| Spring 2022 | **Status Report 2** |  |
| --- | --- | --- |

*Music Matcher*

Group 2: Ray Huang, Amy Paul, Mario Verdugo, Amir Yazdani

1. Introduction

For this iteration, the desired overall customer experience is to be able to use the app functionally. That is, the customer should be able to link their music accounts and chat with nearby users that they want to match with. As a team, we accomplished mostly all we wanted to. The ability to link accounts has been finished, as has finding nearby users and chatting with them. We are also able to show the initial Spotify profile details. Although we have not yet done as much testing as we would perhaps like to since feature development has been slower than anticipated, we are in a very good position going forward, and we are happy with our progress.

We have remained faithful to the proposal in the major respects, so there is little to report in the way of changes. However, if Apple Music Android authentication is not able to function as we expect, we may need to discontinue that feature, which would be a major change. An update on that will come at the end of the next sprint.

2. Customer Value

For this iteration, the customer’s desired value is that of functionality. We wish for the customer to be able to start using our app as we intended. That is, the customer should be able to make connections with others using our UI and backend easily and message with those connections.

3. Problem Definition

Our customer problem remains the same as the last report: making connections has been difficult due to the COVID-19 pandemic. Our project seeks to address this problem by providing the ability to connect with people based on choice in music. The project will provide meaningful connections to people based on music preferences. Often connection apps can be superficial, but providing connections on music, which is often very meaningful to individuals, provides a new way to meet others.

4. System Description

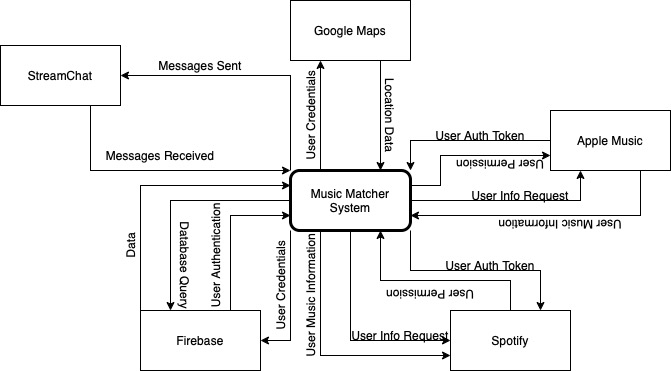
System Overview

Currently, the system is composed of a user interface which will let the user login/signup to the app, a database for managing backend operations (e.g., authentication), a location module, and a chat module. The system uses Firebase as a cloud server to keep/maintain user credentials for future authentication and to store users’ connections and music data. Firebase provides prebuilt methods to ease authentication and storage/retrieval. As for external interactions, Spotify and Apple Music have been integrated with to get users’ music data, StreamChat has been integrated for the chatting services, and Google Maps has been integrated with to get and display location data. The biggest challenges are integrating with these external interactions, as well as effectively displaying data to the user. We may change how the data is displayed based on user feedback as well as feature implementation.

Design Rationale

So far, our design lets the users signup/login to the app using their own usernames and profiles which is crucial for accessibility and authentication of Music-Matcher app. Our design also encapsulates a profile creation feature where users can include their personal information. These profiles are saved in our database and shown later on to potentially well-matched profiles. To let matched-profiles communicate with one another, we include a chat feature in our app to let users communicate through this interface. Moreover, we will use google location based services by Google Maps Flutter plugin to save the latest geolocation information of users on our database which will later serve as a filtering mechanism to select potential profiles to show to a certain user according to the distance range within which the user is looking for the potential match. The sole purpose of the app is to connect users based on their music taste. So, the users are assumed to use music streaming services such as Spotify and Apple Music. We merge the former services into our design by using their APIs. Users authenticate themselves using the former APIs, and later on profile and music related data are saved on our database to serve as filtering features.

Context Diagram



External Interactions

* Spotify: The Spotify API is used to authenticate users and collect their access token to further gather Spotify music/profile related information. After authenticating users using Spotify API, the system fetches profile details, such as display account name, email address, country, etc., from Spotify.
* Apple Music: The Apple Music API is used to authenticate users and collect their access token to further gather Apple Music music/profile related information. After authenticating users using the Apple Music API, the system fetches profile details, such as display account name, email address, country, etc., from Apple Music.
* Google Maps: The system is interacting with Google Maps through a Flutter plugin called Google Maps for Flutter. The system can add maps based on Google maps data, and the plugin can handle access to the Google Maps servers, map display, and response to user gestures such as clicks and drags. The system can also add markers to your map. These objects provide additional information for map locations, and allow the user to interact with the map.
* Firebase: The system interacts with an external Firebase database for data storage and retrieval as well as authentication protocols.
* StreamChat: The system interacts with StreamChat to provide users the ability to message with their connections.

