1. Introduction

The music industry of today is making it hard for upcoming artists and skilled amateurs to break through on the big streaming services. Therefore, they often upload their music and videos to YouTube instead. But youtube does not only consist of music but also a lot of unrelated videos, which makes it hard for users to find the exact music number they are looking for.

The goal of this project is to develop a community-based streaming service. The plan is to make it optimized for listening and make it seamless to discover new artists and popular tracks. New materials are added by the users which makes it more appealing since it makes it possible to add tracks you can't find on the big services. Revenue for the artists is generated on a yearly basis, based on how many times their songs are played. The system has a lot in common with some of the big streaming services like Spotify, YouTube Music, and Tidel. What separates this system from the others is the freedom as a user to upload any music of your choosing. The service has the potential to have a greater library than some of its competitors since you could upload music from movies and tv-shows.

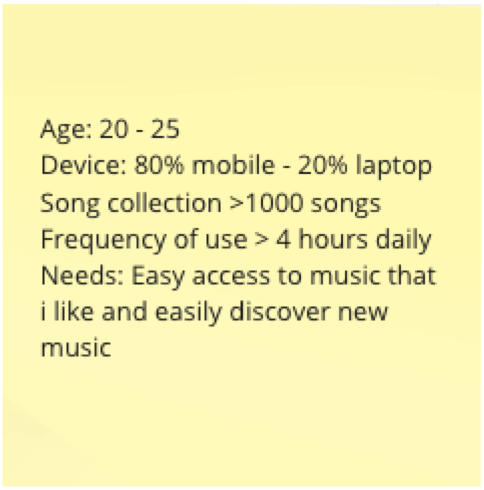
The whole project has been split up into 13 subgroups with their own responsibilities. This group's main responsibility is to design the User interface and optimize the user experience for the whole system. Furthermore, the group is responsible for implementing playlist functionality so it’s possible to create and edit a playlist and language support to make it possible to change the language for the whole application.

2. Analysis

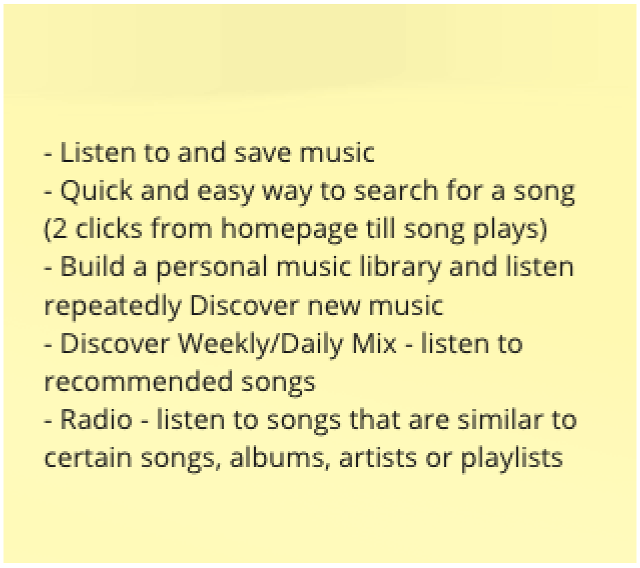
During the analysis, different analysis tools form both software design and UI/UX design. To start off the analysis the group looked at how Spotify, our biggest competitor, has designed their user experience. On that basis there will be made a use case diagram to analyse how our microservices could be implemented.

2.1 Spotify UX Analysis

2.1.1 The user

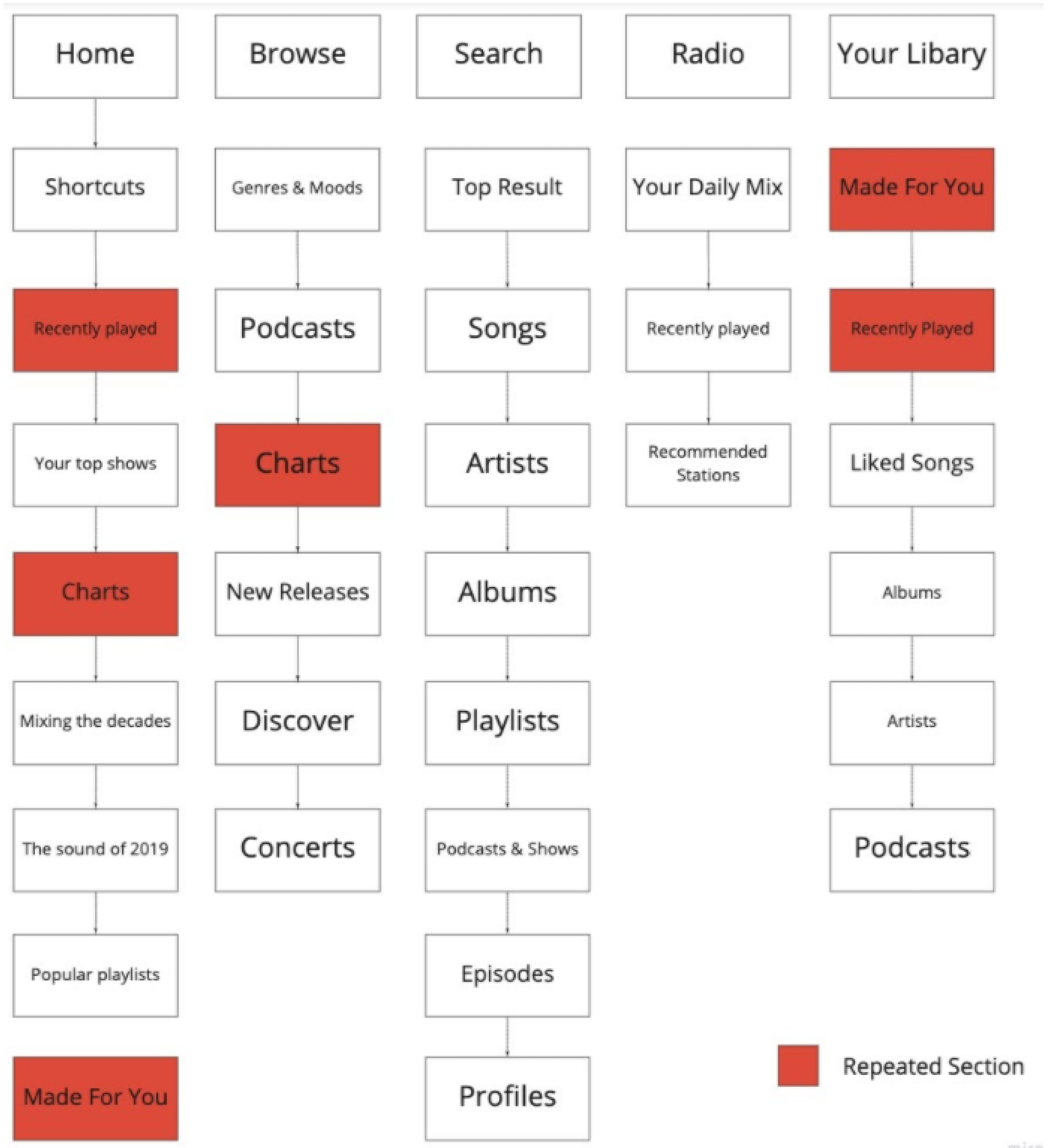

Age: 20 - 25
Device: 80% mobile - 20% laptop
Song collection >1000 songs
Frequency of use > 4 hours daily
Needs: Easy access to music that i like and easily discover new music

2.1.2 User Goals

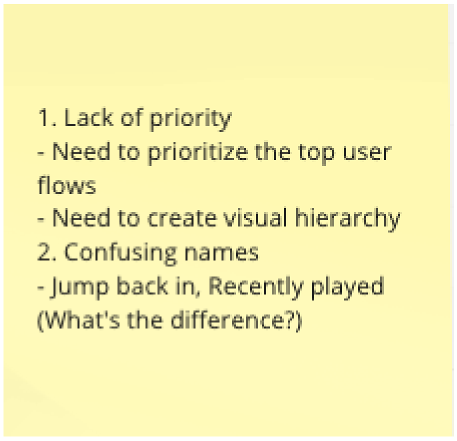


- Listen to and save music
- Quick and easy way to search for a song (2 clicks from homepage till song plays)
- Build a personal music library and listen repeatedly Discover new music
- Discover Weekly/Daily Mix - listen to recommended songs
- Radio - listen to songs that are similar to certain songs, albums, artists or playlists

2.1.3 Information Architecture



2.1.4 Main issues

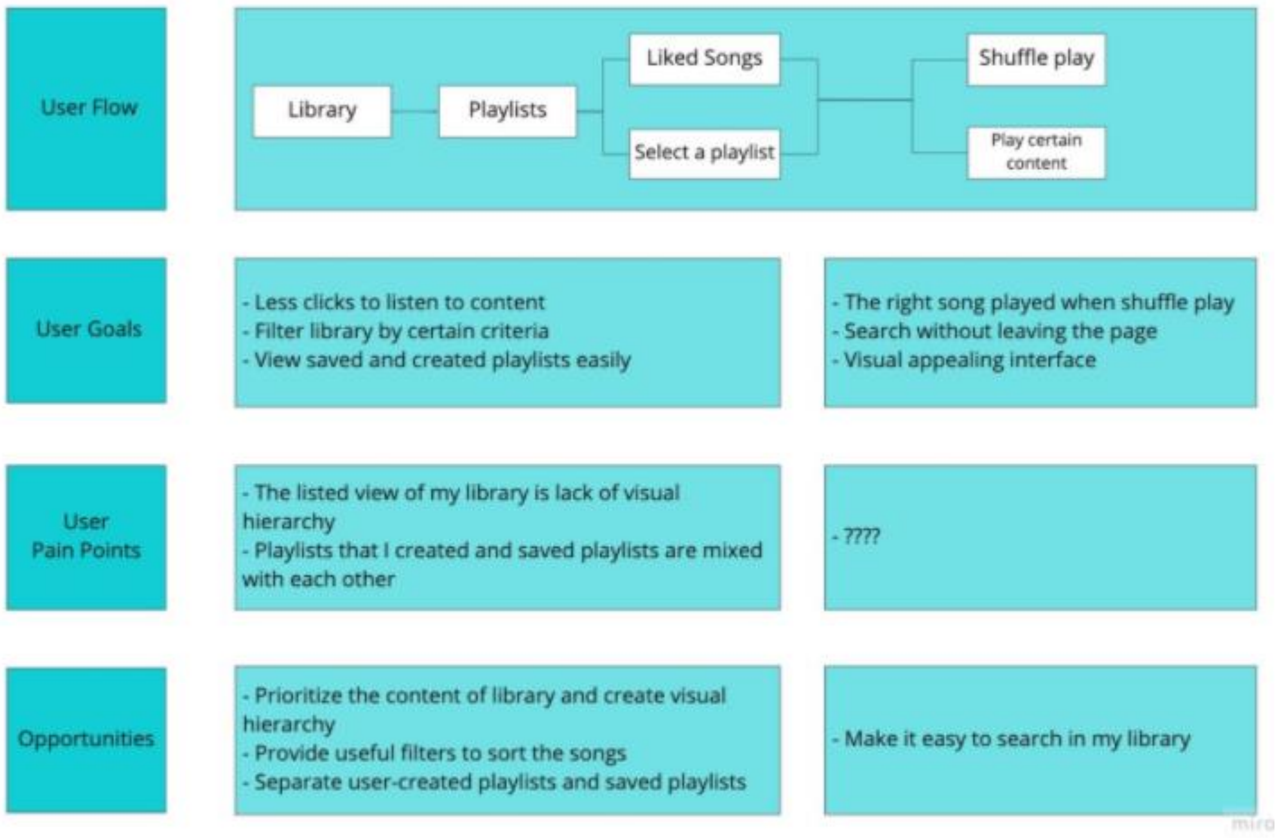


1. Lack of priority
- Need to prioritize the top user flows
- Need to create visual hierarchy
2. Confusing names
- Jump back in, Recently played (What's the difference?)

2.1.5 User Journey Map

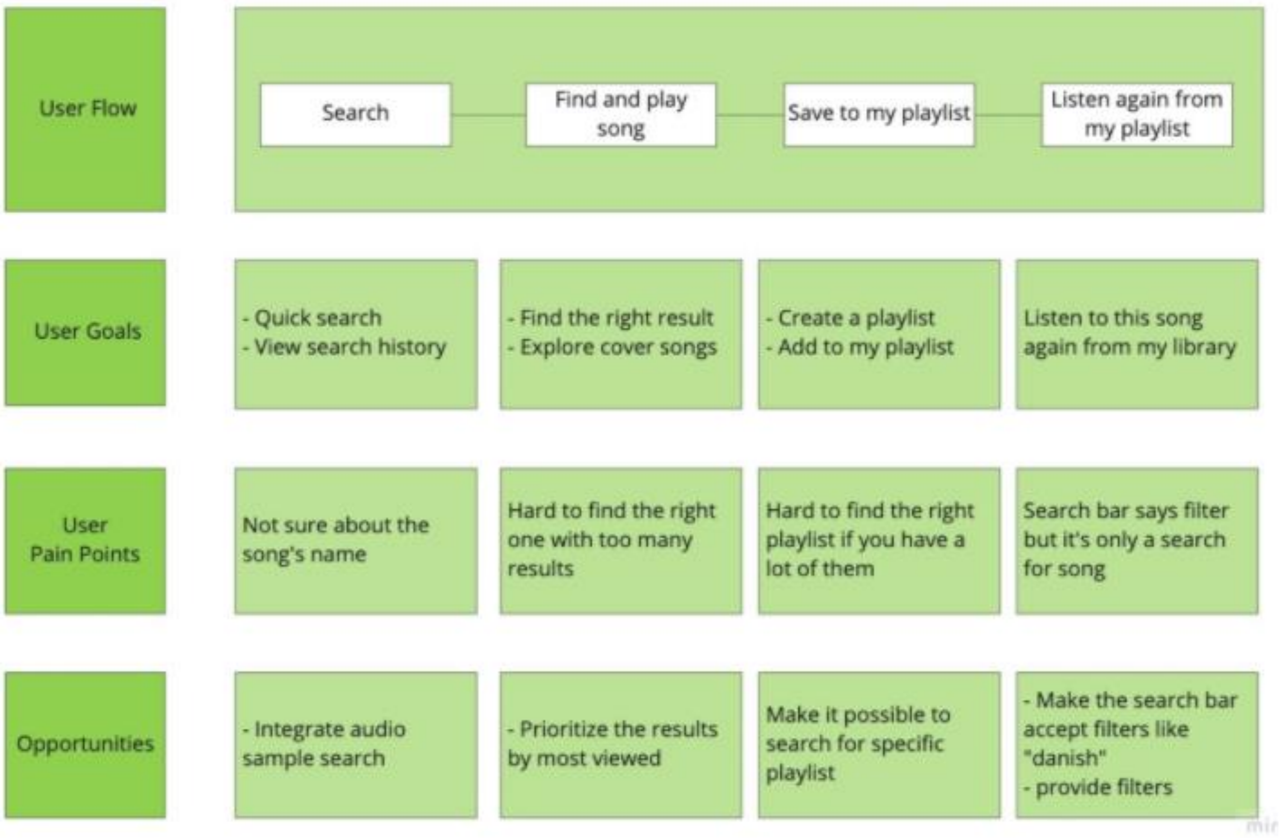
A user journey map allows you to identify and strategize for key moments. We mapped out the top three user flows with user goals and pain points at each step in order to define key opportunities to improve the overall user experience.

2.1.5.1 Spotify User Experience Map 1: Listen to saved songs



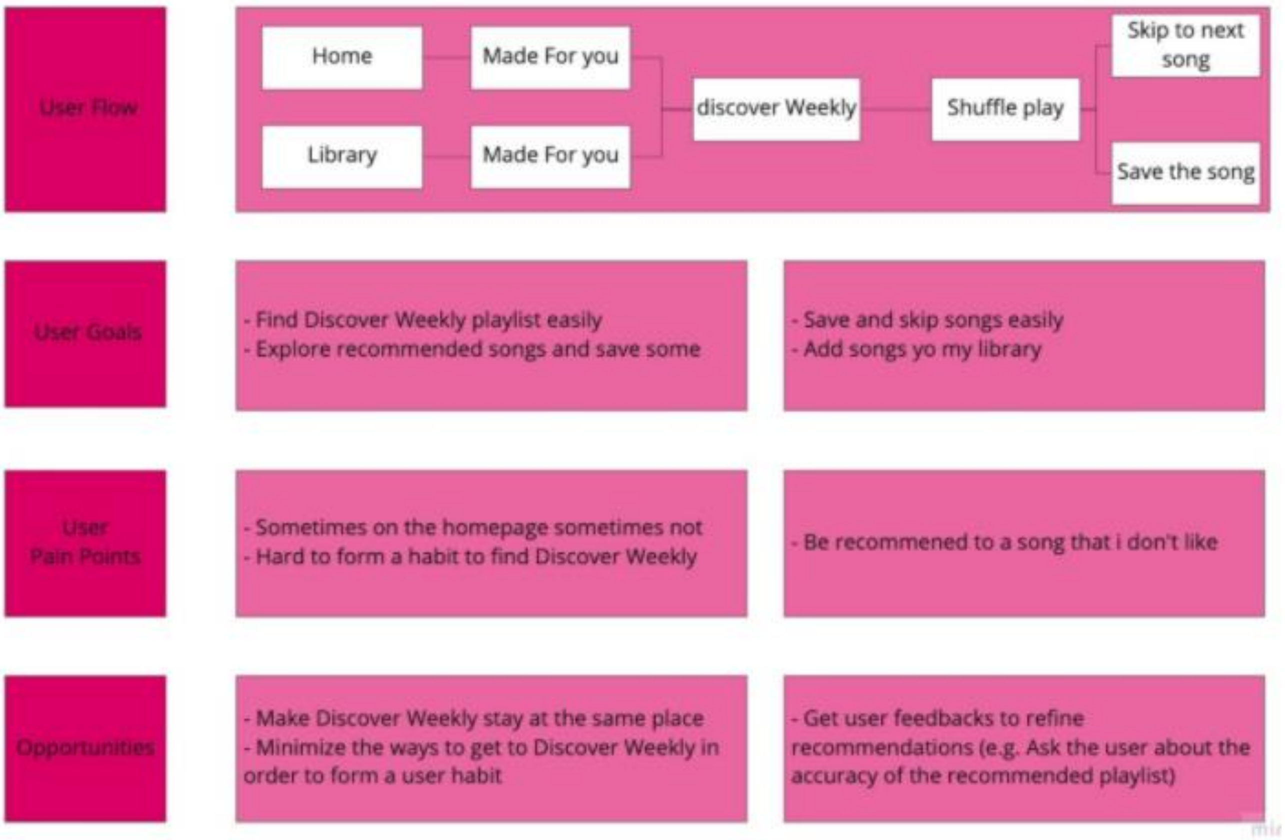
User Goals	- Less clicks to listen to content - Filter library by certain criteria - View saved and created playlists easily	- The right song played when shuffle play - Search without leaving the page - Visual appealing interface
User Pain Points	- The listed view of my library is lack of visual hierarchy - Playlists that I created and saved playlists are mixed with each other	- ????
Opportunities	- Prioritize the content of library and create visual hierarchy - Provide useful filters to sort the songs - Separate user-created playlists and saved playlists	- Make it easy to search in my library

