MTPB - Movie Ticket Booking Platform

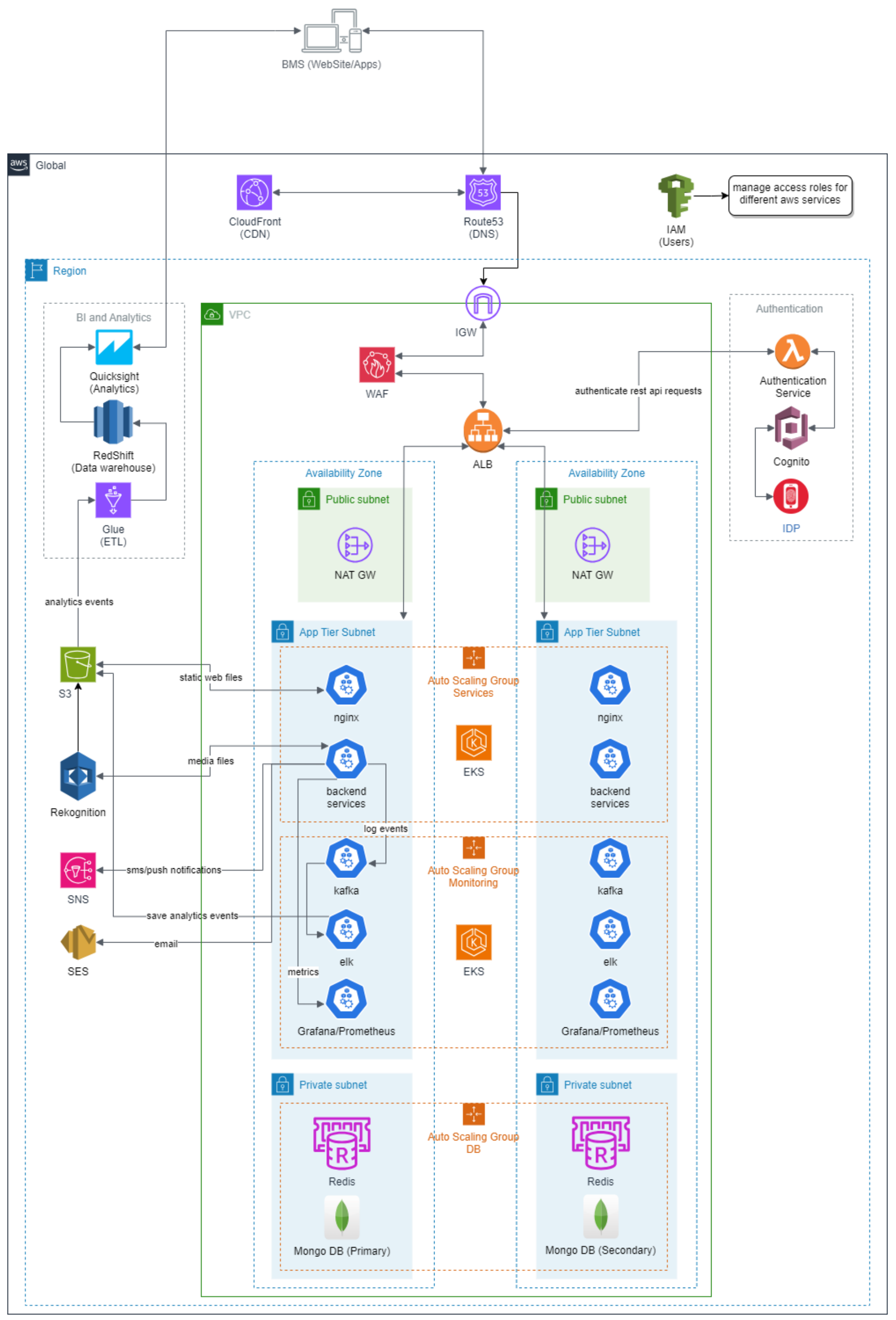
-- Backend System Design and Project Plan Overview

Overview

This document talks about the backend system design of a movie ticket booking platform (MTBP) and discusses in brief on below mentioned points

- ❑ High level architecture of the platform
- ❑ Discuss on the tools and services required to bring up the platform
- ❑ Discuss the design points and challenges of the system
- ❑ Discuss about the functional requirement to be implemented
- ❑ List out the REST APIs required
- ❑ High Level view of the DB Model
- ❑ Discuss the points on how to monetize the platform
- ❑ Discuss the best practices and nonfunctional requirements
- ❑ High Level Project Plan

Architecture Diagram



Tech Stack - Overview

- **Networking**
  - AWS Route53 (DNS)
  - AWS WAF (Firewall)
  - AWS VPC (Network Groups)
  - ALB (Load Balancer, TLS termination)
- **Backend**
  - Spring Boot (Micro-services)
  - AWS Lambda (Authentication)
- **Message Queue**
  - Kafka
- **Database**
  - MongoDB (NoSQL)
  - AWS S3 (Object Store)
  - AWS Cognito (Authentication)
  - AWS RedShift (Data warehouse)
- **Cache**
  - AWS CloudFront (Media)
  - Redis (DB)
- **Notifications**
  - Simple Notification Service (SNS)
  - Simple Email Service (SES)
- **Monitoring**
  - Elasticsearch
  - Logstash (Kafka Connector)
  - Kibana (Logs and Events)
  - Prometheues (Metrics)
  - Grafana (Dash boards)
- **Deployment**
  - Maven (Build Tool)
  - Jenkins (CI/CD)
  - Docker (Containers)
  - AWS EKS (Kubernetes)
- **Testing**
  - JUnit (Unit Testing)
  - Mockito (Mocking Framework)
  - REST-Assured (REST API Testing)
  - Postman (REST API Testing)
  - Gatling (Performance Testing)
- **Analytics**
  - AWS Glue (ETL)
  - AWS Quicksight (Reports and Dashboards)