Architectural Views

*Logical View*

Primary Diagram:

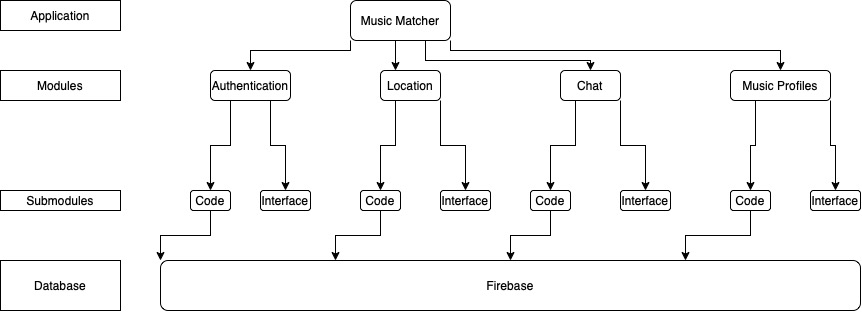
The view shows the logic of the system; that is, the interface sits on top of the application logic and external service integrations.

Element Catalog:

1. Mobile Interface – Provides the user experience for the app for all customers. Includes user-interactable elements such as buttons and forms. Connects to the Presentation Layer of the application by providing a visual representation of the code.
   1. Owner: Mario
2. Presentation Layer – Provides the code for the user experience. Written in Dart and uses widgets to create UI elements. Connects to the Application Logic by updating itself based on data it receives from calls to that logic. Connects to the mobile interface by providing the necessary code to display UI elements.
   1. Owner: Amy
3. Application Logic – Provides the logic to run the app and alter the presentation layer. Contains things like logic for API calls and data storage. Connects to the Presentation Layer by providing needed data and acting as a pipeline from the External Services to the Presentation Layer. Connects to the External Services layer through API calls and HTTP requests, as well as database queries.
   1. Owner: Amir
4. External Services – Provides the services part of the application. Allows for data to be stored and retrieved from outside the application system itself, and allows for more modular code since not all of the data comes from within the system. Connects to the Application Logic through API calls made by that logic.
   1. Team Owner: Amy
5. Firebase – Provides authentication services and data storage/retrieval. Connects to the Application Logic layer through queries made by the application, as well as calls using Firebase authentication functions, which also use the external service.
   1. Team Owner: Amy
6. Google Maps – Provides location services. Connects to the Application Logic layer through API calls.
   1. Team Owner: Ray
7. StreamChat – Provides messaging services. Connects to the Application Logic layer through API calls.
   1. Team Owner: Mario
8. Spotify – Provides music data. Used to get information about the user to match them with potential connections. Connects to the Application Logic layer through API calls.
   1. Team Owner: Amir
9. Apple Music – Provides music data. Used to get information about the user to match them with potential connections. Connects to the Application Logic layer through API calls.
   1. Team Owner: Amy

*Development View*

Primary Diagram:

