



Book My Movie

DESIGN | ARCHITECTURE | DATA MODELING | TECHNOLOGIES

A CASE STUDY - BY ASHISH KAW

Problem Statement

Problem statement (Version 1.3.3)

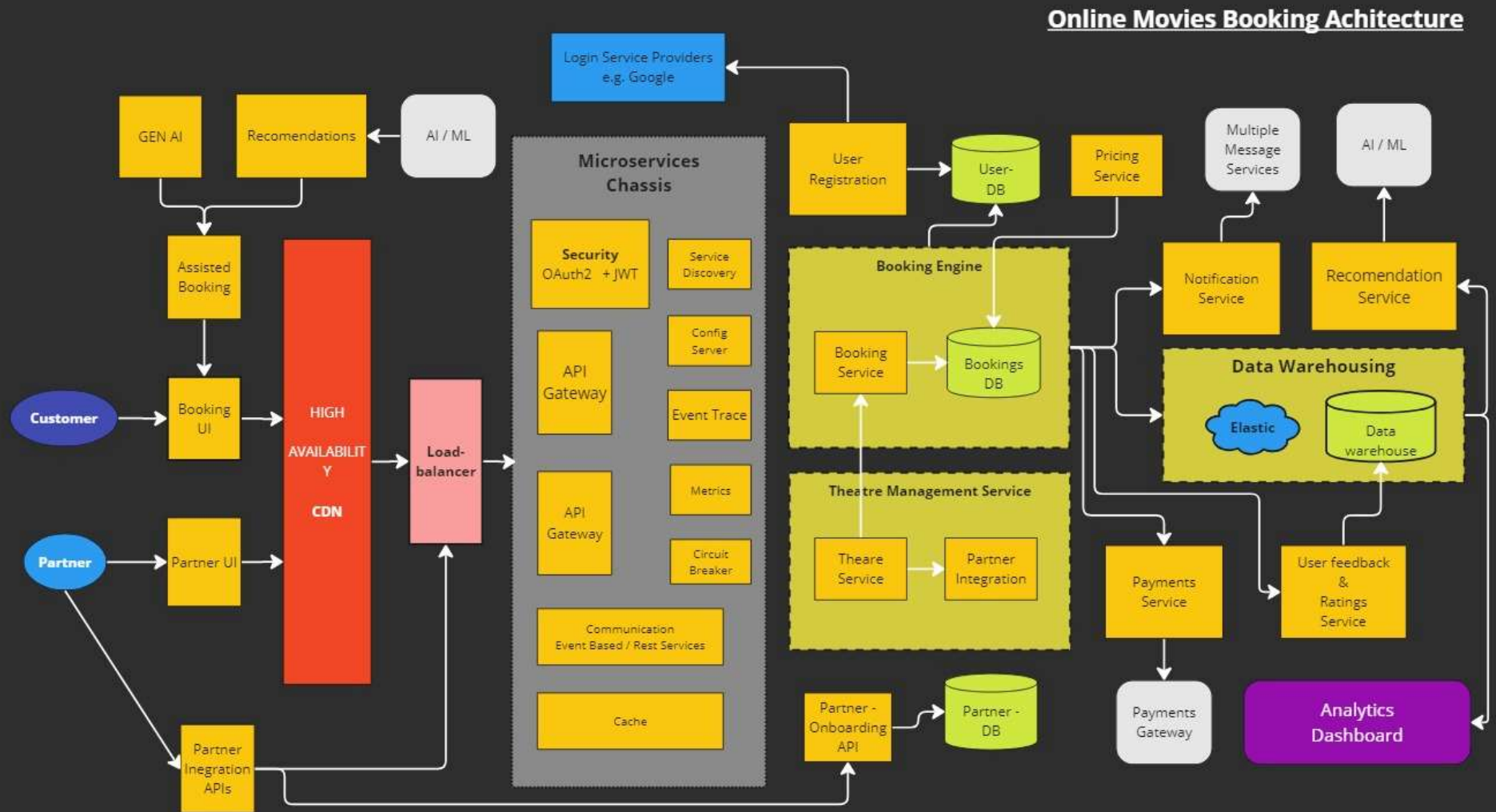
- XYZ wants to build an online movie ticket booking platform that caters to both B2B (theatre partners) and B2C (end customers) clients.
- Key goals it wants accomplished as part of its solution:
- Enable theatre partners to onboard their theatres over this platform and get access to a bigger customer 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.
- Technologies recommended

