Social Music Recommendation Platform

Design Document

Brendan Armstrong, Brooke Brown, Alyssa Kutney, Caitlin McElwee,

Caroline Grala, Kristelle Lucero

Github: https://github.com/LuckyOwl13/SMuRP

Table of Contents:

1. Product Specification
   1. Product Overview
   2. Product Features
   3. Product Architecture
2. Problem Solving Approaches
   1. Frontend approaches
   2. Backend approaches
3. Frontend
   1. Screen Mockup and Navigation
4. Backend
   1. List of Restful API Endpoints
   2. Database Schema
   3. User Authentication/Data Security
5. Screen Functionality
6. Tech Stack
7. Mid-assessment
   1. List of Goals to be Accomplished
   2. Task Breakdown
8. Works Cited

Product Specification

Product Overview

SMuRP is an app that allows users to discover new music and make friends based on their musical tastes. Users record the music that they listen to (marking their likes and dislikes), and a machine learning algorithm matches them to music they are likely to enjoy based on their musical preferences. In addition, the service also suggests other users to follow with similar musical tastes. Users can then view what music the people they follow are listening to.

Product Features

* Seamless recording of listening history on supported platform(s)
* Recommend music based on the user’s tastes
* Recommend friends based on both users’ tastes
* Ability to like or dislike recommendations

Disclaimer: SMuRP is not intended to be a typical social media experience. Users will not be able to upload photos or post status updates.

Product Architecture

Our SMuRP product can be broken up into two parts: the frontend and the backend. The frontend contains the user-to-product interaction, i.e. the phone application. This is where the user can create a SMuRP account and then browse through the app to check their feed. Through the interface the user is able to call commands that like/dislike songs, grab recommendations, have a user profile, and more (listed further in this document).

The backend is composed of the database and a RESTful API that connects the frontend to the backend. The database holds all appropriate data: user data, song data (including albums, songs on album, artists), and recommendations. The RESTful API provides all the needed methods for the phone application: database calls (adding a user, grabbing a users likes/dislikes, checking password), and methods relating to the recommendation functionality.

Problem-Solving Approaches

Frontend Approaches

We considered creating our app using Xcode. Xcode is the software that people use to develop apps for IOS. We thought about it because some of our group members had previous experience developing in Xcode and other members would have liked to learn how to develop iPhone apps. We ultimately decided against Xcode, however, because not everybody in our group had regular access to an Apple computer. Xcode is only available on Mac devices.

We also considered creating our app using Android Studio. Android Studio is the software used to develop Android apps. We considered it because of the fact that Android Studio is open to everybody. It was an alternative choice that meant we would not need to worry about what type of computers we had to work with. Android Studio was also a good choice because there is a lot of documentation online and we had a few members with prior experience. We ended up deciding against Android Studio, however, because we wanted to take this project as an opportunity to develop new skills.

In the end, we decided to develop our app using Flutter. Flutter allows for the creation of an app that can be supported by both Apple and Android phones. With Flutter, we could develop an app without having to worry about whether or not the phone that downloaded our app was an Apple phone or an Android phone. We decided to work with Flutter because it was developed fairly recently and we wanted to try something new.

Backend Approaches

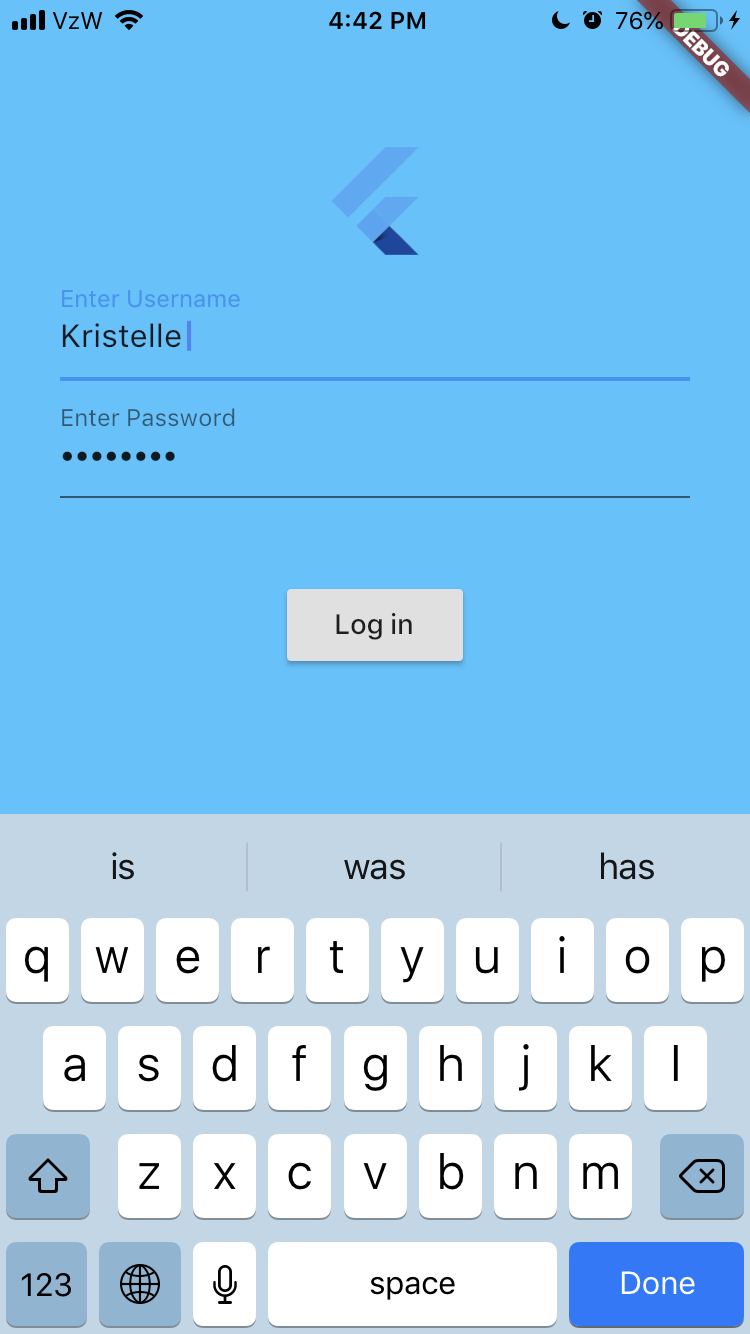
We considered both SQL and NoSQL solutions for the database. While NoSQL solutions do offer high scalability, we elected to use MySQL due to high availability of support. We also considered both a traditional relational approach as well as an ORM approach for the database. Ultimately, the convenience and readability of the ORM approach provided by SQLAlchemy won us over.

