Design Document

Music Affect Data Collection App

Tanya Bihari, Mia Bilka, Michael Murad, and Chris Somnitz

CS 461-400 Fall 2022

Oregon State University

Abstract

There exists a growing interest in the collection and analysis of human affectual responses to music. Current databases of music affect responses are generally limited or not publicly accessible. As a solution to this problem, our team will develop a music affect crowdsourcing application which allows users to self-report their responses to songs streamed via Spotify. This document specifies the overall architecture of that application as well as the functionality of its four core components.

Contents

Revisions

#	Version	Date	Notes
1	1.0	3 November 2022	Initial revision.
2	2.0	21 November 2022	Edited revision for submission.

Overview

Scope

This document will detail the specifications and functionality of the application to be developed. It will cover the organizational layout and flow of our application, the backend and frontend tools and specifications used to develop the app, and other relevant materials and details covering the app's use and development. The document will not cover any post-deployment plans, including potential research use, maintenance plans, or development beyond the Minimum Viable Product (MVP).

Purpose

This document will function as a reference for the app's development through to the MVP. It will also detail the app's functionality from the frontend and backend perspectives. This document may undergo revisions during development to reflect changing design decisions, but we expect the majority of development to proceed generally based on the application architecture detailed here.

Intended Audience

The Intended Audience for this document is the development team, project sponsor, and Senior Software Engineering Project instructional team. This audience may also include individuals planning to use the app in their research who are interested in its underlying functionality.

Definitions

The following terms are defined for the project:

- **Affect Data** - A person's emotional response to a song, represented with valence (positive-negative) and arousal (excited-calm).
- **SDK** - Software development kit.
- **User Metadata** - Information about a user, such as age, geographical location, or user ID.
- **Song Metadata** - Information about a song, such as the associated artist, album, release date, or length.
- **AWS** - Amazon Web Services, a cloud platform that can host databases and code.

Introduction

An increasing interest in understanding human affective responses to music has led to attempts to collect relevant data through several observational studies and AI-based predictive tests [1]. Expanding the available data could refine existing algorithms involving human emotional response to music and would facilitate the development of theoretical but data-lacking technologies. It could also refine existing technologies used to predict and/or build around certain emotional models and experiences, such as mood-based playlists, automatic song suggestion tools, or music recommendation algorithms.

Existing Solutions

To our knowledge, there does not currently exist a crowdsourced, research-oriented application intended to collect emotional affect data. Popular music streaming services such as Spotify have shown interest in collecting, analyzing, and predicting affectual responses to music [2], but this data is largely private, internal, and used for marketing or curation purposes as opposed to academic research. In addition, copyright laws and other logistical barriers make extensive data collection difficult for independent academic researchers, resulting in only a small number of datasets that are sparse and difficult to generalize from.

Existing datasets include the results of the AMG1608, DEAM, PmEmo, Deezer, and DEAP studies. Aside from the Deezer dataset (which used a neural network instead of human response) [3], each dataset has less than 2,000 songs and less than 700 participants, thus providing the need for more extensive data collection.

Proposed Solution

We seek to fill the existing gap in music affect data by creating an application which allows users to self-report their emotional responses to music, expanding the range of testable music by integrating with the APIs of existing streaming services and storing collected test data in the cloud. An application-based method of acquiring music affect data from self-reporting participants is a new approach to prevalent issues around music affect data collection. Users will be able to listen to individual songs via an existing music streaming platform, report emotional affect data via a simple interface, and submit their response to a cloud-based database. Our project, once released and thoroughly tested, could result in a substantial dataset representing both concentrated studies of selections of music as well as that of the participants' general listening habits. A working application has the potential to solve multiple problems around music affect data collection while introducing new possibilities to this area of research.

Proposed Solution

System Architecture

The proposed system is a multi-platform web and mobile application. The system will have a front-end that includes the user input and Spotify streaming, and a back-end with a server and database to record the responses. These system components are each explained in more detail in the User Input, Spotify Integration, Backend Server, and Database sections below.

