My (Musical) Life

Design Document

Team Number: sddec20-13
Adviser & Client: Dr. Henry Duwe

Team Members:
Christian Hernandez - Project Manager
Chaz Clark - iOS Developer
Daksh Goel - Backend Developer
Vignesh Krishnan - Frontend Developer
Vatsal Bhatt - Backend Developer

Team Email: sddec20-13@iastate.edu
Team Website: http://sddec20-13.sd.ece.iastate.edu/

Executive Summary

Development Standards & Practices Used

- Development Standards
  - Commented Code
  - Quality
  - Efficiency
  - Apple Developer Standards
  - Waterfall Design
- Practices
  - Test code regularly
  - Agile Development
- Engineering Standards
  - Quality
  - Performance
  - Safety

Summary of Requirements

- **Functional requirements**
  - User Data (from their mobile device)
    - Location
    - Weather (Team was not able to get to this due to time)
    - Schedule
  - Spotify Installed on iPhone
  - Music Recommendations
  - Mapping Sensor Inputs to Songs/Playlists
  - Volume Control (Team was not able to get to this due to time)
- **Non-functional requirements**
  - Security (SSL, TLS, WPA2)
    - Account logins
    - Location information
    - Calendar data
    - Music preferences
  - AWS Security
    - Database
    - Lambda data
  - Response time and performance
    - Crash Rate: 1-2%
    - API Latency: 1 sec
    - End-to-end app latency: <3 sec
- **Engineering Constraints**
  - Apple developer account
  - Coronavirus restrictions limiting movement
  - Must have an iPhone

    - MacOS for development
    - Developing app for both Android and iOS
    - User Buying Spotify Premium Account
- **Economical requirements**
    - Spotify Premium Subscription
- **Environmental requirements**
    - Network reception in the user's mobile device
    - iOS device (iPhone)
- **Apple Design Guideline Requirements**
    - Consistency
    - Feedback
    - Direct manipulation
    - User control

Applicable Courses from Iowa State University Curriculum

- S E 185 - Problem Solving in Software Engineering
- CPR E 185 - Introduction to Computer Engineering and Problem Solving I
- COM S 227 - Object-Oriented Programming
- COM S 228 - Introduction to Data Structures
- COM S 309 - Software Development Practices
- CPR E 310 - Theoretical Foundations of Computer Engineering
- COM S 311 - Introduction to the Design and Analysis of Algorithms
- S E 319 - Construction of User Interfaces
- S E 329 - Software Project Management
- S E 339 - Software Architecture and Design
- COM S 363 - Introduction to Database Management Systems
- ENGL 314 - Technical Communication

New Skills/Knowledge acquired that was not taught in courses

- Swift
- iOS Development
- Machine Learning
- Using Spotify's API
- Amazon Web Services

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

We would like to first thank Dr. Henry Duwe for meeting with us weekly and providing us with guidance and advice as we develop our senior design project. Dr. Duwe has done an amazing job in regards to helping us set up this project by giving us small assignments to complete each week for him. We would also like to thank the Electronics Technology Group for providing us with a website and a Git project. Lastly, we would like to thank the TAs and the professors (Dr. Lotfi Ben-Othmane, Dr. Daji Qiao, and Dr. Thomas Daniels) for their guidance and help.

1.2 Problem and Project Statement

Have you ever been in a sad mood and listened to sad music despite it not helping and making you even sadder? Do you typically play high-tempo music when heading to the gym and while working out at the gym? Do you listen to softer, calmer music as you study for your next exam at the library? If you answered yes to some of the questions, *My (Musical) Life* app will be perfect for you! For people who love music, this app will be great to use daily.

Our app will use data from multiple different, possible sources (location, calendar, schedule, time of day, etc.) to determine which song is the best to pipe directly into your ears. The app will require little user input, and the music suggestions will improve as the user continues to use the app. Overall, as long as the app is open on your phone, the app will continue to play music based on the different sources listed above.

1.3 Operational Environment

The end product of our project is an iOS mobile application, as demanded by the client. *My (Musical) Life* will be able to be installed and used by anyone owning Apple's mobile device, namely, iPhone. Upon installation and registration, our app will require some permissions from the user including access to their mobile device's location and some user data. Our app is only supported by iPhone's Operating System and will be not available for use in mobile devices running Android.

1.4 Requirements

The requirements of our project are as follows:

- **Functional requirements**
    - User Data (from their mobile device)
        - Location
        - Weather
            - Unable to get to this requirement
        - Schedule

    - Spotify Installed on iPhone
    - Music Recommendations
    - Mapping Sensor Inputs to Songs/Playlists
    - Volume Control
        - Unable to get to this requirement
- **Economical requirements**
    - Spotify Premium Subscription
    - Apple Developer Account

- **Environmental requirements**
    - Network reception in the user's mobile device
    - iOS device (iPhone)

- **Apple Design Guideline Requirements**
    - Consistency
    - Feedback
    - Direct manipulation
    - User control

1.4.1 Engineering Constraints and Non-Functional Requirements

- **Engineering Constraints**
    - Apple developer account
    - Coronavirus restrictions limiting movement
    - Must have an iPhone
    - MacOS for development
    - Developing app for both Android and iOS
    - User Buying Spotify Premium Account
- **Non-functional requirements**
    - Security (SSL, TLS, WPA2)
        - Account logins
        - Location information
        - Calendar data
        - Music preferences
    - AWS Security
        - Database
        - Lambda data
    - Response Time/Performance
        - Crash Rate: 1-2%
        - API Latency: 1 sec
        - End-to-end app latency: <3 sec

1.5 Intended Users and Uses

An intended user for our iOS mobile application, *My (Musical) Life*, will be someone who loves to listen to music and wants to listen to their favorite music with minimal to no user input at specific times and during specific events of their day. Our app will create personalized playlists for each user depending on their mood, location, and schedule. Other factors including time of the day will also play a role in creating these playlists. Weather is something that the team did not get to, but this can be added to the app in the future. The main purpose of the app is to start playing music on a user's mobile device without their input, during specific times of the day when the user would possibly be wanting to listen to music. An example use case will be a student wanting to listen to soft and calm music while studying in the library. Our app will determine that the user i.e., the student is in the library through the location of their mobile device and will start playing soft and calm music in their mobile device while having five other playlist recommendations in case the user did not like the music that is being played currently.

Pictured below is a Venn diagram consisting of all the use cases for our project.

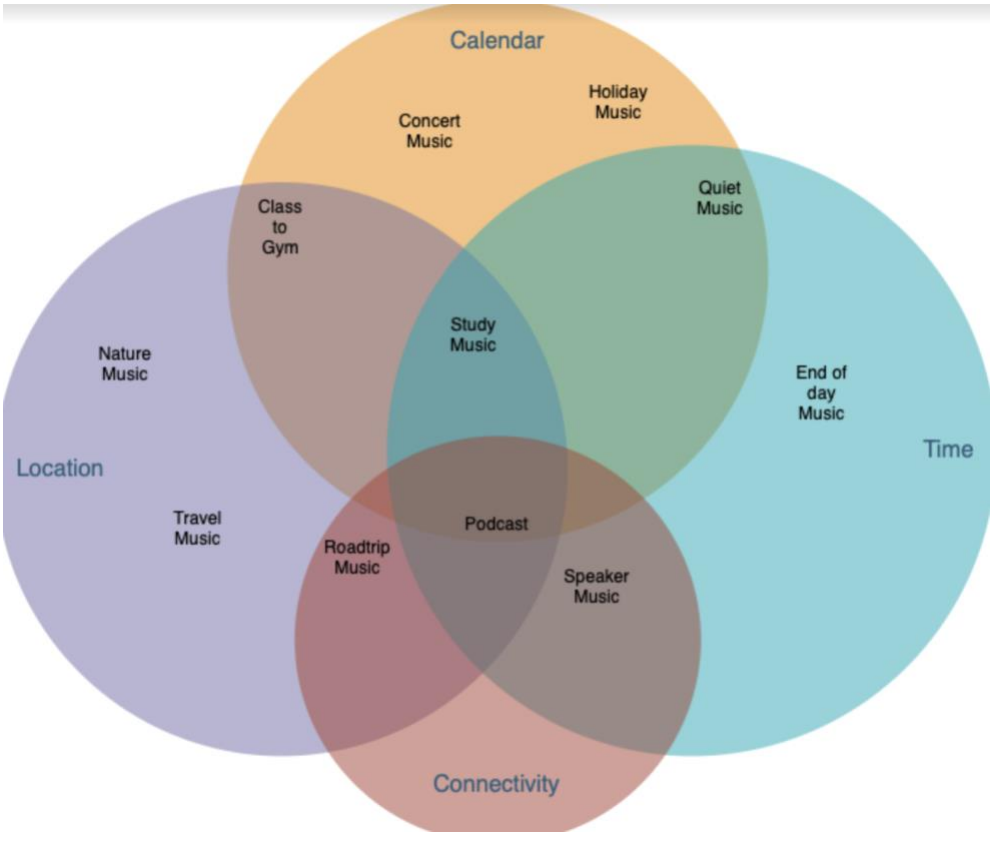


Figure 1: Use Case Venn Diagram

1.6 Assumptions and Limitations

Assumptions:

1. The user will have an existing account with Spotify music streaming service or will be willing to create one for the use of the application.
2. The user will have an existing Spotify library to choose songs from, otherwise, songs may be chosen from a random Spotify playlist for a specific mood.
3. The user will have an existing Google account or will be willing to create one for the use of the application.
    a. Originally, this was an assumption for the app. However, the user will not need a Google account for the app anymore.
4. The user is comfortable with the private data and permissions that the application requires to provide the most intelligent song selections. Ex. location, calendar, and Bluetooth.

Limitations:

1. The application will only be available on iOS devices as required by the client.
2. Users with streaming services other than Spotify or local music storage will not be able to integrate their music library.
3. Users without a Google account will not be able to use features that use personal data for selecting songs.
    a. Originally, this was a limitation for the app. However, a user will not need a Google account for the application.
4. There will have minimal user input and songs will only be able to be played automatically with a skip feature, not chosen specifically.
5. The user will need to allow device location permissions at all times to receive location-based song selections in real-time.
6. The user will need to allow Bluetooth device connection permissions at all times to receive device-based song selections in real-time.
    a. Originally, this was a limitation for the app, However, a user will not always need a Bluetooth connection.
7. The user will not be allowed to test the app everywhere if there are local restrictions due to COVID-19

1.7 Expected End Product and Deliverables

The *My (Musical) Life* iOS application will be the final deliverable to our client and it will be commercialized on the Apple App Store.

Description:

My (Musical) Life is a music streaming application that changes your music based on what you do every day. This is a great app for listeners who love having music playing throughout the day, almost like a soundtrack for their life! Finding the right song, playlist, or genre for a particular activity can be difficult and time-consuming. *My (Musical) Life* can take the pain out of choosing music by predicting and playing the songs you like. Powered by Spotify, *My (Musical) Life* will select your favorite songs that fit your daily activities. Whether it be working out at the gym, working in your office, studying in the library, or taking a road trip, never worry about changing

the song again. *My (Musical) Life* will recommend and play music that best fits your daily routine. Using your location, calendar, and Bluetooth connectivity data, it will select the genre of music that best matches your mood for a specific activity. As you provide feedback on selections made by *My (Musical) Life*, it will build a personalized music profile that will only play the songs that you love.

2. Specifications and Analysis

2.1 Proposed Approach

The proposed approach to our app is to develop an iOS application for our users to interact with. The iOS application will then communicate to various third-party APIs. In order to play music, our app will use Spotify's API and stream music from there.

The application will be mapping users' sensor data to songs and playlist. The mapping of this data will be organized using a bin model approach where users will have bins that are built upon the user's sensor data and will contain songs associated with that data. To see if a bin already exists for the user's current situation, we will be designing AWS lambda functions to search for appropriate bins, if none exists a new one will be created.

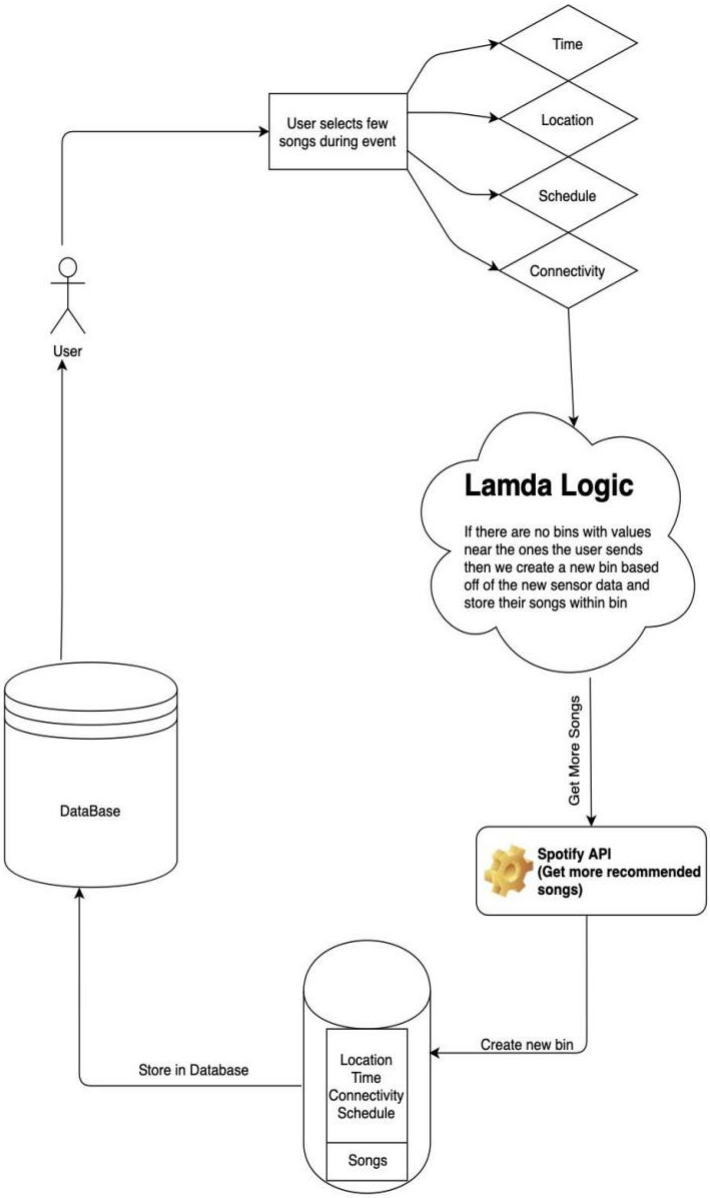


Figure 2: Bin Creation Diagram

For our data storage, we plan to use a hybrid of on-device storage and AWS. The on-device storage will store user-login, and settings. Our AWS backend will build a profile of the type of music the users listen to, which will help us recommend music. We will also use the backend to recommend the next song(s) to the user. This can either be ML or a predictive algorithm (as stated above). The approach the team took for predicting the next songs was the predictive algorithm. The team has implemented an application that uses Spotify's API to play music. The

past months have involved learning, developing, and testing to ensure the app meets this proposed approach.

