SongSurf

<https://github.com/comp195/SongSurf>

Shahbaj Sohal - [s\_sohal2@u.pacific.edu](mailto:s_sohal2@u.pacific.edu)

[William Balbuena](mailto:w_balbuena@u.pacific.edu)- [w\_balbuena@u.pacific.edu](mailto:w_balbuena@u.pacific.edu)

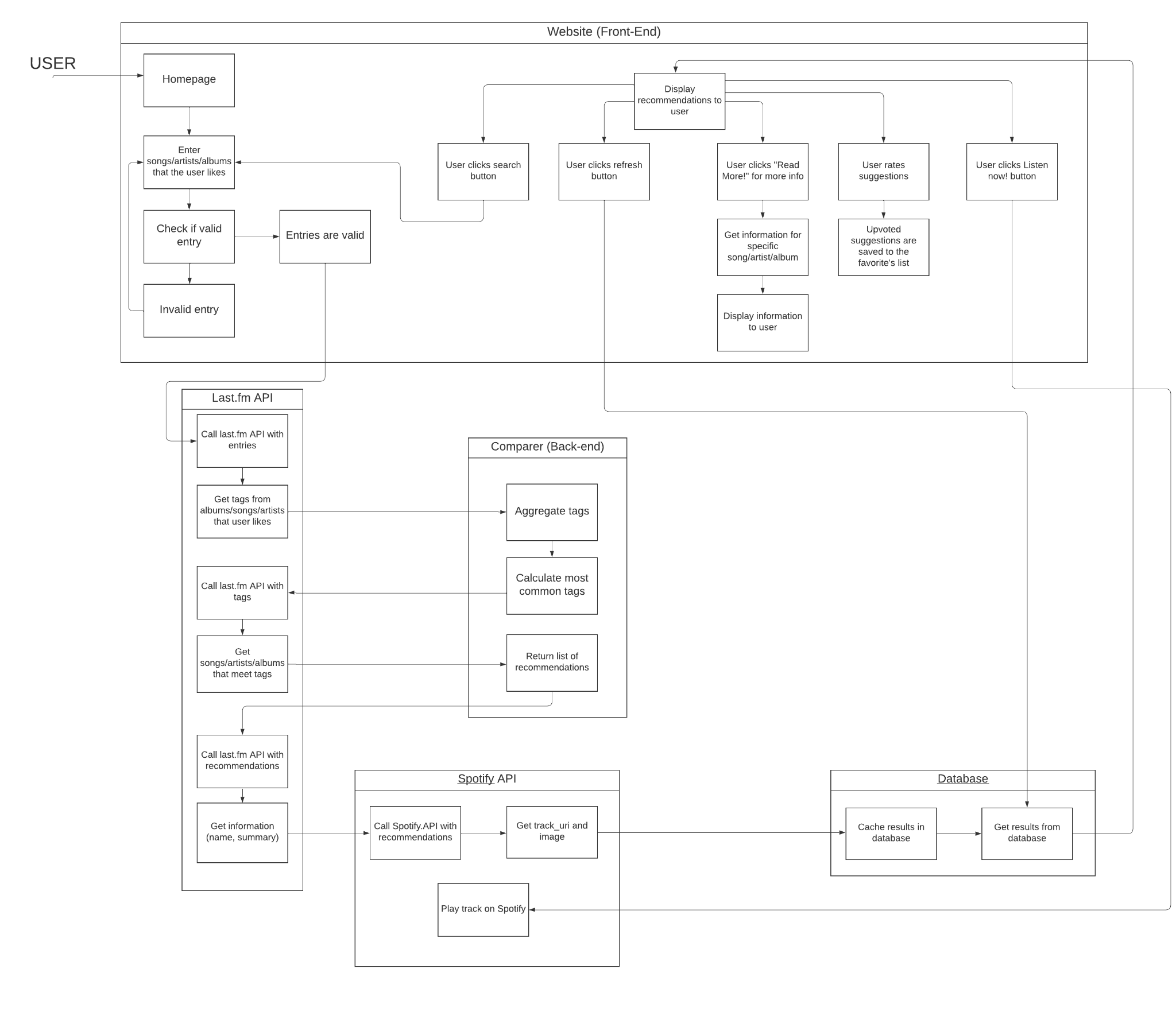
[Patrick Nisperos](mailto:p_nisperos@u.pacific.edu) - [p\_nisperos@u.pacific.edu](mailto:p_nisperos@u.pacific.edu)

May 5, 2023

Introduction

SongSurf is a website/tool that music enthusiasts or someone looking for new music to listen to can utilize to find new artists, songs, or albums to listen to. As a group we have decided to create a website with a sleek and user-friendly user interface. Using API calls to a music database we will create recommendations based on what the user inputs into the website. The user will be able to input artists, albums, or songs that they enjoy into a text field and also select whether they want their recommendation to be an artist, album, or song and in return get a few recommendations based on their inputs. The user will also be able to select whether they like or dislike the given recommendation and continue to get more recommendations. Our goal as a team is to get an algorithm working to give the best possible recommendations and have an easy-to-use and a fantastic website design. We will be working together over the course of this semester to accomplish this goal using what we have learned from our previous classes.

