Musicfun Documentation

Group E
Dominik Hirsch, Julian Khin, Felix Knöpfle, Lukas Kowalsky, Heyun Yang & Artur Zelik
February 7, 2023

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1 Motivation

Spotify [1], Apple Music [2], YouTube Music [3] and many other music streaming service providers are world-wide popular and attract millions of users. Inspired by them, we built our own music streaming app in the past six months: Musicfun. Musicfun not only provides with streamed music, but also present social interaction. Users can befriend with each other, send chat messages, as well as share a playlist and listen together. Beyond that, our app also includes lyrics and album covers, which increases the fun of listening to music.

Figure 1.1: Musicfun

2 Technical Overview

We made use of several modern technologies to realize our music streaming. The frontend application is an Android application that is developed natively with the Android SDK. A Node.js application is operating as our backend. It serves an Express, HTTP Live Streaming, and WebSocket server using packages including Express, HLS-server, and Socket.IO. The Express server is responsible to handle HTTP requests such as rendering administrative web pages or user and account management. The features of sending chat messages to friends and listening to the same playlist with friends require a bidirectional connection between the server and Android app for a good user experience. Therefore, we are using a WebSocket connection. For the actual music streaming we use HTTP Live Streaming. Music files are stored in the file system on the server side. Other data is persisted using MongoDB as non-relational database.

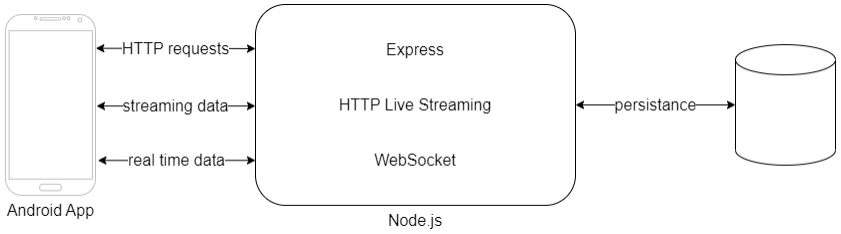


Figure 2.1: Technical concept of Musicfun

2.1 Backend Architecture

2.1.1 Privacy

It is not mandatory for users to create an account but the functionalities without login are very limited to those where no user data has to be persisted, i.e. only listening to music. Unauthorized users can neither create playlists nor use the social system and listen to music together.
Further, we secure our web service in terms of authentication. For this purpose, we make use of the package *passport*. It provides an authentication middleware, which is called when a route is accessed before the actual handler function takes over. Using the package *passport-jwt* allows us to define an authentication schema using JSON Web Token. To access endpoints which are secured by authentication, a query parameter *auth_token=<token>* has to be placed in the URL.

```
router.get('/', passport.authenticate('jwt'),
  async (req, res) => {
    res.sendStatus(204);
  });
```

Figure 2.2: Example endpoint secured with passport authentication

To enable user registration and login by using a username and password, we use the *passport-local* package. Registration and login functionality is configured inside */auth/auth.js*. When a user creates an account, a document is added to the database collection *Users* using the transferred username and password of the request. Inside */data/models/users.js* is a pre hook defined, which is called when a user is saved into the database. This hook makes sure that the password is hashed before it gets stored in the database. To achieve this hashing process, we use the hash function of the package *bcrypt*. A huge benefit of it is that we do not keep track of any salts to be secure against rainbow table attacks, since the hash function will automatically generate such salt.
When a user logs in with his credentials, the server will check if there is a user with matching credentials in the database. If this is the case, the server creates a JSON Web Token using the *jsonwebtoken* package. The payload of the token is composed of the id of the user and his roles. It is important to mention that the secret which signs the token should be replaced by a more secure one when the system is going to production. After the token is successfully created it will be sent back to the client where it now can be used to authenticate to the server.

2.1.2 Database

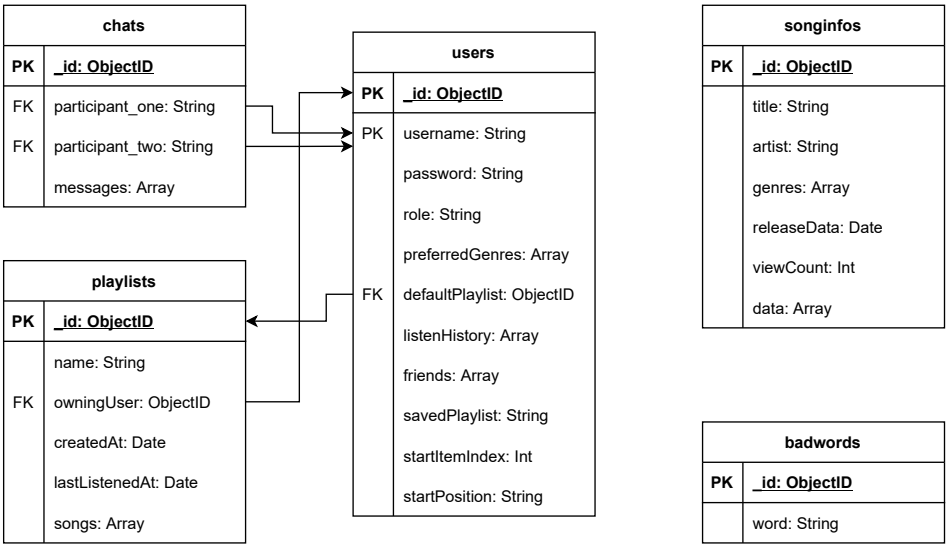


Figure 2.3: Entity relationship diagram of the database tables

To store and manage our data, we use the cloud-hosted database MongoDB Atlas. MongoDB is a non-relational, document-oriented database that manages JSON-based documents. The credentials of our MongoDB Atlas account are: E-Mail: IPTK2022@gmail.com Password: IPTKPassword. MongoDB provides a NodeJS library and can easily be accessed from the backend server. On app startup, the server connects to the cloud instance of MongoDB through the connection URI. The server can then send queries to the database and reliably receives the retrieved data.
In the Musicfun-server repository, you can find our data schemes which determine object attributes, data types, required attributes and so on. The schemes define the way how data is stored in the database.
In the database, there are five collections (MongoDB equivalent to tables) that contain all the necessary data. The collection **users** stores information about users who registered in our app, for example their name, hashed password, a list of their friends etc. The collection **songinfos** persists data of all the songs that can be listened to in our app, for example the song title and artist, to which genres a song belongs and how many times it has been listened to. The song itself is not stored in the database but on the backend server. The collection **badwords** holds data about expressions that we deem not appropriate to use. Based on these words the chat messages are filtered for inappropriate content. In the

collection **chats** each document stores a chat interaction between two users. The chats contain the usernames of both participants and a list of messages between them including the timestamp of each message. When a user clicks on a chat with one of his friends, the previous messages are requested from the database so that the chat history between the users can be viewed. As the name suggests, the **playlists** collection stores playlists. Each playlist contains for example a number of songs, a name and an owner.