Tech Stack - Details

- **AWS (Networking)**
  - AWS Route53 service can be used as DNS server for the platform.
  - Use the Region, AZ model of AWS to distribute workload and promise high availability of the platform services.
  - Use public and private subnetworks provided by AWS VPC to deploy applications in a secure way.
  - Use AWS WAF to mitigate major security threats like XSS or DDOS attacks.
  - AWS ALB can be used for load balancing, api authentication, TLS termination etc.
- **Spring Boot (Backend)**
  - Easy to develop, highly customizable and extensible services using Java.
  - Huge community and documentation to aid development.
  - Abstractions available for all major use cases like rest apis, db management etc.
  - Good amount of testing frameworks available.
  - Follows best practices in development e.g., S.O.L.I.D and many other design patterns.
  - Good support for non-functional aspects like logging, security etc.
- **AWS (API Authentication)**
  - Use AWS Cognito to manage user authentication.
  - AWS Lambda helps to connect ALB with Cognito to verify token.
  - IDP provider is used to generate Oauth2.0 token for authentication.

Tech Stack - Details

- **Kafka (Message Queue)**
  - Event driven communication between backend services.
  - Log application events for debugging.
  - Log analytics events for BI and analytics.
- **MongoDB (NoSQL DB)**
  - Highly scalable and reliable document-based NoSQL DB service.
  - Store partners and customer accounts data and movies meta data.
  - Integrate with Spring boot services easily using JPA
- **AWS Simple Storage Service (Object Store DB)**
  - Store media and files uploaded by users.
  - Store static web pages used by front end applications.
  - Store analytics events that can be accessed by to ETL service.
- **AWS (Cloud Hosting)**
  - Provide a secure and reliable infrastructure to deploy the platform services
  - Able to leverage aws managed services to help accelerate the project development.
  - Very good support and community available for reference and support.

Tech Stack - Details

- **AWS (Notifications)**
  - Use AWS SNS service to send SMS notifications and Push Notifications to the users.
  - User AWS SES service to send email notifications to users.
  - Users can be notified of – new movies, festivals offers etc.
  - Send booking confirmations and movie reminders as notifications
- **ELK (Logging and Debugging)**
  - Leverage the capabilities of ELK stack for a better log monitoring.
  - Kibana UI can be used for easy debugging of issues by making use of its filtering capabilities
  - Logstash is used as a kafka connector to forward events to Elasticsearch and S3 for further use by debugging and analytics tools.
- **Prometheus/Grafana (Metrics & KPIs)**
  - Collect system and application metrics (KPIs) with the help of Prometheus and Micrometer
  - Grafana is used as a dashboard to display metrics by connecting to Prometheus.
- **AWS (Analytics)**
  - Use AWS managed services for BI and analytics
  - Helps to avoid complex infrastructure management of a BI platform
  - Provides an easy-to-use ETL service in AWS Glue
  - AWS Quicksight can be used to create reports and dashboards and can be integrated with front end apps using REST apis.

AWS Services

Route 53
- ❑ Provides DNS functionality.

CloudFront
- ❑ Provides CDN caching for the platform
- ❑ Caching based on special occasions and popular movies

Web Application Firewall (WAF)
- ❑ Mitigates common security threats.

Application Load Balancer (ALB)
- ❑ Load balancing
- ❑ Circuit breaker
- ❑ TLS termination

Elastic Kubernetes Service (EKS)
- ❑ Manage backend services as containers in k8s
- ❑ Manage Kafka cluster as containers in k8s
- ❑ Manage ELK cluster as containers in k8s
- ❑ Manage Prometheus and Grafana as containers in k8s

Lambda
- ❑ Lambda service to authenticate rest api requests.

Cognito
- ❑ Store and validate user authentication information.

Simple Storage Service (S3)
- ❑ Stores static web files
- ❑ Media files storage (posters, trailers etc)

Redis (Elasticache)
- ❑ Cache DB queries with the help of Redis Cache

Rekognition
- ❑ Content filtering service
- ❑ Helps to filter media(posters, trailers etc) uploaded by users

Simple Notification Service (SNS)
- ❑ Send sms/push notifications to users.

Simple Email Service (SES)
- ❑ Send email notifications to user via SMTP

Glue (ETL)
- ❑ Logs created by services for analytics can be taken from s3 and ETL can be performed according to requirements.

Redshift (Data warehouse)
- ❑ Store analytics data for further processing by BI tools.

QuickSight (BI & Analytics)
- ❑ Perform BI on analytics data.
- ❑ Generate reports and dash boards for data visualization.

