CO4353

DISTRIBUTED SYSTEMS

DS-VIDEO STREAMING SYSTEM

Group No. : 1

17 / ENG / 020

17 / ENG / 024

17 / ENG / 046

17 / ENG / 076

17 / ENG / 095

Table of Contents

Introduction

This mini project is an attempt to replicate the basic working subsystems of a video streaming service such as YouTube TV, Netflix, Disney+ etc. Here the main aim was to develop the system based on a distributed architecture. To achieve that, implementing micro services were chosen. The users can enter the website and register by using a username and a password. The registered users are prompted to enter their personally preferred movie categories and when they log in to the system, they will receive suggested movies according to their preferences. When a movie is selected, the ratings that were given by other users are also displayed and the user can add ratings for movies as well. To implement the microservices the Spring Boot Framework was extensively used with the Netflix Open-Source Stack. Netflix OSS has provided the Edureka Discovery Server, Zuul API Gateway and a load balancing stack named Ribbon. For the front-end design AngularJS framework was used.

Architecture

When developing a project with a large scope the architecture of the entire system must be well thought out because once the system is built changing the architecture is nearly impossible. The two main architectural debates in the ongoing tech world are between monolithic and microservices. While both possess advantages and disadvantages over the other, both architectures are heavily used in occasions that are suitable for them. When a streaming service is considered, the service must serve millions of people concurrently and thus creates a huge demand for the core sub systems that are in the service such as logging in, signing up new users, movie searching systems, recommendation engines etc. And since a streaming service evolves from time to time by adding more features, it must be done with minimum effect on the end user's experience. Not only that a streaming service should utilize back-end resources efficiently and must allow scaling in both directions as well because it needs to be implemented in many countries around the world. When considering these facts, a microservice architecture provides more pros in the sense of a streaming service rather than a monolithic architecture because it allows all the above advantages to the streaming service. Hence, to complete this project a microservice architecture was chosen.

The entire system can be summarized as shown in figure 1. There are 4 core services in the system named Login and Authentication Service, Categorized Movies, Movie Preferences and Movie Reviews. The specialty in the proposed microservice architecture is, each of these services have

its own H2-memory database. As the service registry Eureka Discovery Server is used. The discovery server registers the services according to their IP addresses and port numbers. When a user is requesting a service the discovery server serves the IP address of the requested service and the API request can then perform its task.