2.1.3 Remote Server

Our backend is hosted on a Raspberry Pi 3 Model B. The server can be accessed by the IP https://100.110.104.112:3000/. In order to access the remote server, we use Tailscale as described in the ”remote-access.md” readme file in the Orga repository, That means that the computer running the server as well as all devices that want to connect to the server have to be connected to the Tailscale VPN. If somebody wants to access the server, he requires an invite link. This enables us to control who can make requests to the server. Since this Raspberry Pi model does not have the fastest Internet chip, there chould be limited bandwidth and the user performance is not optimal.

2.1.4 Testing

We provide automated endpoint testing for the most endpoints in the web service in order to prevent unexpected behavior and ensure the correctness.
We chose *Jest* as a testing framework. This is an open-source framework developed and maintained by Meta and well known for its simplicity. In addition to *Jest*, the package *SuperTest* is used. That allows us to test HTTP endpoints on a higher level of abstraction. *SuperTest* provides a method “request” to which we pass our express app as a parameter. Port bindings do not have to be set manually since *SuperTest* manages them to ensure a port will not be used twice. The passed instance of the app is then used to execute HTTP requests. This setup allows us to execute HTTP requests to our server by simply chaining the required request HTTP attributes, for example request method, path, and data, with functions that are provided by the *SuperTest* package.

Another significant package for testing is *mongo-db-memory* server. This package allows us to make use of an in memory mongodb instance rather than a real instance, so that there are no availability issues and it is ensured test are running independent from the result of a previous test.

```
it('test name', async () => {
  const res = await request(app)
   .get('/')
   .send({message: 'Hello'});

  expect(res.statusCode).toBe(204);
});
```

Figure 2.4: Example test using Jest and SuperTest

To be sure that the currently state of the application inside the version control system is valid and the web server can run at any time, we established a continuous integration pipeline that runs with the latest image of node. Our pipeline is composed of the two stages build and test. In the build stage *npm install* is executed. The resulting *node_modules* folder is cached so it can be used by the test stage. After the build stage is finished, the test stage runs the *npm test* script. The artifact of this stage is a coverage report.
We tried to cover as many useful test cases as possible by creating tests for the most endpoints of our web service. Currently our tests cover 68.16% of the entire code base with an ascending trend. The number is provided by making use of the *jest-junit* package. This package allows to create coverage reports in different presentation formats, including the cobertura coverage format that is required to show the coverage in GitLab.

2.1.5 Admin Rights

Users of our app belong to one of two roles: 'User' or 'Admin'. This is defined in the users schema in musicfun-server/data/model/users.js. The role of each user is stored in the users collection in the database.

Admins have more rights than ordinary users. They can login with their username and password at https://100.110.104.112:3000/songs/login and get redirected to a page where they can upload new songs and they are able to delete songs as well. Ordinary users can only listen to songs but not upload or delete them. Additionally, admins can edit the list of bad words. They can add or delete bad words that should not be used in chat communication. After login, this functionality is available to admins at https://100.110.104.112:3000/badwords/view?auth_token=<auth token>. Furthermore, admins have the ability to ban or delete users in case of misconduct.

2.1.6 Recommendation System

Besides filtering for the latest releases, highest view-count, amount of friends that listened a specific song or genres, our app provides song recommendations based on a user's listen history. Therefore when uploading songs to the server, additional parameters will be saved for each song. These parameters are taken from a Spotify dataset provided by Kaggle [**Kaggle**]. The dataset saved on the server is not available anymore , but a similar data set [**KaggleDataset**] can be found here on Kaggle.
The parameters saved in the data array (figure 2.3) for each song are acousticness, danceability, energy, instrumentalness, loudness, speechiness and tempo. These are then used to calculate the average vector for a user's music taste. Afterwards the songs a user hasn't listened yet will be compared to this vector by calculating the cosine distance between the two vectors and a list of songs will be provided to the user starting with the lowest cosine distance between a song and a users music taste.
During the register process, users will be asked about their taste of music. So for a brand new user, the songs are recommended based on the selected genres. If users listen to a song for more than 10 seconds later on, it will be saved as his listen history. It will then be used to analyze his specific music taste and the list of recommended songs will be updated.

2.2 Communication between frontend and backend

2.2.1 HLS Streaming

For the music streaming we used an HLS-server (HTTP-streaming). Therefore when a new song is uploaded to the server, it gets split into segments (.ts format) using fluent-ffmpeg [**ffmpeg**] and an index file (.m3u8 format) is created. The index file includes basic information about the segments.
If now a user wants to listen to a song, the .m3u8 file is downloaded from the main server. The file is then used to stream the song segment by segment from the HLS-server which allows fluent streaming even with a shaky connection. The HLS-server itself works with a request-response system providing the app with the requested segments.

2.2.2 HTTP Request

Most of the frontend-backend connections are based on HTTP requests. Instead of the traditional way of sending requests from the app through HTTPClient, we chose Volley [4] library instead. Volley makes networking for Android apps easier and faster, and handles requests asynchronously. Other advantages are for example, it can automatically schedule network requests and support for request prioritization etc. The data types for requests and responses are either JSONObject or JSONArray so that it is clear for both app and server side which property is sent.
Although we are aware of different kinds of HTTP request methods, we have to use a POST request in some cases where a GET or DELETE request should be put into use. It is because Volley only supports for POST and PUT requests with body parameters so far. If a GET request is sent with body parameters, the server will not receive them and the whole request-response process will not work.

2.2.3 WebSocket

The implementation of the websocket on the app side has the *SocketIOClient* class as its base.

- import java.net.URISyntaxException;
- import io.socket.client.IO;
- import io.socket.client.Socket;

The SocketIOClient class is built over these imports. By means of

mSocket = IO.socket(”https//10.0.2.2:3001”)

a socket object is defined, where at the beginning of the brackets the link to the corresponding address is specified, where the websocket is located (in our case the address of the server) and after the colon the port is specified, which is accessed. The socket object can be accessed via the

public Socket getSocket() {return mSocket;}

method.

In another class in which the websocket is then required, a connection to the specified socket can then be established using the

```
socketIOClient.mSocket.connect();
```

method. The

```
socketIOClient.mSocket.on("new_msg", onNewMessage);
```

method can be used to set listeners to specific events that are triggered by the server. In our example the "new_msg" event, which is then responded to on the app side with the onNewMessage Emitter.Listener. Using the

```
socketIOClient.mSocket.emit("join", channelName);
```

method, events can be triggered on the app side, which are then registered on the server side. In our example the event "join" is triggered, where we also pass a JSONObject called "channelName" to the server.

The

```
socketIOClient.mSocket.disconnect();
```

method can be used to terminate the connection to the websocket.