2.1.1 Sensor Inputs

Pictured below is a list of sensor inputs and the results for each of these. More specifically, these are APIs or frameworks the team will use to gather the inputs. Using these APIs or frameworks, we will be able to gather the result needed. For example, the app will be able to use the Core Location API in order to obtain the location, movement, etc. of the user.

Inputs (API/Framework)	**Result**
Core Location	Location, Movement
Healthkit/Core Motion (Didn't get to this)	Steps, Heart Rate, Movement, Date of Birth
IO Kit (Didn't get to this)	Bluetooth Connectivity
Core Motion (Didn't get to this)	Start Date, End Date, Number of steps, Distance, etc.
IOBluetooth UI (Didn't get to this)	Bluetooth Connectivity
CarPlay (Didn't get to this)	Bluetooth Connectivity to car
Asset Playback (Didn't get to this)	Control Volume
Google API (Didn't get to this)	Schedule/Calendar, Sign-In
Spotify	Return Songs, connection status, etc.
Open Weather API (Didn't get to this)	Temperature, Wind Speed, Cloudiness Percentage, etc.
Dates and Times	Date and Time

Figure 3: Sensor Input Table

2.1.2 Database Design

Below is a diagram of our database design. As you can see, we will use the User table to store user information. Originally, the team was planning on using Google authentication which would help with storing passwords. However, since the team did not go that route, we needed to store passwords. We will store hashed email addresses as well as the age of the user. The team planned to store the region the user resides in. However, the team did not end up adding this information. Next, we have the UserSettings table which is where we store the values for the personalized UserSettings such as if they would like to enable or disable location services and other information. Volume control is another setting that the team did not implement for the app. Each User will have many bins. To model this

one-to-many relationship, we have the Bin table. Each Bin will have a user associated with it. The bin will be defined by the various sensor inputs that we have discussed so we have a field for each of these inputs. Similar to the user-bin relationship, each bin will have many songs, however, different bins may share the same song, therefore this is a many-to-many relationship. To represent this, we have created a Songs table, where each SongName will be associated with a bin. The primary key here ensures no duplicate values. The revised database design is shown in Figure 5.

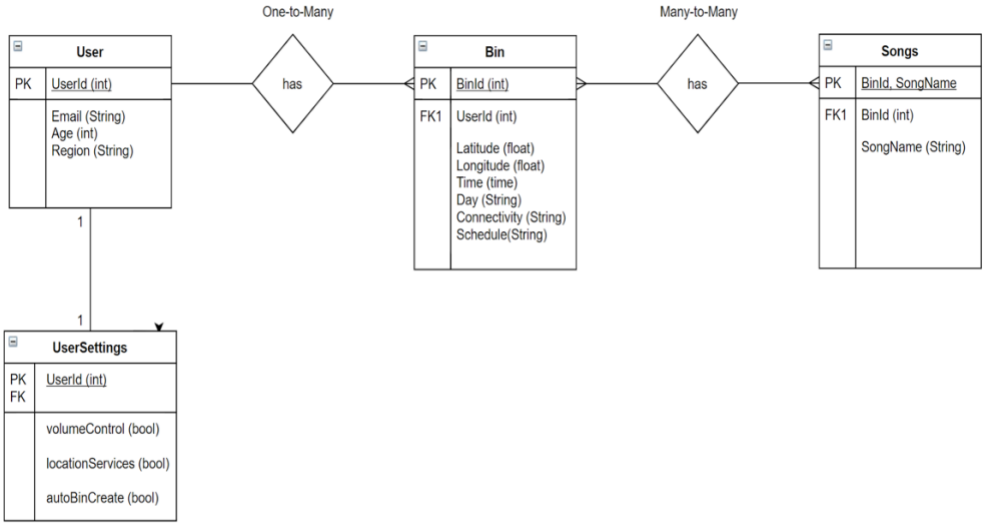


Figure 4: Database Design

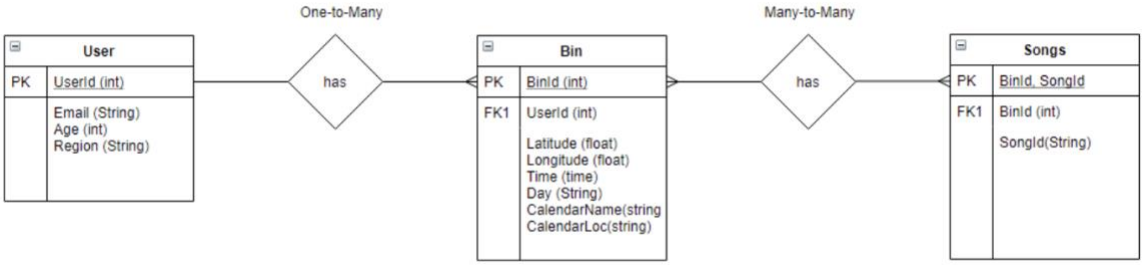


Figure 5: Revised Database Design

2.1.3 Onboarding Diagram

When a user first logs in to his/her account, we will take them through an initial process to get their recommendation system setup. This diagram shows our onboarding process for each user. First, they will log in to Spotify and simply choose a song from their own library or any song on Spotify they'd like to listen to at that moment. As one can see, a Google sign-in was planned for the application. However, the user does not need to sign

in to Google anymore. From that particular song, their feature vector will be derived from their current sensor inputs. The combination of these will create their first bin with this song in it and a feature vector to describe the bin. We will then give similar song recommendations from their library or Spotify based on the song metadata. The user will give feedback on these recommendations which will help us decide whether to keep adding these songs to the same bin or create new bins if the user wants to differentiate any particular song.

Also, please note that the revised Onboarding Diagram is Figure 7.

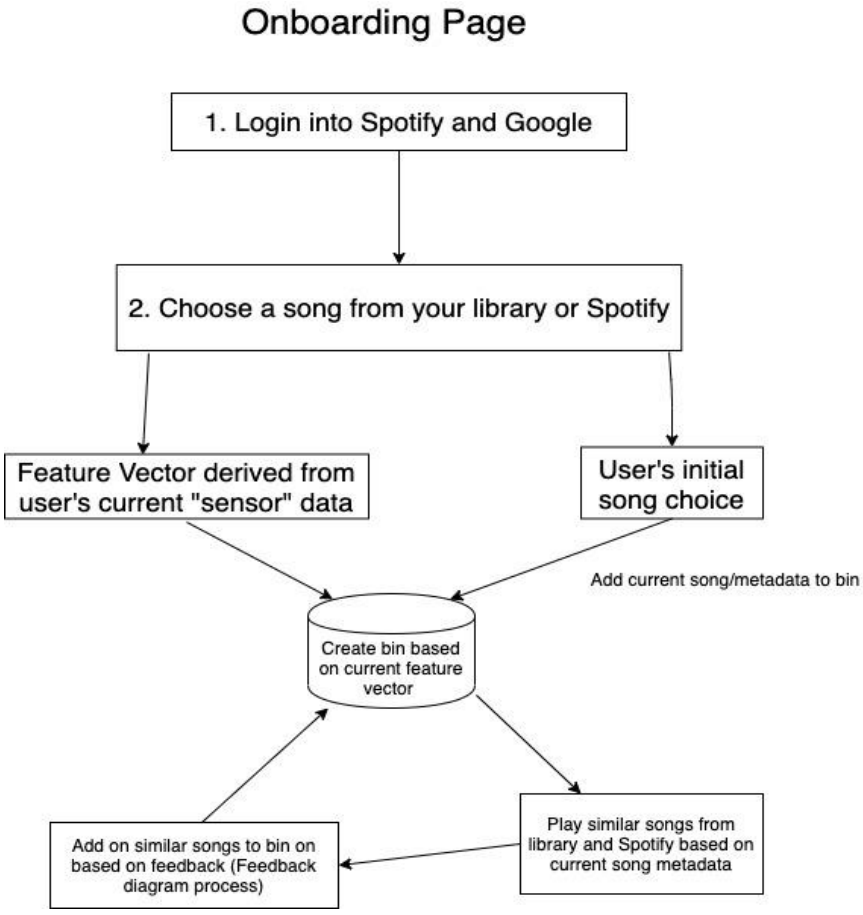


Figure 6: Onboarding Diagram

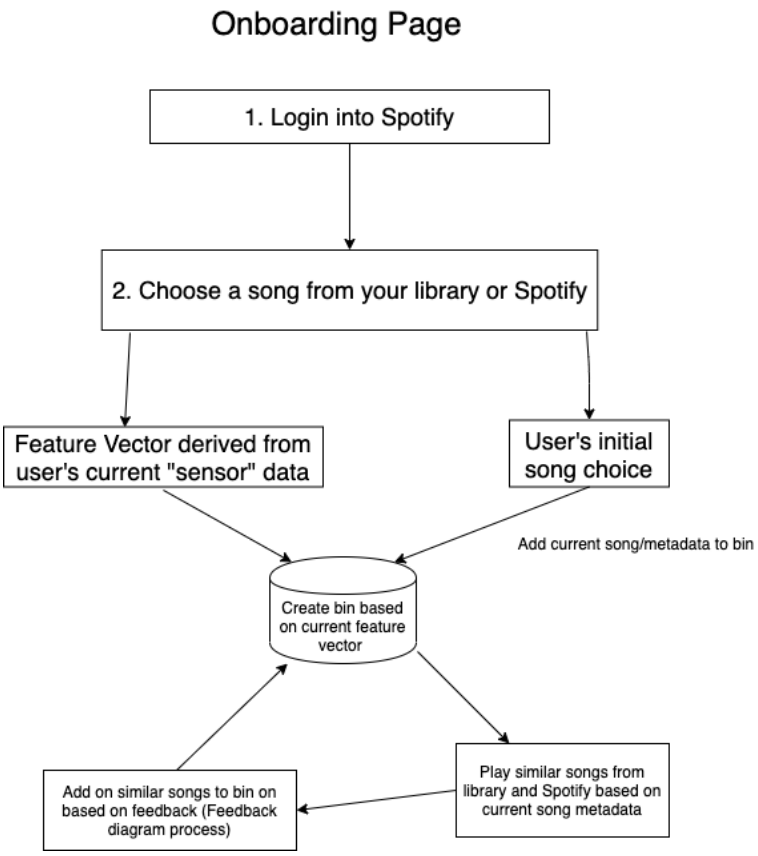


Figure 7: Revised Onboarding Diagram

2.1.4 Feature Vector Workflow

Once our user is onboarded, in order to choose songs accurately the next time they use the application, we will use this feature vector workflow method. First, we will receive all of the sensor inputs in their respective format and send them to our backend lambda function. This lambda function will uniquely check whether this feature in the vector corresponds to any specific existing feature in another bin. Depending on the correlation, we will give a priority weight to each feature that was checked and these weights will be passed to another lambda which will decide which feature is most relevant based on the weighting. The highest correlating feature will decide which bin we choose songs from at that moment.

Also, please note that Figure 9 displays the new Feature Vector Workflow. This includes a point system where the calendar location and calendar name will return 0 points if it is a match. Otherwise, if there is not a match, it will return 20 points. For day, if both days are weekdays, then the bin selection algorithm will return 0 points. Otherwise, if one of the days is a weekend day, then the algorithm will return 10 points. For time, we use the following formula: ***score = (difference in hours)*2.*** For location (distance in meters) we use the following formula: ***score = distance/10, up to 16.*** The bin with the lowest score is chosen.