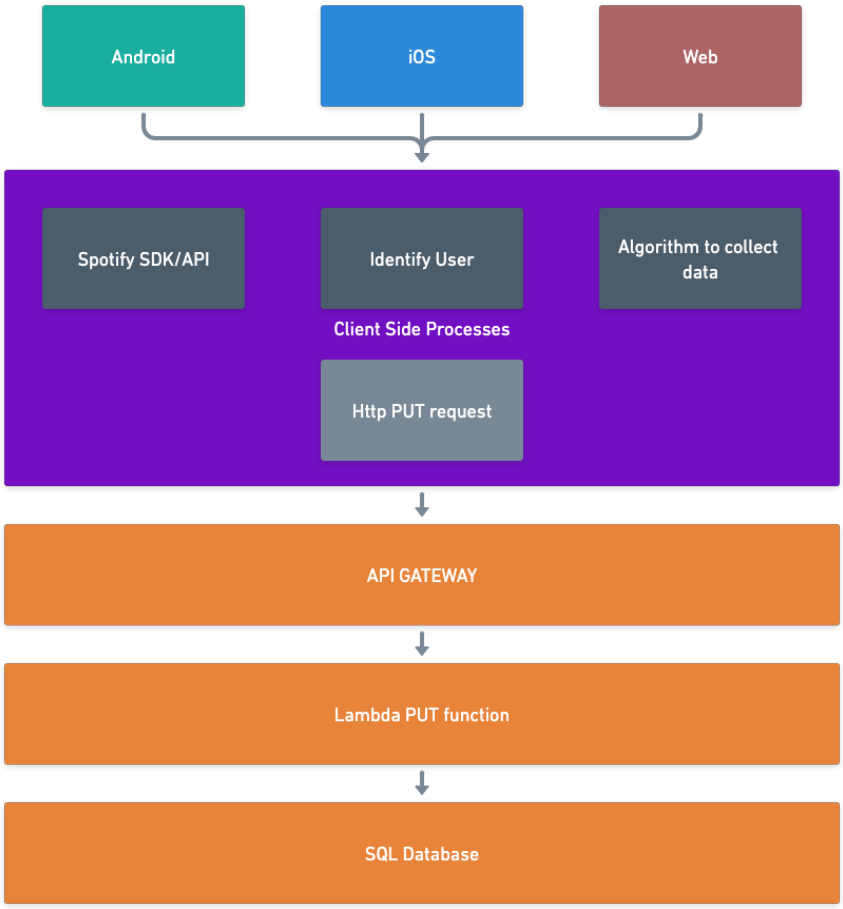

Figure 1

A diagram of the system architecture.

User Input

On the user end, the application will generally be very simple, consisting of a few pages and processes. A mockup of the page designs can be found in the appendix.

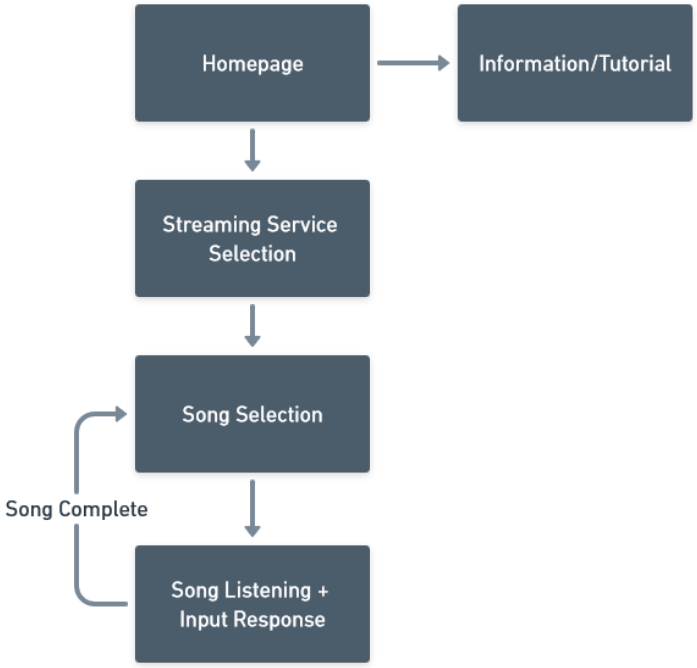

Figure 2

User experience flowchart.

As shown in Figure 2, the application will consist of a homepage that allows a user to traverse to a simple tutorial that introduces the user to the basics on how to enter data. Then a user can select a song, hit play, and enter responses. This process will then be looped.

To collect user music affect data, we will need an interface that allows users to enter their emotional response. The goals are collecting a rich dataset, while still having a simple, intuitive interface. There is a balancing act that needs to be done, where an interface that is too simple will yield lacking results, whereas an informationally dense UI can yield a more valuable dataset, yet may not be as usable. The goal is to find the perfect balance.

Based on previous field studies, such as Deezer [3] or DEAM [4], the arousal/valence model seems to be the gold standard emotional model.

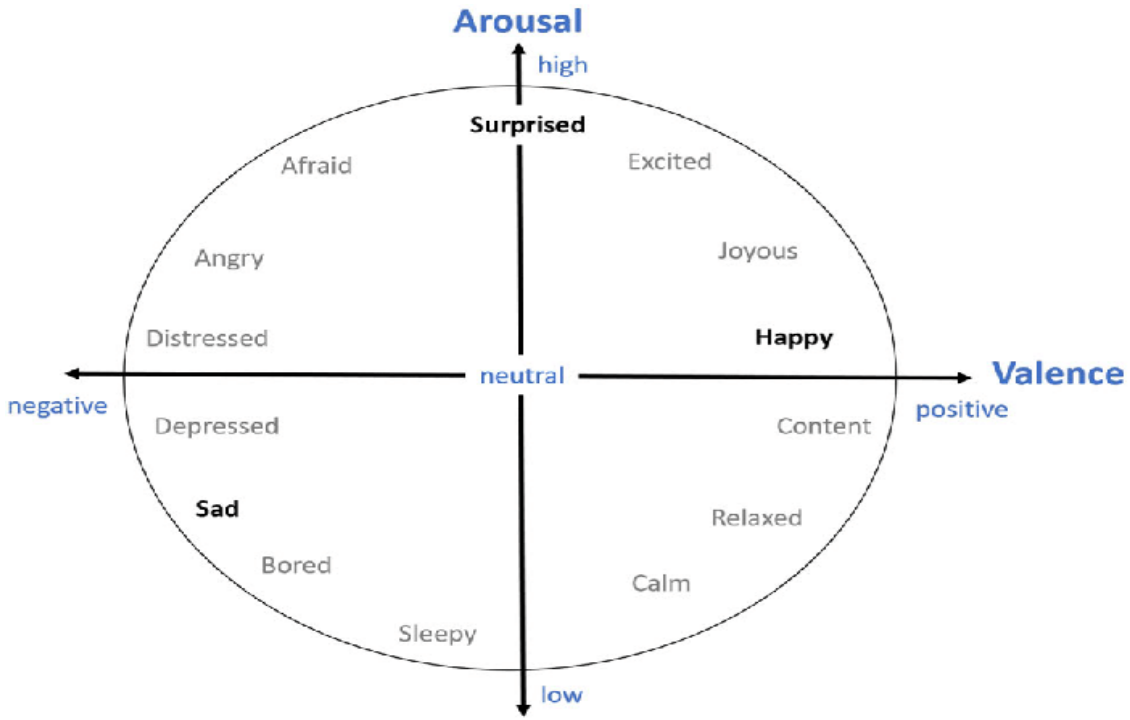

Figure 3

James Russell's circumplex model of emotions.

