PICTURE PERFECT

An online movie ticket booking, review, and rating service

Design Documentation

This includes the purpose and scope of the project, alone with deliverables, the technology stack, system architecture and a tentative timeline.

Raghav Mahajan
1raghavmahajan@gmail.com

Table of Contents

Introduction

Picture Perfect is an online movie ticket booking, review and rating service. The service helps users generate review and rating content for movies across the world.

Purpose:

This project is meant for you to do now as a learning experience to get familiarized with the tech-stack that is in use at Clumio, Ltd.

Scope

The Picture Perfect online platform should facilitate ticket booking, rating and reviewing of movies, along with user profile management options. The website should be responsive, adaptive to multiple platforms and i18n enabled. A backend console that allowed for administrative controls to the website and its content should be provided as well, along with public API methods (given in the design doc). The service has to be highly concurrent, scalable and secure with high availability.

The following deliverables need to be provided for the completion of the project:

1. A public REST API that supports the microservices mentioned in system architecture.
2. Desktop website with the following:
    a. Responsive UI that adapts to different screen sizes (mobile, tablet, desktop)
    b. Home page - Listing of top movies across different categories
    c. Login page
    d. Simple and easy navigation, via categories menu and bread crumbs
    e. Search and listing of results, along with filters and pagination of results
    f. Ratings component - Adding and updating movie ratings from a rating widget
    g. Reviews component - Adding and updating reviews from a review widget
    h. i18n enabled – The reviews can be in any language
3. A Backend Admin Portal that supports the following:
    a. Adding/Removing/Disabling cineplexes
    b. Updating movie catalogue
    c. Updating movie ratings
    d. Updating movie reviews

Milestones

Tentative timeline	Topic
3rd week of April ‘20	Comprehensive design document incorporating the following points. Technology Choices AWS Services to be used UX - Layout mocks High-level test strategy Deployment strategy
2nd Week of May ‘20	API contracts: URL, payload and response structure Minimum 1 user flow (ex: Catalogue functionality for user and admin) end to end implemented Flow completion includes Good looking UI with CSS REST API definitions and implementation DB Schema
End of May ‘20	Jenkins Integration
2nd week of June ’20	Remaining flows Testing Fit and finish

High Level Design

Overview

As mentioned in the initiation document, the service comprises of the following components:

- Desktop website
- Desktop Web-based administration and management console for backend operations
- Public REST API for integration with other movie and media portals

Technology Stack

Technology	Purpose
Golang	Microservice architecture
Typescript	Client side language
ReactJS + Redux + Sass	UI
Enzyme	Client side testing
Selenium	UI automation
DynamoDB + PostgreSQL	Content storage + Database
Postman	REST API
Jenkins	CI/CD

Golang

GoLang is chosen as the language for the microservice architecture because of its high Performance, low memory footprint, fast compile times, fast launch times and proper concurrency. It's simplicity and ease-of-code attributed to absence of any rigid programming paradigm constraint makes Golang standout among other server-side technologies.

TypeScript

Developed and maintained by the Microsoft Corporation, it is a superset of JavaScript and contains all of its power along with additional features like code-completion compilation and error correction abilities, support of OOP paradigms and type definition support.

ReactJS + Redux + Sass

By offering better reusability of system components, React allows large web applications to change data, without reloading the page. This makes them fast, scalable, and simple.

Redux is a predictable state container for JavaScript applications. This will help write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

Sass will enable us to write more stable, powerful, and elegant CSS code.

DynamoDB + PostgreSQL

The data will be stored according to the use case and performance considerations. The

movie catalogue and reviews data, being highly dynamic and requiring high concurrency, would be stored in a NoSQL database due to its ability to handle unstructured content and scale easily. And the transactional data, requiring ACID compliance will be handled by a Relational Database System.

Schema flexibility lets DynamoDB store complex hierarchical data within a single item and composite key design lets it store related items close together on the same table. This amplifies the Speed, Scalability and Availability of DynamoDB that makes it a great database solution for high concurrency web apps.

PostgreSQL is chosen as the RDS as it ensures swifter execution of complex queries as well as better data integrity compared to other RDBMS like MySQL.

System Architecture

Backend

The following API services will be implemented:

- Catalogue - This service is to retrieve and maintain the catalogue of movies, documentaries and television programs.
  - Typical user operations would include (both logged in or an unauthenticated user)
    - GET /movies/catalogue - Get a paginated list of movies, along with the associated media (links to the thumbnail pictures)
    - GET /movies/catalogue/{name} - Get a movie/documentary by name with detailed info and the media links images, videos
    - GET /movies/catalogue?{query} - Search a movie with filter and sort criteria
      - Filter - could be on any attribute name, language,
      - Sort - Sort the results in ascending or descending order
      - Paginate - To paginate the results to obtain the results in chunks
  - Typical backend admin operations would include
    - POST /movies/catalogue - Add a new item to the catalogue
    - PUT /movies/catalogue - Update an item in the catalogue
    - PATCH /movies/catalogue - Update a specific attribute to an item in the catalogue

- IAM – Identity and Access management - this is to authenticate a user, and identify if the user is general user or somebody who can manage the PicturePerfect operations based on a role and privilege
  - A generic user role - Should not have access to the backend console but only to the PicturePerfect website
    - POST /login - create a new token for a login session
    - POST /logout - Invalidate the session and logout
    - POST /reset - Reset the password to a new one
  - An admin user role - Should have access to both the backend console and the PicturePerfect website
    - POST /login - create a new token for a login session
    - POST /logout - Invalidate the session and logout

