Iuliana Pascotescu

Software Engineering

Music Application – Microservices and Microfrontends

The application describes a music library, containing a set of artists. We can access the songs of every artist and, for each song, we can get its duration from an external api.

Diagram

Description automatically generated

The Songs Microservice is responsible for storing the songs of each artist and it runs on port 8080.

The External Api is called Spotify API and it was used for retrieving the duration of a song.

The Artists Microservice is the server which stores the artists and calls the other two services. It runs on port 8081. It accesses the songs microservice through a feign client called “songsapp”. In that feign client, the two methods that are called are defined. For connecting to the external api, I created an http request, specifying the URL (which contained, as a query parameter, the name of the selected song), the headers that were needed and the method type. As a response, I received a json which was mapped to a song dto and from that dto I extracted the duration of the song.

The first frontend service runs on port 3000 and it opens a login page in which the user enters his username and password, that are then verified on the server. After the login was successful, we are able to see the list of artists that is retrieved from the BE. When clicking an artist’s name, that element is expanded and it shows all the songs of that artist. For each song, we have a button that, when is clicked, we can see the duration of that song. The component that shows the duration comes from the other microfrontend, which runs on port 4000. For making the connection, I used webpack module federation and imported in the main service the component that is defined in the songs microfrontend.

For containerization, I used docker. All the services have a docker image which was made using a Dockerfile and running the command ‘docker build -t name-of-the-image .’. After I had the four images, I created a docker compose file for managing all the services and creating a network for all of them. In the docker compose, I specified the name of each service, together with the images, the ports that I want to use, the memory limit, and using the depends-on property, I could specify the order in which I wanted to deploy my services.