# System Architecture



# 

# Hardware, Software and System Requirements

**Hardware Requirements**

* **Display -** Any inch display, must have minimum resolution of 852x480 (480p)
* **Disk Storage -** N/A
* **Communication Network -** N/A
* **Input Devices -** 
  + Mouse & Keyboard
* **Minimum network bandwidth -** Minimum 5MB/s

**Software Requirements**

* **Languages** - Python, Javascript, CSS, HTML
* **Framework** - Flask
* **Libraries** - Python and Javascript standard libraries
* **Tools** - SQLite, SQLAlchemy
* **APIs** - Last.fm, Spotify

**System Requirements**

* **Operating System** - Any OS since our project is a website

# External Interfaces

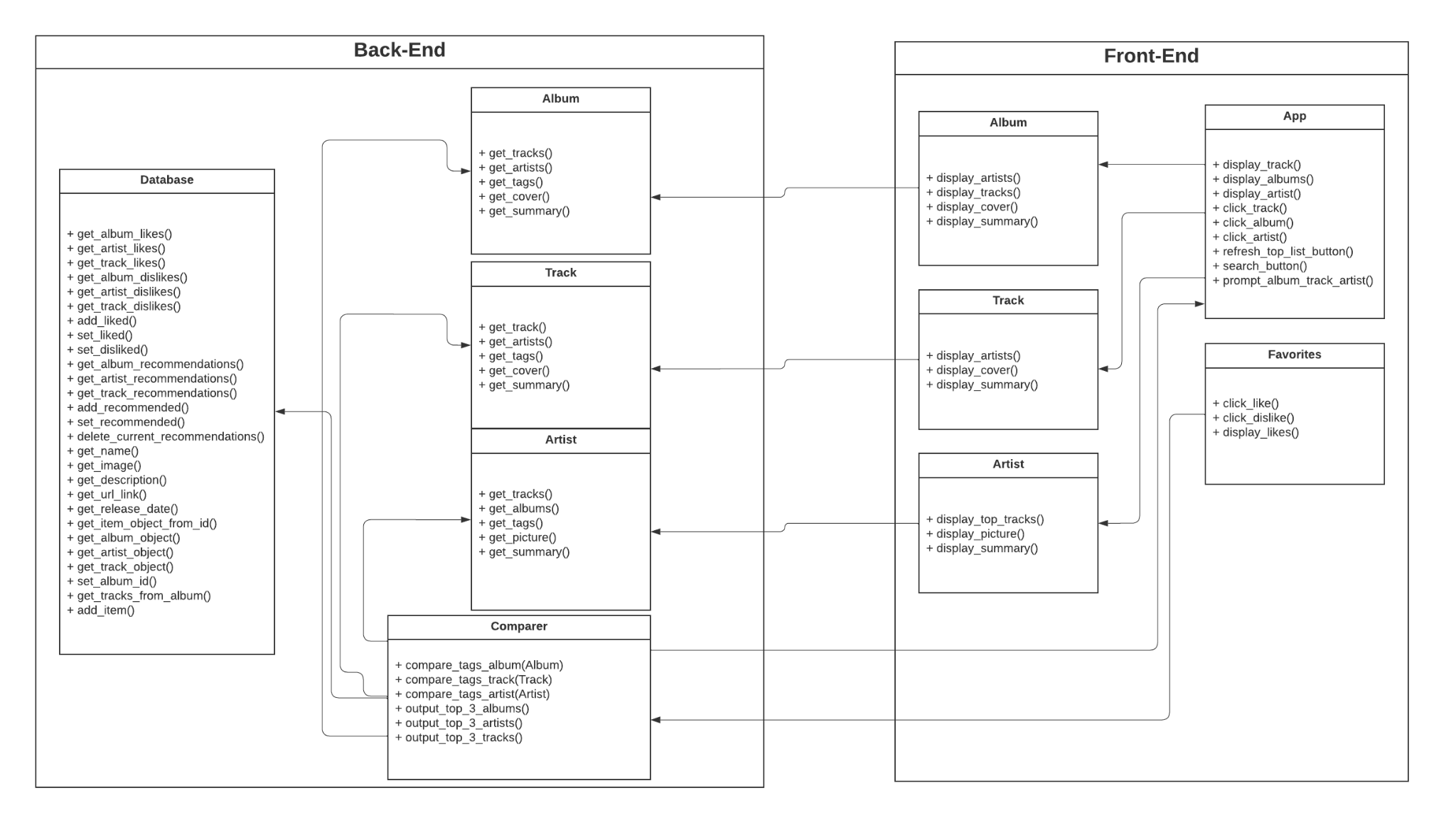
The Last.fm API is well documented and can be accessed through this link: <https://www.last.fm/api#getting-started>. To utilize the Last.fm API, an API account is required. When making calls to the API an identifiable User-Agent header and a set of parameters that include an API key are required for each request.

The Spotify API is also well documented and can be accessed through this link:

<https://developer.spotify.com/documentation/web-api>. To utilize Spotify’s API, a developer must have a Spotify account and be logged into your account. To play music with a playable link one must have their premium account signed in and open. Just like the Last.fm API when making calls to the API an identifiable User-Agent header and a set of parameters that include an API key are required for each request.

# Software Design

## Class Diagram and Specifications



#### Front End

* **Display**

The *Display* class is the most important class for the front-end, responsible for displaying all front-end interactable components to the end user. In addition to the foundational album/track/artist information, this class will also incorporate “refresh” functionality that will save the likes/dislikes of previous set of album/track/artist and account for them in the new set of album/track/artist it will produce. This class will communicate with the *album/artist/track* classes.

* **Favorites**

The *favorites* class is responsible for likes/dislikes functionality and interface. It will also be responsible for displaying all the user’s likes. This class will communicate with the *comparer* class.

* **Album**

The *Album* class has the ability to display the artists, all the tracks in the album, the album summary, and the album cover picture. This class will communicate with the back-end *Album* class to fetch the raw data. It will also communicate with the *Display* class.