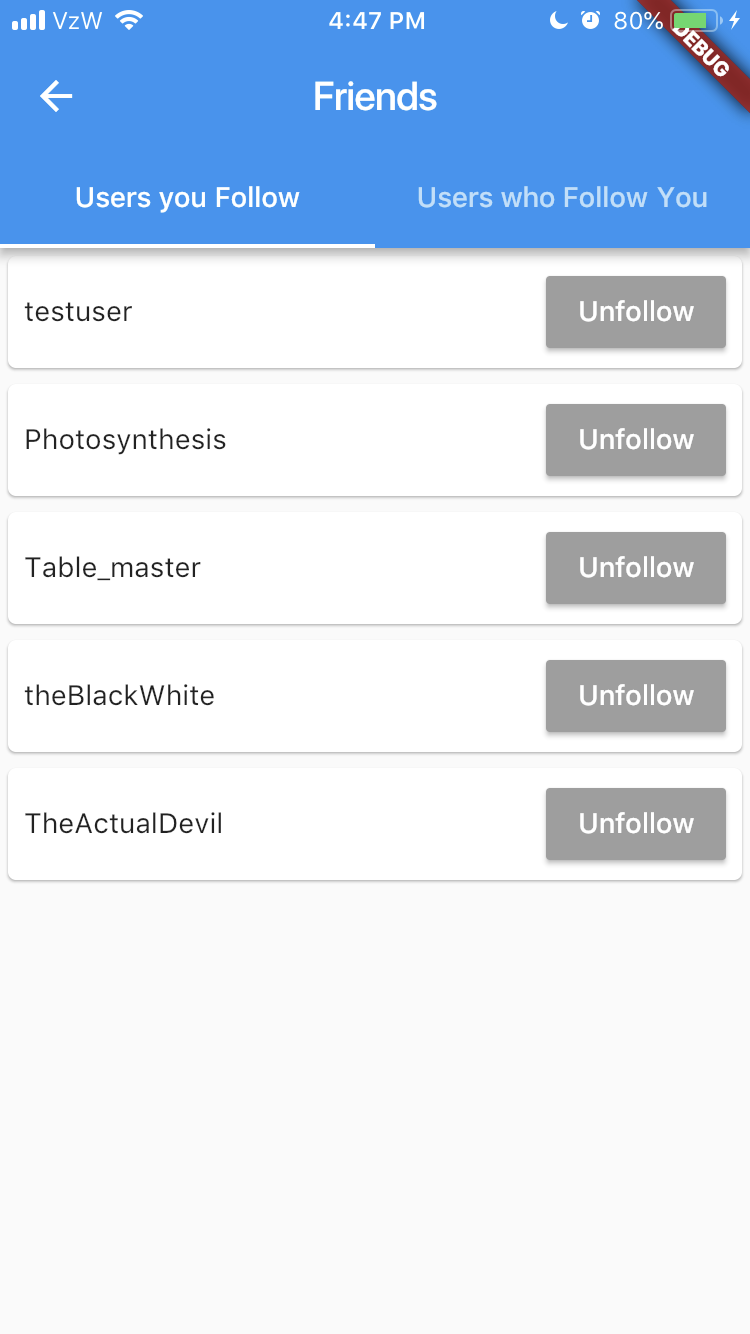
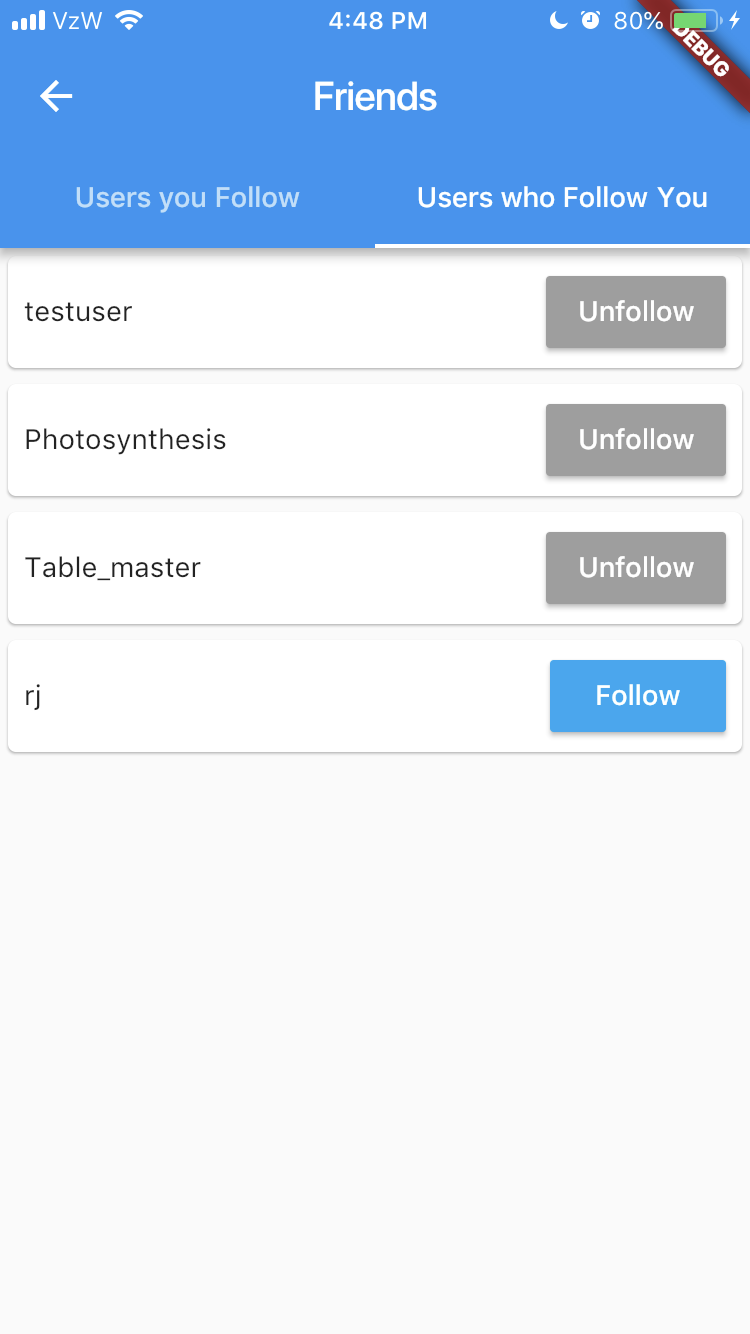
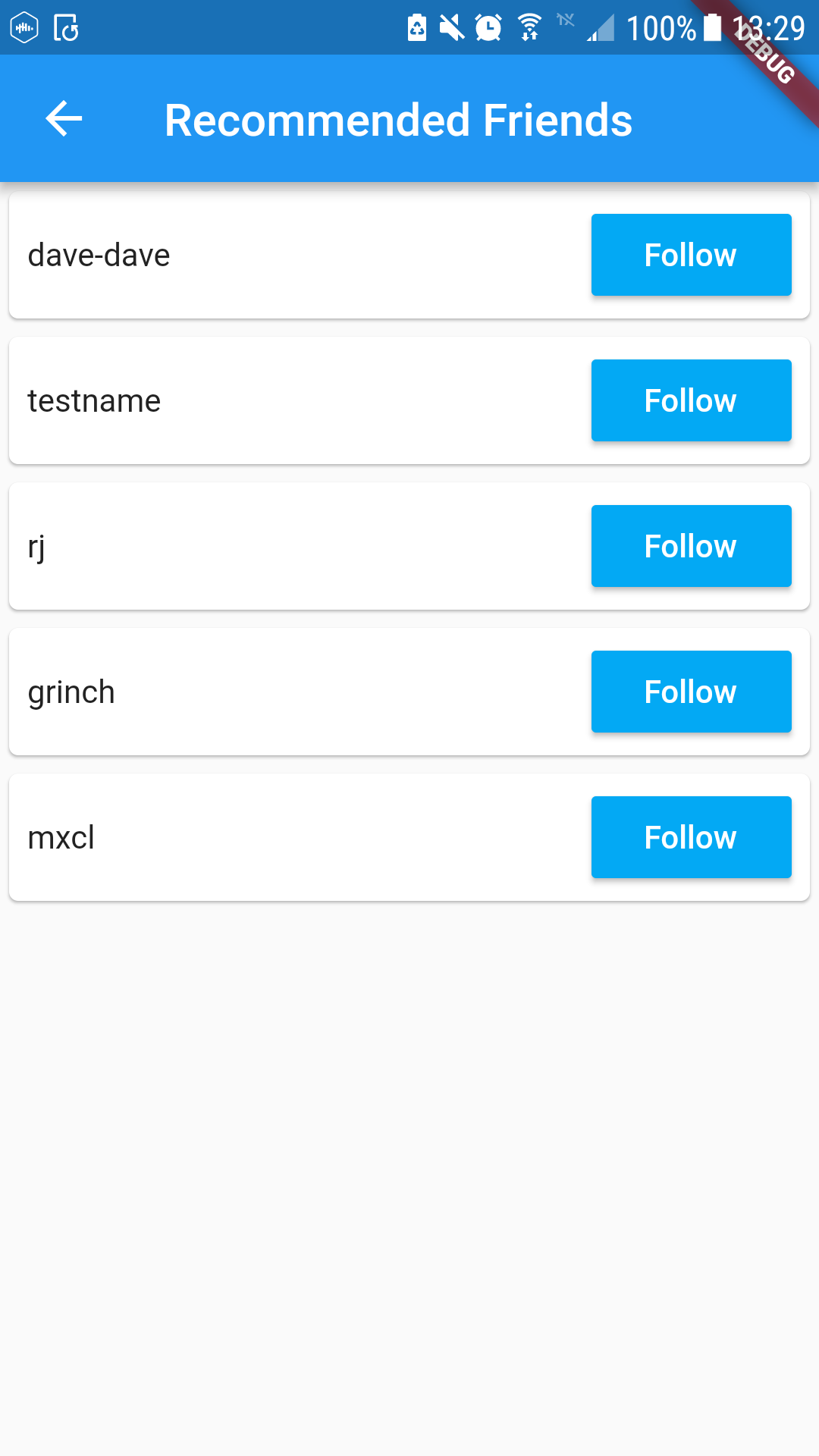
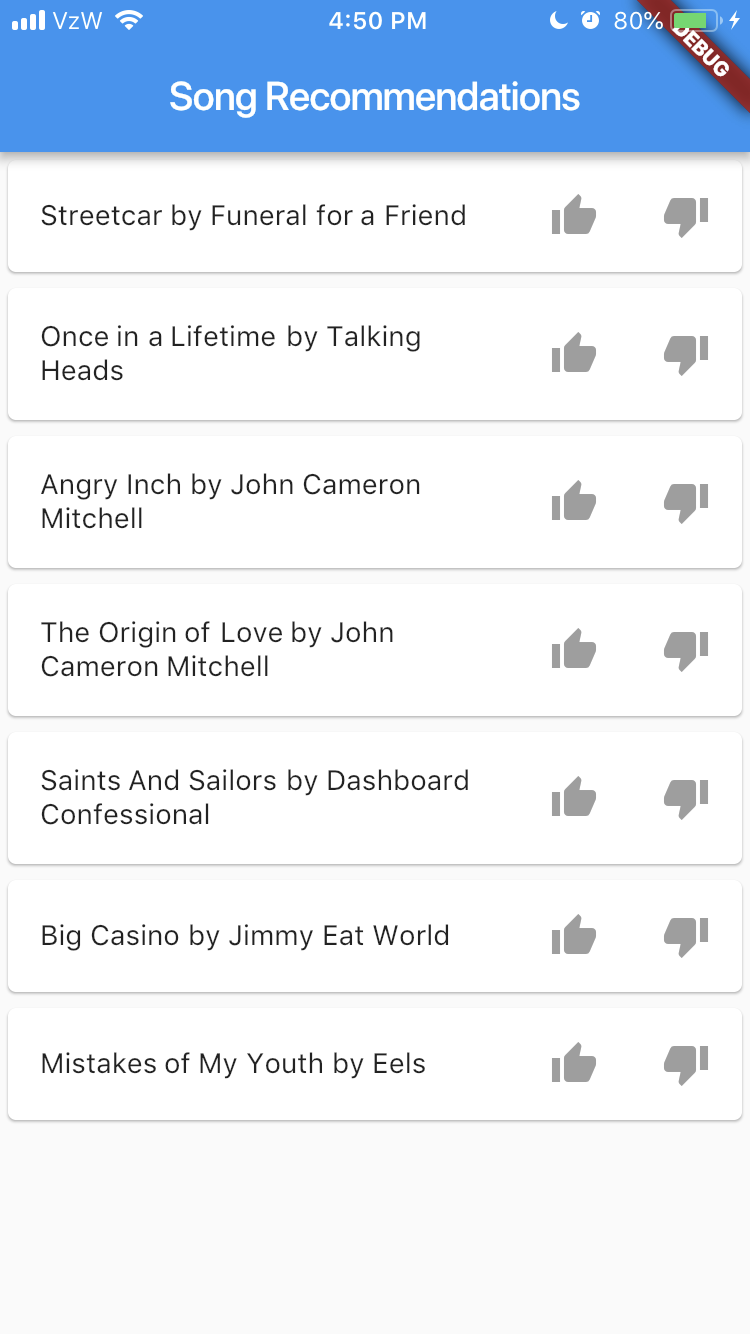
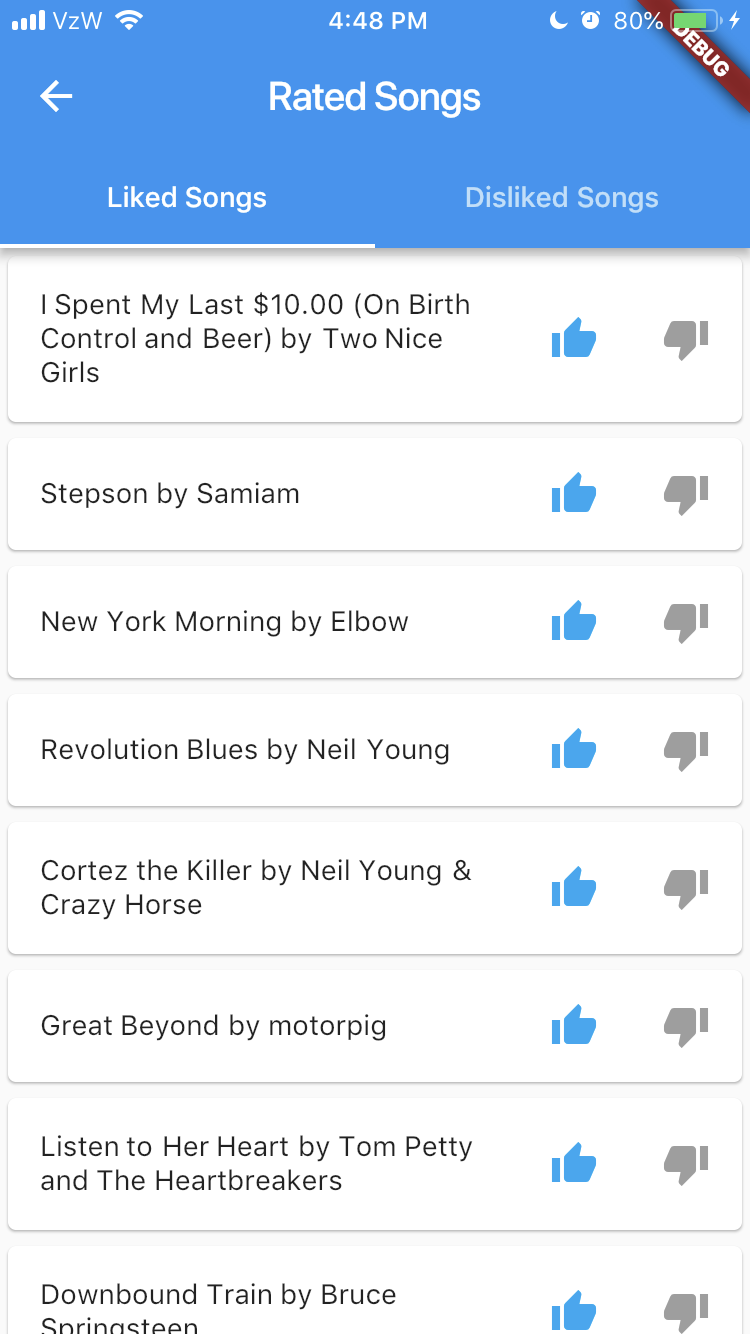