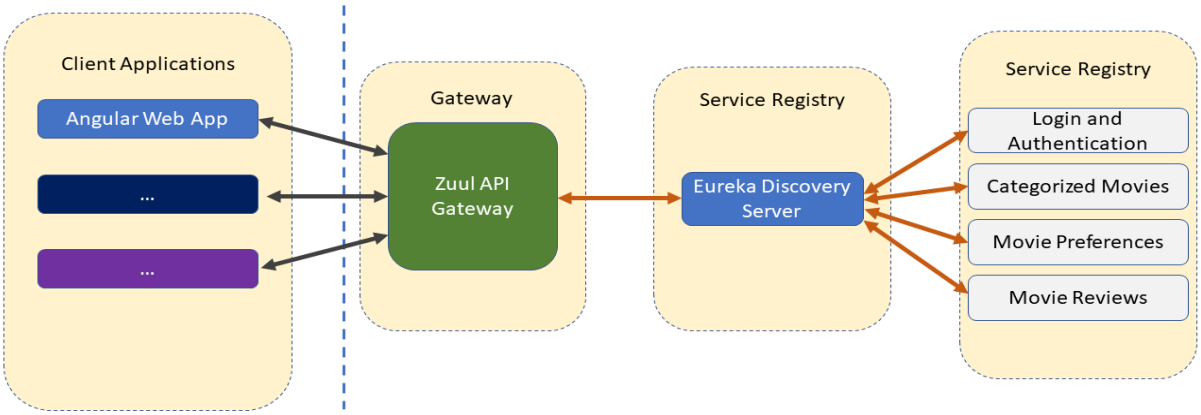

Figure 1: The higher-level architecture for the system

The Zuul API gateway acts as the bridge between the microservices and the front end of the applications. By having an API gateway, integration of new services to the system is very easy. Apart from that it improves the security of the overall system because all the requests flow through the API gateway and it allows the requests to be scanned before they are sent to the backend. Another advantage of having an API gateway is that it allows to monitor the load each microservice receives and the maintainers can easily perform load balancing tasks. All the communication between the services and the front end and the backend are performed using RESTful API calls. And when building the RESTful API calls the standards of HTTP 2.0 were followed with response codes.

The backend services are then split out into distinct services based on domain knowledge. Each service is assigned its own set of tasks. We employed various micro services for our primary features. Inter-microservice communications may also exist within the backend. However, to meet the aims of the microservice architecture, these inter-service communications should be kept to a bare minimum. Single responsibility, loose coupling, interface segregation, deployability and availability over consistency were all considered when building the entire system.

In the implementation, the Zuul API gateway and the Eureka Discovery server given in the Netflix stack were used to configure the workflow of between the microservices and the frontend. Apart from the service of providing the addresses of the microservices required by the API calls from clients, the discovery server provides the service of acting as the agent between interservice connections as well. Whenever a service requires to run another microservice the Eureka discovery server is contacted to gain the address of the required microservice. As discussed earlier an API gateway acts as a guide to the front-end users to engage with the backend of the system but in theory, a client issuing requests to all the microservices available in the backend of the web app is not feasible. Because it will add too many round trips back and forth and would increase the loading time of the components and increase the latency. Hence the system must have a limited number of microservices to achieve the entire functionality of the system which is why the given 4 microservices were decided. As an example, the login and authentication services can be coupled together to create one microservice because the jobs of the two sub systems overlap and creating two microservices add an additional overhead to them. By considering all these facts the final architecture of the system was decided.

Core Services of the System

The core services implemented in the system are Login and Authentication, Movie Preferences, Movie Review and Movie Category services. Each of these services have different port numbers to identify them uniquely in the discovery server and have different functionalities, inter service communications and numerous endpoints to provide various services. Apart from that each microservice contains h2 database tables to maintain the related data. The specialty here is it can scale up the number of services needed without any regard on the frontend of the system which is the main reason for the emergence of the micro services architecture.

Login and Authentication Service

The login and authentication service provides users with the capability to either register to the system, provide their movie preferences (categories) and login to the system. This micro service interacts with the Movie Preferences service and the Movie Category Service to serve its course to the end user.

Table 1: Details about the Login and Authentication Service

Service	**Login and Authentication Service**	
Port Number	**8081**	
Functionality	Logs in a user to the system using the username and password	
	Authenticates the user	
	Sign in a new user to the system	
Endpoints	GET/login	Requests the login details of the user and returns it.
	GET/signup	An error is returned if an accidental request is sent to this endpoint.
	POST/signup	Send the data of a new user to the server to register.
	POST/login	Send the username and password of the user to login.
	GET/auth	Perform the authentication of a user.
Inter-Service Communications	Movie Preference Service	After logging in, the preferences of the user are loaded.
	Movie Category Service	The movies corresponding to the user's preferred categories are loaded.
H2 Database Tables	**UserCredentials Table**	
	username	Primary Key
	password	

Movie Preferences Service

The Movie Preference micro service has the duty of preserving the user's preferences and it serves the preferences of each user when requested to the correct endpoint as well.

Table 2: Details about the Movie Preference Service

Service	**Movie Preference Service**	
Port	**8082**	
Functionalities	Control the user preferences Add and update user preferences	
EndPoints	POST/preferences	Save users preferences.
	GET/preferences/{username}	Get preferences for specific users.
	POST/addpreferences	Update user preferences
	POST/removepreferences	Remove user preferences
Interservice Communications	Login and authentication service	When user needs to change the personal preferences for the own profile, credentials are loaded from this microservice.
DB tables	**UserPreferences**	
	username	Primary Key
	choices	

Movie Review Service

The movie review service has the responsibility of adding movie review, ratings and comments that are submitted for the users for specific movies and serve the ratings of a movie when it is requested to a specific endpoint.

