Distributed Systems

Assignment 2 : MicroServices

3 Ba INF
2022-2023

Elias Dams

May 16, 2023

Contents

1 Introduction

In this report I explain my design choices related to microservices. The report will consist of 2 major parts. In the first part I will discuss the decomposed microservices, what features each microservice implements, what data it stores and what connections it has with other microservices. In the second part, I will document the implemented endpoints for each required Feature.

2 Microservices Decomposition

2.1 Decomposition

I was given some services in advance. The **gui** service which is responsible for the User interface and the **songs** service which keeps track of all the songs. To provide the rest of the functionality, I created the following services.

User Management:
User registration and login functionality are critical components of any Web application. So I chose to put this functionality in a separate service. Separating these functions into their own microservice can improve security and scalability. By centralizing user data and authentication, we can better manage user data and implement more advanced authentication methods (should this be needed in the future, as it was not needed for this assignment)

Playlist management:
Playlist management is a core function of the proposed platform, and separating this functionality into its own microservice can help improve performance and maintainability. By separating this from other functionality, we can isolate and optimize the codebase for this particular function, making it easier to scale up and add new features over time.

Activity Feed:
The activity feed is also a very important function of the proposed platform, and so separating this functionality into its own microservice is also very important. This service is basically en event log of all the important interactions that a user takes on the platform.

By breaking down the platform into these microservices, we can create a more modular and scalable architecture that is easier to maintain and update over time. We can also better isolate and manage failures (graceful failure) so that the platform remains available and reliable for users. I implemented this by catching the error when a service is not reachable so that is does not cause the entire application to fail. Every microservice also retries N times when a service is not reachable. After the N times it considers the service as unreachable.

2.2 Features of the services

User Management:

- A user can create a profile (Register).
- The site can verify username & password combinations (Login)
- A user has the ability to add other users as friends.
- The user can view a list of all its friends.

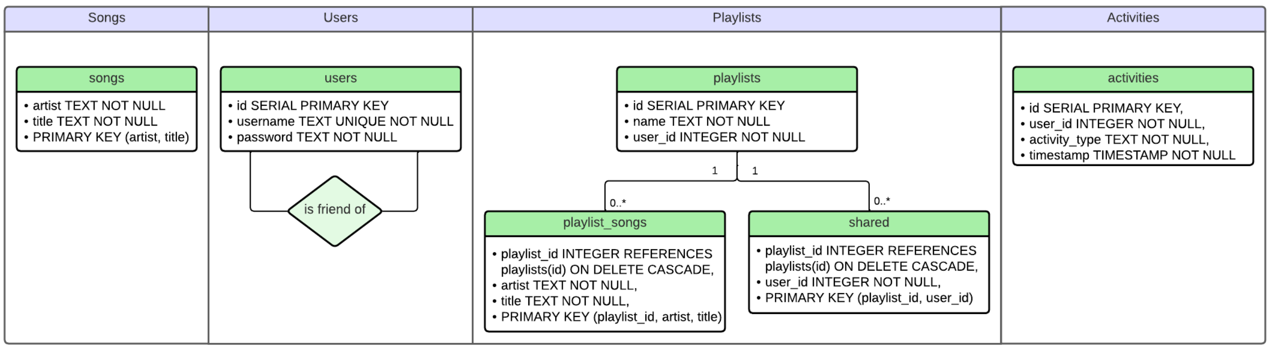
Playlist management:

- The user can create playlists.
- The user can add songs to a playlist.
- The user can view all songs in a playlist.
- The user can share a playlist with another user.

Activity Feed:

- Each user has a feed that lists the last N activities of its friends.

2.3 Database of the services


Songs
Users
Playlists
Activities
songs
• artist TEXT NOT NULL
• title TEXT NOT NULL
• PRIMARY KEY (artist, title)
users
• id SERIAL PRIMARY KEY
• username TEXT UNIQUE NOT NULL
• password TEXT NOT NULL
is friend of
playlists
• id SERIAL PRIMARY KEY
• name TEXT NOT NULL
• user_id INTEGER NOT NULL
1
1
0..*
0..*
playlist_songs
• playlist_id INTEGER REFERENCES playlists(id) ON DELETE CASCADE,
• artist TEXT NOT NULL,
• title TEXT NOT NULL,
• PRIMARY KEY (playlist_id, artist, title)
shared
• playlist_id INTEGER REFERENCES playlists(id) ON DELETE CASCADE,
• user_id INTEGER NOT NULL,
• PRIMARY KEY (playlist_id, user_id)
activities
• id SERIAL PRIMARY KEY,
• user_id INTEGER NOT NULL,
• activity_type TEXT NOT NULL,
• timestamp TIMESTAMP NOT NULL

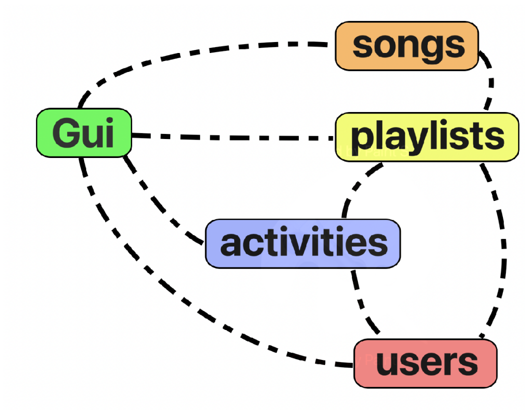
Songs database:
The songs table stores information about songs, specifically the artist and title of each song.