2.1.5.2 Spotify User Experience Map 2: Search for a song and save it



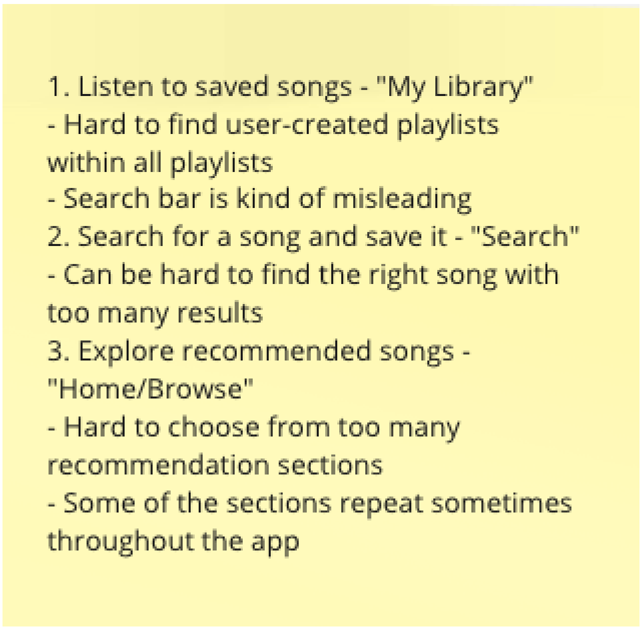
User Flow	Search	Find and play song	Save to my playlist	Listen again from my playlist
User Goals	- Quick search - View search history	- Find the right result - Explore cover songs	- Create a playlist - Add to my playlist	Listen to this song again from my library
User Pain Points	Not sure about the song's name	Hard to find the right one with too many results	Hard to find the right playlist if you have a lot of them	Search bar says filter but it's only a search for song
Opportunities	- Integrate audio sample search	- Prioritize the results by most viewed	Make it possible to search for specific playlist	- Make the search bar accept filters like "danish" - provide filters

2.1.5.3 Spotify User Experience Map 3: Explore Discover Weekly



User Goals	- Find Discover Weekly playlist easily - Explore recommended songs and save some	- Save and skip songs easily - Add songs yo my library
User Pain Points	- Sometimes on the homepage sometimes not - Hard to form a habit to find Discover Weekly	- Be recommened to a song that i don't like
Opportunities	- Make Discover Weekly stay at the same place - Minimize the ways to get to Discover Weekly in order to form a user habit	- Get user feedbacks to refine recommendations (e.g. Ask the user about the accuracy of the recommended playlist)

2.1.6 Main User Pain Points

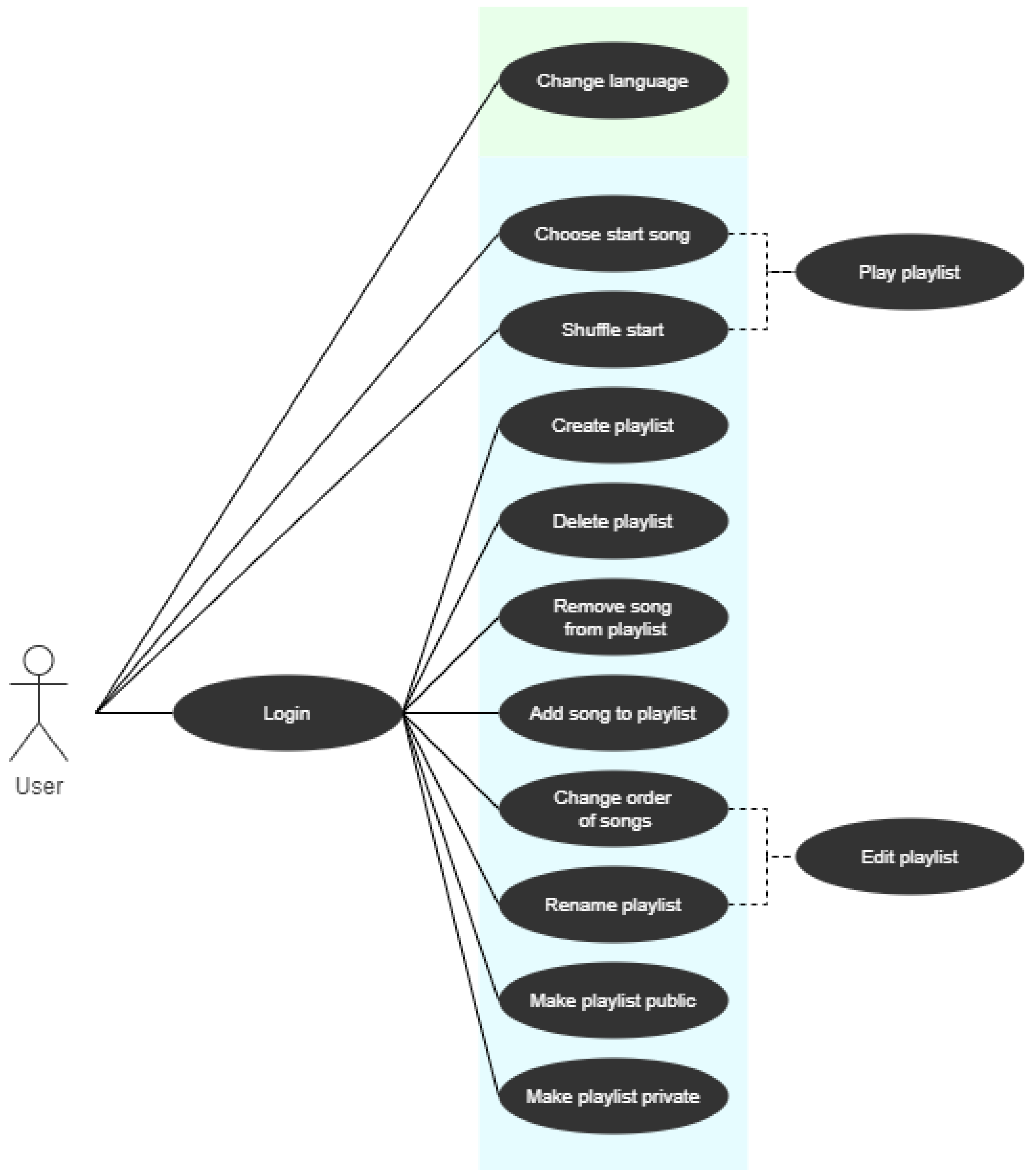


1. Listen to saved songs - "My Library"
- Hard to find user-created playlists within all playlists
- Search bar is kind of misleading
2. Search for a song and save it - "Search"
- Can be hard to find the right song with too many results
3. Explore recommended songs - "Home/Browse"
- Hard to choose from too many recommendation sections
- Some of the sections repeat sometimes throughout the app

2.2 Use Case

The process of creating the use case diagram begins with a quick brainstorm based on the project kickoff slides regarding the functionality of the playlist integration and language support. With the listing of different functionalities, it is possible to draft the use case diagram. An example of this type of use case is “display hint-text when hovering over icons”, the general conclusion of the discussion is that it is expected of a web browser to display the hint-text automatically. It is the group's responsibility to translate the hint-text but not displaying it to the user.

A use case list is then created containing every use case with an designated id, name, and description.



Green = Language support requirements

Blue = Playlist requirements

2.3 Group analysis

Through discussion internally in the group about what functions a playlist API and language API should contain, and some inspiration from other streaming services, the group came up with a set of requirements the group saw as crucial for the implementation of playlists and the changing of languages.

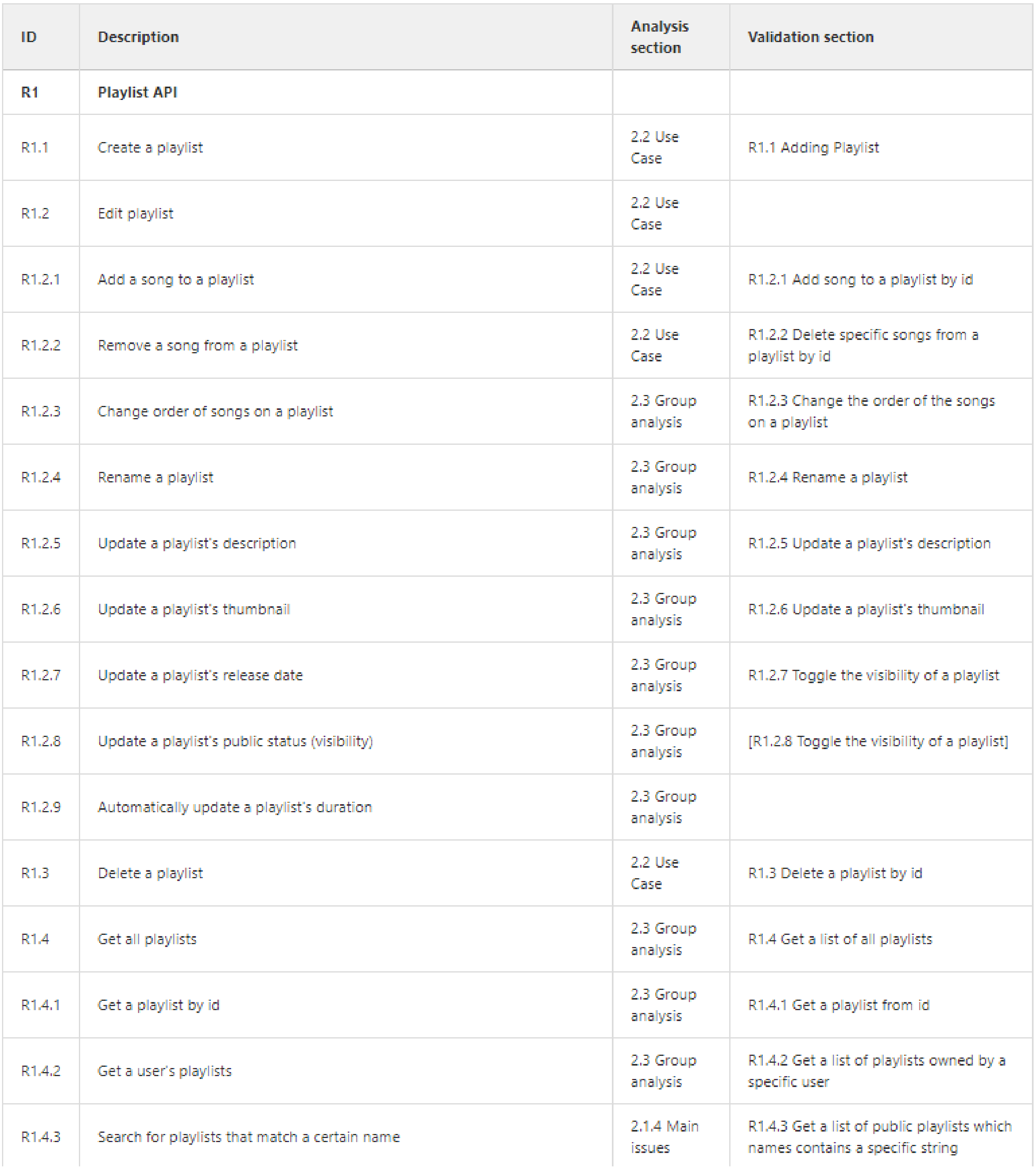
These requirements are not based on a specific analysis tools but rather what features the group considers a "must", and what requirements makes sense to include to secure a smooth and easy user experience, thus making our service able to compete with the big streaming services already on the market.

The groups that we had specific requests from were the web team, the data search team, the media player team, and the data collection team. Web needed a access to all playlists from a specific user, data searched needed a way to find playlists by name, media player needed to access a specific playlist so they could play the songs from it, and the data collection team would like a way of getting all playlists.

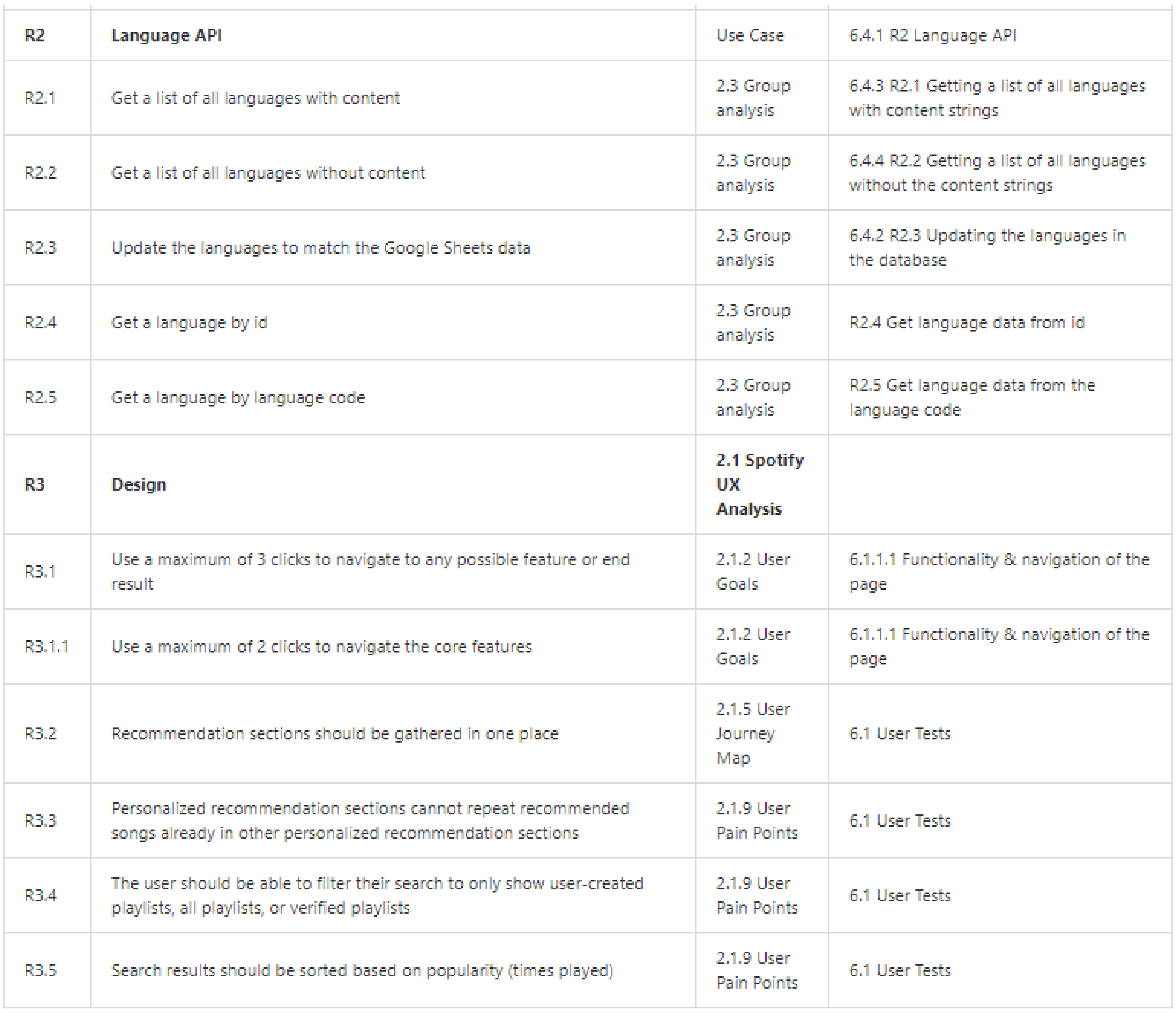
2.4 Conclusion

This analysis of Spotify’s UX can help the group understand the important and main points of focus. This includes what features the group might want to “borrow” from Spotify and what features that can be improved upon. Exactly why Spotify was chosen for this kind of analysis, is because of their very well known interface and minimalistic/modern design. These things may play a part in Spotify’s success, but which can always be improved. This analysis lays the foundation for the design-wise requirements later defined in the Requirements Specification. The use case also defines Create Playlist- and Language Settings requirements found under Functionality in Requirements Specification.

3. Requirements Specification



ID	Description	Analysis section	Validation section
R1	**Playlist API**		
R1.1	Create a playlist	2.2 Use Case	R1.1 Adding Playlist
R1.2	Edit playlist	2.2 Use Case	
R1.2.1	Add a song to a playlist	2.2 Use Case	R1.2.1 Add song to a playlist by id
R1.2.2	Remove a song from a playlist	2.2 Use Case	R1.2.2 Delete specific songs from a playlist by id
R1.2.3	Change order of songs on a playlist	2.3 Group analysis	R1.2.3 Change the order of the songs on a playlist
R1.2.4	Rename a playlist	2.3 Group analysis	R1.2.4 Rename a playlist
R1.2.5	Update a playlist's description	2.3 Group analysis	R1.2.5 Update a playlist's description
R1.2.6	Update a playlist's thumbnail	2.3 Group analysis	R1.2.6 Update a playlist's thumbnail
R1.2.7	Update a playlist's release date	2.3 Group analysis	R1.2.7 Toggle the visibility of a playlist
R1.2.8	Update a playlist's public status (visibility)	2.3 Group analysis	[R1.2.8 Toggle the visibility of a playlist]
R1.2.9	Automatically update a playlist's duration	2.3 Group analysis	
R1.3	Delete a playlist	2.2 Use Case	R1.3 Delete a playlist by id
R1.4	Get all playlists	2.3 Group analysis	R1.4 Get a list of all playlists
R1.4.1	Get a playlist by id	2.3 Group analysis	R1.4.1 Get a playlist from id
R1.4.2	Get a user's playlists	2.3 Group analysis	R1.4.2 Get a list of playlists owned by a specific user
R1.4.3	Search for playlists that match a certain name	2.1.4 Main issues	R1.4.3 Get a list of public playlists which names contains a specific string



R2	**Language API**	Use Case	6.4.1 R2 Language API
R2.1	Get a list of all languages with content	2.3 Group analysis	6.4.3 R2.1 Getting a list of all languages with content strings
R2.2	Get a list of all languages without content	2.3 Group analysis	6.4.4 R2.2 Getting a list of all languages without the content strings
R2.3	Update the languages to match the Google Sheets data	2.3 Group analysis	6.4.2 R2.3 Updating the languages in the database
R2.4	Get a language by id	2.3 Group analysis	R2.4 Get language data from id
R2.5	Get a language by language code	2.3 Group analysis	R2.5 Get language data from the language code
R3	**Design**	**2.1 Spotify UX Analysis**	
R3.1	Use a maximum of 3 clicks to navigate to any possible feature or end result	2.1.2 User Goals	6.1.1 Functionality & navigation of the page
R3.1.1	Use a maximum of 2 clicks to navigate the core features	2.1.2 User Goals	6.1.1 Functionality & navigation of the page
R3.2	Recommendation sections should be gathered in one place	2.1.5 User Journey Map	6.1 User Tests
R3.3	Personalized recommendation sections cannot repeat recommended songs already in other personalized recommendation sections	2.1.9 User Pain Points	6.1 User Tests
R3.4	The user should be able to filter their search to only show user-created playlists, all playlists, or verified playlists	2.1.9 User Pain Points	6.1 User Tests
R3.5	Search results should be sorted based on popularity (times played)	2.1.9 User Pain Points	6.1 User Tests

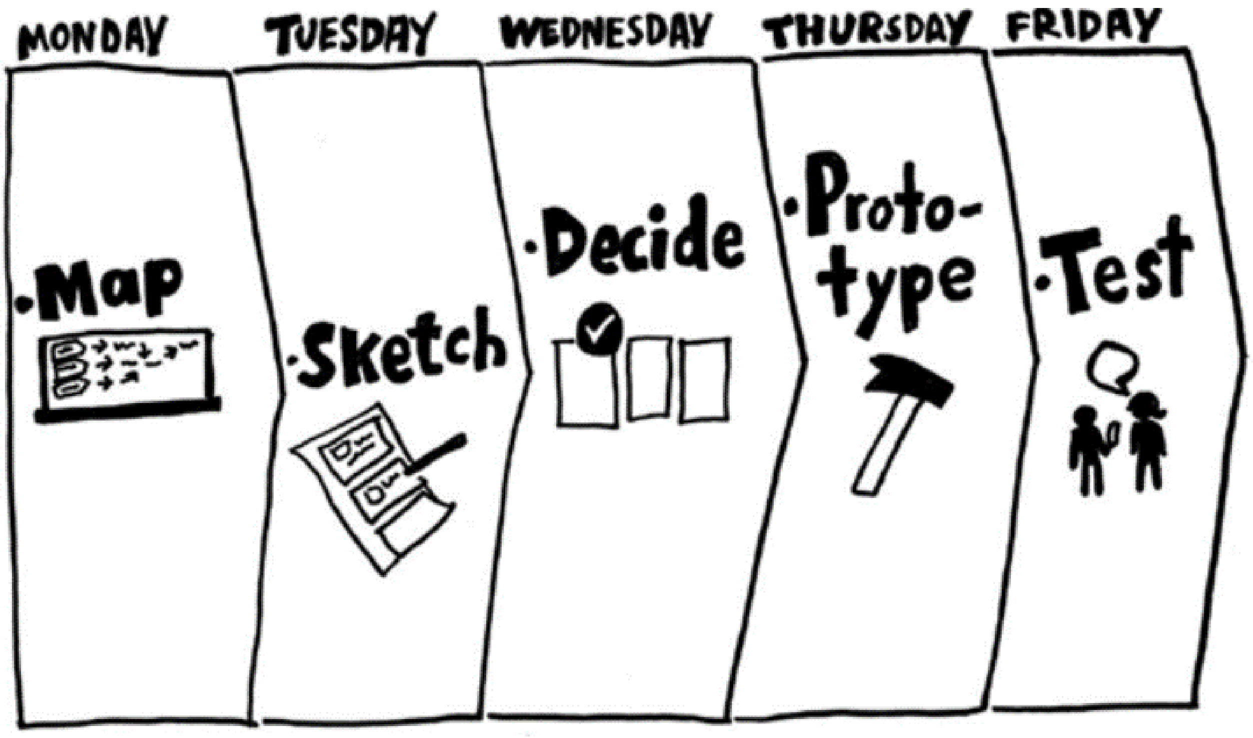
4. Design

4.1 Design Sprint

In order to create a good, user-friendly service, the group did a design sprint. A design sprint is a time-constrained, five-phase process that aims to reduce risk when bringing a new product, service, or feature to the market.

