ERSMS, Group F, Project Documentation

Hubert Dwornik, Michał Łezka, Jakub Mazur
Michał Sar, Krzysztof Rudnicki

June 18, 2024

1 System Architecture

We have designed and implemented 4 microservices, all of microservices are written in Python using Flask framework

AI recommendations Based on a list of movies ids, calculates and returns list of ids recommended for user who likes given movies

Analytics Holds information about number of ratings, number of users, average movie ratings, ratings of given user which are later used in webinterface to show data for admins concerning website usage

Notifications Notifies user whenever there is a new movie recommend by AI recommender for them

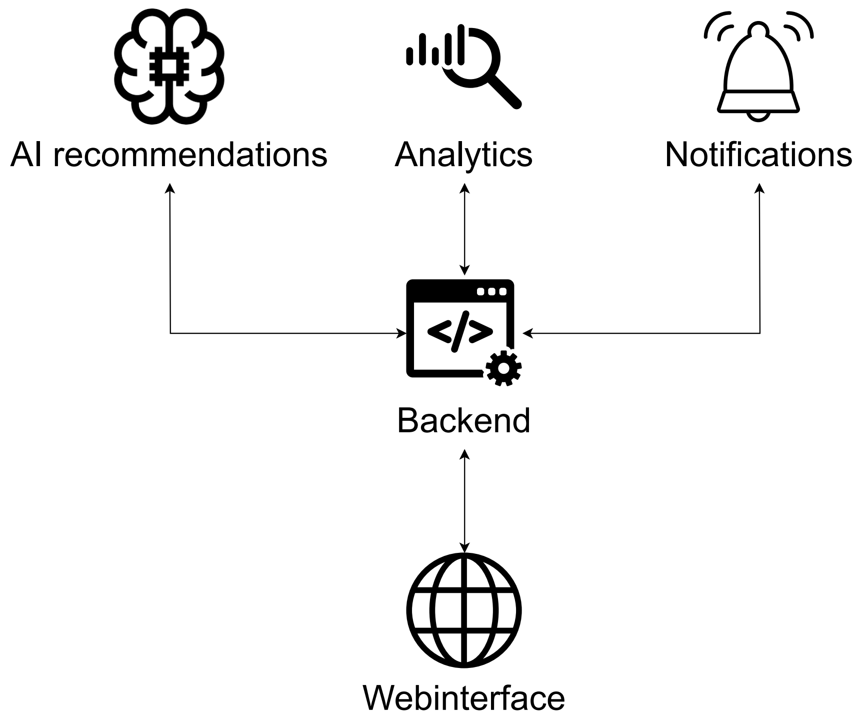
Backend Updates database and mantains all ongoing and incoming communication between all microservices, both between microservies and from microservices to the webinterface

Caching We implemented two caches

1. Backend is cached inside Analytics Service
   Analytics service holds tata about users and movies, in order to not pull all the data from backend every time we update analytics (for example every day), we keep the cache of backend date in our analytics service
2. AI recommendations are cached inside Notification Service
   AI recommendations can change whenever new movie appears, notification service keeps the cache of ai recommendation service in order to not pull all the data from the ai recommendation every time it wants to notify the users about new movies to recommend

Database We use postgresql database to contain data about users, movies and user ratings

Figure 1: System architecture representation, webinterface although not part of microservices included to show relation with backend



2 Automated Infrastructure Management solution

We use **Dockerfiles** for each microservice, webinterface and database which later get combined in docker compose file, after each commit on **GitHub** main repository docker compose gets automatically run on **Google Cloud** platform and deployed