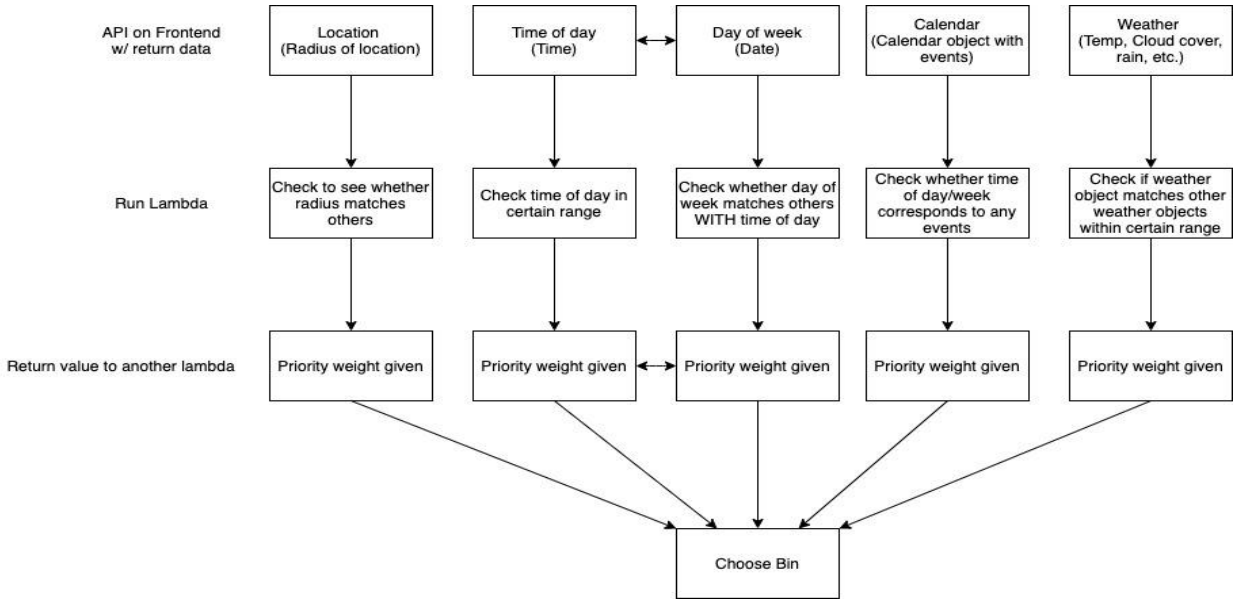


Figure 8: Feature Vector Workflow

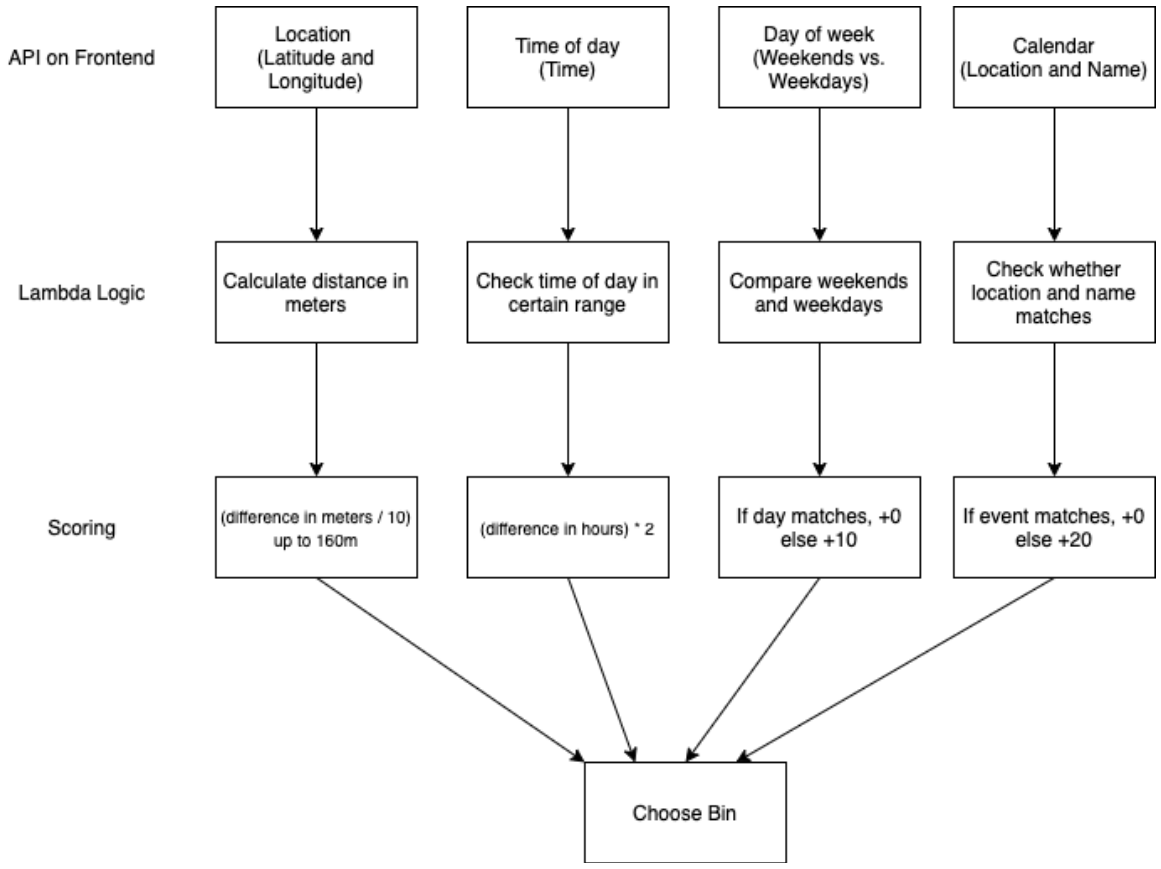


Figure 9: Revised Feature Vector Workflow

2.1.5 Feedback Diagram

Another important part of our recommendation system is the feedback process. The diagram below highlights our feedback process flow. First, our feedback page will ask the user if they like the current song first. If they like the song then it will keep playing and be added to the bin if it doesn't already exist there. If they respond no, we will change the song to another one in that bin and increase a counter. If the user responds no to 3 songs in a row, we know that type of music doesn't belong in that bin and they need a change. We will then run our backend function to find the next best bin and play a song from that bin. The entire process continues cyclically.

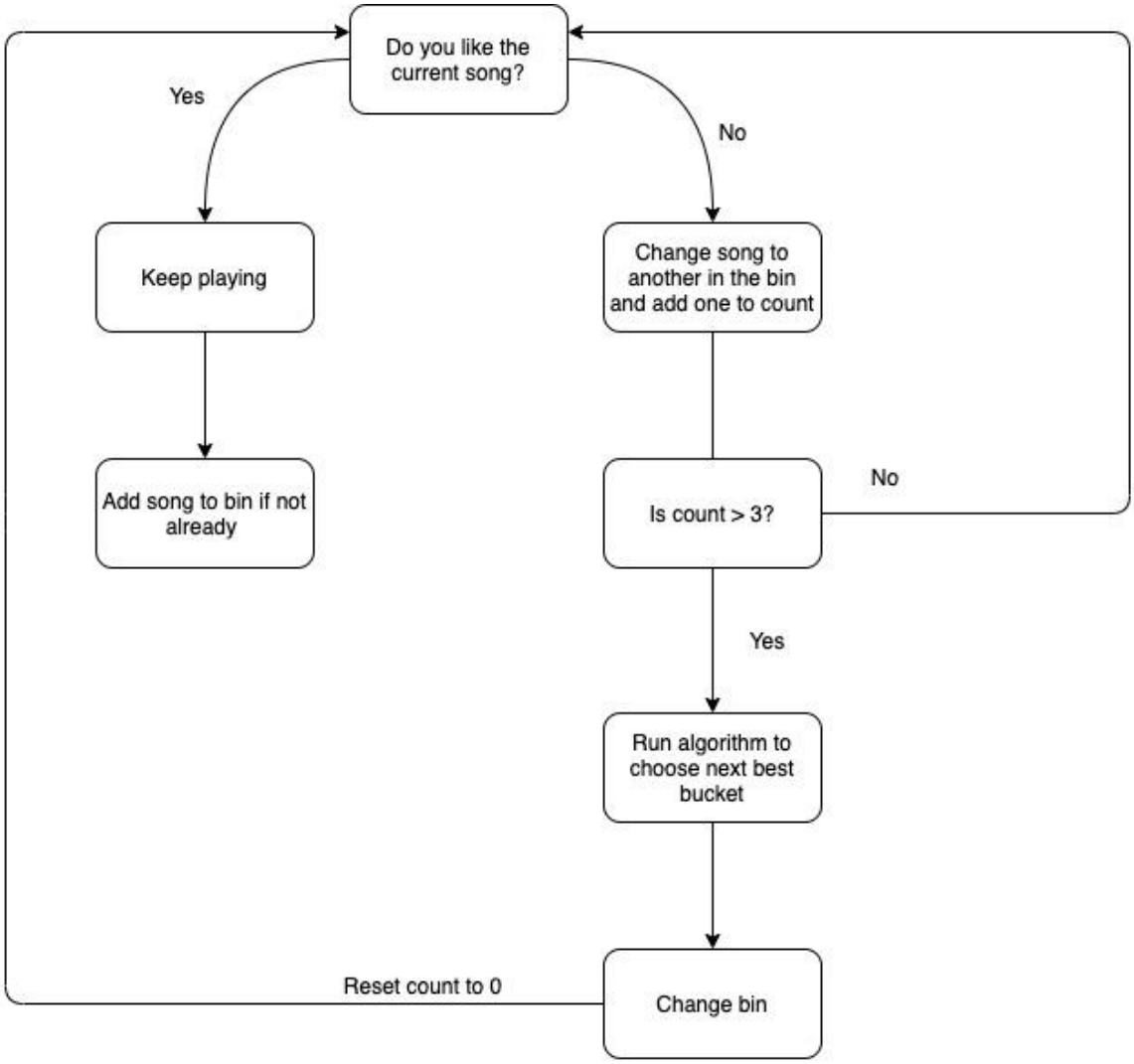


Figure 10: Feedback Diagram

2.2 Design Analysis

There are multiple modules that our application will need to function. We start with the User. The User will be responsible for providing feedback on our music predictions as well as

inputting their music preferences. The application will then take in the user feedback to send to our processing components as well as storing local API logins. The application will then send the user's sensor data to our AWS Cloud functions to predict the next song that should be played for the user. Once that prediction is complete it will then call Spotify API to request a song and load that into the queue to be played. We will store the user's liked and disliked songs in our AWS database.

As far as the analysis of this design goes, a strength of this app includes limited inputs from the user. This is the goal of the app. Additionally, this is what our client (Dr. Duwe) would like to see. The design will allow the team to create an app that requires little input from the user. As long as the user has the app running, the app will not require too much action. The team does not believe there is a weakness in this design. With Dr. Duwe's help, the team feels as if it constructed a robust design.

2.3 Development Process

During the SE 491 course, the team followed a Waterfall method. The team completed the Planning & Requirements stage and the System & Software development design stage. This helped the team gain a high-level perspective on the project. During the Fall 2020 semester, the team adopted an Agile approach towards the project. This approach allows the team to implement continuous client feedback into our application. Throughout the semester, the team continuously developed, tested, and received feedback from Dr. Duwe. For example, the team would spend a week developing. Then, the team would use unit testing, UI testing, Postman, real data, etc. to test the changes before merging to the master branch. After developing and testing, each week, the team would bring the status of the part the team was working on to Dr. Duwe.

2.4 Conceptual Sketch

Our System-level diagram contains multiple modules that will deliver the necessary requirements. *My (Musical) Life* will consist of 5 modules. Local Storage, AWS Database, AWS Cloud Processing, Third-party API, and User Feedback. The application is designed to take minimal user input and provide a nice automated experience. To do so there will be a minimal user interface. The user will input password and login credentials that will be stored locally on the device. This data will be used to access their third-party accounts such as their calendar. As mentioned previously, the team did not have time to use Google's API. Therefore, the team went with the iPhone's calendar. The local storage data will also be used to deliver an audio streaming service. Our largest requirement is to predict what type of sounds the user wants to listen to and deliver it. Part of this processing will be done on Amazon web services. We are storing detailed user models in the database to build a personalized profile for each user. This will serve as a key factor when we try to predict the user's sounds. To minimize the amount of on-device processing we will utilize AWS cloud functions to perform more of the heavy-duty computation in prediction.

