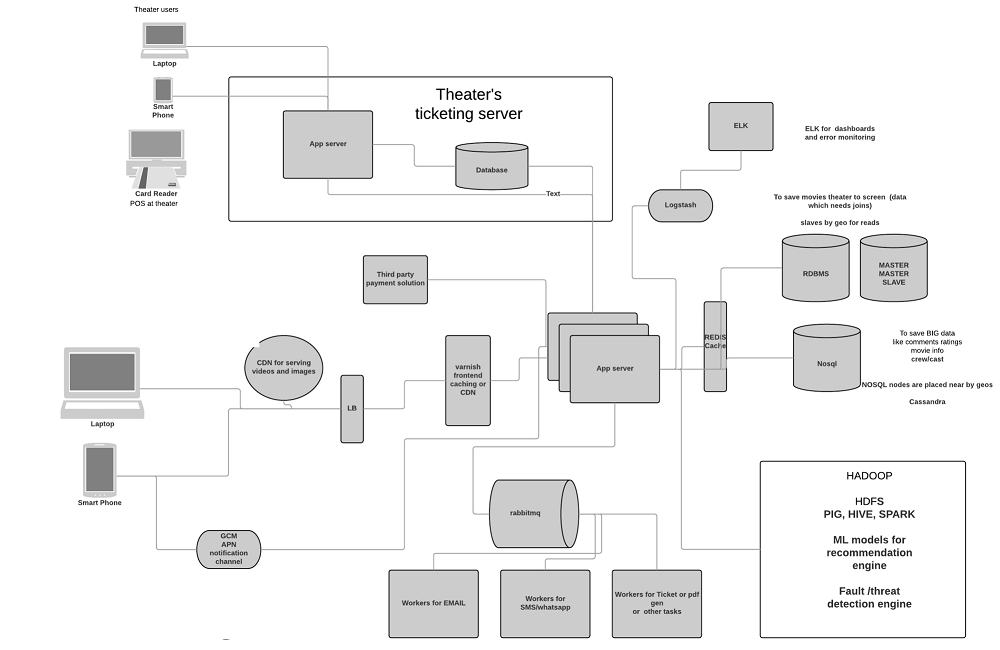
**Requirement Points**

* The portal should list down the different cities where the theatres are located. (RDBMS)
* Once the user selects the city it should display the movies released in that particular city to that user.
* Once the user selects the movie, the portal should display the cinemas running that movie and the available shows.
* Users should be able to select the show at a particular theatre & book the tickets (third-party payment support for ex razorPay/POS etc).
* Send a copy of tickets via SMS notification or Email.
* The portal should display the seating arrangement of the cinema hall to the user.
* Users should be able to select multiple seats according to their choice.
* Users should be able to hold the seats for 5-10 minutes before he/she finalized the payment.
* The portal should serve the tickets in a First In First Out manner
* Comments and rating
* The system should be highly concurrent because there will be multiple booking requests for the same seat at the same time.
* The core thing of the portal is ticket bookings which mean financial transactions. So, the system should be secure and ACID compliant.
* Responsive design (ReactJS and Bootstrap) to run on devices of various sizes like mobile, tablet, desktop, etc.
* Meta data of Movie information.
* Locking system for a seat should be enabled on the system which is to be shared with app as well as theatre so two different users were not able to visit and select/book the same seats.

**High Level Architecture**



**Following infra will be needed to support and let the system works with high availability:**

1. Load Balancer
2. App servers/Docker PODS
3. Caches
4. Database (SQL, NOSQL)
5. Log management (ELK/Cloud Watch)
6. Storage account (images/Videos)
7. AWS media service
8. Cloud Front/CDN
9. Route 53

* Load balancer Url will be used by Front end to communicate with the BE apis and behind the scene of LB app servers/PODS can be auto scale.
* App Servers/Docker PODS will be used to deploy the BE microservices application.
* Redis cache can be used to store the common static data to make the performance of the application much faster
* SQL database will be used wherever we want/have to maintain the relation between data, while if we don’t need relation and just on the basis of id or some other conent we can put the data we go ahead and use No Sql.
* Logging of the services can be done on the Cloud watch or else we can create own CLK stack for logging.
* Storage accounts – S3 buckets will be used to save/get the audio/video or images.
* AWS media service can be used to stream the video on runtime faster.
* Cloud Front/CDN is for caching the content on the edge locations.
* Jmeter can be used to measure the performance of the APIs for bulk load to improve and optimize the response time.

**APIs**

Following APIs can be created to achieve the required behavior

* Get list of cities
* Get list of events in the City
* Get event detail by city and location detail
* Get show timing by event and location
* Get available seats in a theatre on basis of event, location and show time
* Get the shows available on the theatre by location and theatre detail
* Verify the users selected seat is not blocked by anyone via event, location, time and seats selected
* Block the seats selected by user so no one else can try to book those for specific time like 5-10 mins
* Book the user selected seats and verify the payment
* Send timeout error if user not able to book the ticket in a specified time.
* Unblock the blocked seats if not booked or after specified time.
* Add/update API for theatre to add/update show/movie detail.
* API for theatre to add/update seat detail if booked by theatre directly.
* API to add new theatre or remove theatre from the system.

**RDBMS Tables**

* Location
* Theatres
* Screens
* Tiers (tier of seats)
* Seats
* Movies
* Offers
* Tickets
* Users

**Relationship Between RDBMS Tables:**

* One to many: Place and theatre.
* One to many: Theatre and screen
* One to many: Screen and Tier
* One to many: Tier and seats
* One to one: Screen and Movie
* One to many: User and Tickets
* One to many: Tickets and Seats

**NoSQL Tables**

There will be no relationship between these tables.

* Comments
* Ratings
* Movie Information
* Trailers or Gallery
* Artists
* Cast and Crew
* Reviews
* Analytics Data

**Technologies**

* User Interface: ReactJS/React native & BootStrapJS
* Server language and Framework: Java, Spring Boot, Hibernate
* Security: Spring Security
* Database: MySQL/NOSQL
* Server: Tomcat
* Caching: Redis Cache
* Notifications: SNS/SQS/Google firebase. A Distributed message queue for push notifications.
* Payment API: Paypal, Razorpay, POS machine etc
* Deployment: App servers/ AWS EKS
* Code repository: Git/AWS code commit
* Logging: Log4J
* Log Management: Cloud watch
* Load balancer: Application load balancer
* Unit Testing: Junit/Mockito
* We can used ASG on app servers or if we use EKS we can configure ASG there to create PODS for scaling horizontally

**OWASP Vulnerability**

* Take Zero trust approach – Zero-Trust approach must be adopted whether it is users, employees, vendors, or third-party service providers. This helps in protecting against a majority of OWASP Top 10 vulnerabilities including brute force attacks, XSS attacks, injections etc
* Should use Web Application firewall (WAF) to prevent vulnerabilities being exploited.
* Should have strong password Policy and Multi factor authentication (MFA)
* Always encrypt all the sensitive data before sending it to FE
* Validate all the input before using it. Validation ensure that all the data came into the system is not malformed.
* Effective logging and monitoring of the application should be implemented.
* Should regularly do the audit, Penetration testing etc to find any loopholes or vulnerability
* Should follow the best security coding practices like encoding, validation, principle of least privilege etc
* Should always use the component or software required to be from the reliable and trusted sources only.

**monetize platform**

We can run ads on our website related to monetize the platform. Google Ads can be used for the same.

We can take commission from the theatre for providing and maintaining data and ticket booking.

We can take the service charge/Internet handling charges etc from the user for the service.