SongSwipe

Christopher Coco, Nena Heng, Raj Ray, Anthony Simao, Katiana Sourn

April 28, 2025

Abstract

SongSwipe is a web application that allows for an easier, quicker, and more user-friendly experience of managing a Spotify playlist. It allows for a simple Tinder-inspired swipe left or right motion to decide whether a song will remain in your playlist or be removed. Previews of the song will be played automatically. Not only can a user easily remove unwanted old songs from their own playlists, but also filter out songs from a copy of another user's playlist to make it more personal to the user.

1 The Problem

With over 250 million users and 31.7% of the music streaming platform market share [2], Spotify [3] has revolutionized how we access and enjoy music. Over the years, Spotify's user interface has gone through numerous changes and redesigns, many of which have been met with frustration. One of the most notable changes was the remapping of playlist functions and buttons, leaving many users confused as they felt there was no need to fix what wasn't broken. Despite its unpopularity, the change remains in effect to this day, leaving the process of managing playlists a cumbersome one.

Over time, playlists tend to become bloated with songs that were once enjoyed by the user but are now often skipped over. Many users find themselves wanting to update their playlists. Unfortunately, Spotify offers no easy or convenient solution to do this, and removing unwanted songs from a playlist is a tedious process that gets more frustrating the more times a user has to do it. The steps are:

1. Click the playlist button in the bottom right corner.

   → This opens a list of all playlists the current song is in.

2. Uncheck the playlist you want the song removed from.

   → This will remove the current song from that specific playlist.

3. Click the done button in the bottom center.

   → This will return you to the now-playing screen.

4. Swipe to the next song and repeat.

   → Repeat until all unwanted songs are removed.

With each song removed costing 3 button presses, removing 100 songs would require at least 300 button presses in combination with swiping between each track. In combination with the time requirement of listening to song introductions, the process is both time-consuming and frustrating. This single design flaw further tarnishes the overall user experience, as users end up skipping songs repeatedly. Spotify also allows users to copy existing playlists created by other users to their own library. Yet many people do not use this feature, as there is no way to easily remove the songs that they don't like from the copied playlist.

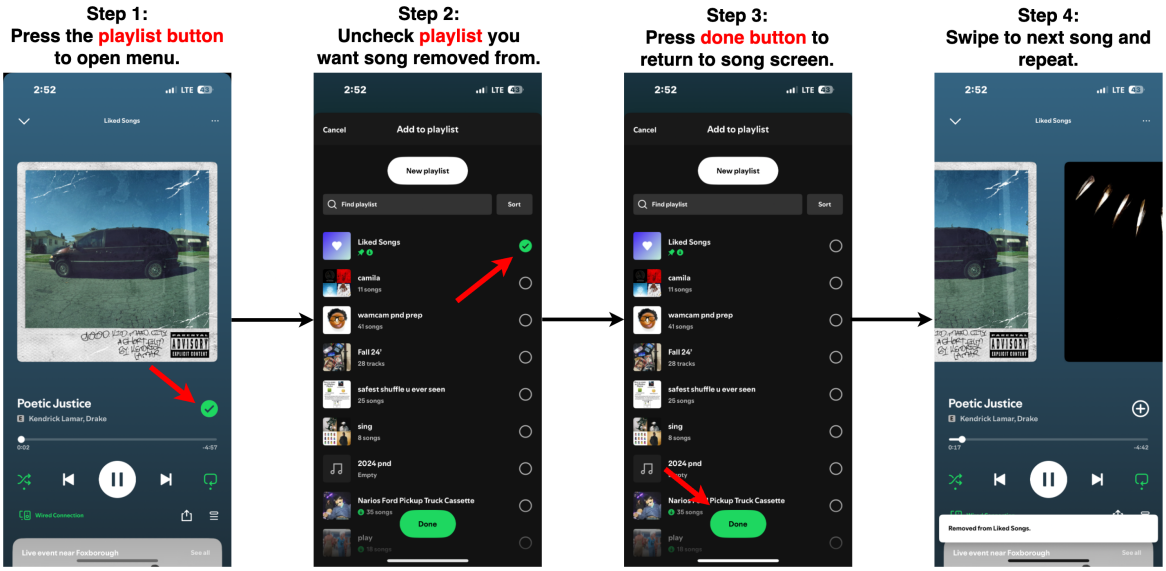


Figure 1: The steps it takes to remove a song from a playlist.

2 Current Solutions

Many Spotify users struggle with the platform's limited playlist management features. People change, and so do their music preferences. Spotify's inefficient platform makes it difficult for users to keep playlists organized and up to date.

Spotify's platform is lacking in playlist management, such as efficient revision of playlists. Some users end up creating a new playlist of what they currently like, as it is seen as a better alternative, which causes users to have a cluttered Spotify library filled with many playlists. While Spotify's built-in playlist management is limited, several external tools have emerged, such as Spotify Dedup, Playmixify, Swipefy, and Skiley.net.

2.1 Spotify Dedup

Spotify Dedup is a web app that analyzes your library and removes duplicates. It uses the Spotify API and links to access your account. It's simple UI shows your playlist that contains duplicates, details of the artist and song, and then the option to remove duplicates from your playlist. [8] The web app Spotify Dedup is successful as it is very direct about its approach - simple UI and simple function as seen in Fig. 2. On the other hand, that is all it will do. What makes our product better than Spotify Dedup is that we will allow you to not only remove duplicates, but also have an interactive UI that makes managing a playlist less tedious and more entertaining.

