Harmony

Team #16

Design Document

Thomas Hynes

Aleksei Natalenko

Alex Richardson

Michael Con

Pratham Agrawal

Purpose **2**
Functional Requirements 3
Non-Functional Requirements 7
Design Outline **8**
High Level **8**
Interaction Between Components 10
Design Issues **12**
Functional 12
Non-Functional 13
Design Details **16**
Class Design 16
Description of Classes and Interaction between Classes 17
Diagrams 20
UI Mockups **26**

Purpose

With the advent of music streaming through platforms like Spotify and Youtube, the amount of music we consume as a society has drastically increased in the past decades. Furthermore, the internet has allowed people to listen to a variety of music, different from how it used to be when everyone listened to the same radio or TV channels with the same music. Harmony is a music based social media platform that capitalizes on this growing trend by incorporating a social aspect to how we experience and share music among our friends. This aspect of music is a medium that has not been explored by many other applications, so Harmony aims to fill this large gap in the industry and appeal to a wide range of consumers. We would also like to incorporate a daily activity into our application, as it will not only encourage users to log in to the app at least once a day but also make sure the community of social music listeners is active. The idea is to make this daily activity a spontaneous event, challenging users to keep a daily streak going and strengthening the bond between the consumers.

While other platforms are available for music-related content, we believe that none of them integrates a social aspect as to how we consume music. For example, Spotify shows recently played songs from your friends and recently added the ability to create shared playlists. But, these remain minor features in the overall platform and are not core components of how users interact with Spotify.

We seek to address these problems with Harmony, adding a more robust social aspect. We will build upon Spotify's features by creating dynamic joint playlists that are automatically generated according to the top songs listened to recently by the members of the playlist, and allowing users to see the song their friend is currently listening to.

Functional Requirements

1. Users can create and customize their account
    a. As a user, I would like to be able to register for a Harmony account
    b. As a user, I would like to be able to sync with my Spotify account
    c. As a user, I would like to be able to un-sync with my Spotify account
    d. As a user, I would like to be able to edit my name within the profile settings
    e. As a user, I would like to be able to change my profile picture within the profile settings (with certain preset images to choose from)
    f. As a user, I would like to be able to change my email/password within the profile settings
    g. As a user, I would like to be able to use my email to recover my password if I forget it.
    h. As a user, I would like to be able to adjust my notification settings to allow/not allow for certain or all alerts
    i. As a user, I would like to be able to set my location for time zone purposes
2. Users can create, and edit a circle of friends
    a. As a user, I would like to see my friend's profile information such as their profile picture, recent songs, and favorite genres
    b. As a user, I would like to see suggestions for people to befriend in Harmony
    c. As a user, I would like to be able to access a list of my friends
    d. As a user, I would like to be able to send friend requests to other users
    e. As a user, I would like to be able to unfriend people or deny friend requests

f. As a user, I would like to be able to block people from seeing my account or contacting me
g. As a user, I would like to receive a notification when someone sends me a friend request
h. As a user, I would like to receive a notification when someone accepts my friend request

3. Users can share their music tastes with their friends
    a. As a user, I would like to have an auto-generated playlist based on songs that I and my friends listen to
    b. As a user, I would like to be able to see songs that my friends and I listen too
    c. As a user, I would like to be able to publicly rate a song
    d. As a user, I would like to be able to publicly rate an album
    e. As a user, I would like to be able to publicly rate an artist/band
    f. As a user, I would like to be able to share a snapshot of my profile on other social media platforms
4. Users can directly communicate with their friends
    a. As a user, I would like to be able to receive messages from my friends within the app
    b. As a user, I would like to be able to send messages to my friends within the app
    c. As a user, I would like to receive notifications when I receive a message
    d. As a user, I would like to be able to share a playlist with my friend through the app

e. As a user, I would like to be able to share an album with my friend through the app
f. As a user, I would like to be able to share an artist/band with my friend through the app
g. As a user, I would like to be able to share a song with my friend through the app

5. Users can learn more about their music taste
    a. As a user, I would like to see an analytics page for the music I listen to
    b. As a user, I would like to receive recommendations on possible songs I would like based on my own tastes and my friend's taste
    c. As a user, I would like to receive recommendations on new artists I may not be aware of
6. Users can use the app as expected and provide feedback when they can't
    a. As a user, I would like to be able to provide feedback on the app to the developers
    b. As a user, I would like to report bugs to the developers in the app
    c. As a user, I would like to be able to log in so that I can access my account and communicate publicly with a recognizable name
    d. As a user, I would like to be able to download the app onto my Android phone
    e. As a user, I would like to be able to download the app onto my Apple phone
7. Users can participate and interact with a daily activity
    a. As a user, I would like to participate in a daily activity
    b. As a user, I would like the daily activity to happen at a spontaneous moment of the day
    c. As a user, I would like the daily activity to disappear after 24 hours

d. As a user, I would like to have a private history of my own submissions for the daily activity
e. As a user, I would like to see my friend's results from their daily activity
f. As a user, I would like to respond to my friend's results from the daily activity
g. As a user, I would like to receive a notification when it is time to participate in the daily activity

Non-Functional Requirements

1. Client Requirements

   As a Developer

   a. I want the application to be available for Android devices.
   b. If time permits, I would like the application to be available for web browsers.

2. Server Requirements

   As a Developer

   a. I want our server to save the User's last actions and state.
   b. I want our server to support real-time communication with the client.
   c. I want our server to send User information to the database.

3. Design Requirements

   As a Developer

   a. I want our application to retrieve data from the Spotify API seamlessly and without a prompt from the User.

b. I want our database to store information such as posts for a limited amount of time.

4. Performance Requirements

As a Developer

a. I want our application to be able to handle multiple concurrent users.
b. I want our application to be responsive to User input.
c. I want our application to handle errors and unexpected bugs without causing a major disruption to the User.

Design Outline

High Level

This project will be an app where users can share their music and find out what kind of music their friends are listening to. It will be designed using React with a client-server architecture as seen in the diagram below. The client will communicate with the server, which can communicate with either the database or the Spotify API to fulfill the client's request. Furthermore, it will use a database to store information about a user's account, as well as information gotten from a Spotify API relevant to their Spotify account.

