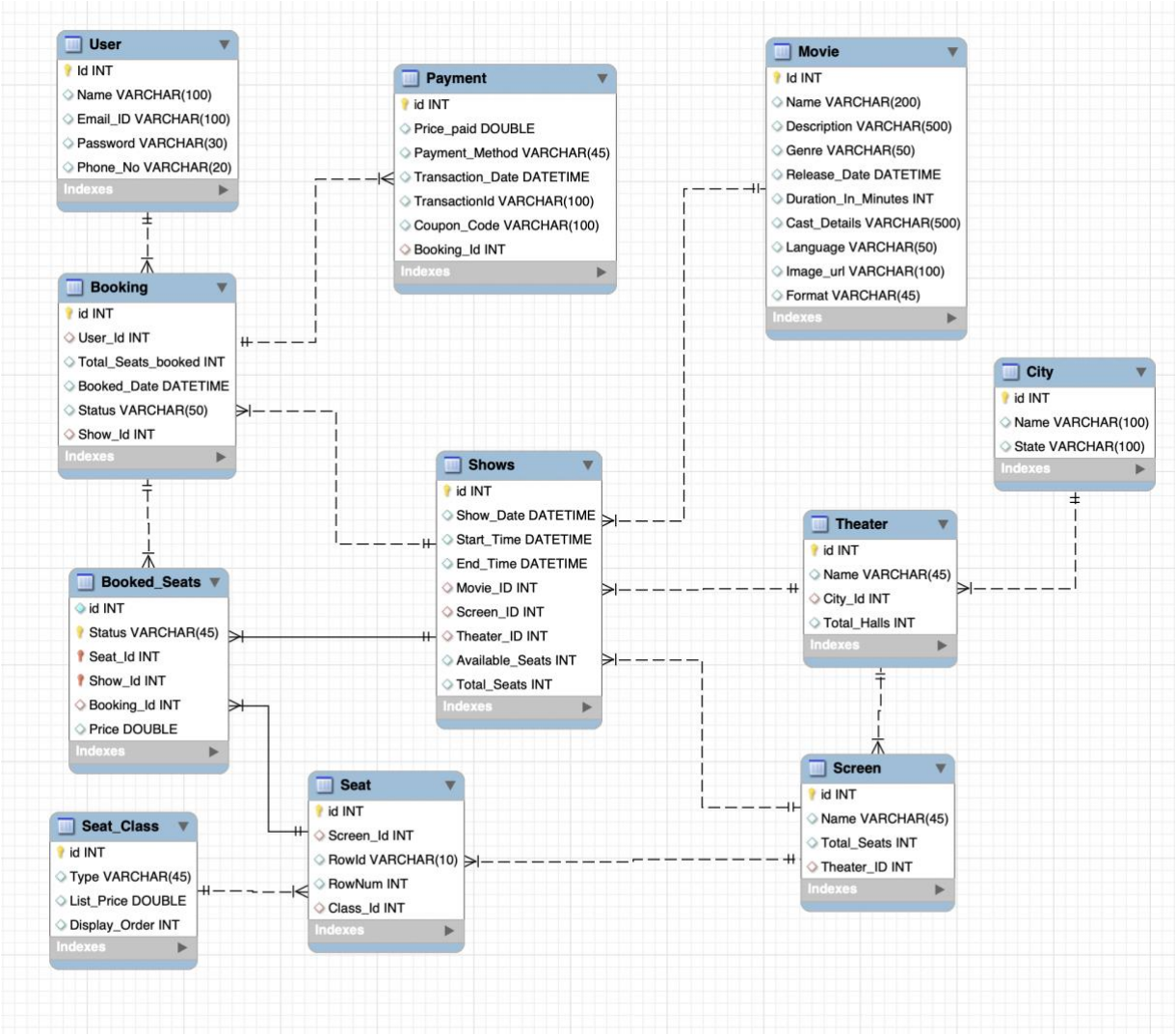
- Create/Modify/Delete shows of the movies running in theatres.

Other Features

- Send email/sms notification on successful booking.
- Ticket generation and billing
- Polling service to poll show details and booking details from theaters registered with the platform.