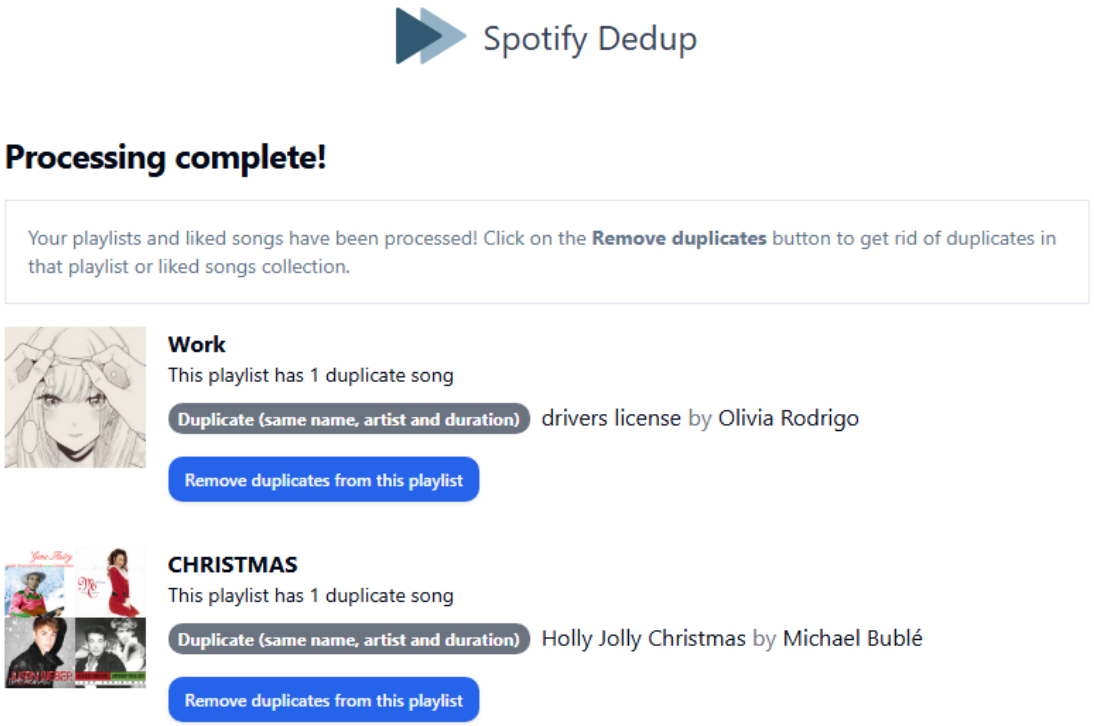


Figure 2: The Spotify Dedup desktop GUI.

2.2 Playmixify

Playmixify is a web application that "mixes your Spotify playlists into an ultimate playlist and keeps them in sync with the original playlists" [3]. This application allows you to combine playlists from your library into one massive playlist as seen in Fig. 3. It uses the Spotify API to view your library and make a playlist from existing songs in your library. Like Spotify Dedup, it is very direct and simple in its approach, but just like Dedup, that is all it does. Our product is distinguished from other external tools by offering a more interactive experience with a wider range of playlist management features, going beyond just creating or deleting specific types of playlists.

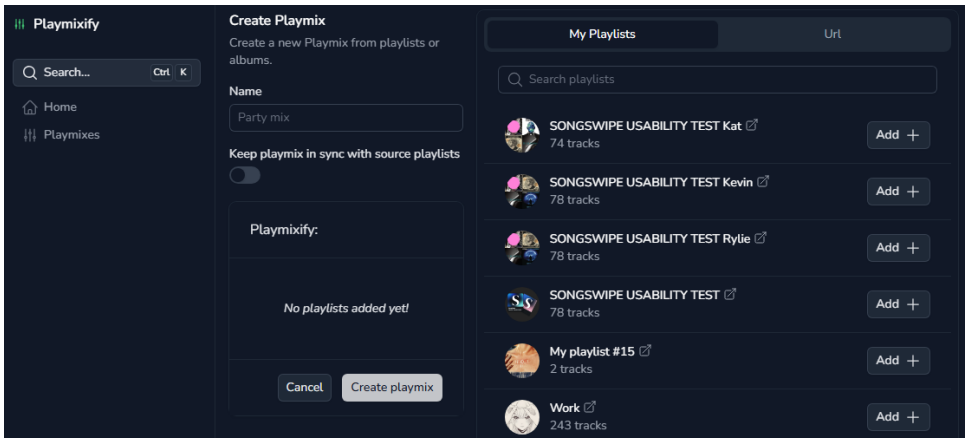


Figure 3: The Playmixify desktop GUI.

2.3 Swipefy for Spotify

Swipefy for Spotify is a mobile app available on IOS and Android that uses the Spotify API to introduce Spotifyś consumers to new music using an AI algorithm and allows them to seamlessly add these songs to their playlists with an interactive UI - where users can Tinder-Swipe these songs on their app and into their Spotify playlist. [14] Swipefy's main objective is to introduce songs that the consumer may like based on filters that you can access in the UI and based on similar songs found in a playlist that the consumer wishes to add to, as seen in Fig. 4. The Swipefy for Spotify is successful in its approach by making an interactive way to allow Spotify consumers to discover new music that is curated for them by using the information found in their profile and playlist.

One issue with Swipefy for Spotify is that it is only available as a mobile app. Our team believes that anyone with access to Spotify, regardless of their device or operating system - should be able to use our tool. Therefore, to ensure access to all Spotify users, our application is not only accessible on iOS, but on most devices with a web browser.

Figure 4: Samples of Swipefy GUI.

2.4 Skiley.net

Skiley.net is a platform that essentially extends the Spotify UI by including numerous features such as song discovery and playlist management. Furthermore, using the Spotify API they replicate the Spotify UI by showing what song you're listening to, playback history, and the lyrics, and extend their UI by including additional features such as providing music videos, and similar song recommendations based on the current song you are listening to, and facts associated with the song. They also offer playlist management via deduplication, which removes duplicates found in the playlist. [7]

Although Skiley.net extends the Spotify UI in terms of additional information (Fig. 5), they limit playlist management to 3 edits of any of the free service options they provide and offer more services behind a one-time subscription. As Spotify requires consumers to pay a monthly subscription fee to remove ads when playing music, it would seem redundant to purchase an external Spotify tool for playlist management and to remove ads from their platform. Furthermore, they provide different ways to organize your playlist, but there is still a lack of an efficient playlist revision tool.

SongSwipe has its main focus on revising the playlist revision process to reduce the amount of time it takes to revise a playlist, which, although Skiley.net has other amazing features, it does not offer the same service we provide.

