

**spotifynd  
friends**

# TABLE OF CONTENTS

## PHASE 1

MOTIVATION/PROBLEM STATEMENT  
USERS  
PLAN OF APPROACH  
MINIMUM VIABLE PRODUCT  
DESIGN DECISIONS  
RETRO 1  
DIFFICULTIES

## PHASE 2

DESIGN REVIEW  
APP ARCHITECTURE  
RETRO 2  
RETRO 3  
DIFFICULTIES

## PHASE 3

FEATURES  
DIFFICULTIES

## PHASE 4

DESIGN/FEATURE HIGHLIGHTS  
FUTURE WORK  
CHALLENGES OF THE WHOLE PROJECT

# TABLE OF CONTENTS

## EVALUATION/TESTING

MVP EVALUATION

FINAL PROJECT EVALUATION

UNIT TESTING LIBRARIES

## ACKNOWLEDGEMENT OF RESOURCES

# PHASE 1

## I. MOTIVATION/PROBLEM STATEMENT


The original motivation for the web app was to allow users to easily connect with others with similar music tastes to help them find friends to go to concerts with. Lan-Chi, who came up with the idea for the app, was inspired by an instance where she wanted to attend an Illenium concert in San Francisco, but none of her friends were interested in Illenium. She realized that an app that would allow her to find friends who listen to similar music would be helpful in finding people to attend concerts with in the future.

## II. USERS

### a. PERSONAS

We initially hoped for our app to cater towards a diverse audience. Below are two personas we used while brainstorming what our app's functionality would look like:

### Jessica



Creative

Artistic

Young

Smart

#### Goals

- Attend her favorite indie artist's concert
- Find more indie music to listen to
- Make friends who are familiar with her favorite artists

#### Frustrations

- Music tastes are extremely obscure
- New to Chicago, Illinois
- Busy schedule reduces time to meet new people

#### Bio

Jessica is a young adult who only listens to indie music. No one she knows listens to the same music that she does, so no one gives her the aux cord in the car. One of the bands that she listens to is having an impromptu concert during the weekend, but she does not drive and currently has no way of getting to the concert.

#### Personality

Introvert	Extrovert
Thinking	Feeling
Sensing	Intuition
Judging	Perceiving

#### Motivation

Social	
Music Discovery	
Taste Analysis	

#### Preferred Channels

Traditional Ads	
Online & Social Media	
Referral	
Guerrilla Efforts & PR	

## Jim Smith



Senior

Clever

Controversial

### Goals

- Find people to carpool with
- Understand what his music tastes are
- Attend his first music concert

"The good thing about being old is not being young"

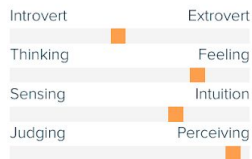
Age: 71

Work: Farmer

Family: Single

Location: Eudora, Kansas

### Personality



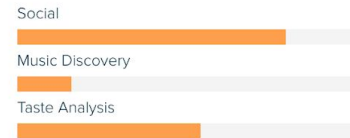
### Frustrations

- No family living near him
- Lives far from the nearest big city
- Poor health prevents him from driving

### Bio

Jim Smith is 71 and lives in the middle of nowhere. Because of his bad back, he is unable to drive long distances to go to the concerts that he wants to go to. He is searching for people to drive him to those concerts. He has no other forms of social media and only wants to find people who like the same people as him.

### Motivation



### Preferred Channels



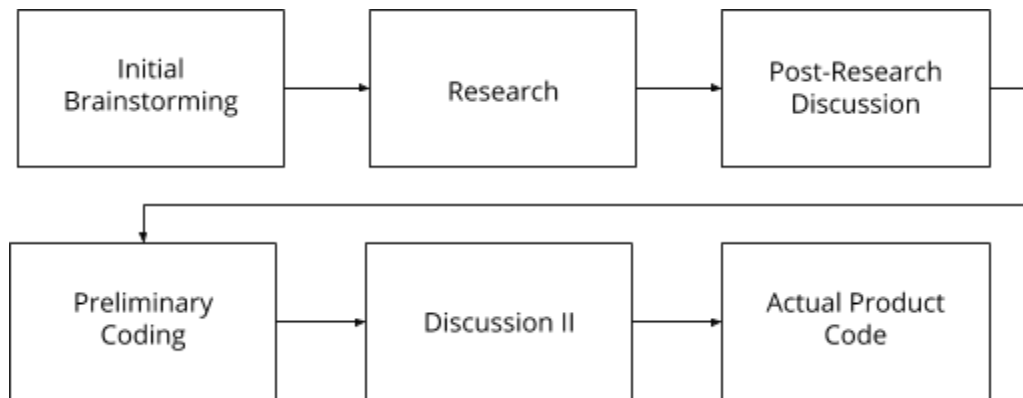
## b. USER STORIES

Based on our personas, we developed three primary user stories for us to initially focus on:

1. As a Spotify user I can put in my Spotify playlists so that I see a compatibility percentage with other people's playlists.
2. As a lonely person, I can interact with people with similar music taste so that I can find friends in my area.
3. As a concertgoer, I can find people going to the same concert as me.

## III. PLAN OF APPROACH

Due to our relative inexperience with creating apps, we decided that the most suitable plan of approach would involve ample research and testing, with regular discussion interspersed to help us maintain a shared understanding of what we should achieve next. While making sure to follow Agile development practices and CS48 class requirements, our team's progress followed the general plan of action as seen below:



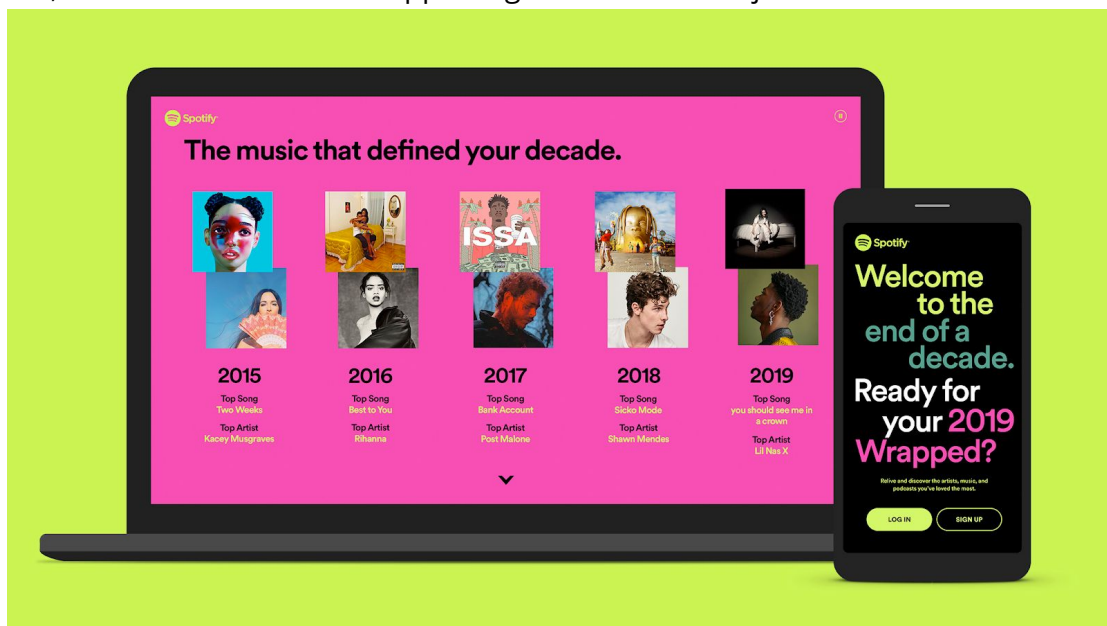
**Diagram of Our Plan of Approach**

**a. INITIAL BRAINSTORMING**

Our team began with the intention of making an iOS app as we felt this would best cater towards our intended audience (Spotify users). **We wanted our app to have two main features: comparing with other users near you and comparing with your Facebook friends.** Because none of us had much experience in creating apps, we decided that the team would benefit from a relatively extensive research phase.

**b. RESEARCH + POST-RESEARCH DISCUSSION**

Based on our research, our team realized that creating a web app rather than an iOS app would be more suitable. This was primarily due to the greater amount of support the Spotify API provided for web apps, and to the inspiration we found in the Spotify Wrapped webpage. Following this decision, we decided to use Javascript as our main language, and React and Node.js as our frameworks. We also agreed on a green and black theme as it would be most in line with Spotify's design and help maintain a sense of continuity. After this, we moved on to developing preliminary code, such as a "Hello World" app using React and Node.js.



**Spotify Wrapped**

During this phase, the team members who were assigned research tasks produced documents detailing the results of their research. These documents are linked here:

- [React Notes](#)
- [Node.js Notes](#)

### c. PRELIMINARY CODING + DISCUSSION II

While making our “Hello World” apps, we made use of a variety of frameworks and toolchains, including Create React App, Next.js, and Express. Eventually, we decided on using Next.js for our app as it seemed as though multiple members had positive experiences with it, especially with Heroku deployment. In our team’s discussion following the preliminary coding phase, we also decided on the functionalities we would like to see in our MVP.