Backend Services

Movie Service
❑ Manages all available movies and its meta data
❑ Search for movies using filters (genre, language, country etc)

Theater Service
❑ Manages theaters, screens and shows related services.
❑ Search for theaters using filters (city, pin code etc)
❑ Search for shows using filters (city, date, movie etc)

Payment Service
❑ Handles integration with different payment gateways.

Review Service
❑ Deals with the review and comments given by customers on cinema and theatres.

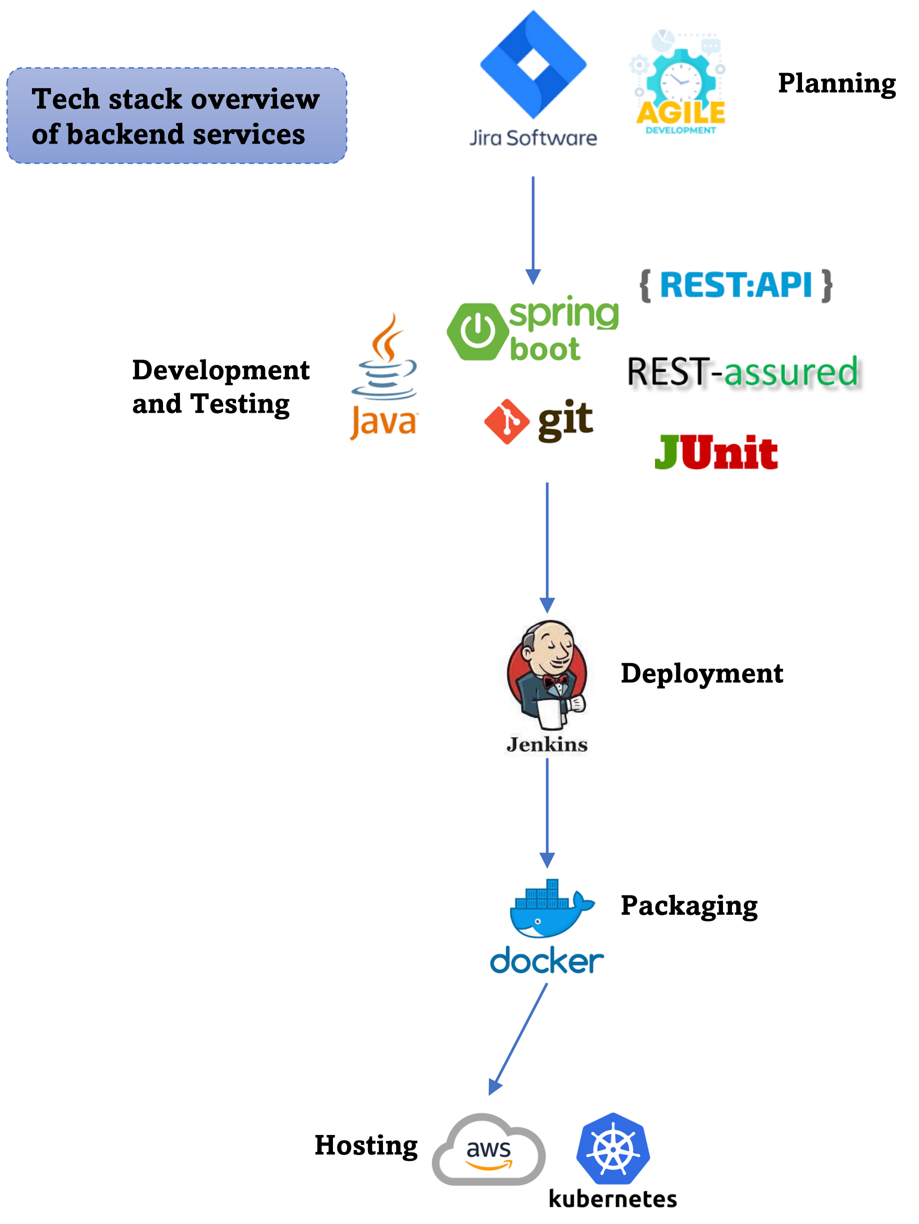
Media Service
❑ Stores media related to movies (posters, trailers etc) or other meta data.
❑ Connect to available CDNs for caching and better performance.

Ticketing Service
❑ Handles ticket reservation and booking without conflicts

User Service
❑ Provisions basic user information for partners, customers and brokers.

Offers Service
❑ Manages offers and discounts created by partners.
❑ Avails offers to customers based on their eligibility.

Ads Service
❑ Helps to show relevant ads for the end users
❑ Integrates with different ad providers



Open-Source Services

MongoDB
- ❑ Main database for the backend services.
- ❑ MongoDB is a distributed, highly scalable and reliable NoSQL db.
- ❑ Easy to integrate with spring boot-based applications
- ❑ Enterprise edition will be required for production and UAT, but community edition can be used for development purposes

Prometheus
- ❑ Collect application metrics from spring boot applications using micrometer apis.
- ❑ Collect system metrics using collectors.
- ❑ Integrate with Grafana to display the metrics in dashboards

Grafana
- ❑ Integrate with Prometheus and display the metrics dash boards.
- ❑ Generate alarms on configured KPI violations.
- ❑ Enterprise Grafana cloud is an alternative option to incorporate most of the observability requirements like logging, metrics etc.

Nginx
- ❑ Acts as a web server for the front-end applications
- ❑ Can connect to S3 and serve static web pages.