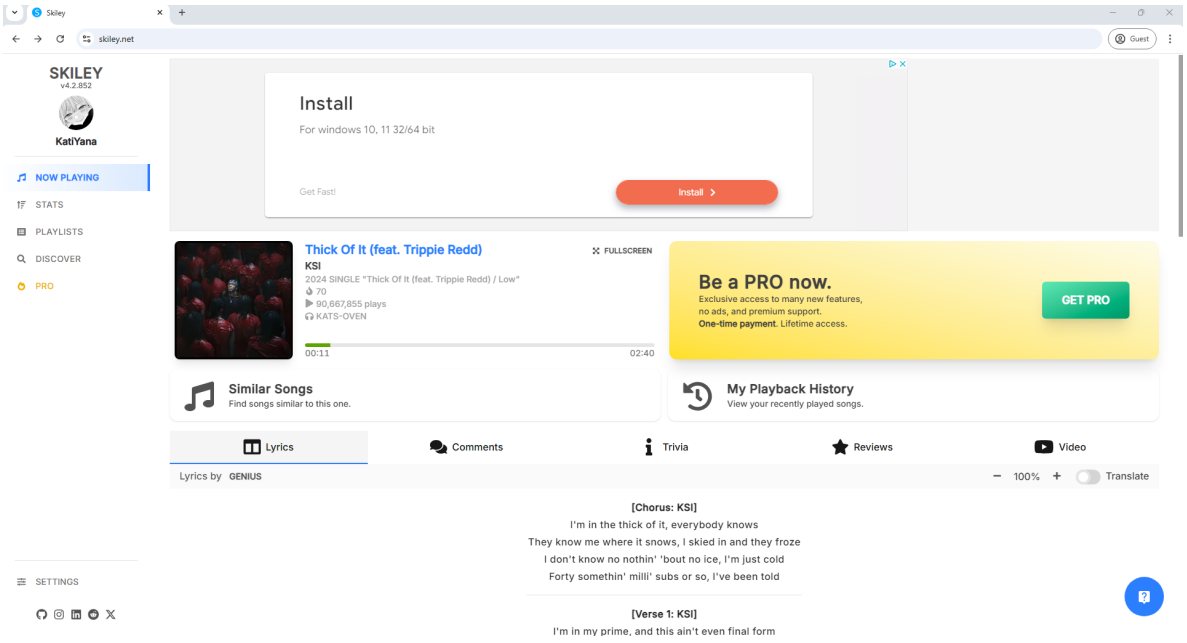


Figure 5: The Skiley desktop GUI.

While Spotify is a major platform for music streaming, its playlist management features fail to meet the needs of its users. The current process for managing playlists can be tedious, leading users to create new playlists repeatedly or seek external solutions. Tools like Spotify Dedup, Swipefy for Spotify, Skiley.net, and Playmixify have attempted to address these shortcomings by leveraging the Spotify API to provide additional functionalities such as deduplication, song discovery, and playlist synchronization. However, these tools focus on niche aspects of playlist management and lack of comprehensive and interactive experience that users desire. Our solution will be a discernible tool against other external Spotify tools by offering a free, accessible web app that not only simplifies playlist management but also makes the process more engaging and user-friendly.

3 Our Solution

The function of our application mainly revolves around managing playlists on Spotify. The user signs into their account so that the application has permission to access user information, such as their

playlists. This also grants the application permission to manage the user's playlists as well. After selecting a playlist, the user will be brought to a screen that resembles the interface of dating apps. SongSwipe's main screen has some buttons for playback functionality as well as information about the song, including: song title, artist info, and album cover art. When a song is displayed, it will start playing. The user can swipe left on the song if they don't like it, and right if they do. The swiping motions make it easier to select which songs you want to keep or remove.

The two playback buttons control both playing and pausing, as well as restarting the current song's preview. There is also a button to go to the staging area, where all changes live before being committed to Spotify. Each decision the user makes is tracked and saved to the staging area, which shows all changes that have been made to the playlist. During or after a user is done going through a playlist, they can see an overview of their changes in this area and then choose to revert any recent changes they have made. After reviewing the staging area and confirming the changes, the application automatically updates the user's Spotify playlist.

Another major function of our application is copying playlists from other users. Currently, there is no easy way to copy another user's playlists on the mobile version of Spotify. Our application improves user satisfaction by allowing consumers to search for a user to copy a playlist from or providing the playlist URL if the playlist is public. Once the consumer copies a playlist, they can use the functions just described to help manage the playlist and adjust it to their liking.

Our solution differs from the existing solutions through a highly navigable and intuitive user interface. The UI is inspired by dating applications like Tinder. This single swipe replaces what would be three separate button presses on the Spotify app. It reduces room for error and decreases the amount of input/time that is needed to remove a song from your playlist.

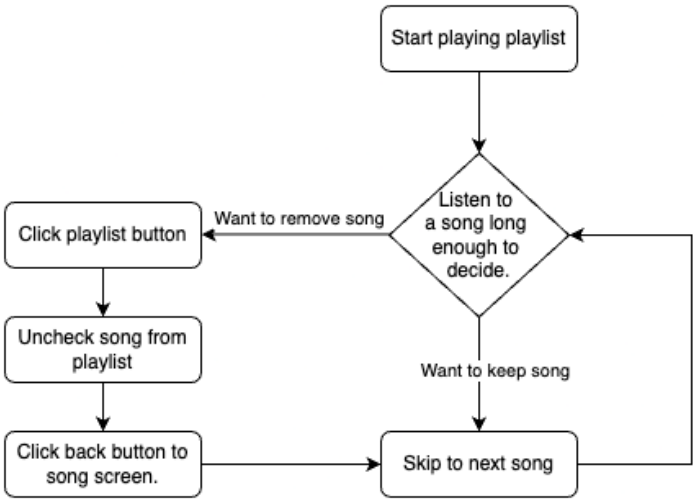
Being able to play the song's preview before deciding whether to remove or keep it is another major improvement that our app achieves. Normally, a user would have to listen to the song from the beginning. However, sometimes songs can have a long introduction before they get to the part where you would be able to determine if it's worth keeping. The preview for a song on Spotify showcases a part of the song that could be considered the best part of it, and playing this preview helps Spotify consumers determine whether they like the song or not in a time-effective manner, rather than having them listen to the whole song from start to finish. This feature helps Spotify consumers manage their playlists in an optimal manner.

