Problem statement

XYZ wants to build an online movie ticket booking platform that caters to both B2B (theater partners) and B2C (end customers) clients.

Key goals it wants accomplished as part of its solution:

- Enable theater partners to onboard their theaters 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.

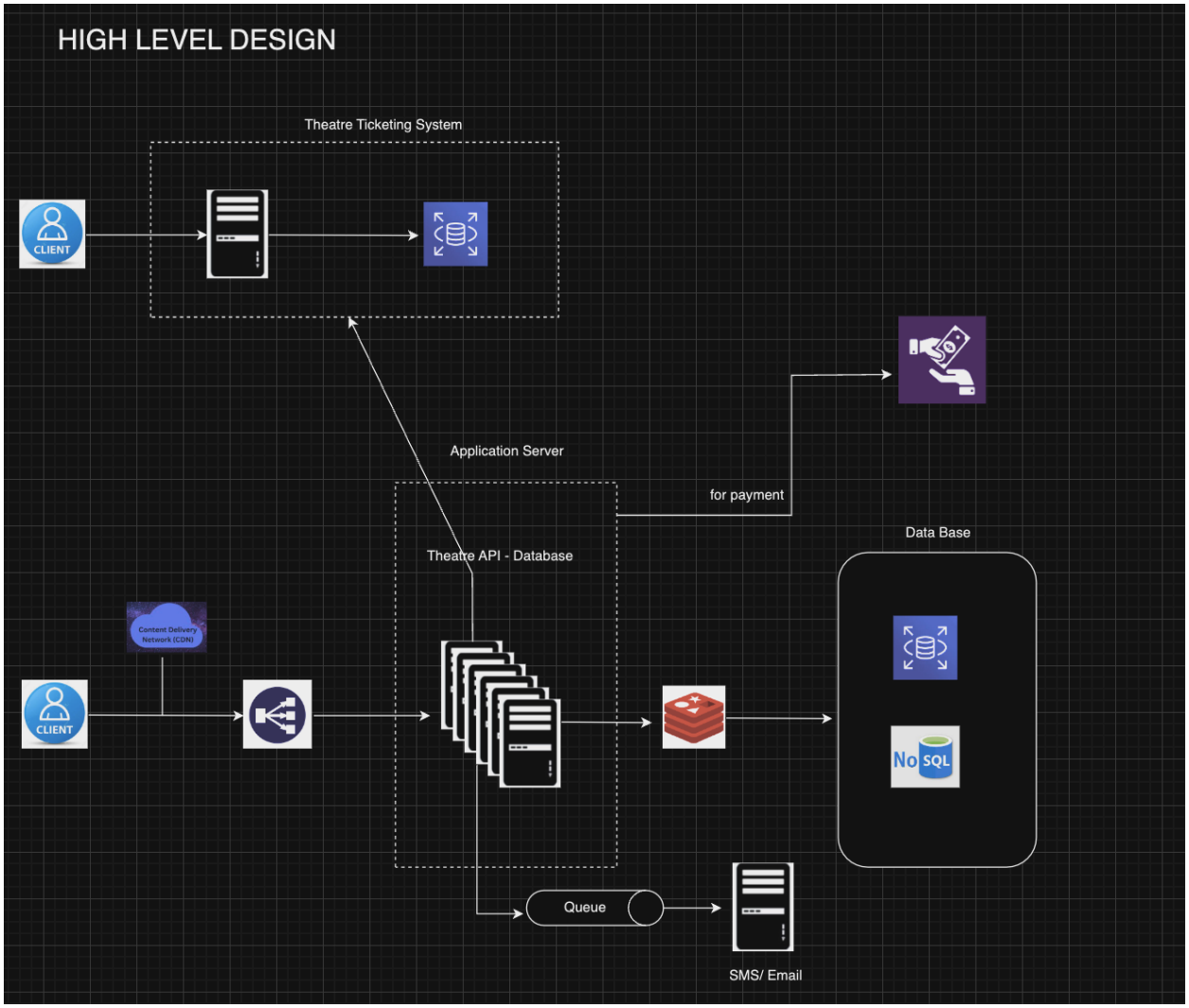
Some Design Considerations

- Though Payment is the integral part of any show booking, our solution has not scoped it
- Our service doesn't require user authentication
- No partial orders
- Data examples consider each hall to have max 6 seats
- The system should be scalable, highly available to cope up with the surge in traffic

Technology Used

- Language - Java
- Frameworks - Spring Boot
- DataBase - MySql
- Distributed Caching - Redis (for caching informations like theaters, cities, movies etc)
- Preferred Editor - IntelliJ
- Cloud Technologies - Load Balancer, CDN