The arousal / valence model in Figure 3, seems to suit itself well to be translated to a UI that a user can enter musical affect data with. Questions arise though, if discrete values should be used versus values in a range, and if so, how intuitive is it for users? Perhaps the intermediate emotions can be overlaid on the UI. Intermediate values would yield a rich dataset, yet might be confusing to users if not executed properly.

Using a 2x2 grid, similar to the model shown in Figure 3, could easily map valence to the x axis and arousal to the y axis. In Flutter, you can determine the relative location of a tap. It will provide a tuple of Cartesian coordinates to be mapped to the arousal / valence model directly.

The collection of data would commence when the user initiates the playing of a song. Perhaps over a specified interval, the last cartesian coordinates entered could be saved in an array of objects containing the interval and cartesian coordinates.

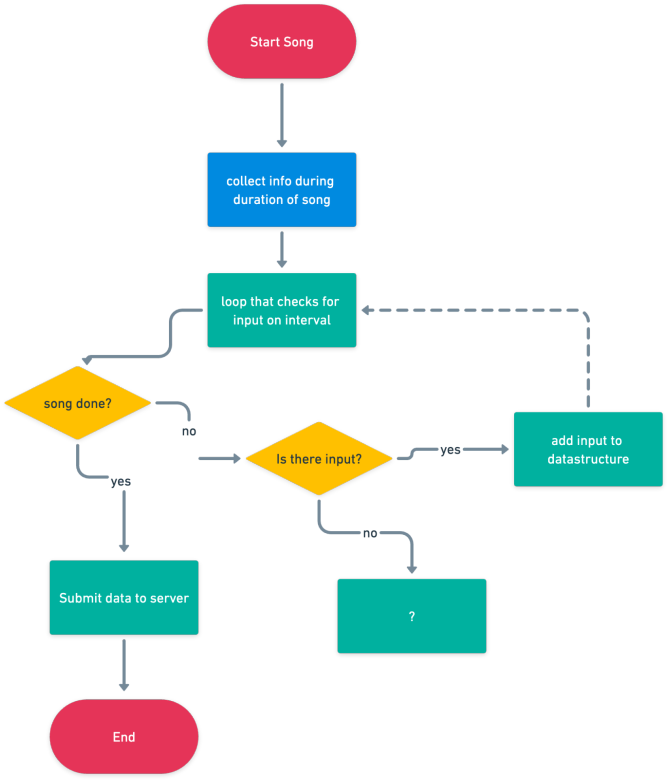

Figure 4
Possible process for collecting music affect data and sending it to the server.

- **Music affect data collection algorithm -** The algorithm begins when a user starts a song, sets the conditions to collect user inputted data, and then upon completion, sends to to the server
    - **Loop collecting data** - User input is collected on an interval
        - Every 5 seconds save last entered user input
    - **Store data in data structure** - Data to be stored in an array of objects, which contain the interval and cartesian coordinates that map to arousal / valence model.
    - **Send data to server** - Upon completion of the song, send data to the server

Figure 4 shows a process for collecting data, yet there still remain some questions. What happens if a user never enters any data? Also what happens if a user enters some data in quick succession before the interval elapses? Do we store a visual indicator for previously entered data? There will need to be multiple proof of concepts (POC) created, and edge cases will need to be ironed out.

Proposed Solution

To ensure that a rich dataset is created, an interface consisting of a clickable 2x2 grid (whether square or circle) will represent an arousal / valence emotional model, and allow users to input music affect data.

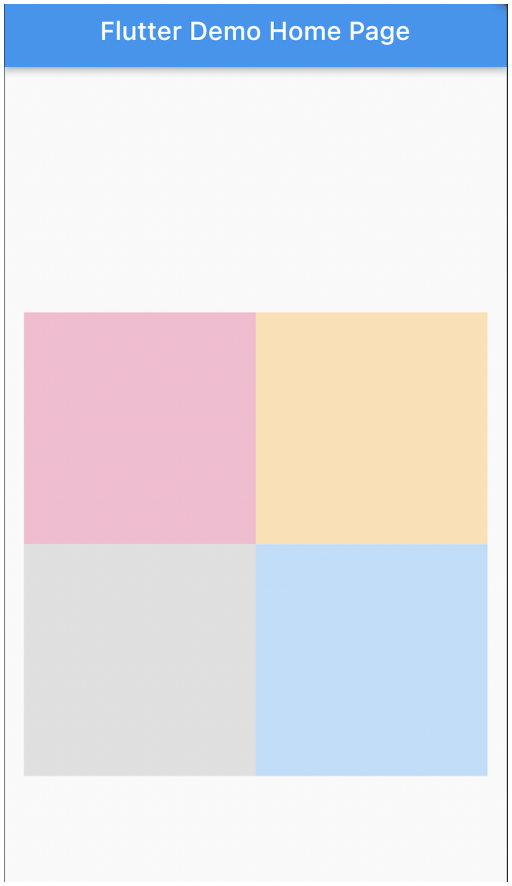

Figure 5
A rough mock up of a tappable 2x2 grid

Upon starting a song, the user will be able to tap a portion of the screen that will map touch location to Cardesian coordinates. These coordinates will then be stored in an array of objects along with the interval. The x axis will represent arousal, the y axis will represent valence.

