Spotify-Companion

Dominik Scheffler
Philipp Steib

January 17, 2021

Contents

1 Overwiev

1.1 Project Intention

The Project revolves around the automation of Playlist Modification via the Spotify-AppRemoteAPI. The general idea is the automatic removal of commonly skipped songs and addition of those the User appears to favor.
The main objective is making the process of creating playlists less tedious and possibly improve some other aspects, mainly quality of life additions.

1.2 Current State

No possibility to create and add to existing playlists automatically nor to remove frequently avoided tracks. The official app often has loading times when selecting song-options such as queuing and adding to a list. This further inconveniences the use and makes management quite strenuous.

1.3 Desired State

The user barely ever has to adjust the lists manually after creation, since the companion manages removal and addition of titles to lists according to certain rules.

1.4 Similar Apps

"Companion For Spotify" exists but is more focused on presenting changes to playlists and newly released titles. It does allow for concatenation of playlists, this is a feature we cloud add for convenience but did not include in the original mockup.

2 Requirements

2.1 Must Have

- (M3) comprehensible UI
- (M1) removal of repeatedly skipped songs
- (M2) adding not skipped songs to chosen playlist

2.2 Should Have

- (S4) intuitive and clean UI conforming to common usability standards found in most prevailing apps.

2.3 Could Have

- (C5) custom app logo
- (C6) better car-mode-player
  - (C7) interface with track-playtime-slider
  - (C8) improve suggested lists (list of favorites)
  - (C9) add buttons for specific songs and lists
  - (C10) show next title

2.4 Won't Have

- (W11) standalone player, the app is meant to accompany and control the official application

3 Non-functional requirements

This section lists the non-functional requirements. These describe non-technical requirements as well as physically intangible requirements.

- Usability:
  The application must ensure that the controls can be used quickly and intuitively.
- Error avoidance
  The application must ensure that accidental error made by the user can be made undone and, if possible, can be avoided.
- Clarity
  Each view within the application must not be overloaded with too much information.
- App design guidelines
  The app must comply with the design guidelines of the marketplaces.In our case the "Play Store (Google)". Furthermore it must comply with the branding guidelines of Spotify (Found here: Branding guidelines)
- Compatibility
  The app must be compatible to the API, rules and guidelines of the Spotify music platform.
- Authenticity
  The app must ensure that all actions are bound to the authenticated user profile.

4 Architecture

4.1 Structure

This chapter will focus on the java-implementation, it's classes with some relevant attributes and the packages reside in. Here we will also explain some of the choices and considerations we had to make in the development stages.

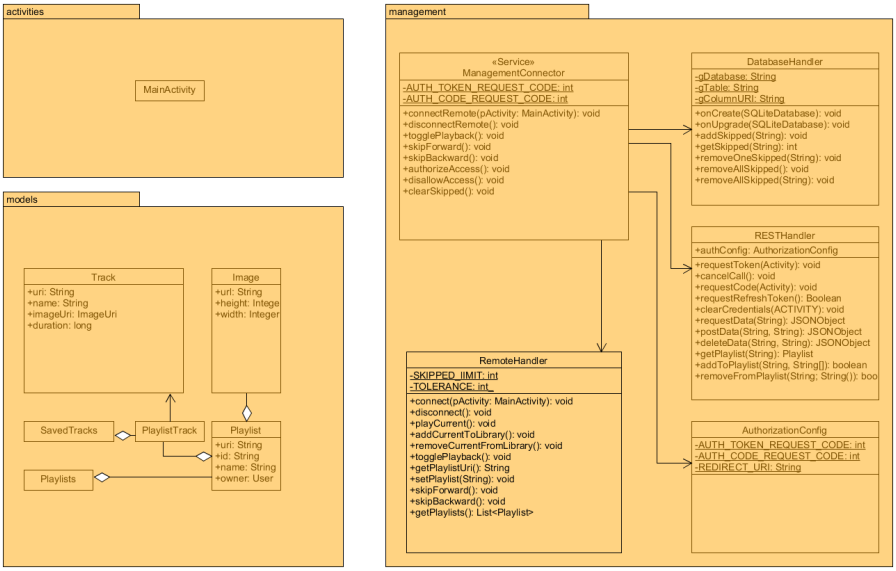

Figure 1: UML-Diagram - Overview of packages and classes with constituents

Our source code is distributed in three packages:
activities, management and models.

Activities only has one class, MainActivity, since the application has the intention of being a simple companion with everything important in view everything can be setup and managed from there. The settings are placed in the drawer, in there are options for selecting the playlists to play from and save in as well as the settings for what actions to perform.

Management holds the Service that runs in the background while the application is not open and in focus. It is executed from the ManagementConnector which also serves as the interface from our logic classes that perform actions and handle everything in the background, to the view classes (MainActivity in this case).