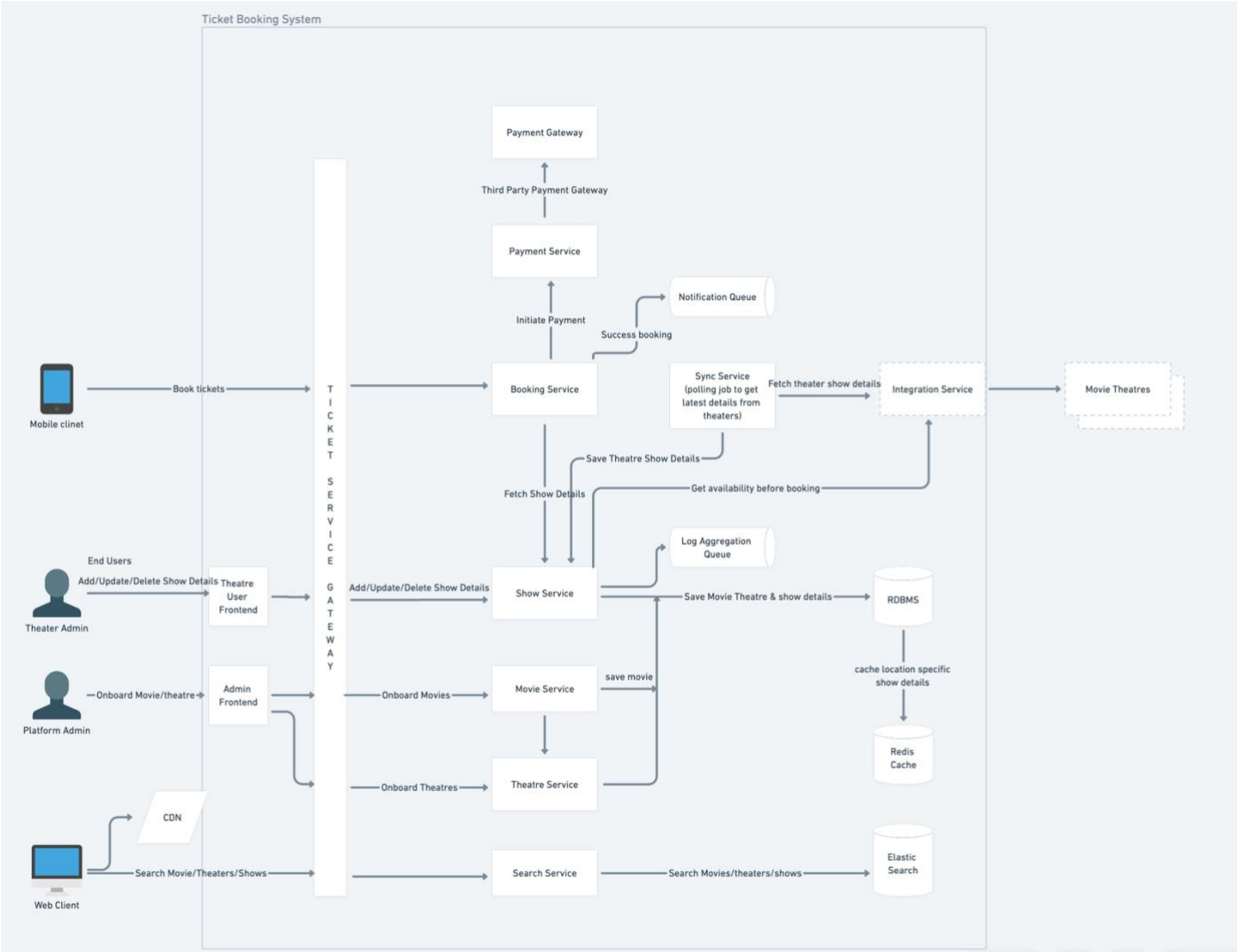
Database Design

1.) Each city will be mapped to multiple theaters.
2.) Each theater will have basic details like total screens, name etc.
3.) Each theater will have multiple screens.
4.) Each screen will have multiple seats.
5.) Each seat will have a seat class tagged to it (Platinum, Gold, Silver etc). Seats will be designated with row and column value.
6.) Seat class will have a price details and order in which they should be displayed in booking system.
7.) Shows will be tagged to a theater, screen, and movie.
8.) Each user details will be captured in user table.
9.) All booking information will be stored in booking table. Each user can do multiple booking.
10.) Each booking will have booked seats stored in separate table and other booking information like booked date, number of seats booked etc.
11.) Locked seats table will have status column to depict if seat is LOCKED, RELEASED or RESERVED.
12.) Payment details like payment method type and unique transaction id will be stored in Payment table. Each booking will be mapped to payment information.



