Software Design Document

MoviMingle

Contents

Introduction

The application I will be developing for this semester, Movimingle is designed to allow movie selection for croups, which will be aimed at supporting a scalable user base from 1 million to over 100 million. This document will demonstrate an application of the professional standards, solve problems through investigative methodologies and will ensure a scalable, secure and efficient architecture. The main goal of the application is to provide a seamless user experience by focusing on data security and scalability.

User Stories

Story ID	As a/an	I want to...	So that I can...
UST-1	User	browse through a curated list of movies	find films matching my interests easily.
UST-2	User	be able to add movies to a personal watchlist	keep track of movies that I like.
UST-3	User	be able to join a watch party	connect with people I want to watch a movie together.
UST-5	User	submit watch movie suggestions for the random voting	contribute to the roster of movies in the watch party and the system will select one of them.
UST-6	User	create a watch party	Connect with people I want to watch a movie together.
UST-7	New user	sign up with minimal steps	quickly start using the application.
UST-8	User	control who sees my activity	maintain my privacy.
UST-9	User	see movie posters and trailers	make better-informed decisions.
UST-10	User	save movies to a watchlist	remember to watch them later or use them in the voting process.
UST-11	User	see content in my language	fully understand and enjoy the service.
UST-12	User	access the application from anywhere	enjoy a seamless movie selection experience.
UST-13	User	receive personalized movie suggestions	explore movies suited to my tastes globally.
UST-15	User	find movies based on group preferences	discover new films that fit our collective taste.

Functional requirements

ID	Requirement	Priority
FR-1	Users can create accounts	**High**
FR-2	Users can create and manage groups/parties	**High**
FR-3	Users can browse movies	**Medium**
FR-4	Users can vote on movies to watch, once in a party	**High**
FR-5	System selects a random movie from those with the highest votes in the party	**High**
FR-6	Users can view movie description	**High**
FR-7	Users can view trailers and ratings of a movie	**Medium**
FR-8	Users can receive personalized movie recommendations	**Medium**
FR-9	Users can share their selections on social media	**Low**
FR-10	Users can submit and view reviews	**Medium**
FR-11	Users can manage their account settings and preferences	**Medium**
FR-12	The application supports multiple languages for international users	**Low**
FR-13	Users can sign up using email	**High**
FR-14	Users can sign up/in using social media accounts	**Low**
FR-15	Users can set privacy levels for their activity on the platform	**Medium**
FR-16	Users can add movies to a personalized watchlist	**High**
FR-17	The application is accessible from anywhere, adapting to different screen sizes and resolutions	**Medium**

Non-functional requirements

ID	Requirement	Priority
NFR-1	Ensure data privacy and security	High
NFR-2	Interface must be intuitive and user-friendly	Low
NFR-3	System scalability for growing user base	High
NFR-4	Optimize application performance for responsiveness	High
NFR-5	Minimal downtime during updates and maintenance	Medium
NFR-6	Support cross-platform compatibility	Low
NFR-7	Implement robust error handling and feedback mechanisms	High
NFR-8	Comply with global data protection regulations	High
NFR-9	Adaptive UI for different screen sizes and resolutions	Medium
NFR-10	Localization for various languages and formats	Low

Technology stack

The chosen technology choices align well with the project's non-functional requirements about scalability, performance, availability, and security. The following sections detail the exact technology choices and a bit about the rationale. For more detailed rationale, you can see the documents *Database Choice Document* and *Backend Technology Rationale*

Backend Framework

Spring Boot

For the backend I have chosen Spring boot due to its ability to handle extensive load and user demands. This is very helpful as it will be very cost effective, since it will need less scaling. As well as that, Spring Boot significantly simplifies the development process with its relatively easy setup for microservices. The familiarity to it is also a fact that had a significant weight in my choice as that will make the development process even easier and faster and I will have more time to work on the rest of the requirements for the semester.

Authentication and Authorization

Auth0

Auth0 is chosen for the authentication and authorization of users due to its ease of implementation and strong security. I have decided to use external services for the security since the responsibility for the handling of the personal user data is not handled by something created from scratch by me. The time efficiency is an additional factor worth considering and saving on development time will not only help with developing the rest of the application on time, but in a scenario where a company has to decide what auth to use, choosing something like Auth0 may be more cost effective than allocating resources for the development of something from scratch. Also, Auth0 provides a quick and secure way for users to log in using various identity providers (such as Google, Facebook, and email/password).

External API Integration

The Movie Database

Since the use of The Movie Database (TMDB) in my case is free, this means that not only is it time efficient to implement this, instead of developing a whole microservice with dedicated database, but also the costs are no different. By using the external API, the users are presented with always up to date information about all of the movies they are interested in.

Containerization and Orchestration

Docker

Docker is very useful for the use of containerization for the microservices. It ensures consistent environments across development, testing and production. The consistency is very important for an application with the intent of being an enterprise level software product as it streamlines work processes

and leads to increased efficiency. For example, any one of the microservices can be deployed and Docker will ensure that all dependencies are correctly configured and the service behaves identically on different machines (developer's machine, in production, etc.).