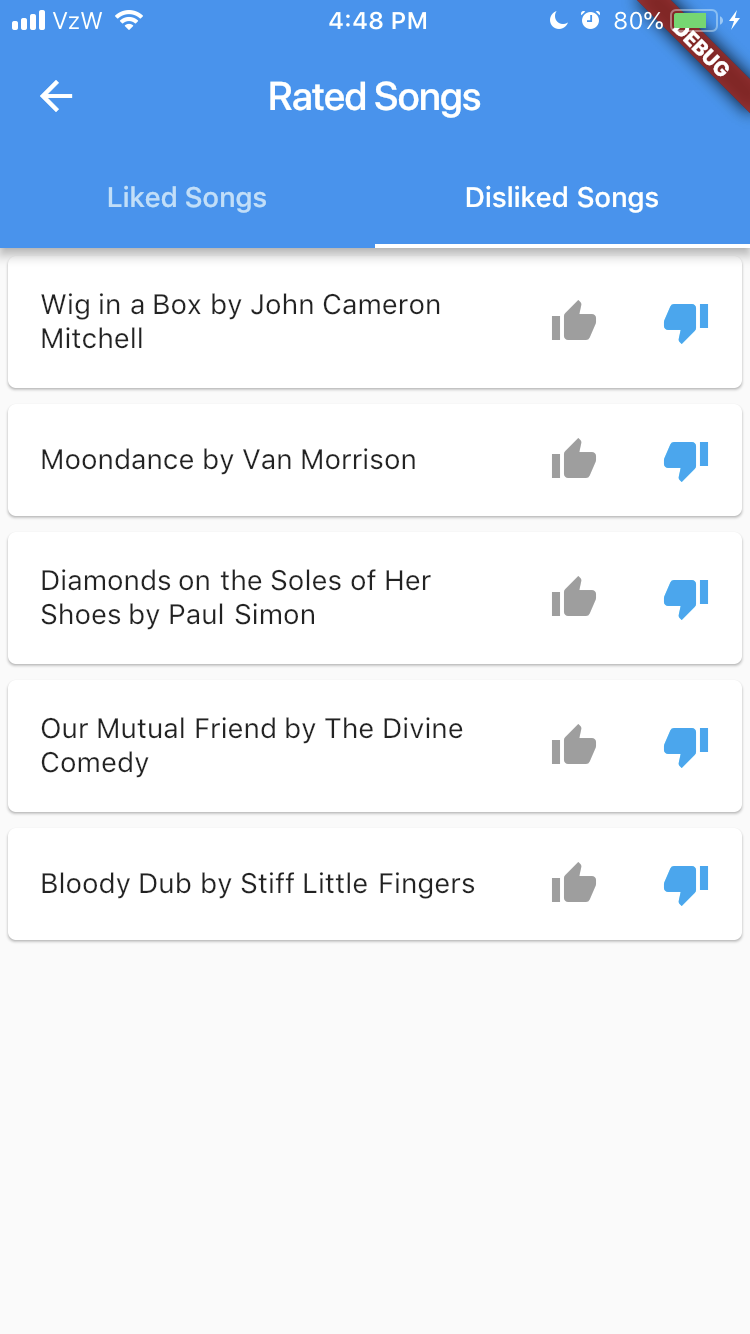
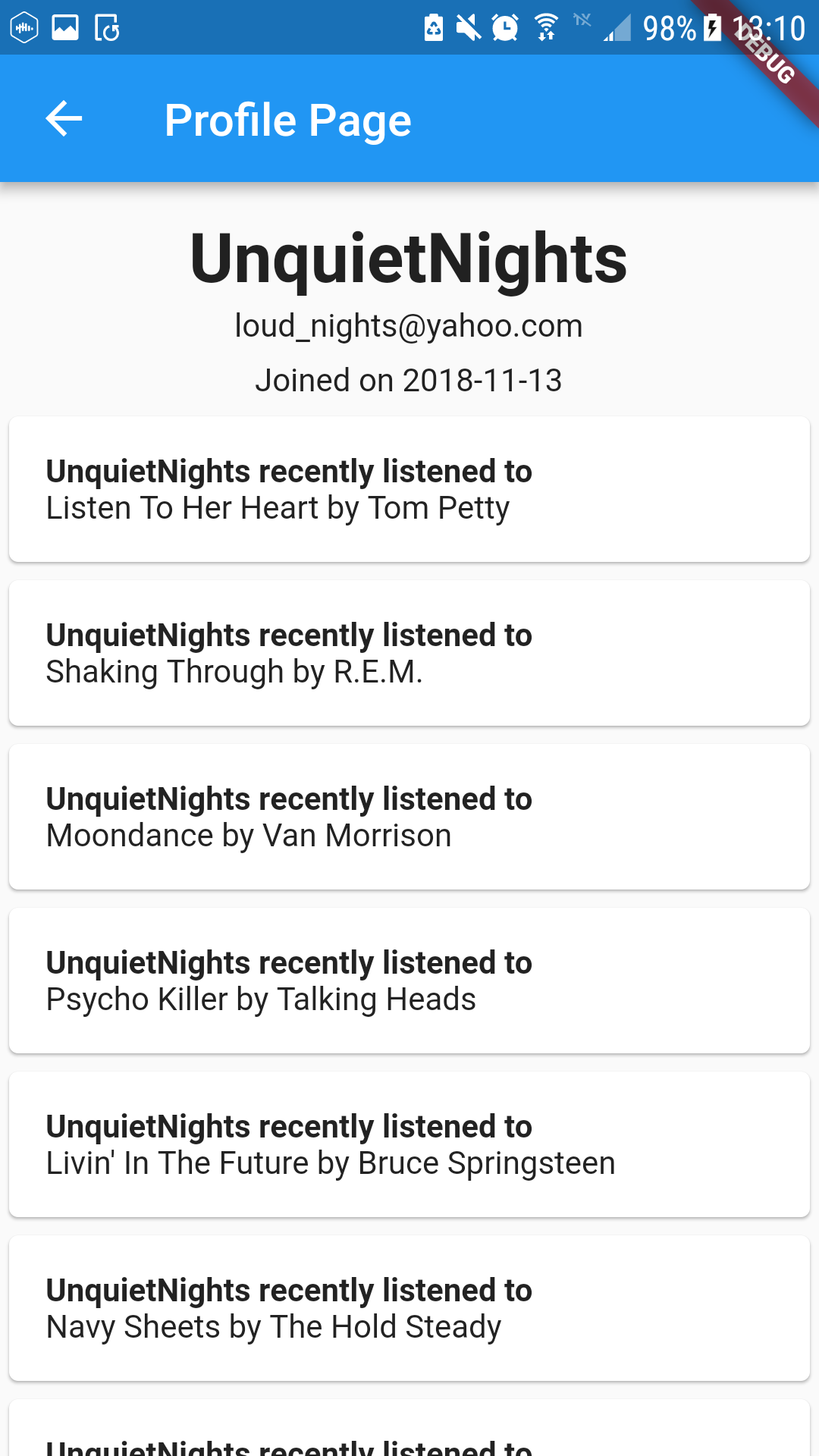
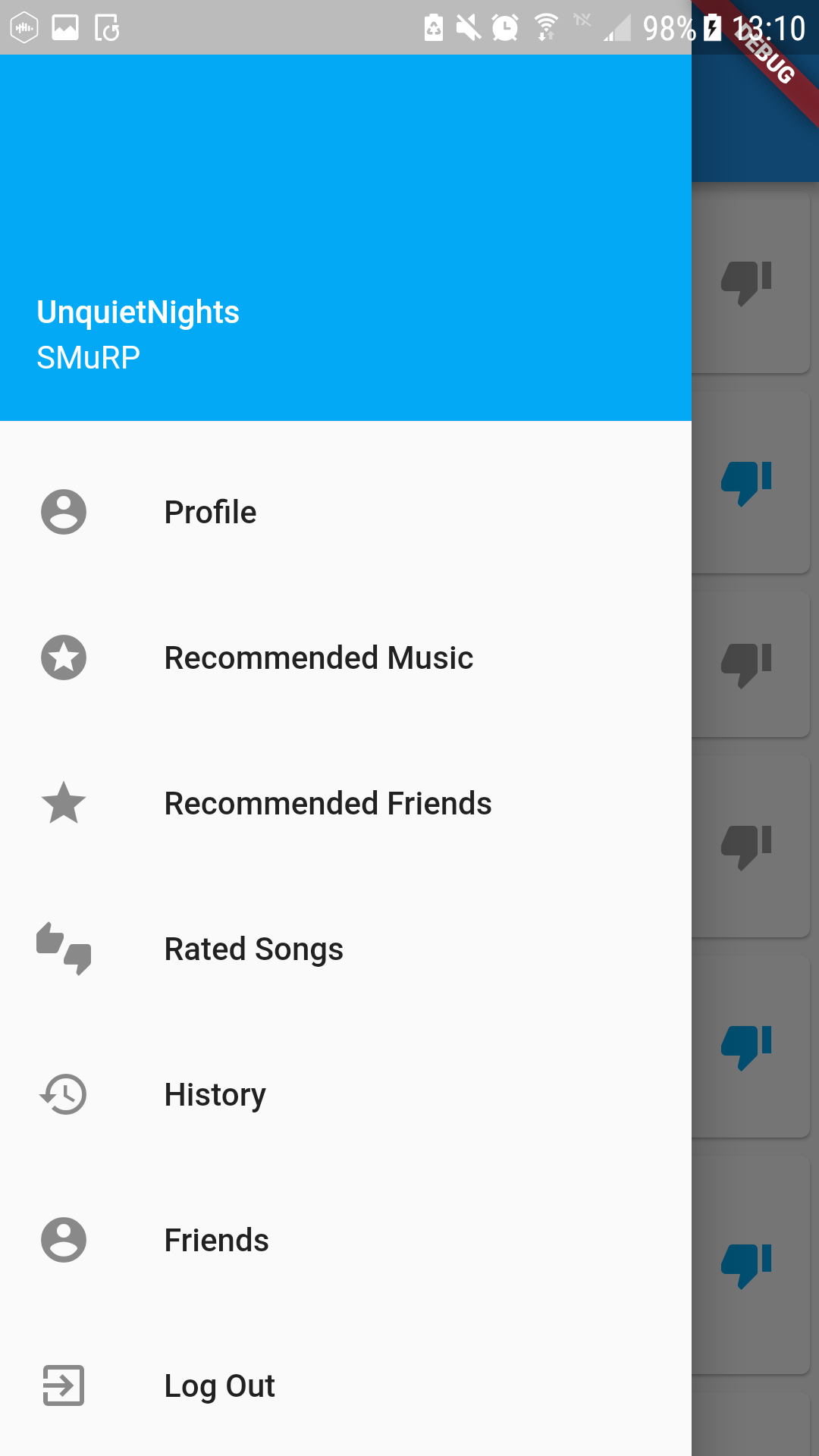
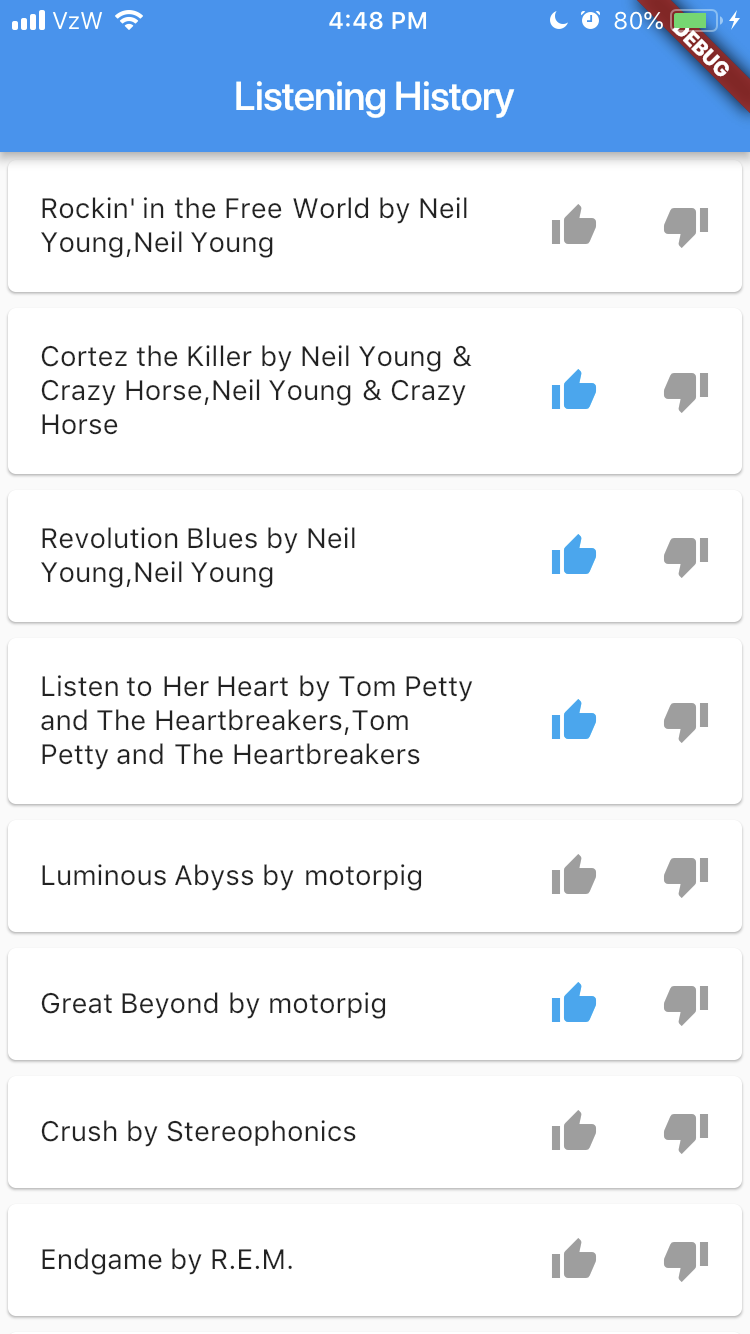
We wanted to use Python for backend development, because it would be a learning opportunity for the team and also the best approach for the machine learning aspect of our project. In regards to the RESTful API, our team was recommended to utilize Flask to create the RESTful API. Another Python framework we looked into during our research was Django. Through our research, we learned Flask gives you more control over which components to use, such as the database type and interactions. We ultimately decided to use Flask because of its simplicity and and flexibility.