Table 3: Details about the Movie Review Service

Service	**Movie Review Service**	
Port Number	**8083**	
Functionality	Allowing users to add comments and reviews for specific movies.	
Endpoints	GET/moviereview/{moviename}	Retrieving reviews for specific movie
	POST/moviereview	Save movie reviews in the database.
	GET/getreviewbyrating/{ratingvalue}	Retrieving movies by rating given by registered users.
	GET/getreviewbyusername/{username}	Get all the reviews added by a specific user to the database.
Interservice Communications	Categorized movie service	Load the movie poster image when loading the reviews for specific movie.
DB Tables	**MovieReview**	
	id	Primary key
	moviename	
	username	
	ratings	
	comments	

Movie Category Service

Movie Category service is the service with the responsibility of maintaining the movie categories, returning all the categories and the movies according to the categories when requests come from the users. This service communicates with the Login service to display the related movie suggestions for the users according to their preferences after the login.

Table 4: Details about the Movie Category Service

Service	**Movie Category Service**	
Port Number	**8084**	
Functionality	Categorize the movies	
	Maintain the categorized movies	
Endpoints	POST/categorizedmovie/	Saving the movies according to the category.
	GET/categorizedmovie/{category}	Get all the results available for a specific category.
	GET/getallcategories	Get all the movies in the database.
Inter-Service Communications		
H2 Database Tables	**CategorizedMovies Table**	
	id	Primary Key
	category	
	moviename	
	url	Holds the url for the movie poster

Discovery Server

The Eureka Discovery server which is provided by the Netflix OSS and must be configured with a different IP and a port number for the API gateway to send requests to it. The configurations done to the Eureka server can be depicted as follows.

```
eureka.client.registerWithEureka=false

eureka.client.fetchRegistry=false

server.port=8761

spring.application.name=Movie_Services-Discovery-Server
```

API Gateway - Zuul

The Zuul API gateway must also be configured just as the Eureka server with an IP and a port number. Apart from that the API gateway must also know the paths for the services, the IP addresses, and their names to be able to request. These must be pre-configured as well.

```
spring.application.name=api-gateway

zuul.sensitive-headers=null

zuul.prefix=/api


zuul.routes.Movie-Login.path=/Movie-Login/**

zuul.routes.Movie-Login.service-id=MOVIE-LOGIN


zuul.routes.Movie-Preferences.path=/Movie-Preferences/**

zuul.routes.Movie-Preferences.service-id=MOVIE-PREFERENCES


zuul.routes.Movie-Reviews.path=/Movie-Reviews/**

zuul.routes.Movie-Reviews.service-id=MOVIE-REVIEWS


zuul.routes.Categorized-Movies.path=/Categorized-Movies/**

zuul.routes.Categorized-Movies.service-id=CATEGO
```

User Interface

For the initial testing of the system, Postman was used. Postman can send API requests to the API gateway and catch the returning response. In the following figure 2, it depicts the login request and the figure 3 depicts the response arrived from the service in return.