Server:

- Interacts with any number of Clients by receiving and handling requests then either communicating with the database or performing functions itself in order to send a response back

- The server will be able to handle requests such as login requests, sign-in requests, messaging requests, and more
- The server will be receiving and requesting information such as user credentials, message contents, post contents, and song information
- Interacts with the database by sending requests for information and waiting for a response or sending new data to the database
- Uses Spotify API to communicate with Spotify and retrieves the user's listening to information

Client:

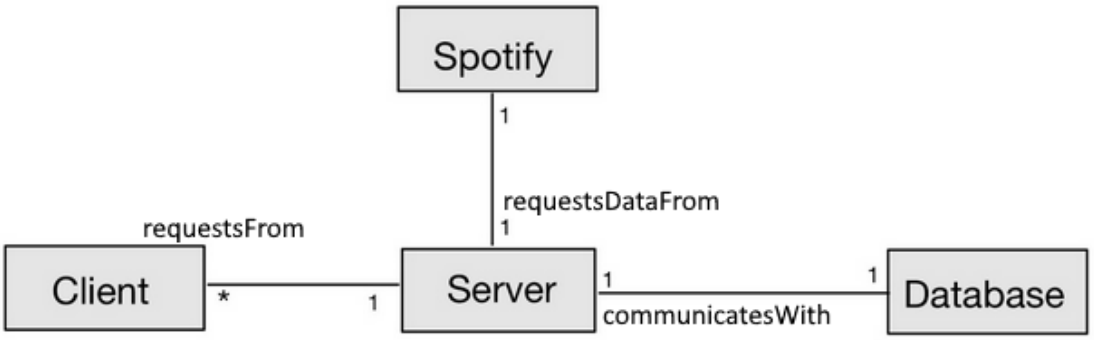
- Any number of clients can communicate with one server
- Connects with the server, once connected responds to any requests and can also send requests

Database:

- Stores user information both generated in Harmony or gotten from Spotify
- Only interacts with Server as explained above

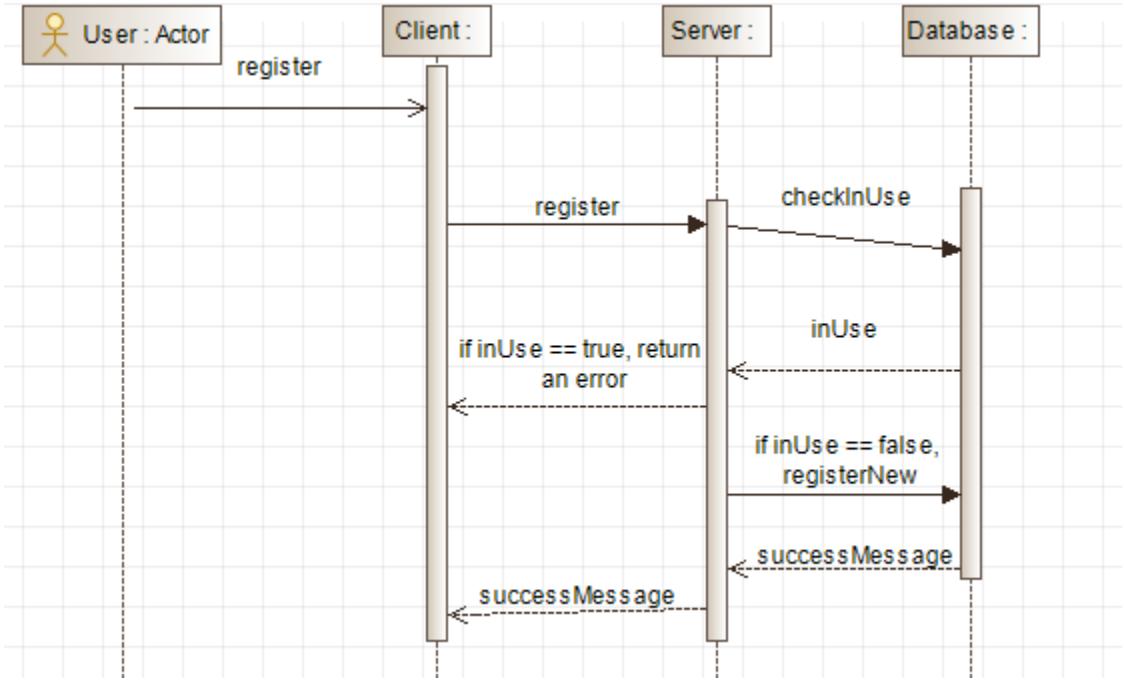
Spotify API:

- Not owned or developed by us
- The server will use a public API to get data from Spotify



Interaction Between Components

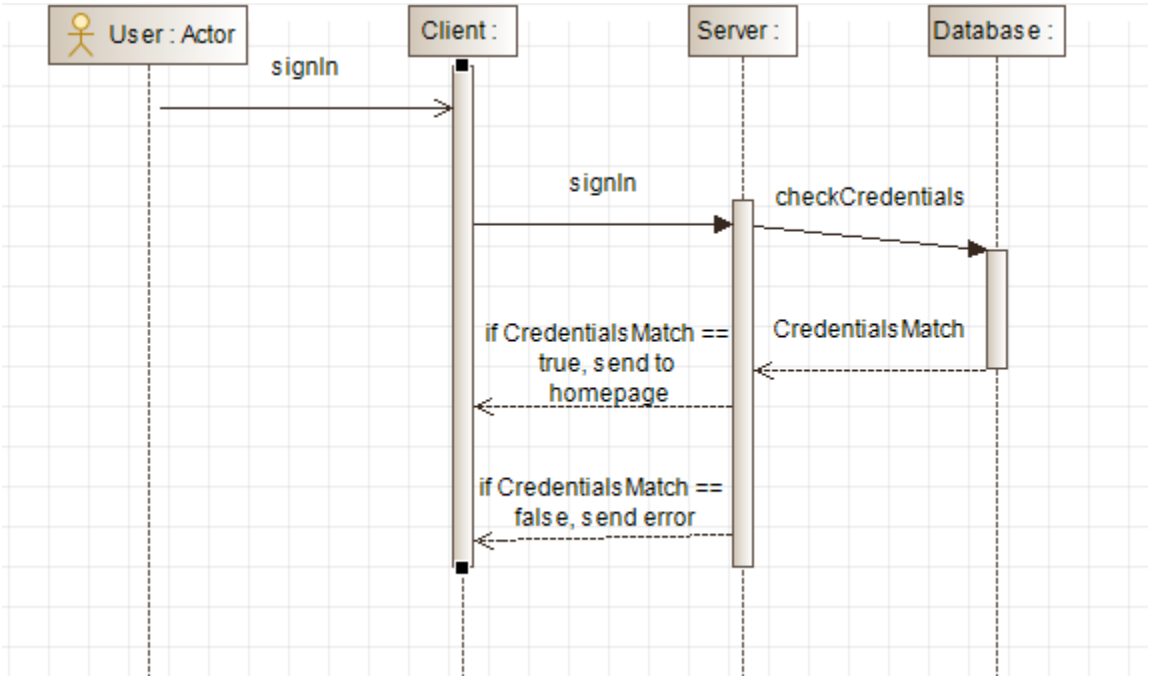
Sequence diagram for when the user registers their account:



Explanation: When the user registers a new account, the client sends a registration request to the server containing the user's information(username, email, and password). The server then checks

if the email or username is already being used by a registered account, it then returns the variable inUse accordingly. If inUse == true, then the username or email is already in use and an error is displayed to the user and the client exits out of the function. But if inUse is false then the email and username are available and the server creates a new account, sends that new account info to the database, and returns a success message to the client.