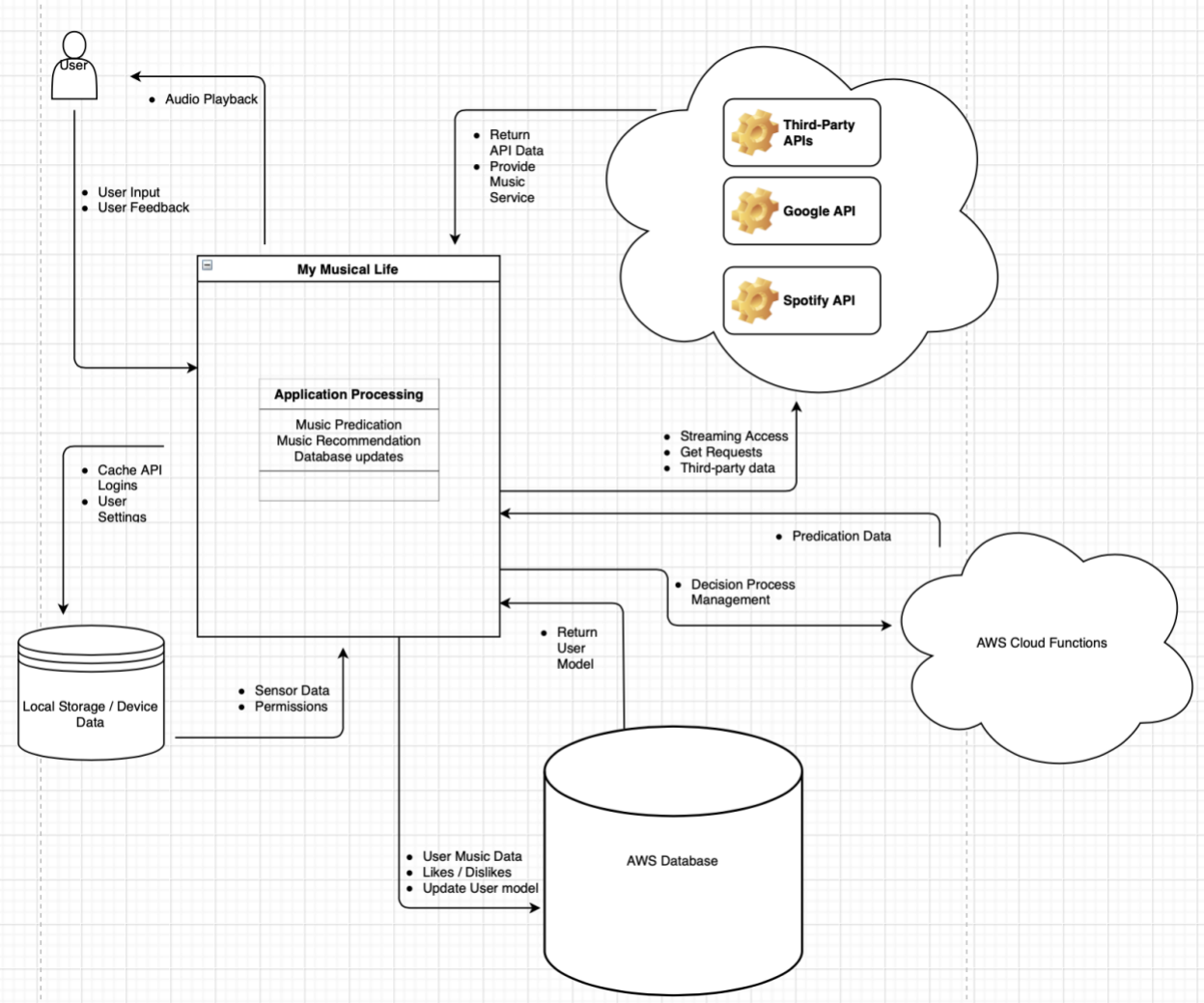


Figure 11: Conceptual Sketch

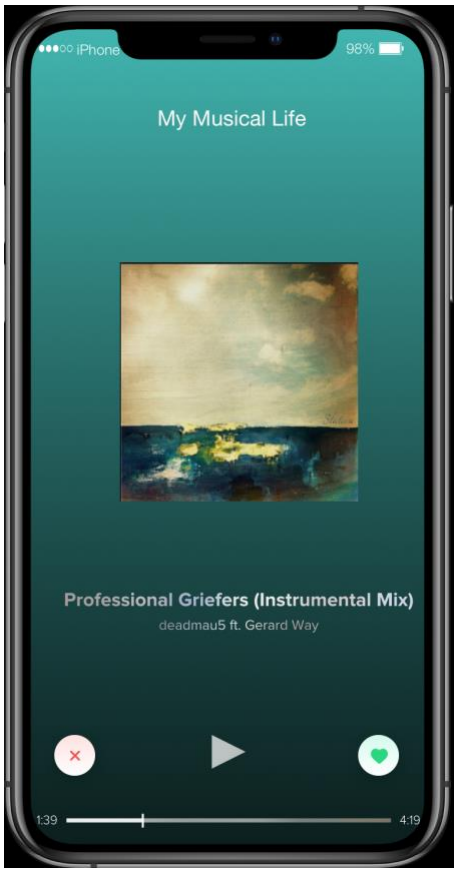


Figure 12: iOS Mockup

Our application will consist of a simple UI to encourage an autonomous feel to the experience. We will gather feedback from our users about their listening experience through the like and dislike buttons. Pictured above (Figure 12), was a rough sketch of what the team was envisioning the app to look like. The team went with a slightly different approach. The team has implemented a “Thumbs Up” and “Thumbs Down” feature for the app.

3. Statement of Work

3.1 Previous Work And Literature

As for previous work that has been done, there are apps that have been created that do something similar to the app we plan on making. However, our app will be different than the three that we will mention.

First, there is an app by the name of *MusicFit.* The goal of this app is to generate music based on the user's body movements through the iPhone's sensors [3]. Ultimately, this app will generate music based on the user's change of pace while working out. The app will only generate music based on four genres (techno, electro, idm, chill) [3]. *My (Musical) Life* will be different because our app will generate music based on a couple of factors (calendar, time of day, weather, connectivity, location, etc.). Our app will predict what type of music the user would like to listen to based on those factors. Additionally, our app will not be limited to only four genres.

Next, another similar application that exists is *Musicovery.* For this app, "recommendations of tracks, artists, genres, and playlists are personalized in real-time to each listener, according to his music preferences, listening behavior, and listening history" [2]. Again, this is different from *My (Musical) Life* since our app will generate music based on different factors. Furthermore, *Musicovery* seemed to have been an app at some point, but now it is only a website. We could not find it in the app store.

Lastly, the last application that seemed to be similar to our project is *Songza*. Songza would use date, time, and past listening history to generate "playlists based on predictions about the user's mood and/or activity at the time" [1]. However, this app would allow the user to also search for playlists based on genres, mood, and artists. This is a feature that *My (Musical) Life* will not have. Adding on, the app was shut down, but it was integrated into Google Play Music. The difference between *My (Musical) Life* and *Songza* (now integrated into Google Play Music) is the fact that *My (Musical) Life* is planned to be an app with minimal to no user input. The app is planned to be extremely simple to use for the user. Furthermore, Songza had the ability to ask the user what he/she is doing [4]. *My (Musical) Life* will be different because, with the help of Google APIs, *My (Musical) Life* will have access to the user's calendar. The app will know what the user is doing at a specific time of the day. Overall, *Songza* seems to have the most similar idea to *My (Musical) Life*, but with the plan of making our app play music based on location, connectivity, weather, movement, number of steps, etc., *My (Musical) Life* will be different than Songza since Songza does not use more factors other than the date, time, and past listening history.

3.2 Technology Considerations

- **Spotify API**
    - **Strengths**: Spotify has tutorials and many webpages on how to integrate their API into an app.
    - **Weaknesses:** Each member of the team has not had much experience using Spotify's API. Therefore, there will be a learning curve.

- **AWS**
    - **Strengths:** AWS seems to be simple to use and integrated into an app. There are many tutorials that can be followed to use AWS.
    - **Weaknesses:** Some members of the team do not have experience with using AWS
- **Google API**
    - **Strengths:** There are many tutorials one can follow with Google API. Additionally, this API seems to be relatively simple to integrate into an app.
    - **Weaknesses:** Experience with using Google API.
    - ****Note: Google API is something the team did not end up using for the application.**
- **iOS**
    - **Strengths:** There are many nice tools to use while developing iOS applications. Apple provides some tutorials for iOS development as well.
    - **Weaknesses:** Apple has many restrictions on iOS apps. There are many guidelines we must follow. Also, one out of the five team members has experience with iOS development.

3.3 Task Decomposition

Tasks for the project:

- Research/Learning APIs
    - All team members need to learn APIs needed for the project.
    - This involved lots of research into different APIs and libraries the team can use.
- Swift User Interfaces
    - Only one member (out of 5) is experienced in iOS development. Thus, each team member needed to learn Swift and become more comfortable with the language.
- Spotify API
    - Vatsal worked on using and setting up Spotify for the app.
    - Vatsal had to research how to set up Spotify and use it correctly for the app.
- AlamoFire
    - This was research by the team. However, this is something the team did not end up using for the app.
- AWS
    - Throughout the Spring 2020 semester, all team members researched libraries and APIs needed for the project.
    - This was used a lot for connecting to the backend and setting up the backend.
    - Vatsal, Vignesh, and Daksh primarily worked on using AWS and setting up the backend.

        - All three handled Lambda functions for the frontend and backend.
    - Chaz and Christian helped a little with this as well.
- Setting up a database to store data
    - This task was taken on by Daksh. Daksh was able to successfully set up the backend and the database.
- Setting up frontend
    - Vignesh, Vatsal, Chaz, and Christian all helped with the frontend.
- Sign Up and Login Functionality
    - The app needs functionality for a user to sign up and log in to the app. Chaz will be focusing on this portion of the app.
- Obtaining User Data
    - The application needs to obtain user data, such as calendar and location. This will be done by Chaz.
- UI Development
    - This involves making the application appealing to the user. Also, this involves making the application user-friendly. This will be worked on by all team members.
- Creation of Application Icon
    - This will be completed by Vignesh. The app icon is what a user will see on his/her iPhone home page.
- Google APIs
    - Google APIs were not used for this project due to time constraints.
- Testing
    - Christian took on researching and setting up the testing part of this project.
    - This involved learning how to develop UI and unit testing in Swift.
    - Additionally, this involved researching and using iOSSnapshotTestCase.
    - Testing was also done using Postman. This was used by Vignesh and Daksh.
- Bin Selection Algorithm
    - Daksh was the leader of this task. Daksh took on the ranking of the bins and implementing the algorithm.
    - All the team members helped a little with the creation of the bin selection algorithm.

3.4 Possible Risks And Risk Management

As we develop the app, there will be multiple issues that we will run into. The first issue is the user not having a Spotify Premium account. This may make it difficult to access all types of music for the user. Although we may not be able to provide the same experience for non-

premium users, we will be able to work with the Spotify API in order to still access songs and playlists to recommend to our users.

Next, trying to figure out an algorithm to select which bin needs to be selected or if a bin needs to be created will be a challenge. This algorithm can be somewhat subjective which adds to the challenge. However, even if we are not able to come up with a perfect solution, the rest of the application will still have its intended functionality as we will still be able to recommend songs for the users.

Developing an iOS app will create some roadblocks for the team. Apple has many restrictions and guidelines one must follow. Creating an app in iOS rather than Android is a bit more difficult. Only one out of the five team members is familiar with iOS development. Thus, there will be a learning curve for the four other team members.

Obtaining user data, such as accessing the user's calendar, location, etc. may be a difficult task for the team. There may be some restrictions on the amount of data we can receive from each user.

3.5 Project Proposed Milestones and Evaluation Criteria

The first milestone that needs to be completed is, by the end of May 2020, the team plans on having parts of the app completed. That includes the setup of AWS, the setup of the front-end and back-end, setup of the database, and our app should be able to connect to Spotify. Additionally, the team should have a repo setup as well. There are not many evaluation criteria needed to complete this milestone. As long as some basic parts of the app are created, that will be sufficient. As for testing, the team should ensure that the separate parts of the app are somewhat functional.

The second milestone is the frontend and backend connecting. This was set to be completed by the beginning of the fall semester. The goal was to have both ends fully communicating. For example, a user should be able to sign in and have its details stored in the backend. Testing should include running the app on a simulator or an actual device and checking the backend to see if the inputs on the frontend are successfully making it to the backend.

The third milestone is AWS API Gateway. This involves being able to develop APIs that will be able to effectively communicate with the backend and frontend. This will also help with the creation of the lambda functions and setting up lambda function triggers. Testing will consist of using the simulator for testing and Postman.

The fourth milestone is integrating Spotify. This milestone will include connecting Spotify with the app. The app should be able to use Spotify to play music. Basic functionality, such as stop, play, change song, etc., should be evident. Thus, testing will include testing the team's app on an actual device.

The fifth milestone is the bin selection algorithm. This algorithm should correctly identify the bin needed to play for the user based on sensor data. This is a complex algorithm that will take time. The algorithm needs to correctly identify bins, rank bins, and weigh the sensor data

correctly. The best way to test this algorithm is with real data and testing the algorithm while the app is running. Another great way to test the app is to load the app on the iPhone, and actually have the app running. This will determine if the bin selection algorithm is correctly choosing songs.

Overall, the most important milestone for the team is to have a fully functional app by November 2020. The evaluation criteria for this milestone is an app that is meeting all of its functional and non-functional requirements. Additionally, this app needs to be able to satisfy the client. That is Dr. Duwe. Testing involves unit and UI unit testing. Additionally, testing with real data and on an actual device is the best way to go to ensure that the user will have a great experience.

3.6 Project Tracking Procedures

The group will use GitLab to track progress throughout the course of this and next semester. GitLab has a section called “Issues” where one can add issues to an “Issues Board.” Thus, this is the project management tool the team will use throughout the duration of the project.

Additionally, the team will be tracking progress by meeting some basic code requirements. That is passing the unit tests and a fully functional app before pushing the changes. The team will also discuss code changes before pushing to the master branch. Throughout the semester, the team would have mini code reviews.

3.7 Expected Results and Validation

The desired outcome is to have a fully functional app by November 2020. *My (Musical) Life* app will be in the app store free for users to use. The application will predict the user’s mood based on many factors and generate music based on that mood. Additionally, the application will be bug-free.

The plan was to confirm the application will work at a high level by running an extensive list of tests (System Tests, Unit Tests, Stress Tests, Regression Tests, etc). The team implemented unit tests to test the app. Another way of testing the app was by running the app on an actual iPhone. Additionally, the team decided to use real data to test the app. Additionally, the app must satisfy our client (Dr. Duwe) and other potential users.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

This section will include the initial Gantt Chart the team created during the Spring 2020 semester. The first Gantt chart's description was modified to represent what the team did accomplish and did not accomplish on the chart. The second Gantt chart reveals the actual length of time that it took to complete tasks.

