Assignment 2 : MicroServices: Report

Bert Livens

23 mei 2023

1 MicroServices

I decided to decompose the problem in a few microservices, one for the gui, one for all functionality related to accounts and users, one for all functionality related to songs and collections of songs (playlists) and one for everything 'social'.

1.1 Users

The users microservice contains a database with usernames and passwords. It implements the functionality of checking a username-password combo, adding a new user and checking if a user exists. It doesn't call with any other microservice, it only communicates with its database.

1.2 Songs

The songs microservice has a database listing all songs and one listing all playlists. It implements the functionality to get a list of all songs, add a new song, check if a song exists, create a playlist, get all playlists owned by a user, get all songs from a playlist, add a song to a playlist and check if a playlist exists (returns the name if it exists, it is also used to get the name of a playlist from ID). It doesn't call other microservices.

1.3 Social

The social microservice has a database with all friend-combinations, the central feed with all interesting events and a table with all the playlists that are shared with other users and who they are shared with. It implements functionality to post an event to the central feed, get all events from the feed for a specific user, get all friends of a user, add a friend to a user, get all playlists that are shared with a user and share a playlist with an other user. It walls the user-microservice when adding a friend to check if the other user exists, it called songs to get the name of a playlist when sharing it (to add to the feed) and when getting all shared playlists, again to get the name from the playlist ID.

1.4 GUI

The GUI microservice has no database and calls virtually every public function from the other microservices.

2 Rationale

The hardest choice was where playlists would live. They are collections of songs and so naturally belong next to the songs themselves but they are also social because they can be shared and collaboratively edited. I decided to put them in their natural place, together with the songs the contain but their shared, social aspect is completely owned by the social microservice. This has, as an added benefit, that if the social microservice is offline, your own playlists will still be retrieved by the GUI and you can still use them as normal.

3 API

For every microservice, we assume the url pointing to it is known. When an endpoint returns a boolean, it signifies if the action was successful or there was an error in any way.

3.1 GUI

The API of the GUI is unchanged from the assignment.

3.2 Users

- `GET /users/<string:username>/`

  Check if a user exists, returns a boolean.

- `PUT /users/add/`

  Add a new user, give arguments 'username' and 'password', returns a boolean.

- `POST /users/login/`

  Log in, give arguments 'username' and 'password', returns a boolean.

3.3 Social

- `POST /social/feed/<string: username>`

  Get the feed for a user, give argument 'limit' for the last N items in the feed, returns a list[(timestamp, event, friendname)].

- `PUT /social/feed/<string: username>`

  Add an event to the feed, give argument 'event'

- `GET /social/friends/<string: username`

  Get a list of friends for a user, returns a list[username]

- `POST /social/friends/<string: username>`

  Add a new friend, give argument 'friendname', returns a boolean

- `GET /social/shared_playlists/<string: username>`

  Get playlists shared with a user, returns a List[(playlist_id, playlist_name)]

- `POST /social/share_playlists/`

  Share a playlist, give arguments 'username' and 'friendname' (the recipient, doesn't have to be a friend), returns a boolean.

3.4 Songs

- `POST /playlists/`

  Check if a playlist exists, give argument 'playlist_id', returns the name of the playlist.

- `GET /playlists/<int: playlist_id>`

  Returns all songs from a playlist, list[(title, artist)]

- `PUT /playlists/<int: playlist_id>`

  Add a song to a playlist, argument 'title' and 'artist'

- `GET /userplaylists/<string: username>`

  Get all playlists of a user list[(id,name)]

- `POST /userplaylists/<string: username>`

  Create new playlist, argument 'name'