Users database:
The user database stores user management information, while the user table itself stores user data, the friend table will contain information about friendships. In this implantation the friends relation goes both ways. This means that if i am your friend you are also my friend.

Playlist database:
playlists database includes three tables that store information related to playlist management. The playlists table has a one-to-many relationship with both the shared and playlist_songs tables. This means that one playlist can have many shared users and many playlist songs, but each shared user and playlist song can only belong to one playlist.

Activites database:
The activities database includes a single table that stores information related to user activities.

2.4 connections of the services



First we have the GUI Service. This service serves as the user interface and handles user requests. It interacts with all other microservices to gather and present the necessary data to the end users. Second is the Songs Service. This service focuses on managing songs-related functionalities. It communicates with the Playlist Service and GUI Service. It just handles the songs data. Then we have the Playlist Service. This service handles playlist-related operations. It communicates with the Songs Service to check if the song exists, Users Service to check if the user exists, and GUI Service to present the data. Next we have the Users Service. The Users Service manages user-related functionalities. It interacts with the Playlist Service, Activities Service, and GUI Service. finally, we have the Activities Service. The Activities Service focuses on tracking and managing user activities. It communicates with the Users Service when a user makes a friend. It also communicates with the playlists to keeps records of activities, such as creating playlists, adding songs, and other relevant actions.

3 Implemented Endpoints

3.1 User Management

GET	/user/	**Description**: Checks if a user exists.
	Parameters • **username** (string) - Username of the user. • **password** (string) - password of the user.	**Responses** • **200 OK** - User exists. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - User not found

GET	/user/data	**Description**: Retrieves user data.
	Parameters • **user_id** (integer) - ID of the user.	**Responses** • **200 OK** - User data retrieved successfully. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - User does not exist.

GET	/user/friends	**Description:** Retrieves the friends of a user.
	Parameters • **user** (string) - Username of the user. • **by_id** (boolean) - get users by id.	**Responses** • **200 OK** - Friends retrieved successfully. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - User does not exist.

POST	/user/add	**Description:** Adds a new user.
	Parameters • **username** (string) - Username of the user. • **password** (string) - password of the user.	**Responses** • **200 OK** - User added successfully. • **400 Bad Request** - Invalid request parameters.

POST	/user/add_friend	**Description:** Adds a friend to a user's friend list.
	Parameters • **user_1** (string) - Username of the user. • **user_2** (string) - Username of the friend.	**Responses** • **200 OK** - Friend added successfully. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - User or friend does not exist.

3.2 Playlist management

GET	/playlists/	**Description:** Get all playlists
	Parameters • **shared** (boolean) - get shared playlists (else return not created playlitst).	**Responses** • **200 OK** - successfully returned playlists. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - no playlists found. • **503 Service Unavailable** - can't reach 3rd party service.

GET	/playlists/songs	**Description:** Get songs in a playlist
	Parameters • **playlist_id** (integer) - ID of the playlist.	**Responses** • **200 OK** - successfully returned songs. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - playlist not found. • **503 Service Unavailable** - can't reach 3rd party service.

POST	/playlists/create	**Description:** Create a new playlist
	Parameters • **user** (string) - Username of the user. • **title** (string) - title of the playlist.	**Responses** • **200 OK** - successfully created playlist. • **400 Bad Request** - Invalid request parameters. • **503 Service Unavailable** - can't reach 3rd party service.

POST	/playlists/add_song	**Description:** Add a song to a playlist
	Parameters • **title** (string) - title of the song. • **artist** (string) - artist of the song. • **playlist_id** (integer) - id of the playlist.	**Responses** • **200 OK** - successfully added songs. • **400 Bad Request** - Invalid request parameters. • **503 Service Unavailable** - can't reach 3rd party service.

POST	/playlists/share	**Description:** Share a playlist
	Parameters • **user** (string) - Username of the user. • **recipient** (string) - Username of the recipient. • **playlist_id** (integer) - id of the playlist.	**Responses** • **200 OK** - successfully shared playlist. • **400 Bad Request** - Invalid request parameters. • **503 Service Unavailable** - can't reach 3rd party service.

3.3 Activity Feed

GET	/activities	**Description:** Get N activities
	Parameters • **username** (string) - username of the user. • **amount** (integer) - number of activities.	**Responses** • **200 OK** - successfully returned activities. • **400 Bad Request** - Invalid request parameters. • **404 Not Found** - playlist not found. • **503 Service Unavailable** - can't reach 3rd party service.

POST	/activities/add	**Description:** Add an activity
	Parameters • **username** (string) - Username of the user. • **activity** (string) - the activity.	**Responses** • **200 OK** - successfully added the activity. • **400 Bad Request** - Invalid request parameters. • **503 Service Unavailable** - can't reach 3rd party service.