On the server side, we set up the websocket using socket.io. The server runs on top of the remote HTTPS server, so the websocket messages are secure. The websocket is used to implement the chat system and the shared playlists.

```
var io = require('socket.io')(https_server);
```

We use the Socket.IO functions to define the behavior of the WebSocket. For example: When a user clicks on a chat with another user, the user joins a room so that the room name is the username of the user. That means each user is in a separate room which can be addressed simply with his username. To send a live message, the user emits the message to the WebSocket. This message includes the text he wants to send and the username of the person he is writing to. The WebSocket server can then emit this message to the room where the other user is currently inside by addressing that room by the username. The message is also sent to the database to be able to retrieve it later and show the chat history. This is not implemented with the WebSocket but it's a normal HTTP request.

3 User Manual

3.1 Installation

3.1.1 Remote Server

We have a running server on the Raspberry Pi mentioned in Section 2.1.3. In order to connect your phone with the remote server, you need an invite link posted in our Gitlab Musicfun-Server repository. If the links are all not available any more, please contact us and we will generate a new one.

3.1.2 Localhost

If the remote server is not running as expected, you may setup a localhost. The corresponding code is in Gitlab Musicfun-Server repository in branch **localhostServer**. Please install *Node.js* and type

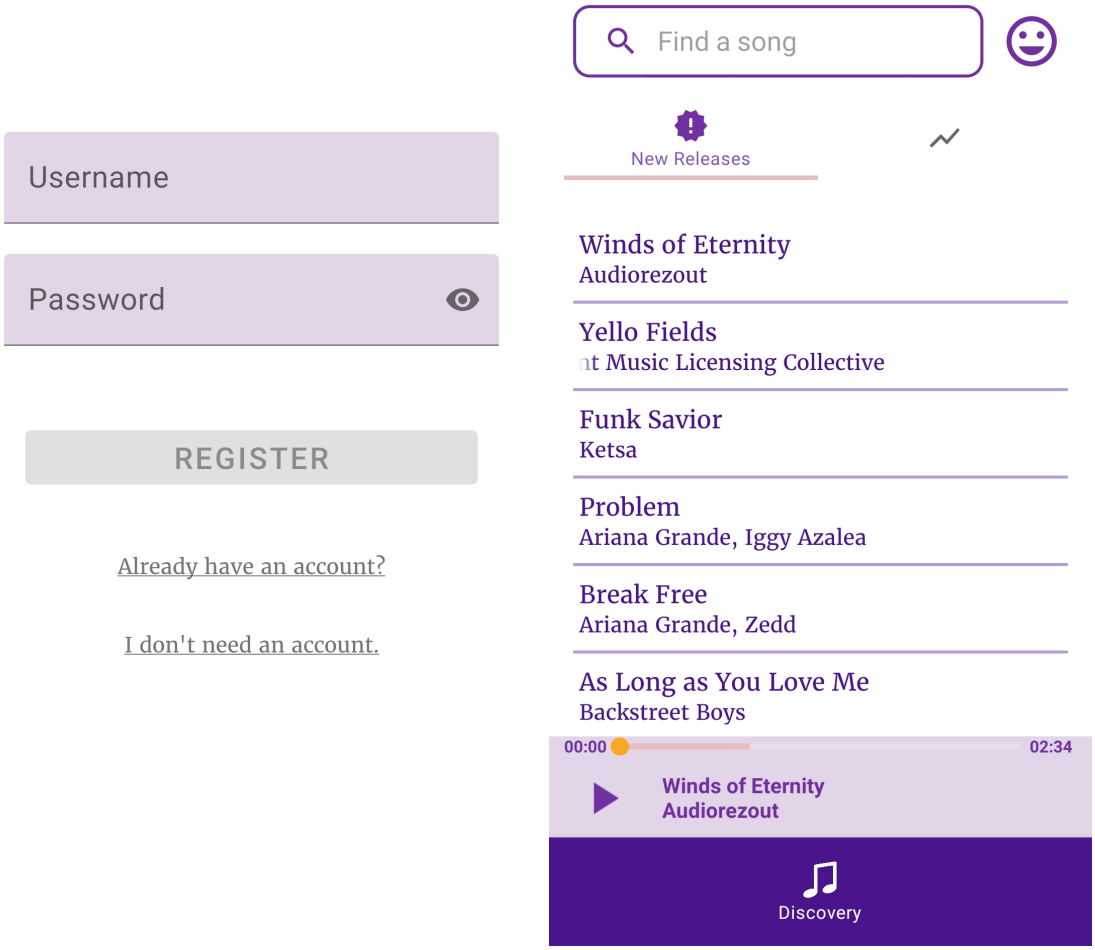
```
npm install
npm start
```

If you choose to run our localhost, the *.apk* file that we included in Gitlab Musicfun repository will not work.

3.1.3 Front-end

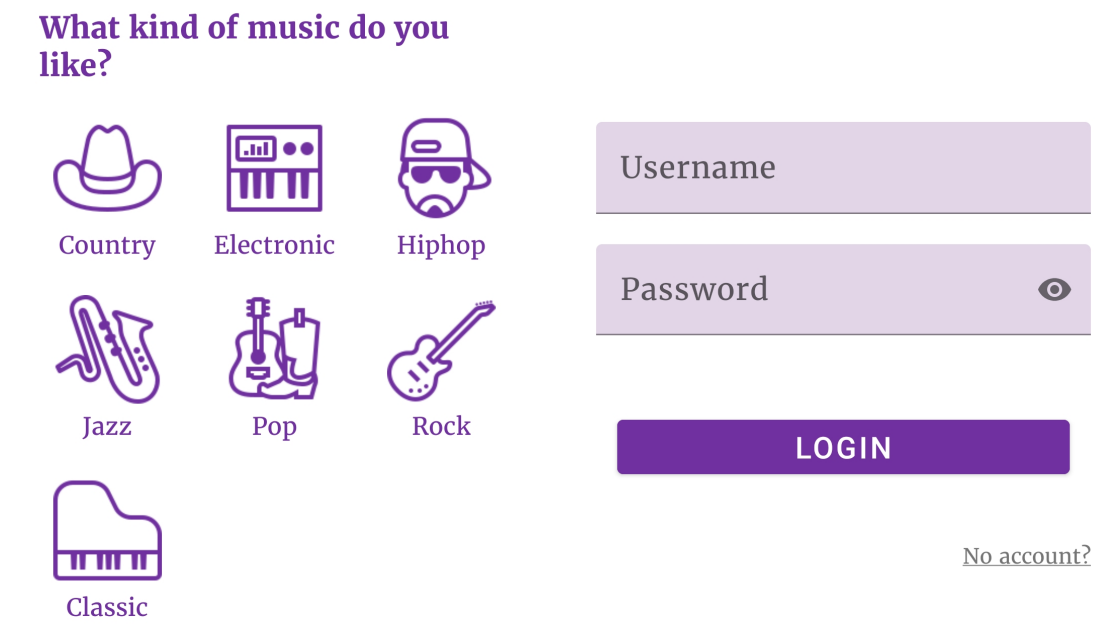
Our Musicfun app is implemented in Android Studio and is available as an *.apk* file which can be downloaded from the **main** branch in Musicfun repository. If the direct installation doesn't work, you can also install the app directly from Android Studio through a USB-connection. Please enable developer options on your device and make sure you have internet connection. Clone the main branch from our Musicfun repository, then click on "Sync Project with Gradle Files" in "File" option menu in Android Studio, so that all front-end relevant libraries can be installed.
If you run the server in localhost as mentioned in the previous sub-section, please pull the code from the **master** branch in Musicfun repository.