Furthermore, a feature that makes SongSwipe distinguishable from other external Spotify tools is the ability to copy a friend's playlist, go through the songs in that playlist, and filter them. Currently, the only way to copy another user's playlist on Spotify is to either save it or use the shortcut CTRL + A, then copy all of the songs to your playlist. There are two major issues with these solutions. For saving, this is not creating a copy of the other user's playlist, but more of saving their playlist to your library so you can easily have access to it. You cannot make changes as well unless they have invited you as a collaborator. The issue with selecting all shortcuts is that it is only available on the desktop Spotify client and not on mobile. This is significant because the majority of Spotify users are mobile users, listening on the go, and this limits people who primarily use Spotify on their mobile devices. Our solution copies the songs from a user's playlist into your own so you can use our management features to keep and remove songs. Overall, these two major features help improve in two places where Spotify falls short. The decision flowcharts for our solution and Spotify in Fig. 6 show that unneeded redundancy is removed.

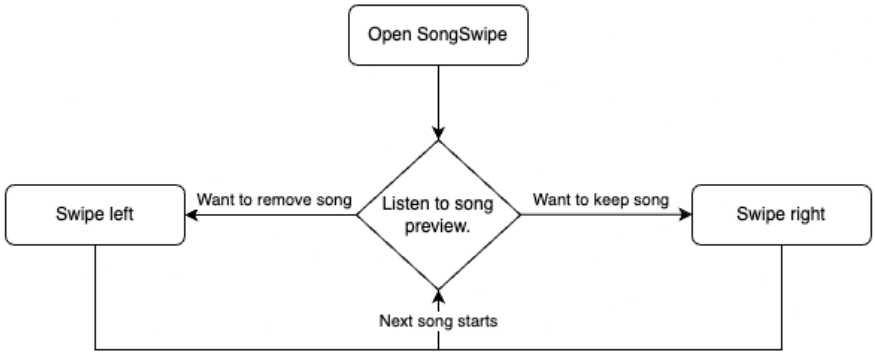
4 Project Architecture

4.1 Application Architecture

The overall application architecture can be seen in Fig. 7. The three major components of our application are the user interface, the Spotify API, and a JSON file. The user interface is the core of our application and will perform the majority of functions that we plan to offer. The two main functions of the UI, in accordance with the other major components, are pulling data from the Spotify



(a) With Spotify's app



(b) With SongSwipe

Figure 6: Flowcharts demonstrating the processes of managing a Spotify playlist.

API and saving progress to a local JSON file. The data that is pulled from the Spotify API includes information about playlists and songs. That data is used to prompt the playlist management functions of the UI. The UI also saves the progress of the staging area to a JSON file that is local to the user's system.

The local JSON holds information about the staging area and acts as a cache for that information. It is saved after every action made. This ensures that the user can pick up where they left off. The JSON file is also used in the process of writing changes to the Spotify playlist through the API because it contains the records fo the changes to be made. Once the user finishes managing a playlist, the information for that playlist will be removed from the JSON.

4.2 User Interface

The user interface consists of three main components: a playlist selection screen, a main swiping screen with cards for each song, and a staging area where any changes can be committed to the Spotify library. They can be seen in Fig 8. The playlist selection screen consists of individual elements showing all user playlists and an area to enter URLs of public playlists. Clicking on a playlist will take you to the card selection screen for that playlist.

Once a playlist has been selected, the main swiping screen displays the first song as a large card with the song's information. Users can swipe left or right on each song to decide whether to keep or remove the song from the playlist. Any songs that have been left-swiped will be sent to the staging area to be removed.

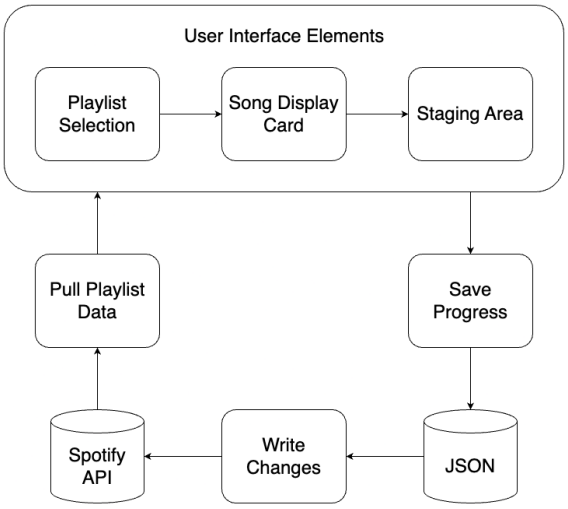


Figure 7: Architecture diagram

The staging area shows every song that will be removed from the current playlist. Users can choose to remove songs from this list if they change their mind. Once changes have been submitted, meaning that the user can no longer undo changes, it is passed off to a back-end endpoint, where the Spotify API will commit the changes to their library.

4.3 Staging Area Cache

When managing a playlist, there is a cache of all the decisions made so far to keep progress. In the case where the application is exited unexpectedly, the progress can be restored. This cache acts as the internal representation of the staging area as well and is read from when changes to the playlist are written. It is stored as JSON in the user's local storage. Using JSON gives the benefits of easy parsing and setup. An example of the proposed JSON structure can be seen in Fig. 9. The JSON is partitioned into sections that are determined by the Spotify ID of the playlist. A Spotify ID is a base 62 identifier of an artist, album, track, or playlist [4]. The Spotify ID is guaranteed to be unique for each playlist. Therefore, it is the best way to partition the data because a playlist's name may change or could have the same name as another playlist. It can also help search for that playlist again to pick up where you left off.