3 Federated authorization and authentication management in the project

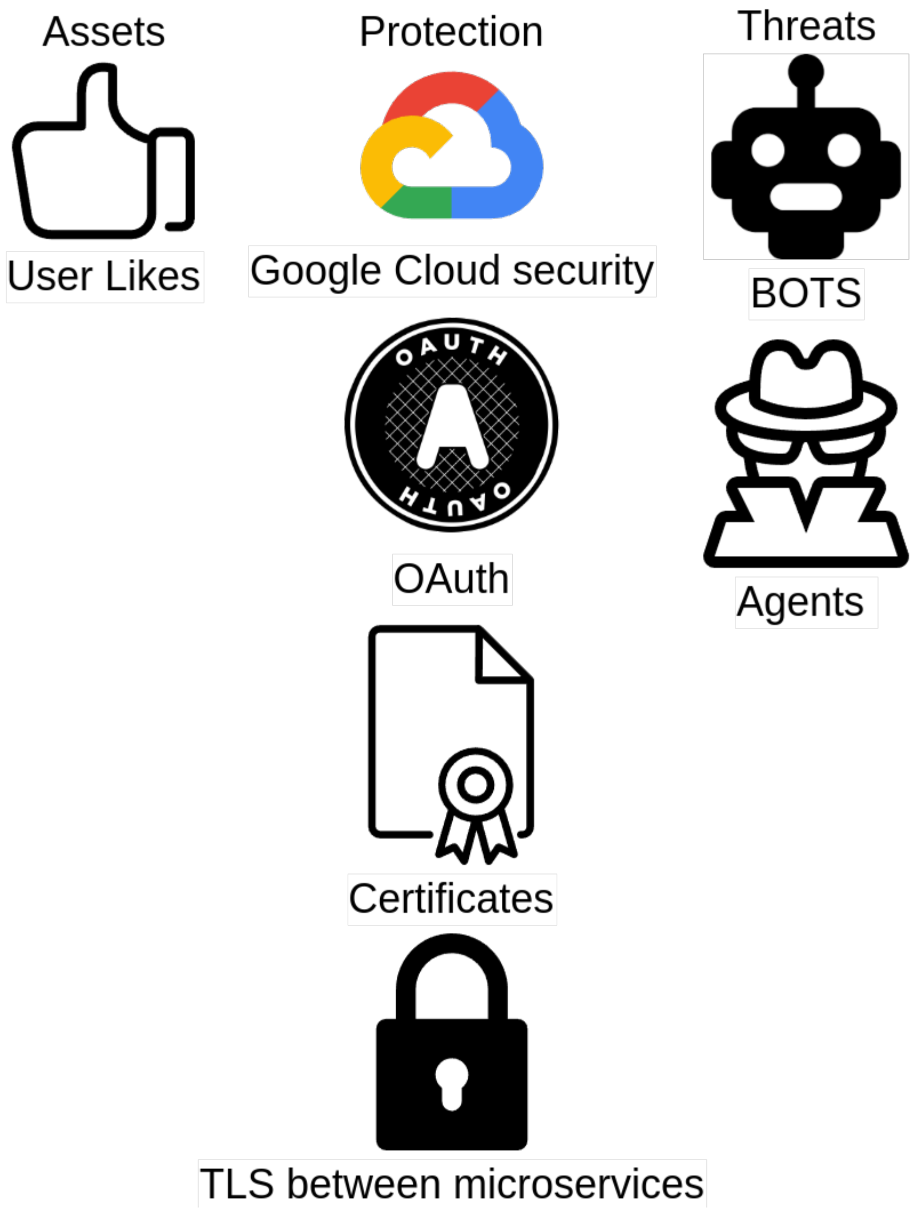
We use industry standard **OAuth** protocol in our webinterface, user creates their account and logs in, we use user token to authorize their access on backend to their ratings and recommendations. We use **firebase** services to manage OAuth protocol.

4 Threat model with mitigations

Our single most important asset are user likes for specific movies
We expect either bots or human agents trying to access those likes for a specific users or to modify user ratings to improve or decrease certain movies ratings
To mitigate that we use:

1. Certificates on our frontend, which encrypt data transmitted between website and user
2. OAuth which is used to authenticate user and lower amount of bots accessing our Infrastructure
3. TLS encryption between our microservices so that even our inside communication is encrypted
4. Google cloud default security policies allowing us to monitor odd and potentially harmfull behaviours

Figure 2: Threat model


Assets
Protection
Threats
User Likes
Google Cloud security
BOTS
OAuth
Agents
Certificates
TLS between microservices