3.2 Register, Login and Settings



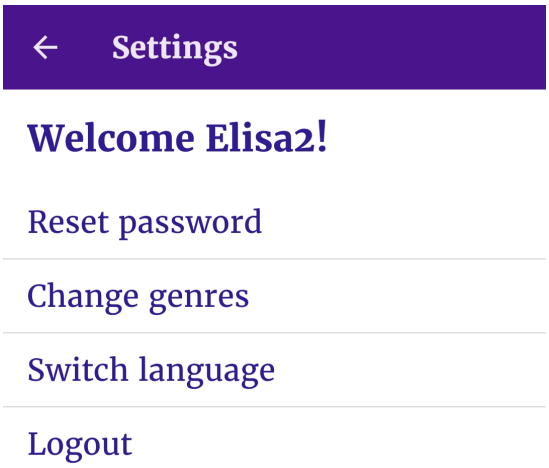
Register: You can choose your username freely, as long as the name does not exist in our database. This name is also used in all functionalities which are relevant to social interactions.

Without login: In order to use the full functionalities of Musicfun, it is recommended to register and log in to our app. Of course, you can still use the app without login, but only the basic feature - music streaming is kept.



Select genres: You will be asked to choose several genres which represent your taste of the music. These are the basic information for song recommendations as described in Section 2.1.6..

Login: You can login if you already have an account.



Settings:
In Settings page, you can change your password. The current password is needed to authenticate your account. You may also change your initial choice of music genres. We currently support English and Simplified Chinese for the app. If you choose to logout, the current music timestamp and playlist will be saved, so that the music can continue playing at the same timestamp after you login again.

3.3 Discovery

In the Discovery section we are trying to support the user finding songs you might want to listen to. We provide you with four different kinds of song recommendations split among the available tabs and present information such as the song title and artists.

- In the “New Releases” tab the latest song releases are listed.

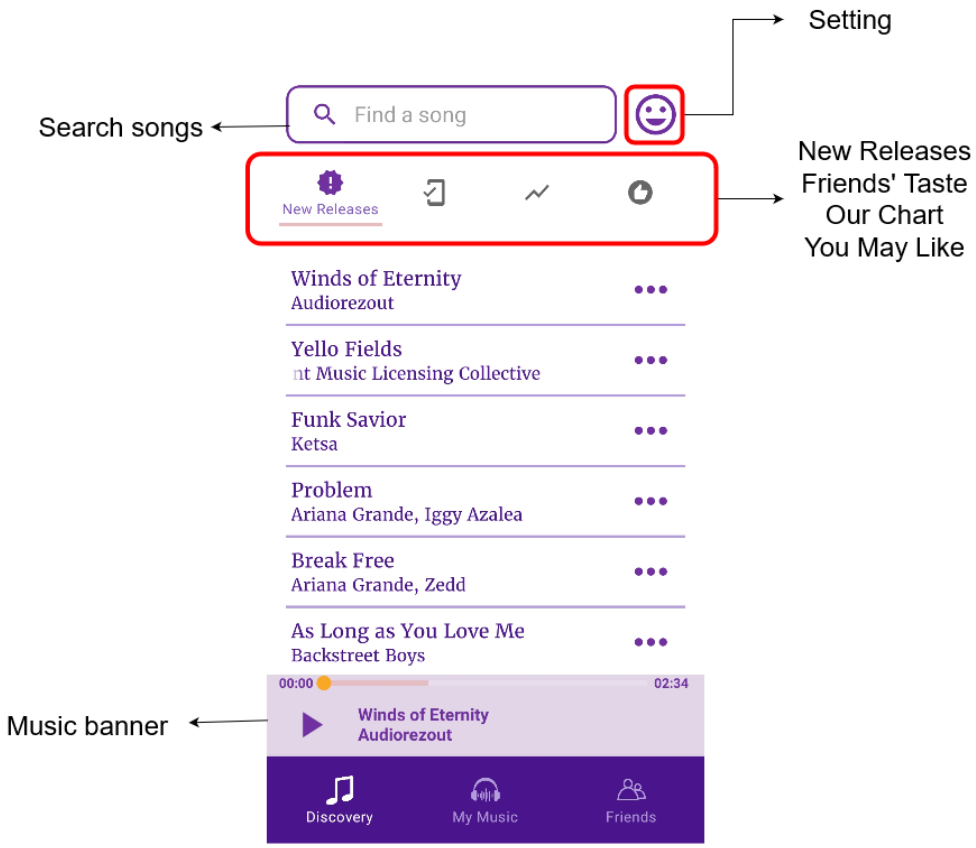


Figure 3.1: Discovery view

- The “Friends’ Taste” gives you an impression of what songs your friends like to hear. As this recommendation is based on a list of friends, this can only be provided if you are logged in and have added at least one friend. The list represents the amount of different friends that have listened to this songs in the near past, hence still saved in their own listen history.
- The “Our Chart” tab is quite similar to the “Friends’ Taste”, but instead of just looking up your friends’ listen history, here we fetch data from all users using our app and share a list of the most heard songs.
- In the “You May Like” tab we provide you with a specific song recommendation. Just

like the “Friends’ Taste” tab, you need to login first. How this list is generated can be found in Section 2.1.6

- If you already knows which song you want to listen to, then just search for a specific song name.