Each playlist has the following attributes defined within it: the playlist name, the progress (number of songs gone through), the total number of songs, the songs that you have chosen to save, and the songs that you have picked to be removed. The playlist name, progress, and total songs are saved simply for display purposes. For the choice selection attributes, it is a list of objects that hold some information about the songs: the Spotify ID for the song and the name of the song. The Spotify ID is needed for the same reason as the ID for the playlist. The song name is for display purposes. Album cover data is not be cached due to requiring an image download, which will take up unnecessary space on the user's device.

A benefit of using this structure is that it can easily scale. If in the future we decide to update the application to hold more account information, this may lead to the use of a database. This structure we defined could easily be added as an attribute in a user record. This would especially work for a non-relational database that follows the document model.

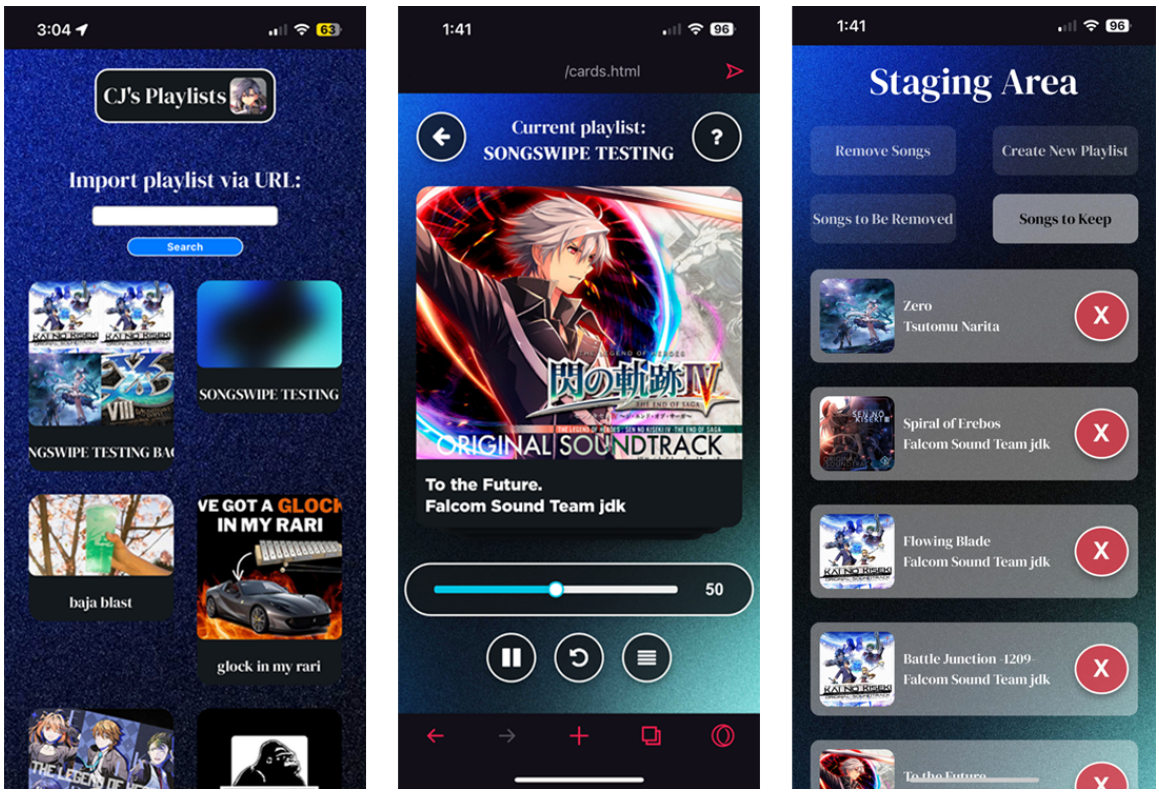


Figure 8: Preview of the UI for the playlist selection and card swiping. Left is playlists page, middle is the card swiping screen and the right is the staging area.

5 Technology Stack

5.1 User Interface Technologies

The user interface is built with the following technologies: HTML, CSS, and JavaScript with JQuery [17]. This simple, yet effective stack allowed us to make a responsive and effective UI. Using these web development technologies allows SongSwipe to be easily accessible from the web on all platforms. HTML was used to lay out each of the pages for the UI components. CSS was used to stylize the elements that appeared on each of the pages. And finally, JavaScript was used to add interactivity to each of the pages. The interactivity includes fetching and writing data from the API service and allowing the user to perform actions. jQuery enhanced adding interactivity by allowing for better event handling and smoother animations.

5.2 Spotify API Interactions

The main technologies used for the backend are Typescript, ExpressJS [18], and the Spotify API [4]. Typescript was chosen due to its static typing. This minimized type errors with the various data that was worked with. For interacting with the Spotify API, we originally intended to use Spotify's TypeScript SDK. However, it was not compatible with the front-end stack that we decided. The library was built to work with technologies like React, where the application can be rendered on the server. Our front-end does not support this, so we needed to develop a Representative State Transfer (REST) API that acts as a wrapper for the Spotify API. This allows the front-end client to interact with the Spotify API by providing the authorization access token to the REST API endpoints that were developed. It was developed using ExpressJS, a web application framework that is useful for building REST APIs. The Spotify API, as mentioned, was used to interact with data on Spotify. One feature that was deprecated from Spotify's API was the ability to get the link to a song's preview. A library [13] provided by GitHub user Rexdotsh, provides a workaround for this by extracting the link