High Level Design



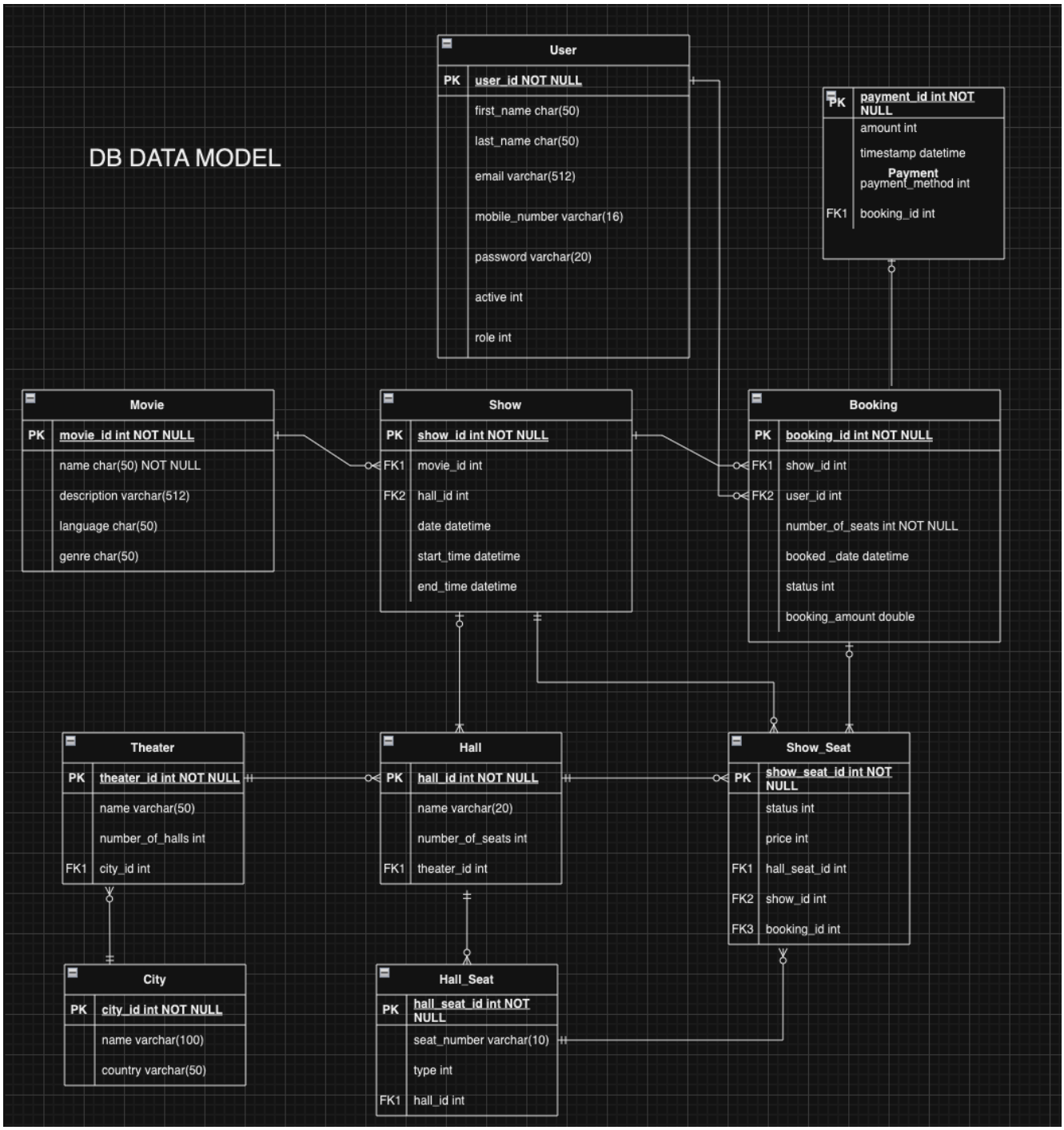
Platform Solution Details

Platform will comprise on below mentioned components

1. Load Balancer -> To distribute load different instances of the application server. To start with layer 4 load balancer, unless we have specific balancing requirements like stickiness etc, which needs layer 7 load balancing. Load balancer algorithm can be defaulted to “Round robin”
2. Orchestration Service -> to host multiple instances of the application service with auto scaling feature to scale up and down based on traffic. Eg AWS Elastic beanstalk, Elastic Container Service
3. Distributed Cache -> to store the data for which the state doesn't change frequently. Eg Movie, Theater, Hall, Seats in the hall etc. This will reduce load on the database
4. Database -> to store the data of the system and maintain transactions. It should be highly available using master slave architecture. To scale we can add shards.