To create a functioning recommendation system, we will be using SciPy’s Singular Value Decomposition (SVD) algorithm. SVD is a popular algorithm for building a recommender system. We used SVD as a collaborative filtering algorithm to predict songs a user would like based on their previous ratings of songs. We considered using Amazon’s Sage Maker, but ultimately chose SciPy’s SVD algorithm because of its extensive documentation and detailed examples.

Frontend

Screen Mockup and Navigation:





Backend

List of all RESTful API endpoints:

/getfollowing(user\_id)

* Input payload: integer user\_id
* Functionality: /getfollowing will take in a user\_id of a specific user and will return a list of User objects of all the users that the specified user is following.
* Output payload: json of User objects.

/getfollowers(user\_id)

* Input payload: integer userID
* Functionality: getFollowers() will take in a userID of a specific user and will return a list of User objects who are following the specified user.
* Output payload: json of User objects.

/follows(user\_id1, user\_id2, session\_key)

* Input payload: integer user\_id1, integer user\_id2, String session\_key
* Functionality: /follows creates the ability for a specific user to follow another user. To allow User1 to follow User2, a Session Key is set to verify that User1 is a valid logged in user that is trying to perform this function. Once the Session Key is verified, User1 will then be a follower of User2. If the function is successful, a boolean with the value of “true” is returned. If it is unsuccessful, a boolean with the value of “false” is returned.
* Output payload: returns successful message if User1 successfully follows User2. If User1 does not successfully follow User2, an unsuccessful message will be returned. The session\_key will also be returned

/unfollows(user\_id1, user\_id2, session\_key)

* Input payload: integer user\_id1, integer user\_id2, String session\_key
* Functionality: /unfollows creates the ability for a specific user to unfollow another user. To allow User1 to unfollow User2, a Session Key is set to verify that User1 is a valid logged in user that is trying to perform this function. Once the Session Key is verified, User1 will no longer be a follower of User2. If the function is successful, a boolean with the value of “true” is returned. If it is unsuccessful, a boolean with the value of “false” is returned.
* Output payload: Returns successful message if User1 successfully follows User2. If User1 does not successfully follow User2, an unsuccessful message will be returned. The session\_key will also be returned.

/addListenedTo(int userID, int songID)

* Input payload: integer userID1, integer songID
* Functionality: addListenedTo() takes in the userID of a specific user and a songID of a song and adds that the user has listened to that song at the current time, forming that relationship in the database.This endpoint is used for testing purposes only.
* Output payload: returns success message if relationship was created successfully.error message if unsuccessful

/getListened(int userID)

* Input payload: integer userID
* Functionality: /getListened takes in the userID of a specific user and outputs all the songs that a user has listened to.
* Output payload: a json of songs including title and artists.

/likedsongs(int userID)

* Input payload: integer userID
* Functionality: /getlikedsongs will take in a userID and return all the songs a user has liked through the app.
* Output payload: json of songs including title and artists.

/dislikedsongs(int userID)

* Input payload: integer userID
* Functionality: /getdislikedsongs will take in a userID and return all the songs a user has disliked through the app.
* Output payload: json of songs including title and artists.

/loginuser(username, password)

* Input payload: string username, and string password
* Functionality: /loginuser will hash the user’s password and compare the value to that of the hashed value in the database. Returns json of session key or failure if login fails. Everytime a user logs in, last.fm api method getrecenttracks() is called for this user, and loops through to retrieve and store a user’s most recent songs that have been added since last login.
* Output payload: json of encrypted session key if login is successful. Returns json of with error message if login fails.

/getfeed(user\_id, user\_only, session\_key)

* Input payload: Integer user\_id, integer user\_only, string session\_key
* Functionality: /getfeed returns information about users the selected user is following, including what songs the users has listened to and what songs they have liked/disliked. User\_only acts out as a boolean (MySQL can’t store boolean values and can be a 0 or 1. If user\_only is set to 1, get\_feed only returns information about the user who’s user\_id is input. If user\_only is 0, get\_feed returns information about the selected user and users they are following.
* output payload: JSON list of updated information that includes information on what the user and who user follows has liked/disliked and listened to recently.

/logout(username, session\_key)

* Input payload: String username, String session\_key
* Functionality: /logout logs out a user
* Output payload: a String stating that the user has logged out.

/like(user\_id, song\_id, session\_key)

* Input payload: Integer user\_id, Integer song\_id, String session\_key
* Functionality: /like allows the selected user to like a song. A session key is set in place to verify that the specific user\_id is a valid logged in user. Once verified, the song\_id will then be liked by the user.
* Output payload: successful/unsuccessful, session\_key

/dislike(user\_id, song\_id, session\_key)

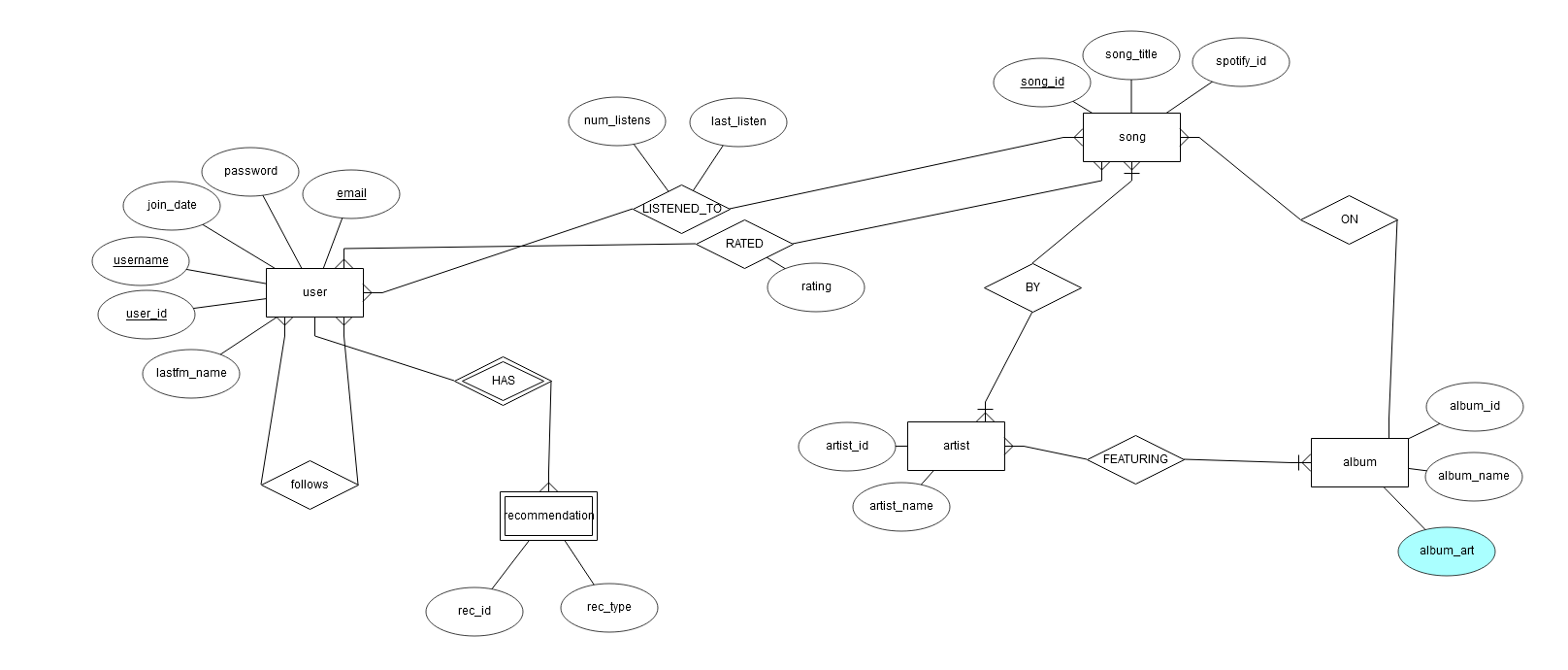
* Input payload: Integer user\_id, Integer song\_id, String session\_key
* Functionality: /dislike takes in a user\_id, a song\_id, and session\_key. A session key is set in place to verify that the specific user\_id is a valid logged in user. Once verified, the song\_id will then be liked by the user\_id.
* Output payload: successful/unsuccessful, session\_key

/recommend(user\_id, session\_key)

* Input payload: Integer user\_id, String session\_key
* Functionality: /recommend will take in a user\_id and a session\_key, making a recommendation of songs based upon the user’s already rated songs. The session\_key is used to verify that the user is a valid logged in user.
* Output payload: json of predicted songs for the user\_id

/recommendusers(user\_id, session\_key)

* Input payload: Integer user\_id, String session\_key
* Functionality: /recommendusers takes in a user\_id and session\_key. Recommends potential users for a specifc user to follow based on similar music interests.
* Output payload: json of user\_ids and usernames of recommended users.