Sequence diagram for when the user signs in:



Explanation: When the user attempts to sign in, the client sends a sign-in request to the server containing the user's information(username and password). The server then checks with the database to see if the password matches the associated username, or if the username even exists in the database. If the credentials do match, the client switches to the homepage. But if the credentials do not match, an error is presented to the user.

Design Issues

Functional

Functional Issue #1 - What information will be required to sign up?

- Option 1: Just email and password
- Option 2: Just email, username, and password
- Option 3: Email, username, password, and Spotify account
- **Choice - Option3: Just email, username, and password**

Reasoning: We will allow users to create an account without their Spotify linked, but certain features will only be available if you have your Spotify account linked. This way you can be involved and interact with your friends through the app even if you don't have a Spotify account, or just don’t want to link it. There will be an option in the profile settings to link your Spotify account. We believe that these three attributes of email, username, and password are the bare minimum of personal information necessary for a potential customer to sign up. Additionally, these attributes allow us to implement certain security measures. These security measures include password minimum security requirements, hiding the email from other users, and email verification upon signing up.

Functional Issue #2 - How will we allow users to navigate the different pages of the app?

- Option 1: Drop down menu at the top that contains a list of pages to navigate to
- Option 2: A “homepage that the user must navigate back to in order to go to a different page

- Option 3: A small bar at the bottom or top of the page that contains icons representing the different pages
- **Choice - Option 3: A small bar at the bottom and top of the page that contains icons representing the different pages**

Option 3 is the most intuitive option for mobile applications. It simplifies the UI for the users by having the fewest amount of menus and moving parts. Additionally, since the buttons will be on the top and bottom it frees up the center of the screen for the important content that the user will be interested in. Having a homepage as option 2 describes, will make the application feel complex and clunky. The user should be shown the important content right as they open the application. We also proposed using drop-down menus as described by option 1, but we decided that this format for menus would be better suited for a web page on a big screen. The small screen of a phone wouldn't be ideal for it. Additionally, a drop-down menu tends to fill the screen with different options that can be distracting for the User if they need to search for a specific option.

Non-Functional

Design Issue #1 - Which framework/technology should we use to develop the front end?

- Option 1: Flutter
- Option 2: React Native
- Option 3: Ionic
- Option 4: Alamofire
- **Choice - Option 1: Flutter**

Reasoning: Among all the choices, Flutter performs the best in terms of speed and app optimization. It is an easy technology to learn and adapt to. Although Flutter is fairly new when compared to the other options, Flutter has thousands of packages and a large community of developers that will help speed up development and help us with any errors we might encounter. Additionally, Flutter will be the easiest to connect to and communicate with Firebase because both technologies were created by Google. Furthermore, some of our team members have some familiarity with Flutter which will make the learning curve easier for the whole team,

Design Issue #2 - What web service will we use for our backend server?

- Option 1: Amazon Web Service
- Option 2: Azure
- Option 3: Firebase
- **Choice - Option 3: Firebase**

Reasoning: Because we decided to proceed with Flutter, Firebase seemed like the most sensible option. Both technologies were created by Google, hence they are easier to integrate into one another. Secondly, Firebase supports RealTime Database, which will be used for In-App chat. On top of that, Firebase has Google Cloud Functions that will allow the app to send notifications to the users. Finally, one of our members has relevant experience using Firebase which will save us time in learning new technologies.

Design Issue #3 - What database will we use?

- Option 1: RealTime Database
- Option 2: The Cloud Firestore

- Option 3: RealTime Database + The Cloud Firestore

Choice - Option 2: The Cloud Firestore

Reasoning: The reason we need to use Cloud Firestore is mainly optimization. Cloud Firestore is very scalable and it will be easier to build upon previous data already in the database. In a nutshell, Firestore has better-structured data and provides better querying. Another big reason is cost. Because Firestore doesn't make server calls as frequently as RealTime Database, it is much cheaper to run and maintain it.

Design Issue #4 - What API will we use to access Spotify data?

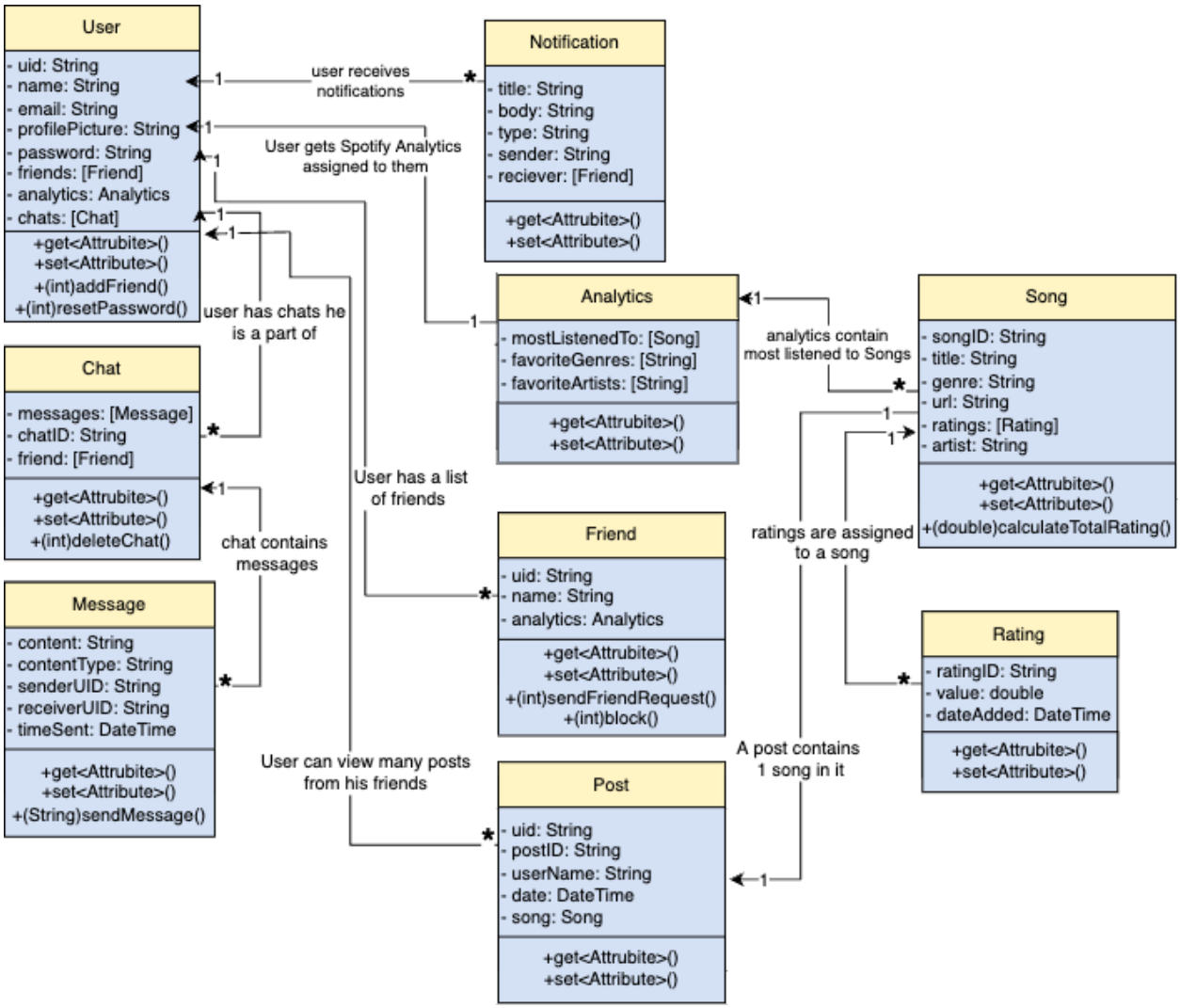
- Option 1: Web API
- Option 2: Spotify's Player API
- Option 3: SpotifyUserAPI
- Option 4: Use both Web API and Spotify's Player API
- Option 5: Use both SpotifyUserAPI and Spotify's Player API