Kubernetes

Due to its capabilities in automatic scaling, load balancing, and self-healing I have chosen Kubernetes for orchestrating the Docker containers. For my application Kubernetes would dynamically scale the Voting and Movie Selection Services based on real-time user activity. When the number of users requests for the voting increases, Kubernetes will automatically instantiate additional service instances to manage the load. By doing this it will maintain optimal performance and customer experience.

API Gateway

Envoy

I have selected Envoy, firstly because it has advanced routing and load balancing. Another reason to chose an API Gateway from an external provider is that I will not lose any of the limited time during this semester on developing a gateway from scratch. This of course has some drawbacks as if something breaks or does not work like expected, I will need more time to see where the mistake is and potentially will make the configuration process even longer than developing the gateway from scratch. The role of envoy is to efficiently distribute the requests to the appropriate microservices and their multiple instances, so that a service does not become a bottleneck for the performance.

Databases

Cassandra

For the database I have chosen Cassandra for its horizontal scalability and high write and read throughput. Initially my intentions were to use it just for the Voting and Movie Selection service and the rest would use a different, SQL database. However, after rethinking this decision I have made the conclusion that since I am not using the features of an SQL database like PostgreSQL, I only need Cassandra for my database. The implementation will include the PostgreSQL database, since making changes at the later stages of the semester may compromise my ability to finish all of the other necessary implementation steps, but is something that is worth noting.

PostgreSQL

As mentioned in the above paragraph, using the PostgreSQL was backed with the argument that transactional integrity and complex query support are crucial for the User Management and Favorites Management services. After the talk with Marcel Boelaars, one of my teachers and mentors for this semester, I came to the agreement with him, that since the software does not handle sensitive financial or medical data, where transactional integrity is important, I will not need the added complexity of two different database technologies. I have mentioned this in the previous paragraph, but I will not be changing my database setup for this semester as it could put unnecessary obstacles along my way right in the end of the semester.

Project architecture

The project architecture of the project is designed with scalability and performance in mind. Below are the key components of the architecture.

User Interface

React Application

In the plans for this project exists a user interface, built on React. However, due to the time restrictions of the semester and because the UI is not something that is a must for the completion of the Learning Outcomes that determine my grade, an implementation this semester will not be present.

Microservices

The application consists of several microservices, each of them responsible for a specific functionality. This setup ensures that the development, deployment, and scaling is different for each service.

- User Management Service: Manages user account information. (due to the decision of using external auth provides, the relevance of this microservice will need to be assessed)
- Favorites Management Service: Manages favorite movies for each user and is responsible for the retrieval of movie data from the external movie API.
- Voting and Movie Selection Service: Manages voting processes and determines final movie choices, based on each of the watch party members' list of favorite movies.

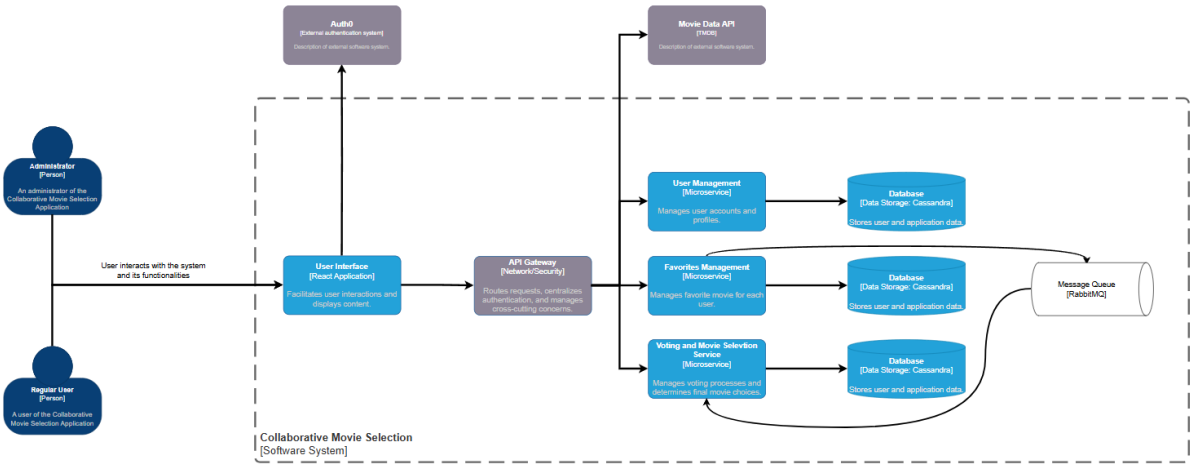
API Gateway

Envoy is used as the API Gateway, routing requests to the appropriate microservices and providing features like load balancing. This setup provides efficient request handling and system reliability.

Message Queue

For the asynchronous communication between the different microservices a messaging queue is used.

Architectural diagram



Sequence Diagram

Below is a sequence diagram representing the flow in one of the features implemented in the application.