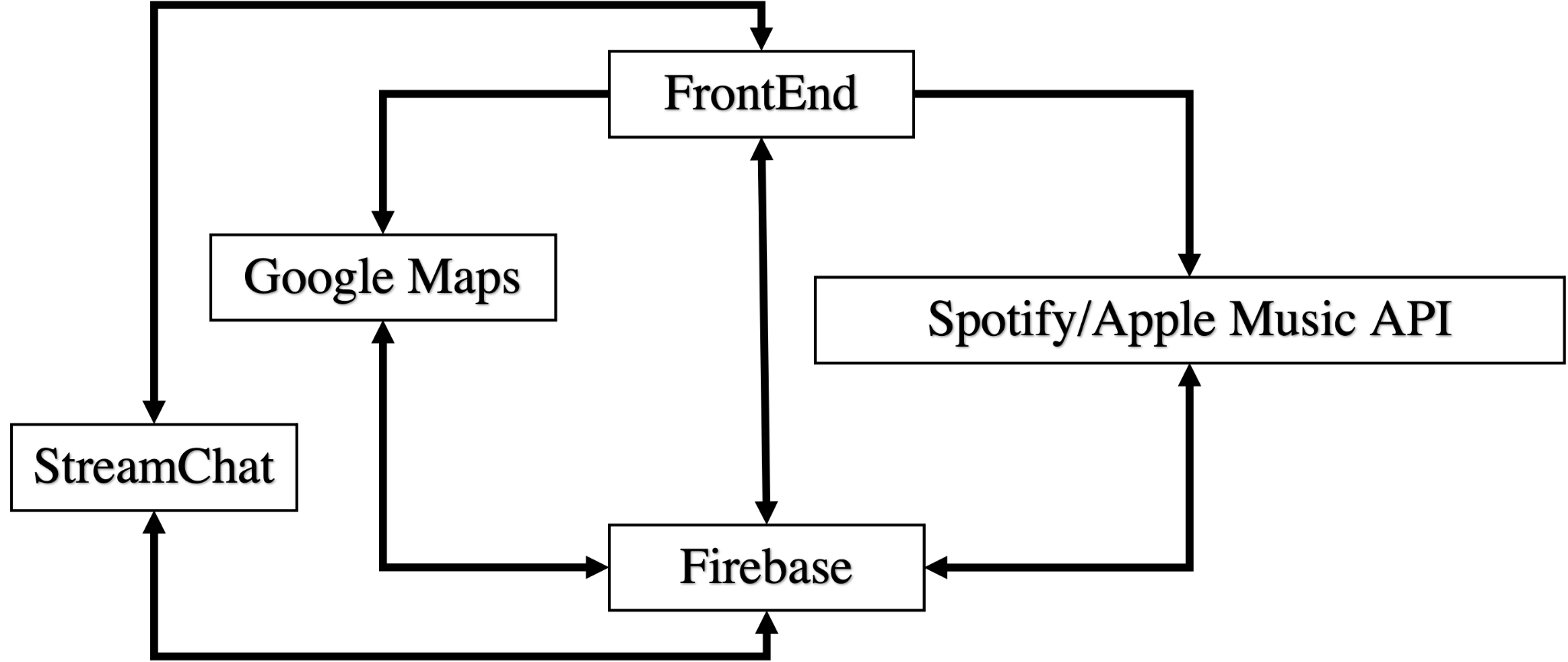
**

The view shows how the application connects to and communicates with modules and submodules, as well as the database layer. Each module is shown separately and split into its submodules, and each interacts with the database layer, as well.

Element Catalog:

1. Music Matcher – The entire application, not yet split into its components. Each module comes together to complete the application.Owner: Team
   1. Authentication – The Authentication module includes the sign-in flow for the app, the ability to sign up, and the ability to create profiles as well as the logic for the same. Owner: Amy
      1. Code – The Application Logic and Presentation Layer for the Authentication module.
      2. Interface – The Mobile Interface for the Authentication module.
   2. Location – The Location module includes the service connections for getting users’ locations and displaying nearby users. It includes some external service calls. Owner: Ray
      1. Code – The Application Logic and Presentation Layer for the Location module.
      2. Interface – The Mobile Interface for the Location module.
   3. Chat – The Chat module includes the messaging flow, allowing users to chat with connections. It includes some external service calls. Owner: Ray
      1. Code – The Application Logic and Presentation Layer for the Chat module.
      2. Interface – The Mobile Interface for the Chat module.
   4. Music Profiles – The Music Profiles module includes the flow to get data from users’ connected music services, as well as flows to authenticate with those services. It includes some external service calls. Owner: Amir
      1. Code – The Application Logic and Presentation Layer for the Music Profiles module.
      2. Interface – The Mobile Interface for the Music Profiles module.
2. Firebase – The database backend for the application. Includes secure authentication flows and storage as well as basic data storage and retrieval. Is queried by the different modules to provide its data and authentication services.
   1. Owner: Amy

*Dynamic View*



The view shows how the front-end of the application connects to external services and databases.