The process helps the team in clearly defining goals, validating assumptions, and deciding on a product roadmap before starting development.

The process for a Design sprint looks as follows:

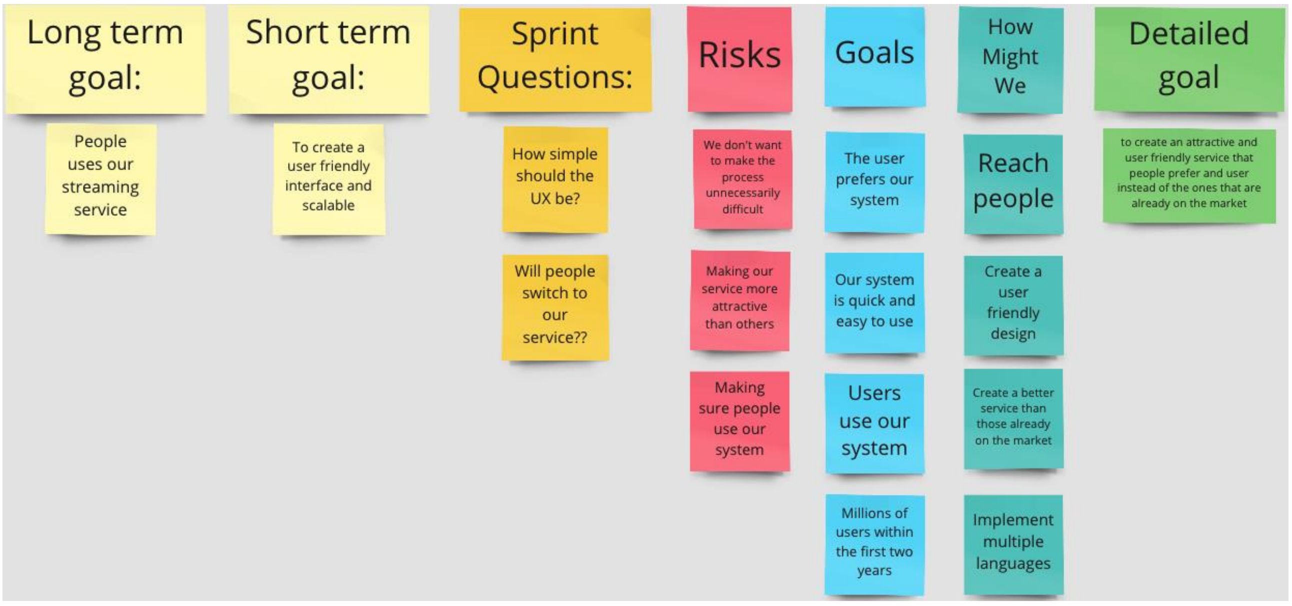


This can all be found on our Miro board

4.1.1 Monday - Mapping

On the first day of the sprint, the goal is to build a foundation and set a focus for the rest of the sprint. This is done by setting short term and long term goals, deciding on a core demographic as well as evaluating potential risks.

By the end of the day, the goal is to have maps, walking through the steps a user would take, going through our service.



Long term goal:

- People uses our streaming service

Short term goal:

- To create a user friendly interface and scalable

Sprint Questions:

- How simple should the UX be?
- Will people switch to our service??

Risks

- We don't want to make the process unnecessarily difficult
- Making our service more attractive than others
- Making sure people use our system

Goals

- The user prefers our system
- Our system is quick and easy to use
- Users use our system
- Millions of users within the first two years

How Might We

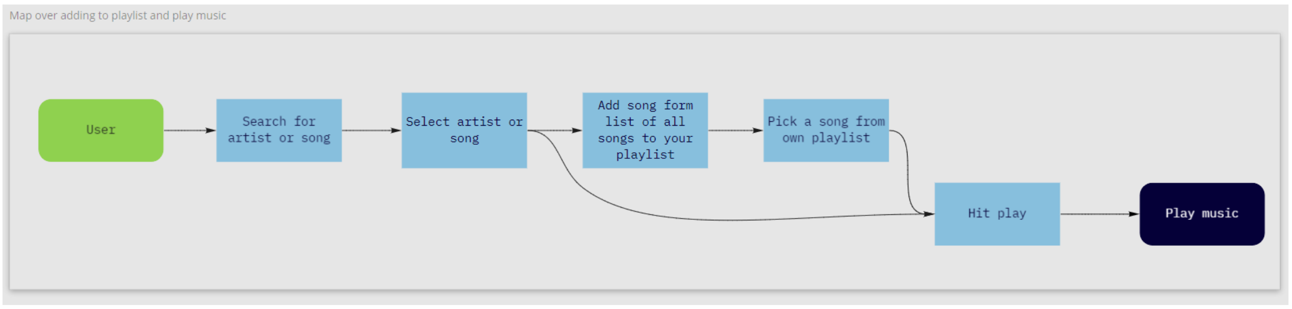
- Reach people
- Create a user friendly design
- Create a better service than those already on the market
- Implement multiple languages

Detailed goal

- to create an attractive and user friendly service that people prefer and user instead of the ones that are already on the market

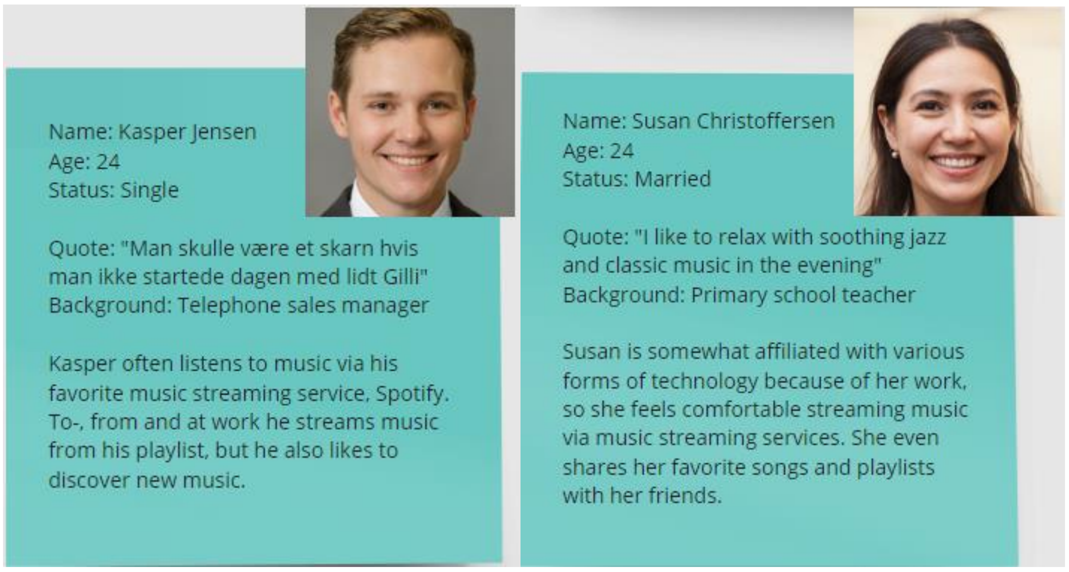
The maps below show the steps through two of the service's main features, that the group was responsible for

Map - Play music


Map over adding to playlist and play music
User
Search for artist or song
Select artist or song
Add song form list of all songs to your playlist
Pick a song from own playlist
Hit play
Play music

Personas

Personas were created in order to show what a typical user of our service might look like



Name: Kasper Jensen
Age: 24
Status: Single

Quote: "Man skulle være et skarn hvis man ikke startede dagen med lidt Gilli"
Background: Telephone sales manager

Kasper often listens to music via his favorite music streaming service, Spotify. To-, from and at work he streams music from his playlist, but he also likes to discover new music.

Name: Susan Christoffersen
Age: 24
Status: Married

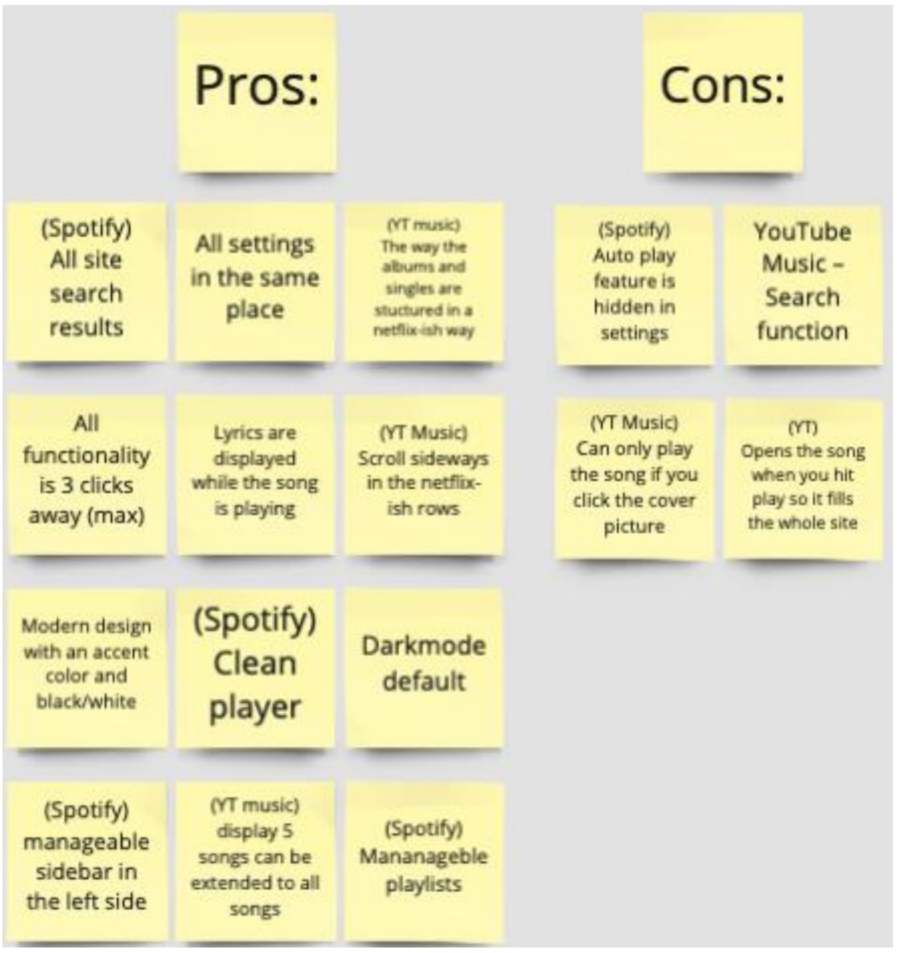
Quote: "I like to relax with soothing jazz and classic music in the evening"
Background: Primary school teacher

Susan is somewhat affiliated with various forms of technology because of her work, so she feels comfortable streaming music via music streaming services. She even shares her favorite songs and playlists with her friends.

4.1.2 Tuesday - Sketching

The goal of Tuesday was to define the challenge and choose a target. It starts with lightning Demos: a review of existing ideas to improve. Here, the group focused on the existing streaming services that we use.

Lightning Demos



Pros:

(Spotify) All site search results

All settings in the same place

(YT music) The way the albums and singles are stuctured in a netflix-ish way

All functionality is 3 clicks away (max)

Lyrics are displayed while the song is playing

(YT Music) Scroll sideways in the netflix-ish rows

Modern design with an accent color and black/white

(Spotify) Clean player

Darkmode default

(Spotify) manageable sidebar in the left side

(YT music) display 5 songs can be extended to all songs

(Spotify) Mananageble playlists

Cons:

(Spotify) Auto play feature is hidden in settings

YouTube Music - Search function

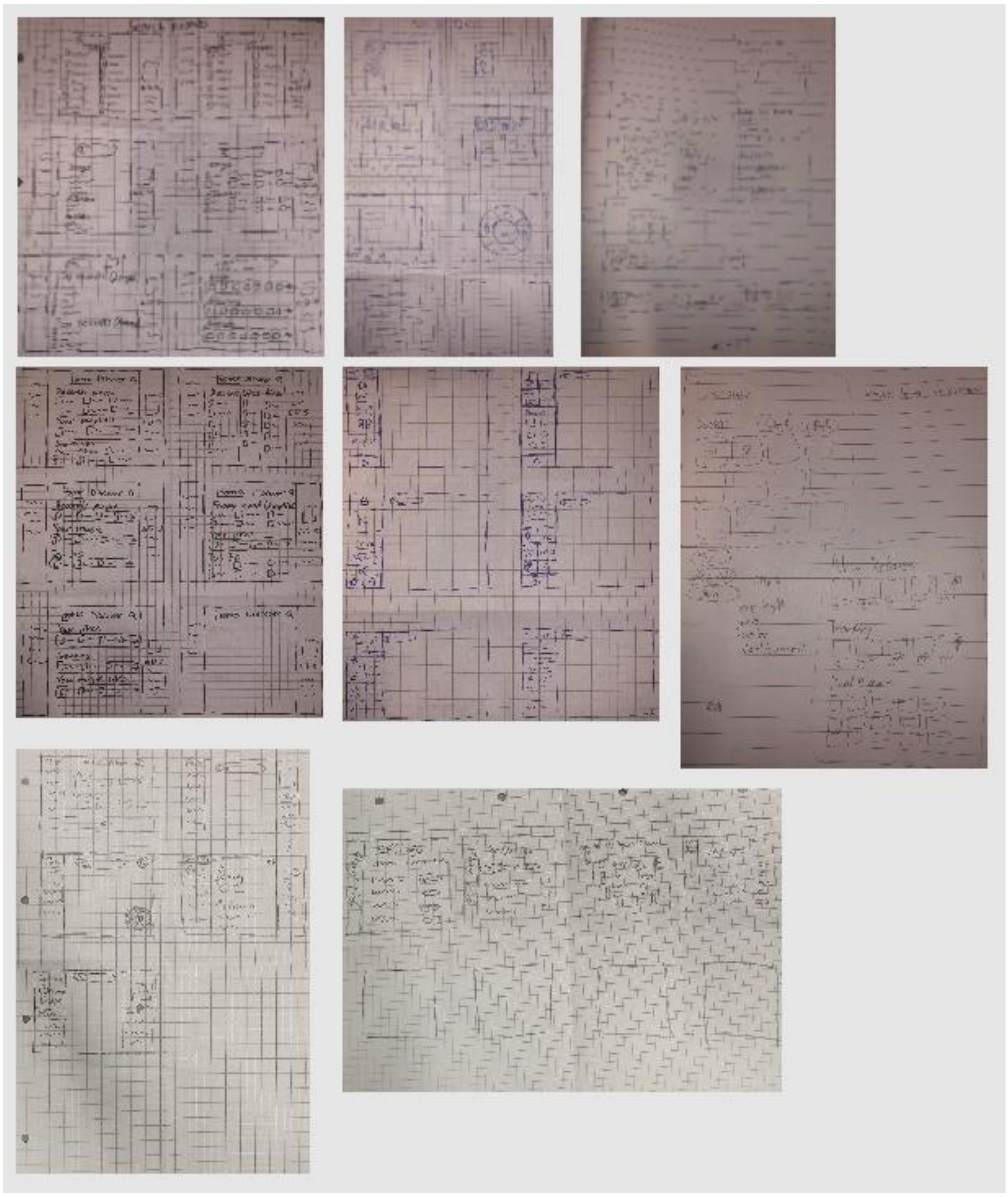
(YT Music) Can only play the song if you click the cover picture

(YT) Opens the song when you hit play so it fills the whole site

Crazy 8's

Based on the lightning demos, the group did the exercise "crazy 8's". Here, 8 drawings of the service were created in just 8 minutes.