* **Track**

The *Track* class has the ability to display the artists, the album cover picture, and the track summary. This class will communicate with the back-end *Track* class to fetch the raw data. It will also communicate with the *Display* class.

* **Artist**

The *Artist* class has the ability to display the top tracks, the picture of the artist, and the artist summary/bio. This class will communicate with the back-end *Artist* class to fetch the raw data. It will also communicate with the *Display* class.

Back End

* **Comparer**

The *comparer* class is the most important class for the back-end responsible for the algorithms to most accurately discover the top albums/tracks/artists for the end user. This class will communicate with all the back-end classes to fetch tags, to *search\_box* to get items searched for, and to *favorites* class to fetch likes/dislikes.

* **Album**

The *Album* class is responsible for fetching all data regarding tracks, artists, tags, and summary associated with the album. This fetching will utilize the last.fm API. This class will communicate with the *comparer*, and the front-end *Album* class.

* **Track**

The *Track* class is responsible for fetching all data regarding album, artists, tags, and summary associated with the track. This fetching will utilize the last.fm API. This class will communicate with the *comparer*, and the front-end *Track* class.

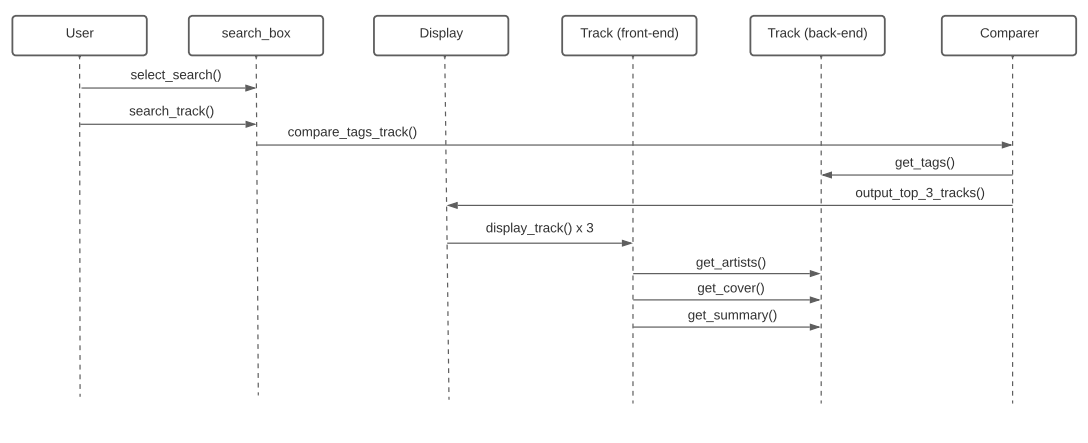
* **Artist**

The *Artist* class is responsible for fetching all data regarding tracks, albums, tags, and summary/bio associated with the artist. This fetching will utilize the last.fm API. This class will communicate with the *comparer*, and the front-end *Artist* class.

* **Database**

The *Database* class is responsible for catching all data regarding tracks, artists, albums, recommendations, user login, storing favorites. This class abstracts all of the database queries into functions that can easily be called anywhere in the code base.