Database Schema:

User Authentication/Data Security:

In order to protect our users’ data along with all the data in our database, we will utilize two methods. Specifically with user passwords, we will be using the python package passlib. We will first hash the inputted password at the login screen. passlib hashes passwords with either the md5 or SHA256 algorithm. We will be utilizing SHA256 as it is inherently better than md5. md5 is considered “broken” due to large numbers of people discovering ways to create collisions (Information Security Stock Exchange). After the password has been hashed using the SHA256 algorithm, we then compare the hashed inputted password to the hashed password in the database. If the two values match, the user can login - otherwise they are given an “incorrect password” warning (Password Hashing with Flask Tutorial). This process will occur on the database server.

As for protecting the database, the database is stored on the AWS server. Therefore, no one can have access to the database unless they know the login for the AWS server. The AWS server’s login credentials are stored in a safe place on everyone’s computer so no one can access the AWS.

For session key encryption, we used Cryptography’s Fernet package. Fenet is an implementation of symmetric authenticated cryptography. We use Fernet to generate a hashed key and store it on our server. We used this key to encrypt in the log in endpoint, and decrypt in all the endpoints that modify data. The way we create session keys are by appending usernames with the current date and time. After we decrypt in an endpoint, we grab verify that the user making the request is a valid logged in user.

Screen Functionality

Login:

The *login* screen will be the first screen that is displayed. It will take the user’s username and password as input. It will then pass those values along by hitting the */loginuser* endpoint. If successful, the user will then proceed onto the next screen.

It hits the following endpoint: /loginuser

Home Feed:

The *home feed* shows a list of updates from people the user follows. It will display their friends’ activity. Activity includes a list of songs the user’s friends have recently *liked* or *listened to*. The home feed also houses the navigation bar that directs to other pages and offers the user the ability to logout.

It hits the following endpoints: /getFeed, /like, /dislike, /logout

Profile:

The *profile* screen displays user information. The information displayed include the user’s name, the user’s join date, email, and a list of recent songs that the user has *liked* or *listened to*.

It hits the following endpoints: /get\_user, /getFeed

Music Recommendations:

The *music* *recommendations* screen displays a list of songs that the app recommends.

It hits the following endpoints: /recommend

Friend Recommendations:

The *recommendations* screen displays a list of songs that the app recommends along with a list of potential friend recommendations for the user.

It hits the following endpoints: /recommendusers

Friends:

The *friends* screen has two tabs that you can switch between. The first tab contains *Users you Follow*. This tab displays a list of all the people who the user follows. It also includes a button next to the person’s name which will provide the option to unfollow that person. The second tab contains *Users who Follow You.* This tab displays a list of all the people who follow the user. It also includes a button next to the follower’s name which will provide the option to follow that user.

It hits the following endpoints: /getFollowing, /unfollows, /getFollowers, /follows

History:

The *history* screen is a listing of all songs listened to or interacted with by the user. All songs are listed with artist of the song, the song’s title, and then two icons: a + and a -. If the user taps the (+), the song is “liked” and added to their list of liked songs; the same happens with (-) and their list of disliked songs. When one of those icons is tapped, it turns into an orange thumbs-up (if +) or thumbs-down (if -). Tapping one of the icons when it is orange reverts it to the +/- and removes it from the respective list. If one icon is currently orange and the other icon is tapped, the currently-orange icon reverts to its initial state and the other icon turns orange (denoting the song being removed from the one list and being added to the other list).

It hits the following endpoints: /getListened, /like, /disliked

Rated:

The *Rated* screen (“rated” screen) is a listing of all songs that have been liked or disliked, depending on which list is requested. On it, users see all songs on the respective list. Songs shown have the same functionality for changing their rating as described in the History screen.

It hits the following endpoints: /likedsongs, /dislikedsongs

Navigation Menu:

The *navigation menu* is not a separate screen. However, this menu will show an array of options which the user can tap to switch from screen to screen. Options include the ability to switch to the user’s profile, their music recommendations screen, friends recommendation screen, rated songs, history, and friends. It will also provide the option to log out of the app.

Tech Stack

Platform: Both Android and iOS

Development Tools: Android Studio, Xcode, Flutter

RESTful APIs: last.fm API

Backend: AWS EC2 Linux, Python Flask, MySQL, SQLALchemy

Designated Project Programming Languages

Frontend: Dart

Backend: Python

Mid-Assessment

List of Goals to be Accomplish

* Allow users to login and use persistent accounts
* Keep track of the music users have been listening to
* Allow users to create and maintain a list of followed users
* Allow users to view the listening history of other users
* Match users to music using machine learning techniques
* Match users to other users based on the similarity of their listening history

Stretch goals

* The ability for a user to recommend specific music to their followers
* A messaging system for mutual followers
* Recommendations for albums and artists
* Playing music through embedded links

Task Breakdown

Frontend- Kristelle Lucero and Caitlin McElwee

Backend- Brooke Brown, Alyssa Kutney, and Caroline Grala

Database- Brendan Armstrong

Works Cited

K. (2011, February 21). What are the differences between MD5, SHA and RSA? Retrieved

October 23, 2018, from https://security.stackexchange.com/questions/2298/what-are-the-differences-between-md5-sha-and-rsa

Password Hashing with Flask Tutorial. (n.d.). Retrieved October 23, 2018, from

https://pythonprogramming.net/password-hashing-flask-tutorial/

Dwyer, G. (n.d.). Flask vs. Django: Why Flask Might Be Better. Retrieved October 23, 2018, from https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v

“Scipy.linalg.svd¶.” *Scipy.stats.trim\_mean - SciPy v1.1.0 Reference Guide*,

docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html.

“Fernet (Symmetric Encryption)¶.” *Asymmetric Algorithms - Cryptography 2.4.dev1*

*Documentation*, cryptography.io/en/latest/fernet/.