01-27-330pm Meeting Notes

```
-Darian will lead first retro
-deciding on MVP
  -log in with spotify
  -clickable list of your playlists
  -when you click on a playlist, app will compare with dummy playlist
    -return number of songs, artists in common
  -all text based
```

### Initial MVP Features

## IV. MINIMUM VIABLE PRODUCT

### a. MVP USER STORIES

We settled on three user stories to focus on for our minimum viable product:

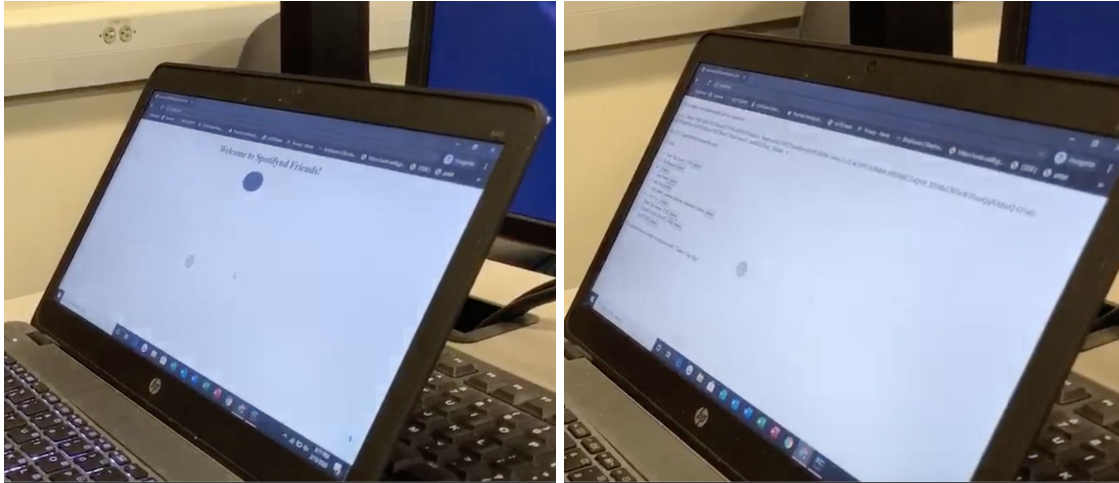
1. As a Spotify user, I can press a log in button so that I can log in with my Spotify account.
2. As a Spotify user, I can see a list of my playlists so that I can choose one to use for comparison.
3. As a Spotify user, I can choose a playlist to compare with Spotify's "Today's Top Hits" so that I can see how mainstream my music taste is.

### b. MVP FUNCTIONALITIES

1. HOME PAGE: Users see a homepage with the title “Spotifynd Friends” and can click a “Login” button that links to Spotify’s OAuth user authorization page.
2. LOG-IN: Using OAuth, Spotify users are able to use their Spotify accounts to log in to our app.
3. DISPLAY SPOTIFY PLAYLISTS: Through GET requests, our app gets a list of the logged in user’s playlists and displays them on a page.
4. COMPARE WITH DUMMY PLAYLIST: The user is able to select one of their playlists to compare with Spotify’s “Today’s Top Hits” playlist. This made use of an initial iteration of our comparison algorithm that simply returned a percentage from 0-100% and did not provide any additional information regarding the meaning of this percentage. This algorithm did not return 100% when two playlists were the same and required further refinement.

### c. MVP REVIEW

Based on our MVP review, it seemed like our app was most lacking in its UI. Our reviewer thought that we had an innovative idea, but noted that our user interface could be improved. He noted that, as our app only compared the user's playlists with "Today's Top Hits," it did not provide much functionality to the user, but did mention that it would be significantly improved once the ability to compare with other users was added.



**Home Page and User Page as of MVP Review**

## V. DESIGN DECISIONS

Many of the design decisions we made during Phase I were eventually changed—however, the main concept and idea behind our app remained consistent throughout the project.

### a. DESIGN DECISIONS EVENTUALLY USED

1. COLOR SCHEME: We decided on a black and green color scheme based off of Spotify's design.
2. LOCATION: To avoid extensive processing times and minimize strain on our app, we decided that users would only be compared with others in their area rather than all other users. We also agreed that this would make more sense from the user's perspective than a comparison with all users.
3. PROFILE PAGE: As our goal is for our app to have value in a social sense, implementing a profile page for each user was important to us.

### b. DESIGN DECISIONS EVENTUALLY CHANGED

1. FACEBOOK INTEGRATION: Users will be able to link their Facebook account and compare their playlists with their Facebook friends who also use our app.
2. CONCERTS LIST: Users can see what concerts are planned for their area in the near future.
3. CARPOOL: Users can make carpool plans for concerts through our app.



## VI. RETRO 1

In Retro 1, we noted many points of improvement—most of which were related to making better use of Agile methodologies. We organized our retro using a “Start, Stop, Continue” template.

### a. MAJOR “START” POINTS

- Timeboxing
- Pair programming
- Dedicated team coding time
- Communication

### b. MAJOR “STOP” POINTS

- Overly deliberating on decisions

### c. MAJOR “CONTINUE” POINTS

- Using Google Docs to convey notes
- Making sure everyone is on same page
- Working as a team rather than individuals

8 Start	1 Stop	3 Continue
<input checked="" type="checkbox"/> Timeboxing work on an issue so that we can commit to actually working on a delegated task. Added by spenceryzou	<input checked="" type="checkbox"/> Overly deliberating on which frameworks we want to use Added by spenceryzou	<input checked="" type="checkbox"/> using the google doc to share information Added by NickLavoie1
<input checked="" type="checkbox"/> have pairs work on certain aspects of our web app instead of everyone learning things individually Added by eduardogalvez		<input checked="" type="checkbox"/> making sure everyone is on the same page Added by yangevelyn
<input checked="" type="checkbox"/> doing more pair programming Added by lanch17		<input checked="" type="checkbox"/> I think we should continue to have more than 1 person learn how to implement functions because it gives us different ways to do one thing. Added by lanch17
<input checked="" type="checkbox"/> dedicating some of our meeting time to coding rather than just planning Added by yangevelyn		
<input checked="" type="checkbox"/> Having the same naming convention for commit messages Added by lanch17		
<input checked="" type="checkbox"/> Communicating with each other about the knowledge we have acquired so far to see where we differ Added by eduardogalvez		
<input checked="" type="checkbox"/> turning our research into concrete code Added by yangevelyn		
<input checked="" type="checkbox"/> Setting deadlines for steps towards MVP Added by NickLavoie1		

Retro 1 Artifact

## **VII. DIFFICULTIES**

Our first major hurdle was due to the fact that none of us had much familiarity with creating web apps. As a result, we dedicated a significant amount of time to researching different frameworks rather than creating actual functionality. Perhaps our biggest difficulty during Phase I was due to issues involving logging in and Spotify OAuth. We initially struggled to properly perform GET requests, and it took a good amount of time for us to get OAuth working. Additionally, an unfamiliarity with Agile made our scrums ineffective. Overall, in Phase I, we achieved much less than we hoped to.

# PHASE 2

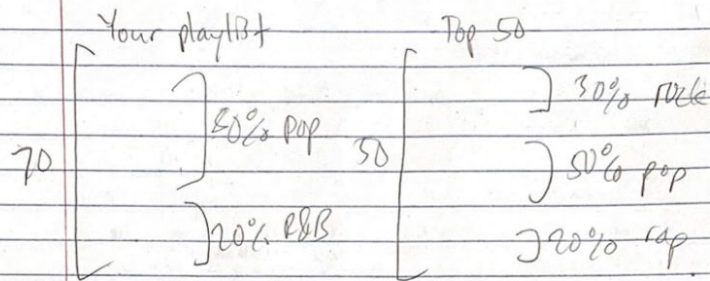
## I. DESIGN REVIEW

### a. DESIGN DECISIONS

Taking from our MVP feedback, we decided to start improving the UI by integrating bootstrap and increasing functionality for the user. This included a details button for the compatibility result that would explain how the two playlists are similar with a chart. We also implemented usage of Firebase to store details about users in preparation for comparisons between user playlists.

### b. NEW FEATURES

1. IMPROVED ALGORITHM: Although the MVP algorithm fulfilled the goal of a user being able to see a numeric compatibility result, the number provided was quite misleading. With the old algorithm averaging scores generated by a nested for-loop between two playlists, then taking the average of all the song scores, we noticed the compatibility score always ended up being similar no matter what playlist was selected. This was due to the fact that even if there are multiple same songs in both playlists, only those songs will have high compatibility, becoming an outlier that ultimately doesn't contribute much to the average. In addition, this algorithm failed a scenario that we knew we wanted to be certain: that two playlists of the same songs ought to have 100% compatibility. Keeping these in mind, we brainstormed multiple solutions before using a modified version of the temporary solution seen on the next page. This algorithm involved saving the highest score found from the nested for loop between the left and right playlist, then taking the average. This is followed by the same algorithm done starting from the right playlist compared to the left playlist. Both numbers are saved and the lower of the two is used for the compatibility score.