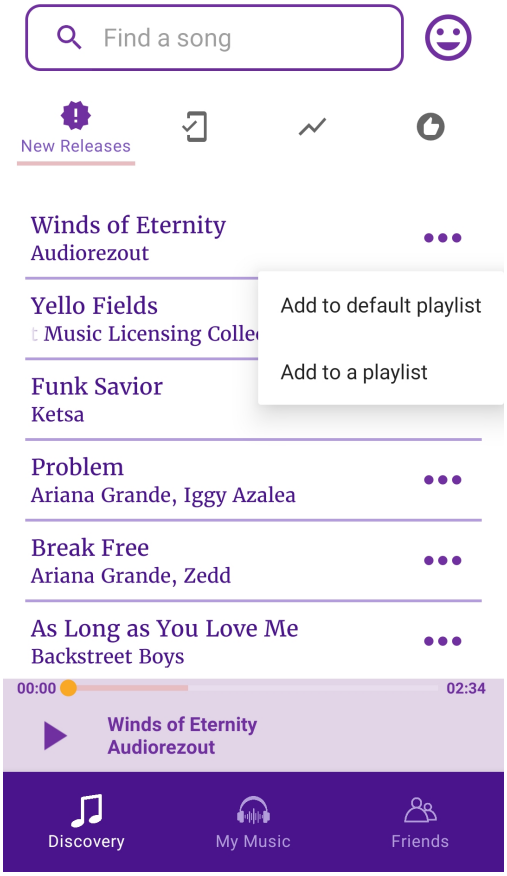


Figure 3.2: Two options in Discovery

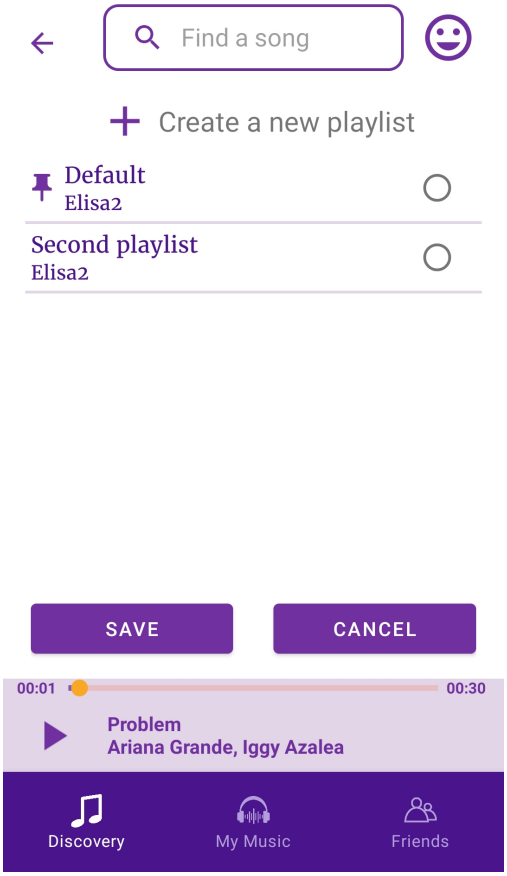


Figure 3.3: Add a song to a playlist

Songs can also be added to any of your playlists from the discovery tabs. We offer two options: add song to default playlist, and add song to a playlist. The second option will lead you to a list of playlists which you own. Since you might have shared one of your playlist with friends but didn’t rename it, you can tell your personal from shared playlists by the user icon on the left. It indicates that this playlist is shared and you are the owner.

3.4 Music Banner

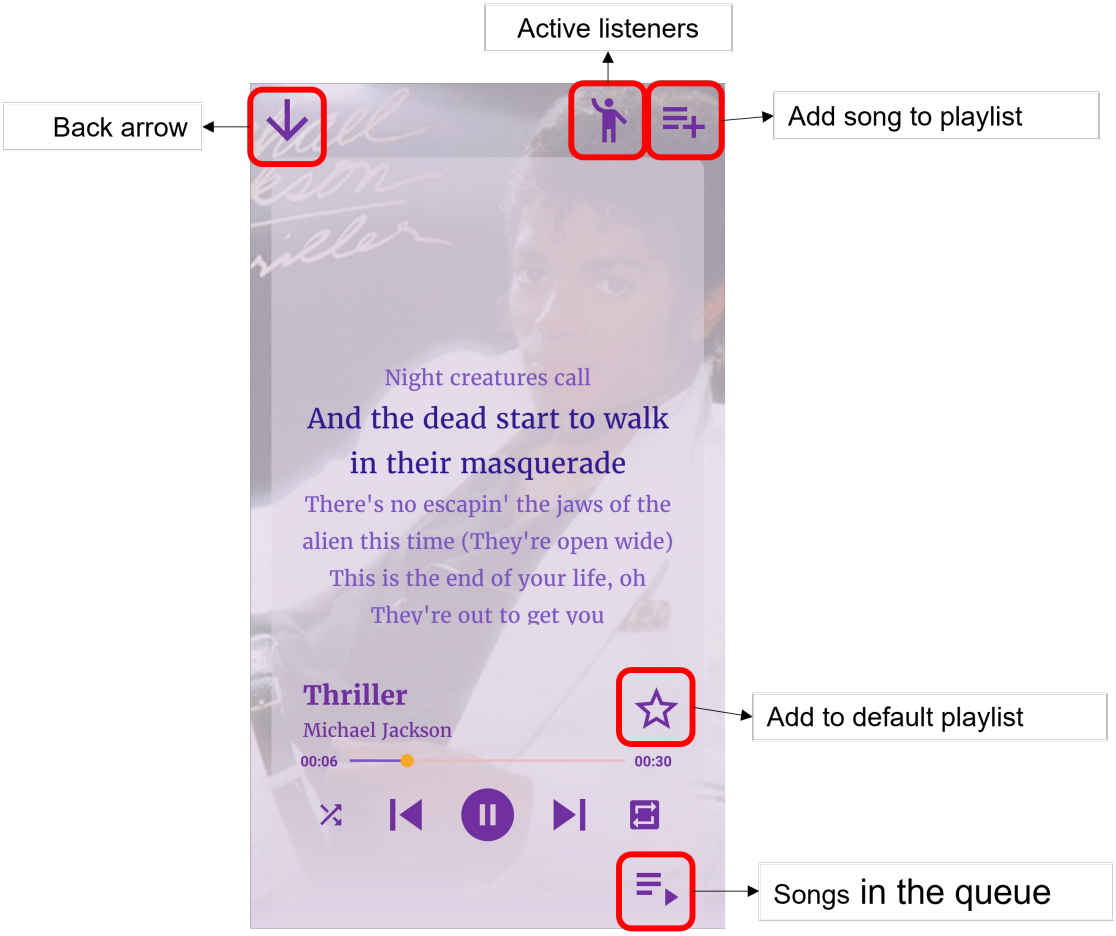


Figure 3.4: Expanded music banner

The music banner in figure 3.1 displays the current song title and the corresponding artist. By pressing on it, the banner can be expanded to a new fragment where more detailed information is shown. Lyrics and album covers will be displayed if the corresponding files are available in our server. The lyrics scrolls along with the music and highlights the current line. We offer five player controls which are Play/Pause, Next, Previous, Shuffle and Repeat. Songs in the playing queue can also be displayed. The star icon serves as a

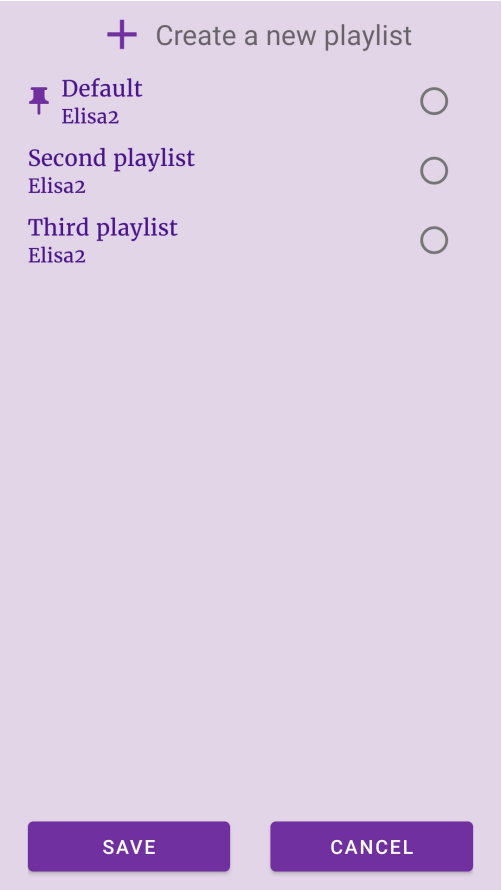


Figure 3.5: Add a song to a playlist

Figure 3.6: Playlist queue

shortcut to add this song to your default playlist. But you can also choose another playlist by pressing the option menu at the top-right corner.
If there is a small figure icon, it indicates you are in a room and listen with other friends. A more detailed explanation can be found in Section 3.6.