High Level System Architecture

At a high level, application microservice based architecture to manage ticket system transactions. Each service will interact each other in synchronous or asynchronous manner to allow various system users to perform their tasks.



Technology Choices

- Microservices are developed using Java, Spring, SpringBoot, Hibernate and other Java based frameworks/libraries.
- RDBMS database MYSQL to be used for saving data and it can be partitioned based on regions. Postgresql and Oracle can also be leveraged.
- Caching purpose Redis will be used.
- Elastic search DB will be used for search functionality.
- Standalone applications (e.g. a scheduled jobs) can be developed using Python, Node.js etc
- AWS cloud and its various managed services like API gateway, load balancers, ECS, SNS, SQS etc can be used as cloud provider.

Detailed component description

Front end UI Components

Web Client – Frontend based application which will be hosted as separate application. Application will interact with backend services using Ticket service gateway.

Mobile Client – Android or IOS app which will be available on user mobile to interact with ticketing system.

Admin Frontend – Frontend hosted for Admins of ticketing platform to perform activities like Onboarding of new Movie or Theatre.

Theatre User Frontend – Frontend made available to theatre admins to perform daily activities related to managing of shows running in their respective theatres.

Ticket Service Gateway

This component will serve multiple roles stated below

- Load balancing
- Decouple clients from backend services.
- Routing the request to appropriate backend service
- Rate limiting
- Security – Authorization/Authentication
- Hide the complexity of the system from the outside world.

Some examples of popular API gateways: AWS API gateway, Kong API Gateway, APIGEE

CDN

Content delivery network will be used to cache static content like Movie information (description, ratings, thumbnails etc), cache show details for popular cities and popular movies.
CDN will be hosted close to user location and application will have very low latency.
During peak times, CDN can be used to improve application scalability and performance.

Popular CDNs: Akamai, AWS Cloudfront

Database

RDBMS – Relational database will be used to store all data for movies, theatres, cities, shows, seats, users, bookings etc. Relational database will help to maintain ACID properties of transactions as payments and booking is involved. Detailed database design is discussed in previous section.
Database partitioning will be based on region and replication will be done in multiple availability zones.

Popular Relational Database: MySQL, AWS RDS, Postgresql, Oracle etc.

Redis cache

Redis cache will be used to cache all the important and frequently used transactions, like show details for theatres and cities. Different backend services will update redis cache whenever there is update on frequently accessed data.

Cache eviction policy will be based on show dates. Shows past the current date will be evicted and shows for next 7 days will be only cached. Eviction policy can be created for movies and theatres also.
Redis cluster will be deployed in multiple availability zones to support fault tolerance and high availability

Elastic Search

Searchable data like movie and theatres information can be put in elastic search for faster search results. This will be called by separate search service for searching capabilities. Search results can be prebuilt and saved in elastic search for faster search results.

Backend Services

All backend services will act as separate microservice based applications.

1.) Theater Service – This service will host various endpoints to allow create/modify/delete theatres. It will store data in RDBMS database in theatre table.
2.) Movie Service - This service will host various endpoints to allow create/modify/delete movies. It will store data in RDBMS database in Movie table.
3.) Search Service – This service will host endpoints to search movies, shows running in cities for a particular movie etc.
4.) Show service – This service will host endpoints to manage shows. Shows for selected movie and theatre can be created/modifies/deleted using show service. This service will also interact with integration service to fetch show details from other theatres. Booking service will also interact with show service to fetch latest show availability.
5.) Booking Service – This service will host endpoints to perform ticket booking. User will be able to select seats and book available seats using booking service. Booking information will be persisted in RDBMS. On confirmation of booking, payment service will be called to initiate booking.
6.) Payment Service – This service will interact with third party payment merchant to perform payments using user selected payment method.
7.) Sync Service – This service will poll data from theatres to sync shows and movie details of the theaters not using ticket platform to manage shows.
8.) Integration service – This service will call endpoints of theatres to fetch latest show and movies from theatres not using ticket platform to manage shows.