5. NoSql database(optional) -> in case we need to store non schematic data like reviews, comments etc. This will help us build a recommendation system later
6. Queue -> All the notifications to users about the booking can run in async mode. So messages will be send to a queue, from which the consumer will take care of notification
7. External Theater System -> All the external theater systems will communicate through our api's.
8. Content Delivery Network -> to store the thumbnails, html or other relevant static data

DB DATA MODEL



Database design Consideration

1. Each City can have multiple Theaters.
2. Each Theater will have multiple Halls.
3. Each Hall will have multiple seats.
4. Each Movie will have multiple Shows, and each Show will have multiple Bookings.
5. Each Show Seat in a Hall will be one to one mapped to Hall Seat
6. A user can have multiple Bookings.

Functional features to implement Good to have - Code Implementation (Read scenario)):

Browse theaters currently running the show (movie selected) in the town, including show timing by a chosen date (Implemented)

`/movie/city/{city}/date/{showDate}/name/{movieName}`

curl -X GET "http://localhost:8090/movie/city/Bangalore/date/2023-10-23/name/Jawan" -H "accept: application/json"

Response


```
[
 {
  "showId": 1,
  "showDate": "2023-10-13",
  "startTime": "2023-10-13T16:15:59.42",
  "endTime": "2023-10-13T19:15:59.42",
  "movie": {
   "movieId": 1,
   "movieName": "Jawan",
   "movieLanguage": "Hindi",
   "movieGenre": "Action",
   "description": "string"
  },
  "hall": {
   "hallId": 1,
   "name": "gold-east",
   "totalSeats": 6,
   "theater": {
    "theaterId": 1,
    "name": "Cental-PVR",
    "numberOfHalls": 4,
    "city": {
     "cityId": 1,
     "name": "Bangalore",
```

```
        "country": "India"
      }
    }
  }
},
{
  "showId": 3,
  "showDate": "2023-10-13",
  "startTime": "2023-10-13T16:01:06.572",
  "endTime": "2023-10-13T18:01:06.572",
  "movie": {
    "movieId": 1,
    "movieName": "Jawan",
    "movieLanguage": "Hindi",
    "movieGenre": "Action",
    "description": "string"
  },
  "hall": {
    "hallId": 2,
    "name": "gold-west",
    "totalSeats": 6,
    "theater": {
      "theaterId": 1,
      "name": "Cental-PVR",
      "numberOfHalls": 4,
      "city": {
        "cityId": 1,
        "name": "Bangalore",
        "country": "India"
      }
    }
  }
},
{
  "showId": 4,
  "showDate": "2023-10-13",
  "startTime": "2023-10-13T14:01:06.572",
  "endTime": "2023-10-13T16:01:06.572",
  "movie": {
    "movieId": 1,
    "movieName": "Jawan",
    "movieLanguage": "Hindi",
    "movieGenre": "Action",
    "description": "string"
  },
  "hall": {
    "hallId": 1,
    "name": "gold-east",
    "totalSeats": 6,
```

```
        "theater": {
          "theaterId": 1,
          "name": "Cental-PVR",
          "numberOfHalls": 4,
          "city": {
            "cityId": 1,
            "name": "Bangalore",
            "country": "India"
          }
        }
      }
    }
  ]
```


Approach

1. With City name provided, find the City entity, get the city id
2. With Movie name provided find the Movie entity, get the movie id
3. With city id, movie id and date; join Show, Theater and Hall with respective foreign keys; and filter the data to get list of Show elements to return

In MovieService.java

```
public List<Show> getAllShowByCityByNameAndDate(String cityName, String
showDate, String movieName){
   City city  = cityRepository.findByName(cityName).orElse(null);
   Movie movie =  movieRepository.findByMovieName(movieName);
   List<Show> showList = new ArrayList<>();
   if(city == null) return showList;
   return
showRepository.findAllShowByDateAndByMovieNameAndByCity(city.getCityId(),showD
ate,movie.getMovieId());
}
```

In ShowRepository.java

```
@Query(value = "select s.* from show_table s inner join hall h inner join
theater t " +
       "on s.hall_id = h.hall_id and t.theater_id = h.theater_id where
s.show_date = ?2 and s.movie_id = ?3 and t.city_id = ?1" , nativeQuery = true)
List<Show> findAllShowByDateAndByMovieNameAndByCity(int cityId, String
showDate, int movieId);
```

Booking platform offers in selected cities and theaters

- 50% discount on the third ticket
- Tickets booked for the afternoon show get a 20% discount

Book movie tickets by selecting a theater, timing, and preferred seats for the day (Implemented)