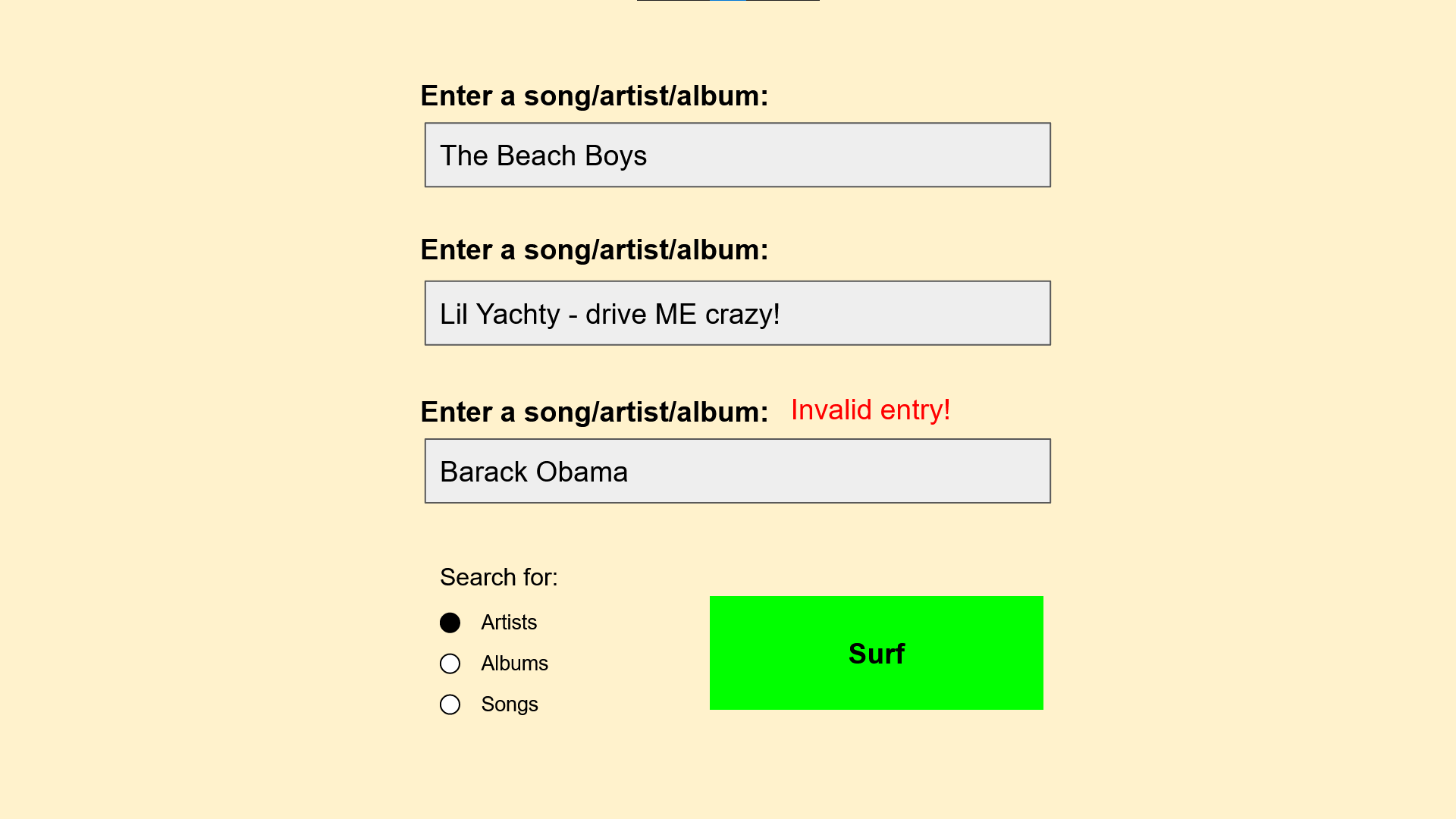
## Interactive Diagrams



## Design Considerations

* Simple, straightforward functionality that leaves room for more features
* Making sure the user is inputting correct information (such as real artists or songs)
* Making an efficient website that is not super laggy
* Maintain a clean codebase that everyone can work with
* Did not want to create god class and split everything up into different classes
* Being specific with function definitions
* Wanted user to be able to like and dislike recommendations
* Also be able to see the recommendations that they liked

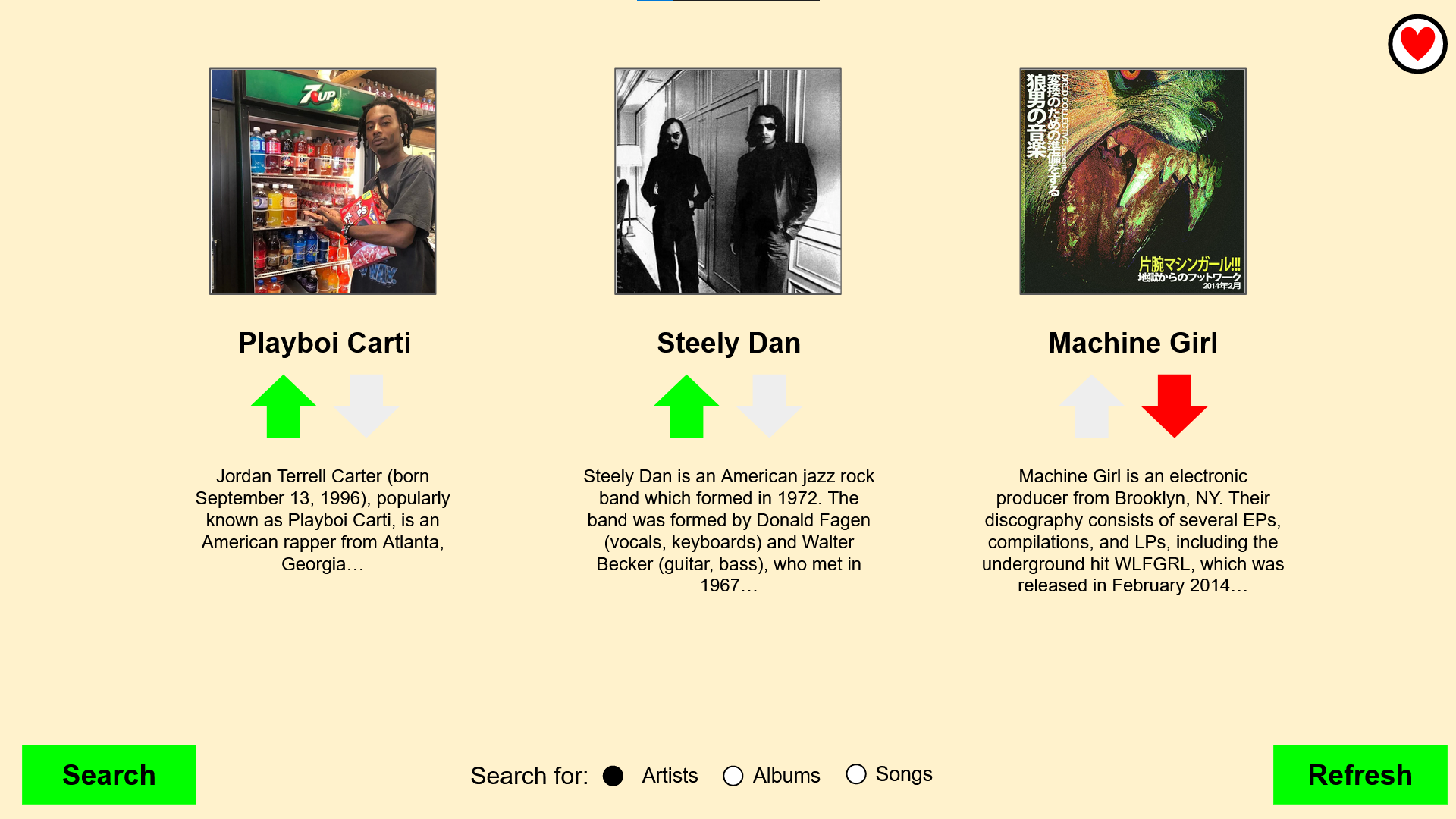
# User Interface Design



*Mock up of entry page*

When the user first opens the website, they will be prompted to enter up to three songs, albums, or artists that they enjoy. We will check whether or not their entries are valid and will notify them if such. They will also be asked to choose between either artists, songs, or albums to search for.