Models is a helper package and is only used for the serialization and de-serialization of JSONObjects for the communication to the WEB Api spotify provides. The instances of these classes are used by the management package only and they serve the sole purpose of holding and transmitting data.

Notable is that we accounted for future additions to the interface, may the opportunity arise. While the user-interface always had been planned and designed with simplicity and usability in mind, there are many opportunities to extend it. During the development and user testing this has been constantly verified. Possible extensions of the current app are

- a better car mode
  - quick-action for immediate song playback
  - selection from lists based on listening history
  - offering a list similar to the radio function
- lists created via recommendations
- quick-action interface for selected tracks or lists
- and more settings
  - adjustment for the skipped-limit
  - removal from queues if a song is in a list or registered as skipped
  - settings for adding to queues from similar or recommended songs

4.2 Service

Our service is started upon opening our application, spotify also provides sample code for registering a receiver with an intent-filter to automatically start an app. We decided against implementing this feature, since it should always be a conscious action to give up the control over the user data and let an automated process manipulate it.

The way we went is implementing a foreground type service, therefore it only works with an unlocked phone for more then a minute.
Considering the other available types of services an Alarm-Manager may have been the way to go instead.
We only found out the service did not work for more then a minute in lock-screen after we had a colleague, who was interested int the app and saw some potential, test on a real phone instead of the Android Studio inbuilt AVR. The VMs on the AVR never automatically locked themselves while testing, hence the assumption that is works indefinitely after exiting the application it self.

In the Activity we also setup a periodic timer with the task of updating the progress bar displaying the current position in the track. And since we have to calculate the system time the song is supposed to end at for detecting when a user has decided to skip a title we also already have calculated the next wake time. This is due to the fact, that neither of the libraries of registration of a listener for a skip-event, only state changes in general.

The issue may also be resolved by implementing the reveiver, we were considering this. Now that the app has reached a certain point and has been tested we no longer have to assume the possibility of modifying or deleting data unjustly due to a unforeseen event. If we continue this project this may be one of the next steps.

4.3 App Remote

For the implementation we will be using the "Spotify Andorid SDK" and the "Spotify App Remote Library".
The Companion will run alongside the official application, all actions regarding connection, audio playback, resource fetching and database updates will still be handled as they have been previously.
We only manage the requests for Spotify to perform certain actions.

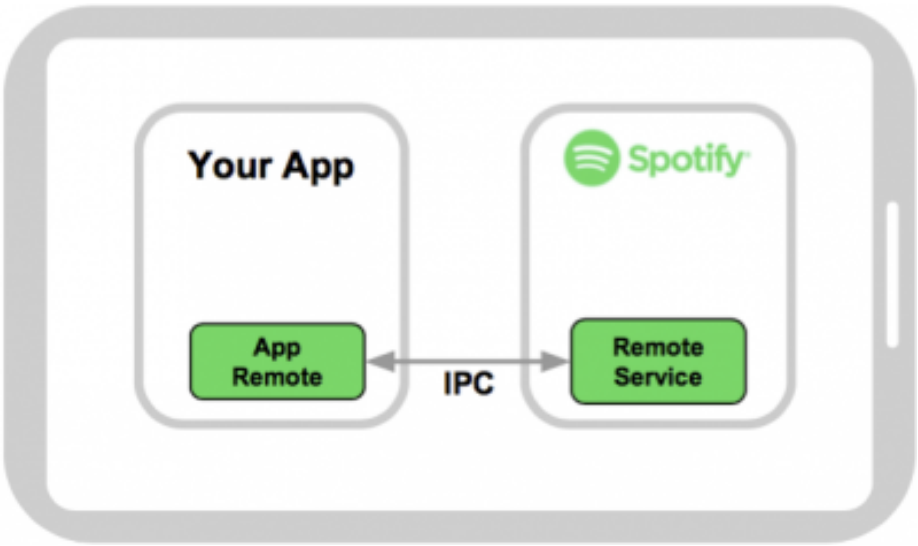

Figure 2: Spotify App Remote Library [4]

The image above displays how the "App Remote Library" is intended to be used. Our architecture will be based upon that.
The App Remote allows for retrieval of intel such as the currently playing title and the selected list it originates from. Also possible is tracking and modifying the state of playback (play/pause), song duration and the current position, playback speed, shuffle-mode, repeat-modes, adding and removing from the users library/liked tracks.
Any information regarding the user, lists and their contents or the queue and next tracks is not accessible via this library.
Most of the afore mentioned are however retrievable from the WEB Api, we initially did not plan to utilise those capabilities but decided to do so since adding or modifying playlist data is not possible with only the App Remote Api.