Figure 2: Postman POST request for the login

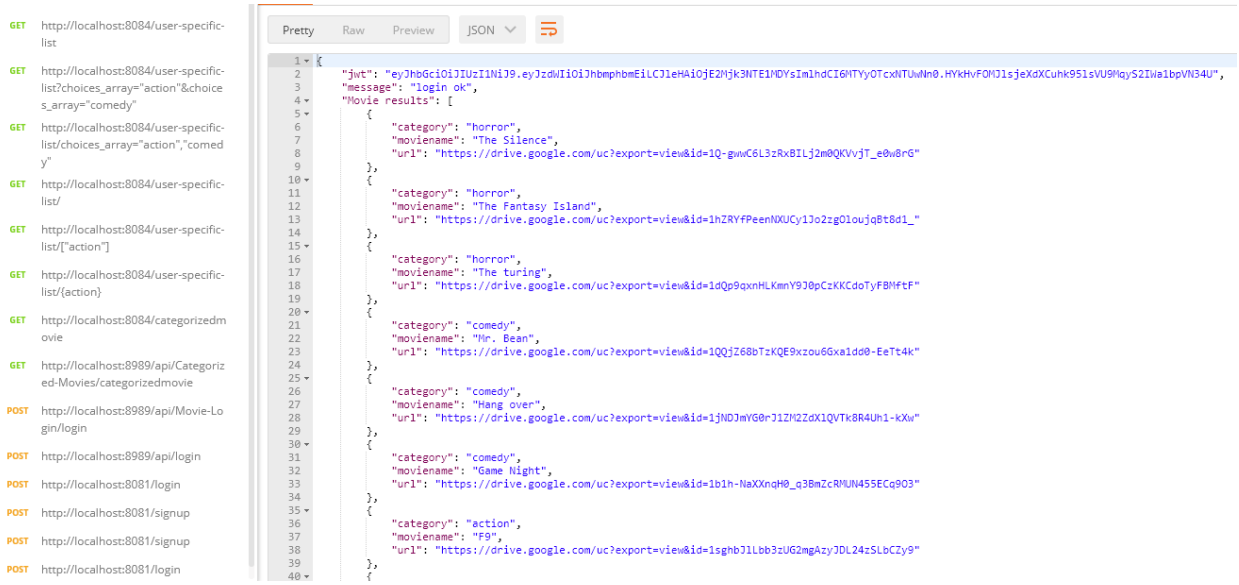

Figure 3: POST response for the login request

As depicted in figure 3, the response for a successful login request will contains the preferences of the user and the related movies for each of those preferences. The response arrives as a JSON and later in the implementation of the UI it is then broken down into an array and used to generate cards for each of the movies with their posters as depicted in figure 5.

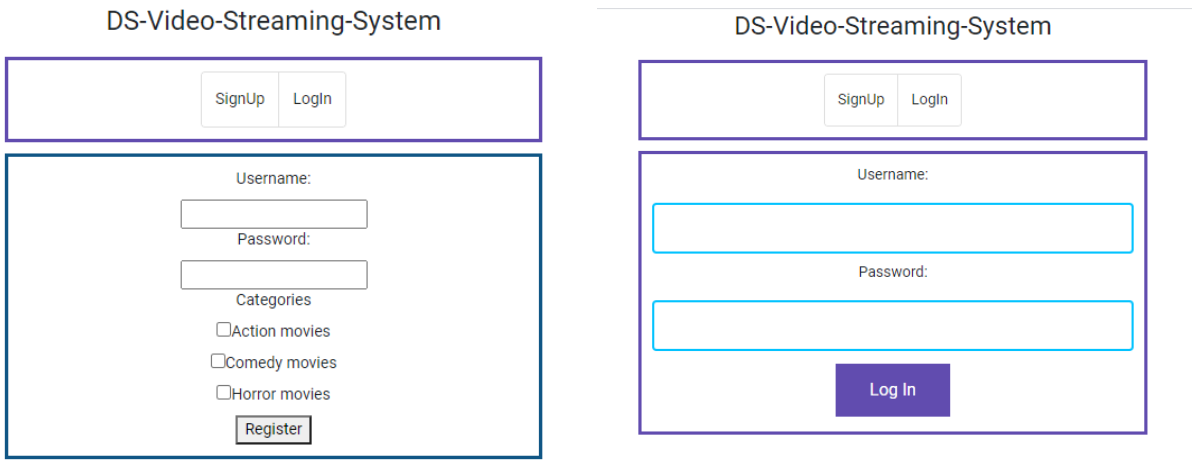

Figure 4: Login interface (left) and Signup interface (right)