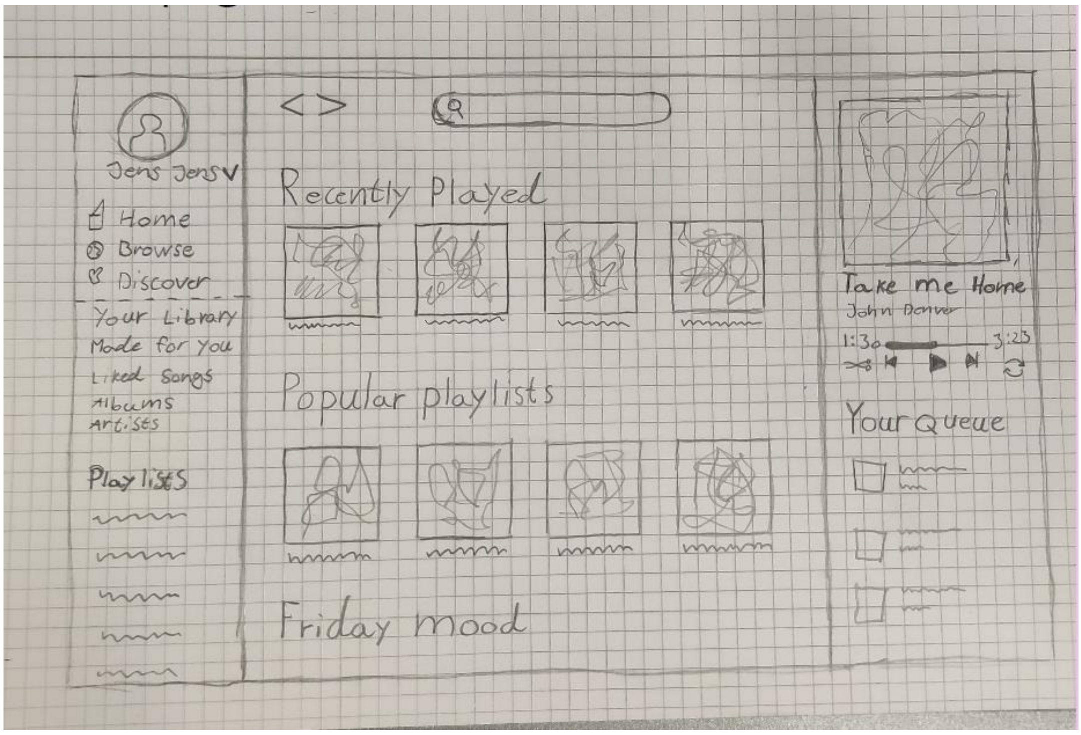
To zoom in, please check the [Miro board]()



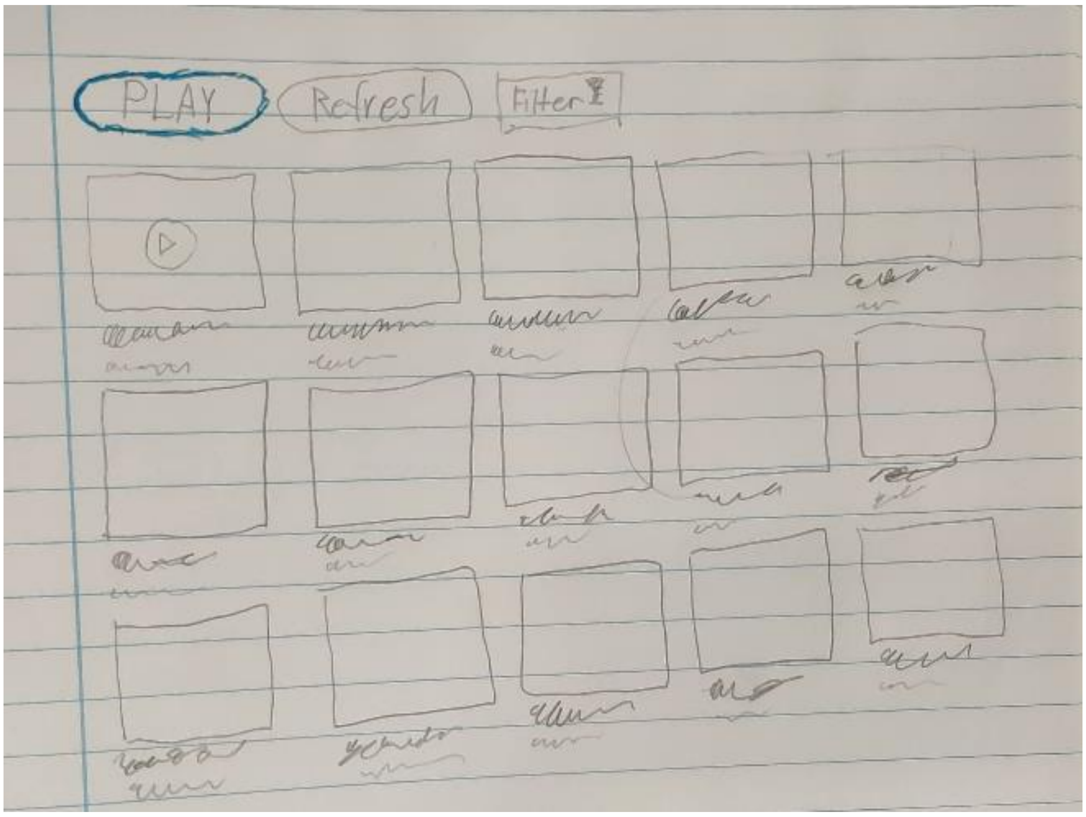
Solutions

Based on the ideas generated during the crazy 8's exercise, more detailed solutions were drawn, showing what we would like specific parts of the service to look like.

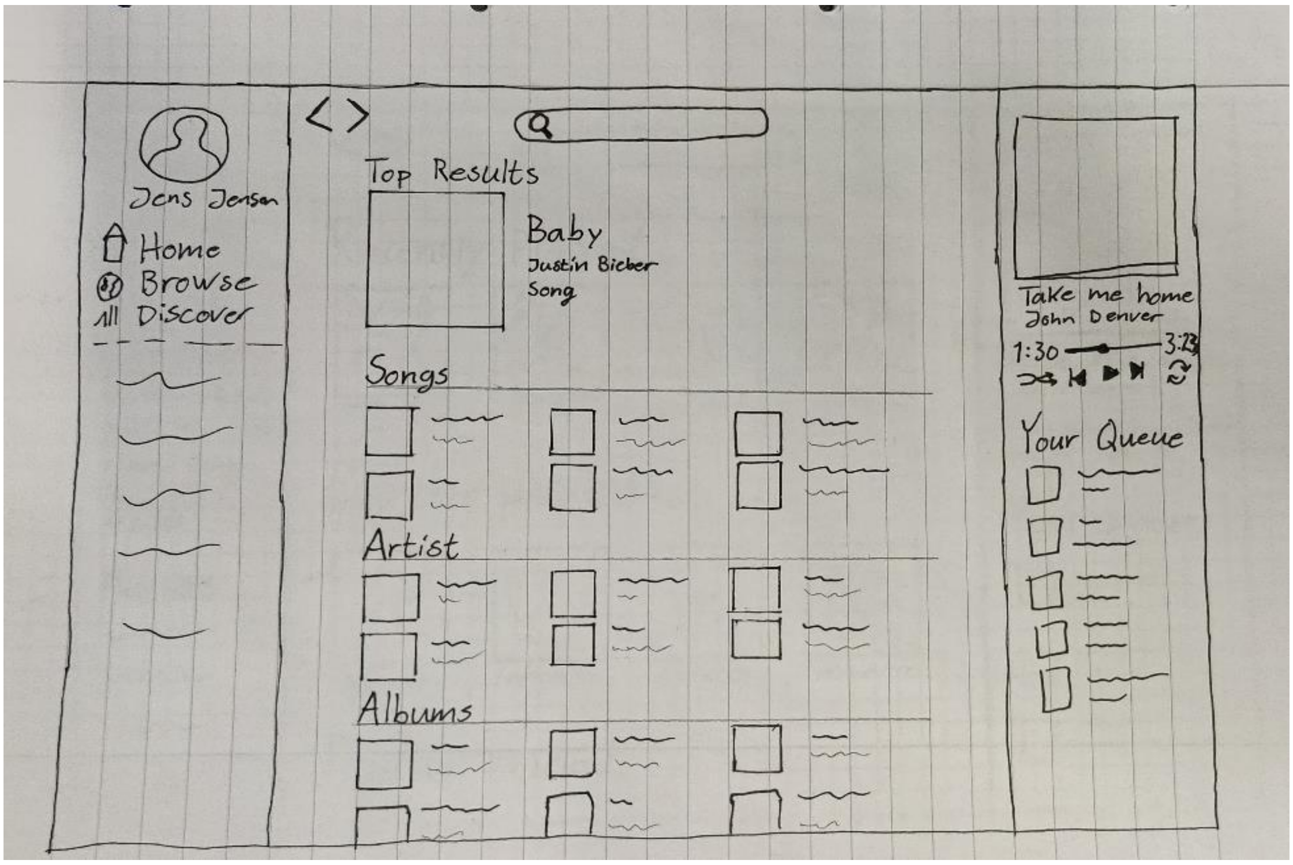
Home Page:



Discover:



Search function:

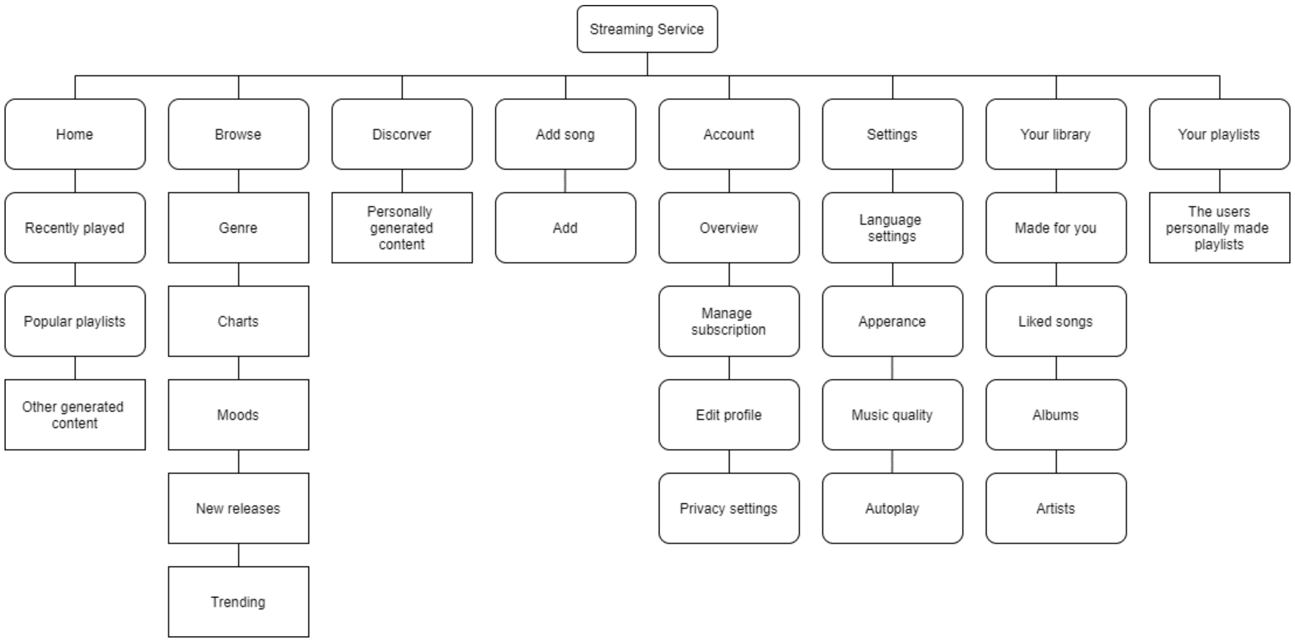


4.1.3 Wednesday - Deciding

The goal of this step is to settle completely on the idea and decide how the service will work.

Storyboards were created, depicting the core features and which steps the user would take to do certain things on the service.

The group also created a sitemap, creating an overview and something to base the prototype on



4.1.4 Thursday - Prototyping

A prototype was created based on the storyboards from Wednesday

4.1.5 Friday - Testing

To test the prototype, the group gathered and interviewed a test panel consisting of five people aged 21-29, a summary of these can be seen under Validation

The interviews were done according to the guidelines learned in Human-Computer Interaction

Lastly, the prototype was edited based on the results of the interviews.

4.2 UI/UX

A streaming service like this can easily turn into a mess if the design is not made with care and simplicity in mind. Many streaming services alike already exist and people may have used a particular one for a long time. Therefore people are used to the design being in a very certain way. Any drastic variations to these designs will do more harm than good because people do not like unfamiliar things. Borrowing some inspiration from liked services such as Spotify and YouTube Music is almost a must if the system being made doesn't want to deviate too much from the designs people are used to. We don’t necessarily want to reinvent the wheel, but rather improve the already existing one. The overall layout of the design resembles Spotify quite a lot, a service many are very familiar with. This familiarity was later praised during the interviews covered in Analysis.

Based on the liked features, the group could start building the prototype using Figma. The final prototype made in the Design Sprint is very detailed and consists of (almost) every possible site a user may want to have in a system for music streaming.

4.3 Design choices

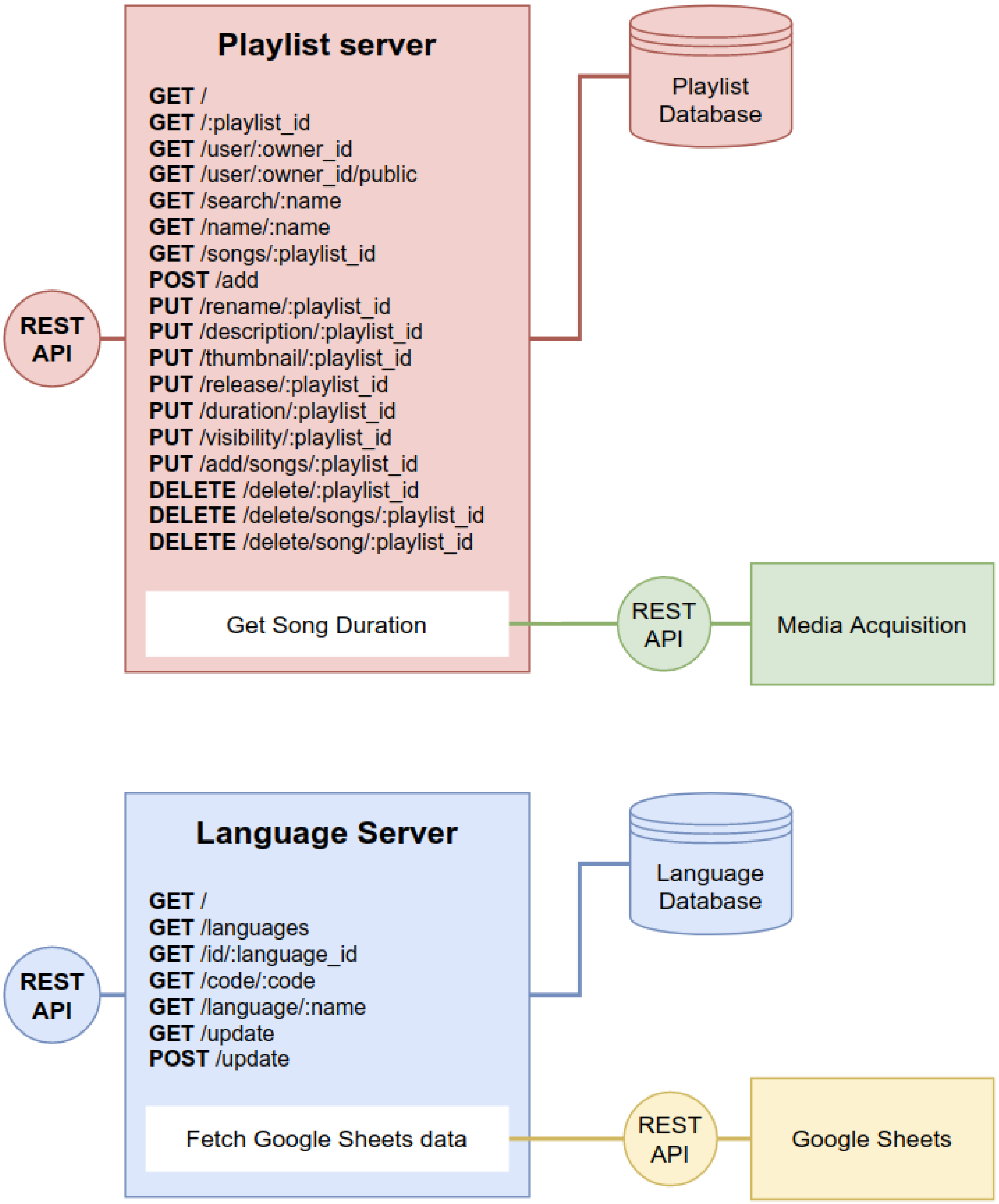
The Design Sprint provided the group with many useful tools to develop the wanted look of the system. One of the first things the group decided on was the general layout of the home page and the linked sites. These consist of a “main” content page in the middle, a left sidebar with links to almost every useful feature, and a right sidebar with the mediaplayer and song queue.

The default appearance of the system is dark. The use of darkmode is on the rise and is considered to be modern and easy on the eyes. In settings it is possible to enable two gradient colors of choice. This feature gives the user some freedom to customize the site to their liking, this will also provide a clean looking contrast between the elements. The look of the buttons around the site are kept consistent to provide familiarity. The “Login”- and “Create account”-buttons on the Log in-page have a very certain look to them and that style is kept for all buttons around the system. Here the use of gradience, round corners and the hover-feature makes the buttons stand out. Round corners are used many times throughout the system as it is more pleasant to look at. Another thing that makes the system more pleasant to look at, is how the elements are not separated with boxes or lines but rather with the use of whitespace. By aligning the elements correctly and using clear proportions, a sense of boxes is created but with a more intuitive look. The use of icons, especially in the left sidebar also adds familiarity and intuitiveness and can help guide the user around. To further simplify the overall look, only one font has been used throughout the system. Font size and weight vary to differentiate between headers and paragraphs.

4.4 Software Design

As the team was responsible for two aspects of the project, language support and playlist support, which is a small but essential part of the general system. Hence the architecture that made sense to apply to the development of our software, was to go with the microservices architecture. To approach this, the two systems should both be in their own isolated environments where the only way to access these systems is through a REST API. These API endpoints allow other groups to use the system regardless of the programming language they use as the systems would communicate in JSON. To achieve this the systems are going to be containerized with a backend server and a database. Enabling CRUD operations through HTTP calls to the different REST APIs.

4.4.1 Deployment Diagram



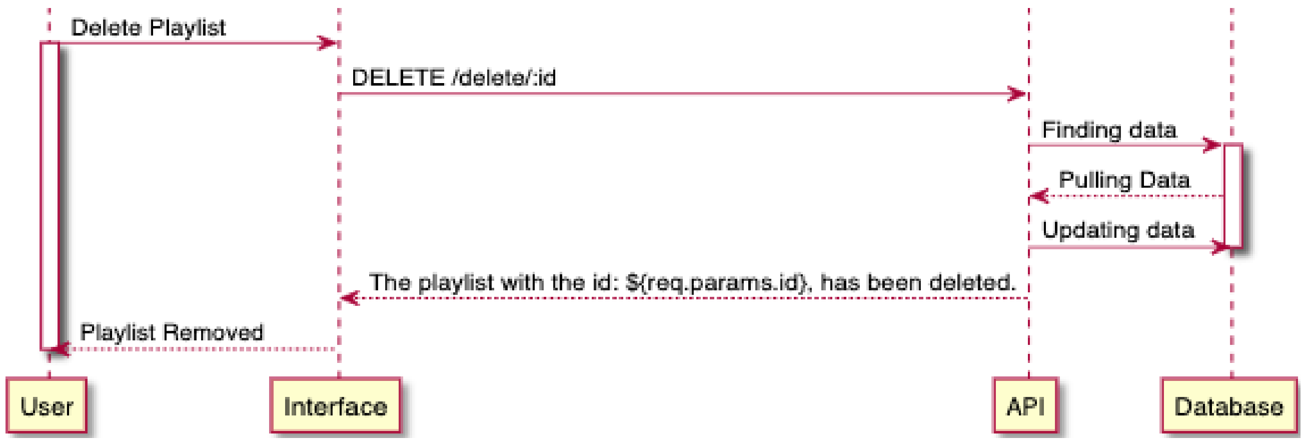
The deployment diagram depicts the software architecture of the two APIs. Red represents the playlist API, and the available endpoints are listed in the box. Green represents the data the service needs in order to function, which is an API call to media acquisitions API. Blue represents the language API, and yellow represents the API call to the google sheets document, that the language server uses to access its changes.