Choice - Option 5: Use both SpotifyUserAPI and Spotify's Player API

Reasoning: Between Web API and SpotifyUserAPI, SpotifyUserAPI is the preferred choice for retrieving user-specific data such as playlists and saved tracks. Web API is much broader and doesn't have as many tools for accessing user-specific data. However, we also need Spotify's Player API in order to control the song the user is listening to, which is especially ideal for making social listening experiences. This can allow us to sync songs a user and their friends are listening to for example.

Design Details

Class Design



Description of Classes and Interaction between Classes

The classes described in the diagram above are based on the classes we have planned for the application. Each class has a list of attributes, operations and the corresponding relationships with other classes. These classes

User

- The user object is created when a new customer signs up for our application.
- Each user will be assigned a unique user ID.
- Each user will provide a name, email and password for login purposes.
- Each user will have a list of friends. These friends are other users on the application.
- Each user will have analytics. This analytics will be a variety of data points regarding the user's music consumption. The data will be displayed on the user's homepage.
- Each user will have a list of chats with other friends. Chats will only be available between friends and will consist of text messages.

Friend

- When a new User object is created, a Friends object will also be created. This friend object will initialize as empty but will gradually get populated as the User befriends other users.
- The friend class acts like a subclass of the class User.
- It contains information regarding the friends of a User.
- It contains the user IDs of each friend.

Chat

- Each chat will be assigned a unique chat ID.
- A chat consists of messages between two or more users.
- Messages will be strings and links to songs that users share between them.

Message

- Messages are sent between users and are of type String.
- Each message will have a sender ID and a Receiver ID.
- Messages are designated to go to a specific chat.
- We will record a timestamp of when the message was originally sent.

Notification

- A user receives notifications.
- Each notification will present itself to the user with a title and a body. The body will provide details about the notification such as “Michael sent you a friend request” or “You have a new message from Thomas”
- Our daily activities will be notifications,

Song

- A song object has a unique song ID, this will probably be provided by the Spotify API.
- Each song will have a series of attributes like title, artist, genre and rating.

- The rating for each song will be a compilation of user ratings for a specific song. This attribute will be empty at first and thus we should display a default value like ***none.*** For when it has no user reviews.
- Songs interact with the Analytics class. Analytics will have multiple songs such as the most listened to song, most popular song, favorite genres, and so forth.

Analytics

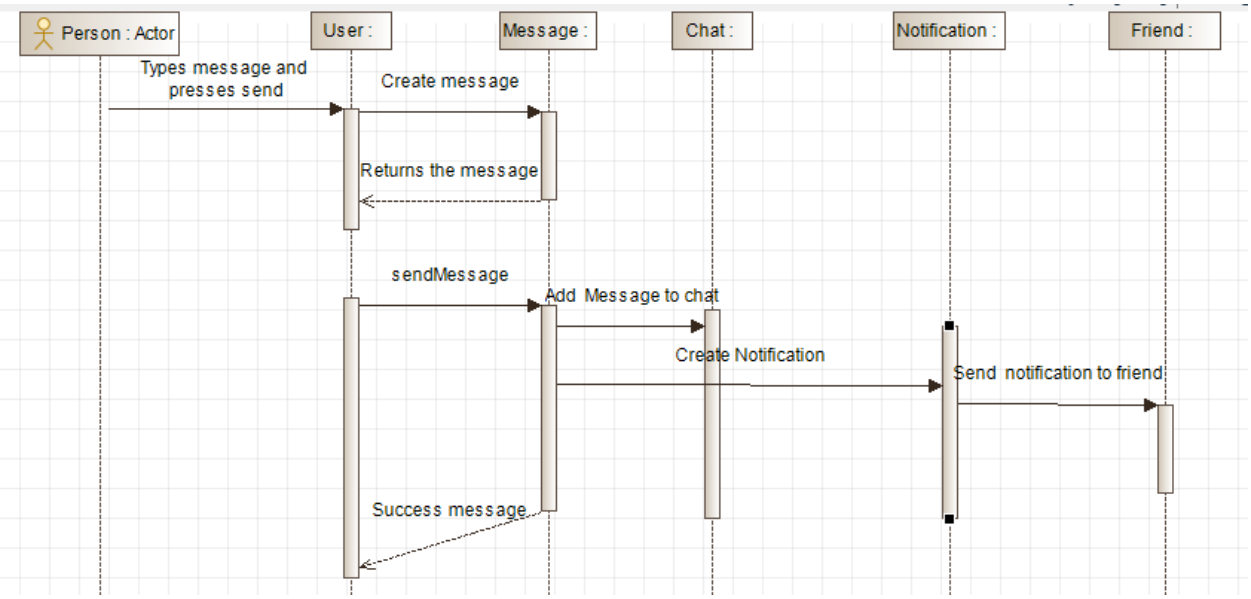
- The Analytics object contains information that is retrieved from the Spotify API connection.
- This class should only be created once the User establishes a connection to their personal Spotify account.
- It holds pertinent information such as the user's most listened-to song, favorite artist, and favorite genre. This can be expanded if need be.
- The User will create only one instance of the analytics object, this instance should be stored in the database.

Rating

- Each rating will have a unique rating ID from a given user.
- Ratings are calculated from multiple ratings from multiple users.
- Every rating corresponds to a specific song.
- Every song object should also create an object of type rating. This newly created instance of rating will have a null value which should be displayed as ***none*** or something similar.