Language - Java and other add-on languages

- Frameworks- Any
- AI-Suggest
- Database - Any
- Integration and Data technologies- Any
- Cloud technologies- Any
- Preferred editor to build and present solution

→ [More Details about the problem](#)

High Level Architecture



Architecture Explained

User Experience

- User can be either Customer looking for booking for Partner for Theatre and Show Management
- Here I have not focused or elaborated on UI Options and if we go in that direction there are multiple options for a customer to engage in booking movies, possible options are:-
 - Mobile APP and Web Interface.
- For a good user experience and high performance we can go with :
 - **Micro-frontend Architecture** using **React JS Library**
 - **React JS Library** is considered good for single page applications and also provides support for mobile development with React Native.
- **Assistive Booking** can prove has Add-on in this competitive market era using **AI Recommendations** and based on users last bookings, location, language, current trends etc. and generating review and movie summary or different movie trailers using GEN AI can also be put in scope for future.
- Partners should also be provided Web Portal to manage theatre and shows, provide and register APIs and document schema for seamless integration along with usage metrics and reporting.
- **Content Delivery Network (CDN)** : If we are looking for global or regional presence for our movie booking platform the seems less user experience and high performance and low latency throughput CDN can be the best choice however we may not forget cost consideration that comes along with it, so deciding factor is always the amount of user base and geographical presence.
- **Load Balancers** : In case we do not go with CDN we still can also consider multiple load balancers configured with multiple API gateways and combination of Load balancers and CDN can prove very efficient for global presence and high performance.

Architecture Explained

Backend & Microservices

- **Microservices Chassis** : Considering which feature of microservices we need to develop to manage, scale, monitor, secure, decouple and failsafe our microapps and related / supporting services.
- We can consider **Messaging services like Kafka** for communicating with in-sync systems specially booking services which need to notify in no time as ticket is confirmed / cancelled / unblocked to make it available for other customers as well as for theatre backend services.
- **API Gateway** : Deciding API Gateways can be based on features it provides and cost estimation like for low cost we can opt for Zuul API Gateway. We can consider APIGEE as it provides more features like request rate limiting and secured access using OAuth2 along with integrations with IDP providers.
- **Circuit Breaker – Hystrix** will be required to monitor our micro apps health based on standard threshold to alert and disassociate that micro-app from receiving anymore requests if the time it recovers.
- **Cache**: we can consider application level if we are not opting for CDN and we should always consider for database level cache to save db hits and round trips. Redis can be considered for database level cache as it supports few data structures and has rich set of features.
- **Event Trace** : we should always write log asynchronously based on events and treat them as stream.
- **Config Server (Spring Cloud Config)**, **Service Discovery (Eureka)** are always required. **Metrics tools like Prometheus or Dynatrace** are good to have to monitor overall stats.
- **Data Warehousing** is important for getting over all insights and analysis about sales, reach, feedback, user interests, monthly / yearly usage reports. This is all very beneficial to calculate KPIs and business performance. Teradata, BigQuery, IBM DB2 and many more options are there to consider based on cost and features. For inhouse – we can consider Teradata or Snowflake or cloud based we can opt for native solutions like BigQuery in GCP.
- **Elastic** has many ways to deal and visualize unstructured data using Analytics dashboards like Kibana like log streams or audit logs.