4.5 Sequence diagrams

Playlist API

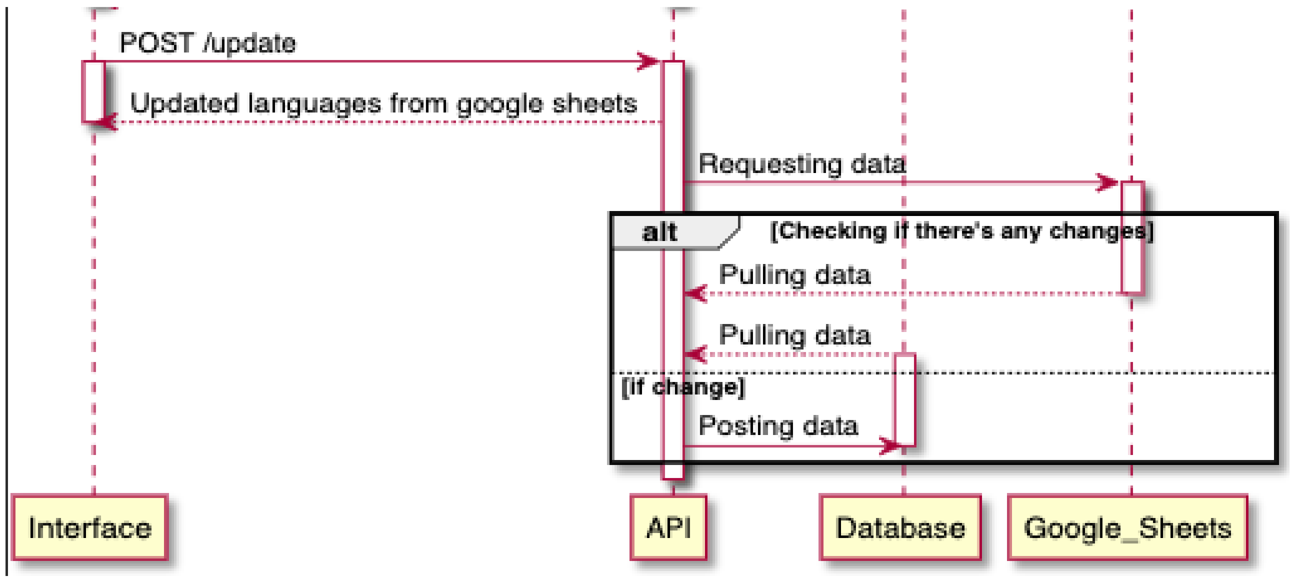
To illustrate how our APIs work we decided to use sequence diagrams for each of our APIs. Because most of the HTTP requests for the playlist API work the same we will only dig into delete a playlist works. The rest of the sequence diagram can be found here

Here you can see the sequence diagram for deleting a playlist. When the user wants to delete a playlist, it starts by sending a DELETE request with the id for the playlist to the API. The API then looks in the database for that id and pulls it, when the API has the data it will delete it and update the database. The API then sends a status back to the interface.



Language API

Here is the sequence diagram for /update in the language API. When the interface sends a POST request to the API it starts by requesting data from the google sheets, then it will pull the sheets and the database to check for change if there are any changes it will then post the data to the database. The rest of the diagram can be found here



6. Validation

Interviews are summarized. Full interviews can be found by clicking on a test subject's name

6.1 User Tests

6.1.1 Prototype - 1st mockup

Test Panel

The test panel consisted of five people aged 21-29, of which most were students. They all enjoy using other streaming services at the moment when doing tasks, commuting or relaxing at home. Every test subject was asked to navigate the page, comment on the general setup and functionality, their like/dislikes, and what could be done to make it overall more user friendly and appealing.

Zabina, 23, Sabbatical year before university

Steven, 21, student

Angelique, 29, Deputy Manager

Camilla, 21, Student

Jonas, 24, Student

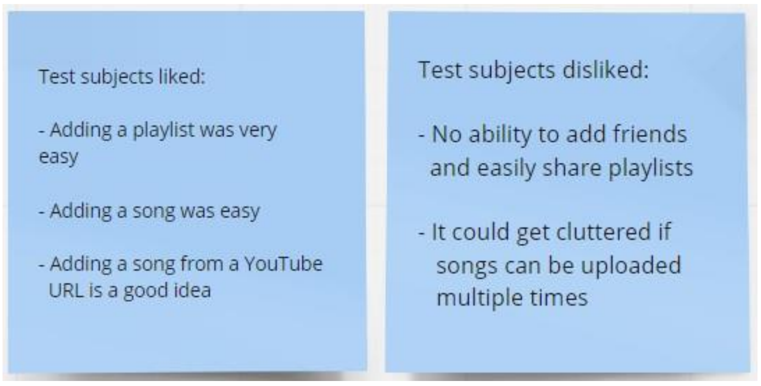
6.1.1.1 Functionality & navigation of the page

A part of the prototype testing was to explore the system on their own before being given the tasks. This simulates a natural way of using the service and contributes to making the person feel comfortable navigating the streaming service. After a bit of exploring on their own, a couple of tasks were given to the test subject. The tasks were rather simple and had the purpose of testing how easy the different functions were to find.

Find Settings: The test subjects were asked to find the settings. Most of the test subjects had in general no problem finding out how the settings are accessed. The function was generally found within 5-10 seconds of the task being given. Only one person had trouble finding settings (the person was used to settings being in the top right corner on their current streaming service). Another person was looking for settings on the home page, before eventually finding it.

Create a playlist: The test subjects were all asked to create a playlist. They all enjoyed the 2-click process and easy way of creating playlists simply by clicking the plus sign and managed to do it within a few seconds of getting the task. A couple of the test subjects commented that the way a playlist is created is rather intuitive.

Add Song: Some test subjects thought the "add song" feature only made the song available for the user and not the whole system at first. But they all enjoyed the feature and thought it was a good idea to make it possible to add youtube URLs. They also found it simple and easy to do approx. 3 clicks.


Test subjects liked:

- Adding a playlist was very easy

- Adding a song was easy

- Adding a song from a YouTube URL is a good idea

Test subjects disliked:

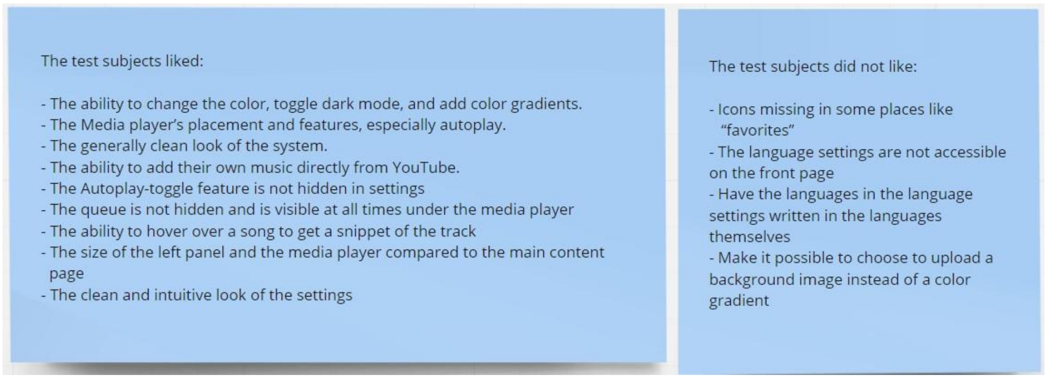
- No ability to add friends and easily share playlists

- It could get cluttered if songs can be uploaded multiple times

6.1.1.2 Appearance/Page setup

All test subjects loved the overall setup: Menu on the left, browsing in the middle, and media player on the right.

They commented on how easy it was to use.


The test subjects liked:

- The ability to change the color, toggle dark mode, and add color gradients.
- The Media player's placement and features, especially autoplay.
- The generally clean look of the system.
- The ability to add their own music directly from YouTube.
- The Autoplay-toggle feature is not hidden in settings
- The queue is not hidden and is visible at all times under the media player
- The ability to hover over a song to get a snippet of the track
- The size of the left panel and the media player compared to the main content page
- The clean and intuitive look of the settings

The test subjects did not like:

- Icons missing in some places like "favorites"
- The language settings are not accessible on the front page
- Have the languages in the language settings written in the languages themselves
- Make it possible to choose to upload a background image instead of a color gradient

6.1.1.3 Necessary improvements

Based on the user tests, it can be concluded that in order to improve our service, the following changes would be favorable:

- Add more icons, like a star next to "favorites".
-
- Make the language settings accessible directly from the front page rather than having it in the settings
-
- Block users from uploading songs that are already in the database, in order to avoid

clutter Let users add friends and easily see their friends' playlists

6.1.2 Prototype - Final mockup

Test panel: The test panel consisted of five people aged 21-27. They all enjoyed using other streaming services like Youtube and Spotify.

Nabila, 23, Student

Rebecca, 24, Unemployed

Sasha, 27, Media

Simon, 22, Student

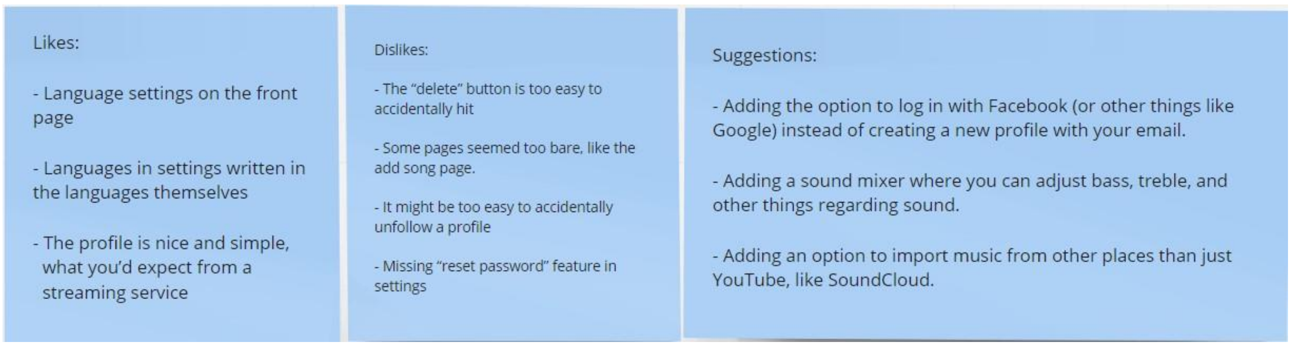
Pia, 21, Student

6.1.2.1 Functionality & navigation of the page

Not much changed from the first version in terms of navigation, except a few more features were added. As in the last test, all of the test subjects were able to create a playlist, add a song, and find the settings within seconds of being given the task. People enjoyed the way you could easily change the language on the login page and that languages were named in the language itself. The test subjects also found creating a profile easy and simple and were able to find their profile and see their followers. Some test subjects were concerned that the quality of music on youtube isn't always good and won't live up to the standards that are expected from a streaming service like this one.

6.1.2.2 Appearance/Setup

All test subjects enjoyed the service overall and thought it looked nice and modern. They all enjoyed the same things mentioned as likes in the first test and also enjoyed some of the changes we made based on the last test.



Likes:

- Language settings on the front page

- Languages in settings written in the languages themselves

- The profile is nice and simple, what you'd expect from a streaming service

Dislikes:

- The "delete" button is too easy to accidentally hit

- Some pages seemed too bare, like the add song page.

- It might be too easy to accidentally unfollow a profile

- Missing "reset password" feature in settings

Suggestions:

- Adding the option to log in with Facebook (or other things like Google) instead of creating a new profile with your email.

- Adding a sound mixer where you can adjust bass, treble, and other things regarding sound.

- Adding an option to import music from other places than just YouTube, like SoundCloud.

6.1.3 Conclusion

The first test helped the group find mistakes and points that needed improvement. After finishing and testing the final version of the prototype, the test panel had only very minor dislikes and suggestions. Overall, the prototype depicts a service that users in the demographic would enjoy streaming music from, according to the user tests conducted. If the group had been able to work on the project for a longer period of time, the final service would have benefited from adding a little more "spice" to the look of some pages, to avoid them looking too bare as well as making the delete and unfollow buttons harder to accidentally hit. To optimize the service, the streaming service would be better if it had the three suggestions mentioned by the test subjects (Facebook login, sound mixer, and music import from more places). With these changes, the service would be optimal for the desired audience, according to the answers received from the test subjects. The issues found in the first version of the prototype were also solved and no longer commented on by the last test panel.

6.2 API Tests

Automatic unit testing was problematic because testing an API that is connected to a database is difficult to do in a closed environment and the group didn't want to run a test on the public version of the API, as it would essentially add and remove test data which could prove fatal for the database if some tests went wrong.

So to verify the different API endpoints, we did some manual tests based on an expected output instead. This testing process takes a long time and is not very reliable compared to automated tests. However, as the automated approach is not an option, this will have to suffice.

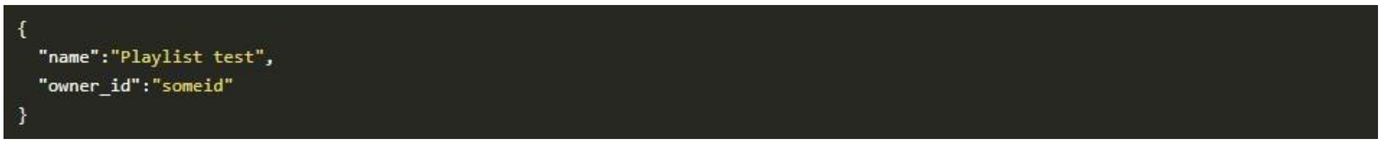
All the manual tests are done with the service, Postman.

R1 Playlist API

The playlist API is a full CRUD API and contains almost three times as many endpoints as the language API. This test is therefore going to be quite long as all the different endpoints will be tested.

R1.1 Adding a playlist

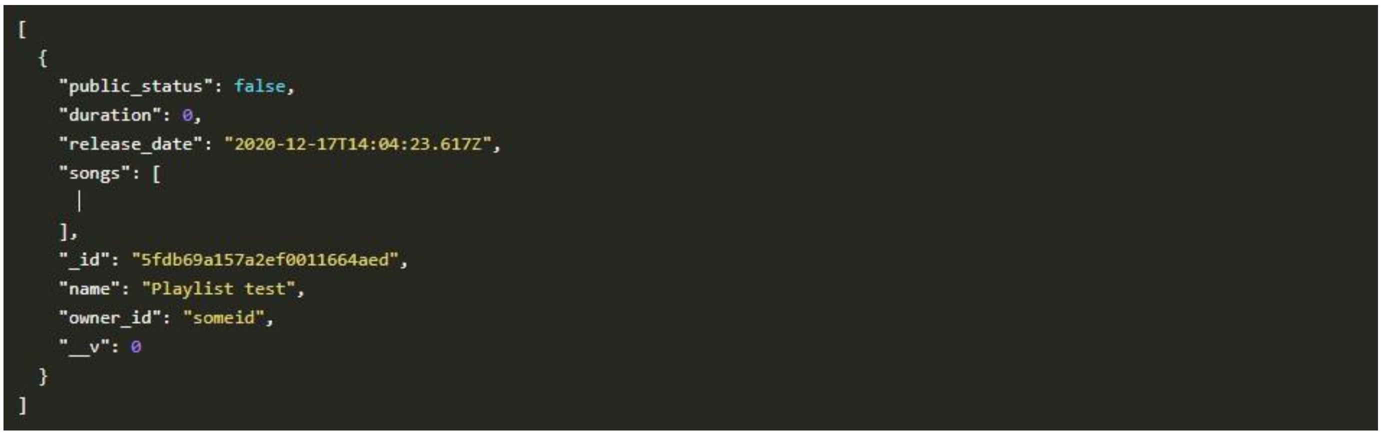
To add a playlist, we send a POST request to `http://mint.stream.stud-srv.sdu.dk/playlist/add` with the following body:



```
{
  "name":"Playlist test",
  "owner_id":"someid"
}
```

To test if the playlist has been added, we send a GET request to `http://mint.stream.studsrv.sdu.dk/playlist/` to see if the playlist gets displayed. We expect to see a playlist object with the name attribute = "Playlist test" and the owner_id attribute = "someid".

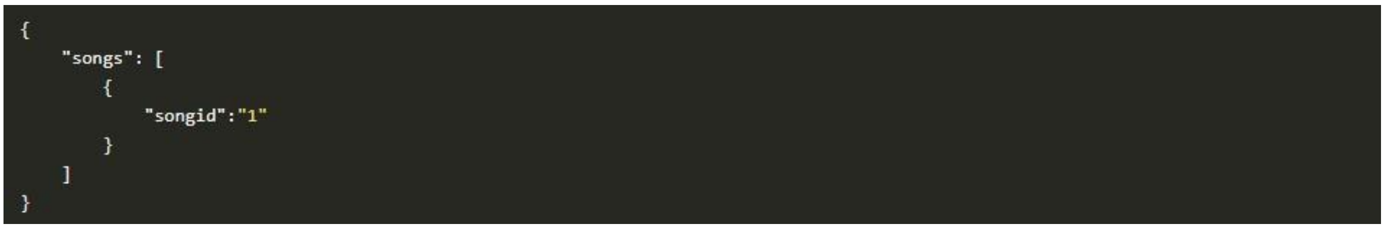
Output:



```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [
      |
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0
  }
]
```

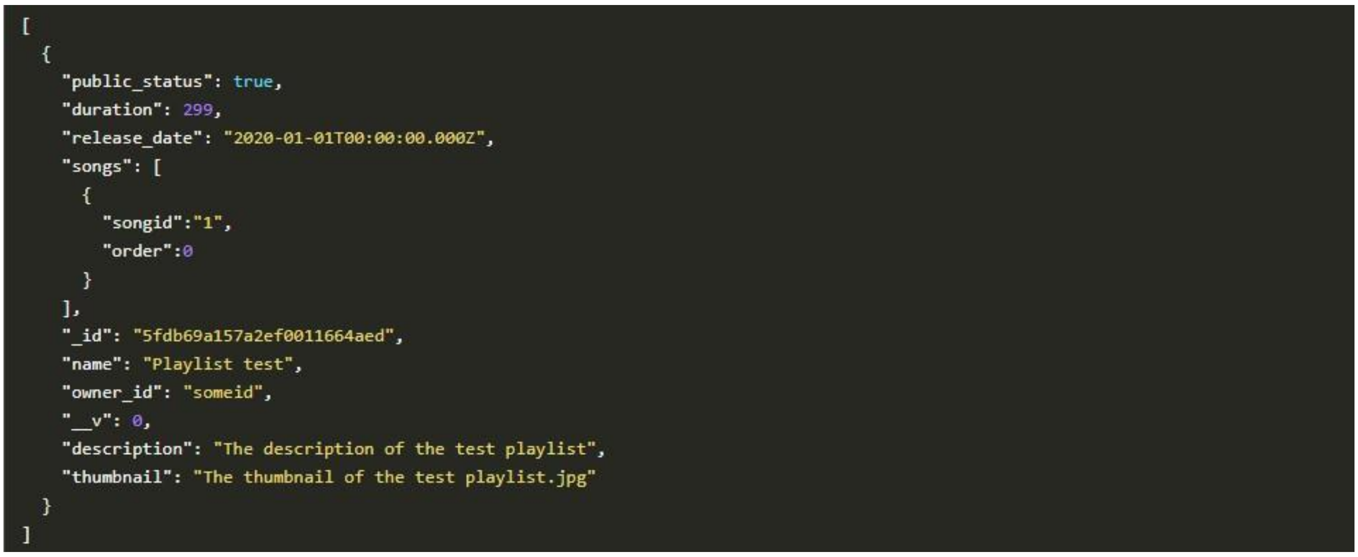
R1.2.1 Add a song to a playlist by id

To add a song with the id "1" to a playlist we will make a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/add/songs/5fdb69a157a2ef0011664aed` with the body:



```
{
    "songs": [
        {
            "songid":"1"
        }
    ]
}
```