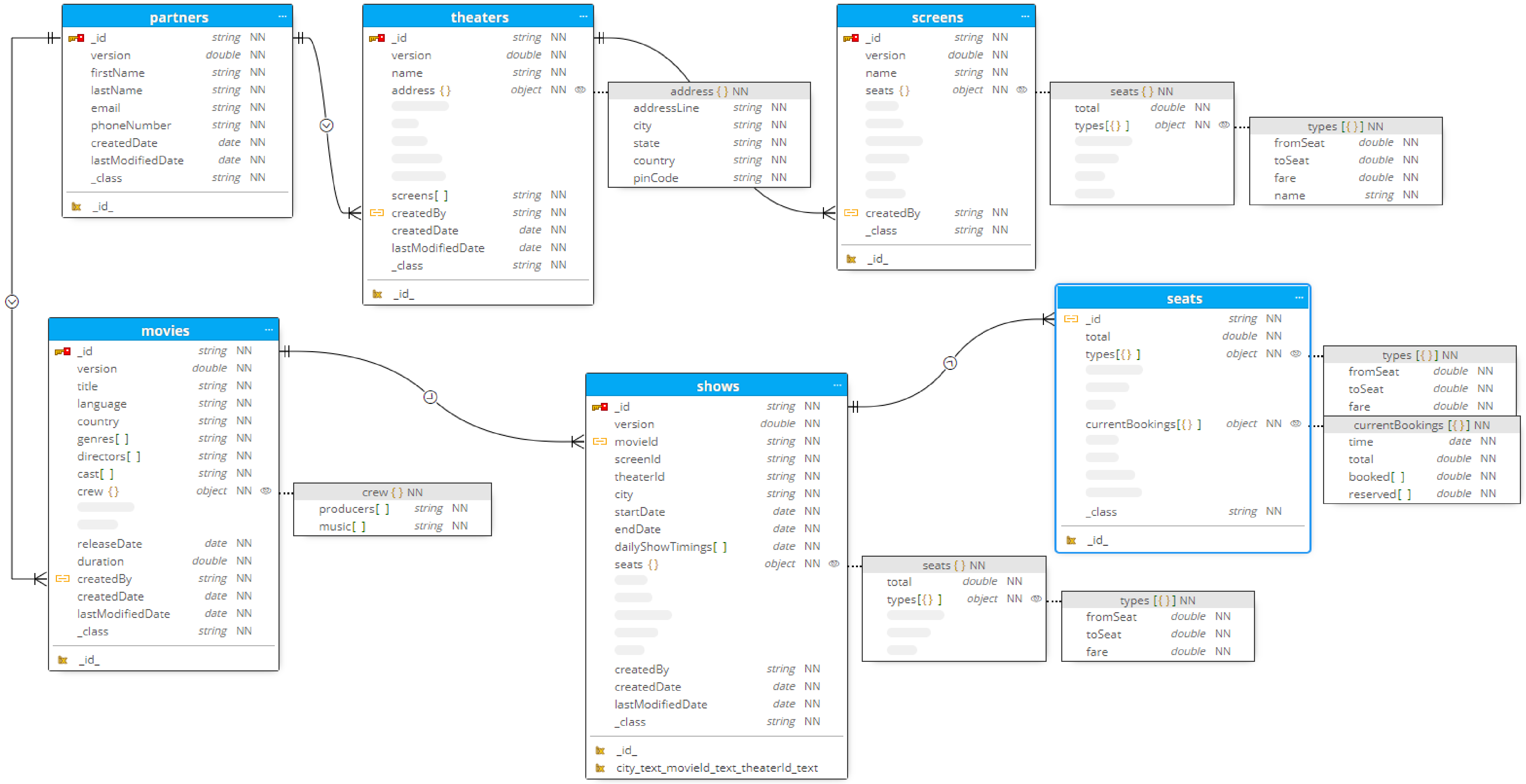
Kafka
- ❑ Kafka is used to queue messages that can be used for debugging and analysis
- ❑ Use logstash connector to send log events to Elasticsearch and later Kibana UI can be used to filter and debug with these log information.
- ❑ Send KPIs and analytics events like offers redeemed, total no of tickets booked for a show etc to logstash and store them in S3. AWS ETL services can be used to extract these data to perform business analytics.

Elasticsearch
- ❑ Log monitoring
- ❑ Store log events from backend services
- ❑ Create fast performing querys on log data for faster debugging.

Logstash
- ❑ Use to ingest, transform and load data to Elasticsearch and S3 from Kafka topics

Kibana
- ❑ Debug applications using logs pushed into elastic search
- ❑ Provides a UI to filter and search for log events