4.4 WEB API

After analyzing the scope of the Spotify Android SDK[5], we came to the conclusion that it was inevitable to include the Web API in this project.
The Android SDK simply doesn't provide some of the functionality our app needs, which we now get from including the much more comprehensive Web API in our project. This includes loading any playlists and songs, personal data and saved tracks, as well as manipulating playlist data and saved tracks.

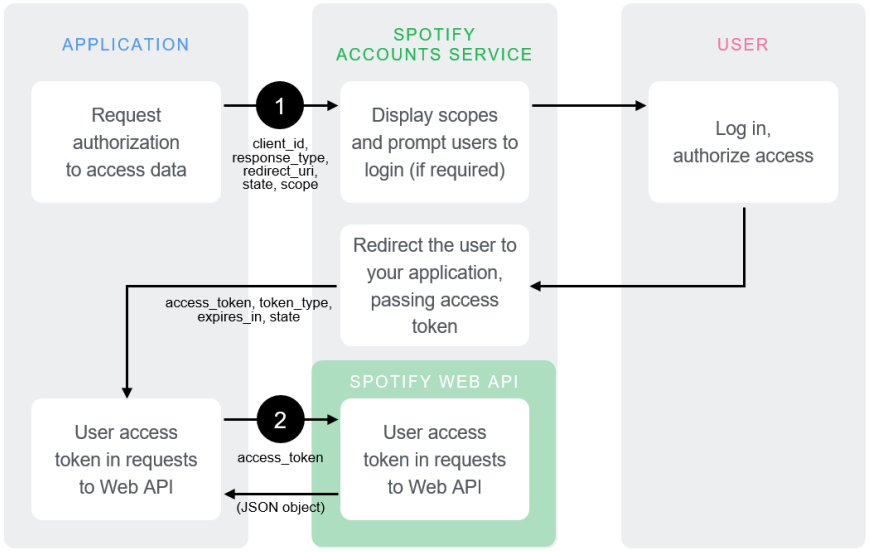

Figure 3: Spotify Communication Auth Flow [2]

We have implemented an interface that communicates with the Web API and accordingly returns the requested records in JSON format. The structure of these records can be found in the official documentation of the Spotify Web API. The records need to be parsed in a readable format for our app to use the containing data.

For this reason, a data model has been designed that closely mirrors that of the rendered data of the Web API. The data model includes models for tracks, playlists and favorite songs (saved tracks), as well as wrapper elements such as the playlist tracks (containing information about the track that is in the respective playlist), images (containing frame information about the image) and playlists (additional information about the user's collection of playlists).

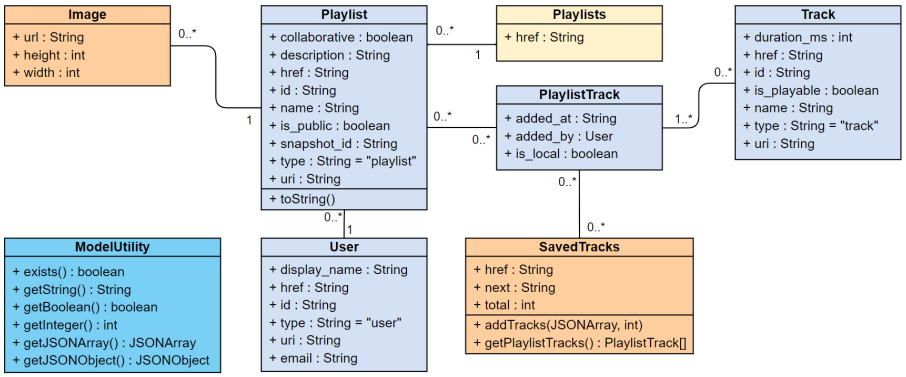

Figure 4: API Data Model Classes [7]

At one point a change had to be made in the data model to ensure compatibility with our software. The returned records of the user's favorite songs are represented by the Web API in a different format than that of the common playlist objects. In order to be able to handle both playlists and favorite songs in the same way, we implemented a conversion from favorite songs to playlist object. On a side note it is to mention that we could not implement the desired Authorization Code Flow with Proof Key for Code Exchange (PKCE), as seen in the documentation for the Web API. As we decided to user the Android SDK as a main interface between Spotify and our application, we used mentioned SDKs functionality to perform the authorization process, not knowing that the developers of that SDK did not and will not add the exchange of a refresh access key, which exists to improve continuous accessibility, for unknown reasons. For that reason we are restarting the authorization process instead of refreshing the access key in case of an authorization timeout.

4.5 Database

For this porject we decided to use a local, simple lightweight database, SQLite3. It is the standard for this kind of usage to save some data internally, and also is supported by android natively. To access it we also used the SQLiteOpenHelper[8] library.
The database has been kept rather simple for this project. It is entirely possible to improve upon this aspect and add several new features.
We decided to keep it as simple as possible. One table with two columns:
an auto-increment integer as id and a simple text-string for the track's url.