Output:



```
[
  {
    "public_status": true,
    "duration": 299,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [
      {
        "songid":"1",
        "order":0
      }
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

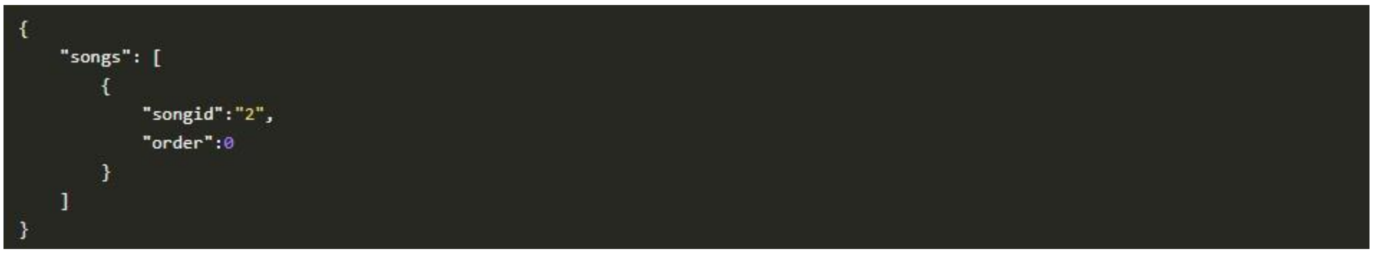
As we can see the API automatically adds an order number, to define the position of the song in the playlist.

We can also see that the duration of the playlist has been changed to the length of the song. This is done by looping over each song and making a **GET request to `http://manjaro.stream.stud-srv.sdu.dk/service01/getMusic?`**

`musicId=${songid}` where `${songid} is the songid of the song on the playlist. It then takes the SongDuration attribute for all the songs in the playlist and sums them.

We can also specify which entry that the song should be on. By default the songs will get added to the end of the playlist, however, it is possible to define an "order". If we make another **PUT** request to

`http://mint.stream.stud-srv.sdu.dk/playlist/add/songs/5fdb69a157a2ef0011664aed` with the body:

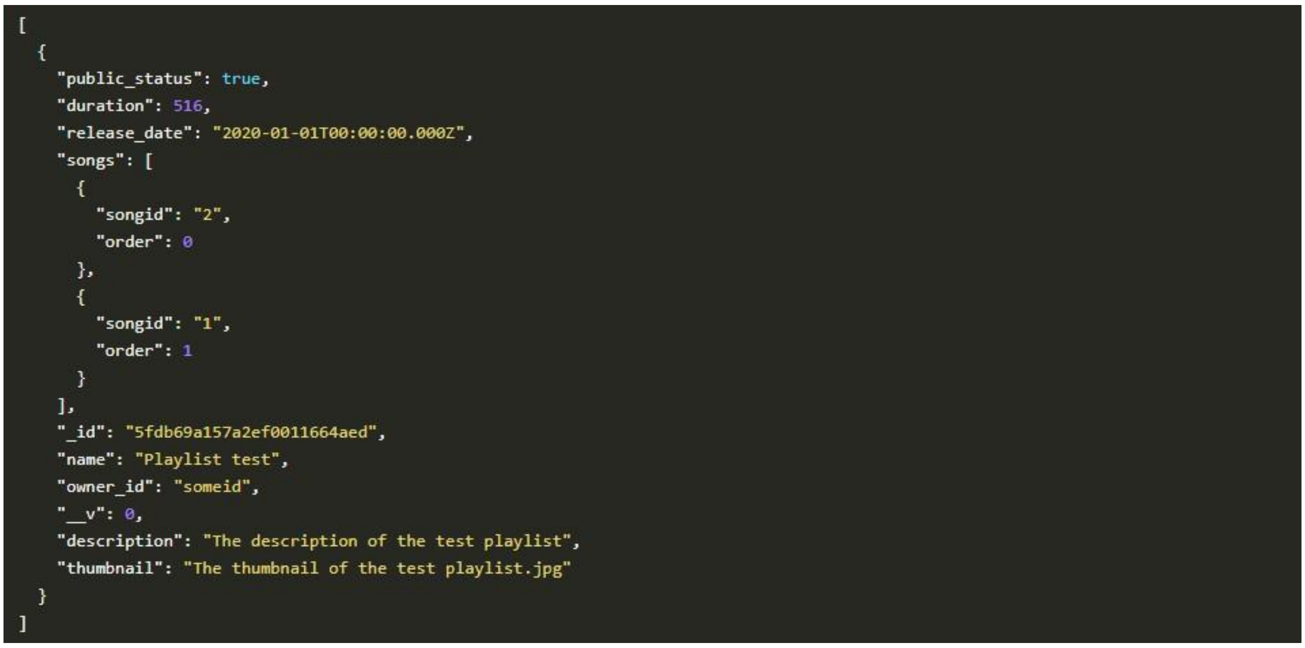


```
{
    "songs": [
        {
            "songid":"2",
            "order":0
        }
    ]
}
```

We can expect to see that the new song, with the `"songid":"2"` , gets pushed to `"order":0` while the prior

`"order":0` gets pushed down to `"order":1` . We can also expect that the duration will increase, as we are adding another song.

Output:



```
[
  {
    "public_status": true,
    "duration": 516,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [
      {
        "songid": "2",
        "order": 0
      },
      {
        "songid": "1",
        "order": 1
      }
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

R1.2.2 Delete specific songs from a playlist by id

Much like how we added a song on the test above, we will now remove a song. We will add some more songs in order to better test this. So we send a few **PUT** requests to **`http://mint.stream.studsrv.sdu.dk/playlist/add/songs/5fdb69a157a2ef0011664aed`** with different bodies like the test above.

Output:

```
[
  {
    "public_status": true,
    "duration": 2490,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [
      {
        "songid": "2",
        "order": 0
      },
      {
        "songid": "1",
        "order": 1
      },
      {
        "songid": "1",
        "order": 2
      },
      {
        "songid": "1",
        "order": 3
      },
      {
        "songid": "2",
        "order": 4
      },
      {
        "songid": "2",
        "order": 5
      },
      {
        "songid": "2",
        "order": 6
      },
      {
        "songid": "3",
        "order": 7
      },
      {
        "songid": "3",
        "order": 8
      },
```

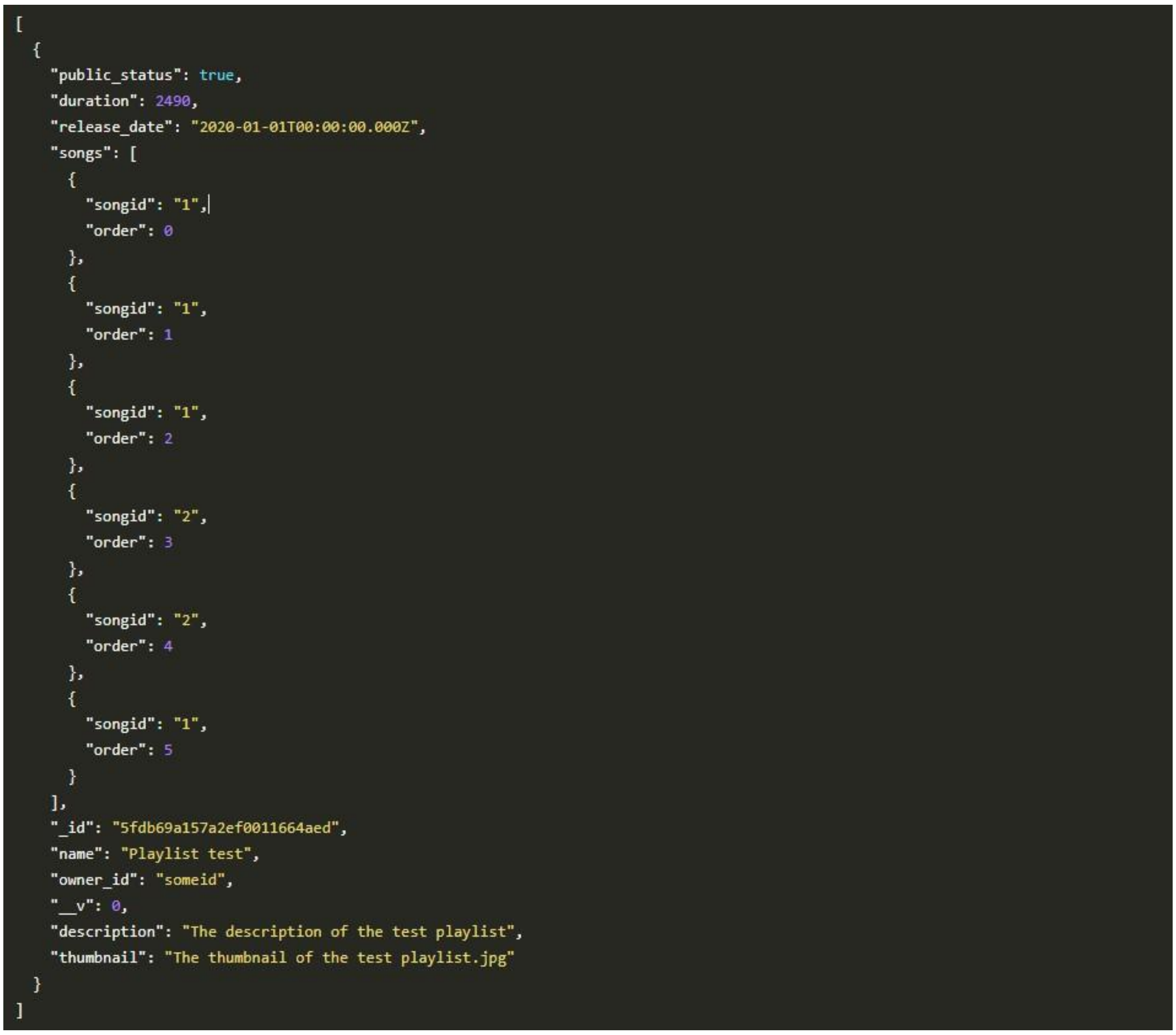
```
      {
        "songid": "1",
        "order": 9
      }
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

We can now send a DELETE request to `http://mint.stream.studsrv.sdu.dk/playlist/delete/song/5fdb69a157a2ef0011664aed` with the body:

```
{
    "songs": [
        {
            "songid":"2",
            "order":5
        },
        {
            "songid":"3"
        },
        {
            "order":0
        },
    ]
}
```

We can expect that the song on `"order":5` with the songid "2", every song that has the songid "3", as well as the song entry at `"order":0` .

Output:



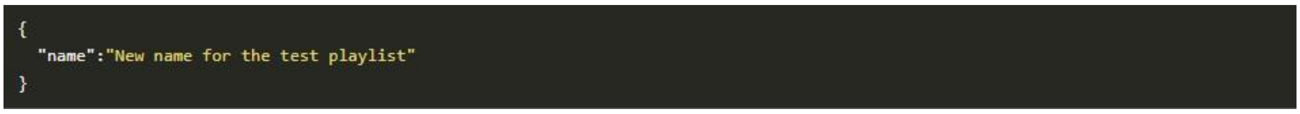
```
[
  {
    "public_status": true,
    "duration": 2490,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [
      {
        "songid": "1",
        "order": 0
      },
      {
        "songid": "1",
        "order": 1
      },
      {
        "songid": "1",
        "order": 2
      },
      {
        "songid": "2",
        "order": 3
      },
      {
        "songid": "2",
        "order": 4
      },
      {
        "songid": "1",
        "order": 5
      }
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

R1.2.3 Change the order of the songs on a playlist

By combining the add song (R1.2.1) and delete song (R1.2.2) functions, the user is able to control where each song should go.

R1.2.4 Rename a playlist

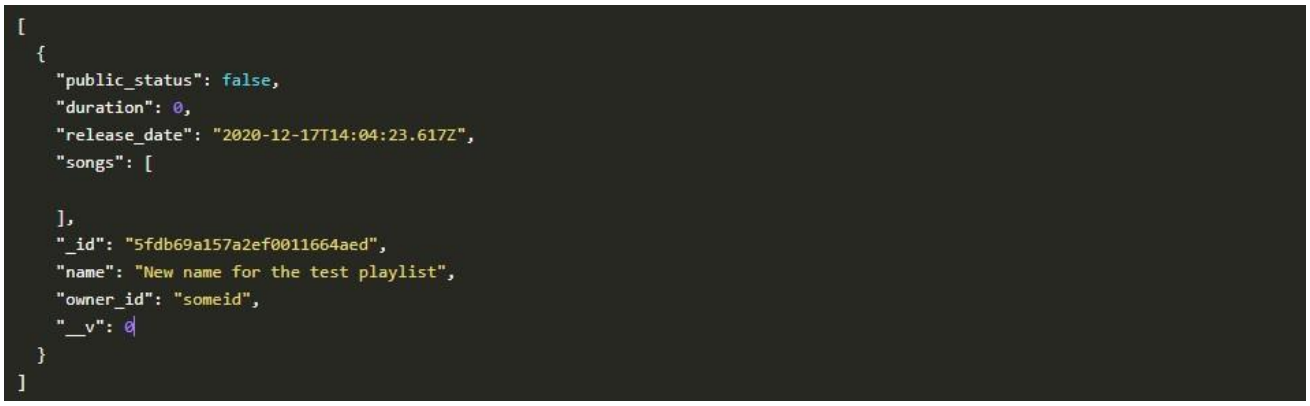
To rename a playlist, we send a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/rename/5fdb69a157a2ef0011664aed` , with the body:



```
{
  "name":"New name for the test playlist"
}
```

We expect to see the name change from "Playlist test" to "New name for the test playlist".

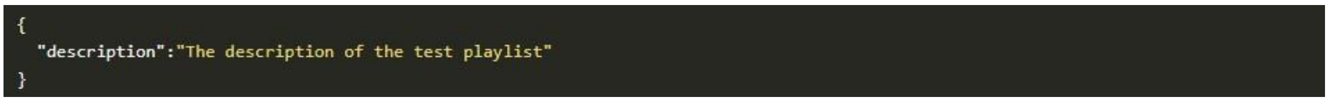
Output:



```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0
  }
]
```

R1.2.5 Update a playlist's description

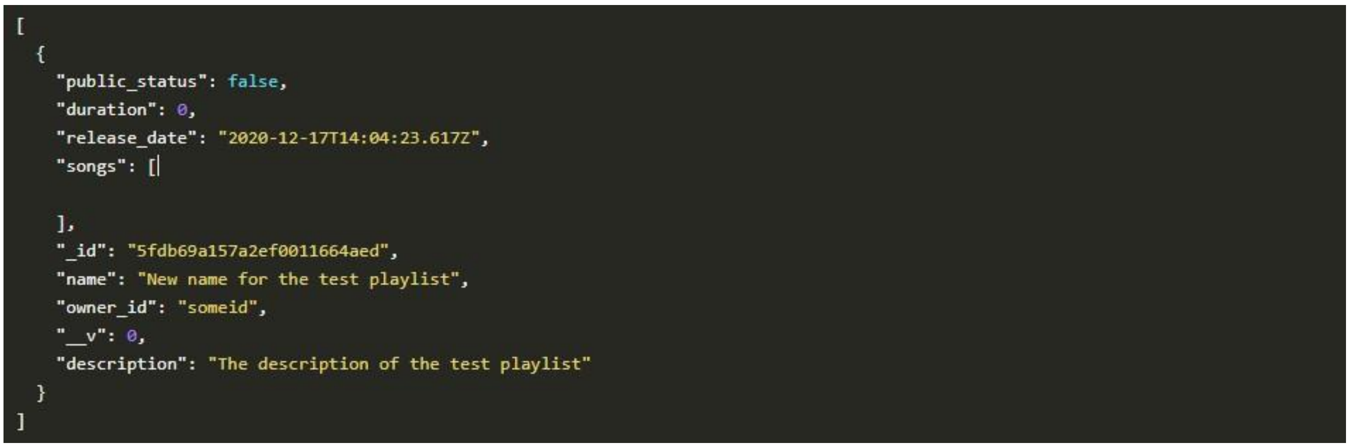
To update a playlist's description, we send a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/description/5fdb69a157a2ef0011664aed` , with the body:



```
{
  "description":"The description of the test playlist"
}
```

We expect to see the description attribute getting added to the object with the value "The description of the test playlist".

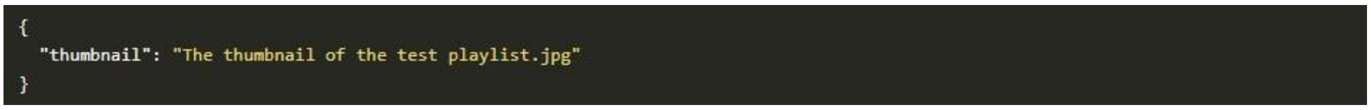
Output:



```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist"
  }
]
```

R1.2.6 Update a playlist's thumbnail

To update a playlist's thumbnail, we send a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/thumbnail/5fdb69a157a2ef0011664aed` , with the body:



```
{
  "thumbnail": "The thumbnail of the test playlist.jpg"
}
```

We expect to see the thumbnail attribute getting added to the object with the value "The thumbnail of the test playlist.jpg".

Output:

```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

R1.2.7 Update a playlist's release date

To update a playlist's release date, we send a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/release/5fdb69a157a2ef0011664aed` , with the body:

```
{
  "release_date":"01/01/2020"
}
```

We expect to see the release_date attribute getting changed to "2020-01-01" followed by a timestamp, as the output gets formatted.

Output:

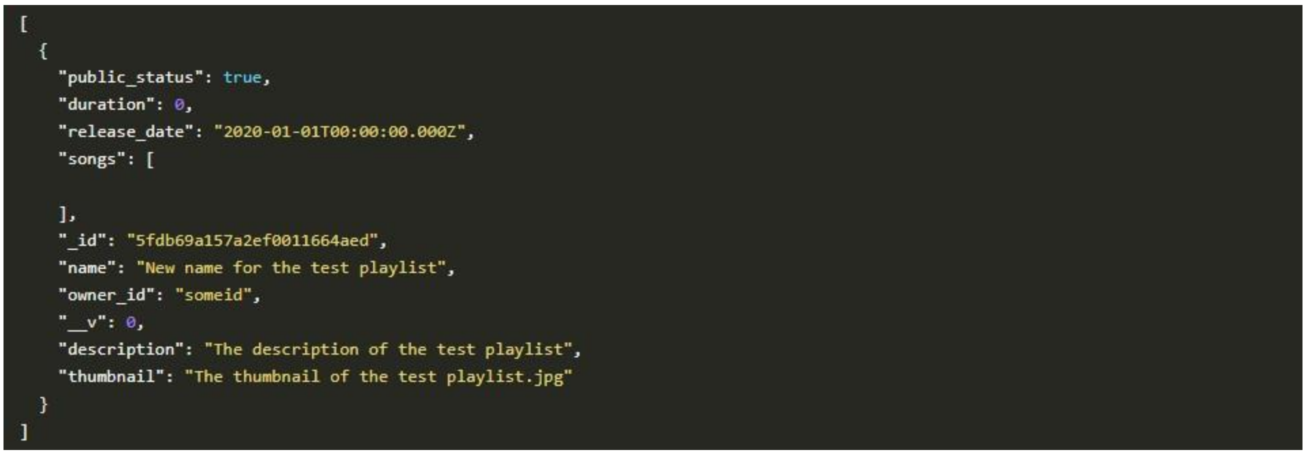
```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [
      |
    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

R1.2.8 Toggle the visibility of a playlist

To toggle the visibility of a playlist, we send a PUT request to `http://mint.stream.studsrv.sdu.dk/playlist/visibility/5fdb69a157a2ef0011664aed` .

We expect to see the public_status attribute changing from false to true.