The UI for the system was implemented using AngularJS and figure 4 shows the basic user interface of the system. In the login screen it asks the user to input their username and the password and in the signup interface it asks for the username, password, and the choice of user's preferences in the given three categories. Here only three categories are included in the implementation, but more can be added. After the user correctly enter the details a POST request like the one shown in figure 2 will be sent to the Login service. Specifically, the login requests will go the Login end point and the Signup end point will be receiving the signup requests. After the login is completed by the user, they are redirected to the home page where the recommendations are shown as depicted in figure 5.

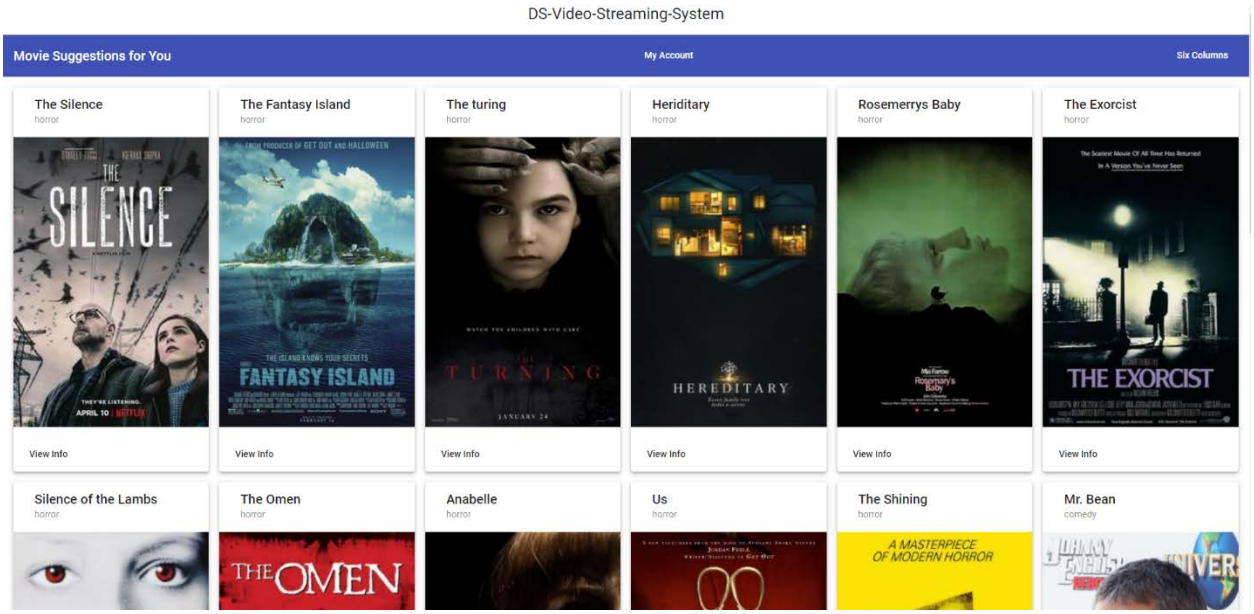

Figure 5: home interface with the movie suggestions based on user's preferred categories.

After that, as depicted in the figure 6 the users are eligible to add reviews, a star rating and see the reviews given by the other users for the same movie as well.