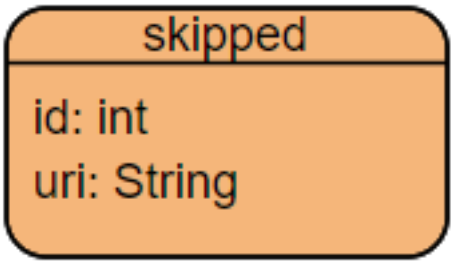

Figure 5: The current state of the database

The other possibility was to save a score for every track in a table referencing the playlist the data it accounts for. Meaning one table per playlist.

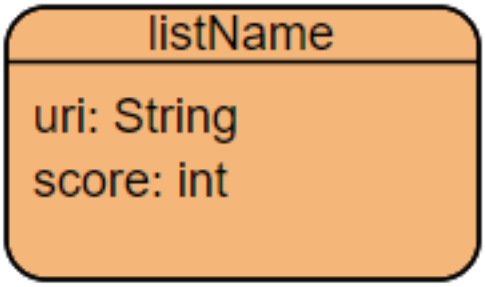

Figure 6: One possible, more sophisticated, solution

This way would provide way more possibilities for saving data independently for each playlist. At the point in time our schema was created and implemented this was sadly not an option, because the data was previously not accessible to us.

5 GUI Prototyping

A clickable prototype has been developed to combine a general mockup with a more specific design- and color guide in mind. This accelerates development cycles with less reiterations needed to match the developed design guide.

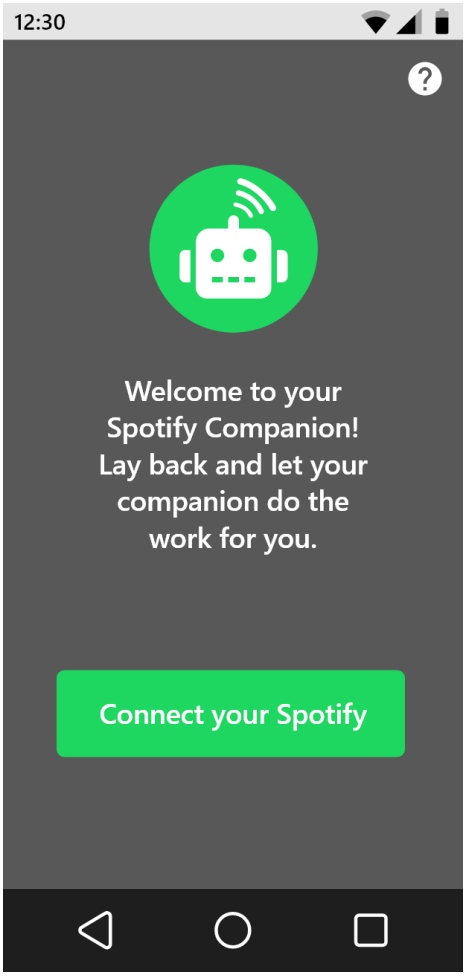

Figure 7: First Welcome Screen

The image above displays how user is greeted and has the option to launch the companion and connect it to an active Spotify-App.

5.1 Collector Mode

The Collector Mode is used when the user browses foreign playlist or mixes. If the user listens a track for a set time, the track will be added to a set playlist.

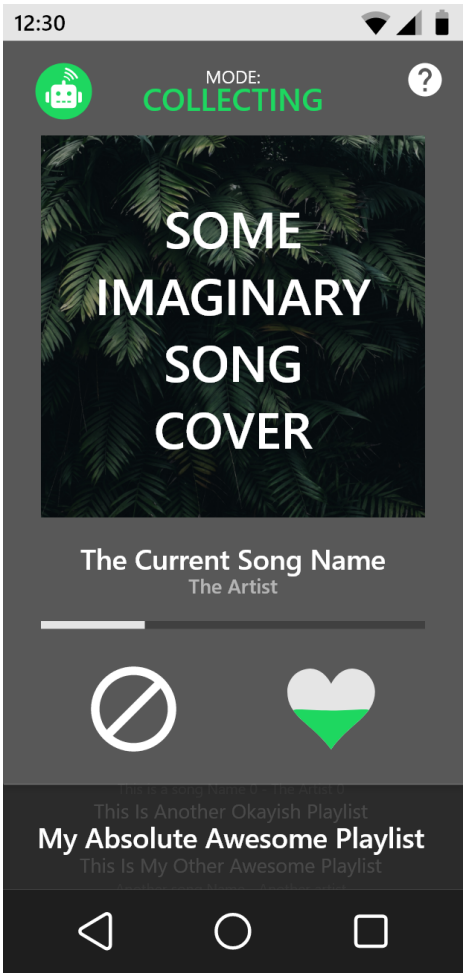

Figure 8: Collector Mode - short listening time

The demonstrated scenario shows the app with an active track in Collector-Mode, when the user did not reach the threshold for the companion to add said track to the selected playlist. The user can manually toggle the "add to playlist"-queue with a press on the heart button. If the user skips the track, it will not be added to the playlist.