Approach

1. From the UI user will hit the below mentioned api to get List of shows for the selected movie, running for the selected date and where seats are available

Call **GET**
/movie/city/{city}/date/{showDate}/name/{movieName}/available

http://localhost:8090/movie/city/Bangalore/date/2023-10-13/name/Jawan/available


```
[
  {
    "seats": "gold-3,gold-4,gold-5,gold-6",
    "showId": 1,
    "showDate": "2023-10-13",
    "theaterName": "Cental-PVR",
    "hallName": "gold-east",
    "seatCount": 4,
    "showSeats": "3,4,5,6"
  },
  {
    "seats": "gold-1,gold-2,gold-3,gold-4,gold-5,gold-6",
    "showId": 3,
    "showDate": "2023-10-13",
    "theaterName": "Tulsi",
    "hallName": "gold-west",
    "seatCount": 6,
    "showSeats": "13,14,15,16,17,18"
  },
  {
    "seats": "gold-1,gold-2,gold-3,gold-4,gold-5,gold-6",
    "showId": 4,
    "showDate": "2023-10-13",
    "theaterName": "Cental-PVR",
    "hallName": "gold-east",
    "seatCount": 6,
    "showSeats": "19,20,21,22,23,24"
  }
]
```


2. From the information displayed on the UI, he will select the show and the respective seat numbers and below mentioned api will be called.

POST
/bookTicket/new

Payload

```
{
  "showId": 4,
  "showSeatIdList": [
    19,
    20,
    21
  ],
  "userId": 1
}
```


Theaters can create, update, and delete shows for the day.

CRUD api's provided for theater, hall, hall seat, show, show seat

Bulk booking (2 endpoints provided for booking)

1. First endpoint, you can mention the exact seat number

   POST /bookTicket/new

   Payload
   {
   "showId": 1,
   "showSeatIdList": [1, 2, 3],"userId": 1 }

2. Second endpoint, you just mention the number of seats (so any available seat will be booked, no preference taken)

   POST /bookTicket/new/user/{userId}/show/{showId}/count/{countOfSeats}

   Example
   /bookTicket/new/user/1/show/2/count/3

Cancellation of Booking

POST /bookTicket/cancel/{Id}

Theaters can allocate seat inventory and update them for the show

CRUD api' provided for show seat

Non-functional requirements-(Mandatory -Design/Arch solution & Optional Implementation):

- Describe transactional scenarios and design decisions to address the same.

  The typical problem in such a platform is that two users might be trying to book the same seats for the same show. We can use transactions in SQL databases to avoid any clashes.

  While the booking api call after the seat numbers are known, going for booking the seats, the seats that are selected for the show, in our case, from the "show_seat" table, need to be locked, and post successful booking, status needs to be updated as NOT_AVAILABLE(1). On unsuccessful booking, lock will be released after 5 mins.

  Have implemented using SERIALIZABLE isolation.

  SERIALIZABLE is the highest level of isolation. It prevents all mentioned concurrency side effects, but can lead to the lowest concurrent access rate because it executes concurrent calls sequentially. Within a transaction if we read rows we get a write lock on them so that they can't be updated by anyone else.

```
1 usage
public Booking commitBookingWithSeatList(int bookingId, List<Seat> seatList){
    List<Integer> seats = seatList.stream().map(Seat::getShowSeatId).collect(Collectors.toList());
    return commitBooking(bookingId, seats);
}

2 usages
@Transactional(isolation = Isolation.SERIALIZABLE)
public Booking commitBooking(int bookingId, List<Integer> seatList){
    double bookingAmount=0;
    for(int seat : seatList){
        bookingAmount += Objects.requireNonNull(showSeatRepository.findById(seat).orElse( other: null)).getPrice();
        showSeatRepository.updateBookings(bookingId, seat);
    }
    bookingRepository.updateBookingStatusAndAmount(bookingId,bookingAmount);
    return bookingRepository.findById(bookingId).orElse( other: null);
}
```

- Integrate with theaters having existing IT system and new theaters and localization(movies)

  Integration will happen though api's, their primary database for seats at their and our show_seats needs to be in sync. So while booking both the relevant rows in the seats table need to be locked. Saga design patterns for microservices can be used.

- How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?

1. Databases can be scaled using sharding based on fn(city, country) deployed in multiple AZ
   Regular backup and Snapshot. AWS aurora db is a good choice.
   Implement read replicas for reading.
   AWS Elasticache with read replicas enabled for redis.
2. Services will be deployed in multiple availability zones under a sufficiently configured auto-scaling group.
   Use serverless SQS for queue.

- How do you monetize the platform?
  Provision for ads