Architecture Explained

Security

- **Data Security in Transit:** We need to make sure data being transmitted is encrypted and securely transmitted using secure protocols like HTTPS, TLS. Applying message digests and signature verification.
- **Data Access :** Minimum privilege should be provided using roles and policies that is required to get job done and data should be will labelled by its criticality.
- **Authentication:** Customer login should have an option for MFA.
- **Data and services should be well protected** by external different type of cyber attacks like SSRF, Injections, XSS, DDOS by fixing vulnerabilities and running scans for it, timely schedules of penetration testing and timely software / hardware updates and Auditing.
- **Data Privacy:** We should always take care of regional data protections policies like GDPR, COPPA etc.
- **API Security** can be placed at API Gateway level : tokenization's with JWT using IDP and Credential Grant can used.
- As **MFA** we can also add IP White listing for the services being exposed to other Theatre providers.
- Data Security at rest should be applied by encrypting sensitive data at rest like passwords, card numbers, identity number like Aadhaar Card, Pan Cards. This can done by using data encryption techniques like using DEK
- We should follow Standard Payments Polices like PCI – Payments Card Industry while dealing with online payments

Architecture Explained

Artifacts:

- **Booking Engine** : performs booking related services using in-sync shows and theatre data provided by the theatre providers.
- **Pricing Service:** Provides the tickets pricing based on different price calculation formulas this will include platform fees and application profile margins.
- **Theatre Management Service:** This is for theatre management related service like managing theatres, shows and updating seats.
- **Partner Onboarding:** new partners as theatre service providers.
- **User Registration** : onboarding new users using direct sign-up or user details service providers Google, Facebook.
- **Payments service and Payment gateway integration** : Payment Service will act as a wrapper / adaptor for payments gateway interactions.
- **Partner Integrations:** can be further divided into event driven / callback and restful services to make low latency highly resilient connections for interactions with theatre service providers.
- **Recommendation service: AI / ML** we can machine learning to create a recommendation engine for movie recommendations and we can create a AI model and use deep learning to train it to perform Assistive bookings. We can make use of OpenAI and Cloud provided compute systems like Vertex AI / TensorFlow to perform AI / ML operations.
- **Notifications Service:** can be used to send notifications to the customers for booking confirmations, offers also involves subscriptions. Notifications can be sent via multiple channels like SMS, Emails, App Push Messages, Social Media Apps etc.
- **User feedback and ratings service:** will be used to capture users feedback and ratings related to theatre or reviews for the movie

Code Implementation

API RUNNING CODE:

Code On GIT ON MY PUBLIC
REPO :

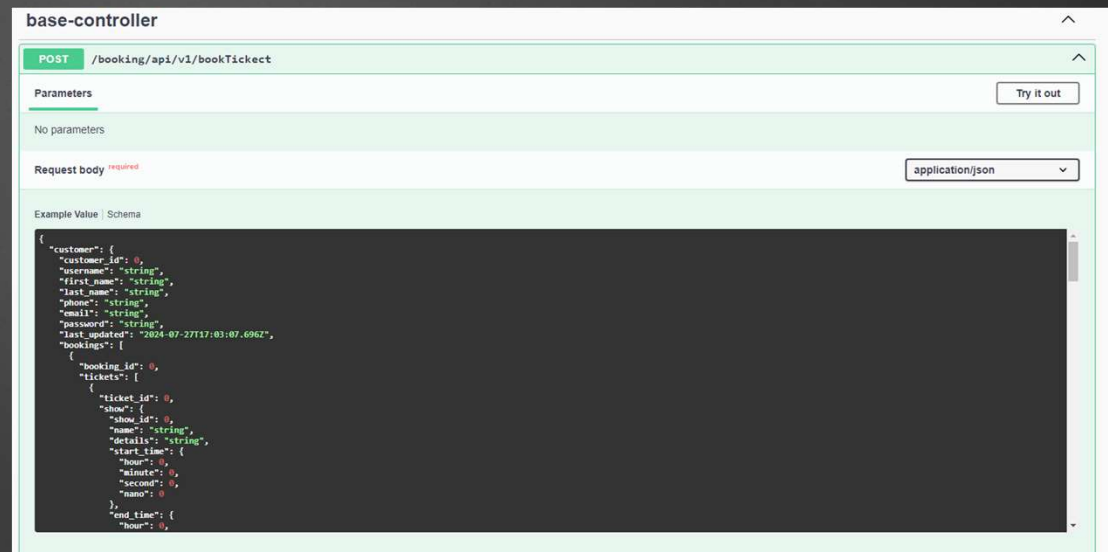
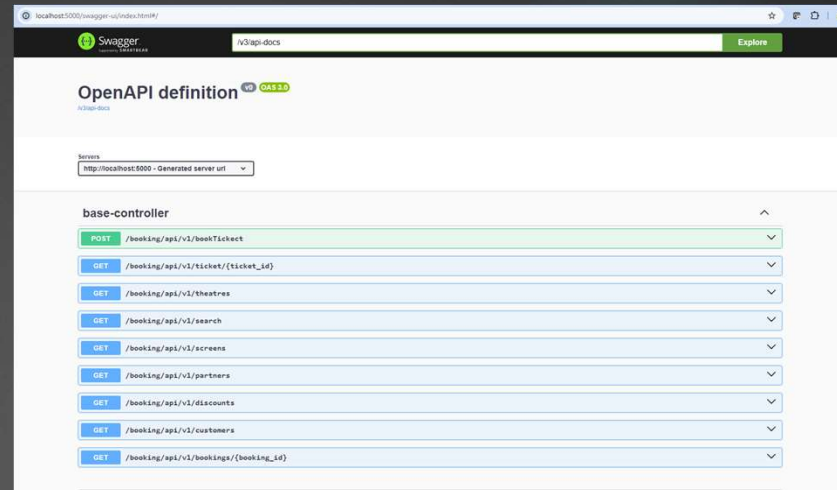
<https://github.com/AshishKaw/bookMyMovie>