Matching song sweep

70%, based on 35 matching songs

Remove from both

Genre sweep

0.8 weight

20 artist

75 genre

5 attribute

145

1 R & B

5 House

90 KPop

1 Rap

140

Temp solution

Sweep genre, if same 90 pts.

Sweep artist, attributes, 5 each

Save the highest score for each song.

Compare vice versa.

Take best 5 KPop

Comparison 5 House

1 Rap

120%

$\frac{5}{100} + \frac{5}{100} + \frac{5}{100} = \frac{15}{100}$

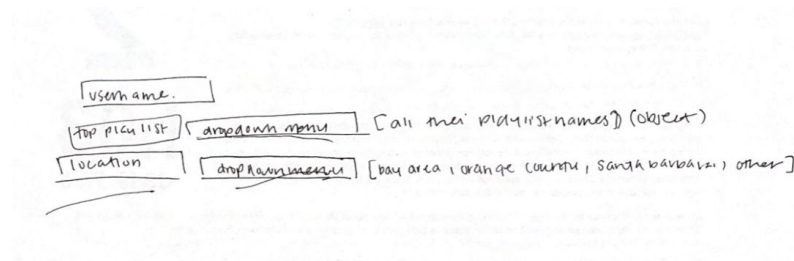
90%

145%

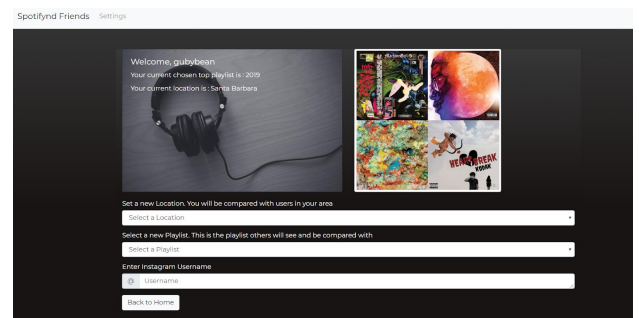
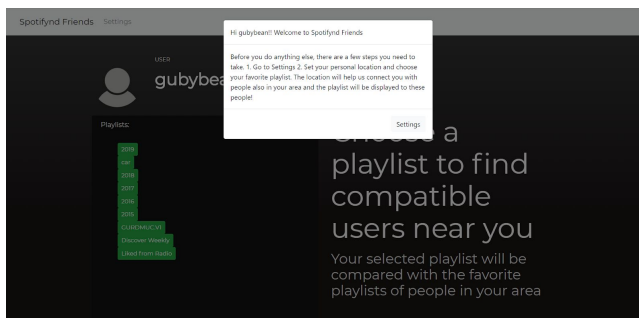
Algorithm Brainstorming

2. FIREBASE INTEGRATION/SETTINGS PAGE: Steps were made towards being able to make user to user comparisons by integrating Google Firebase to make a user information database. A settings page was created that would store the user or lookup the current user in the database, and offer forms for selecting and updating the user's location and top playlist. We wanted to ensure that all users set up their account for comparison, so a first time setup wizard would force users to update their information before allowing them to do anything else. In addition, the playlist selection only lists playlists set to public in Spotify, as we would not want to share users' private playlists.

Initial Vision:



Final Product Feature:



### Setup Wizard and Settings Page

3. COMPATIBILITY DETAILS: With just a number for compatibility, we felt that we needed to add something to explain how the two playlists are similar. To do so, we added a Details button that would generate a donut chart. This donut chart shows in what way (by attribute) selected playlists songs are similar to the other playlist. Clicking on each section of the chart lists the associated songs.



4. STATIC PROFILE PAGE: A template for the profile page was produced that would display the user and a hardcoded embedded player.
5. BOOTSTRAP UI: The UI received an overhaul on all pages except for the landing page. We added a bootstrap header, and reformatted the page layout to fit bootstrap's column and row format. We also added inline CSS styling to change the background color, matching our intention to emulate Spotify's black and green style.

## c. PROGRESS REPORTS

```
# Attendance
Project: Spotifynd Friends
Mentor: Ekta
Meeting Time: 02-26-330pm
Meeting Type: Sprint Planning
Attendance:
* Evelyn - Present
* Lan-Chi - Present
* Darian - Present
* Eduardo - Present
* Nick - Present
* Spencer - Present

# Notes
Eduardo: put playlist objects into database rather than id?
Nick: thought about algorithm fixing approaches
Evelyn: nav bar stuff/access_token carry between pages
Lan-Chi: working on displaying database values, having issues getting it to display on the page, works in console
Spencer: algorithm fixes
Darian: coming up with algorithm fixes

# Attendance
Project: Spotifynd Friends
Mentor: Ekta
Meeting Time: 02-27 6:05Pm
Meeting Type: Scrum
Attendance:
* Evelyn - Present
* Lan-Chi - Present
* Darian - Present
* Eduardo - Present
* Nick - Present
* Spencer - Present

# Notes
Evelyn: Got the nav bar to work, issue is that when you go to settings then go to home then back to settings there is an issue
Eduardo: working on the home page to pull more
Spencer: functional testing, very close to finishing but was working on his own branch, but master had an issue
Darian: helping spencer write the tests
Nick: pass but will work this weekend
Lan-Chi: worked on her issue and was able to the settings page

# Attendance
Project: Spotifynd Friends
Mentor: Ekta
Meeting Time: 03-01 2:00Pm
Meeting Type: Daily Scrum
Attendance:
* Evelyn - Present
* Lan-Chi - Present
* Darian - Present
* Eduardo - Present
* Nick - Present
* Spencer - Present

# Notes
Evelyn - worked on website ux, cleaned up design and show all playlists
Lan-Chi- work on website ux, cleaned up design and show all playlists
Spencer - pass
Nick - static profile page, shows topplaylist, pulling from database needed
Darian - pass
Eduardo - pass
```

```

# Attendance
Project: Spotifynd Friends
Mentor: Ekta
Meeting Time: lab08
Meeting Type: Daily Scrum
Attendance:
* Evelyn - Present
* Lan-Chi - Absent
* Darian - Present
* Eduardo - Present
* Nick - Present
* Spencer - Present

#Notes
Darian: Pass

Spencer: added a wheel that categorizes songs based on which attribute is most similar to the compared playlist,
hover/change color for links, homepage ui

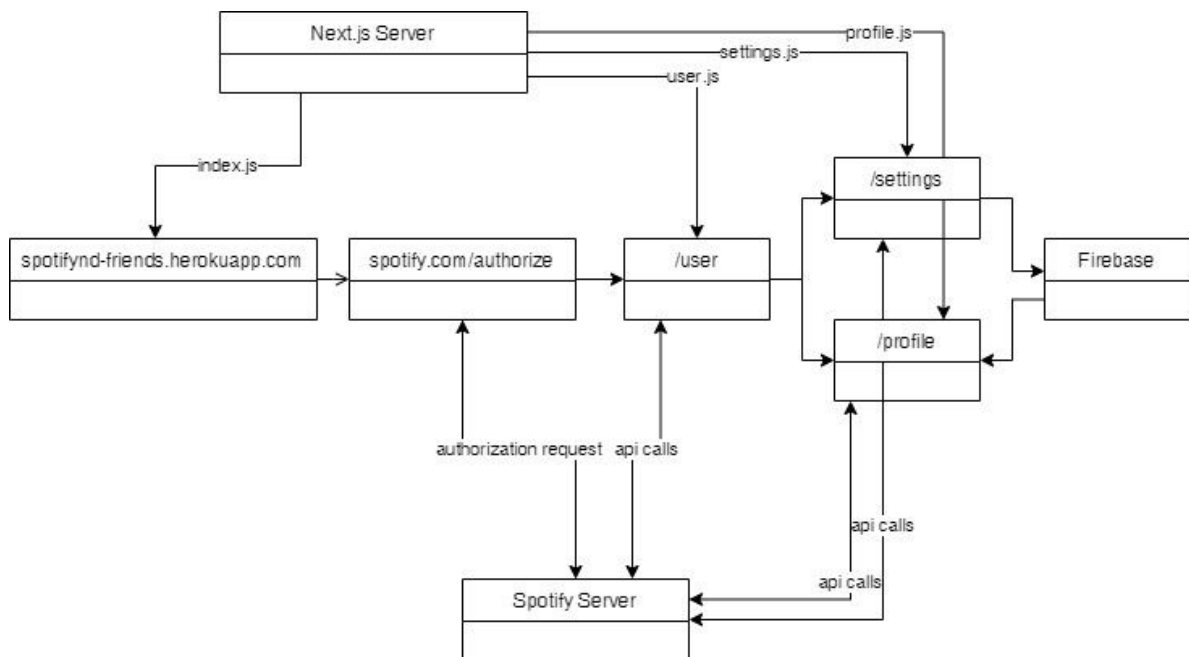
Eduardo: sends track attributes to database (not just playlist id), need to figure out how to prevent users from leaving page before done loading

Nick: user profile, when click on button next to user name, goes to user profile, top playlist embedded on profile

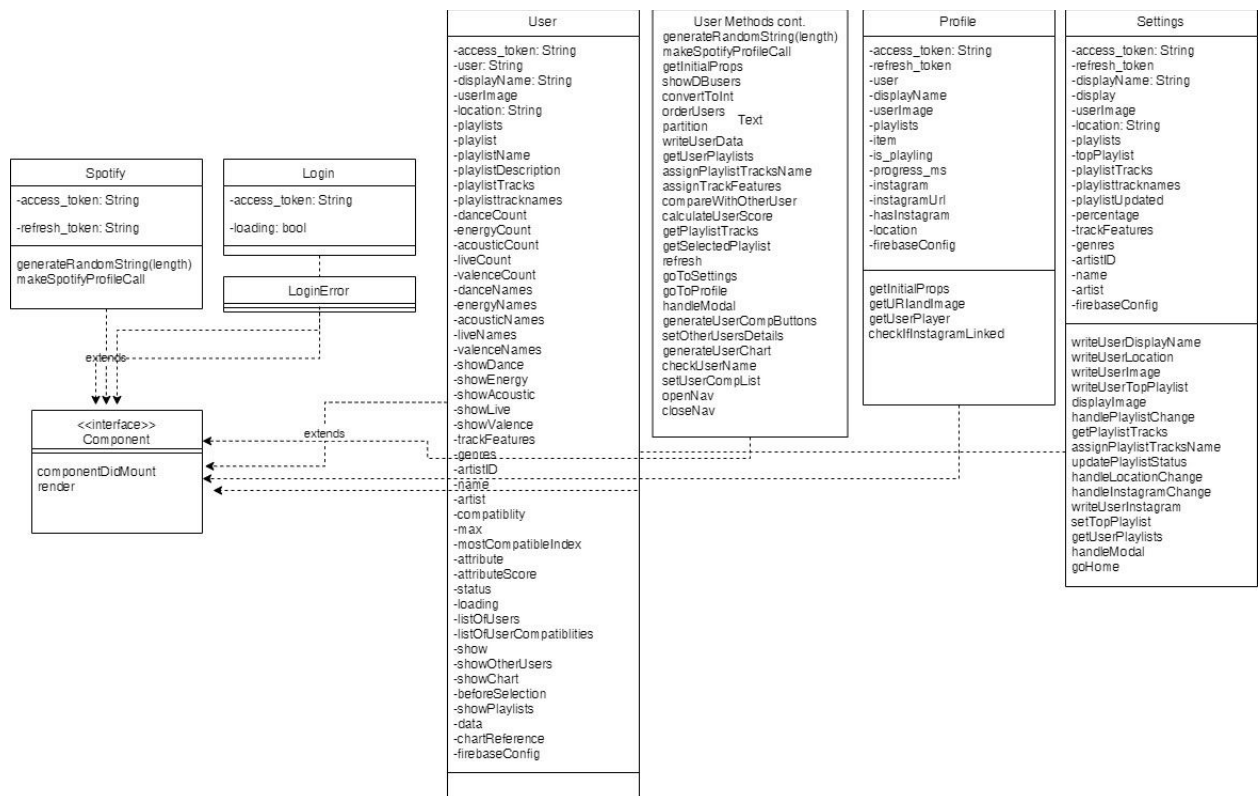
Evelyn: display songs by attribute when slice of pie chart is clicked