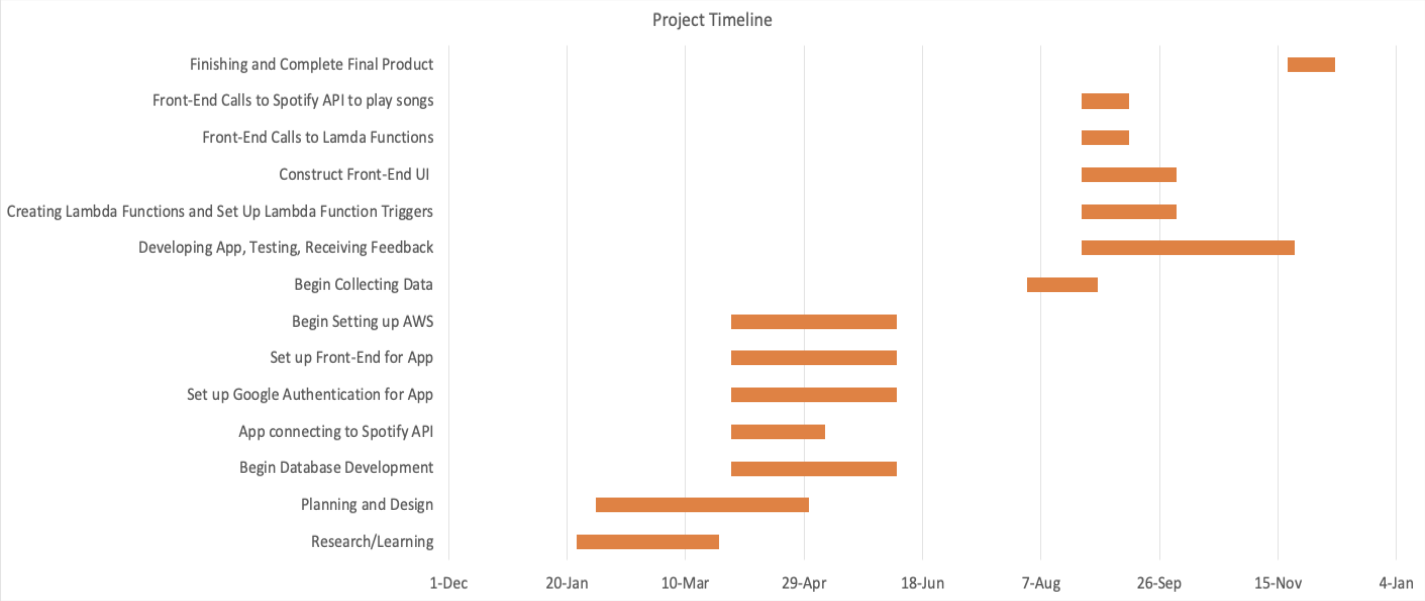


Figure 13: Gantt Chart of Project Timeline (Spring 2020)

The Gantt chart above shows some of the short term and long term goals that we had for this project. We moved out of the phase that primarily consists of design and development towards the end of the Spring 2020 semester. However, there were some core pieces we wanted to cover towards the end of May. First, we wanted to finish setting up AWS for our app. Next, setting up a basic frontend was on track to be completed by the end of May. The goal was for the app to not consist of many pages, but the initial set up of the frontend was projected to take a while. These tasks were projected to be finished by end of May. However, due to the team's quick transition to internships, the team didn't start tackling these tasks until the Fall 2020 semester. Going forward, the plan was to set up Google authentication for the app. Thus, this would involve having the users logging into his/her Google account via the app. This was another task that was on track to be completed by the end of May. Unfortunately, this is something that the team decided to leave as a future task. The team implemented a simple sign-up and login without Google's API. Additionally, the app needed to connect to Spotify and use Spotify. Thus, connecting to Spotify's API is something else the team has accomplished. As mentioned, users will need a Spotify account. Setting up the database is another task that the team worked on. Setting up the database can be difficult and take a while to accomplish. Thus, this was projected to take 70 days or more to complete. The backend team was able to begin and finish setting up the database in the early part of the Fall 2020 semester. As one can see, this does not reflect what the Gantt chart displays above. Furthermore, research and learning were projected to take longer

as we discovered new technologies and software to potentially use during the development process.

We would also like to mention that, by the fall semester, the team planned on collecting data for the development of the app. The team planned to use the data for development. However, some of the data were collected towards the end of the Fall 2020 semester to test the bin selection algorithm. The team did go through a cycle of developing, testing, and receiving feedback from our client/adviser throughout the semester. To be more specific, the team began working on front-end calls to Spotify's API in order to play songs. Also, the team began making the front-end calls to lambda functions. Both of these tasks did not take about 20 days to complete, as originally mentioned. This took more than 20 days. Adding on, the fall semester did involve constructing and finishing the front-end UI of the app as well. This did not take 40 days to complete, as previously mentioned. Constructing the frontend UI took the entire semester to complete. Therefore, around 75 days. Two other tasks the team planned on completing were creating lambda functions and setting up lambda function triggers. This did take about 40 days to complete. By November, the hope was to make the finishing touches.

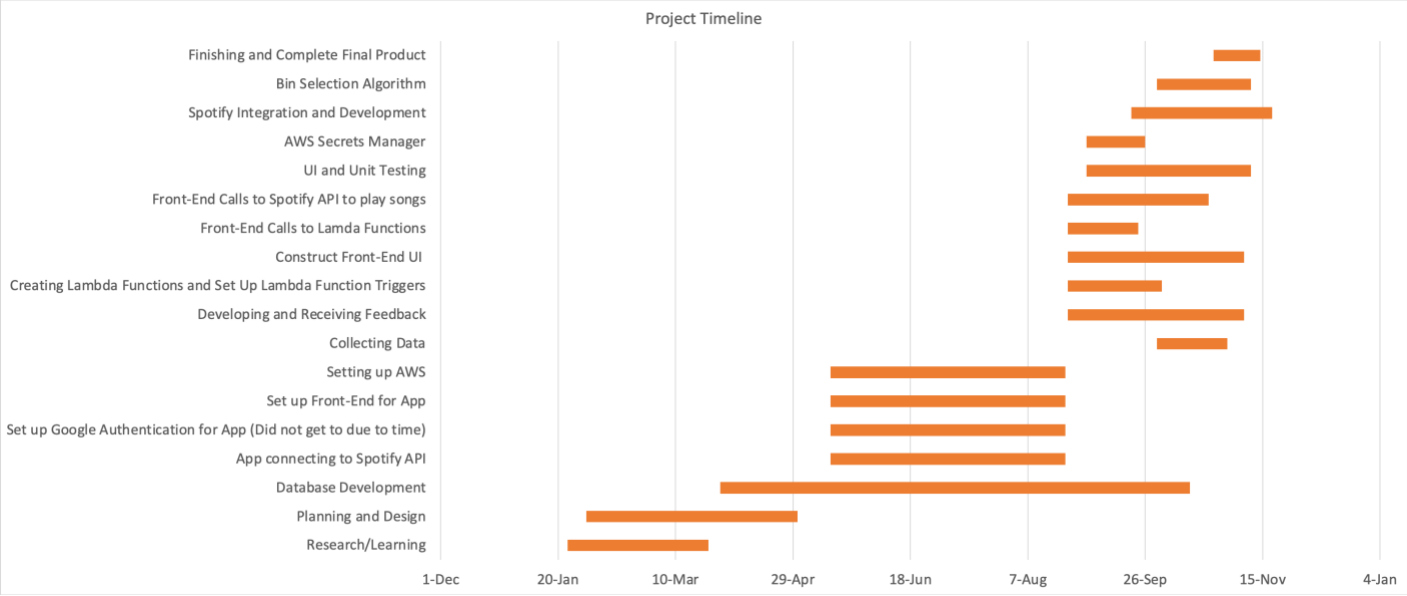


Figure 14: Revised Gantt Chart

As one can see, the Gantt chart needed to be modified a bit. This is due to the Coronavirus Pandemic. The time given to complete this project has been shortened tremendously. The team was able to have a final product towards the end of the semester.

What this second Gantt chart displays is a more accurate overview of how the Spring 2020 and the Fall 2020 semester went. The team did reach its goal in the "Finishing and Complete Final Product" task. This was meant to ultimately put finishing touches to the app. Next, the "Bin Selection Algorithm," maybe the most complex portion of the app, took most of October to complete. Given its complexity, the bin selection algorithm took a while to implement the design. "Spotify Integration and Development" was an addition added to the Gantt chart. This is

another major component of the app that took a while to complete. Some challenges occurred along the way, but the team was successfully able to get the app and Spotify's API to communicate. Next, "AWS Secrets Manager" was another addition added to the Gantt chart. This was one of the first tasks that were taken on by the backend team early in the semester. This only took about 25 days to complete. "UI and Unit Testing" is another new addition. Testing began at the beginning of the semester and took the entire semester. As new changes came in, new tests needed to be developed. Next, "Front End Calls to Spotify API" to play songs is an area of the Gantt chart that needed to be modified. Originally, this was projected to take approximately less than 2 weeks to complete. However, this ended up taking about 60 days to complete. "Front-End Calls to Lamda Functions" did not change from the previous Gantt chart. As mentioned above, "Construct Front-End UI" needed to be modified to reflect the actual time it took. This took about 75 days to complete. "Creating Lamda Functions and Set Up Lamda Function Triggers" did not change. This did take the amount of time that was projected. "Developing and Receiving Feedback" task had a name change. This did take the entire semester, as projected. However, we removed the word "testing" out of the task name and made "Testing" into "UI and Unit Testing." Next, "Collecting Data" section was modified, as mentioned above. This section didn't begin until October. Collecting data was used to test the bin selection algorithm, but the development of the algorithm began at the beginning of October. The basic setup of AWS, the basic setup of the front-end, getting a connection to Spotify's API, database development, and research and learning tasks all did go on as projected. The only task listed that the team did not get to during the Fall semester is setting up Google authentication.

4.2 Feasibility Assessment

Overall, the expectation of this project is to be an iOS app that our client (Dr. Duwe) will be able to download and use. By the end of the Fall semester (November 2020), the goal is for the app to be fully functional with regards to requirements. The application should also meet the expectations of our client, Dr. Duwe.

As for challenges the team ran into, the team ran into challenges using new technologies we do not have experience with. Thus, we had to allocate time for learning new technologies. Another challenge is time. We are given less than a year to create this app; therefore, given our class schedules and other obligations, this has been a challenge. Additionally, the Coronavirus Pandemic has impacted the project as well. The pandemic has resulted in a shorter time frame to complete the project.

Lastly, for 4 out of the 5 members of the team have never developed an iOS application before. 2 members have not had much experience with AWS as well. Given that the majority of the team are new to iOS development, developing this project within this time frame has been challenging.

4.3 Personnel Effort Requirements

Task	Description	Projected Effort and Time
Research/Learning of APIs, Swift User Interfaces, Spotify API, AlamoFire, AWS, Machine Learning, etc.	This task simply includes research and learning any new technologies needed for the project.	This takes quite a bit of effort. Most likely a month is a good timeframe for this task. This took about 60 days to complete.
Create a database to store data locally and in the cloud	A database to store information locally and in the cloud. Currently, data is stored in both places	Setting up databases can be difficult and confusing. Ultimately, a month will be enough time to set up a fully functional database.
App(Frontend) connects to APIs (Google API, Spotify API, Third-Party APIs)	After the research/learning phase, we would like to begin to connect to these APIs. That is, we need to integrate some of these APIs into our app and learn how to use them.	This should about two weeks to accomplish. With the right training and tutorials, this can be done in a short time frame. The team did not implement Google's API.
Use Lambda functions in AWS to update DB	Write push, pull function to update and read our database	This should take about 2-3 weeks depending on how many tables we use.
Connect backend to Spotify API	The backend needs to query /recommendations endpoint in order to get the next song(s)	This should take about one week. However, building the logic for recommendation may take about a month
Implement Google login	The frontend will use Google to authenticate users. This was a task that the team did not get to.	This should take about 2 weeks. However, it should be noted that this is now considered to be a future task
AWS Secrets Manager	Used for security for the app.	Given the team has a couple of experienced users with AWS, this should not take much effort. This is projected to take 2 weeks.
AWS API Gateway REST API	Used for connection between the backend and frontend.	This will be done by Vignesh. Vignesh has experience with this task. There will still be lots of effort to get this to work properly. This is projected to take about 2-3

		weeks. To complete.
iOS Sign Up	The front-end needs functionality for a user to sign up. This will consist of an email address, password, and age.	The team does have an iOS expert (Chaz), therefore, this will help a lot. However, passwords and emails must be kept secure. So, this will take a lot of effort. For the sign-up to work correctly, this should take about a month.
iOS Login	There must be login functionality for the user so that the user can log in to his/her account. Passwords should be secure.	The team does have an iOS expert (Chaz), therefore, this will help a lot. However, passwords and emails must be kept secure. So, this will take a lot of effort. For the login to work correctly, this should take about a month.
iOS UI	The UI should look neat and creative. Also, the UI should be organized and user-friendly. The application will have multiple pages, and the flow to and from pages should work correctly.	This should not be too difficult to design. Given that 4 out of the 5 members have never developed in Swift before, this may take a little more time than expected. This is expected to take about a month and a half to complete.
User Data for iOS	User data, such as the user's calendar and location, is needed for the bin selection algorithm.	This will be a difficult task. Chaz, our lead iOS developer, will be taking charge of this. This will take about 3 weeks to complete.
Bin Selection Workflow	A bin selection workflow is needed. For example, user data will be needed to be sent to the backend to add as inputs for the bin selection algorithm.	This will be a difficult task to complete. The bin selection algorithm is a difficult task as well. So, creating this workflow may take time. We are projecting this to take about 2-3 weeks to complete.
Spotify Integration	The app should have full communication with Spotify given that this is the music streaming service the team	Given that the team does not have much experience with using Spotify for iOS development, this will take a

	went forward with using. Based on the user's sensor data, the bin selection algorithm will determine which type of genre for the user. Spotify is the streaming service that will play the music.	while and lots of effort to complete. The project timeline for Spotify to work correctly in the app is 2-3 months.
Testing (UI and Unit Tests)	The involves all testing required for the project. The team plans on writing unit tests and UI tests. The team will be using frameworks, such as iOSSnapshotTestCase, to help with testing.	This should take a couple a couple of weeks for the setup. After, the implementation and design of the tests typically take multiple months to complete. As more code goes is pushed, more tests need to be written.
Bin Selection Algorithm	Based on sensor data inputs, this algorithm will determine the best bin to play for the user.	This is a rigorous task that needs a lot of effort. Developing an algorithm that correctly identifies the best music to play for a user is difficult. Thus, this is estimated to take 3 to 4 months to design and develop.
Creation of App Icon	Ultimately, this is the "Face of the App." This is what the user will see on the iPhone's homepage.	This will take some effort given that most of the team does not have experience in this. Vignesh is taking on this task, and it should take about 5 days to accomplish.