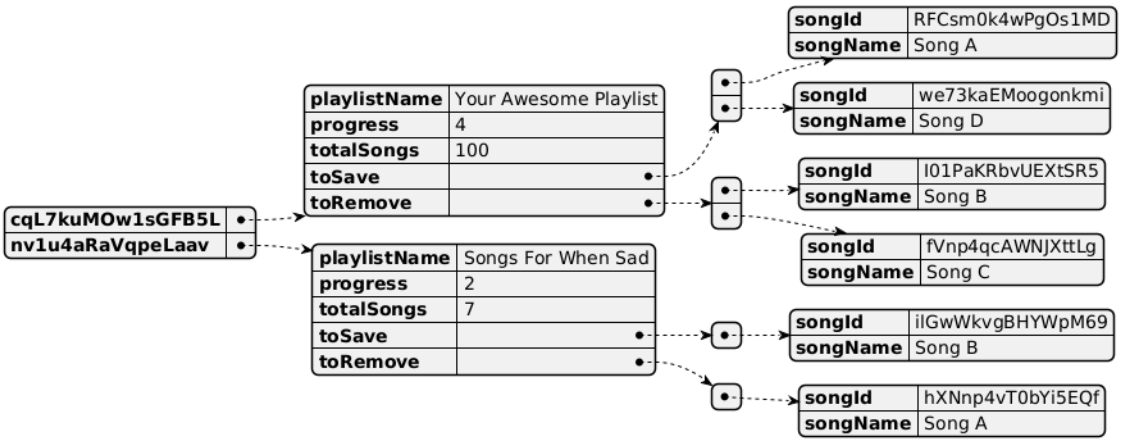


Figure 9: Visualization of our JSON formatting.

from Spotify's embed player HTML.

5.3 Other technologies

Other technologies that we used involve things related to hosting and unit testing. For a hosting service, we chose Netlify [10]. This is a web hosting platform with a generous free tier and easy integration with GitHub, where our repository is hosted. It handles continuous delivery, which removes the need for us to configure a deployment pipeline. It also provides 100 GB of bandwidth and 300 build minutes. Netlify's function service [11] was also used. This service allowed the back-end REST API to be hosted in a serverless manner. Netlify functions were also included in the free tier, which helps being able to host the site for free. If the application sees success, we may need to pivot to a better hosting solution. We used the Jest library for our JavaScript unit testing; it has support for both JavaScript and TypeScript, along with useful features like mocks. Mocks are allowed for testing of API endpoints with mock data. Our unit tests were also integrated with our GitHub repository. Whenever a pull request was opened, it could not be merged if any of the unit tests failed.

6 Project Challenges

The main challenges that we faced were working around the Spotify API, which could prove to be tedious at times. We made use of an external workaround used to get Spotify preview URLs as they no longer support that function by default. We were initially worried Spotify would deprecate a core feature, however, this did not happen. There was also a time during development when the API was down, which held up development for a day.

7 Unit Testing

We used Jest for our unit tests. They were broken up into four different units: API endpoints, Spotify interactions, Spotify preview requests, and utility functions. The API endpoint testing checked to see if each endpoint on our REST API responded with the correct status codes and body. We used Jest's mock functionality to mock data returned from Spotify's API. This allowed us to insert mock data with a similar structure to what was expected from Spotify's API to ensure that correct data was returned when successful. For testing related to Spotify interactions, tests covered the functions that dealt with formatting requests and checking responses for any errors. This ensured that there were no errors from us in sending the requests. For testing the Spotify preview library, the test makes sure that the correct fetch is called to ensure that if the library is updated, it still works as it needs to. We also created tests for our utility functions, such as creating random keys when calling the Spotify API.

8 Usability Testing

To validate our project and its ability to solve the proposed problem, we conducted two types of tests, each targeting one of the application's primary use cases: managing personal Spotify playlists and copying playlists from other users. Our goal for this testing is to directly compare the same task performed with both the Spotify UI, as well as SongSwipes. In our individual unmeasured testing, we could feel the improvement, however, we wanted to quantify it.

One feature we added to our application to test usage efficiency is metric collection. Based on a toggle in the .env file for the project, metrics collection can be enabled. When enabled, whenever a user goes to edit a playlist, a csv and text file will be created to append data to. The csv file will contain the song name, artist, album, the time to make a decision in seconds, and the decision made for all of the songs in the user's playlist that they went through. The text file will contain the total amount of time it took for the user to go through the playlist. Having this data ensures that we have accurate metrics on how our application performed, which were used to judge effectiveness. For collecting metrics for Spotify, we made it easier by writing a Python script that waits for input to indicate a decision and keeps track of the time between decisions. When using this script, the decision was logged once the user went to the next song. This data was then written to a csv file.

8.1 Estimation

As reviewed previously in our presentation, we decided to give an estimation of our product and how it could save you time.

Regarding Spotify, once a song is inside a playlist, you can no longer listen to the preview unless you find the source by searching the song on the platform. Therefore, this makes Spotify playlist revision significantly tedious because you would have to listen to the song, or determine the most popular part of the song, usually the chorus, and then formulate whether you like the song or not, then remove the song through Spotify's tedious song removal process.

We deduced that the process of reviewing songs in a playlist, with regard to Spotify, would take about 30 seconds for each song and another 3 seconds to remove user-selected songs. In regard to SongSwipe, we deduced that it would take 15 seconds to review each song and another second to swipe. At the time of this estimation, we only calculated Spotify Premium users.

Given that x is the number of songs in a playlist and y is the number of songs you want to remove. The estimation equation will look like this:

$$30 \text{ seconds} \times x + 3 \text{ seconds} \times y = \text{Time Consumption Estimation for Spotify} \quad (1)$$

$$15 \text{ seconds} \times x + 1 \text{ seconds} \times y = \text{Time Consumption Estimation for SongSwipe} \quad (2)$$

8.2 Procedure

For our testing, we decided to test multiple methods to encapsulate all the different kinds of users that would be using our application.

For testers, we selected people who first had Spotify. It would take more time to teach someone how to navigate Spotify UI testing, compared to someone who was more familiar with it. They also needed some patience to follow the procedure correctly when asked. Finally, they had to be someone who was willing to spend a good chunk of time testing both the Spotify UI and our SongSwipe UI free of charge because we are too broke to pay for people who were specifically trained to test these types of applications.