```

## II. APP ARCHITECTURE



On a high level, our app architecture can be represented by this UML Component Diagram. Our code is written with the Next.js framework for Node.js, which allows for faster server-side rendering of our pages, rather than client-side rendering. Next.js is structured in pages, each page represented by a Javascript file. The pages make API calls to Spotify's server, which then send JSON information back to process in the Next.js code. Before we can make API calls, however, we have to receive OAuth authorization from Spotify, which we did with the OAuth Authorization Code Flow. We then update our user's information in Firebase, which can be pulled for use in other class methods.



Our backend can be represented by this UML Class Diagram. Each page extends the interface Component, which is a React interface that has a variety of lifecycle methods and methods that tell the website what to render. The fields of each class are kept in the class' state. The Spotify class is the landing page which, on button click, goes to the Spotify authentication URL. Login is the redirect from authentication, which includes lifecycle methods to take the code from the url query and create an access token for API calls. It then pushes to User. User is the functional homepage. From User one can go to either Settings, the user Firebase update page, or Profile, the template profile page for each user.

### III. RETRO 2

In Retro 2, we noted how to structure meetings to best encourage progress, as well as actions that reduce our efficiency. We organized our retro using a "Start, Stop, Continue" template.

#### a. MAJOR "START" POINTS

- Increased designation
- Specific acceptance criteria

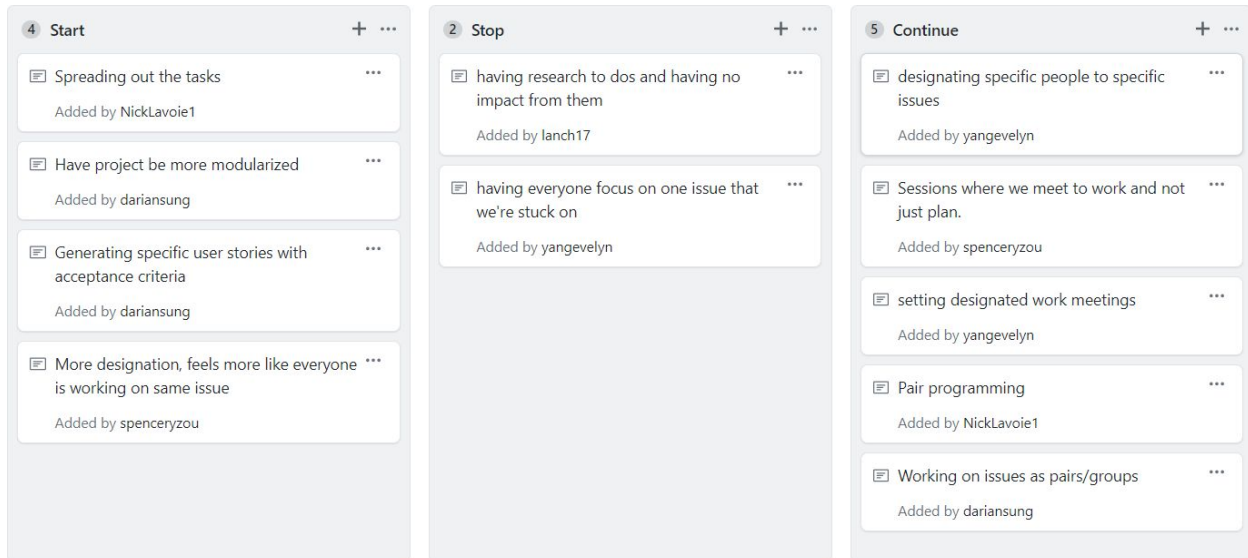
#### b. MAJOR "STOP" POINTS

- Research tasks that without tangible impact
- Everyone focusing on the same issue



### c. MAJOR "CONTINUE" POINTS

- Designated work meetings
- Working in pairs or groups
- Designating people to specific issues



### Retro 2 Artifact

## IV. RETRO 3

In Retro 3, we emphasized the work methods that worked for us in previous sprints as well as noted some specifics in terms of coding practices we could start doing.