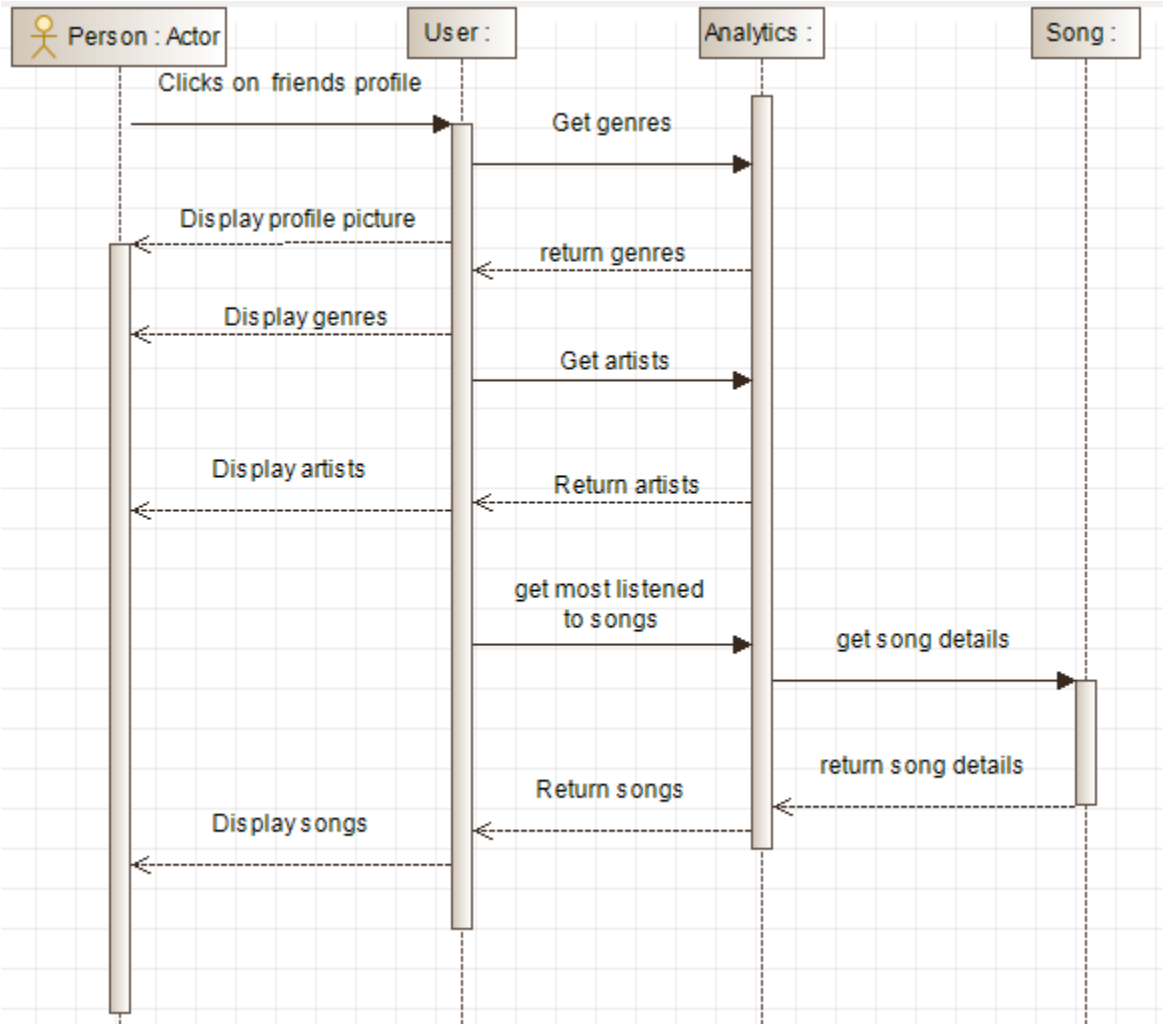
Diagrams

Sequence when a user sends a message



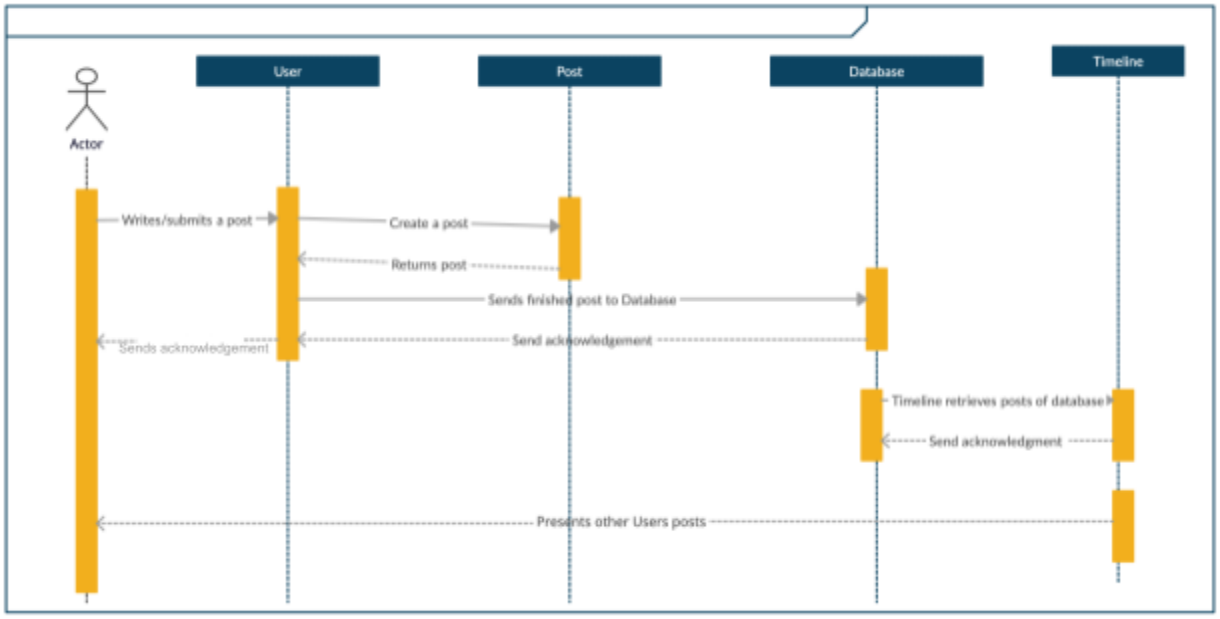
Explanation: When the person using the app types a message and presses send, the User class will create a new Message object. Then the user will call the sendMessage function of the Message Class on that new Message Object, which adds that message to the Chat Object that corresponds to the Chat the User is currently in, then creates a new Notification that is then sent to the recipient Friend Object notifying them that they received a message.