Swagger API Contract:

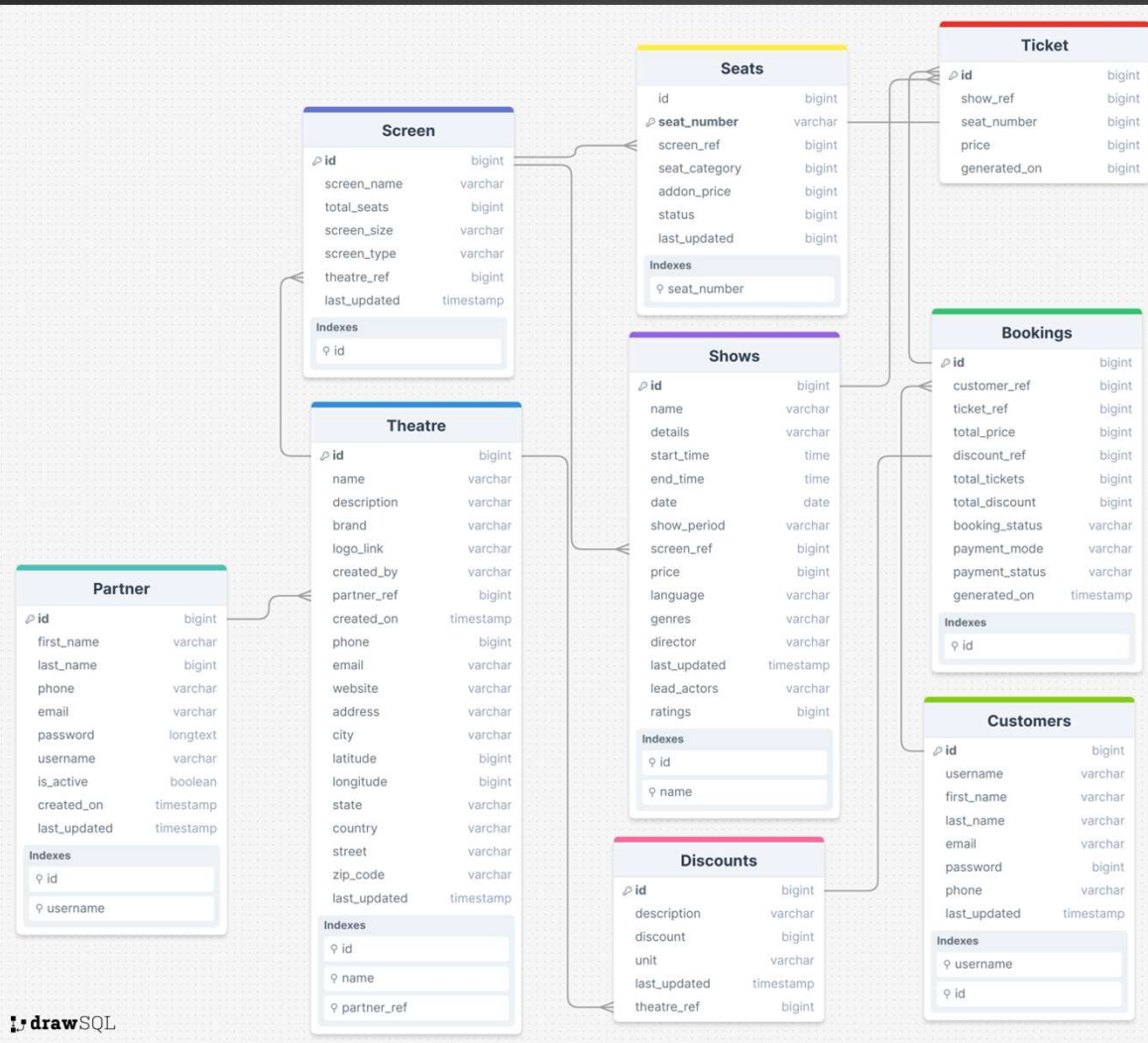
<https://github.com/AshishKaw/bookMyMovie/blob/master/swagger-api-contract-doc>