### d. MAJOR "START" POINTS

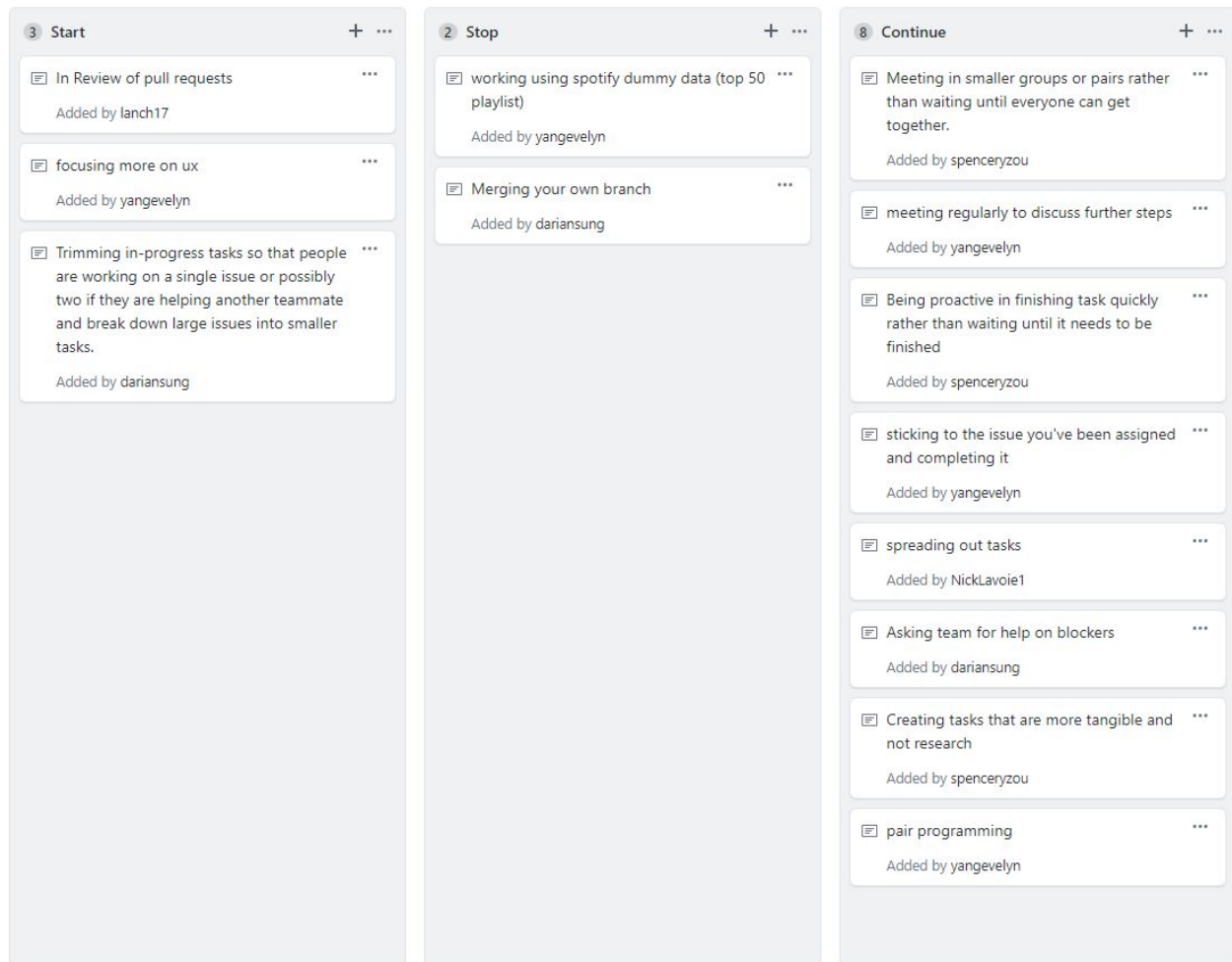
- Focus on user experience and interface
- Trim large issues into smaller tasks
- Review pull requests before merge

### e. MAJOR "STOP" POINTS

- Working with the stub test playlist
- Merging your own pull request

### f. MAJOR "CONTINUE" POINTS

- Meeting in small groups rather than waiting to get together
- Sticking to assigned issue
- Asking for help with blockers

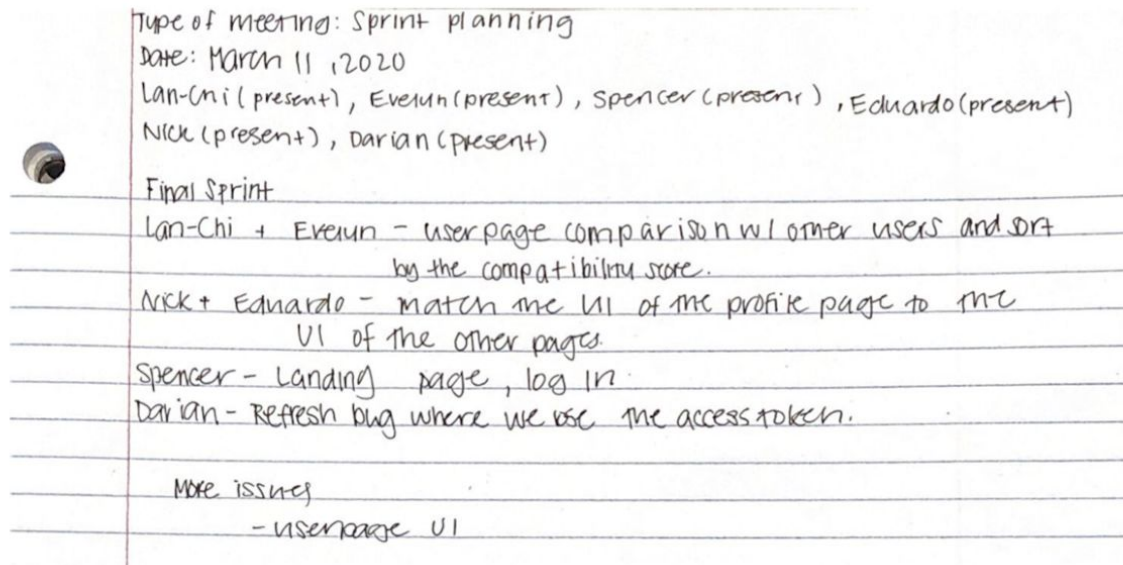


**Retro 3 Artifact**

## **V. DIFFICULTIES**

In phase 2, we had the most trouble figuring out how to use Google Firebase. None of us had any experience working with a database, so we initially had to figure out how a database even worked. This led to intangible research tasks where it felt like little progress was being made. In addition, after creating the database, we had to figure out how to write and read info from the database in our code. We also ran into the obstacle of the Spotify API limiting the amount of requests that can be made in a period of time. With this issue, our analysis algorithm could only handle about 150 songs before Spotify would return a 429 error for too many requests. We addressed this to the best of our ability by making our get requests more efficient through reusing request information. In addition, we realized that eventually comparing all users top playlists to the selected playlist would be impossible due to the get request limit. Thus, we decided to store user top playlist information arrays in the database after the user selects their top playlist in settings. This change would have the added benefit of making the comparison to all users faster.

# PHASE 3

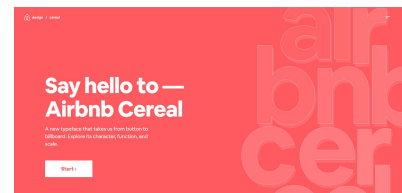
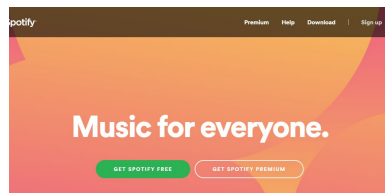
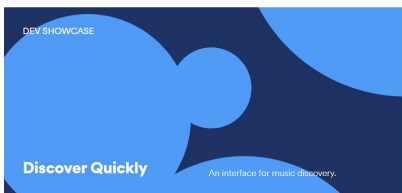


## Final Sprint Notes

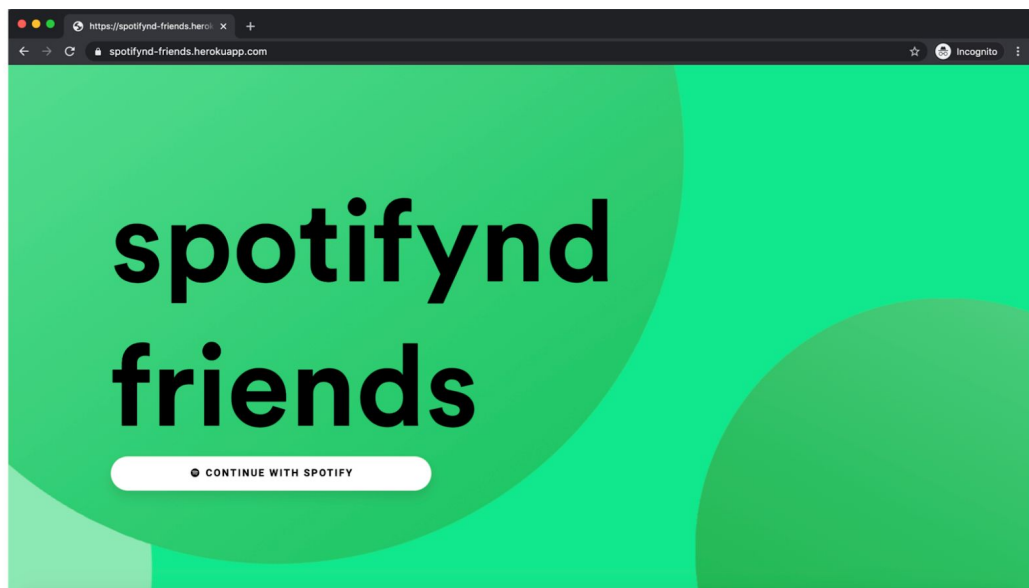
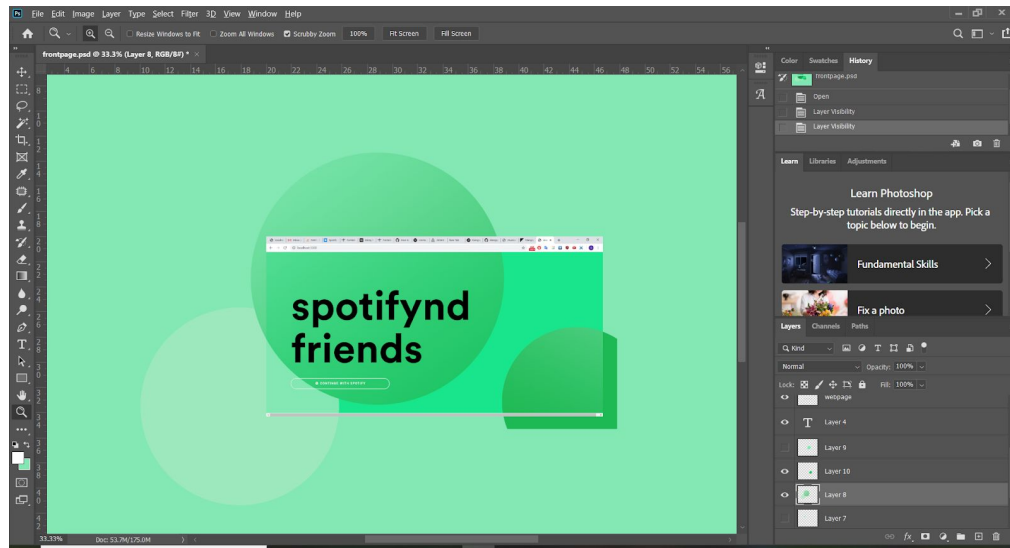
## FEATURES

### I. LANDING PAGE UI

- Inspiration for the landing page was from different pages that exist within Spotify and other tech company designs. The color scheme is very vibrant and bright, and contains animated circles.