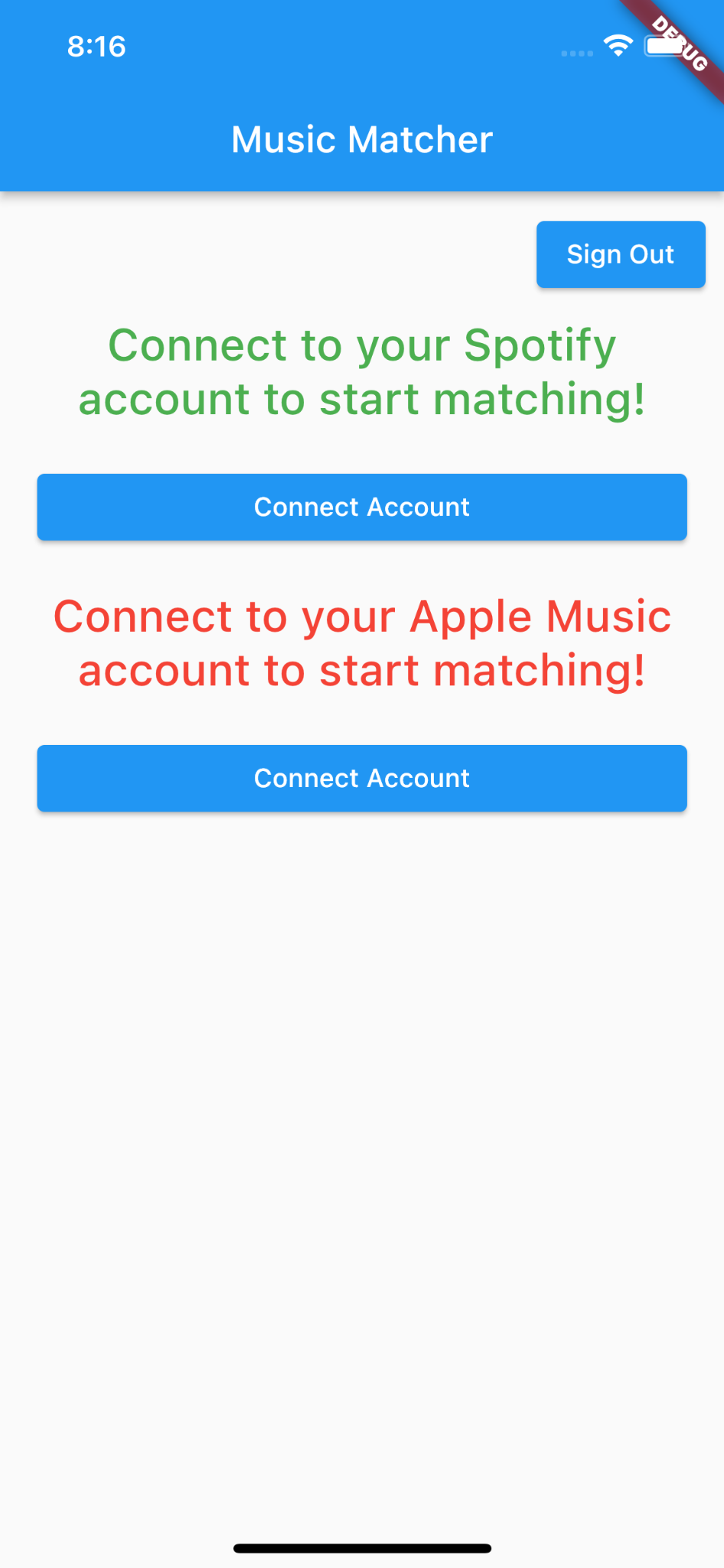
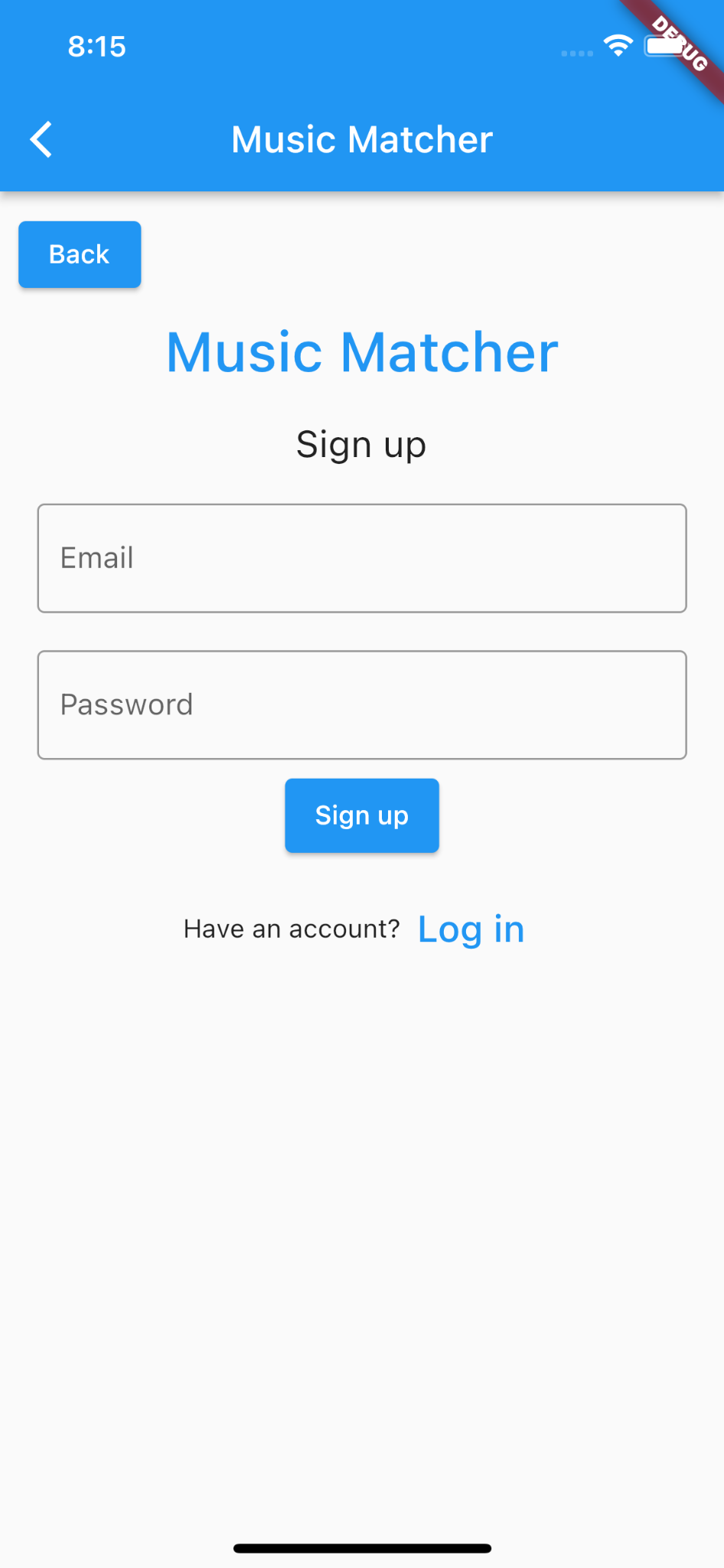