DB Model (Movies and Theatres)



partners		
_id	string	NN
version	double	NN
firstName	string	NN
lastName	string	NN
email	string	NN
phoneNumber	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

theaters		
_id	string	NN
version	double	NN
name	string	NN
address { }	object	NN
screens[ ]	string	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

address { } NN		
addressLine	string	NN
city	string	NN
state	string	NN
country	string	NN
pinCode	string	NN

screens		
_id	string	NN
version	double	NN
name	string	NN
seats { }	object	NN
createdBy	string	NN
_class	string	NN
_id_		

seats { } NN		
total	double	NN
types[{ }]	object	NN

types [{ }] NN		
fromSeat	double	NN
toSeat	double	NN
fare	double	NN
name	string	NN

movies		
_id	string	NN
version	double	NN
title	string	NN
language	string	NN
country	string	NN
genres[ ]	string	NN
directors[ ]	string	NN
cast[ ]	string	NN
crew { }	object	NN
releaseDate	date	NN
duration	double	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

crew { } NN		
producers[ ]	string	NN
music[ ]	string	NN

shows		
_id	string	NN
version	double	NN
movieId	string	NN
screenId	string	NN
theaterId	string	NN
city	string	NN
startDate	date	NN
endDate	date	NN
dailyShowTimings[ ]	date	NN
seats { }	object	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		
city_text_movieId_text_theaterId_text		

seats { } NN		
total	double	NN
types[{ }]	object	NN

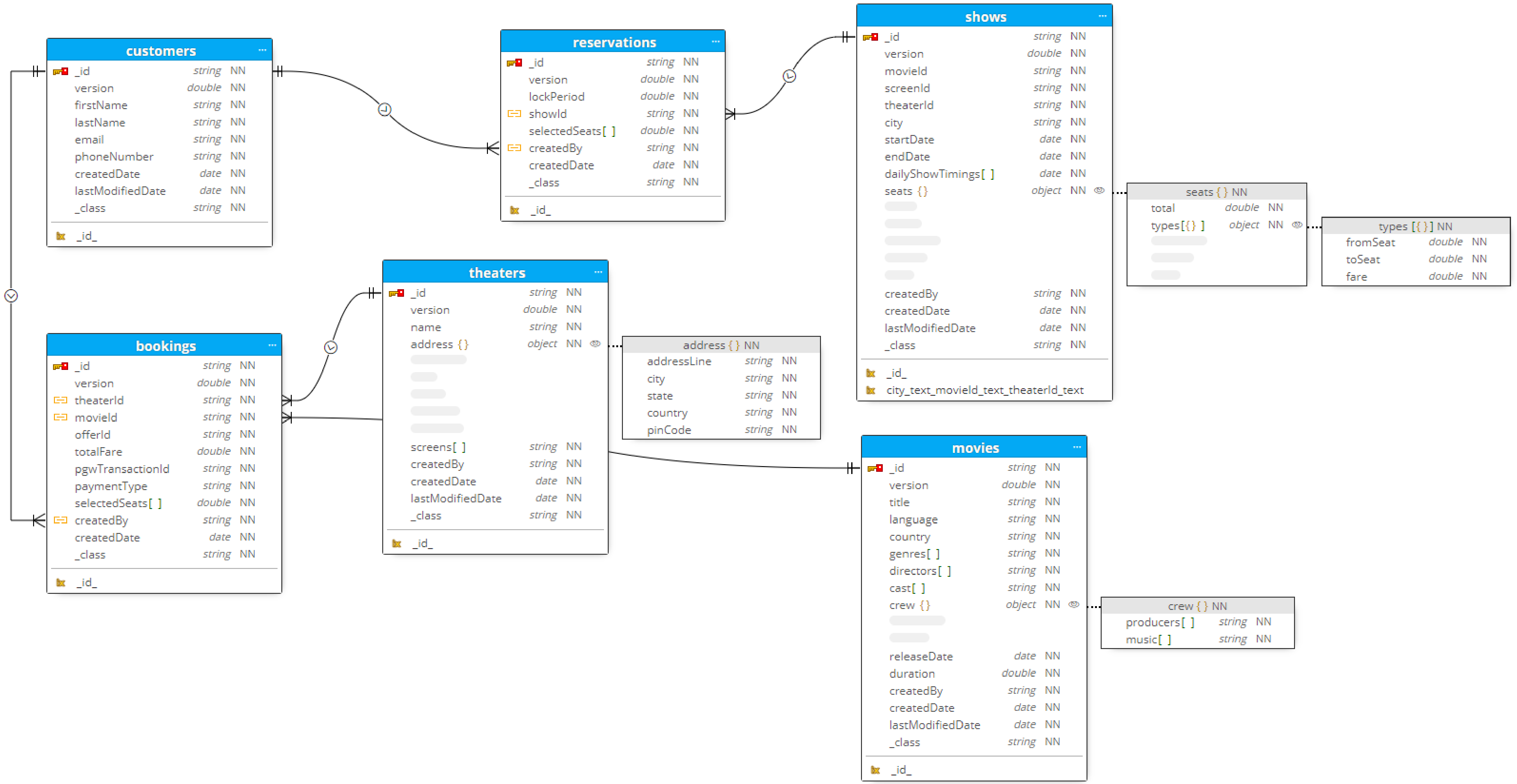
types [{ }] NN		
fromSeat	double	NN
toSeat	double	NN
fare	double	NN

seats		
_id	string	NN
total	double	NN
types[{ }]	object	NN
currentBookings[{ }]	object	NN
_class	string	NN
_id_		

types [{ }] NN		
fromSeat	double	NN
toSeat	double	NN
fare	double	NN

currentBookings [{ }] NN		
time	date	NN
total	double	NN
booked[ ]	double	NN
reserved[ ]	double	NN