- b. The new landing page was originally a white background but was changed to all green.



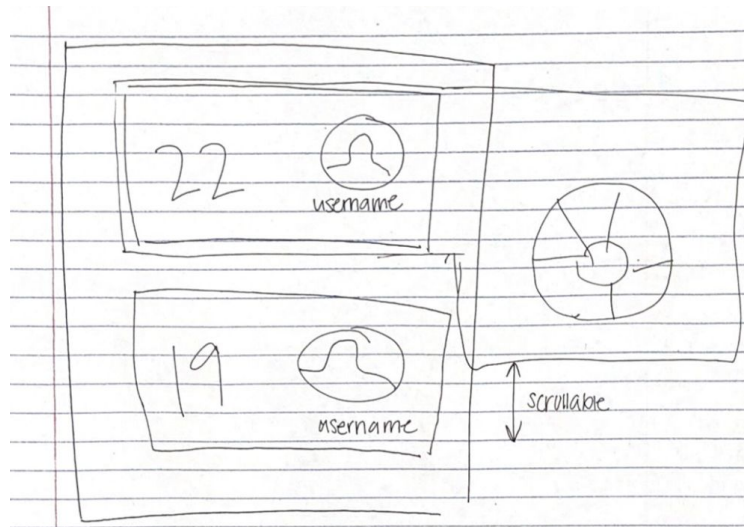
## II. USER COMPARISON

- a. This feature implements user-user interaction. This is a key functionality that allows users to interact with each other.
- b. After clicking a single playlist, instead of comparisons happening with a dummy playlist, comparison happens with every top playlist of users who have the same location set as the current user.
  - 1. Attributes of each top playlist are stored in the database and are fetched
  - 2. Ordering based on compatibility scores from largest to smallest.

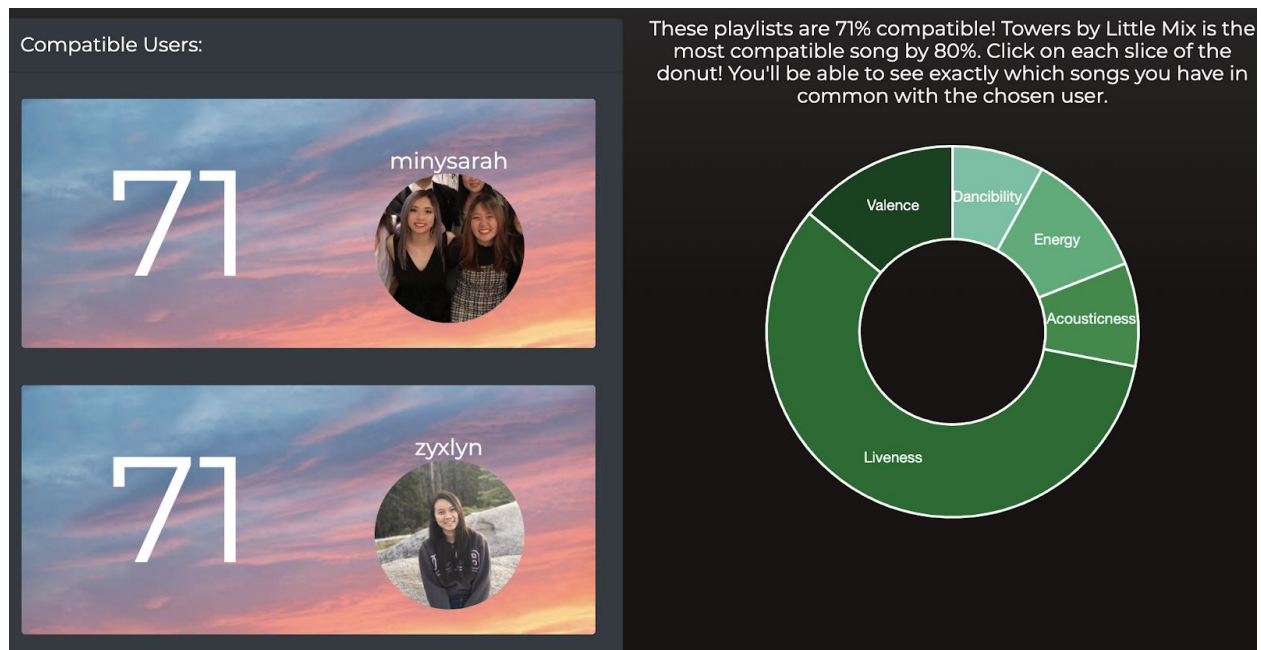
### III. USER CARD UI

- Each user was placed in a Bootstrap Card component, and their profile picture is also pulled from the database.
- Users can click on the compatibility score in order to view a wheel that show attributes of songs that the playlists had most in common with.
- Each slice of the wheel is able to be clicked to reveal which song was most compatible in each attribute genre.

Initial vision:

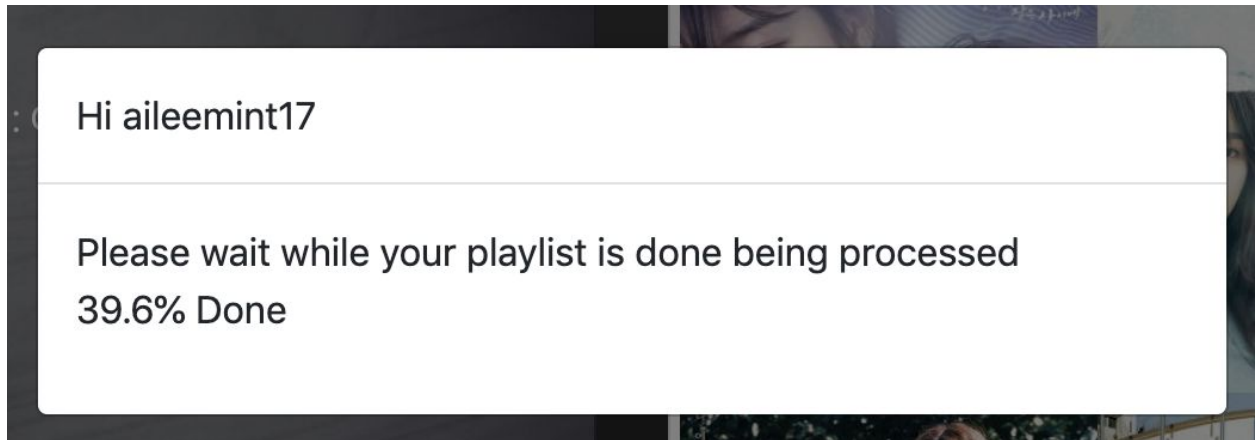


Final product feature:



#### IV. MODAL DURING PROCESSING TIME

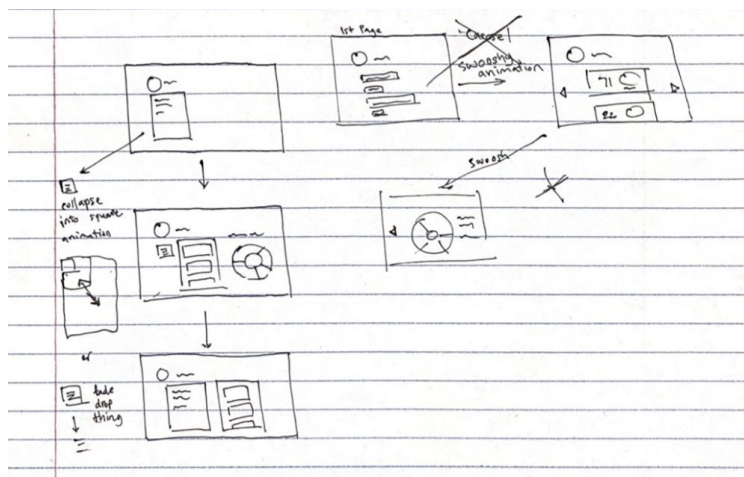
- Initially, an error would occur if the user left the settings page before the playlist attributes were written into the database
- Our solution was to add a Modal that wouldn't allow the user to leave until all