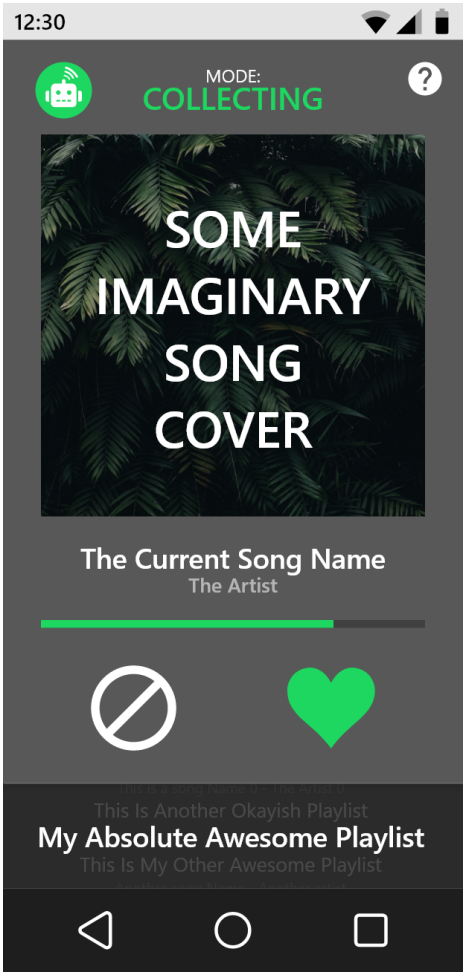

Figure 9: Collector Mode - long listening time

The demonstrated scenario shows the app with an active track in Collector-Mode, when the user did reach the threshold for the companion to add said track to the selected playlist. When the track ends it will be added to the active playlist. The user can prevent that behaviour by pressing the restraint button on the left side and the track will be ignored.

5.2 Refinement Mode

The Refinement Mode is used when the user plays back owned playlists. If the user skips a track before a set time, the track will be removed from the playlist.

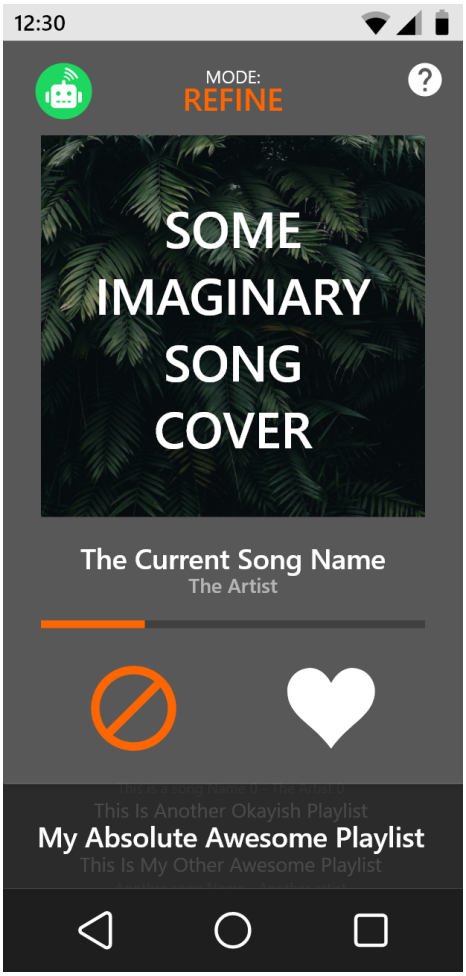

Figure 10: Refinement Mode - short listening time

The demonstrated scenario shows the app with an active track (played from a personal playlist) in Refinement-Mode, when the user did not reach the threshold for the companion to keep said track in the selected playlist. When the track is skipped it will be removed from the played playlist. The user can prevent that behaviour by pressing the heart button on the right side and the track will be ignored.