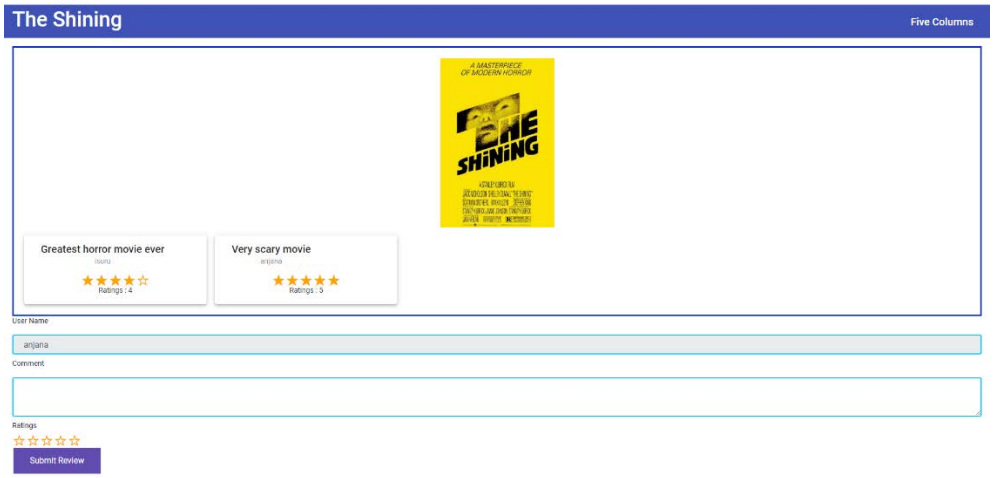

Figure 6: Movie Review Page of the Streaming Service

Deployment

To get the real benefits of the microservices architecture each service can be deployed as individual instances. Here the individual instances are deployed in separate containers (Docker containers with Kubernetes) inside virtual environments. The advantage of this setup is that the services have individually assigned resources and the scalability is very high because individual services can be

scaled up and down according to its traffic. Not only that, but the services can also be built faster using the microservice architecture because the service is an individual component and does not have any dependency on other services.

Challenges

Integration of the load balance was one challenge that we faced. Because databases are independent from each microservice. For one microservice, there is only one database. When multiple instances are created for one microservice, for each of these instances, there are multiple databases. So, if the data in one database is changed, all the related databases must be changed. To achieve this, databases must be synced.

From authentication and authorization, only the authentication part was implemented. Implementation of the authorization was a challenge. Because there are different user levels. Those are admin, users and movie reviewers. Creating the authorization by integrating these different user levels is difficult.

Future Developments

Under the Future Developments, proper Load balances can be integrated. From the Load balancers it will increase the redundancy and availability. Also, a scalable system can be obtained using the Load Balancers. Also, proper User Interfaces can be implemented for better user experience as future development.

In the Login microservice, authorization can be implemented as future developments. For that it is required to have user levels and therefore it is needed to integrate several user levels.

Source Code

https://github.com/AnjanaIshara/DS-Video-Streaming-System.git

References

[1] T. Huston, "What are Microservices? | API Basics | SmartBear", Smartbear.com, 2021. [Online]. Available: https://smartbear.com/solutions/microservices/. [Accessed: 28- May- 2021].

[2] C. Richardson, "What are microservices?", microservices.io, 2021. [Online]. Available: https://microservices.io/. [Accessed: 19- May- 2021].

[3] "Learn Spring Boot Tutorial - javatpoint", www.javatpoint.com, 2021. [Online]. Available: https://www.javatpoint.com/spring-boot-tutorial. [Accessed: 19- May- 2021].

[4] "Netflix Open Source Software Center", Netflix.github.io, 2021. [Online]. Available: https://netflix.github.io/. [Accessed: 23- Jun- 2021].

[5] "Home · Netflix/eureka Wiki", GitHub, 2021. [Online]. Available: https://github.com/Netflix/eureka/wiki. [Accessed: 25- May- 2021].

[6] "Home · Netflix/zuul Wiki", GitHub, 2021. [Online]. Available: https://github.com/Netflix/zuul/wiki. [Accessed: 28- May- 2021].

[7] B. Stopford, "Microservices for a Streaming World", Qcon - 2016, 2016.

[8] "Home · Netflix/ribbon Wiki", GitHub, 2021. [Online]. Available: https://github.com/Netflix/ribbon/wiki. [Accessed: 24- Jun- 2021].