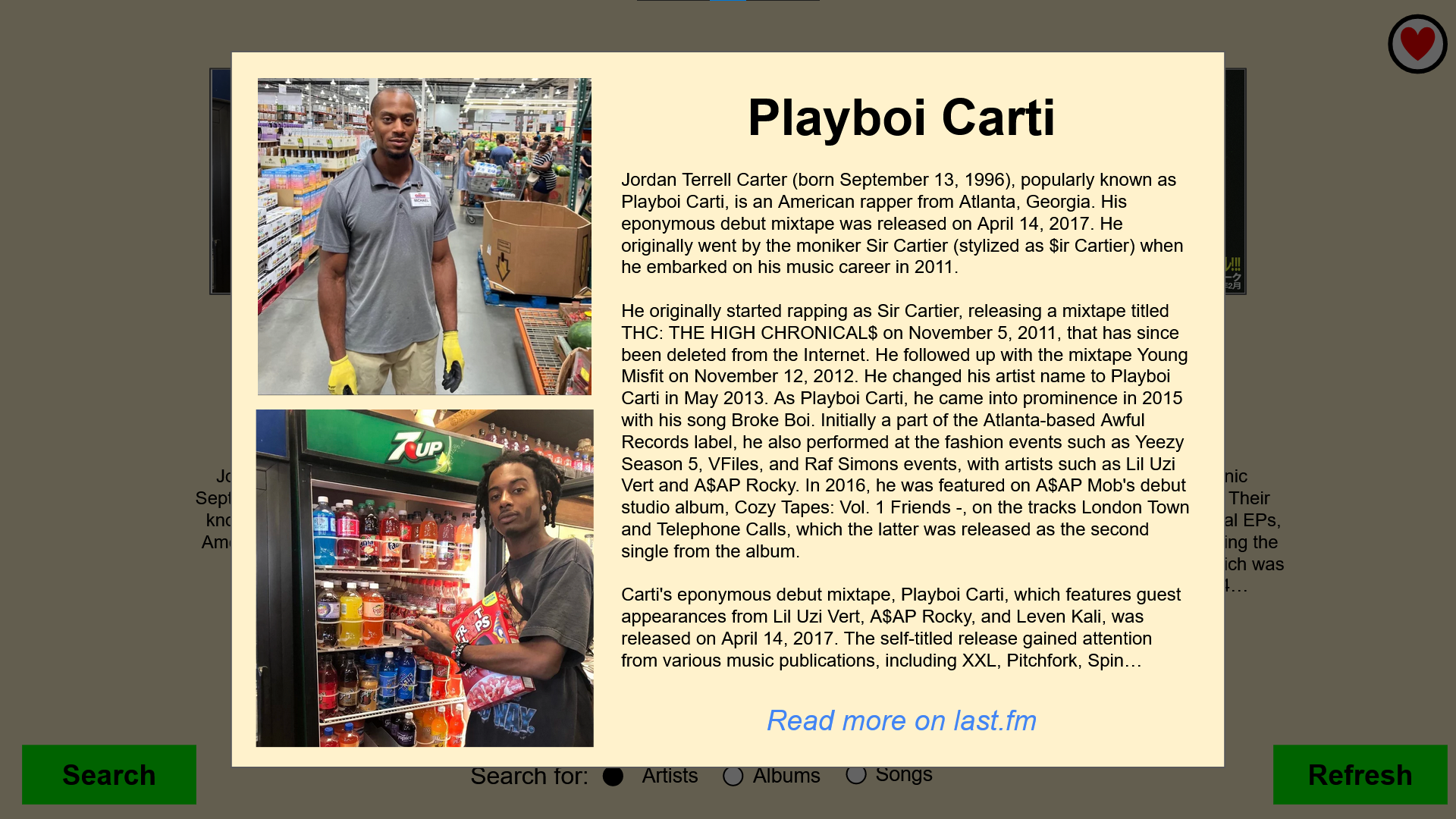
Once they have entered valid artists and have selected a category to search for, they will proceed to the next screen by clicking the “Surf” button.