Sequence when a user opens their friend's profile



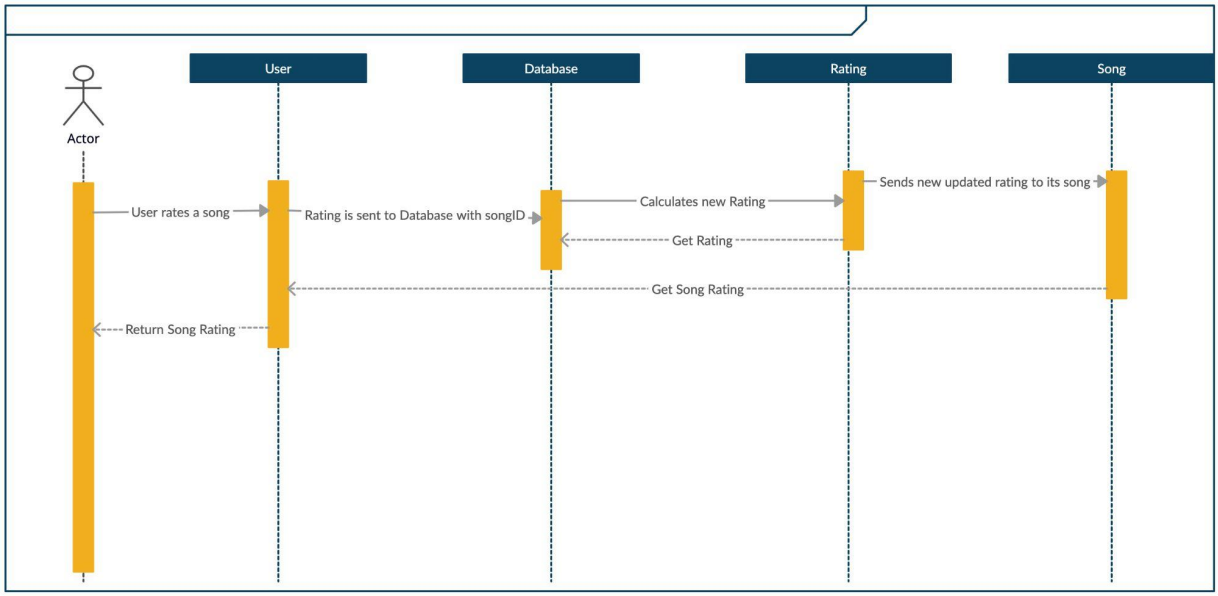
Explanation: When a person using the app selects their friend's profile, that User Object will display the profile picture, then send a call to the corresponding Analytics Object to get the top genres of that User, and then display those. Then a similar call for the top artists of the User and display those as well. Then the User Object sends a call to the Analytics Object to get the most listened to songs, so the Analytics Object has to get the Song information from each Song Object in the list. It then returns that information to the User, which then displays that.

Sequence chart when User uploads a post



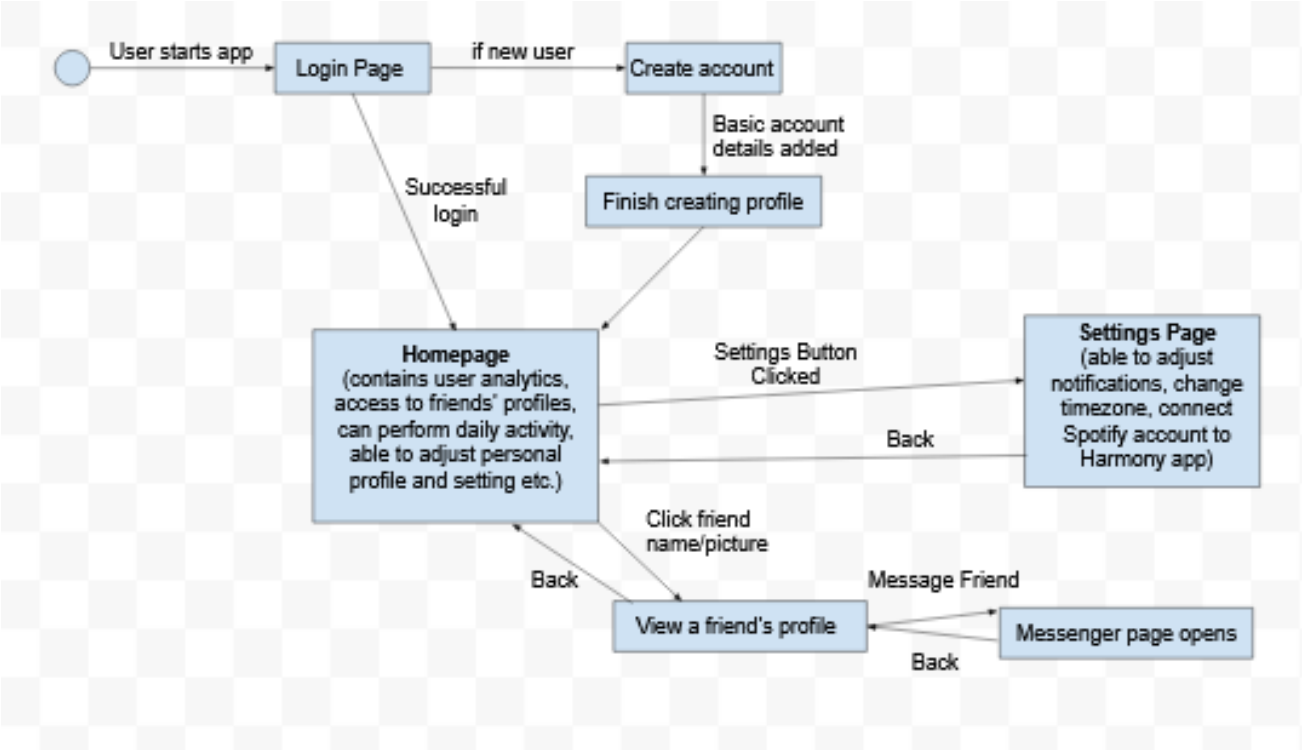
Explanation: When a person uploads a post to the application a post instance will be created by the user. This post is later sent to the database for storage and afterwards, the post is retrieved by the applications timeline. Once the timeline has the post it can display it to other users. It is important to note that posts are planned to be deleted after a specified amount of time. Thus the timeline only contains recent posts.

Sequence chart for User rating a song



Explanation: When a person rates a song this Rating is sent to the application's database. A new rating is calculated as an average from all user's ratings for that specific song. Afterwards, this song will now have access to this new average rating. This new rating will be visible to all users inside the application. Every song object has a unique rating object, it is a one to one relationship.