Although there are still outstanding questions and perhaps a great amount of variations and testing to be done, this paper will serve as a starting point in the journey towards finding the perfect balance between detailed user input and a simple interface.

Next steps will be actually creating POCs and getting feedback from our mentor to make sure that the final product is in line with the initial goals of the project. The POCs we will need to verify would be the UI interface, and the different implementations of it, how it would work, what the UI would look like, and the fine details surrounding it. Also the POC of implementing the Spotify SDK and integrating it into the system. There will be many iterations and deviations needed to come up with a satisfactory final product.

The edge cases will need to be sorted out. Perhaps the user will need to have an initial emotion and that and serve as the default should they not enter any data. We will need to experiment with a detailed overlay (that can be toggled) on the UI that shows a few intermediate emotions. Hopefully a UI element can be created, that is simple and intuitive, that can show previous or current selections.

All in all, this will serve as the starting point, and has hopefully addressed possible edge cases that will need to be addressed. There was a lot learned in this process. And with

everything said, I feel very confident that we will be able to create this system and come up with a decent UI.

Spotify Integration

Currently Spotify is the choice for streaming music integration into the application. The goal will be to have a way for users to choose songs, hit play, and then be able to input their music affect data. Spotify does have Android and iOS SDK's as well as web API. A possible challenge might be integrating a native SDKs into Flutter. Upon initial investigation, there is a Flutter package that wraps the native wrapper called **spotify_sdk**, hopefully this will assist in the process.

Beyond the technical difficulties, there are requirements for us to even use the spotify SDK.

For Android, we will need to register our application on the Spotify Developer Dashboard and obtain a client ID. To complete the process we will need to whitelist a redirect URI that the Spotify Account Service will use to callback to our app after authorization. Moreover, we will need to register App Fingerprints which are used for authentication between our Android Application and the Spotify Service. For our iOS application, we will need to do similar steps.

Hopefully the parts that involve native code can be handled in an easy way or by the **spotify_sdk** package.

Spotify also has a list of requirements to use their SDK that we need to be aware of. Most of these requirements involve us not deriving any data about the users in a specific way. We need to be mindful of these requirements.

Backend Server

This section contains the design specifications for the server-side data collection and storage processes of the app. The backend of the application is concerned with sending user affect data from a user's device to our servers, where the data can be parsed and then stored in a persistent database.

From a server-side perspective, we are interested in a few relevant variables:

1. Song metadata (artist, album/release, track title, etc.)
2. User affect data (x/y coordinates paired with timestamps)
3. User metadata (some anonymized form of user ID associated with the affect data)

The application will contain the user affect data, song metadata, and user metadata within a JSON object that can be sent in the body of a REST request to our API. It is possible that issues may occur during the user data collection process on the application side (e.g. internet connectivity problems, bugs, user error) that interrupt or corrupt response data before a full response is completed. To minimize inaccuracies in the overall dataset, we will not send any user affect data from the application to the server before it is guaranteed, application-side, that the user affect data is complete and accurate, contained within the appropriately formatted JSON object, and ready to be sent to the server.

Once this data is ready to be sent, it will then be transferred from the user's local instance of the application to persistent storage in the cloud database. The application will make a PUT request to our server with a JSON object containing all relevant data. Our server will verify that data and send a response back to the client application as to whether data storage was successful.

We will use Amazon API Gateway to handle the initial PUT request, then send the data to be stored to an AWS Lambda function which will interface with the database as needed. The entire server-side data collection, verification, and storage process will happen in a number of steps, as shown in Figure 6.

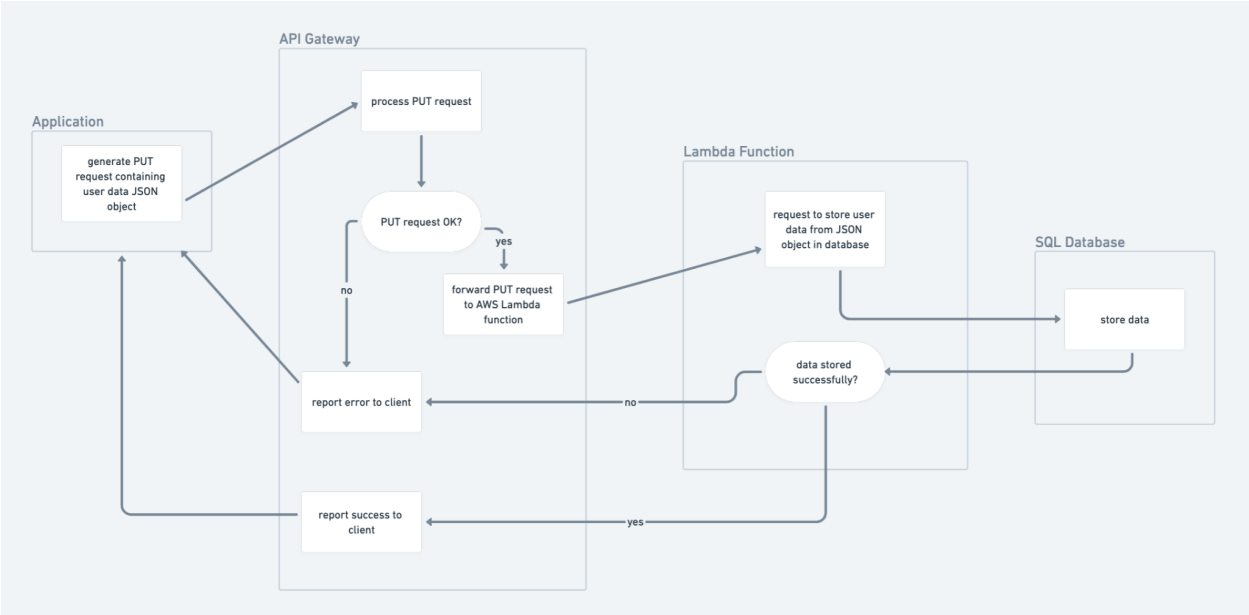

Figure 6

A diagram showing the process of user affect data collection from application to database storage.