*Mock up of recommendation page for artists*

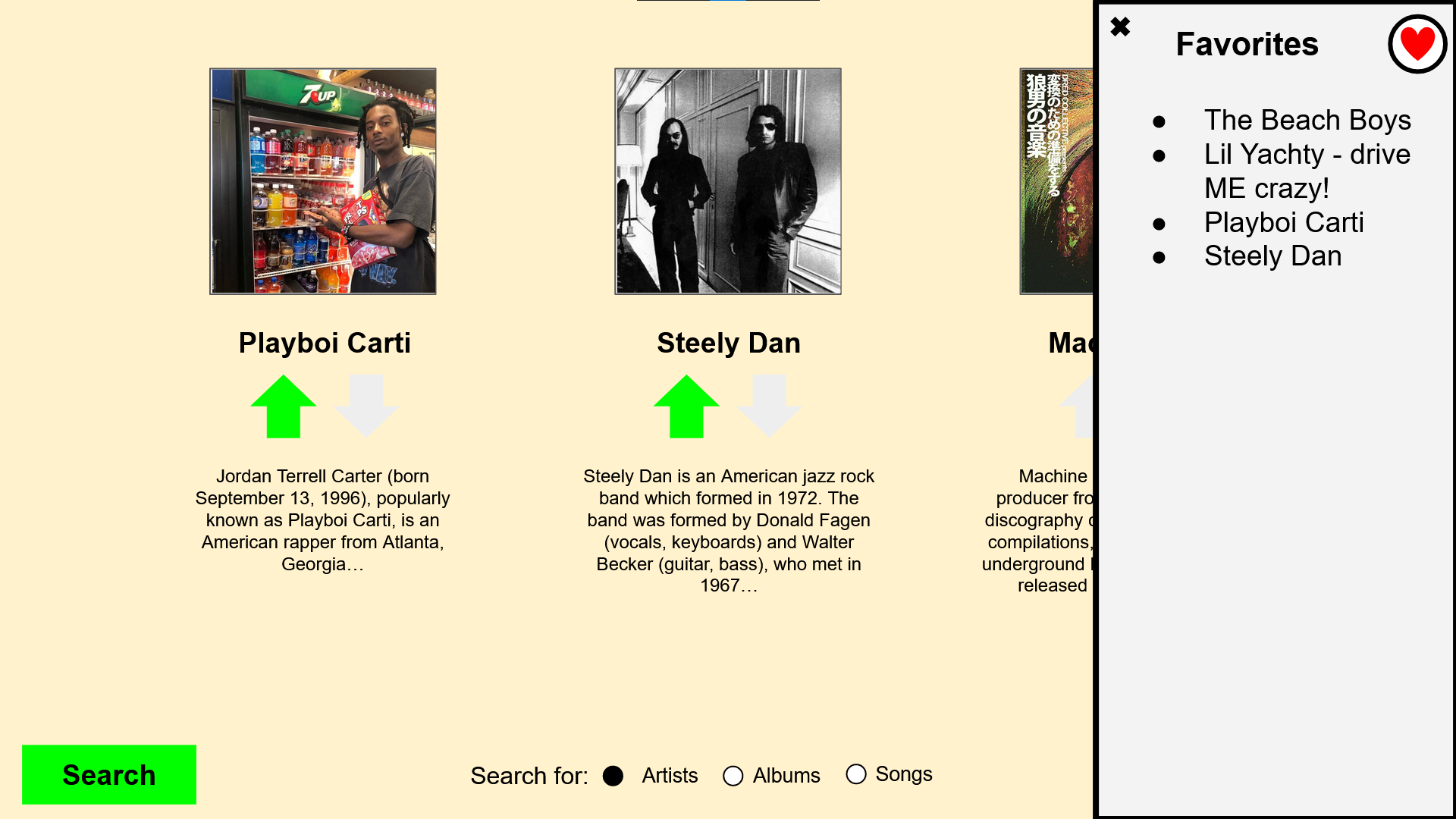
After entering valid entries the user is then presented with three recommendations of songs/artists/albums depending on their choice. They are then able to vote on the recommendations presented to them and can refresh their selections. The next selections are impacted by their initial three entries as well as their reactions to prior recommendations. The user is also able to choose what category their new selections will provide.

The user is also able to restart their search by clicking the “Search” button which will bring them back to the initial screen where they’ll be able to start their search over from scratch.



*Mock up of summary page*

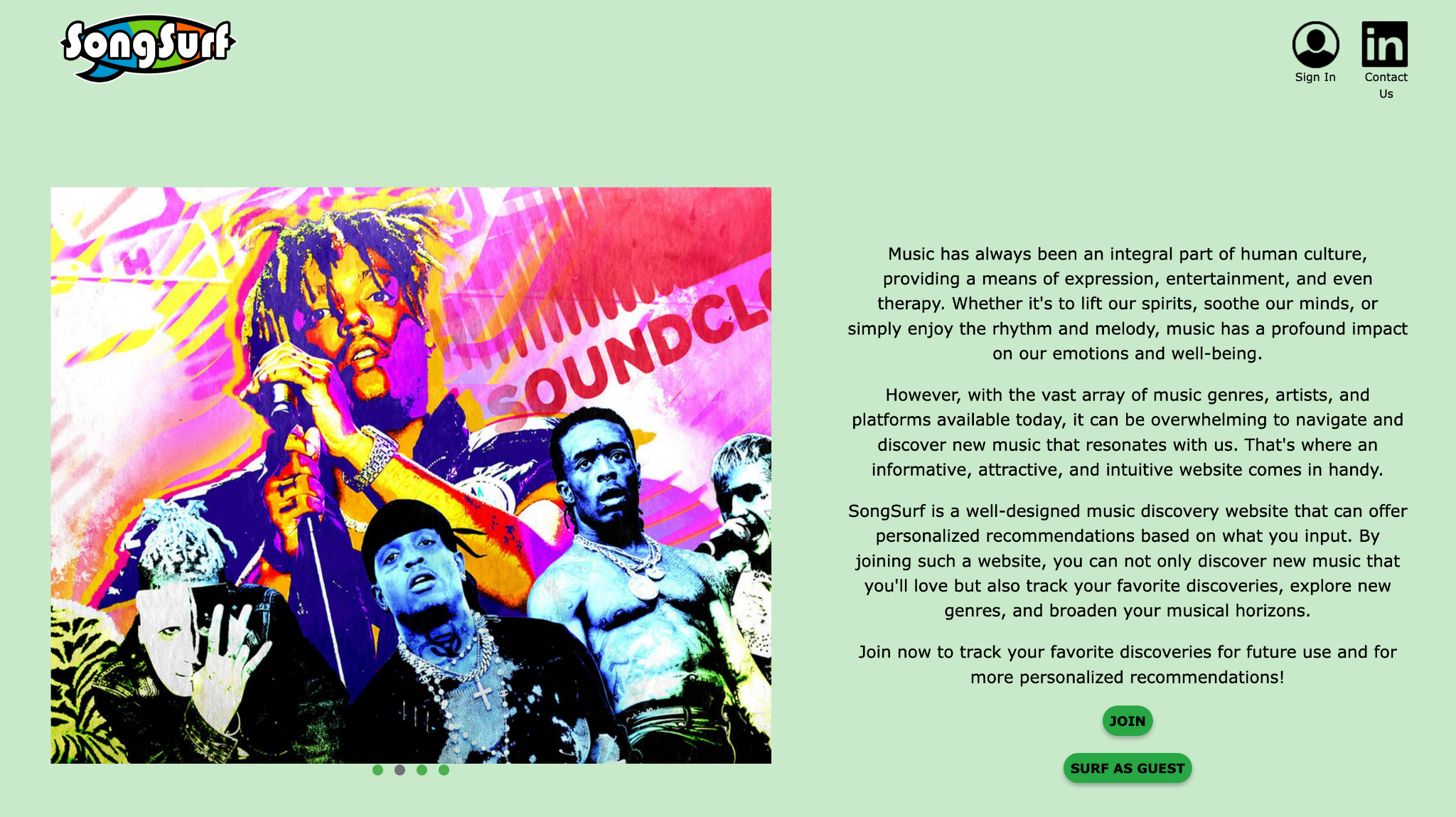
On the selection screen the user is able to click the images of the songs/artists/albums to get more information. The summary also contains a link to its respective last.fm page where the user is able to find out more about the song/artist/album.