4.4 Other Resource Requirements

The project has a few resource requirements in order to function.

Device: Since we have chosen to utilize the newest Swift frameworks available our app needs to be running on an Apple device that is running iOS 13 or higher.

Accounts: A Spotify account will be required to use the app in order to enable streaming data to the user.

Cellular Data or Wifi: A constant connection will need to be enabled in order for the program to function.

4.5 Financial Requirements

Currently, the only requirement is a Spotify Premium account. Previously, the team felt as if the free Spotify account would have been sufficient for the application. However, after much development, it was realized that the user must have a Spotify Premium account. Each team member does have a Spotify Premium account. Thus, this did not cost any team member money.

Overall, a total of $0 was needed for this project.

4.6 Coronavirus Impact

The Coronavirus Pandemic has impacted the outcome and process of the team's project. Face to face interaction has decreased dramatically. Everything has to be done remotely for the team given the danger of the virus. This has negatively impacted the team as the team can not engage in any communication that involves face to face interaction. This will not allow us to build closer relationships with each other outside of the project.

Additionally, the Coronavirus Pandemic has affected the project timeline as well. Originally, the project was projected to be completed by early December. This would be in time for the team's final IRP presentation and demo. However, the pandemic has forced Iowa State to make the decision to shorten the semester by a couple of weeks. Therefore, this causes the team to have to work at a faster rate and some features may not be completed in time.

Next, testing will be impacted. One of the features of the app is the bin selection algorithm that will determine the song that needs to be played based on the user's location. Given the Coronavirus restrictions throughout the country, the team will not be able to fully test this feature since there are places closed. For example, most gyms are closed.

5. Testing and Implementation

5.1 Interface Specifications

The interface should respond to user requests and obtain user data. The team will be using Apple's built-in UI testing to ensure UIkit components are responsive.

Additionally, the user should have the best experience possible. This includes a UI that is simple to understand and use. The UI should not provide any challenges to the user. The user should be able to easily navigate the app.

5.2 Hardware and Software

- Testflight
    - This allows us to distribute our iOS builds among our testers without submitting a build to the app store
    - Note: Given the shortened semester, the team did not have the ability to reach the point to use Testflight.
- PostMan
    - This will allow us to send and retrieve data to our custom-built APIs providing us a detailed view of our JSON packets

5.3 Functional Testing

To ensure the application is fully functional we will be performing various forms of integration testing.

XCTest & XCUITest – Apple's built-in testing software suite. This is what we will be using to test the iOS portion of the application.

- API Integration
    - Third-party APIs will be a big component in our application to meet our deliverables. Testing of these APIs will be done utilizing PostMan and Black box testing
- Music Integration
    - Volume and audio testing will take place to ensure that music is delivered consistently and on-demand
    - This has to be tested on an actual device since Spotify will not work on the simulator
- Sign up / Sign In
    - Authentication testing will ensure users will have profiles with saved personalized models.
    - Sign Up / Sign In Testing will be tested using the simulator and an iPhone.
- Bin Selection Algorithm

- To test the bin creation algorithm, we will be conducting user testing on the developers on the team. The plan is to collect data on recommendation satisfaction and record the progress of this satisfaction as time continues. However, the way the team went on to test the app was by testing on an actual device and using real data.
- Another way the team tested this algorithm is by actually using the app to determine if the bin was selected or not.

5.4 Non-Functional Testing

Once the app is developed, we will perform some performance, usability, security, and compatibility tests.

- Authentication Testing
  - Authentication testing was the plan for the team. However, due to time, the team was unable to get to this task. This is left to be a future task
- Memory Leak Testing
  - Memory leak testing was the plan for the team. The team was unable to get to writing memory leak tests. However, the team was able to view and make sure that memory was not an issue while running the app. In XCode, the user must click on the icon that says, “Show the debug navigator,” as seen in figure 15

Figure 15: The “Show Debug Navigator” Icon

  - Then, one will be able to view the CPU percentage used, memory used, reads/writes for the disk, and network information.
  - Figure 16 displays the app’s memory usage

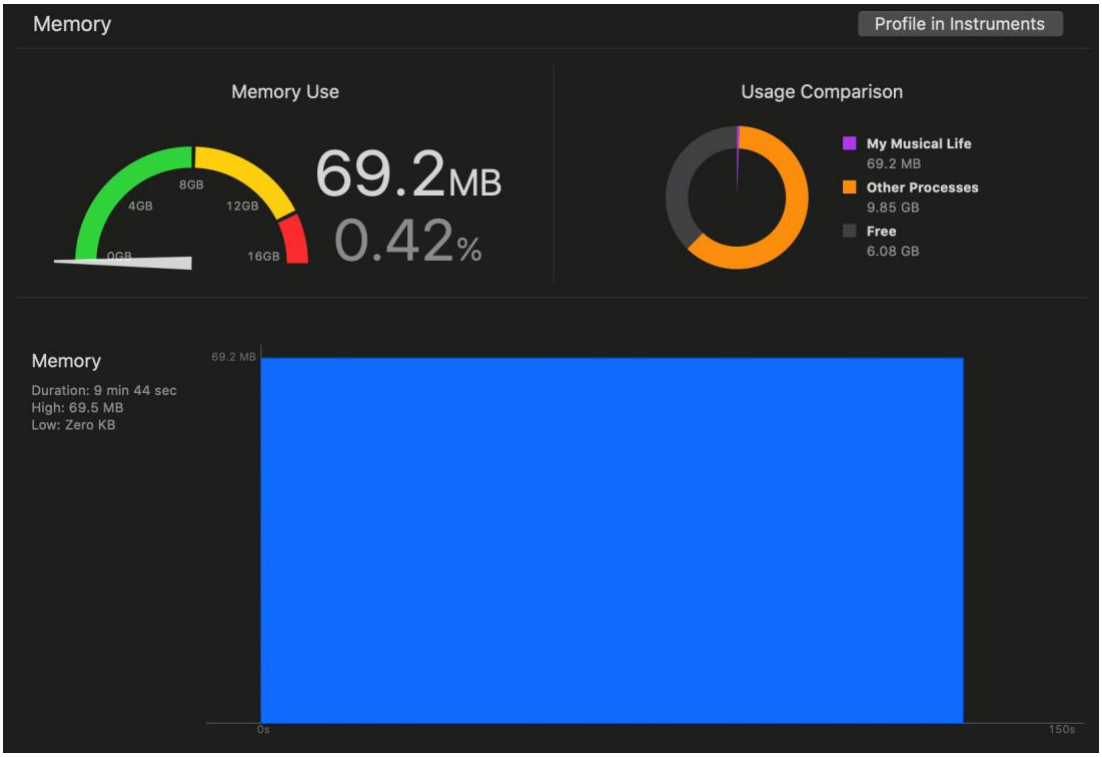


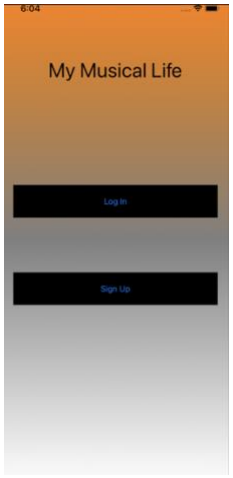
Figure 16: Memory Use of App

- Performance Testing
    - This is something that the team was unable to get to. However, the performance was measured based on testing the app consistently throughout the semester. For example, did a new change slow down the app? Is a user able to quickly switch between pages? Is there any issue with the music playing or not playing correctly?
- UI Testing
    - iOSSnapshotTestCase is used to help test the UI along with the existing XCTest framework provided by Apple.
    - UI testing was done by creating multiple test cases using the XCTest framework. One can either record the tests or write the tests. The team recommends writing the test by hand given that errors can occur while trying to record the tests. Asserts can be used to ensure that some item exists. For example the following line of code
      XCTAssertTrue(XCUIApplication().secureTextFields["pwSignUp"].exists)

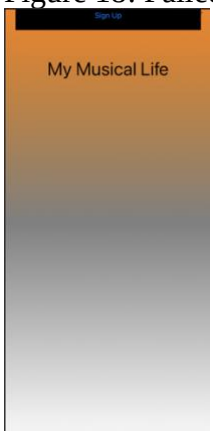
      ensures that that the password secure text field does indeed exist on the signup page.
    - iOSSnapshotTestCase is a framework provided by Uber. What this framework does is record the screen and takes a screenshot. This screenshot will be used as a reference image. Then, after running the test again with recording set to off, the test will compare the new screenshot taken of the screen with its reference image [8]. This framework will always be used to take screenshots. However, when

recording is on, the screenshot will be used as a reference image. Another piece to note is that when recording is on, the tests will always fail. That is normal and okay.

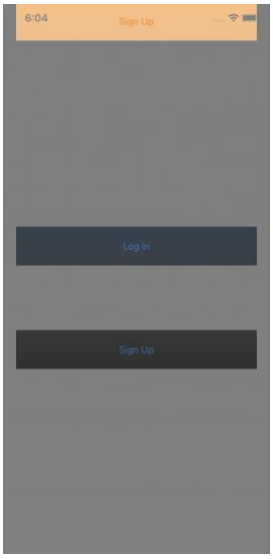
- Additionally, this framework allows the person to create the tests to store the reference images and failed images in a specific location. The tutorial given to set up this framework is very helpful.
- Here is an example of what this test framework can do:
  - Figure 17: Reference Image



  - Figure 18: Failed Image



  - Figure 19: Image Differences



- The reference image (Figure 17) is what the screen should look like. The failed image (Figure 18) is what the image actually looks like. Then, the last picture (Figure 19) displays where the differences occurred.

5.5 Process

Each team member will be responsible to write their own test cases towards code they push to the project. This will help lower the exposure to bugs and complications that the app will have. We will be using various tools in Xcode and Amazon Web Services.

Overall, for testing, the team will have a validation lead. This individual will oversee most of the testing for all features of the app. Given the short timeframe for the project, the validation lead will be responsible for developing as many test cases as possible.

Currently, the process includes the following:

1. Developers write and perform basic testing before pushing to master
2. Christian develops unit and UI tests to test new changes that went into master
    a. Tests changes locally
3. Pushes to master

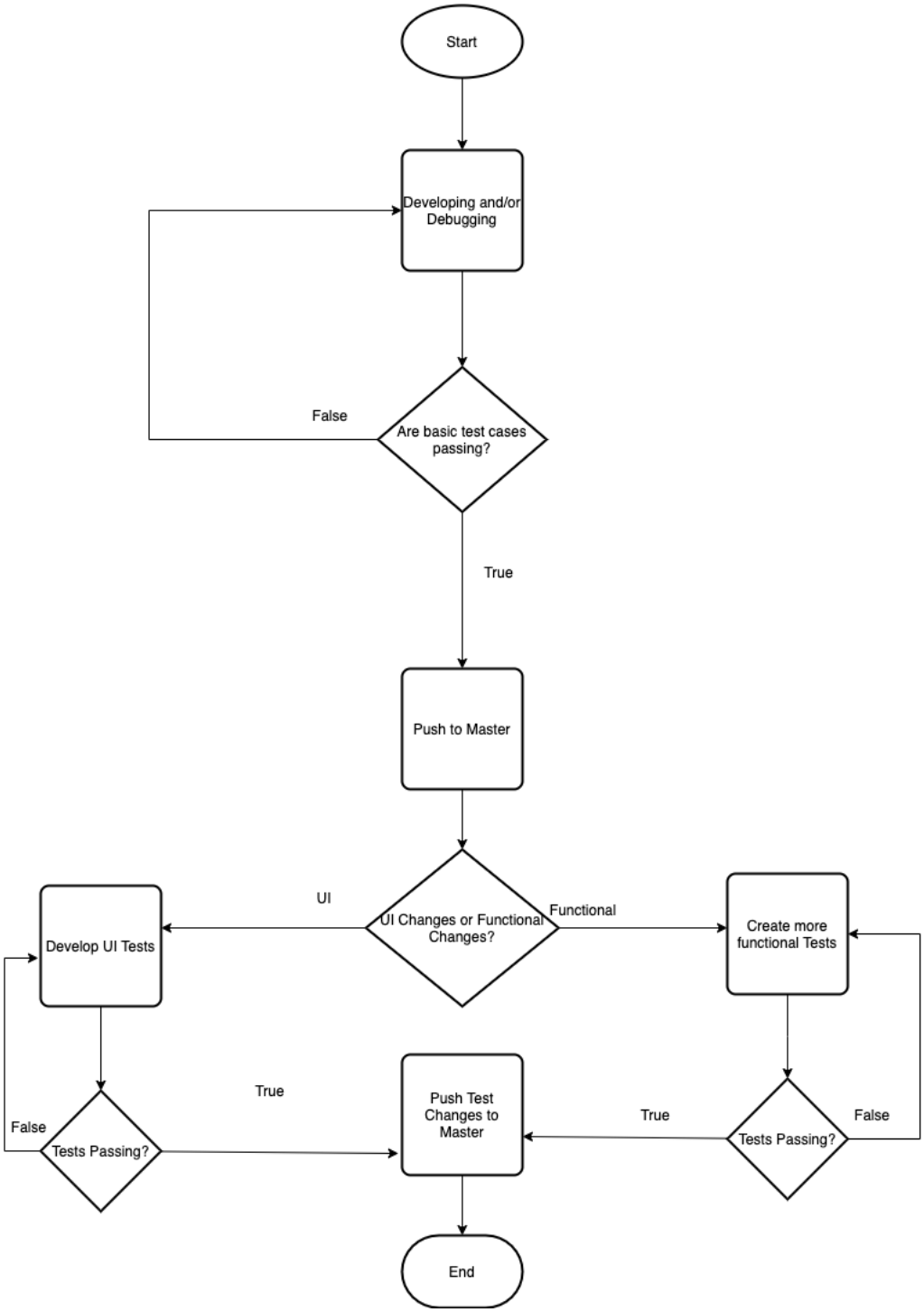


Figure 20: Testing Flowchart

5.6 Results

The results demonstrate that the application is reliable and secure. It delivers the functional deliverables (besides weather and volume control) as specified. Testing ensures that our application will be a great experience.