A successful query would follow this set of steps:

1. The application sends a PUT request to the server, the body of which is a JSON object containing all of the relevant user response data.
2. Using Amazon API Gateway, our API handles the PUT request. If the PUT request is acceptable, our API forwards the data to an AWS Lambda function.
3. The AWS Lambda function interfaces with our database to store the data.
4. If the data is stored successfully, our Lambda function returns this information to our API, which then sends an OK response back to the application.

Database

Description

This project will use a relational database. Getting the users' responses and storing them in the database is the end goal of the app. Once the user has listened and responded to the song, the app will send the stored response to the server where the database is hosted. The response data will be stored as either an array or JSON type which will be an array of arrays that will hold the time interval, valence and arousal coordinates, along with metadata about the response. The database will also include a table of songs, which will be updated whenever response data for a new song is received, and a table of Users, that will allow us to differentiate users and store some metadata about them. Response data will be able to be retrieved by searching for song titles, artists, song lengths, and/or date ranges.

A user ID will be stored so that if the same user responds to the same song twice, the old response will be overwritten. Duplicate responses should be avoided, but anonymity is a concern for this app, so the user ID will be generated using a hash of the user's Spotify username.

The response data will be stored inside the actual database, as a JSON type or as an Array type in the Response table (mentioned below). We will need to do some experimentation to see which is preferable. This would allow us to leverage the relationship between users, songs, and data, while still using a flexible data structure.

Models

The file containing the data of a single response will be a 3-row matrix of comma separated values, with the top row containing the time intervals and the bottom rows containing the valence and arousal positions at each time interval.

The database will include three tables: A Response, User, and a Song table. The Response table will include the data, and the metadata of the response: The user ID, song ID, and date recorded. The Song table will include the song ID, artist, song title, and song length. The User table will consist of the ID, and some metadata of the user, however, based on laws surrounding privacy, such as GDPR, we may not be able to save these. Further research needs to be done.

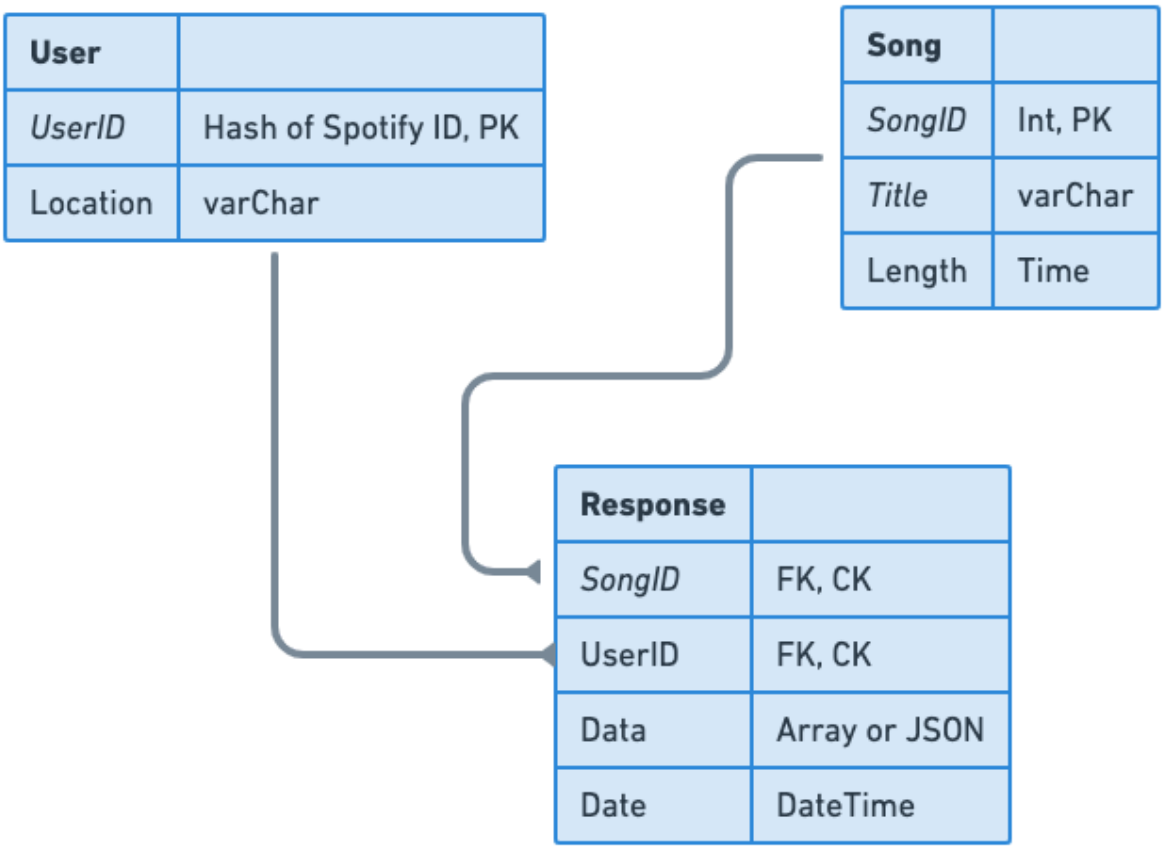

Figure 7

ER Diagram.