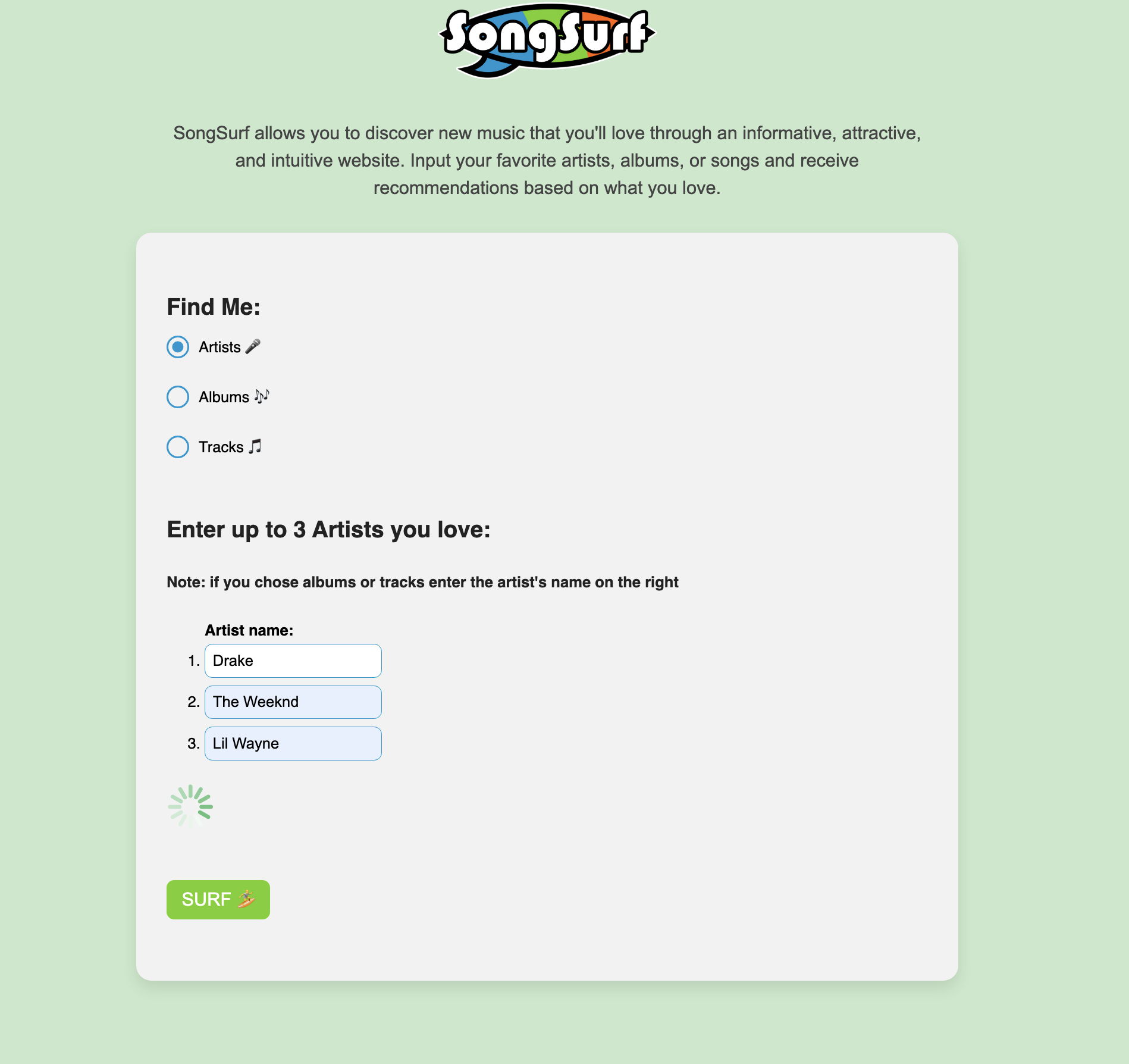
*Mock up of recommendations page with favorites sidebar*

The user is also able to click the favorites icon on the top right to keep track of what they’ve liked so far.