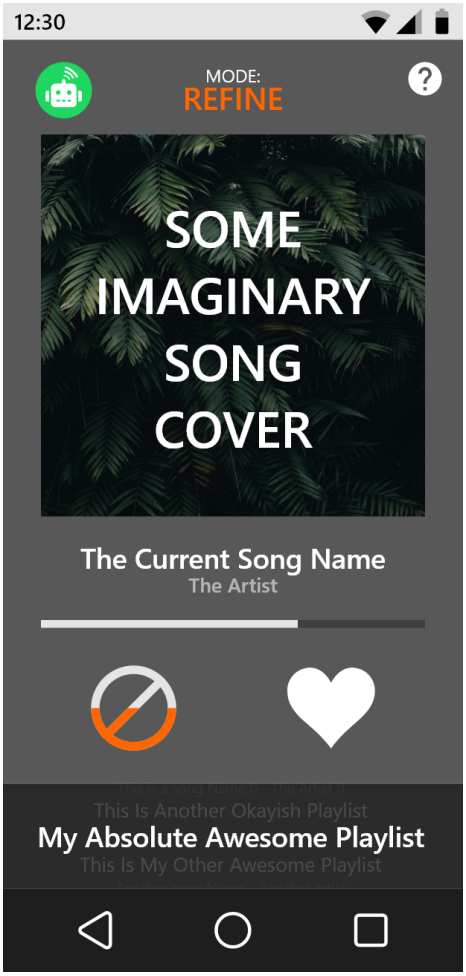

Figure 11: Refinement Mode - long listening time

The demonstrated scenario shows the app with an active track (played from a personal playlist) in Refinement-Mode, when the user did reach the threshold for the companion to keep said track in the selected playlist. When the track is skipped or ends it will stay in the played playlist. The user can prevent that behaviour by pressing the restraint button on the left side. The track will be removed from the list.

5.3 Implementation of the GUI in development

Between the first GUI prototype and the start of development, the GUI still underwent several innovations, partly due to the technology used and partly due to how to use the SDK in a better way.

The first change is the removal of the Welcome screen. The Spotify SDK comes with a method to authenticate the user, which opens a predefined window from Spotify where the user logs in. In addition, the user must authenticate access to use the app, so the window is triggered directly on app launch.

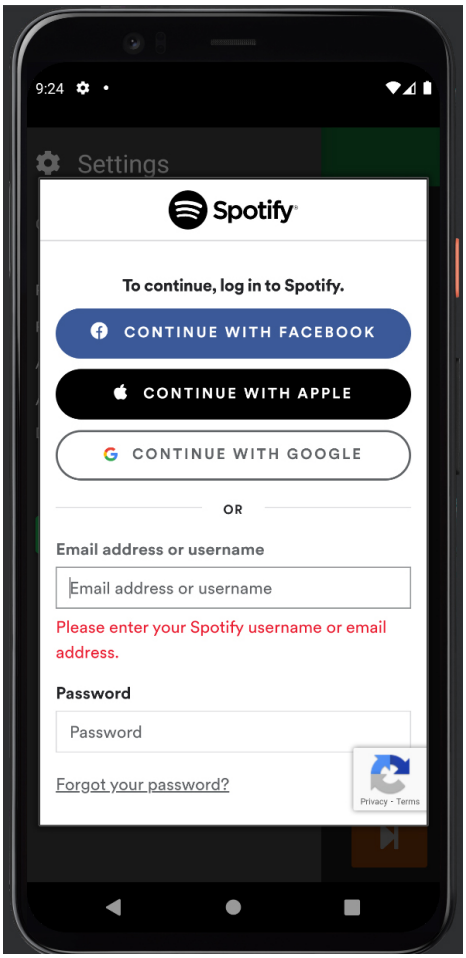

Figure 12: Spotify-Companion Auth Screen[6]

The playlist selection has been moved to the settings menu to get more screen space. The player-controls icons have been replaced with more common ones.

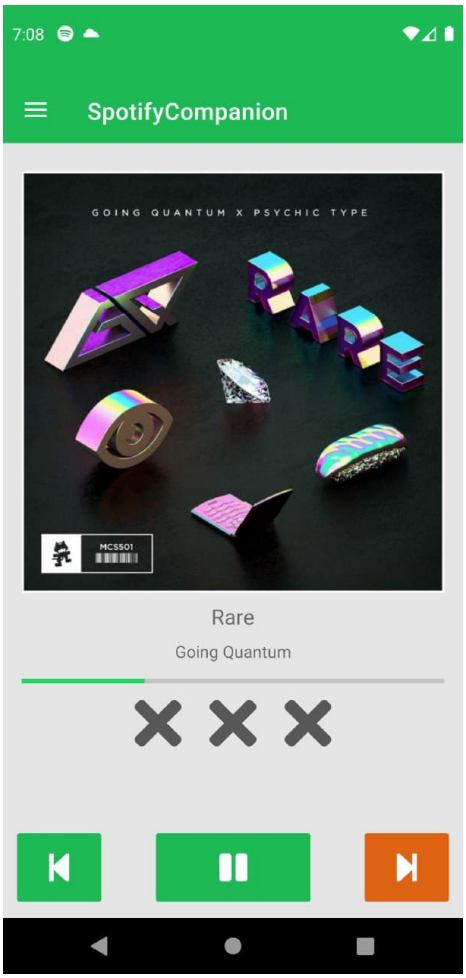

Figure 13: Spotify-Companion Main View (light)

The information how often a song has been skipped was missing in the old design and is now displayed with 3 crosses which turn red, to indicate how often a song got skipped.