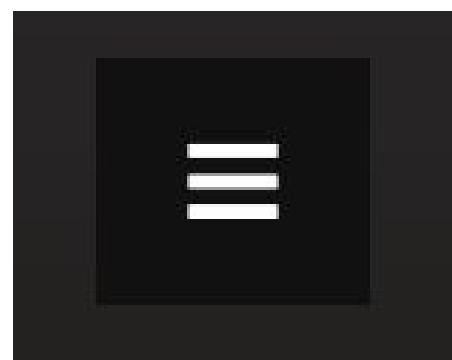
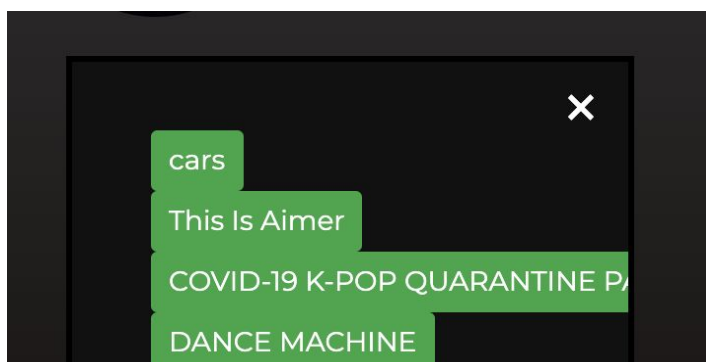
songs in the playlist were analyzed and pushed into the database.

#### V. HAMBURGER ICON AND COLLAPSIBLE COLUMN

- Initial vision: The card that holds all the current user's playlists would be able to be collapsed into a hamburger button.



- Final Product feature: The X button would collapse into the hamburger button and the hamburger button would reopen the playlists to make another comparison.





## **VI. PACMAN LOADING SCREEN**

- a. Initially, we had a login button which would redirect to a similar looking page that contained an enter button. This page had functionality to redirect to the user page, once the enter button was clicked.. Thus, our feature wanted to be a loading screen instead of a enter button, while having the same functionality. This function makes use of a proxy url.



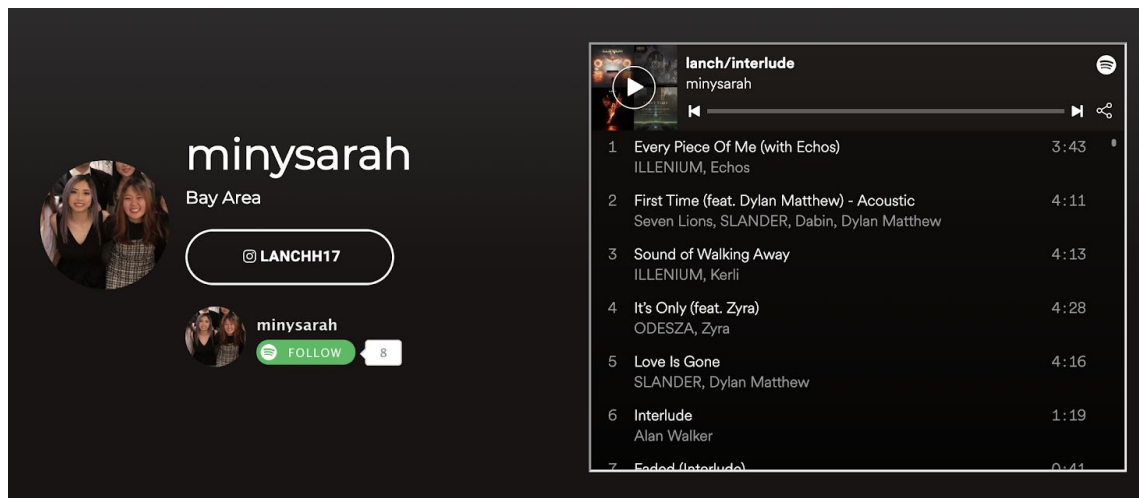
## **VII. WINDOW.STORAGE.SESSION**

- a. Initial issue: when the refresh button was clicked, then the access\_token would be lost. After losing the access\_token, an error would occur and break the program. This function stores the access\_token into window.storage.session so that in the same browser, the access\_token would always be stored.
- b. The refresh\_token is now also able to be used in order to get a new access\_token should the current one be expired.

## **VIII. LOGIN ERROR PAGE**

- a. In the case of a username that contains special characters like: ".", "[", "]", "\*", "#", or "\$", the user would be redirected to a login error page that directed a message to them, indicating that the user would be unable to use the program.
- b. This is to account for the edge case for an error because special characters would not be able to be used as paths in the database.

## IX. PROFILE PAGE UI



- New features were added to make the profile page of each user more interactive and add new social media aspects.
- The instagram of each user would now be displayed and be a button that would redirect the user to the instagram profile page of the selected user.
- A follow button also allows the current user to follow other users on spotify.

## DIFFICULTIES

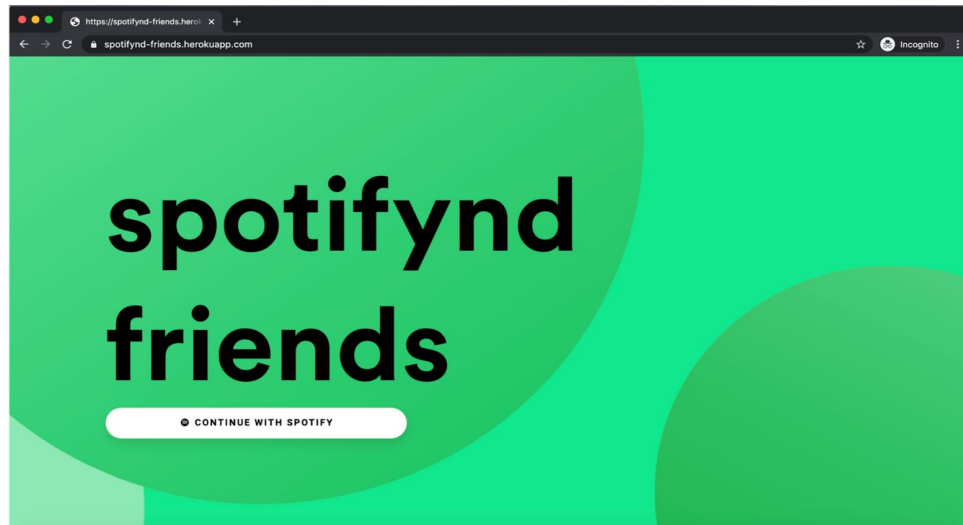
- Initially we did not know about `window.storage.session`, so much of the time was devoted to trying to store the `access_token` into Firebase and using Firebase authentication.
- The error of the special characters in Firebase was such an obscure edge case that it only presented itself within the last hour of development, leading the team with little time to develop a solution.



