Project - Phase 8 Report

Group 14
Tiago Carvalho fc51034
Diogo Lopes fc51058
Miguel Saldanha fc51072
João Roque fc51080
João Afonso fc51111

16/05/2021

1 Motivation

The idea for our project emerged when we were wondering how cool it would be to have an API that could help us decide which shows to watch next based on our personal taste. We figured that such a service could be developed with relative ease if users marked shows as viewed and/or liked.

Because users can watch more than just movies, for example animes, we wanted to use more than one dataset. Both animes and movies can have a lot in common not only with each other but also with books, so we also decided to use a book dataset. Using data from multiple datasets would give us a more realistic experience when it comes to cloud-native applications development, since these applications use data from so many sources.

With three datasets, we aimed to be able to effortlessly search through any of them. We wanted users to be able to mark movies, animes or books as seen or liked, and get suggestions of what to see next. Since the datasets have very similar categories among them, suggesting books mixed with movies and animes would be a possibility that we thought would add value to our application.

Our idea to implement the suggestion mechanism was to base these suggestions on the user's likes and views, which would indicate to us which categories the user prefers, and, therefore, allow us to suggest good movies, animes or books to the user.

We called our API "Seen", since users can see movies, animes and books and then get suggestions based on their profile, on what they have seen.

2 Dataset characterization

2.1 Dataset 1 — IMDB

This dataset provides a lot of information about movies and shows that can be seen in IMDB.

We downloaded the dataset (updated one year ago) from the Kaggle website.

From the whole dataset, these are the columns that were important to us:

Columns	Example
id	606e2683b3fff1da8a207ae9
name	The Arrival of a Train
category	[Action,Documentary,Short]
rating	7.4
type	short

Table 1: Movie example in our database

2.2 Dataset 2 — MyAnimeList

The second dataset, regarding the MyAnimeList website, was obtained from Kaggle.

This dataset not only has a lot of anime content, but also user information, but because we want to connect with the other datasets it doesn't make sense to use that data. We used the following columns:

Columns	Example
id	606e252aebddc73ebfb15507
name	Shakugan no Shana: Season II
category	[Action,Drama,Fantasy,Romance,School,Supernatural]
rating	7.72
imageUrl	https://myanimelist.cdn-dena.com/images/anime/10/18669.jpg

Table 2: Anime example in our database

2.3 Dataset 3 — GoodReads

At last, this dataset represents books from the GoodReads website, also downloaded from Kaggle.

The columns that are meaningful for us to be able to use this dataset together with the animes and movies datasets are the following:

Columns	Example
id	606e25ad5e927a606f534284
name	Of Mice and Men
description	The compelling story of two outsiders [...]
category	[Classics,Fiction,Academic,School,Literature,Historical]
rating	7.7
imageUrl	https://images.gr-assets.com/books/1511302904l/890.jpg

Table 3: Book example in our database

3 Use cases

We have 3 types of Users: an Admin, which is a logged-in user with special permissions, a Regular user, which is a logged-in user, and a not logged-in user that we call Any.

Services	User	Functionalities
Normal	Any	Sign in See Book, Show and Movie Library
	Regular	User Log in Set Book/Show/Movie as seen Set Book/Show/Movie as liked Ask for suggestions to read and/or watch Count how many views a specific Item has Count how many likes a specific Item has Top 10 Items with more likes
	Admin	Add Book/Show/Movie to Library Remove Book/Show/Movie from Library
Spark	Any	See best Director and his movies with cast See which Actor has the most connections

Table 4: Use cases

4 API

User	Path				get	post	put	del	description
Regular	/lib	/{page}			✕				Returns a *page* from the database
	/suggest					✕			List of suggestions to watch
Admin	/item					✕			Creates an item to add to the database
Any	/item	/{type}	/{id}		✕			✕	Gets/Deletes item with specific *id* and *type*
Regular	/item	/{type}	/{id}	/seen			✕		Marks item as seen
	/item	/{type}	/{id}	/like			✕		Marks item as liked
Any	/item	/{type}	/{id}	/views	✕				Returns Item's number of views
	/item	/{type}	/{id}	/likes	✕				Returns Item's number of likes
	/getTopTen	/{type}			✕				Returns top ten most liked Items with *type*
	/user					✕			Creates User
	/user	/login			✕				Logs in
Regular	/user	/logout			✕				Logs out
	/user	/search	/{username}		✕			✕	Searches/Deletes User by username
Any	/{director}				✕				Returns list with the best Director's movies and his cast
	/actor				✕				Returns the Actor's name with movies with the biggest cast in total

5 Architecture (application and technical)

5.1 Diagram

Figure 1: Project's architecture.