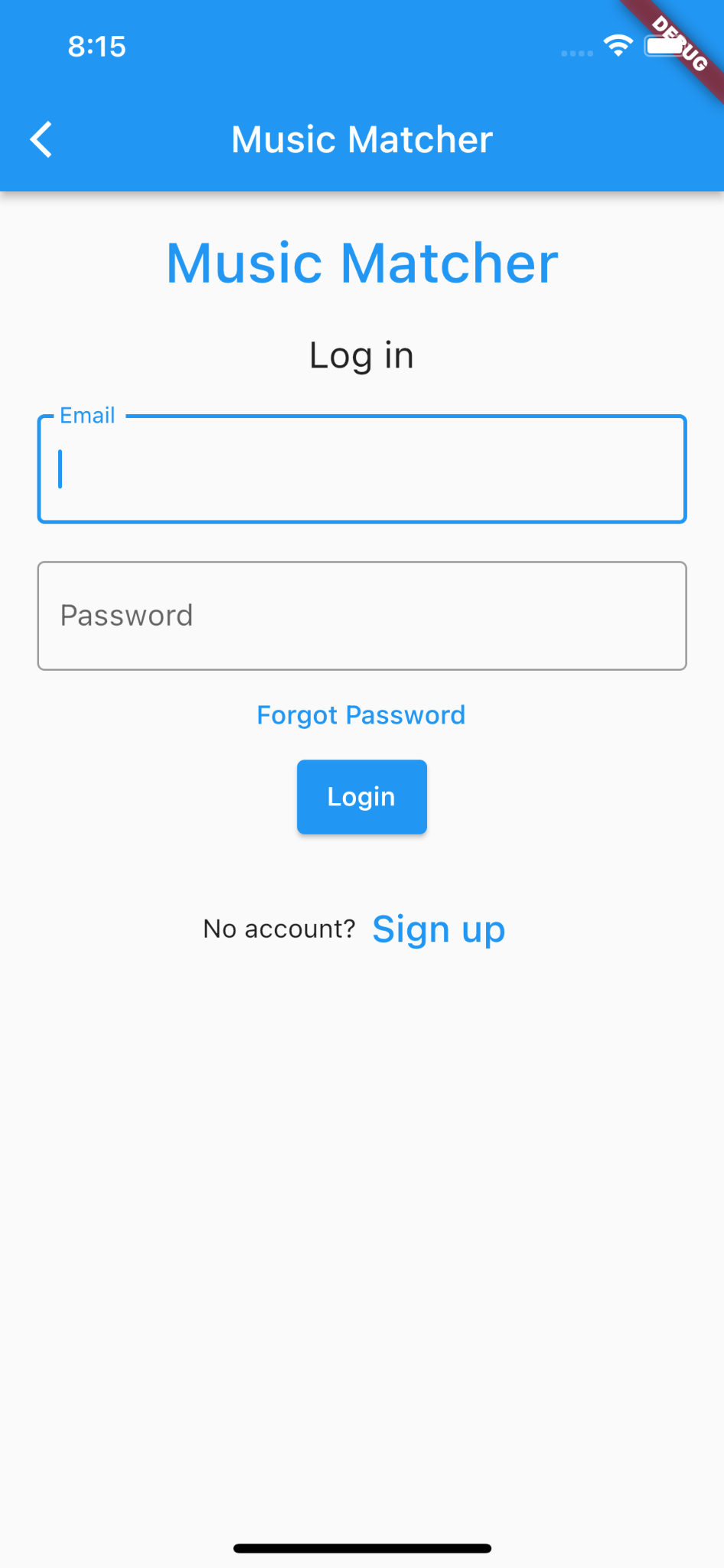
Element catalog:

1. Front-end: Is in charge of initiating requests to (e.g., Firebase for authentication, Stream chat for sending messages) and receiving services from external services (e.g., receiving chat messages, receiving potential profiles from Firebase, etc.). It also is in charge of maintaining the reliability of the software running on user handsets.
   1. Owner: Mario
2. Firebase: Receives authentication requests from FrontEnd. And exchanges control messages with StreamChat, Google Maps, FrontEnd and Spotify/Apple Music API. Our cloud server that saves user profiles and applies filtering logic to suggest potential matches to FrontEnds.
   1. Owner: Amy
3. Spotify/Apple Music API: Receives Spotify authentication requests from front end data fetching requests from Firebase.
   1. Owner: Amir
4. Google Maps: Communicates with FrontEnd and Firebase to provide location specific data for profile filtering and map visualization.
   1. Owner: Ray
5. StreamChat: Provides chat service to users. To do the former the module communicates with FrontEnds and Firebase.
   1. Owner: Mario

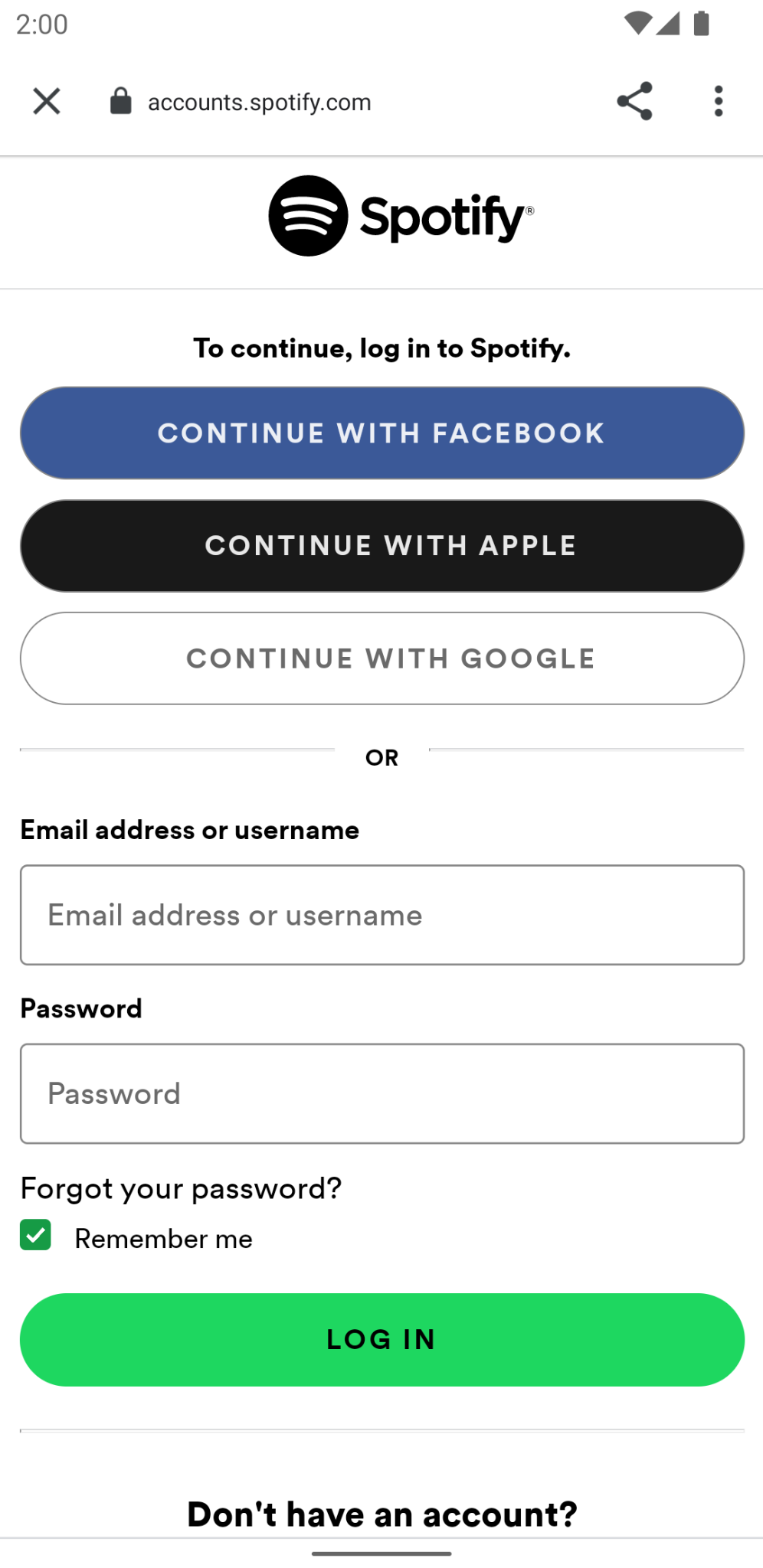
5. Current Status

Working features:

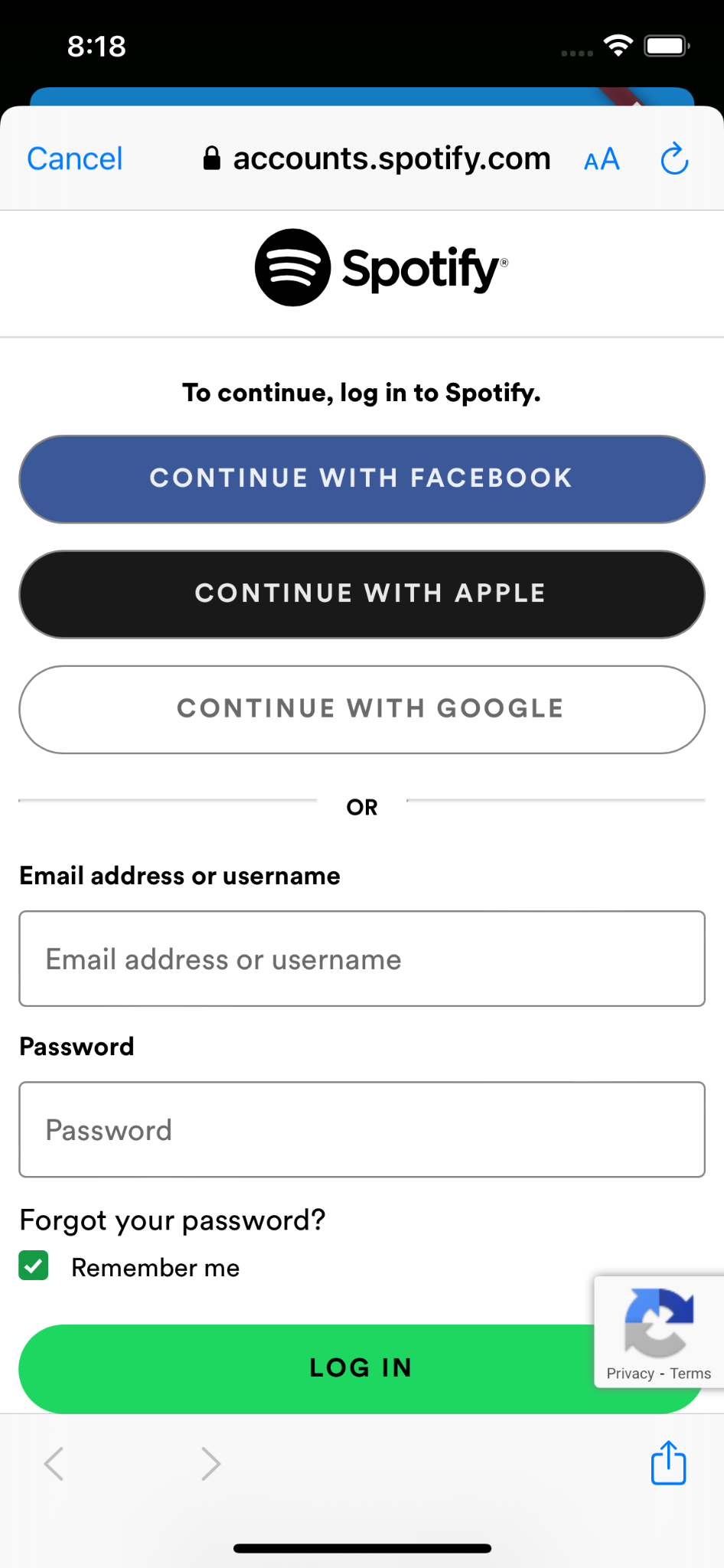
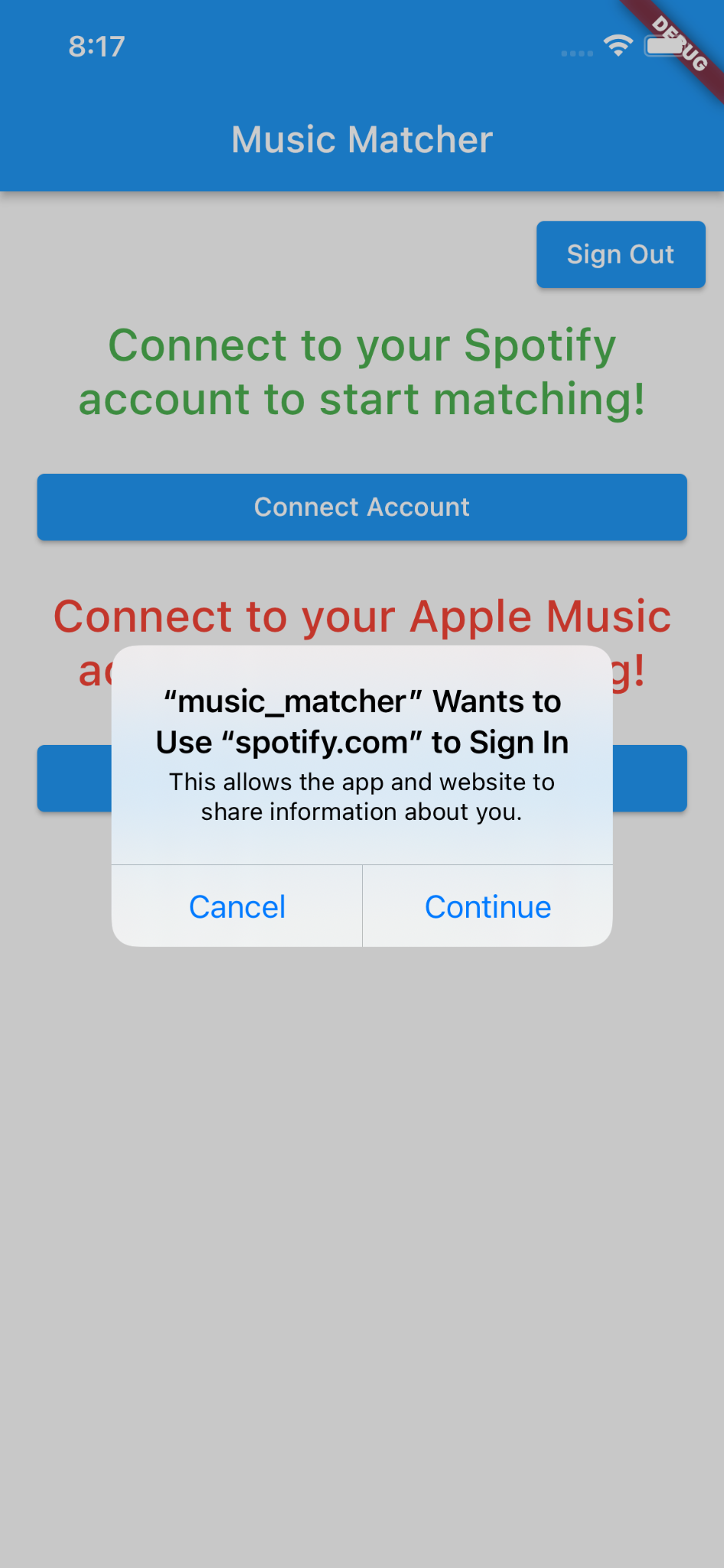
1. App authentication (Logging in and out, Signing up, Recovering Password)