DB Model (Booking and Reservations)



customers

_id	string	NN
version	double	NN
firstName	string	NN
lastName	string	NN
email	string	NN
phoneNumber	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

reservations

_id	string	NN
version	double	NN
lockPeriod	double	NN
showId	string	NN
selectedSeats[ ]	double	NN
createdBy	string	NN
createdDate	date	NN
_class	string	NN
_id_		

shows

_id	string	NN
version	double	NN
movieId	string	NN
screenId	string	NN
theaterId	string	NN
city	string	NN
startDate	date	NN
endDate	date	NN
dailyShowTimings[ ]	date	NN
seats { }	object	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		
city_text_movieId_text_theaterId_text		

seats { } NN

total	double	NN
types[{ }]	object	NN

types [{ }] NN

fromSeat	double	NN
toSeat	double	NN
fare	double	NN

theaters

_id	string	NN
version	double	NN
name	string	NN
address { }	object	NN
screens[ ]	string	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

address { } NN

addressLine	string	NN
city	string	NN
state	string	NN
country	string	NN
pinCode	string	NN

bookings

_id	string	NN
version	double	NN
theaterId	string	NN
movieId	string	NN
offerId	string	NN
totalFare	double	NN
pgwTransactionId	string	NN
paymentType	string	NN
selectedSeats[ ]	double	NN
createdBy	string	NN
createdDate	date	NN
_class	string	NN
_id_		

movies

_id	string	NN
version	double	NN
title	string	NN
language	string	NN
country	string	NN
genres[ ]	string	NN
directors[ ]	string	NN
cast[ ]	string	NN
crew { }	object	NN
releaseDate	date	NN
duration	double	NN
createdBy	string	NN
createdDate	date	NN
lastModifiedDate	date	NN
_class	string	NN
_id_		

crew { } NN

producers[ ]	string	NN
music[ ]	string	NN

Service APIs

User Service

- **Partner accounts**
  - POST /partners
  - PUT /partners/{id}
  - GET /partners/{id}
  - DELETE /partners/{id}
- **Customer accounts**
  - POST /customers
  - PUT /customers/{id}
  - GET /customers/{id}
  - DELETE /customers/{id}

Movie Service

- **Movies**
  - POST /movies
  - PUT /movies/{id}
  - GET /movies
  - GET /movies/{id}
  - DELETE /movies/{id}

Payment Service

- **Payments**
  - POST /payments
  - GET /payments
  - GET /payments/{id}
  - DELETE /payments/{id}

Media Service

- **Files**
  - POST /files
  - GET /f/{id}
  - DELETE /f/{id}

Theater Service

- **Theaters**
  - POST /theaters
  - PUT /theaters/{id}
  - GET /theaters
  - GET /theaters/{id}
  - DELETE /theaters/{id}
  - POST /theaters/{id}/screens
  - PUT /theaters/{id}/screens
  - DELETE /theaters/{id}/screens/{id}
- **Shows**
  - POST /shows
  - PUT /shows/{id}
  - GET /shows/{id}
  - GET /shows?filter=value
  - DELETE /shows/{id}

Ticketing Service

- **Reservations**
  - POST /reservations
  - GET /reservations
  - GET /reservations/{id}
  - DELETE /reservations/{id}
- **Bookings**
  - POST/bookings
  - GET /bookings
  - GET /bookings/{id}
  - DELETE /bookings/{id}

Review Service

- **Reservations**
  - POST /reviews
  - GET /reviews
  - GET /reviews/{id}
  - DELETE /reviews/{id}

Offers Service

- **Offers**
  - POST /offers
  - GET /offers/{id}
  - GET /offers?parternId=xxx
  - GET /offers?customerId=xxx
  - PUT /offers/{id}?action=redeem
  - DELETE /offers/{id}
  - DELETE /offers?parternId=xxx
  - DELETE /offers?parternId=xxx

Subscriptions Service

- **Subscriptions**
  - POST /subscriptions
  - GET /subscriptions/{id}
  - GET /subscriptions?customerId=xxx
  - DELETE /subscriptions/{id}

Platform Features – By Roles

Partners

- ❑ **Accounts**
  - ❑ Login/Logout
  - ❑ Create Account
  - ❑ Update Account
  - ❑ View Account
  - ❑ Delete Account
- ❑ **Movies**
  - ❑ Create Movies
  - ❑ Search Movies
- ❑ **Theaters**
  - ❑ Create Theaters
  - ❑ Update Theaters
  - ❑ View Theaters
  - ❑ Delete Theaters
- ❑ **Screens**
  - ❑ Add Screens
  - ❑ View Screens
  - ❑ Remove Screens
- ❑ **Shows**
  - ❑ Add Shows
  - ❑ Update Shows
  - ❑ View Shows
  - ❑ Remove Shows
- ❑ **Offers**
  - ❑ Add Offers
  - ❑ Update Offers
  - ❑ View Offers
  - ❑ Delete Offers
- ❑ **Billing**
  - ❑ Add Billing Info
  - ❑ Update Billing Info
  - ❑ View Billing Info
  - ❑ Delete Billing Info