Output:

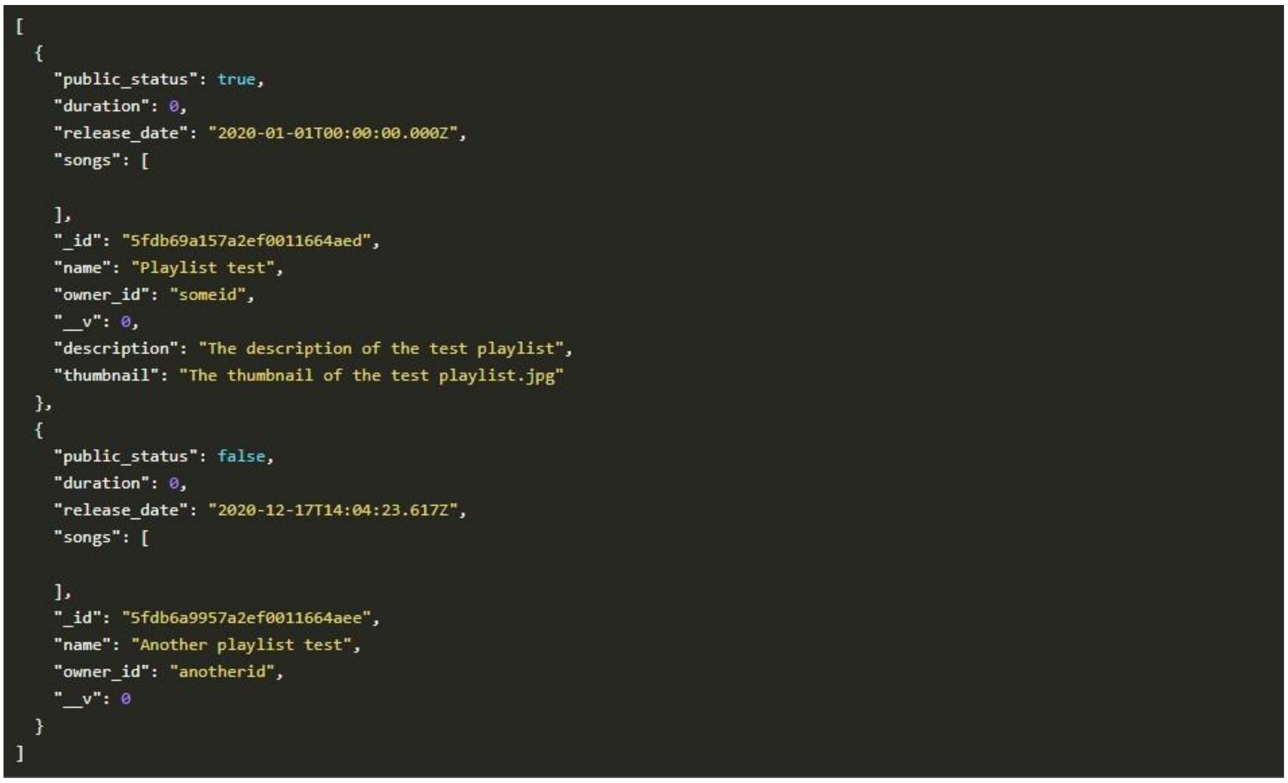


```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

R1.3 Delete a playlist by id

To delete a playlist, we will send a DELETE request to `http://mint.stream.stud-srv.sdu.dk/playlist/delete/5fdb6a9957a2ef0011664aee` . We expect that this will delete the playlist "yet another playlist test" with the id "5fdb6a9957a2ef0011664aee".

Output:



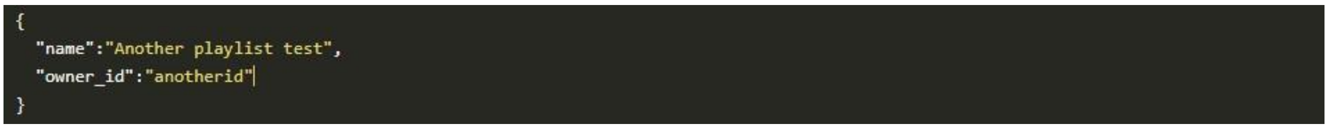
```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  },
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb6a9957a2ef0011664aee",
    "name": "Another playlist test",
    "owner_id": "anotherid",
    "__v": 0
  }
]
```

We can also delete the playlist "Another playlist test", as we only need one playlist for testing the rest of the endpoints.

R1.4 Get a list of all playlists

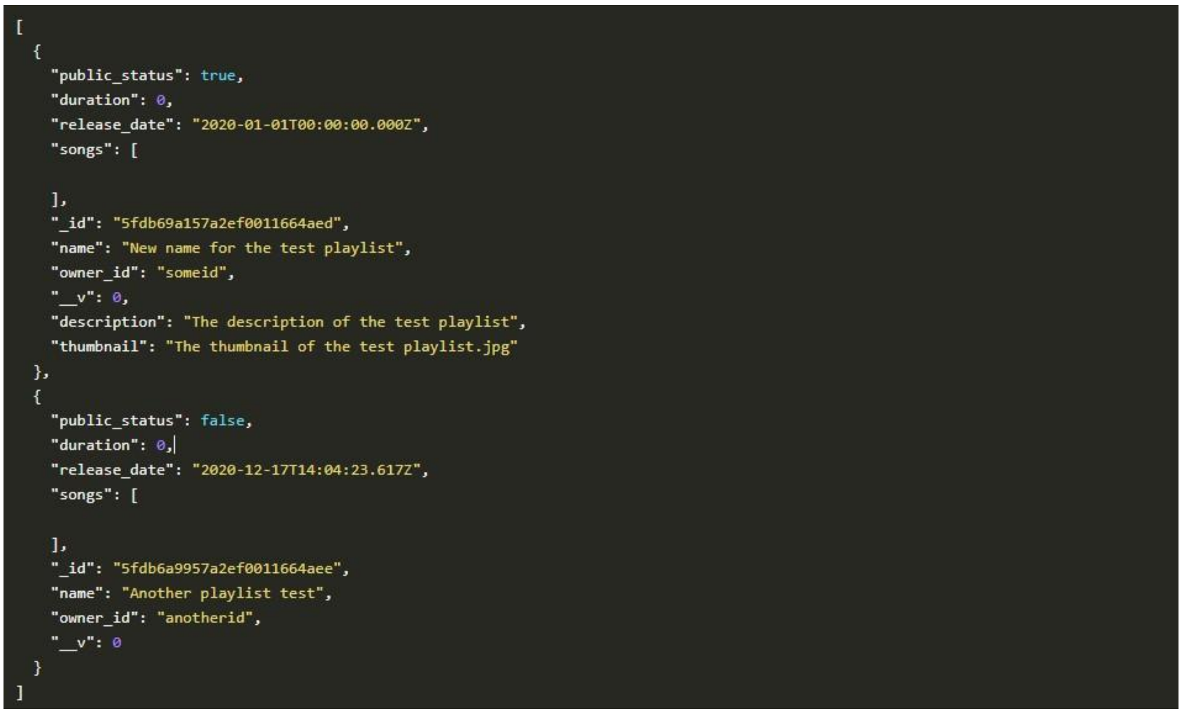
We have actually been testing this throughout the other tests as sending a GET request to the root endpoint `http://mint.stream.stud-srv.sdu.dk/playlist/` is used to display all the playlists in the database.

In order to test that it actually returns all the playlists, we will add another playlist to the database. So we will send a POST request to `http://mint.stream.stud-srv.sdu.dk/playlist/add` with the following body:



```
{
  "name":"Another playlist test",
  "owner_id":"anotherid"
}
```

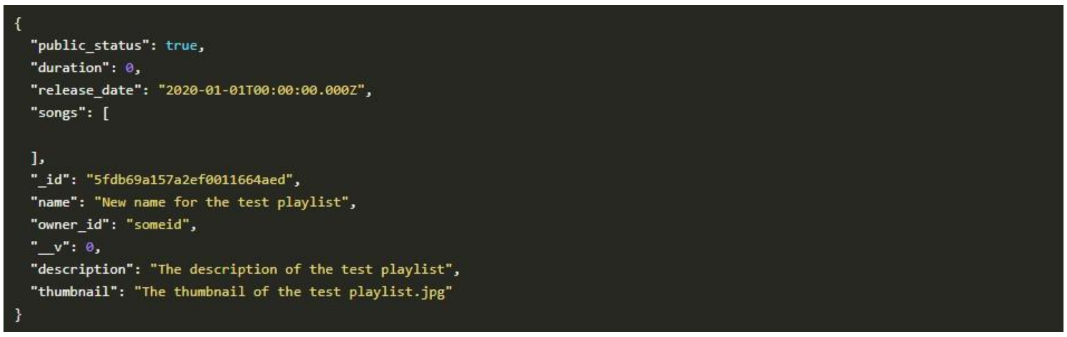
Output:



```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  },
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb6a9957a2ef0011664aee",
    "name": "Another playlist test",
    "owner_id": "anotherid",
    "__v": 0
  }
]
```

R1.4.1 Get a playlist from id

To get a specific playlist by id, we send a GET request to `http://mint.stream.studsrv.sdu.dk/playlist/5fdb69a157a2ef0011664aed` .
We expect to only see the playlist with the id "5fdb69a157a2ef0011664aed".
Output:

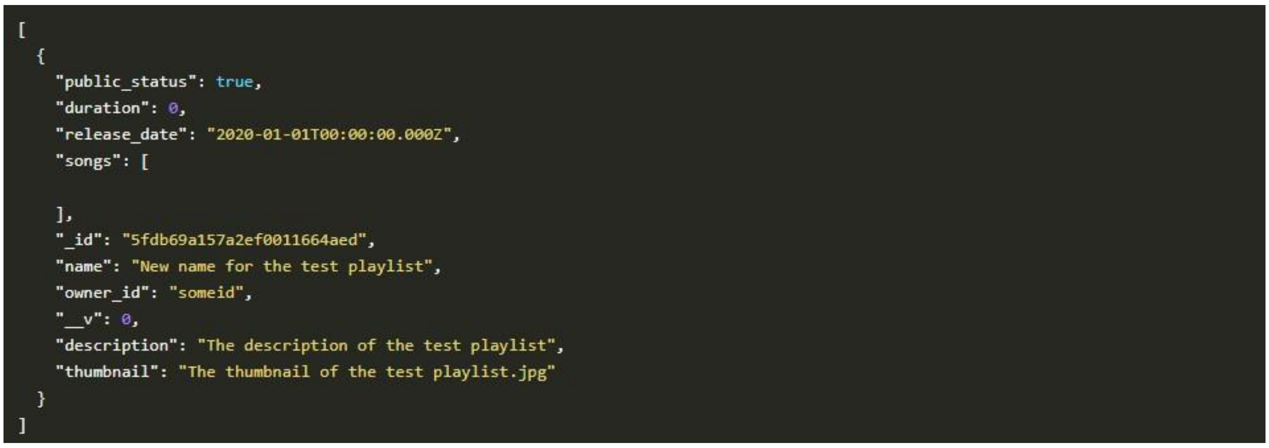


```
{
  "public_status": true,
  "duration": 0,
  "release_date": "2020-01-01T00:00:00.000Z",
  "songs": [

  ],
  "_id": "5fdb69a157a2ef0011664aed",
  "name": "New name for the test playlist",
  "owner_id": "someid",
  "__v": 0,
  "description": "The description of the test playlist",
  "thumbnail": "The thumbnail of the test playlist.jpg"
}
```

R1.4.2 Get a list of playlists owned by a specific user

To get a list of playlists owned by a specific user we will send a GET request to `http://mint.stream.studsrv.sdu.dk/playlist/user/someid` . We expect to only get the playlist with the id as it is the only playlist owned by the user with the owner_id "someid".

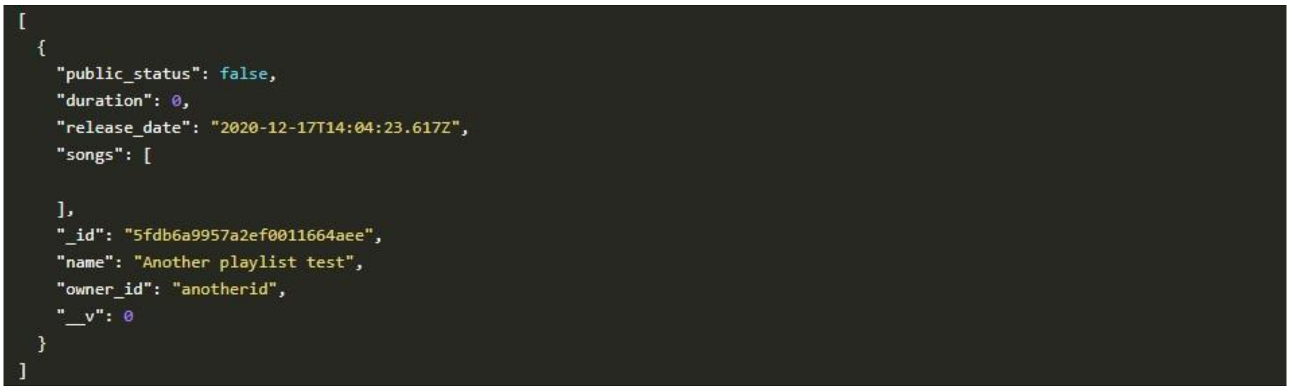
Output:



```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "New name for the test playlist",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

We can also test this by sending a GET request to `http://mint.stream.studsrv.sdu.dk/playlist/user/anotherid` in order to get the other playlist.

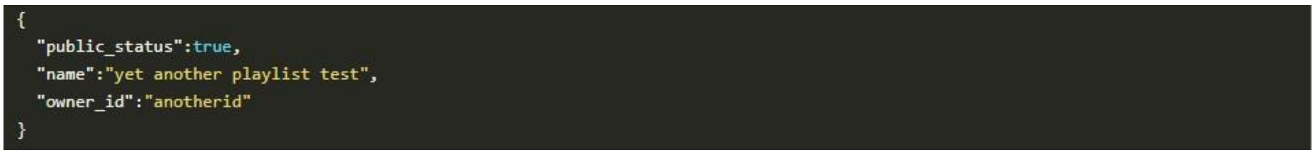
Output:



```
[
  {
    "public_status": false,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb6a9957a2ef0011664aee",
    "name": "Another playlist test",
    "owner_id": "anotherid",
    "__v": 0
  }
]
```

R1.4.3 Get a list of public playlists which names contains a specific string

This endpoint is used for searching through public playlists. We will add another playlist in order to test this. So we start by sending a POST request to `http://mint.stream.stud-srv.sdu.dk/playlist/add` with the following body:

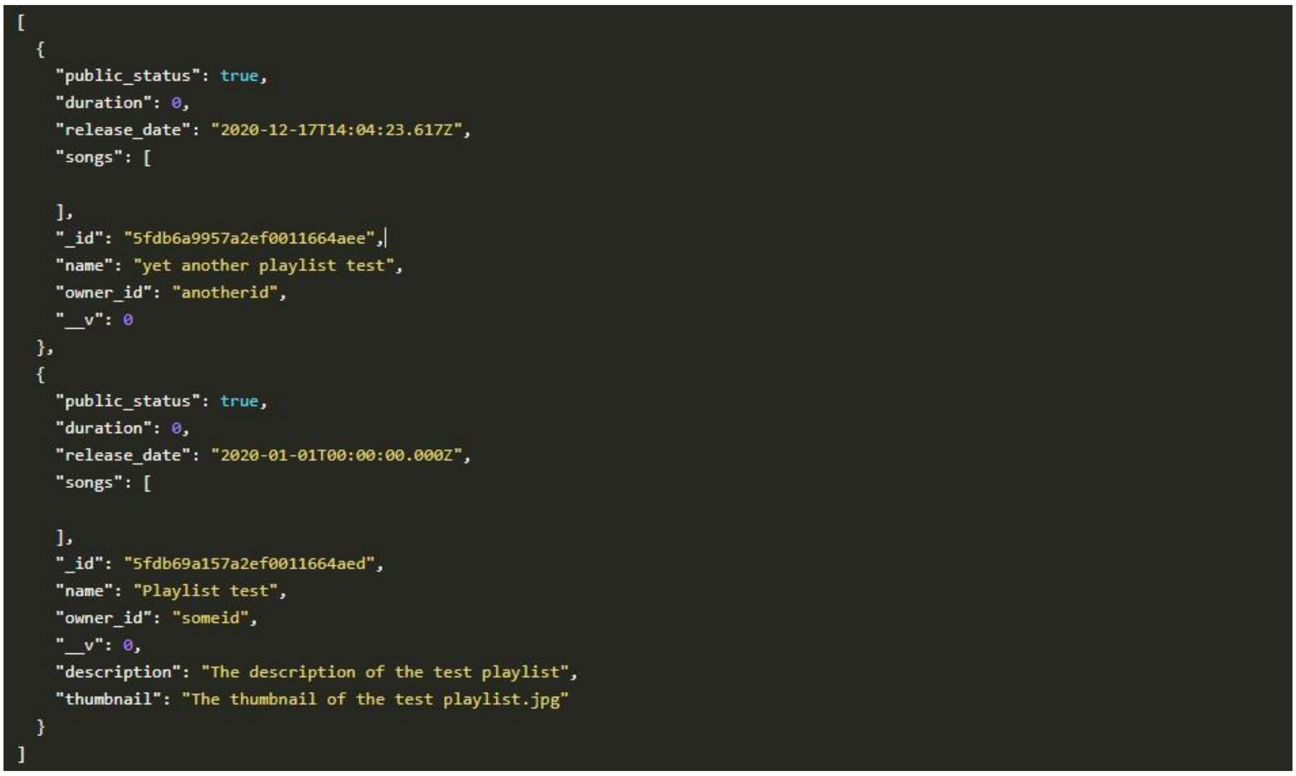


```
{
  "public_status":true,
  "name":"yet another playlist test",
  "owner_id":"anotherid"
}
```

So not we can test if both public playlists get returned if we search for "test" which both playlists have in their names.

So we send a GET request to `http://mint.stream.stud-srv.sdu.dk/playlist/search/test` .

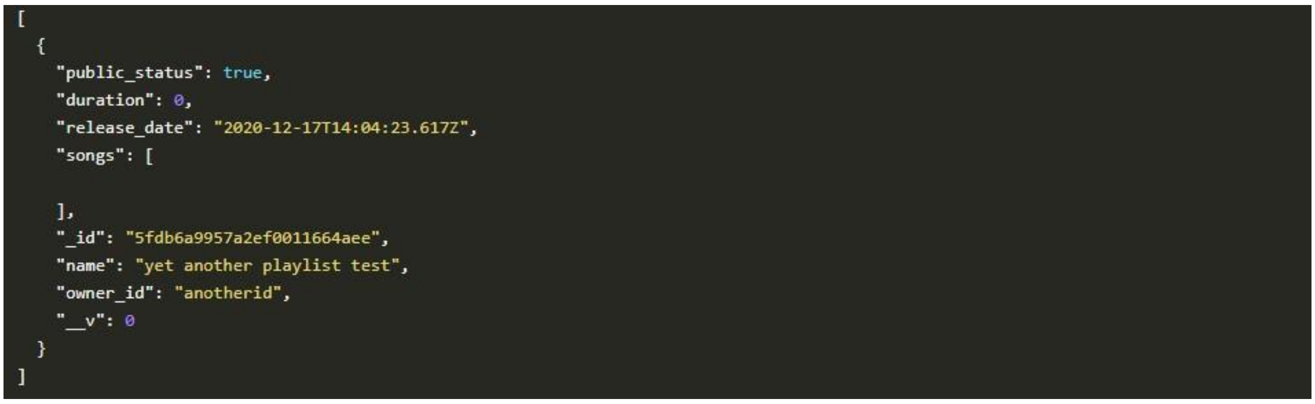
Output:



```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb6a9957a2ef0011664aee",
    "name": "yet another playlist test",
    "owner_id": "anotherid",
    "__v": 0
  },
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-01-01T00:00:00.000Z",
    "songs": [

    ],
    "_id": "5fdb69a157a2ef0011664aed",
    "name": "Playlist test",
    "owner_id": "someid",
    "__v": 0,
    "description": "The description of the test playlist",
    "thumbnail": "The thumbnail of the test playlist.jpg"
  }
]
```

And if we search for "another" which only one of them has, we should expect to only see that one.
We will send a GET request to `http://mint.stream.stud-srv.sdu.dk/playlist/search/another` .