5.2 Application

5.2.1 Client

The Client should be able to access our API on his browser:

$$https://recommendations.sytes.net$$

The Swagger provides a user interface to use and test our calls by adding "/ui" to the end of the url above.

5.2.2 Server

In total there are 7 different microservices working at the same time. Every single one of them runs on Google Cloud, inside the same cluster but in different dockers.

Our reasoning was to have an entrance microservice, which would redirect the request to the microservice responsible for that type of request, for example when sending a request for a page in our library, the API Gateway receives that request and sends it to the Library Service, which is responsible for asking for books, movies and animes to the Book, IMDB and Anime Services, respectively, and then put them together in just one response, which is then sent to the API Gateway, to be shown to the Client.

This API Gateway service also has the responsibility of transforming the REST requests from the Client to gRPC requests that are used internally, between Services.

From the total of 7 microservices, 5 are responsible for the database connection, meaning that they are responsible for translating the request they receive into inserts, updates, removes or queries to the database. They are also responsible for translating responses from the database into responses that can be understood by the other microservices.

5.2.3 Databases

Every database has a service that has the responsibility to access and manage it. While 3 of the databases are hosted by MongoDB, a NoSQL database, the last one is an SQL database hosted on an external server. This last database was

initially hosted on Google Cloud, however we removed it from there, because it was costing us a lot of money.

For the Books', Movies' and Animes' databases, we used a NoSQL database since we might have had to change the format of our documents, meaning that if we had an SQL database we would need to drop the entire database and repopulate it again every time we decided to change the schema. MongoDB provides a very easy and intuitive python implementation to work with, which was also a factor to consider when selecting the type of databases that we would be using.

However, for the Users' database we used an SQL one, because we already knew what we wanted from the User and we knew we would use structured data for it.

5.2.4 Spark

Later in the course of the project, we added the Spark microservice. This service receives the gRPC requests from the API Gateway service and then processes them, creating a job to send to Google Cloud where we have a Cluster with the sole purpose of running this type of jobs.

5.3 Technical

5.3.1 Client

TODO

5.3.2 Server (Microservices)

TODO

5.3.3 Databases

TODO

5.3.4 Spark

TODO

6 Implementation

We actually made 2 different implementations, because before deploying it to the cloud, where we would pay for using the service, we made a local version, that would work locally, by running a script, it's very similar to the cloud implementation, but still has its quirks that are not crucial for the cloud deployment.

6.1   Client

6.2   Server (Microservices)

6.2.1   Docker Compose

For the micro-services deployment we use a docker-compose file, which defines each microservice.

We give them a name, a build stating that we are using a Dockerfile and where the context of this microservice is (its folder, with its required content).

For the image, hostname and container name we use the same name as the docker-compose microservice name we just gave.

At last, we pick which ports we want to use for each microservice, they are crucial for the project to behave properly.

We do the steps stated before for every single microservice inside the docker-compose, but we still have changes to do, since the API Gateway services sends request to the Spark Connector, Account and Library, we add them as environment variables, meaning when running the python file we can fetch the generated IP and then use it to connect with gRPC channels. For the library we add all the services responsible for the databases, which are the Services Book, Imdb, Anime and Account.

6.2.2   Dockers

With the microservices to create a docker image we used dockerfiles, they are really simple, they basically have everything they need in the folder they are in, and just make a new folder called service, and copy the current folder inside the newly created one.

After copying we run the installation of the requirements, using pip and the "requirements.txt" file.

Then the same port, used on the docker-compose file before, it's exposed to be accessible from outside this controller ambient/sandbox.

Finally, we define the entry point as the python file containing all the business of this microservice.

6.2.3   Proto Files