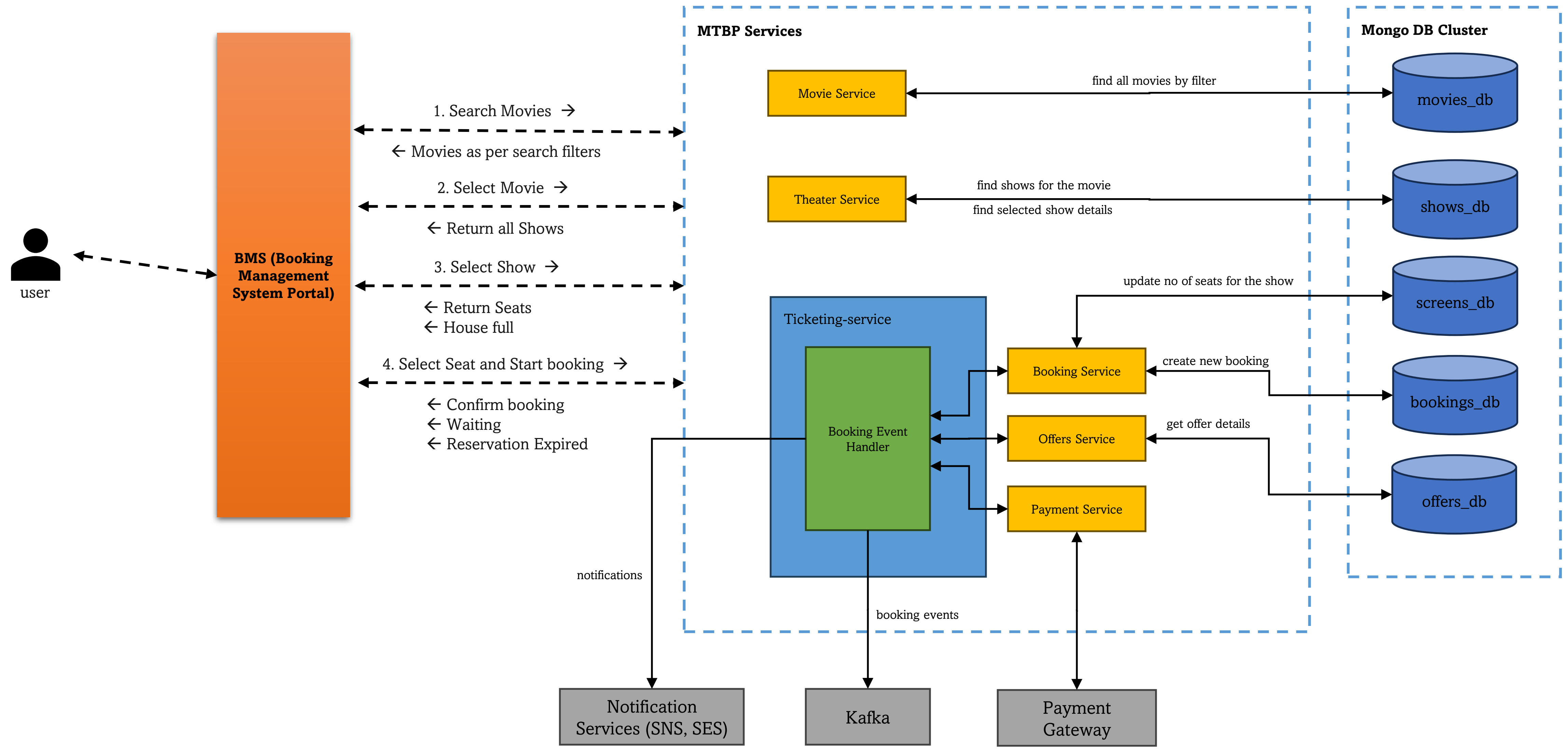
Customers

- ❑ **Accounts**
  - ❑ Login/Logout
  - ❑ Create Account
  - ❑ Update Account
  - ❑ View Account
  - ❑ Delete Account
- ❑ **Movies**
  - ❑ Search Movies
- ❑ **Reservations**
  - ❑ Reserve Seats
  - ❑ Cancel Reservations
- ❑ **Bookings**
  - ❑ Create Bookings
  - ❑ View Bookings
  - ❑ Cancel Bookings
- ❑ **Reviews**
  - ❑ Add Reviews
  - ❑ Update Reviews
  - ❑ Delete Reviews
- ❑ **Offers**
  - ❑ View Offers
  - ❑ Redeem Offers
- ❑ **Subscriptions**
  - ❑ Create Subscription
  - ❑ Update Subscription
  - ❑ View Subscription
  - ❑ Remove Subscription
- ❑ **Payments**
  - ❑ Add Payment Info
  - ❑ Update Payment Info
  - ❑ View Payment Info
  - ❑ Delete Payment Info

System

- ❑ **Notifications**
  - ❑ Send new movie notifications
  - ❑ Send booking notifications
  - ❑ Send cancellation notifications
- ❑ **Ads**
  - ❑ Show relevant ads

Use Case Flow – Ticket Booking



Design Points

- **Availability**
  - Deployment of the services into different regions and availability zones in AWS ensures high availability of the services.
  - AWS promises up to 99.99% average uptime at a region level or single instance level.
  - Using of Kubernetes also helps decrease service downtimes by automatically managing the rescheduling pods in failed nodes to healthy ones.
- **Performance**
  - Using ELB enhances performance of the backend services as the workload balance is maintained.
  - Using of CDN cache helps to exchange static media files faster and efficiently.
  - AWS auto scaling groups ensures horizonal scalability whenever required.
  - MongoDB cluster has horizontal scalability and has a good query performance for the given use cases.
  - Redis Cache can help to improve show search queries.
- **Observability**
  - Leverage health check feature of AWS ELB to make sure that the platform instances are healthy.
  - Use monitoring tools like Grafana and Prometheus to visualize and analyze the performance of the platform.
  - Create analytics events and do Business Intelligence on the generated data with AWS analytics tools.