Method A

For this method, we wanted to target the most common user who would be using our application, which would be:

Someone who plays their playlist on shuffle and skips songs they no longer like, yet still have the song in the playlist, we have implemented the procedure as follows:

Controlled Environment:

- **Playlist:** Standard playlist consisting of 75 songs
- **Target Artist:** Pitbull (9 songs in the playlist)
- **Listening Time:** 5 seconds per song

Spotify Testing:

1. Play the playlist on shuffle mode
2. Locate the chorus and listen to the song for 5 seconds
3. After the 5-second listening period:
    - If the artist is not Pitbull: Skip the song
    - If the artist is Pitbull: Remove the song from the playlist

SongSwipe Testing:

1. Listen to each song for the mandatory 5-second period
2. After the 5-second listening period:
    - Testers are only permitted to swipe left when the artist is Pitbull
    - All other artists should not be swiped left

Both Spotify UI and our SongSwipe UI have text to show the song and artist, which makes the selection accessible both audibly and visually.

Method B

For this method, we wanted to target the other use case of our application, going through a friend's playlist and selecting songs that you like.

Both Spotify UI and our SongSwipe UI have text to show the song and artist, which makes the selection accessible both audibly and visually.

Before beginning these tests, the participants had the use cases of our application explained along with their task. Music for the playlists was also tailored to be music that the participant would not know.

Controlled Environment:

- **Playlist:** Two different playlists with 50 songs that were all different.
- **Task Assigned:** Listen to music for however long you choose and make a decision if you like it or not.

Spotify Testing:

1. Play the playlist on shuffle mode.
2. Listen to the song for any amount of time that is up to the user.
3. After the decision is made:
    - If they like it: Add playlist like shown in Fig. 1
    - If they did not like it: Skip song

SongSwipe Testing:

1. Listen to the preview of a song for however long.
2. After listening:
    - Swipe right if they liked the song
    - Swipe left if they disliked the song

9 Measured Results

User Testing Metrics

Method A

Tester Name	Spotify Subscription	≈ Spotify Time	≈ SongSwipe Time	Spotify Time	SongSwipe Time
TesterA	Free User	00:37:57	00:18:54	$\leq$ 03:02:00	00:06:25
TesterB	Spotify Premium	00:37:57	00:18:54	00:42:05	00:13:38

Method B

Tester Name	Spotify Time	SongSwipe Time	Spotify Average	SongSwipe Average
TesterA	13.03 minutes	8.1 minutes	15.64 seconds	9.72 seconds
TesterB	5.92 minutes	4.08 minutes	5.76 seconds	7.1 seconds

Figures

Fig 10

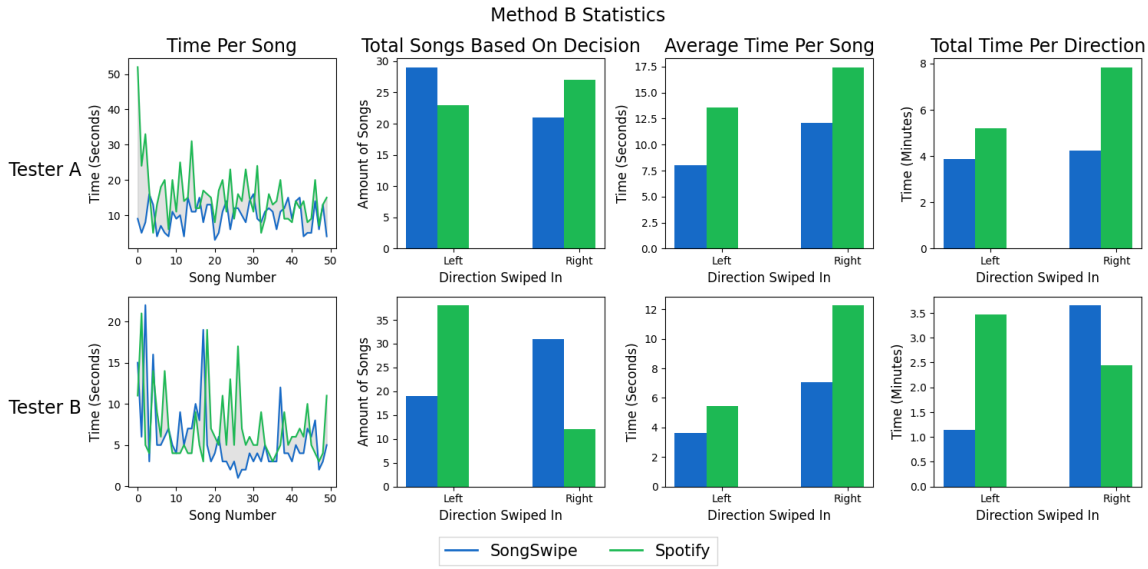


Figure 10: Statistics for Method B. Includes time per song, total songs based on decision, and total and average times. Generated using Matplotlib [20].

9.1 Limitations

When it comes to testing, there will always be some form of limitations. In our project, to scope the most data we can get, we had all the members conduct their own testing with their own procedure, and by doing this,s we would have different results based on the guidelines in their respective procedures.

Spotify limits the number of skips a free user can have, about 6 skips every hour. Tester A, who is a free user, attempted to conduct the test; it was too tedious and time-consuming so we decided to perform an approximation on how long it would take. After realizing free Spotify users exist, this demonstrates that our application will greatly reduce time consumption in the decision-making process as our users will only have to listen to the song for at most 15 seconds per song.

We calculated the Spotify approximation based on the amount of hours it will take to listen to the whole playlist from start to finish. We also deducted that a skip will approximately save a person 4

minutes as a song can range from 3-5 minutes, so 4 is the middle ground.

$$\text{Total Playlist Time} - \lfloor \text{Listening Hours} \rfloor \times 6 \text{ skips/hour} \times 4 \text{ minutes} = \text{Spotify Consumption Approximation} \tag{3}$$

9.2 Discussion

Method A

Based on our experiment, we had 75 songs in our playlist, which had a total listen time of 4 Hours and 38 Minute,s and we removed 9 songs from that playlist.

Using our Estimation equations above (1 and 2), we estimated that it would take 37 Minutes and 57 Seconds for Spotify and 19 Minutes and 54 Seconds for SongSwipe. This has a time reduction of 47.5%. It is observed that the more songs are in a playlist, the time spent on playlist management is significantly reduced.