Interface/API

A variety of functions will be provided to allow the server to access the database. These functions will include:

An INSERT statement to add a new song and return whether it was successful.

An INSERT statement to add a new response and return whether it was successful.

A SELECT statement to check if a response with a certain song ID and user ID already exists and returns true or false.

An UPDATE statement to replace an old response with a new one and return whether it was successful.

And a SELECT statement to retrieve response data about one or more songs, able to be filtered by song ID, date range, artist name, song title, or song length. Returns paths to all relevant response data files, grouped by song.

Persistence

Postgres, or any SQL database hosted on AWS, by default will persist the data. AWS does have backups available as well should we want to create backups in case of a catastrophic failure.

Technologies

We'll be using PostgreSQL as our database system. It's free, widely-used, and is highly compliant with the SQL standard, so it's the easiest option. This database will be very simple, so it won't require anything specialized. MariaDB is also being considered as an option for the same reasons.

System Dependencies

System Software

Our application will make use of a number of software tools.

1. **Development**
    a. Flutter - framework for app development.
    b. GitHub - version control and source code hosting.
    c. Visual Studio Code - IDE for development.
2. **Audio Streaming**
    a. Spotify SDK/API - source of audio streaming data and song metadata.
3. **Server and Database**
    a. AWS - server and tools.
        i. AWS Lambda
        ii. Amazon API Gateway
    b. Hosted SQL database
4. **Deployment**
    a. Google Play Store
    b. Apple App Store
    c. Web server that hosts Flutter web app, possibly hosted by AWS

System Hardware

Users will require an Android or Apple smartphone or desktop device to use the application, but otherwise no special hardware is required. AWS will handle everything in the cloud for us, and they will deal with all the hardware that the server and database run on.

Release / Deployment

We plan to release a mobile application via the Google Play Store and Apple App Store for smartphone users. We also plan to host a web version of the application that can be accessed from desktop browsers. Our plan for application deployment, including timelines for submitting applications to these platforms as well as a chosen host for our web application, is still under development.

Test Plan

Much of this is to be determined at this point but there are some basic testing requirements that we can meet.

Flutter has built in support for unit and integration tests. These will be important to test the input UI and make sure that it is working properly. For these tests we will probably be implementing test driven development and blackbox testing where we simulate a user using the application. Also for the algorithm itself that handles data collection on an interval, we might implement some combination of black/white box testing.

For our API we can use tools like Postman to test our endpoints to make sure that they are performing correctly.

Risk Assessment

As use of the Spotify streaming service is integral to the functionality of our application, we will need to thoroughly review and continue to reference the Spotify Developer Policy throughout development to ensure we are using their API correctly under the published terms. One of these terms prohibits us from creating user “profiles” or collecting identifiable data on individual users through our use of the Spotify API. We will need to ensure that user data is sufficiently anonymized before being stored in our database while also ensuring that we are able to collect a useful amount of user metadata and prevent duplicate responses.

A possible failure scenario would be some sort of issue using streaming SDKs like Spotify. Whether that be a difficulty in implementing the service, or perhaps them suspending the service based on the type of application we are building.

Another possible point of failure would be the App Store refusing to host our iOS application based on their requirements and guidelines. We will need to be very diligent about abiding by these and hopefully the premise of the application will not be prohibitive to them accepting it. Apple is known to be very inconsistent and hard to communicate with during this process.

Another issue might be the ability for the server to handle the capacity of users input. As it stands, we intend on using AWS Lambda functions to handle requests, and these can essentially scale to meet anything we might need it to, however, they will need to be able to connect to the database and there may be issues with connection limits being hit. Depending on how many users the application ends up serving, we may need to find a solution to this problem if it exists.

Appendices

Project Phases and Milestones

Phase I

Date	Milestone	Description	Complete?
October 13, 2022	Problem Statement	Establish the problem to be solved and the potential solution to the problem.	Yes
October 20, 2022	Requirements Document	Establish the outline of the project and its expected resulting product.	Yes
November 3, 2022	Initial Design Document	Describe the product in detail and specify its architectural components.	Yes
November 10, 2022	v0.0.1	Demonstrate very basic functionality of the product's underlying components.	
November 21, 2022	Final Design Document	Edit the Initial Design Document into a finished version, including design changes and external input.	
November 21, 2022	Client Verification	Request approval from the project sponsor to move forward with development.	
November 30, 2022	v0.0.2	Demonstrate basic functionality of the application.	

Our Phase II and Phase III plans will be written at the start of the second and third terms of the project.

