Distributed System Assignment 2: MicroServices

Ferit Murad

May 23, 2023

1 Decomposed MicroServices

The problem is decomposed into 5 microservices. We won't go into detail on the microservices 'gui' and 'songs' since they were provided.
First, we have the microservice 'users' with its database 'users_persistence'. The functionalities in this microservice are register (feature 1) and login (feature 2). Those features are grouped together as they both need the same database. This microservice doesn't communicate with other microservices except for 'gui'. Table 1 shows that this microservice keeps the user's username and password.

Table 1: Users Table

Column	**Data Type**	**Constraints**
username	TEXT NOT NULL	PRIMARY KEY
password	TEXT NOT NULL	

The second microservice is 'friends' with its database 'friends_persistence'. The features adding other users as friends (feature 3) and being able to view a list of all its friends (feature 4) is implemented in this microservice, also the activity of making a friend for feature 9 is implemented here. The microservice communicates with 'users' to check if a user exists when adding a new friend. Feature 3 and feature 4 were grouped together because one of adds information to the database and the other gets information from the database. Because it is straightforward to obtain friend activity from the database, feature 9 (activity of making a friend) was grouped with them. The usernames of friends and the timestamp of creating this friendship are stored in 'friends_persistence' as seen in Table 2.

Table 2: Friends Table

Column	**Data Type**	**Constraints**
username1	TEXT NOT NULL	PRIMARY KEY
username2	TEXT NOT NULL	PRIMARY KEY
created_at	CURRENT_TIMESTAMP	

The last microservice is 'playlist' with the database 'playlist_persistence'. The rest of the features are implemented here: create a playlist (feature 5), add song to a playlist (feature 6), view all songs in a playlist(feature 7), share a playslist with a friend(feature 8), activities except making a friend (feature 9). These features were chosen together because they heavily depend on feature 5. This microservice has to communicate with the microservice 'songs' to check if the song that we want to add to a playlist exists. It is also needed to check if the username that is given exists, so we can create a playlist for the given username. For feature 9 the microservice communicates with 'friends' to get the friends of the user. The data stored in 'playlist_persistence' can be seen in the following tables.

Table 3: Playlists Table

Column	Data Type	Constraints
id	INTEGER	PRIMARY KEY
title	TEXT NOT NULL	
username	TEXT NOT NULL	
created_at	CURRENT_TIMESTAMP	

Table 4: Playlists_Share Table

Column	Data Type	Constraints
playlist_id	INTEGER NOT NULL	FOREIGN KEY (playlists)
shared_by_username	TEXT NOT NULL	
shared_with_username	TEXT NOT NULL	
created_at	CURRENT_TIMESTAMP	

Table 5: Playlists_Song Table

Column	Data Type	Constraints
id	INTEGER	PRIMARY KEY
playlist_id	INTEGER NOT NULL	FOREIGN KEY (playlists)
username	TEXT NOT NULL	
title	TEXT NOT NULL	
artist	TEXT NOT NULL	
created_at	CURRENT_TIMESTAMP	

2 API endpoints

The following endpoints are implemented in the users microservice:

1. `/users/ [GET]`
   This endpoint retrieves a list of usernames and passwords from the `users` table.

2. `/users/exist/ [GET]`
   This endpoint returns a boolean value indicating the existence of the username in the table.
   Query parameters:

   - `username`: The username of the user.

3. `/users/register/ [POST]`
   This endpoint allows user registration by adding a new user to the `users` table. It returns a boolean value indicating the succes of the registration.
   Query parameters:

   - `username`: The username of the user.
   - `password`: The password of the user.

4. `/users/login/ [POST]`
   This endpoint returns true if the given username and password are in the user table and false if they are not.
   Query parameters:

   - `username`: The username of the user.
   - `password`: The password of the user.

The following endpoints are implemented in the friends microservice:

1. `/friends/add_friend/ [POST]`
   This endpoint allows adding a friend connection between two users. If the provided usernames are valid and they are not already friends, a new record is inserted into the `friends` table with the given usernames. It returns a boolean value indicating the success of adding the friend connection.
   Query parameters:

   - `username1`: The username of the first user.
   - `username2`: The username of the second user.

2. `/friends/ [GET]`
   This endpoint returns a list of usernames that are friends of the specified user.
   Query parameters:

   - `username`: The username of the user.

3. `/friends/exist/ [GET]`
   This endpoint returns a boolean value indicating whether the two usernames are friends or not.
   Query parameters:

   - `username1`: The username of the first user.
   - `username2`: The username of the second user.

4. `/friends/activities/ [GET]`
   This endpoint retrieves the activities of a given user's friends from the `friends` table. It returns a list of tuples, each containing the friend's username, the timestamp of the activity, and a description of the activity.
   Query parameters:

   - `username`: The username for which to retrieve the friends activities.

The following endpoints are implemented in the playlists microservice:

1. `/playlists/songs/ [GET]`
   This endpoint returns all the songs for a given playlist_id.
   Query parameters:
   - `playlist_id`: The ID of the playlist.
2. `/playlists/add_song/ [POST]`
   This endpoint allows adding a song to a playlist.
   Query parameters:
   - `playlist_id`: The ID of the playlist.
   - `title`: The title of the song.
   - `artist`: The artist of the song.
   - `username`: The username of the user adding the song.
3. `/playlists/add_playlist/ [POST]`
   This endpoint allows adding a new playlist.
   Query parameters:
   - `title`: The title of the playlist.
   - `username`: The username of the user creating the playlist.
4. `/playlists/share_playlist/ [POST]`
   This endpoint sharing a playlist with another user.
   Query parameters:
   - `playlist_id`: The ID of the playlist to be shared.
   - `username`: The username of the user sharing the playlist.
5. `/playlists/ [GET]`
   This endpoint returns all the playlists of a user.
   Query parameters:
   - `username`: The username of the user.
6. `/playlists/shared/ [GET]`
   This endpoint returns all the playlists shared with a user.
   Query parameters:
   - `username`: The username of the user.
7. `/playlists/activities/ [GET]`
   This endpoint returns the activities related to playlists of a user's friends.
   Query parameters:
   - `username`: The username of the user.