A new Settings menu has been designed to provide a number of choices, such as the selection of origin and destination lists, as well as the behavior of the app regarding adding and deleting from lists and favorites. In addition, the option to log out the logged-in user (Spotify permissions) or to reset the already counted tracks has been included.

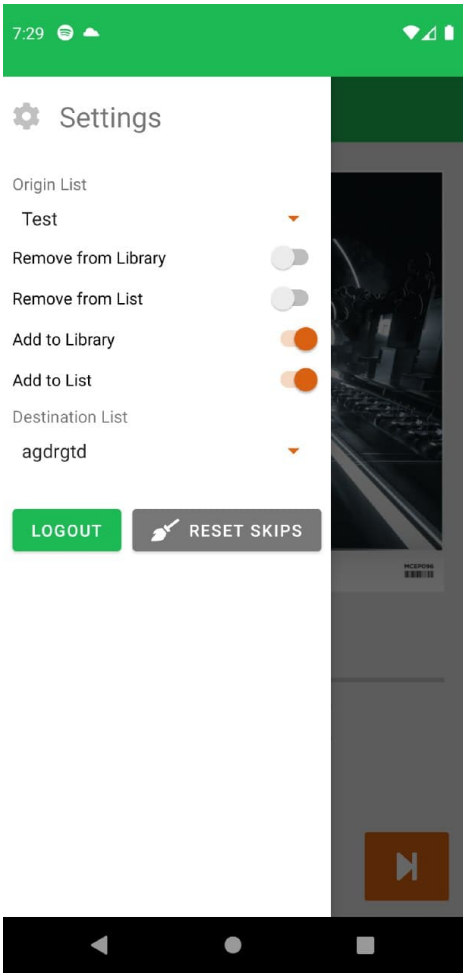

Figure 14: Spotify-Companion Settings View (light)

As an extension of the original design, a color palette for light mode and dark mode was created.

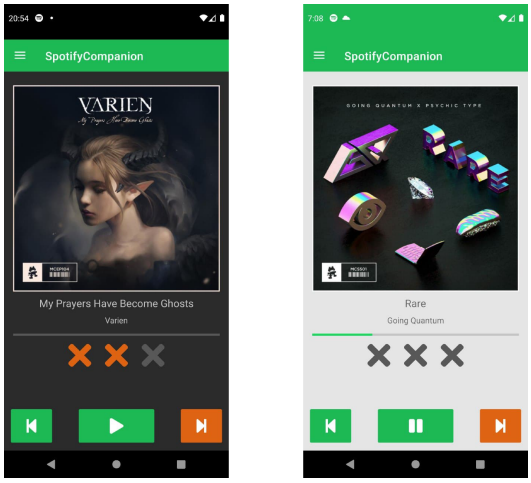

Figure 15: Comparison light-mode / dark-mode

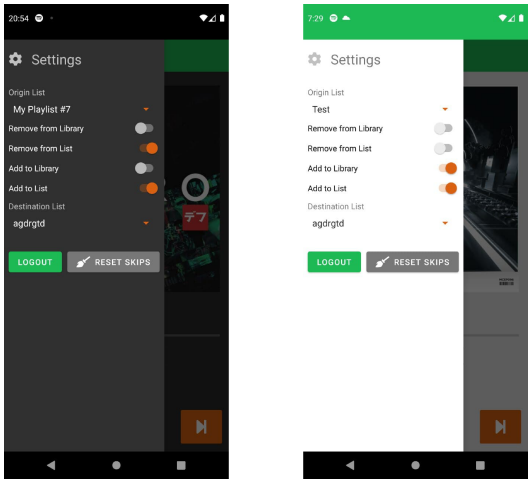

Figure 16: Comparison light-mode / dark-mode Settings

6 Summary

6.1 Requirements

6.1.1 Must Have

- (M3) comprehensible UI ✓
- (M1) removal of repeatedly skipped songs ✓
- (M2) adding not skipped songs to chosen playlist ✓

6.1.2 Should Have

- (S4) intuitive and clean UI conforming to common usability standards found in most prevailing apps. ✓

6.1.3 Could Have

- (C5) custom app logo ✓
- (C6) better car-mode-player ✗
  - (C7) interface with track-playtime-slider ✓
  - (C8) improve suggested lists (list of favorites) ✗
  - (C9) add buttons for specific songs and lists ✗
  - (C10) show next title ✗

All "Must Have" items have been implemented and therefore are checked off. M1 and M2 are satisfied due to implementation, M3 through iterations of the GUI and tests with an independent test group. The same is true for the "Should Have" category, as it contains UI/UX requirements.
"Could Have" could not be checked off entirely, only custom app logo and internal logos as well as the track-playtime-slider, progress bar internally, can be considered to be done. The better car-mode, list suggestions and preview of the coming tracks have not been realized thus far.