Below (Figure 21), one can tell that the app has successfully passed all of the unit tests and the UI tests. The UI tests made more sense here given that one can test what the users see. Additionally, the UI tests ensure that the flow of the screens is correct. For example, when a user clicks the login button, the user should be taken to the login screen.

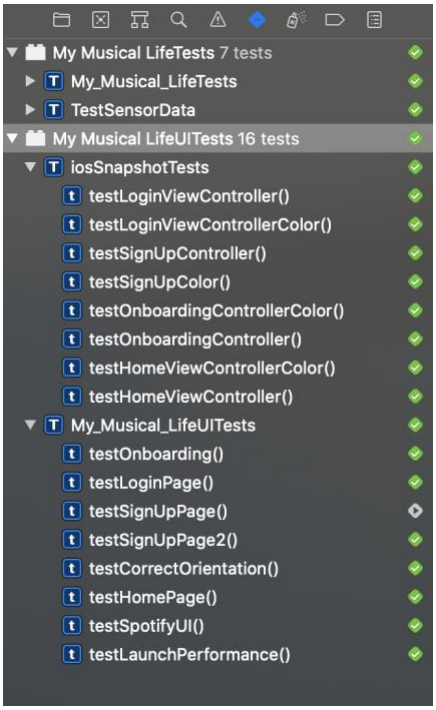


Figure 21: Unit tests and UI tests results

Another set of results the team would like to show are the results of the bin selection algorithm.

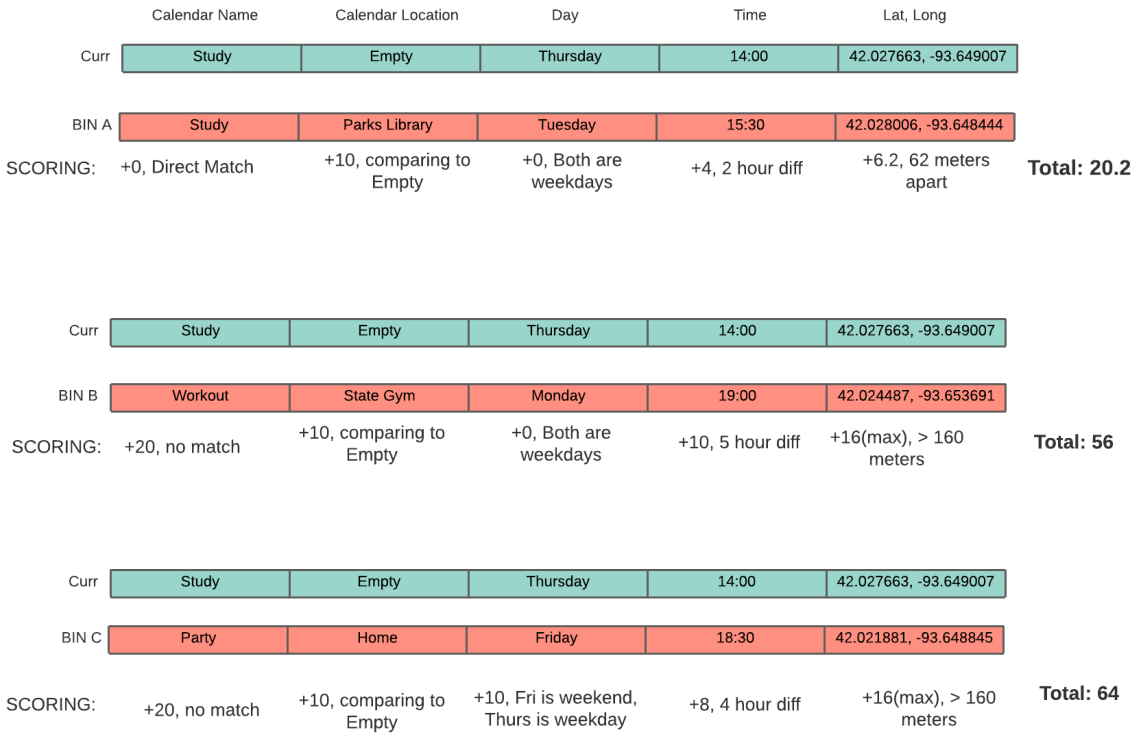


Figure 22: Bin Selection Algorithm Results

Current Feature Vector: *Calendar*: Study, *Calendar Location*: Empty, *Day*: Thursday, *Time*: 14:00, and *Lat, Lon*: 42.027663, -93.649007

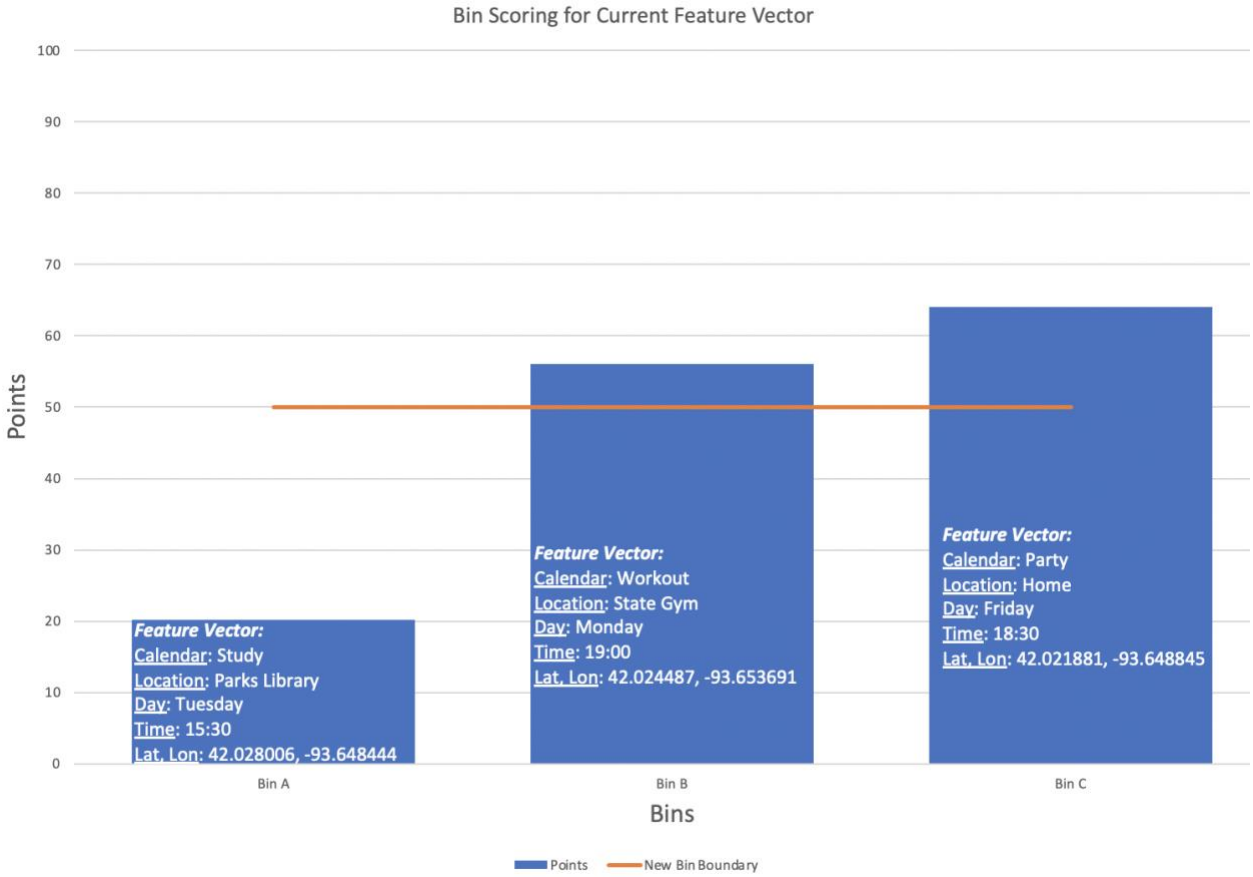


Figure 23: Bin Scoring for Current Feature Vector Bar Graph

Above, Figure 22 displays that the bin selection algorithm is working as expected. A lower score means that the bin is closer to the "Curr" bin. For example, take the example with Bin A and Curr. The result is a score of 20.2. This was accomplished by a ranking system implemented by Daksh for the bin selection algorithm. Both Curr and Bin A have an exact match for the Calendar event and Day. One might ask, "Why are days considered to be the same?" Well, this is because the system is based on weekdays and weekends. If both days are days in a week, then it is considered a match. So, the result is 0 points for those categories. Calendar Location, time, and location are a little different. However, the differences are small. As one can see, Bin B and Bin C have high scores. Any time that a score is above 50, then a new bin is created.

Figure 23 is a bar graph representation of the results from the bin selection algorithm results. This graph does not include the current bin. The purpose of this graph is to display a visual representation of how the algorithm works. First, it should be noted that the orange line is the "New Bin Boundary." Additionally, please note that the current bin has a feature vector with the following data points: ***Calendar: Study, Calendar Location: Empty, Day: Thursday, Time: 14:00, and Lat, Lon: 42.027663, -93.649007***. If the bins points are above that line, then the algorithm will generate a new bin. As one can see, Bin A is the bin that produces the closest

score to the current bin. That is because the score is the lowest and it is below the orange line. There will be a new bin created for Bin B and Bin C since the score is above the “New Bin Boundary.” This is because Bin B and Bin C data points from their feature vectors are not close enough to the data points in the current bin.

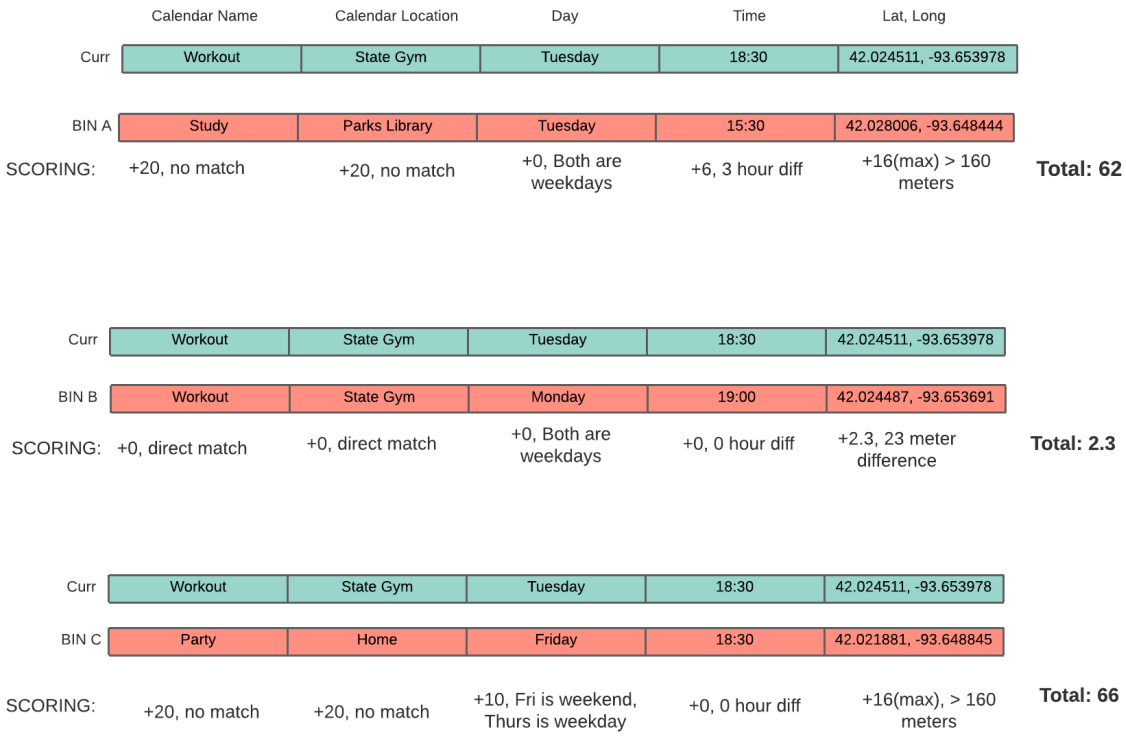


Figure 24: Bin Selection Algorithm Results 2

Current Feature Vector: *Calendar*: Workout, *Calendar Location*: State Gym, *Day*: Tuesday, *Time*: 18:30, and *Lat, Lon*: 42.024511, -93.653978