All microservices to be containerized and Kubernetes to be used for container orchestration.

Notification Queue

Async queue system to be used to send notifications via SMS/Emails to the users on successful booking. AWS SNS can be used to send notifications.

Log Aggregation Queue

Logs from various microservices can be aggregated using ELK stack. Logs will be dumped into Elastic search using Kafka topics and can be visualized using Kibana

Observability/Monitoring

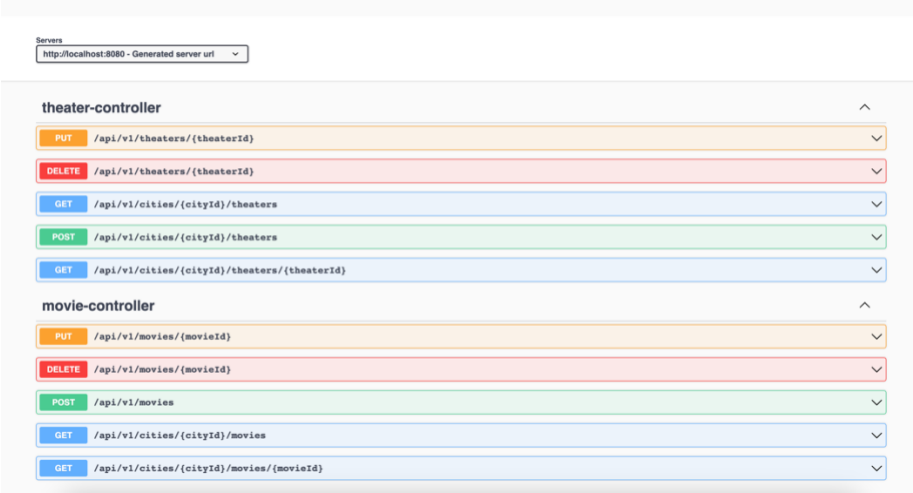
Monitoring and observability can be implemented using CloudWatch, Datadog, Grafana etc to monitor the health of services and collect relevant metrics to analyse state of the system.

Hadoop Cluster

Various types of structured and unstructured data related to user activity, bookings, cancellations, and other relevant information can be posted via Kafka queue to the HDFS cluster. The data engineering team has access to their own infrastructure and can leverage it to generate valuable insights that aid in making informed business decisions. Moreover, modern AI/ML technologies can be utilised for personalisation, recommendations, application analytics, and other purposes.

API endpoints Demo

API details are exposed using Swagger/Open API UI : http://localhost:8080/swagger-ui/index.html#/



Important Flows

Ticket Booking flow

1. User selects city on home page.
2. User is displayed with movies currently running shows in city.
3. User searches a movie.
4. User selects a movie for booking.
5. User clicks on book tickets button on movie details page.
6. User selects language and format for which he wants to book shows.
7. User is displayed with list of theaters and shows running in each theater. Shows running after current date are displayed by default. User has option to update dates, language, and format on show details page.
8. If show is already booked, it will be disabled for the user to select.
9. User selects the show to book and selects number of seats to book. Max 10 tickets are allowed to book in single booking.
10. If show has seats available equal to seats selected by user, seat map is displayed to user.
11. Seat map is created based on the seat class ordering and number of seats available in each class.
12. Price against each seat class is displayed on seat map.
13. If seat is already booked, its grayed out and unavailable to user to book.
14. Once user selects the seats to book and clicks proceed, following scenarios can occur.
    - If seats selected are no longer available to book, user is redirected back to seat booking map.
    - If seats are available to book, user is directed to booking summary page and seats are locked for 10 mins. User can then proceed to make payment and book the tickets. If user

is not able to make payment within 10 minutes, seats reserved will be unlocked and will be available for booking.

Concurrent Seat Booking Scenario

If multiple users are trying to book same seat then we will use a locking mechanism to lock the available seats for 10 mins before user can make payment. This is based on first come first serve basis and whichever user proceeds to confirm seat booking will get a chance to make payment and book seats.