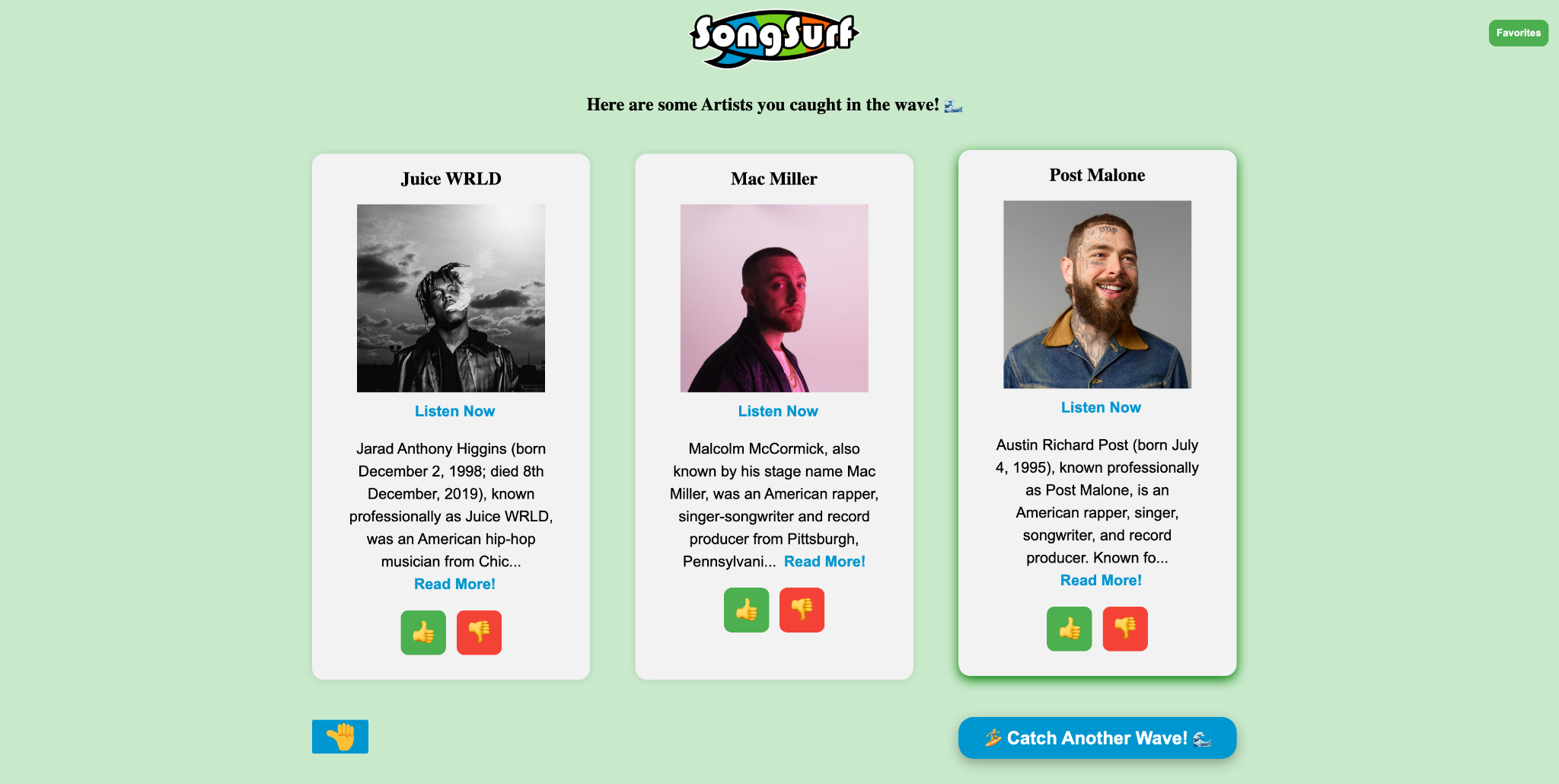
Our actual design ended up being quite similar to our mockups of the website with us having a search page where users select what they want to input and then surf leading them to a recommendation page. The recommendation page shows three recommendations that have short descriptions and have links to more information on the Last.fm website. They also have a link to play the song on Spotify and get the image of the artist, track, or album from Spotify. We also incorporated a home page that allows users to get a feel for what the website is about and it has a place for users to create an account for the website. We locked some of the features such as liking/disliking and viewing favorites behind having an account. We also made the entire website surfing themed to match the name of the website and the logo we designed for it which made the UI have a unified theme and make it stand out more.



*SongSurf Home Page*

**

*SongSurf Search Page*



*SongSurf Recommendation Page*

# Glossary of Terms

**algorithm** - a set of instructions, used to solve problems or perform tasks, based on the understanding of available alternatives

**application programming interface (API)** - a way for two or more computer programs to communicate with each other

**tag** - a label attached to someone or something for the purpose of identification or to give other information

**user interface (UI)** - the means by which the user and a computer system interact, in particular the use of input devices and software

**database -** a structured set of data held in a computer, especially one that is accessible in various ways.

# References

Grupman, Celeste. “Tutorial: Getting Music Data with the LAST.FM API Using Python.”

*Dataquest*, 7 Oct. 2019, www.dataquest.io/blog/last-fm-api-python/.

“Last.fm Music Discovery API.” *Last.fm*, Last.fm, www.last.fm/api#getting-started.

“Web API | Spotify for Developers.” *Spotify*, Spotify, https://developer.spotify.com/documentation/web-api

“Welcome to Flask.” *Welcome to Flask - Flask Documentation (2.2.x)*, Flask,

flask.palletsprojects.com/en/2.2.x/.