    - POST /reset - Reset the password to a new one

- Ratings - This service lets users add, update or delete an existing rating for a movie
  - Typical user operations
    - PUT /movies/rating/{movie} - Adds a new rating for a movie
    - DELETE /movies/rating/{movie} - Delete the rating for a movie added by the user
  - Typical admin operations
    - DELETE /movies/rating/{movie} - Deletes all the ratings applied to a movie
    - DELETE /movies/rating/{user} - Deletes all ratings set by a user
    - DELETE /movies/rating/{movie}/{user} - Delete the rating for a movie added by a user

- Reviews - This service lets users add or update an existing review for a movie
  - PUT /movies/review/{movie} - Add or update a new movie review
  - DELETE /movies/review/{movie} - Delete a movie review created by the user

- Shows - This service lists the cineplexes where the movie is being screened in a given city
  - User operations
    - GET /movies/shows/{city} - List all shows in all cineplexes in a city
      - This should have the ability to filter, paginate and sort the results
    - GET /movies/shows/{city}/{movie} - List the cineplexes screening a particular movie
  - Admin operations - In addition to the user operations above, admins can do the following
    - POST - /movies/shows - Add a new show or add a new cineplex
    - PUT - /movies/shows - Update the show timings
    - DELETE - /movies/shows - Delete a show
    - DELETE - /movies/shows/{movie} - Remove a movie from all screens

Database Design

Relational database systems (RDBMS) and NoSQL databases have different strengths and weaknesses:

In RDBMS, you design for flexibility without worrying about implementation details or performance. Query optimization generally doesn't affect schema design, but normalization is important. But, in NoSQL, you design your schema specifically to make the most common and important queries as fast and as inexpensive as possible. Your data structures are tailored to the specific requirements of your business use cases.

Since an RDS uses expensive joins to reassemble required views of query results, it incurs a huge performance hit while performing complex queries. Whereas, due to the schema flexibility of NoSQL, the same data is close together and can be queried faster.

That’s why, almost all web applications with huge dataset maintain a combination of different databases for their performance, security and scalability tradeoffs. Sites like Amazon and Bookmyshow, both follow the principle of storing their highly traffic website content (like product/movie listings) on high performance NoSQL databases with their static configuration, authentication and transactional data on RDS systems for better data

integrity and security. Keeping the same in mind, the database design combines the scalability and performance of DynamoDB for storing user data, movie listings and details, and a PostgreSQL DB hosted on Amazon RDS for storing booking history, transactions and authentication data.

The tentative data-model is given below:

- User:

Attribute	Data Type	Description
userId	String	Unique UserId
name	String	Full name
emailId	String	Email Address
ifVerified	Bool	Account verification status
city	String	City the user resides
phoneNo	String	Phone number
address	String	Address of the user
password	String	SHA256 Hashed password
role	Enum	Privilege of the user. Default is user, can be changed by authorized admins or higher.
reviews	Array of [review]	Array of review references
bookings	Array of [booking]	Array of booking references

- Movie:

Attribute	Data Type	Description
movieId	String	Unique identifier for movie, with serial numbering
title	String	Title of the movie
languages	Array of strings	Language of the movie
releaseDate	Date	Date of release of the movie
genre	String	Movie genre selected from set of possible genres.
duration	Integer	Duration of the movie (in sec)
thumbnail	String	Link to the static file image
media	Array of string	URL to related media
synopsis	String	Movie synopsis
writtenBy	String	Movie writer
directedBy	String	Movie Director
reviews	Array of [reviews]	Array of references to reviews for that movie

- Cineplex

Attributes	Data Type	Description

cineplexId	SERIAL	Unique identifier for cineplexes, with serial numbering
name	String	Name of the cineplex
city	String	City of the cineplex
address	String	Address of the cineplex
phoneNo	String	Phone number of the cineplex

- Reviews:

Attributes	Data Type	Description
userId	[User]	Reference to userId
movieId	[Movie]	Reference to movieId
rating	Integer	Rating of the movie by the user (0-5)
review	String	Review of the movie by the user

- ShowListing

Attributes	Data Type	Description
showId	String	Unique identifier for cineplexes, with serial numbering
cineplexId	String [Cineplex]	cineplexId of the Cineplex where the show is happening. Foreign key of cineplexId in Cineplex.
movieId	String [Movie]	movieId of the movie being screened. Foreign key of movieId in MovieCatalogue.
datetime	date	Date and time of start of the show.
ticketsLeft	INTEGER	Number of tickets left for the show.

- Booking

Attributes	Data Types	Description
bookingId	String	Unique identifier
userId	[UserInfo(userId)]	userId of the User who has done the booking
showId	[ShowListings(showId)]	showId of the Show booked
tickets	Integer	Number of tickets

Website Interface

The main website will have a feature rich, simple interface that's in line with what's in the market right now. It focuses on easy navigation and design layout and won't confuse users with too many options. The mockups can be viewed online using this link: https://wireframe.cc/pro/pp/b56a294db333823

An offline version of the same is attached herewith.

Admin Portal

The admin portal will offer a simple, barebones interface to perform the following tasks:

- Adding/Removing/Disabling cineplexes
- Updating movie catalogue
- Updating movie ratings
- Updating movie reviews
- Adding/Updating/Disabling promos and vouchers
- Garbage collector ensures invalid, biased and reviews by automated bots are cleared up.

Testing and Automation

For client side testing Enzyme is used. Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output. Enzyme provides additional testing utilities for testing.
Selenium WebDriver is a web automation framework that allows you to cross-test against various browsers. It is one of the most preferred testing tool-suite for automating web applications as it provides support for popular web browsers which makes it very powerful.

We incorporate Continuous Integration and Continuous Deployment (CI/CD) by :

- Implementing a Jenkins job for running unit tests for each commit
- Implementing a Jenkins pipeline for functional testing
- Implementing a Jenkins pipeline for deployment

The pipeline will be used to:

- Deploy and upgrade microservices
- Deploy and upgrade front end UI