3.5 My Music

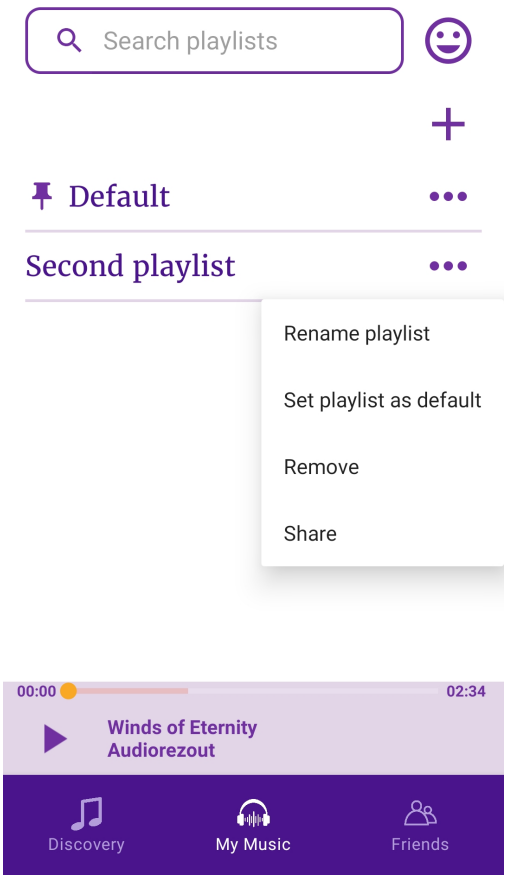


Figure 3.7: A list of playlists

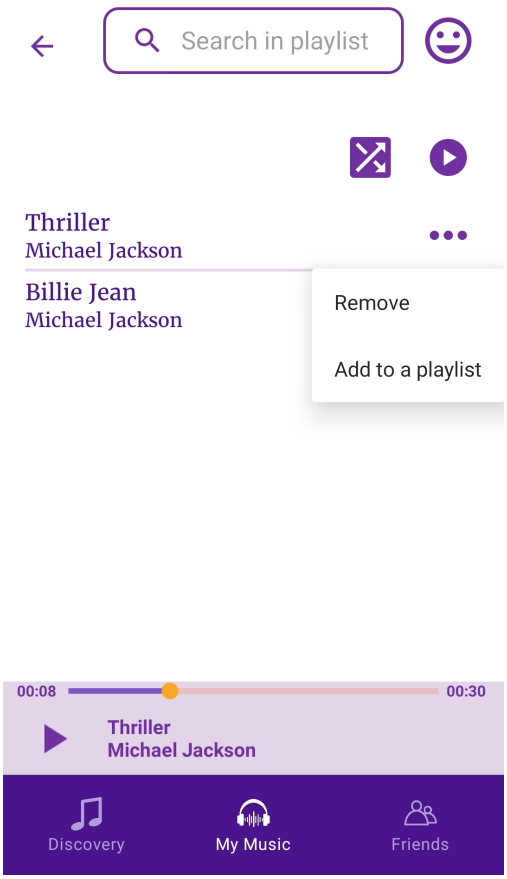


Figure 3.8: Song list in a playlist

Your personal playlists are saved in My Music page. By clicking on the plus icon, you can create a new playlist. The pin icon indicates that this is your default playlist. You can set any of your playlists as default, but it is always unique. For a new account, a default empty playlist is created automatically. By clicking on a star icon, which comes in Section 3.4, this song is added to your default playlist. One of the advantages of using self-defined default playlist together with the star icon is easy categorizing the songs. For example, all songs you want to save in these three months can be added to default playlist named

“Quarter 1” through the star icon shortcut. In the next quarter, just create a new playlist called “Quarter 2” and set it as default.
Playlists are allowed to have identical names, but we do not recommend it for clarity. You can delete a playlist by pressing Remove. Share indicates you create a copy of this playlist to Shared playlist which comes in Section 3.6. If this playlist was already shared, then the content of the shared one will be updated.
Inside a playlist, you can remove a song from it or add a song to another playlist.