DATABASE DUMP (SQL DML and
DDL):

https://github.com/AshishKaw/bookMyMovie/blob/master/BOOKMYMOVIE_SQL_DB_DUMP.zip



Data Model




- It's a high level data model just to preview relationship between different entities that will be part of core booking engine and code demo is based on this data model to effectively maintain data consistency
- This data model is self explanatory and portrays multiple components required to create movie booking engine.
- This data model can be further extended for attaching other important components like Audit Logs, Feedback system etc.

Choosing the right Platform

- Decision for platform and infrastructure depends on factors like Budget, KPIs, ROI, Usage, Audience, Future Scope.
- A preferred solution would be to start with low level setup with resources based on the anticipated Audience.
- **On-Premise:** generally used for high security by banks and financial services providers
 - High Cost
 - Low Scalability
 - High Maintenance
 - Skill Requirement
 - high Availability = Very High Cost.
- **On-Cloud**
 - Will be Preferred for this solution
- For our application choosing Managed Kubernetes service will be beneficial:
 - We can start with multi clustered environment for Prod with Multi region for high regional availability.
 - Instances can be auto scaled based on max limit and min limit.
 - We can keep our lower and Dev environments in low configuration and zonal availability to save costs
 - For Database solutions we can make use e.g. Cloud SQL in GCP to get regional Availability of 99.99%
we can also create read replicas for providing more availability.
- We can use GIT based code version tool with efficient branching strategy and proper pull request and review process should be followed.
- We can make use of GitOps for setting devOps pipelines with help of tools like cloud build or Jenkins.
 - We can also configure Sonar gateway for code quality scans
 - We can also setup Policy AS Code (PAC) tools to capture severities and vulnerabilities in code

High Level Project Plan

- Requirement Study – Create Spike Stories
- List out modules
- Dependency Declarations.
- Finalize Functional Artifacts.
- Technology Considerations based on available skills, documentations, Industry standards, Community Support, Business Functions etc.
- Onboard Additional resources if required based to budget and construct and well organized team.
 - Usually Team Size should be less than 9 with min two resources for each tech involved.
 - If project is big or time to market is less then needs to split by business functions that will create different teams.
- Decide Cloud Infrastructure platform for application based on:
 - Technology Stack
 - KPIs
 - Business Functions;
 - Future Extensibility
 - Scope of load scalability
 - Anticipate Users and Request rate atleast for first 6 months
 - Cost Estimation and Budget.
 - Skill Set.
 - ROI
 - Post Delivery Maintenance Cost
- Start with a POC and engage with Stakeholders for the initial feedback
- Finalize an estimated MVP release or Beta release or initials module release
- Create Stories and plan sprints for MVP release
- Make sure Sprint is 2 weekly and plan releases every 2 – 4 sprints after MVP release.
- Engage with Stakeholders for continuous feedback.



*"I have tried my best to cover main points in short time,
Apologies for any grammar and spelling mistakes as in
short time I may have missed correcting them".*

Thanks!