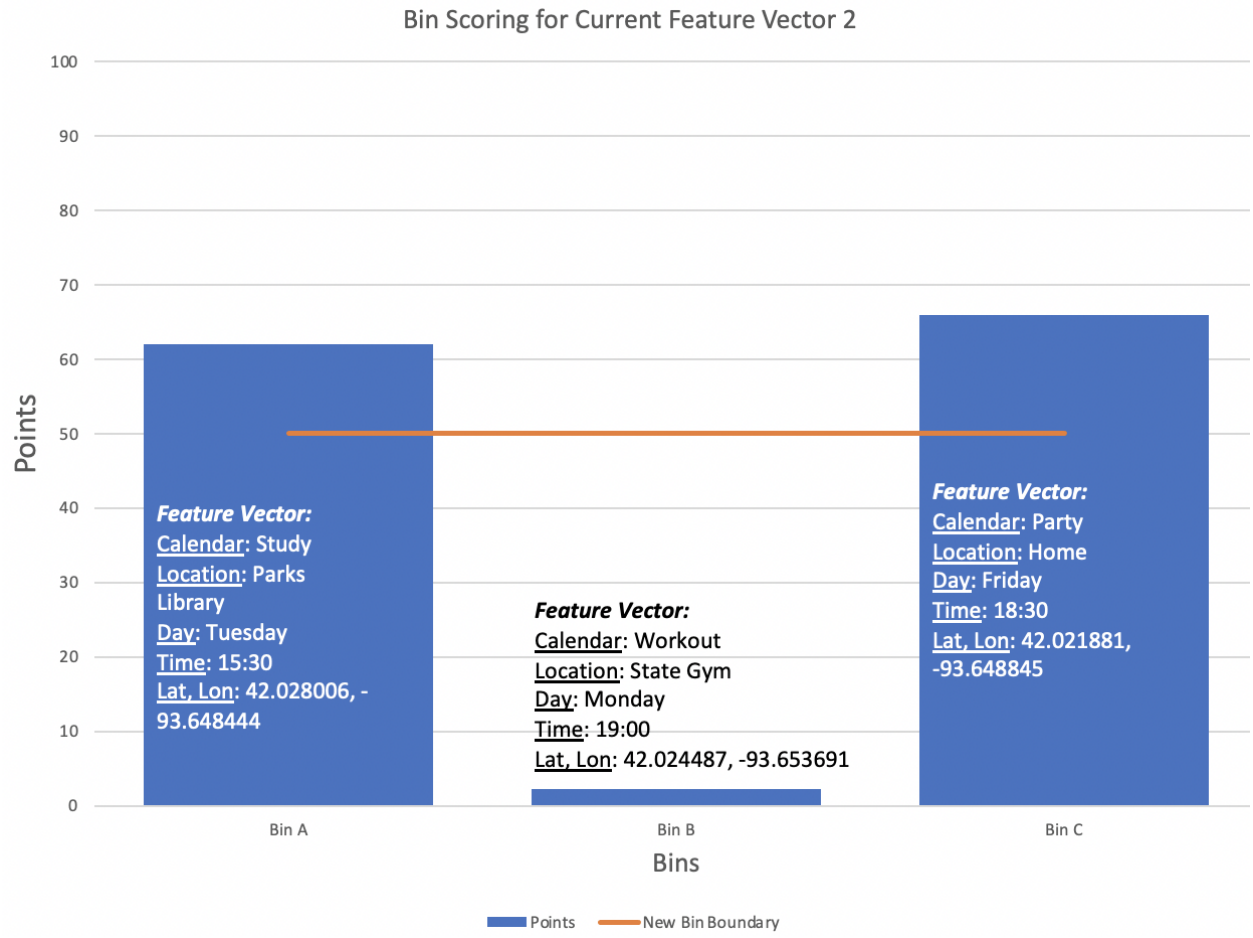


Figure 25: Bin Scoring for Current Feature Vector 2

Above, Figure 24 and 25 are another round of tests that the team ran to test the bin selection algorithm. As one can see, according to Figure 25, Bin B is now the bin that is closest to the current bin. The feature vector for the current bin is as follows: ***Calendar: Workout, Calendar Location: State Gym, Day: Tuesday, Time: 18:30, and Lat, Lon: 42.024511, -93.653978.*** This current feature vector can be viewed on top of Figure 24. Figure 24 is the diagram that is comparing the current feature vector with bin A, bin B, and bin C. Additionally, according to Figure 25, there will be new bins that need to be created for Bin A and Bin C since both bins exceed the "New Bin Boundary." Overall, these two figures display another case with the feature vector data points.

6. Closing Material

6.1 Conclusion

In conclusion, the planning phase included an abundance of research and learning. Our research consisted of tools and technologies the team can potentially use for this app. After a semester spent developing and testing, the team has produced a well-tested app that meets the majority of its initial requirements. Throughout the Fall 2020 semester, the team worked closely with its adviser/client (Dr. Duwe) to ensure that the app is meeting his requirements. Overall, the goal was to create an app that was fully-functional by November 2020. Also, the goal was to have the team's adviser/client (Dr. Duwe) to be satisfied with the final product. Therefore, the team followed the tasks and advice given by Dr. Duwe in order to stay on track throughout the semester.

6.2 References

[1] J. Crook, "Google Will Shut Down Songza App, Songza.com To Fold Into Google Play Music," *TechCrunch*, 02-Dec-2015. [Online]. Available: https://techcrunch.com/2015/12/02/google-will-shut-down-songza-app-songza-com-to-fold-into-google-play-music/. [Accessed: 24-Feb-2020].

[2] "Musicovery B2B," *Musicovery B2B*. [Online]. Available: http://b2b.musicovery.com/. [Accessed: 24-Feb-2020].

[3] "MusicFit," *App Store*, 02-Jan-2018. [Online]. Available: https://apps.apple.com/us/app/musicfit/id1186085097. [Accessed: 24-Feb-2020].

[4] E. Van Buskirk, "Songza's Concierge Picks Free Music for Specific Situations (Now for iPad)," *evolver.fm*, 05-Mar-2012. [Online]. Available: http://evolver.fm/2012/03/05/songzas-new-concierge-picks-free-music-for-your-specific-situation/. [Accessed: 24-Apr-2020].

[5]"iOS App Testing Tutorial: Manual & Automation," Guru99, 24-Mar-2020. [Online]. Available: https://www.guru99.com/getting-started-with-ios-testing.html. [Accessed: 25-Apr-2020].

[6] Guo, Bingkun, "iOS Security" WUSTL, 1-Dec-2014. [Online]. Available: https://www.cse.wustl.edu/~jain/cse571-14/ftp/ios_security/index.html. [Accessed: 25-Apr-2020].

[7] "16 Metrics to ensure mobile app success" App Dynamics, 2015. [Online] Available: https://www.appdynamics.com/media/uploaded-files/1432066155/white-paper-16-metrics-every-mobile-team-should-monitor.pdf. [Accessed 26-April-2020].

[8] "iOSSnapshotTestCase (previously FBSnapshotTestCase)," *GitHub*. [Online]. Available: https://github.com/uber/ios-snapshot-test-case. [Accessed: 09-Nov-2020].

Appendix I: Operation Manual

1. Please install Spotify if you don't have Spotify installed.
2. Download the app from the app store. Currently, the app is not in the app store. Therefore, the way that we are doing this is by installing the application from our computer.
3. If you receive a prompt asking for location permission, please select an option. For the best experience, choose "Allow Once" or "Allow While Using App."
3. If you are a new user, please sign up. Otherwise, log in to your existing account.

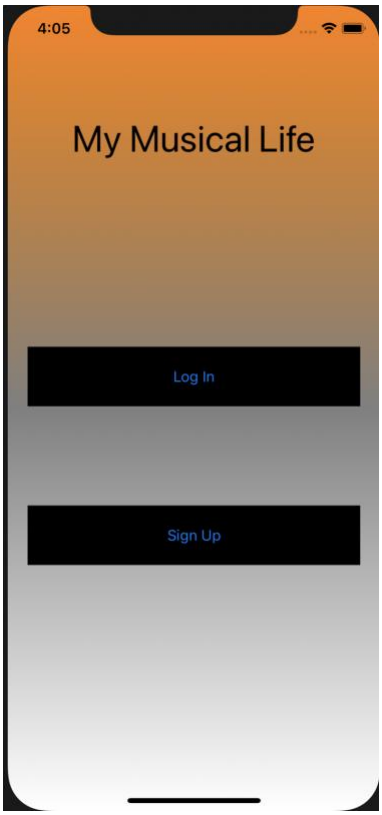


Figure 26: Sign Up or Log In Page

4. Sign Up will include entering in an email address, password, and age. Log In will only include adding an email address and a password.

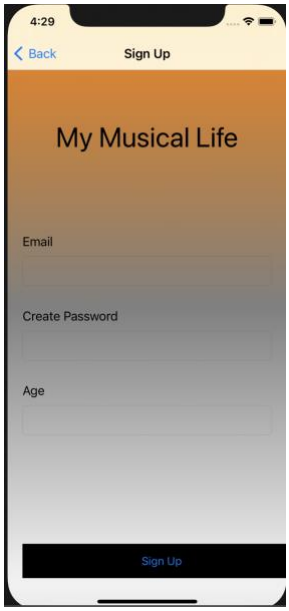
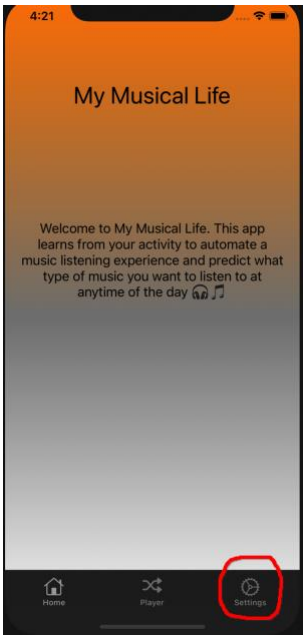


Figure 27: Log In Figure 28: Sign Up

5. After logging in or signing up, if you received a prompt asking for permission to your calendar data, please choose an option. You may always update your options by navigating to the Settings page as shown below.



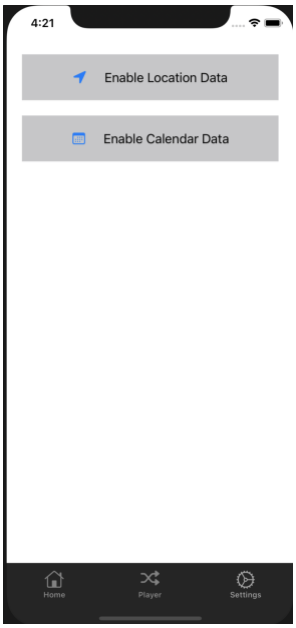


Figure 29: Home Page (Settings Circled) Figure 30: Settings Page

6. If you are ready to listen to music, please click the “Player” icon at the bottom of the screen.

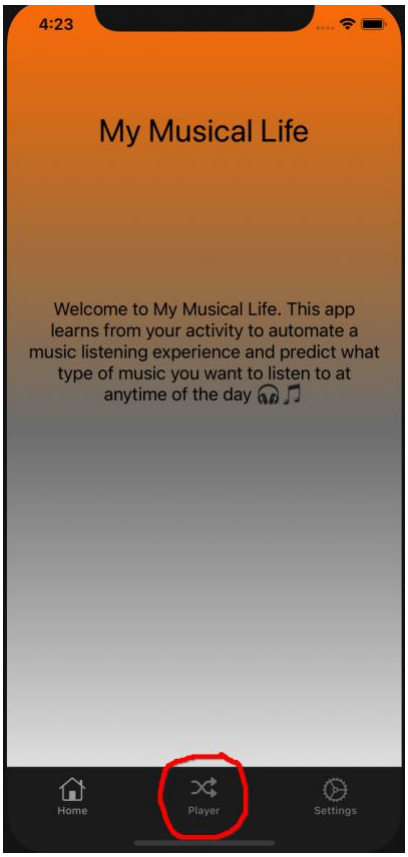


Figure 31: Home Page (Spotify Player Circled)

7. Then, connect to Spotify using your credentials!

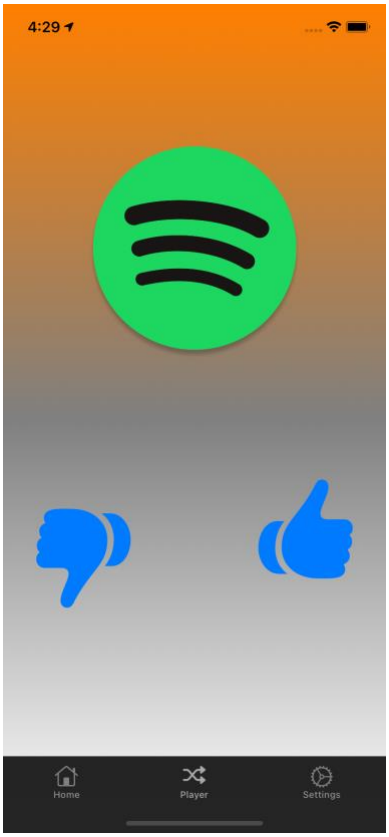


Figure 32: Spotify Player Page

8. Once connected, you can use the “Thumbs Up” or “Thumbs Down” button when songs are playing. A “Thumbs Up” means that you like the song. “Thumbs Down” means you do not like the song.