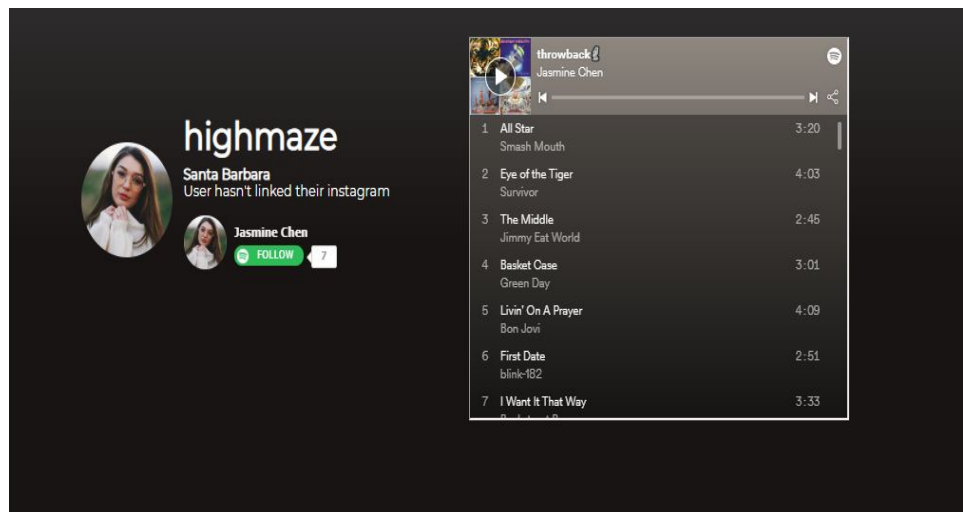
# PHASE 4

## I. DESIGN/FEATURE HIGHLIGHTS

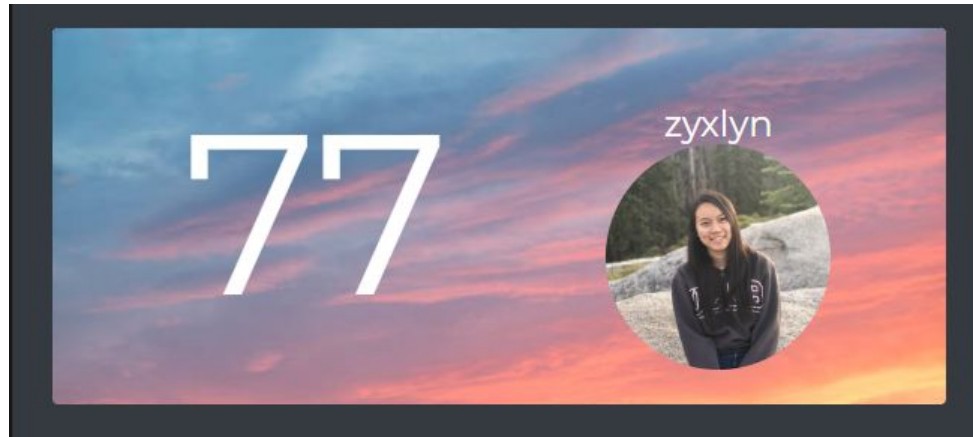
### a. Landing Page UI and Animation



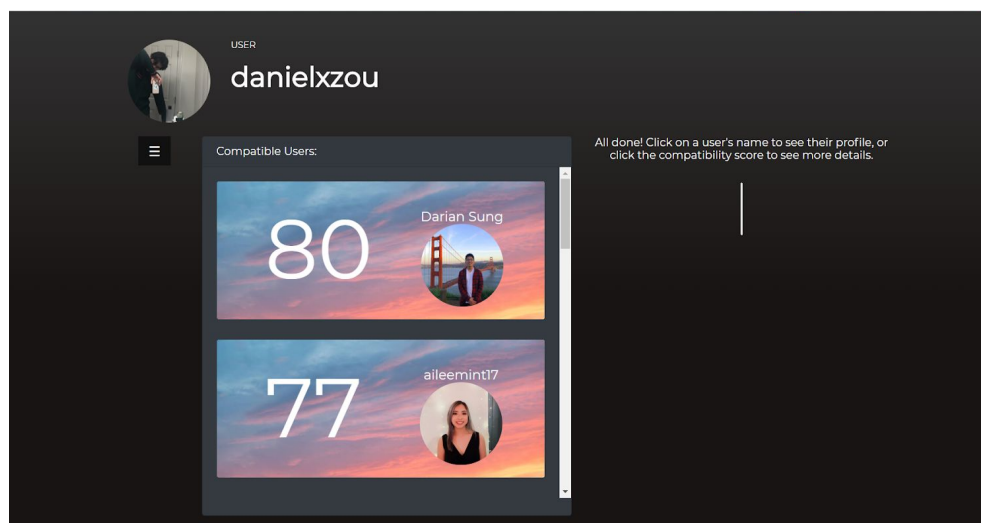
### b. Playable Embedded Top Playlist on Profile



### c. Card Design on UI



### d. Use of Profile Pictures



## **II. FUTURE WORK**

- a. Clean up UI
  - 1. It is hard to tell that the hamburger hides the playlists of the users. This feature needs to be more transparent to the user.
  - 2. Should be able to see another user's top playlist in the compatibility window without having to click on their profile.
  - 3. Random small grammatical errors throughout the web app need to be fixed.
- b. Fix Small Bugs
  - 1. If you click on a compatibility score while information is still loading, the wrong donut will display.
  - 2. Sometimes when you go to user page, there is a bug where some users can't see every other user, it depends on each person.
- c. Facebook Integration
  - 1. We originally wanted to be able to connect your Facebook so you can compare your playlist specifically amongst your Facebook friends.

## **III. CHALLENGES OF THE WHOLE PROJECT**

- a. The ramp up period in the beginning took longer than we expected.
  - 1. Deciding frameworks, languages to use
  - 2. Lack of familiarity with React, Node.JS, Firebase
  - 3. Lots of time dedicated to researching as a result
- b. Having achievable sprints and productive scrum meetings
  - 1. Pair programming was the best solution for more productive sprints
  - 2. It took us a while to get accustomed to agile development, sprint planning and having a vision for the team as a whole for each sprint
- c. Our code is heavily reliant on GET requests from Spotify, it is hard for us to account for bugs that occur on Spotify's side
  - 1. Spotify cannot handle more than approximately 150 get requests—playlists longer than 150 songs sometimes do not work
  - 2. Get requests take a significant amount of time due to large amount of data returned from Spotify for each track/user

Challenge	Solution/Potential Solution
Lack of familiarity with React, Node.js, Firebase	Dedicated significant amount of time to researching the different technologies we could/should use
Too much research, not enough tangible results	Restructuring issues to have smaller, incremental functional improvements
Conducting useful sprints	Pair programming improved sprint productivity
Productive scrum meetings	Practice with Agile development/sprint planning allowed us to develop a solid team vision
Spotify cannot handle more than ~150 GET requests, limiting playlist size	Only show first 150 songs on a playlist
GET requests take significant amount of time	Save more data in our database to reduce number of times we perform GET requests

# EVALUATION/TESTING

## I. MVP EVALUATION

During lab05, our MVP was evaluated by a member of another project team. Our presenter explained the purpose of our app and guided the reviewer through its usage. The reviewer thought the concept of our app was “really cool and innovative” and was excited to see the final product. However, it was lacking in its current state in terms of features and UI. Most notably, the MVP did not allow for comparisons with other users’ playlists which the reviewer thought would have been very useful. It instead only allowed the user to compare their playlist with Spotify’s Top Hits playlist. There was also a bug where only the first 10 playlists were displayed for comparison. The reviewer had suggestions for features we could add in the future, such as displaying playlist pictures next to each playlist name and integrating Facebook Messenger as a chat service between users.

[Click here for a link to the MVP demo.](#)

## II. FINAL PRODUCT EVALUATION

After the code freeze on March 12, we shared our webapp with a campus ministry Facebook group to populate the user base and receive feedback. This would be the first time that people were using the app without one of the developers present for guidance so this would test how easily the user could learn to use our app independently. About 30 users created a profile and used our application and none reported any difficulties with understanding how to use it. Several said that they thought the app was very cool and would like to see the user base expand so there were more people that they didn’t know. One user was unable to set up their profile because their username had a ‘.’ in it which was not acceptable syntax for Firebase paths. Another user was stuck at 0% when loading their playlist in the settings page, possibly due to local songs that were part of the playlist.

## III. UNIT TESTING LIBRARIES

Library	Purpose
Mocha	Used as a test runner and defined the syntax for writing tests
Chai	Easily readable assertions that replaced the classic TDD assert style
Enzyme	Mounted/rendered React components for testing
Sinon.JS	Stubbed out functions for unit tests

# ACKNOWLEDGMENT OF RESOURCES

## **SPOTIFY BRAND GUIDELINES**

All Spotify related images and themes were implemented according to their branding guidelines: [Branding Guidelines](#)

## **SPOTIFY API**

For usage of the Spotify API, we consulted their official documentation: [Documentation](#)

## **FIREBASE API**

For usage of the Firebase API, we consulted their official documentation: [Documentation](#)  
We also looked at this tutorial from tutorialspoint: [Firebase Tutorial](#)

## **REACT API**

For usage of the React API, we consulted their official documentation: [React Top-Level API](#)  
For toolchain recommendations, we consulted the following page: [Create a New React App](#)

## **NEXT.JS API**

For usage of the Next.js API, we consulted their official documentation: [Getting Started - Documentation](#)  
We used this quickstart guide as our foundational project structure: [Learn - Getting Started](#)

## **HTTP REQUESTS**

We used the request library to make many of our HTTP requests: [request](#)  
We also used the axios library to handle some asynchronous functions: [axios](#)

## **UI ELEMENTS**

Various UI elements were sourced from the react-bootstrap library: [React Bootstrap](#)  
The Pacman loading animation after login and playlist analysis loading animation were sourced from the react-spinners library: [react-spinners](#)  
The circular graph with compatibility details was sourced from react-chartjs-2: [jerairrest/react-chartjs-2: React wrapper for Chart.js](#)  
The @emotion/core library was used to define CSS styling in JavaScript: [@emotion/core](#)

## **IMAGES**

Default Profile Picture: [Source](#)  
User Card Background Image: [Source](#)  
Settings Page Card Image: [Source](#)

## **FONTS**

Montserrat:

Copyright 2011 The Montserrat Project Authors

(<https://github.com/JulietaUla/Montserrat>)

Google Fonts page: <https://fonts.google.com/specimen/Montserrat>

Roboto:

Copyright 2011 Google Inc. All Rights Reserved.

Google Fonts page: <https://fonts.google.com/specimen/Roboto>

## **MOCHA API**

For usage of Mocha, we consulted their main website: [Mocha - the fun, simple, flexible JavaScript test framework](#)

## **CHAI API**

For usage of Chai, we consulted their official guide: [Getting Started Guide](#)

## **ENZYME API**

For usage of Enzyme, we consulted their API reference: [API Reference · Enzyme](#)

## **SINON.JS API**

For usage of Sinon.JS, we consulted their API documentation: [API documentation](#)