1. First user U1 selects a preferred seat on seat map and proceeds to book.
2. Seat_Id, Show_Id, Status (LOCKED) and Updated_DateTime is updated in Locked_Seats table.
3. Locked_Seat table have composite key on Seat_Id, Show_Id and Status which maintains ACID properties and same seat for same show cannot be locked twice by different users.
4. Second user U2 visits the seat map for the same show, the seats selected by U1 will be unavailable for U2 as they are already present in LOCKED status in Locked_Seats table. Seat map is generated based on seats status in booked_seats and locked_seats table.
5. If U1 is not able to complete the payment within 10 minutes, locked seats will be released from Locked_Seats table. Below approach can be used to release the lock on seats
    a. A thread will keep polling the locked_seats table, and if seat_id and show_id is locked for more than 10 minutes, it will update the status as RELEASED. Once released seats will be available to other users to book, and U1 will see timeout message when he confirms payment.
    b. Instead of Updated_DateTime in Locked_Seats, we can use column lock_until column. Value of lock_until column will be 10 mins ahead of current time. During booking of seats, Locked_Seats table can be checked for lock_until column. If seats are already locked, same seats cannot be booked.
    c. We can use Redis or Mongo DB

Shows Create/Modify Scenario

Whenever theatre admins try to add shows for their theatres, following validations will be placed in system to avoid creation of conflicting shows.

1. Theater admin can only add shows for current date + 7 days.
2. Theater admin can only add shows for their theatre.
3. Theater admin can add multiple shows for a movie in same request.
4. Theater admin cannot add a show on same screen before the showEndTime of previous show in same screen.
5. ShowEndTime should not be less than total movie duration.

Onboarding of Theatres with own IT system

For theatres not using the booking platform software to manage shows, will have their internal IT system system to manage ticket booking. In order to onboard such theatres, Ticket platform has to sync the shows and movie details with these theatres. For this we need to define some mapping mechanism. So during onboarding on any theatre we can gather all the information from them regarding their constructs and create one time mapping. And later we can keep polling them from our syncing service and sync data in ticketing platform.

1. A syncing service will be running in ticket booking platform which will interact with integration service to fetch show details of theatres.
2. Integration service will have configuration to call APIs of theatres to fetch movie or show details.

3. Sync service will then sync the movies created in ticket platform with movies from theatres based on predefined mappings. Also sync service will sync seat, shows and other booking details with theaters.
4. Show service will call integration service to fetch current show information before booking.

Other Considerations from Future perspective

Monetization

Ticketing platform can get commission from theatres for each new booking. Ticketing platform can also charge other service charges, ease of use charges, and cancellation charges for monetization.

Integration with payment gateways

Integration with third party gateway solutions like Razor pay etc can be leveraged to provide users with multiple options to make payments online.

Scale to multiple cities, countries

In order to scale to different cities, ticketing platform has to collect one time information about the city region etc and also put the various mappings in system to identify the cities uniquely.
For scaling to multiple countries, country specific code changes needs to be in place to cater to the needs to different countries and comply with their policies. Also, infrastructure also needs to be placed close to regions of different countries being onboarded.

Logging and Monitoring

ELK stack to be leveraged for logging. Kibana dashboards can be configured to visualize various error scenarios and monitor them at scale. Slack alerts can be configured based on error scenarios.
Dashboards can be built to see applications in error and support teams can get notifications for errors in applications.

Continuous monitoring should be implemented using Grafana and Influx DB. Necessary alerts and notifications can be configured to alert the teams on error situations and can be acted upon quickly.

KPIs

System to collect multiple KPIs related to user usage behaviour like total number of booked tickets, number of visitors, number of users not making payments.
From system perspective KPIs like average response time, system uptime, system downtime, peak usage times, CPU and memory utilizations etc.

Localization

Region specific language can be stored in movie metadata. Search results in Elastic search will be based on region specific data. Mapping of region and language can be maintained, and region specific caching should be updated based on local language. User home page should also be displayed based on region language as user in certain region will book for movies in certain language only.

Github Repo URL

https://github.com/anshulisinghal/TicketBookingSystem

Postman collection

https://github.com/anshulisinghal/TicketBookingSystem/blob/main/ticket-postman-collectiion.json