Output:



```
[
  {
    "public_status": true,
    "duration": 0,
    "release_date": "2020-12-17T14:04:23.617Z",
    "songs": [

    ],
    "_id": "5fdb6a9957a2ef0011664aee",
    "name": "yet another playlist test",
    "owner_id": "anotherid",
    "__v": 0
  }
]
```

R2 Language API

As the language API only has a few GET methods and one POST method, the user will not be able to explicitly mess up the database. When the POST request is sent, the API updates every language in the database to correspond with the data from the Google Sheets document. If the cells are different or another language gets added, the API will update the database accordingly.

R2.1 Getting a list of all languages with content strings

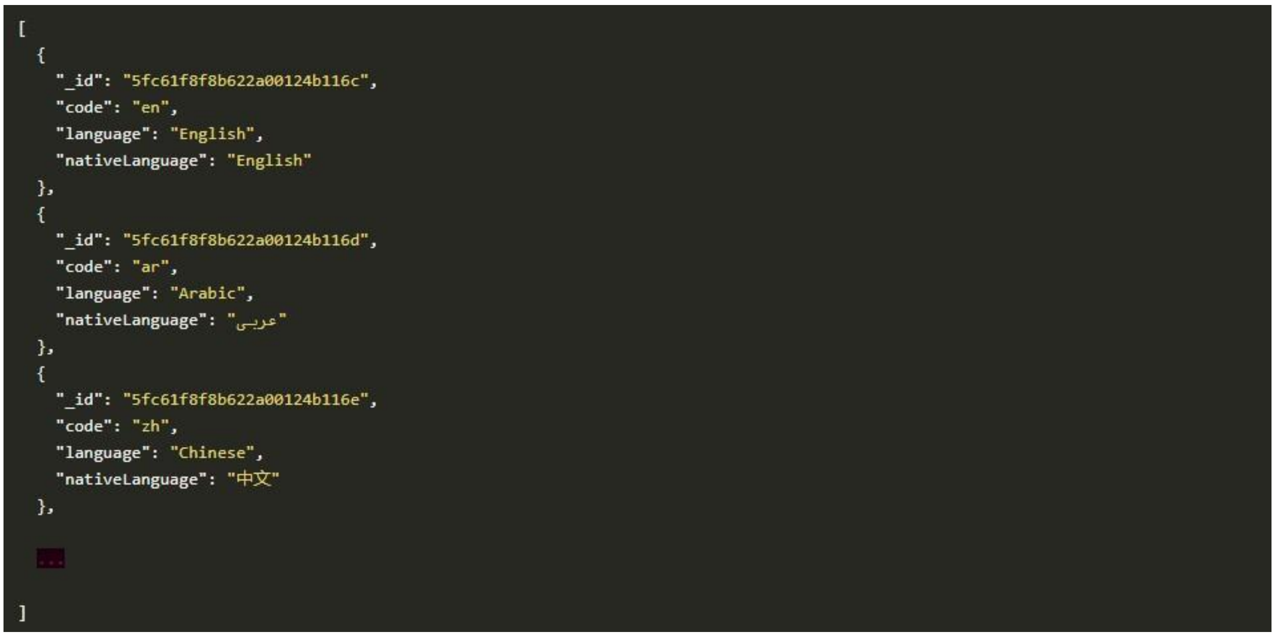
In the test above we actually confirmed that this works too as sending a GET request to `http://mint.stream.stud-srv.sdu.dk/language/` displayed all the languages in the database along with their "strings".

R2.2 Getting a list of all languages without the content strings

This endpoint is used for getting all the languages and their metadata without the strings. It's ideal for listing the languages that are available.

We test this by sending a GET request to
`http://mint.stream.stud-srv.sdu.dk/language/languages` .
We expect to see the same result as the test above, but without the "strings".

Output:



```
[
  {
    "_id": "5fc61f8f8b622a00124b116c",
    "code": "en",
    "language": "English",
    "nativeLanguage": "English"
  },
  {
    "_id": "5fc61f8f8b622a00124b116d",
    "code": "ar",
    "language": "Arabic",
    "nativeLanguage": "عربى"
  },
  {
    "_id": "5fc61f8f8b622a00124b116e",
    "code": "zh",
    "language": "Chinese",
    "nativeLanguage": "中文"
  },

  ...

]
```

The `...` at the end means that the list goes on.

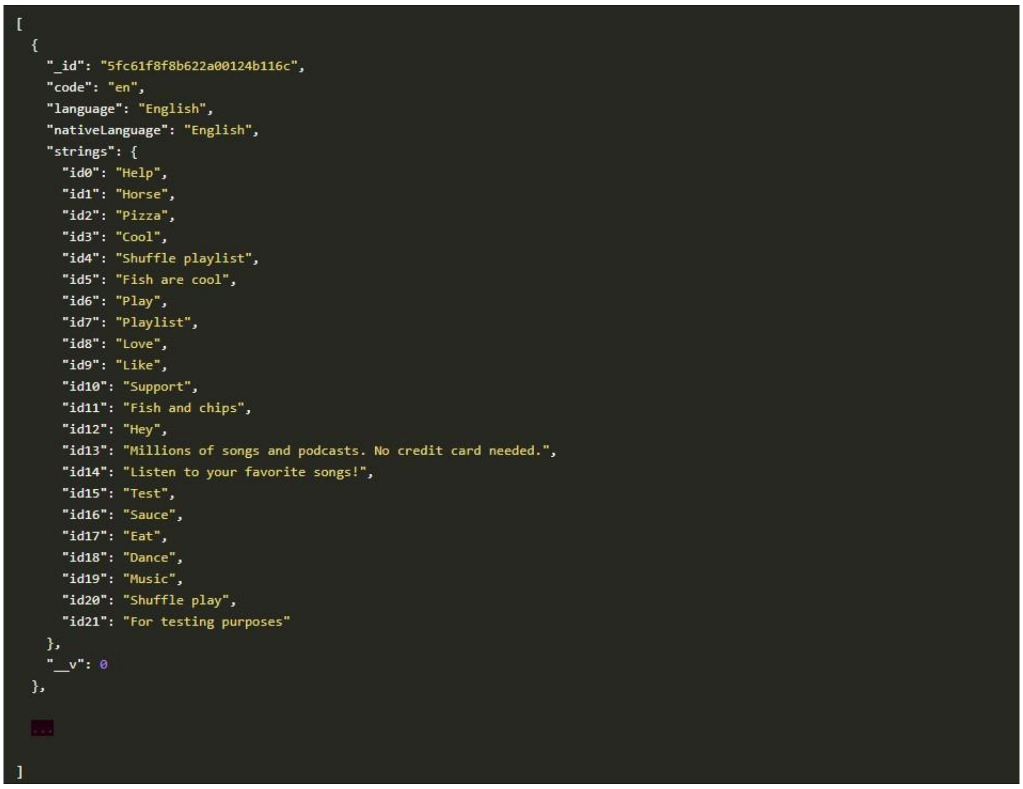
R2.3 Updating the languages in the database

We add the phrase "For testing purposes" to our Google Sheets document. To test if the language data gets updated we send a POST request to `http://mint.stream.stud-srv.sdu.dk/language/update` .

We expect to see a new entry under "strings" with the value "For testing purposes".

To see the output, we send a GET request to `http://mint.stream.stud-srv.sdu.dk/language/` .

Output:



```
[
  {
    "_id": "5fc61f8f8b622a00124b116c",
    "code": "en",
    "language": "English",
    "nativeLanguage": "English",
    "strings": {
      "id0": "Help",
      "id1": "Horse",
      "id2": "Pizza",
      "id3": "Cool",
      "id4": "Shuffle playlist",
      "id5": "Fish are cool",
      "id6": "Play",
      "id7": "Playlist",
      "id8": "Love",
      "id9": "Like",
      "id10": "Support",
      "id11": "Fish and chips",
      "id12": "Hey",
      "id13": "Millions of songs and podcasts. No credit card needed.",
      "id14": "Listen to your favorite songs!",
      "id15": "Test",
      "id16": "Sauce",
      "id17": "Eat",
      "id18": "Dance",
      "id19": "Music",
      "id20": "Shuffle play",
      "id21": "For testing purposes"
    },
    "__v": 0
  },

  ...

]
```

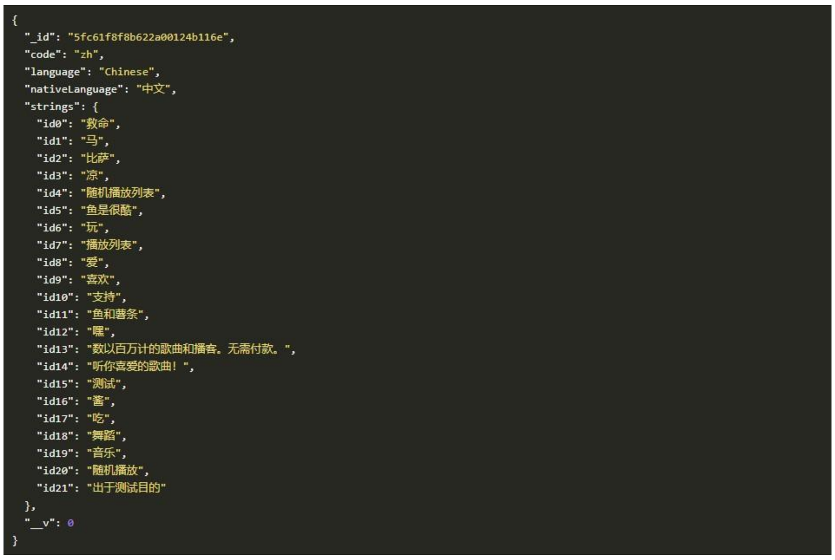
The `...` at the end means that the list goes on.

R2.4 Get language data from id

The id of the playlist is one of the unique ways that the playlist can be referenced. This means that the id can be used for fetching a specific language, instead of every language, which may require a lot of memory depending on the number of strings.

If we use the id of the Chinese language from the `http://mint.stream.stud-srv.sdu.dk/language/languages` test, we can use it to only fetch the Chinese language.

Sending a GET request to `http://mint.stream.stud-srv.sdu.dk/language/id/5fc61f8f8b622a00124b116e` , we expect to see only the Chinese language.
Output:



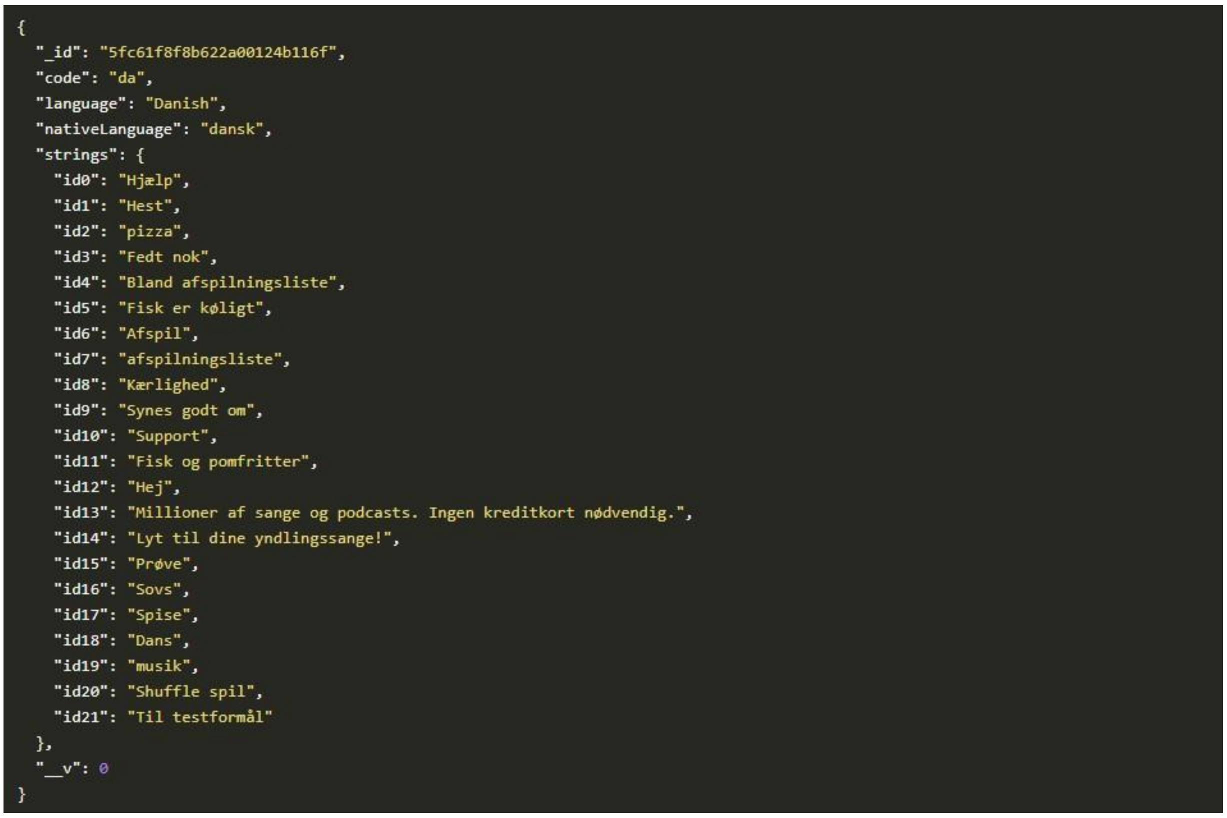
```
{
  "_id": "5fc61f8f8b622a00124b116e",
  "code": "zh",
  "language": "Chinese",
  "nativeLanguage": "中文",
  "strings": {
    "id0": "救命",
    "id1": "马",
    "id2": "比萨",
    "id3": "凉",
    "id4": "随机播放列表",
    "id5": "鱼是很酷",
    "id6": "玩",
    "id7": "播放列表",
    "id8": "爱",
    "id9": "喜欢",
    "id10": "支持",
    "id11": "鱼和薯条",
    "id12": "嘿",
    "id13": "数以百万计的歌曲和播客。无需付款。",
    "id14": "听你喜爱的歌曲！",
    "id15": "测试",
    "id16": "酱",
    "id17": "吃",
    "id18": "舞蹈",
    "id19": "音乐",
    "id20": "随机播放",
    "id21": "出于测试目的"
  },
  "__v": 0
}
```

R2.5 Get language data from the language code

Every language has a unique language code, which is also used when translating in the Google Sheets. This is another way of only fetching one language.

Sending a GET request to `http://mint.stream.stud-srv.sdu.dk/language/code/da` , we expect to see the language associated with the language code "da" (danish).

Output:



```
{
  "_id": "5fc61f8f8b622a00124b116f",
  "code": "da",
  "language": "Danish",
  "nativeLanguage": "dansk",
  "strings": {
    "id0": "Hjælp",
    "id1": "Hest",
    "id2": "pizza",
    "id3": "Fedt nok",
    "id4": "Bland afspilningsliste",
    "id5": "Fisk er køligt",
    "id6": "Afspil",
    "id7": "afspilningsliste",
    "id8": "Kærlighed",
    "id9": "Synes godt om",
    "id10": "Support",
    "id11": "Fisk og pomfritter",
    "id12": "Hej",
    "id13": "Millioner af sange og podcasts. Ingen kreditkort nødvendig.",
    "id14": "Lyt til dine yndlingssange!",
    "id15": "Prøve",
    "id16": "Sovs",
    "id17": "Spise",
    "id18": "Dans",
    "id19": "musik",
    "id20": "Shuffle spil",
    "id21": "Til testformål"
  },
  "__v": 0
}
```

7. Discussion

Going into this project we knew that it would not be possible to test the product in its entirety, since both our prototype for the design and our two microservices were implemented in another group, unlike the other projects we have worked on in the past. That said it would not mean that we did not have to test our own implementation, we just had to do it in a different way than we were used to.

7.1 UI/UX:

Overall the whole design process for the project went very well, without major problems. A Lot of the work with the design was to look at some of the already existing players on the market and see what works and what does not work. This gave us a lot of inspiration for how to design our system. One of the things that helped the group a lot throughout the process was that we went through the exact same process in the course Human-Computer Interaction which meant that if we had some questions that the supervisor could not answer, we could get extra input from the course. One of our key focuses was to communicate with the other groups on how some of the features should work on the UI, an example of this, Connection Security requested and keep me signed in button on the login page for security purposes which lead to a discussion on how we should design that feature with the user in focus while still keep the security high. After some talking between the groups, we all came up with the solution to have a remember me a button that should remember the user's username.

7.1.1 User Tests

The first user test, where version one of the prototype was tested, went incredibly well. Interviews were carried out back to back and the test subjects were interviewed by the same person while the rest of the group took notes. This resulted in a lot of consistency in the interviews as well as very thorough notes. However, when it came to the final version of the prototype, the interviews could not be carried out in the same way, due to the covid lockdown and the group no longer being allowed to meet on campus to work on the project. Therefore, the group members split up and interviewed people individually which resulted in a lot of inconsistency, as not everyone had completely followed the interview script and asked the same questions. The script had also been a bit rushed because of this sudden change. However, the feedback was still usable but if it had been possible, the results would likely have been much better, had the group been able to carry out the interviews as done in the first user test.

7.2 Language API

The language API uses google translate's API to set up a google sheets spreadsheet where all the different words are translated automatically. Using google sheets adds the ability to send a GET request and get the full sheet as a CSV file to use in the implementation. A major struggle during implementation was figuring out how to make sure that the database was up to date when pulling the CSV file. Solving this was a matter of trial and error. At the beginning of the implementation, the group experienced a steep learning curve due to a lack of experience regarding node.js, APIs, and microservices which led to a slow start but research and a lot of trial and error helped the group better understand how APIs and microservices work.

7.3 Playlist API

When implementing the playlist API, the main issue that the group faced was adding the playlist duration. In order to do this, it was necessary to communicate with the other groups' microservices in order to fetch the duration of each song. This was something the group struggled with quite a bit, especially due to the fact that it did not work when trying it locally but would work when pushed to GitLab. During implementation, it was necessary to push every time in order to test if it worked. Another obstacle the group faced was figuring out how to update the playlist duration when deleting or adding songs to a playlist. However, the problem was solved in the end but perhaps it would have been easier if communication between the groups had been better and structured in some way throughout the project.

7.4 Further Work:

Had it been possible, the prototype would have benefited from a few tweaks, so that the final product would be the best possible streaming service. The test subjects in the last user test made a few suggestions, these can also be seen under Validation. If there had been more time to perfect the product, the group would have liked to make some of the pages less bare by adding more details to them and improving the overall look, as test subjects commented on the fact that some pages, especially the "add song" page, looked way too bare which was unappealing. Furthermore, some minor changes to the placement of buttons like "unfollow" and "delete" would have been good, as the test subjects commented on these being too easy to accidentally click.

In addition, if it had been possible to continue working on the streaming service, making it possible to import music from more platforms than just YouTube would have been good. For example, a platform like SoundCloud. This is a platform where many users upload their own music and adding the option to import content from this platform to ours would go well with the idea of making it easy to find new artists as well as making it easier for creators to share their own content more easily.