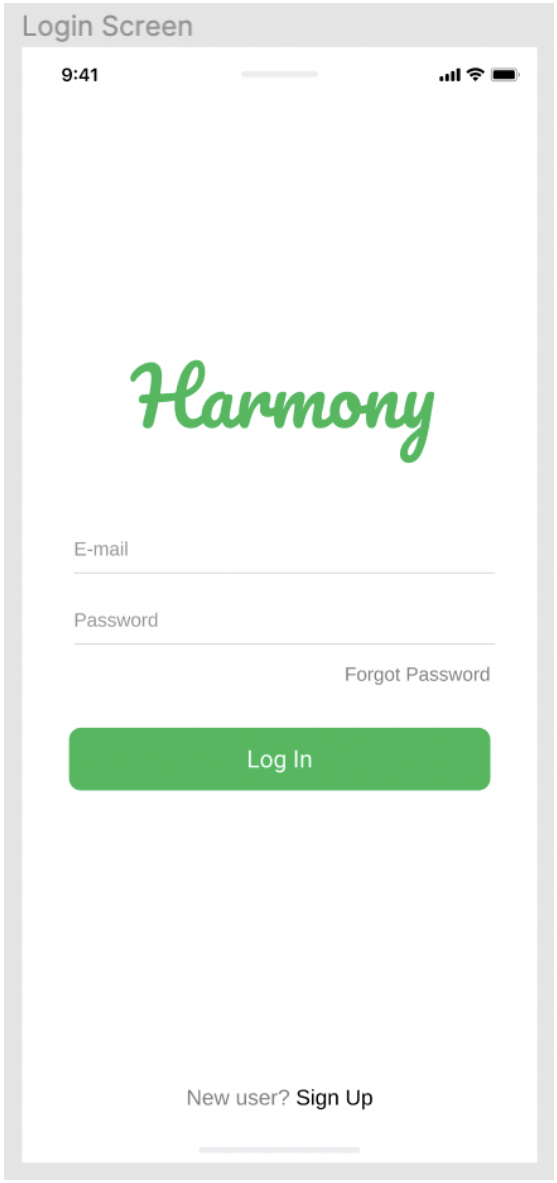
User Interaction Flow Chart

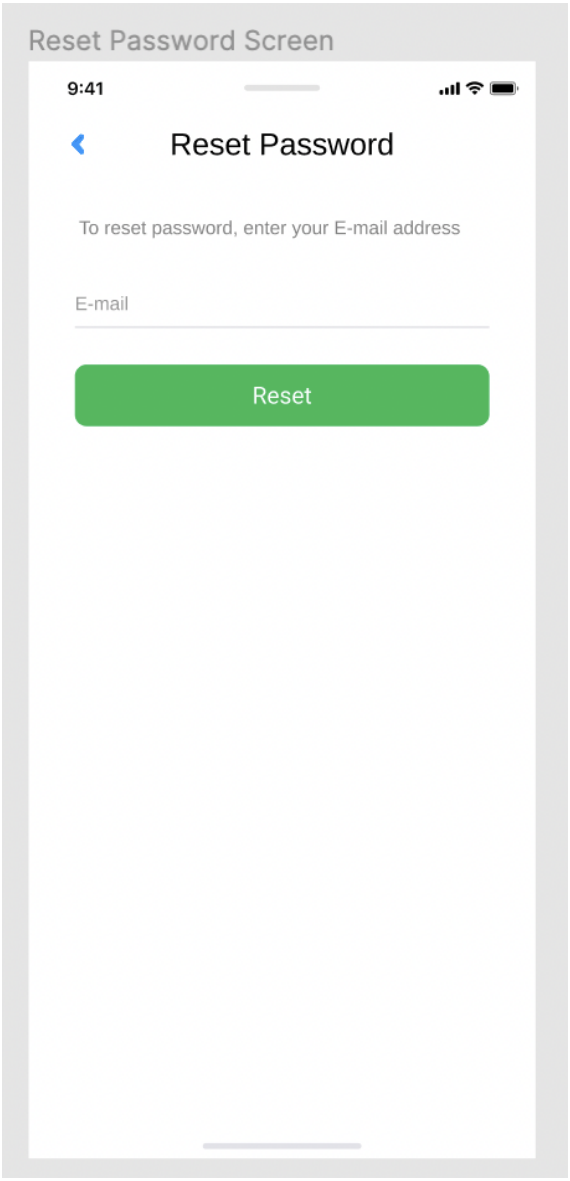


Explanation: The diagram above shows how a user interacts with our application. It shows all possible options and avenues for the User, we can see that most of the application revolves around the Homepage. This approach is meant to simplify the user experience, allowing for an easier learning curve. When a user first opens the app, they have the option to create a new account and populate basic and advanced details into their account. If they already have an account, this step is not necessary and they can just use their credentials to login and access the homepage the same. As we mentioned before, the homepage will contain the majority of information a user would need to access. Our goal is to not make this app complicated but also fun to use, so this homepage allows the user to be able to accomplish their needs in the app just a click/tap away. From the homepage, the user can access a settings page which allows you to adjust when and what notifications you would like to receive from the app, change time zones to

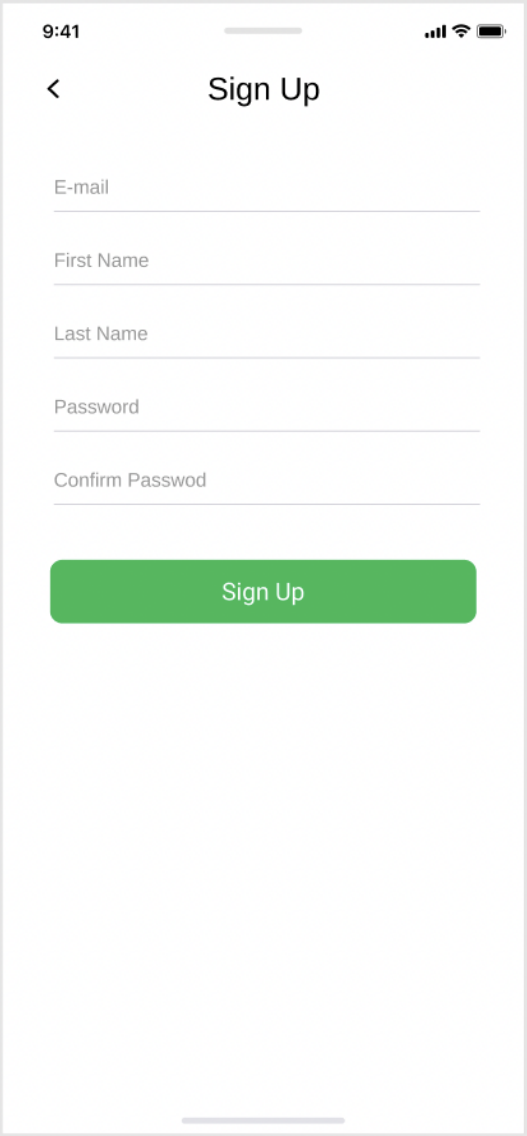
accommodate a user that wants to travel but would still like to participate in the user activity, and connect their Spotify account to the Harmony app in order to participate in the social features of the app. We can also visit a friend's profile and view their analytics or message them through the homepage. This should allow the app's community to be interactive and should also encourage those interested in the app to be persuaded through the various features of the app.