UI Mockups

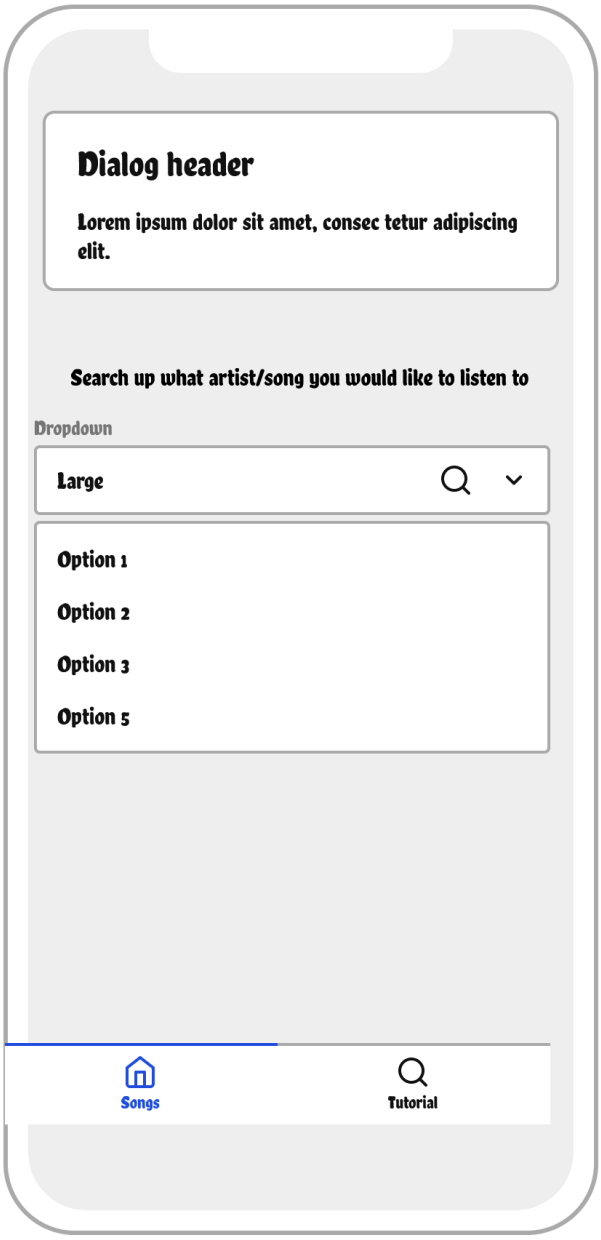

Figure 8
Home page that a user would land on initially.

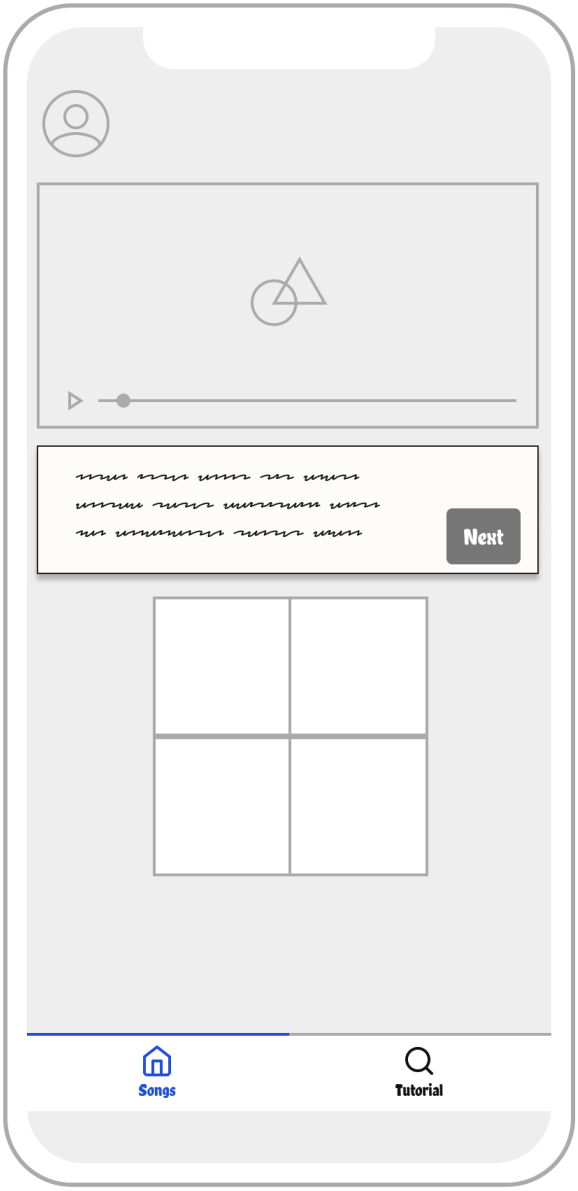

Figure 9

Tutorial page where the user is guided through how to select songs and enter data.