3.6 Friends

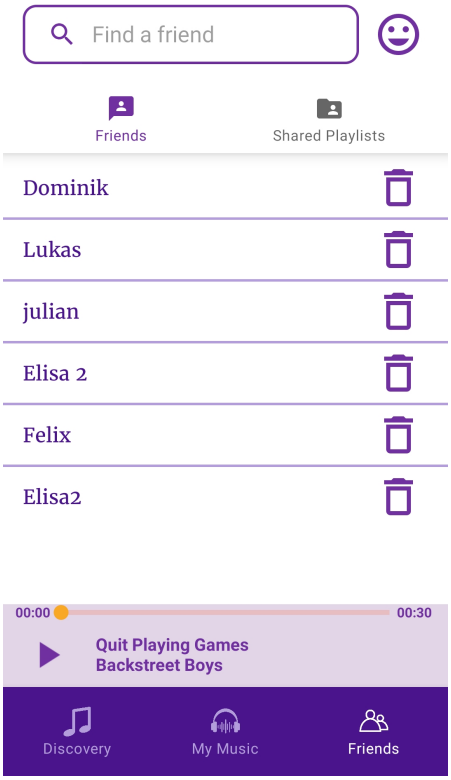


Figure 3.9: Friends

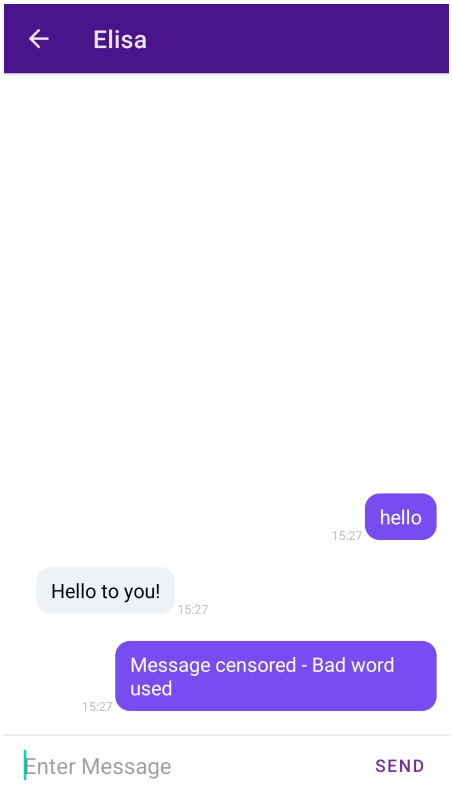
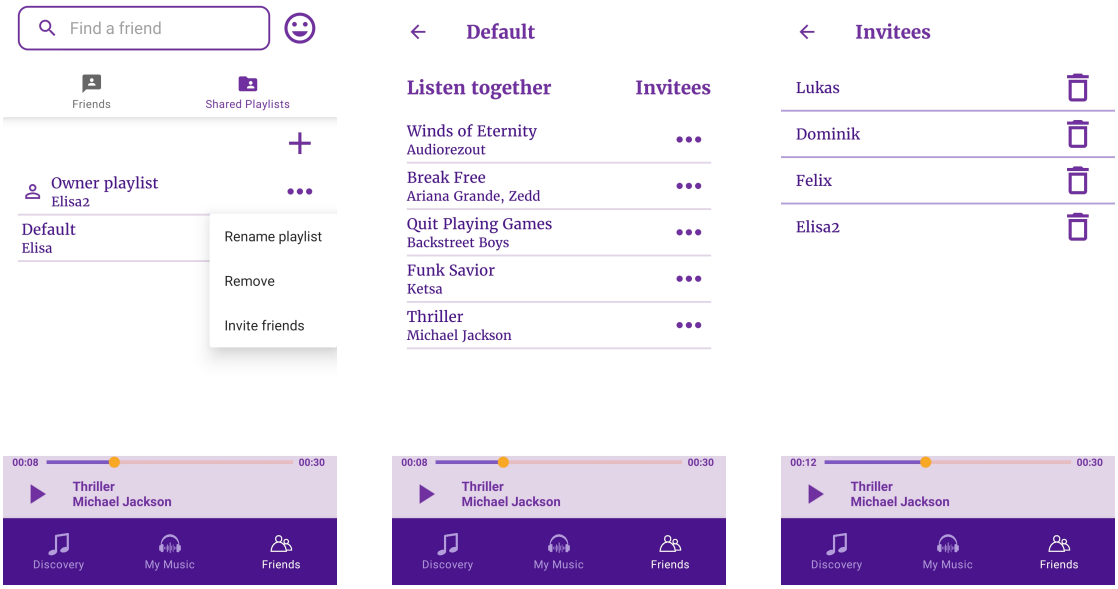


Figure 3.10: Chat

In the Friends page, you can navigate between **Friends** list and **Shared playlists**. Inside the first screen, the entire list of its friends is displayed. With the delete button you can delete a specific friend. If you click on a friend's name, you are able to chat with him. The search view on the top of the screen searches all existed users in our data base. By clicking on a certain friend, the friend request will be sent.



(a) Options for playlists (b) Song list (c) Invitees of this playlist

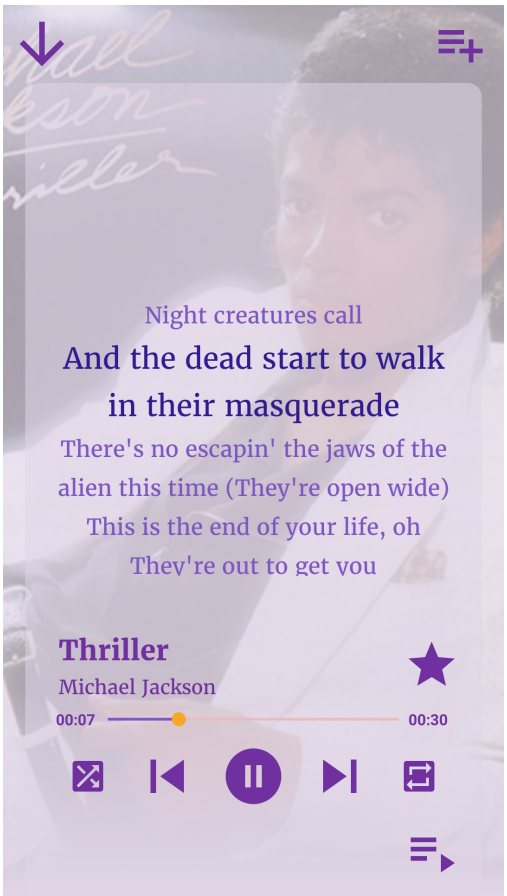
Figure 3.11: Shared playlists

The second view shows the list of shared playlists. Those playlists was either created on your own through the plus icon, or was shared from your own personal playlist or was shared to you by your friends. You can rename a playlist, remove or invite friends to a shared playlist. The third option will open a new window where the list of your friends will be displayed except those who are already the invitees of it.
If you click on a playlist, the songs it contains will be displayed. From this view, you can open the list of invitees or choose the option of listening together which will start a synchronised playback.

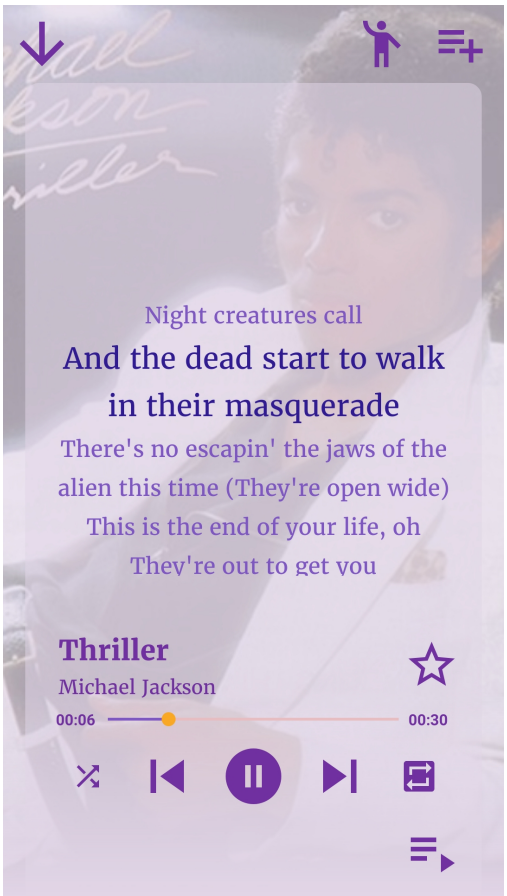
Synchronised playback

After pressing the “Listen Together” button, you will be taken to the expanded music banner view, which presents a cover image for the song, as well as the lyrics, if they are available. You have now joined the synchronized playback. This means that selected actions, such as the Play/Pause, Next/Previous, Repeat/Shuffle as well as scrubbing the progress bar, are applied to all active listeners in this room. The icon of the small man in

the upper right corner indicates you are in a room and leads you to a list which presents all, at this moment, active listeners. To the right of the little man there is an icon consisting of three dots. If the user clicks on this icon, you can add the current song to any of your playlists, including your private playlists and shared playlists that you own. The star icon shows whether this song is your default playlist. It serves as a shortcut to add to or remove from your default playlist. By clicking on the queue icon at the lower right corner, you see a new page with the current song as well as the upcoming songs in the playlist. The arrow at the upper left corner pointing downwards closes the maximized music banner and also ends the synchronized playback, whereby a pop-up appears in the middle of the screen and asks for confirmation before the action is executed.