When looking at our actual results from user testing, we see that TesterA has about a 95% time reduction, meanwhile TesterB has a 67.8% time reduction. This shows that our application goes beyond our expectations by showing a time reduction of more than 50 % and even more when it comes to free Spotify users, who have harsh limitations on song skips.

When comparing the two testers in Method A, it is apparent that one was faster than the other, despite the subscription setback. This is because Tester B was enjoying listening to the song previews while testing instead of waiting only the minimum of 5 seconds, which causes their results to be more obscured. Despite this, they still spent 67.8% less time on SongSwipe than Spotify.

Method B

Based on the experiments for method B, it can be seen that SongSwipe has a clear advantage when it comes to the time spent.

Adding together the total times for both testers for Method B makes the percentage difference in total time 43%. Doing the same for average time brings the percentage difference to 65%. This shows that for going through new music quickly, SongSwipe can aid in that process and lower the time spent. This is supported by the table for Method B and Fig 10.

Some other trends were noticed in testing with the users. For both testers, it was noticed that there was visable frustration when messing up an input on Spotify's client and sometimes getting annoyed when they could not find the playlist to add the song to. While using SongSwipe, this was not observed.

Another trend that happened was that the testers did not really scrub through the song at all while using Spotify. This might have been due to them doing SongSwipe first which shows that there is more of a quick nature to it. More testing could have been done where Spotify was used first to see if this made any impact at all.

10 Lessons Learned

Through this project, we learned a lot of valuable lessons in architecting and developing a solution to a problem and how to work as a team. Throughout the semester, we revised and improved the original idea for the project based on the various feedback we received from our professor and peers. This made us rethink different aspects of our application and helped sharpen our ideas.

We also improved a lot at sharing our ideas. Writing multiple papers throughout the semester has helped. It forced us to think about how we can share our ideas in a way that makes sense. It made us

think more about whether we needed certain things if we were overthinking something. It also had us present aspects of our application through diagrams, which helped with development.

For technical lessons learned, we learned a lot about designing a web application without the more modern tools that are typically used today. This was a nice change of pace and made us think about challenges that are typically solved with the libraries. An example of this would be routing; instead of using a nice routing library, we had to do routing ourselves and use pattern matching to make sure that the application would behave the same if it was being run locally or on a hosting service. We also learned a lot about OAuth and having to manage the lifetimes of authorization tokens due to not being able to utilize the Spotify Typescript SDK. This also allowed us to work more with REST APIs and design endpoints to interact with Spotify.

Appendix A: Contributions

- **Anthony Simao:**
  - Front-end development, HTML, and CSS for page styling and basic animations.
  - Back-end development, JavaScript, and jQuery for basic event handling and animations.
- **Nena Heng:**
  - Front-end development, focusing on CSS styling to ensure a user-friendly GUI.
- **Christopher Coco:**
  - Back-end development in Typescript involves Spotify API calls and credential management, unit testing, and deployment
- **Katiana Sourn:**
  - Back-end development regarding local storage, expected behavior configuration, and back-end UI handling
  - Assisting with Spotify API calls
- **Raj Ray:**
  - QA Tester

Appendix B: Feedback

We received several important pieces of feedback from our instructor to improve the goals and specifications of the project. The section rearrangement was addressed and implemented for fluidity. Furthermore, we made adjustments to our table to show the more recent data collection we performed. We also referenced our figures in our text and revised figure formatting issues. We were advised to compare our product to an external tool such as Skiley.net, but there would be a lack of comparison as the playlist management options are limited to sorting the songs and removing duplicates. This will only have a minuscule effect compared to our product SongSwipe.

After receiving peer reviews, we revised our project proposal. Our peers stressed the risks of using Spotify API because of deprecations. We as a team, are aware of possible issues with this and have taken it into account with what we want for our software. Previews were also a main concern by our peers, considering the nature of how they work. They play an essential part in our project in helping the user determine if they would like to remove a song or not. In terms of the proposal itself, each team member has thought about how we word certain specifications and concepts to best fit our audience. We've also added diagram captions to better explain the figures in our proposal. We have taken both peer and professor advice and detailed the milestones for every week and separated them as backend and frontend milestones. As it was not clear before, we've also explained what each piece of technology clearly does and will provide for our project specifically.s

References

[1] Christopher Coco, Nena Heng, Raj Ray, Anthony Simao, Katiana Sourn, *SongSwipe* Available: https://github.com/ListenToAJ/SongSwipe

[2] Shubham Singh. "Spotify User Statistics 2025." Available: https://www.demandsage.com/Spotify-stats/

[3] Spotify. *Spotify App*. Available: https://www.spotify.com/

[4] Spotify. *Web API Documentation*, Available: https://developer.spotify.com/documentation/web-api

[5] Spotify. *Web API TS SDK*, Available: https://github.com/spotify/spotify-web-api-ts-sdk

[6] Playmixify. Available: https://playmixify.com/

[7] João Vitor Verona Biazibetti. Skiley. Available: https://skiley.net/

[8] JMPerez. Spotify Dedup. Available: https://Spotify-dedup.com/

[9] Stuart Dredge. "Spotify Removes Features From Web API Citing Security Issues," November 2024. Available: https://musically.com/2024/11/28/Spotify-removes-features-from-web-api-citing-security-issues/

[10] Netlify, Available: https://www.netlify.com

[11] Netlify Functions, Available: https://www.netlify.com/platform/core/functions/

[12] Netlify Pricing. *Serivce Pricing*, Available: https://www.netlify.com/pricing/

[13] Rexdotsh. "Spotify Preview URL," 2025. Available: https://github.com/rexdotsh/Spotify-preview-url-workaround?tab=readme-ov-file

[14] Swipefy B.V. Swipefy. Available: https://swipefy.app/

[15] Draw.io, Available: https://draw.io/

[16] PlantUML JSON Diagrams, Available: https://plantuml.com/json

[17] Jquery, Available https://jquery.com/

[18] ExpressJS, Available (https://expressjs.com/)

[19] Jest, Available (https://jestjs.io)

[20] Matplotlib, Available https://matplotlib.org/