6.2 Possible extensions and improvements

Service could be of an "Alarm Manager"[1] type or be substituted by the receiver reacting to actions in the official spotify application.
The database could keep track of songs independently for each list and use a more sophisticated score-system instead of a simple counter.
Adding some activities for special settings an features such as the car-mode, quick-action button panel or list generation and automatic customization.

6.3 Conclusion

This project can be considered successful. All important goals have been achieved, looking at the summary of the requirements the only open, unimplemented entries are the car-mode, custom lists and quick-action-buttons.
All of the above have been mentioned to be possible expansions but were out of the scope for two beginners in mobile-development. Seeing as we have indeed managed to make use of both spotify libraries, App Remote and WEB Api, a simple but efficient database and a clean looking and easily comprehensible user interface. A great help was the webpage codinginflow[3].

7 User Guide

7.1 Prerequisites

Before running the Spotify companion app for the first time, the Spotify app must be installed. You need a Spotify Premium subscription to use our app to its full extent.

7.2 Usage

When the app is first run, the user is redirected to a Spotify login page to authenticate the app to view user data and lists, edit playlists and favorites.

Currently playing songs are displayed with cover, name and artist, as well as a progress bar. To switch to the previous song, pause playback, or skip the current song, press the buttons in the lower area in the order listed. Songs can be skipped 3 times before they exceed the limit at which the app will remove that song from a playlist of your choice. An indicator of how many times a song has been skipped is located below the progress bar. Red crosses indicate the number of skips.

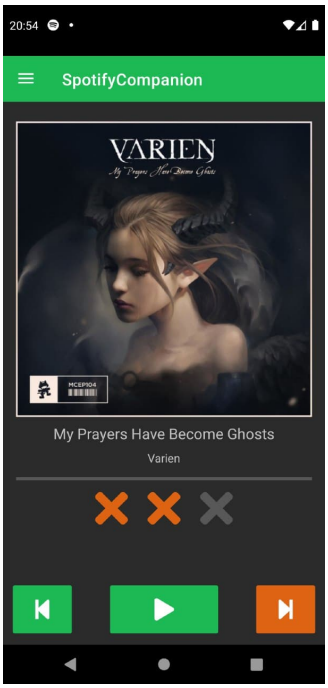

Figure 17: Spotify-Companion Main View

7.2.1 Sort out playlists and favorites

In the Settings menu, select a playlist from which to remove songs you no longer want to listen to (Origin List). Select "Remove from List" to automatically remove the songs. In this way you can also manage the songs of your library (favorites) by selecting the item "Remove from Library".

To reset songs that have already been skipped but not yet removed, click the "Reset Skips" button in the Settings menu.

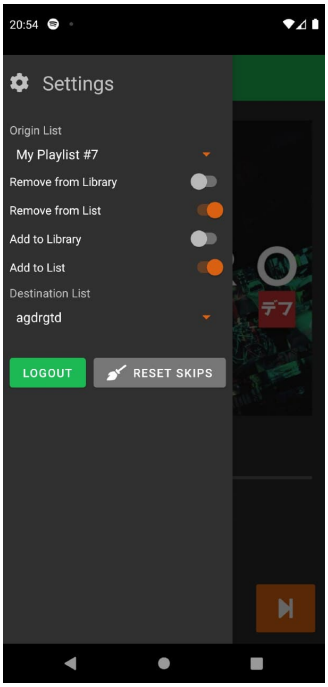

Figure 18: Spotify-Companion Main View

7.2.2 Add to playlists and favorites

In the Settings menu, select a playlist on which to add songs you do want to listen to (Destination List). Select "Add to List" to automatically add the songs. In this way you can also manage the songs of your library (favorites) by selecting the item "Add to Library".

7.3 Permission opt-out, exit using the service

If you want the app to stop accessing your Spotify account, click "Logout" in the Settings menu. This will delete the stored keys that allow access to your account and you will need to log in and authenticate your Spotify account again to use this app.

References

[1] Android service manager. "https://developer.android.com/reference/android/app/AlarmManager".

[2] Authorization guide. "https://developer.spotify.com/documentation/general/guides/authorization-guide/".

[3] codinginflow. "https://codinginflow.com/".

[4] Spotify android api. "https://developer.spotify.com/documentation/android/".

[5] Spotify app remote. "https://spotify.github.io/android-sdk/app-remote-lib/docs/index.html".

[6] Spotify auth screen. "https://developer.spotify.com/documentation/general/guides/authorization-guide/".

[7] Spotify objects index. "https://developer.spotify.com/documentation/web-api/reference-beta/objects-index".

[8] Sqlite helper. "https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper".