Phase 10 – Final report

**Grupo 2:**

Diogo Lança – fc53495

Meuri Canhanga – fc42559

Miguel Silva – fc58974

Pedro Santos – fc46417

Renato Ribeiro – fc57433

1. Motivation

Steam is one of the most important platforms in the world of PC gaming, boasting over 150 million users, and we (project maintainers) are also included.

Due to our use of Steam, being familiar with the game service and having the chance to develop a cloud native application, we thought it would be cool to offer a service like Steam, where it would be possible to review games. These reviews would help other users by recommending games, based on the reviews written by other users.

1. Dataset characterization

For the project, we selected a subset of a **dataset** of around 21 million **user reviews** in multiple languages of around 300 different games on Steam (video game digital distribution service). This dataset has **1,07 GB** worth of user reviews, **last updated on 25 January of 2021**. Worth noting that the dataset was downloaded from Kaggle[[1]](#footnote-1) in a **CSV** **format**.

1. Architecture

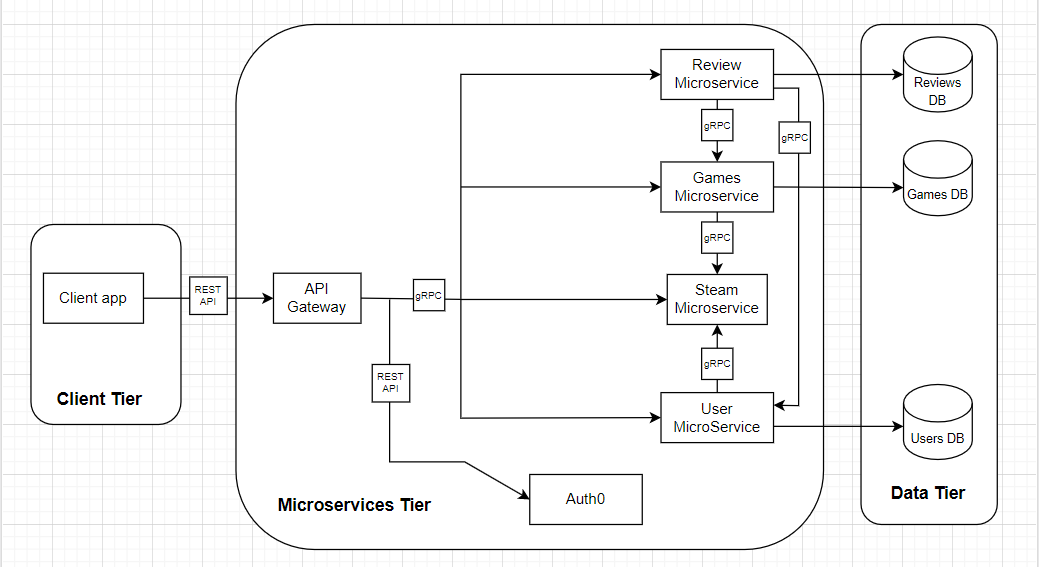


Figure 1 – API architecture

On our architecture the client will use the operations with the Swagger User Interface (UI).

The client will access our project by a public IP address on a browser – for example ‘http://[35.233.14.122/’, where he needs to include ‘ui/](http://35.233.14.122/ui/)’ on the URL, to access the user interface. The IP is obtained on the Google Cloud Platform (GCP) after deployment. On this moment forward the client can “try out” all our implemented operations, which are restricted by roles through the Auth0.

In total we have six microservices working at the same time, but not all run on the google cloud, like - auth0. The main objective or reasoning is the same as Steam, where everything would be around games. We would have a main microservice, where it would show recommended games to anyone (don’t need to authenticate) and by “clicking” on the game would be possible to see the game reviews. Only after the login more operations would be available. For example, if a user logged in and accessed a game, it will not only show the game reviews, but also be possible for that user to review that game, and “upvote” other user’s reviews – like insinuating that review is useful/helpful.

Since our UI is not the typical webpage like [Steam](https://store.steampowered.com/), we adapted the architecture to make sense in the Swagger UI – where lists all operations in the page. Instead of requesting to a main microservice, the gateway communicates with all the other services, with the proper authentication its possible to utilize the respective implemented operations. The API gateway is responsible for transforming the REST requests from the client to gRPC requests that are used internally between services.

Three of the six microservices are responsible for the database connections (all three hosted by [MongoDB](https://www.mongodb.com/)). Since each have its own purpose, the databases are independent of each other. The three microservices translate the requests they receive into inserts, deletes, updates or queries to the respective database and capable of translating responses from the database into responses so that the other microservices can understand.

**Note**: Since leaving a machine deployed uses resources on the GCP, it’s possible that the machine is down, and the IP address won’t work. So, if by any chance the client has access to our git repository and doesn’t have a GCP account with money to utilize googles resources, it’s possible to access it locally – on the bash 🡪 go to the script’s directory 🡪 and run the build\_dockers.sh file (not forgetting to have the docker running). Then on a browser the client can access the project with ‘localhost:5000/ui/’ and “try out” the operations.

1. Dataset source: <https://www.kaggle.com/najzeko/steam-reviews-2021> [↑](#footnote-ref-1)