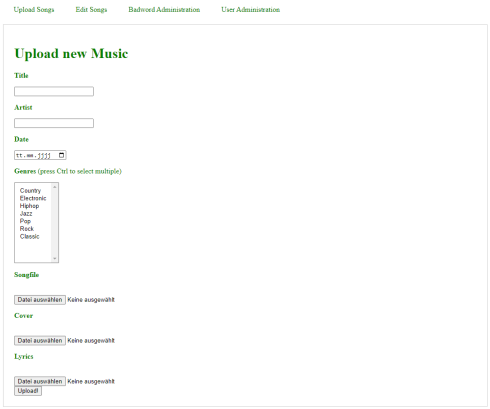


(a) Not in synchronised playback

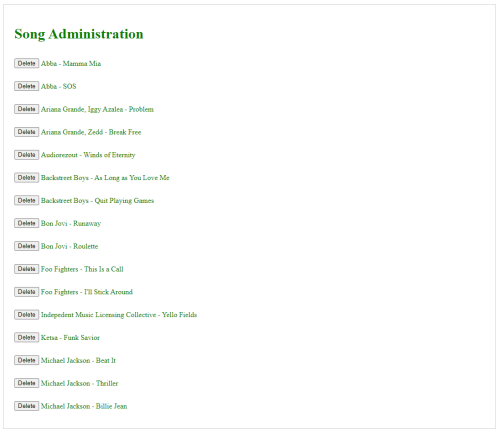


(b) In synchronised playback

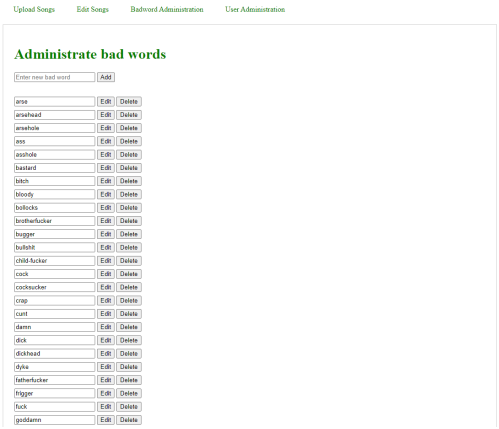
3.7 Admin-Tool



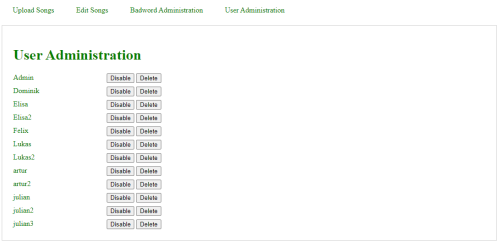
(a) UI for adding songs to the server



(b) UI for deleting songs



(c) UI for adding songs to the server



(d) UI for deleting songs

Figure 3.13: Illustration of the Admin-Tool

If you want to login as an admin and make some changes like uploading a new song, editing the list of bad words or banning a certain user, we provide a special admin access in browser as mentioned in Section 2.1.5. You can login vai https://100.110.104.112:3000/songs/login with an admin account which can be found in the README-file in Musicfun-Server

repository. Other accounts are not authorized to login. After logging in, you have four options. In the first tab, new songs can be uploaded with the lyrics and songcovers not being mandatory in this process. In the second tab, songs can be deleted. The third tab allows you to manage the list of bad words or add new, delete old or edit words. The last tab allows deleting or disabling user accounts which leads to an account being deleted from the database or ban the user from receiving any other data from the server.

4 Feature list

1. Discovery
    - Present the most recently added songs in “New Releases”
    - Present what your friends often hear in“Friends’ Taste”
    - Present the most heard songs from all users in “Our Chart”
    - Present song recommendations according to your genre choices and listen history in “You May Like”
    - Search for any song in our database
2. My Music
    - Create own playlists and add songs to them
    - Set the playlist of as default, which makes it easier to add the songs
    - Rename & delete playlists
    - Share a playlist to make it public
    - Search for a particular playlist & search for a particular song in a playlist

3. Friends
    - Search all users who hasn't appear in the friends list
    - Send out a friend request by clicking on a search result
    - Present a list of all of the users friends (confirmed, requested) in the "Friends"
    - Chat with already confirmed friends in real time
    - Filter the message for bad words and censors the message if bad words occur
    - Present a list of all shared playlists in the "Shared Playlists"
    - Present the owner of a shared playlist and indicates a self-owned shared playlist with an icon
    - Invite friends to a shared playlist
    - Listen to a shared playlist together with invitees
4. Login
    - Register with a username and a password
    - Save login state after closing the app
    - Passwords are hashed and stored on the server
5. Settings
    - Reset password
    - Change selected genres
    - Switch language between English and Chinese without restart the app
    - Logout

6. Music Banner & Expansion
    - Play and Pause in the music banner
    - Expand the music banner or minimize it with button at top left corner
    - Set repeat or shuffle mode
    - Present lyrics & cover picture if available
    - Add the current played song to any of the self-owned playlist
    - Add the current played song to default playlist with the star icon
    - Present the active listeners in a session if in mode “listen together”
    - Present the media items in the queue
7. Admin-Tool
    - Add new songs
    - Delete songs
    - Manage the list of bad words
    - Delete or disable user accounts

5 Conclusion and Outlook

5.1 Conclusion

This project was a milestone for everyone to achieve a better understanding of Android development and to improve our programming skills in general. The scrum system that we used in our project helped us with organising our goals and day-to-day work. Therefore, we could manage to develop all of the mandatory features and add a few new features in addition. Being aware of many issues growing from the live streaming app, our team put the bar very high to make everything smooth and clear. Therefore, we can listen to our favorite songs without any delay and during common parties with our friends we can all open the app and at the same time sing the song with the lyric.
The final result is an application that everyone can use. Our Musicfun brings itself a great user experience along with carefully developed features. Users can have a lot of fun listening to his/her favorite song with other friends at the same time.

5.2 Outlook

Having this state of the application we can improve user experience much more in the future. The karaoke relevant features are still considered as an option, for example recording voice and sending it synchronously to other users, add reverb to the recorded sound so that it is the masterpiece of a user's own, and so on. Moreover, the social features can be improved as well. Inside the chat view things like recording voice message or sending pictures can be added. We can consider buying a license as well. In our application, we use solely free samples of songs. Because of that we can't stream the whole song or sometimes display the proper lyrics for the specific song.

Bibliography

[1] *Spotify*. URL: https://open.spotify.com/.

[2] *Apple Music*. URL: https://www.apple.com/de/apple-music/.

[3] *YouTube Music*. URL: https://music.youtube.com/.

[4] Google. *Volley Library*. URL: https://google.github.io/volley/.