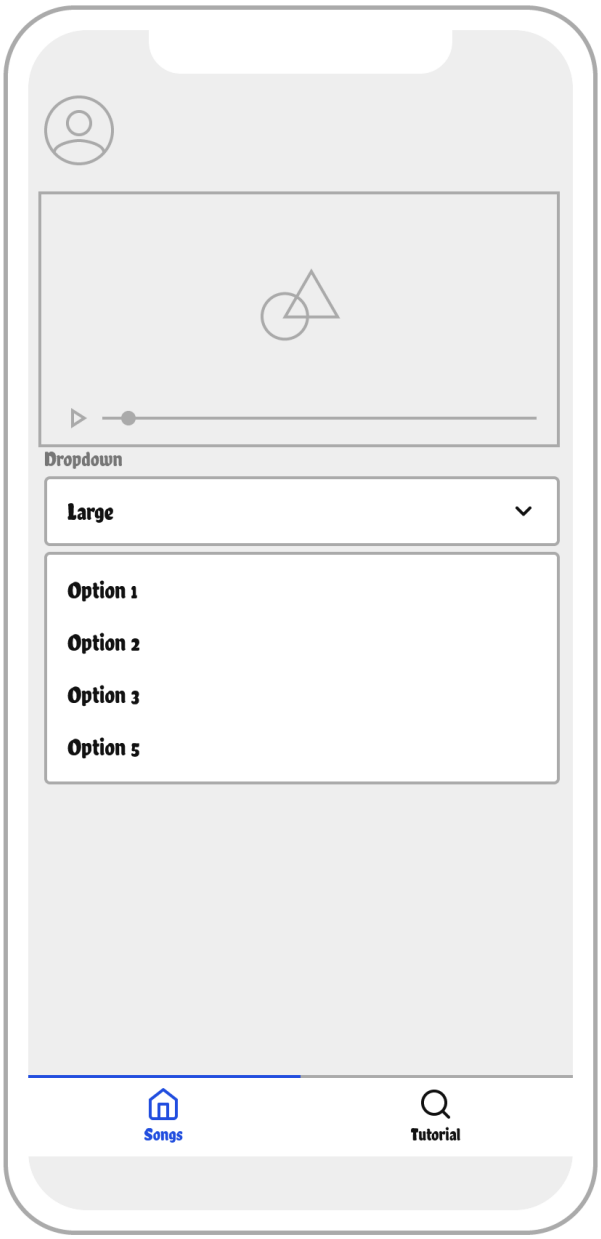

Figure 10

Song selection example.

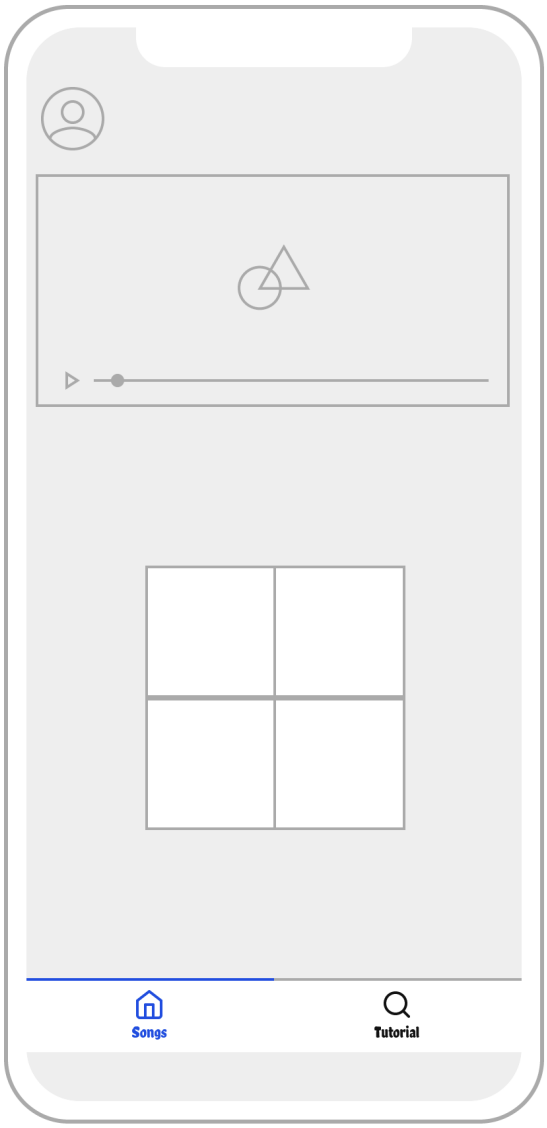

Figure 11
Affect Response Page, where the user would enter data.

Questions / Answers

Q: For input, do we want a range of values for each emotion, meaning, do we want to simply have “excited”, or have a range of how excited, e.g. 0.1 - 1.0?

> A: This depends on the implementation method, but for the most part the more information the better. We can look at existing studies to determine what a theoretical range of values should look like, and how we can standardize these values to compare existing studies with our own results.

Q: Are we setting up the cloud services, (e.g. server, database), ourselves?

> A: We can make use of the university’s AWS resources to work with an existing cloud infrastructure. We may still have to make some of the backend work ourselves but we won’t be starting from scratch or finding out how to host our own data.

Q: Should we prioritize usability (ease of self-reporting emotional data, simpler reporting options) or detail + accuracy of data collection (more complex/time-intensive reporting options but more data per report)?

> A: A balance of the two is important. Our app won’t be effective if no participants are able to or even want to use it, but we also need to gather enough quantitative information per report that our data is meaningful.

Q: Who makes up the user base of this app?

> A: This app is intended to be a crowdsourcing solution to gathering research on music affect data. The app will initially be informally tested via the app’s developers and any other individuals interested in contributing to the quality assurance and bug-fixing stage of the development process. After the app’s development is complete, it will be accessible to the general public, though it may be deployed to particular research groups.

References

[1] A. Beery, “Predicting Music Emotion with Social Media Discourse,” thesis, 2022.

[2] N. Walker, “Spotify Patented Emotional Recognition Technology to Recommend Songs Based on User’s Emotions,” *Journal of Law and Technology*, 11-Jan-2022.

[3] R. Delbouys, R. Hennequin, F. Piccoli, J. Royo-Letelier, and M. Moussallam, “Music Mood Detection Based on Audio and Lyrics with Deep Neural Net," in *Proceedings of the 19th International Society for Music Information Retrieval Conference, 2018, Paris, France* [Online]. Available: IRCAM, http://ircam.fr [Accessed: 14-Oct-2022].

[4] A. Aljanaki1, Y. Yang, and M.Soleymani, “DEAM: MediaEval Database for Emotional Analysis in Music”, PLoS ONE 12(3): e0173392. https://doi.org/10.1371/journal.pone.0173392 2017