UI Mockups

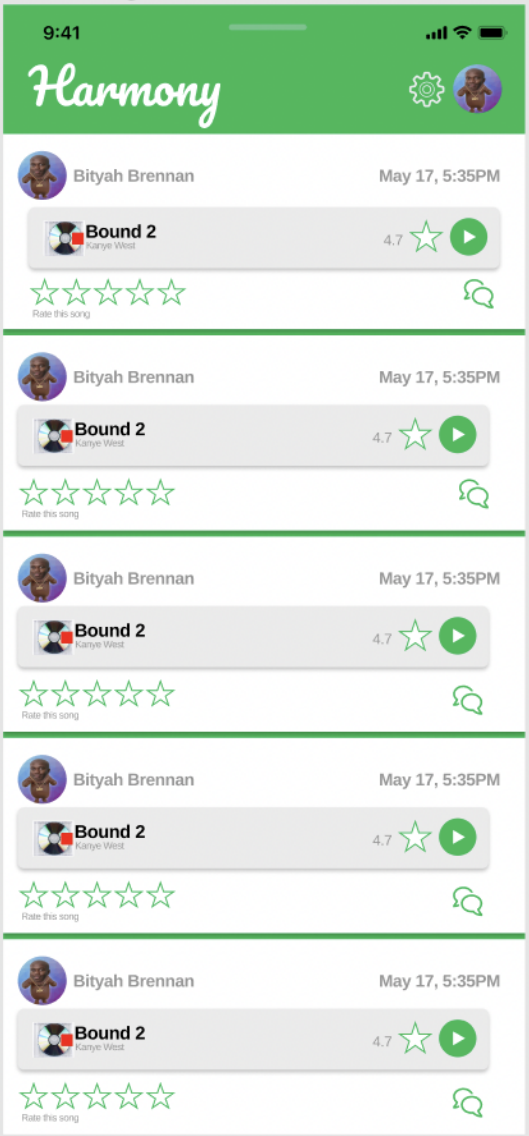

Login Screen
9:41
Harmony
E-mail
Password
Forgot Password
Log In
New user? Sign Up


Reset Password Screen
9:41
Reset Password
To reset password, enter your E-mail address
E-mail
Reset

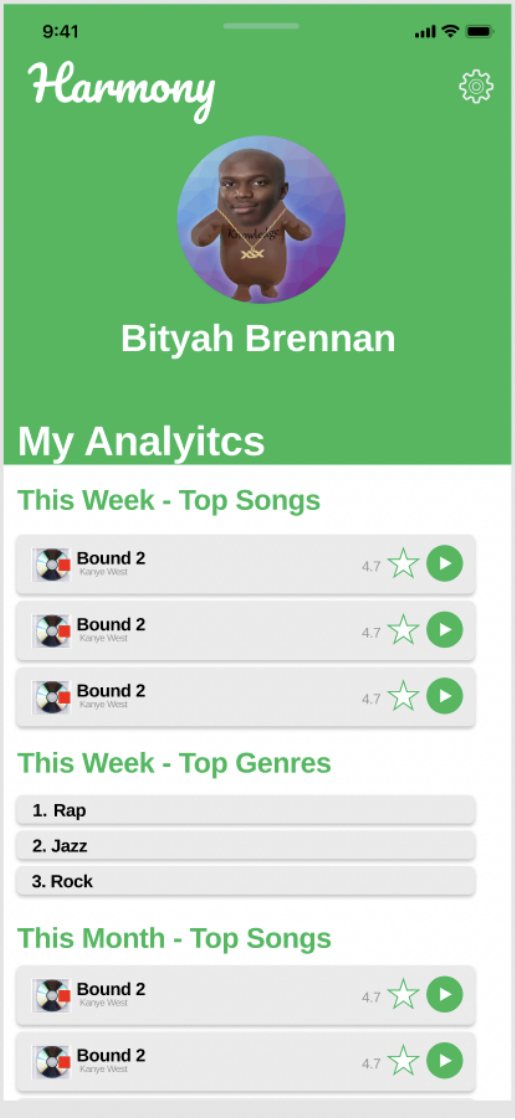
Sign Up Screen



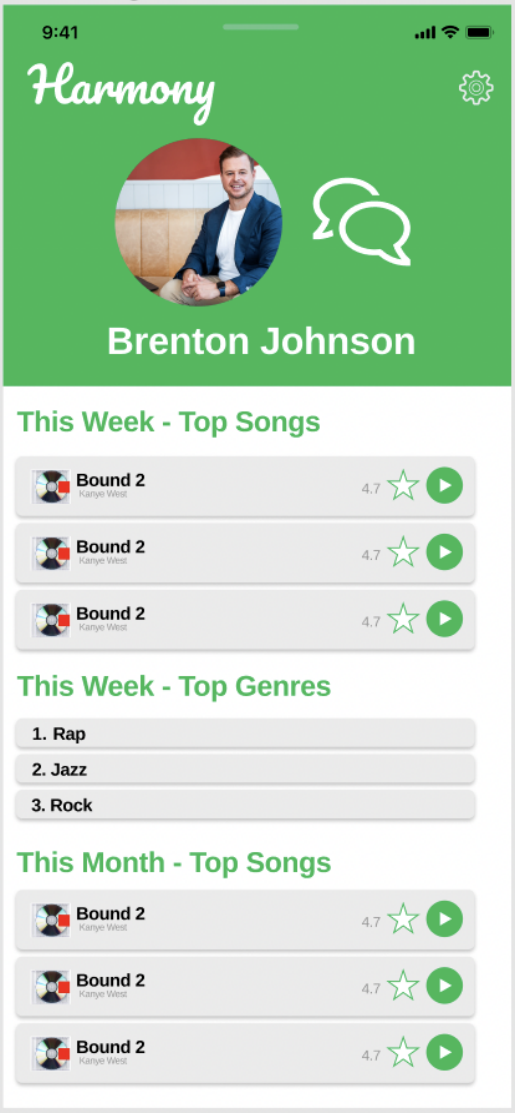
Home Page



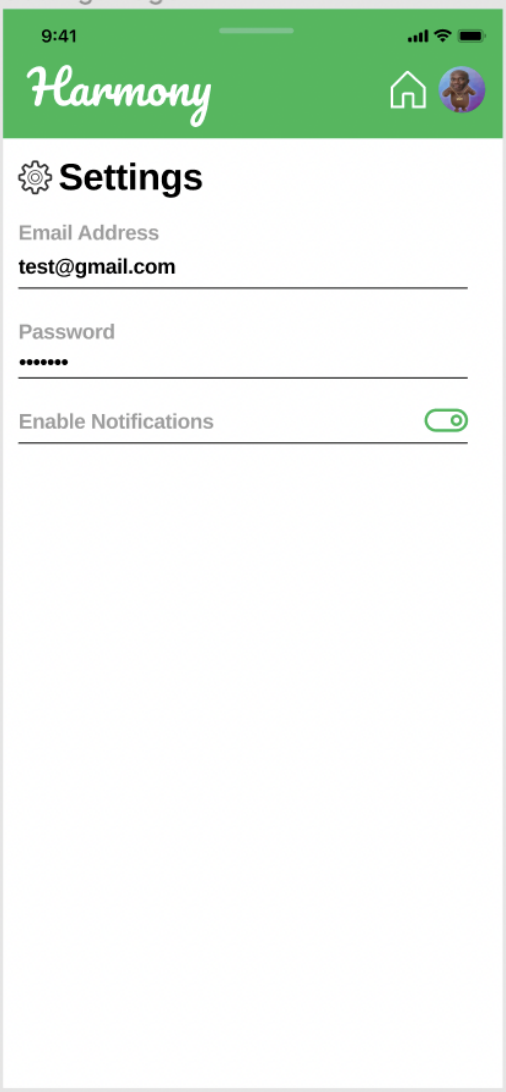
Profile Page



Friend Page



Settings Page



Add Friends Page