Design Points

- **Security & Resilience**
  - Create network groups (public/private) to isolate services based on their access permissions.
  - It's a good idea to deploy all the backend services/other service clusters in a private subnet so that it can be accessed only within the VPC.
  - Use AWS WAF to mitigate threats like XSS and DDoS attacks.
  - Do an authentication check at ELB level using AWS services for the validity of incoming requests. It helps to avoid forwarding unauthenticated requests further deep into the network level.
  - Implement RBAC functionalities using Spring Security for backend services.
  - Use security hardened OS for virtual machines instances wherever possible
  - Avoid storing sensitive information (passwords, banking, card details etc) and if required store them in encrypted form
  - Use AWS Rekognition to do content filtering of media uploaded by users.
  - Use circuit breaker pattern and network timeouts in the backend services wherever required to ensure better resilience for the services.
- **Pricing**
  - Follow an approach of choosing a hybrid model (managed services + open source) while deciding the tech stack for efficient pricing.
  - It is not a good idea to depend only on AWS managed services to build the entire platform use open-source techs wherever suitable, e.g., Kafka, ELK stack, Prometheus, Grafana etc.
  - Use AWS managed services for aspects like notifications (SNS, SES), authentication, analytics. These are necessary components to bring up the platform which are also complicated to setup. They can be migrated services later.

Platform Monetization

- Ticket Reservations
    - Ticket prices booked for any shows from the theater partner.
    - Ticket prices from guest customers.
- Subscriptions
    - Subscribed customers will be eligible for special offers like
        - Discounted ticket prices
        - Special offers.
- Offers
    - Meal vouchers
    - Group booking offers etc
- Ads
    - Relevant advertisements on the web site.
        - Food ads
        - Fashion ads etc

Project Plan – High Level

Time	Sprints		Goals
Month1	Sprint 1	Sprint 2	1. Setup DEV & E2E enviornments
	Prepare and finalize development processes	Sign Up Functionality	2. Able to provision user and theater data
	Prepare and finalze automation testing processes	Integrate with AWS notification services	3. Able to make ticket reservations
	Prepare CI/CD pipelines	Login and Logout Features	4. Able to book tickets and do payment
	Prepare DEV environment	Prepare E2E environment	
Month 2	Sprint 3	Sprint 4	
	CRUD Customers	CRUD Theaters	
	CRUD Partners	CRUD Screens	
	Bug Fixing & Updating Sprint Demo comments	Bug Fixing & Updating Sprint Demo comments	
	E2E Testing - Sprint 2 contents	E2E Testing - Sprint 3 contents	
Month 3	Sprint 5	Sprint 6	
	CRUD Shows	Payment GW Integration	
	CRUD Reservations	CRUD Bookings	
	Bug Fixing & Updating Sprint Demo comments	Bug Fixing & Updating Sprint Demo comments	
	E2E Testing - Sprint 4 contents	E2E Testing - Sprint 5 contents	
Month 4	Sprint 7	Sprint 8	1. Implement Offers, Reviews, Subscription and Invoicing Features
	CRUD Offers	CRUD Subscriptions	2. Performance Testing Env Setup and Automation
	E2E Testing - Sprint 6 contents	Bug Fixing & Updating Sprint Demo comments	
	Bug Fixing & Updating Sprint Demo comments	E2E Testing	
Month 5	Sprint 9	Sprint 10	
	CRUD Invoicing	CRUD Reviews	
	Bug Fixing & Updating Sprint Demo comments	Bug Fixing & Updating Sprint Demo comments	
	E2E Testing - Sprint 8 Contents	E2E Testing - Sprint 9 Contents	
Month 6	Sprint 11	Sprint 12	
	E2E Testing - Sprint 10 Contents	Performance Testing Automation	
	Bug Fixing & Updating Sprint Demo comments	Bug Fixing and Review comments updating	
	Prepare and Deploy Performance Testing Env		
	Prepare scenarios for Performance Testing		
Month 7	Sprint 7	Sprint 8	1. Perforamnce Testing
	Performance Testing Review	Performance testing issue fixes	2. UAT and PROD Setup
	Prepare UAT Testing Enviroment	UAT Testing	3. UAT Testing and Bug Fixing
	Bug Fixing	Bug Fixing	4. Make platfrom production ready for Going Live
Month 8	Sprint 9	Sprint 10	
	UAT Testing	Performance Testing	
	Bug Fixing	Bug Fixing	
	Performance testing issue fixes	Prepare PROD environment	
Month 9	Sprint 11	Sprint 12	
	Run E2E tests on Production		
	Run UAT tests on Production		
	Make the system production ready		

Role	Count
Product Owner	1
Scrum Master	1
SE Team	1
Tech Lead	1
Developers	4
Testers	3
DevOps Team	2
E2E Testing Team	3

END

-- Backend System Design and Project Plan Overview