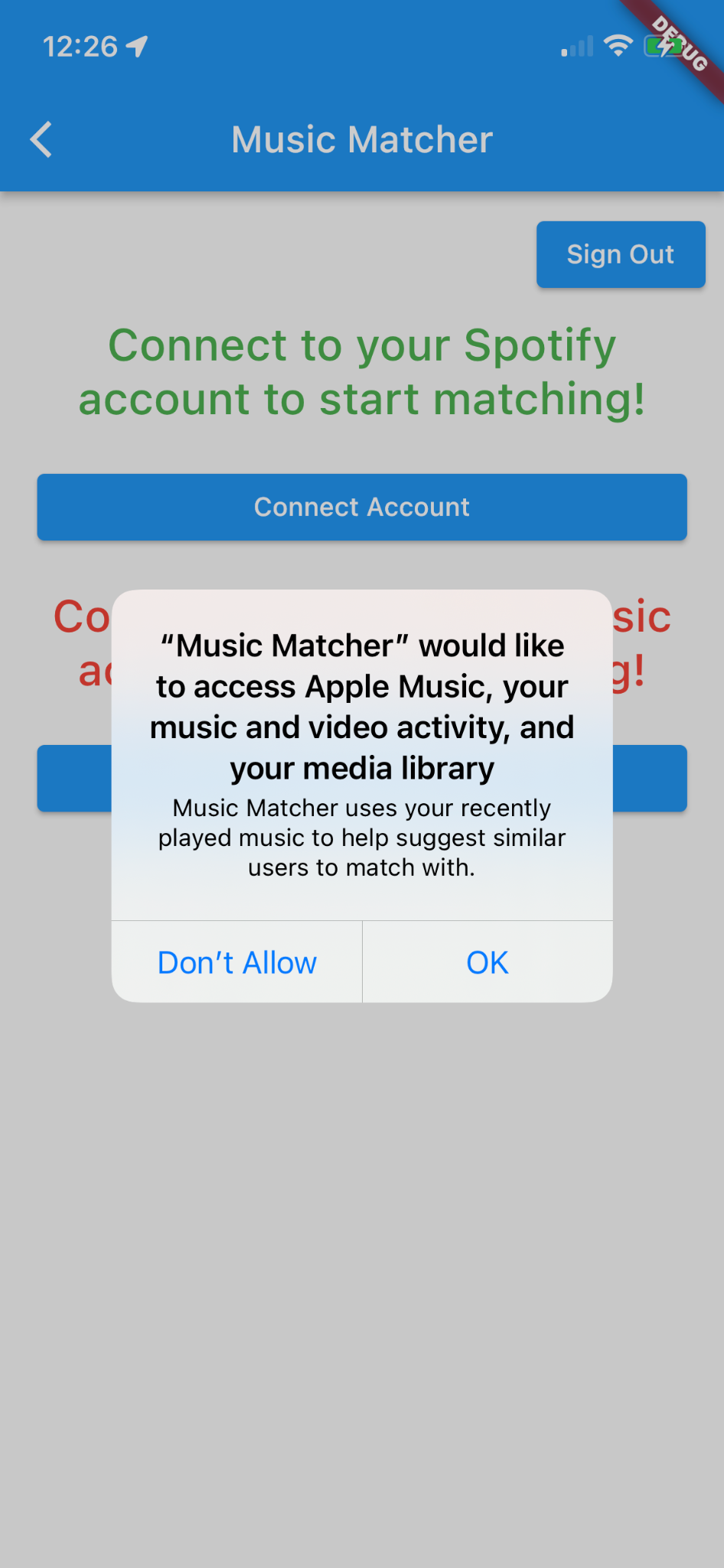
1. Music Account Linking
   1. Spotify (Android)

****

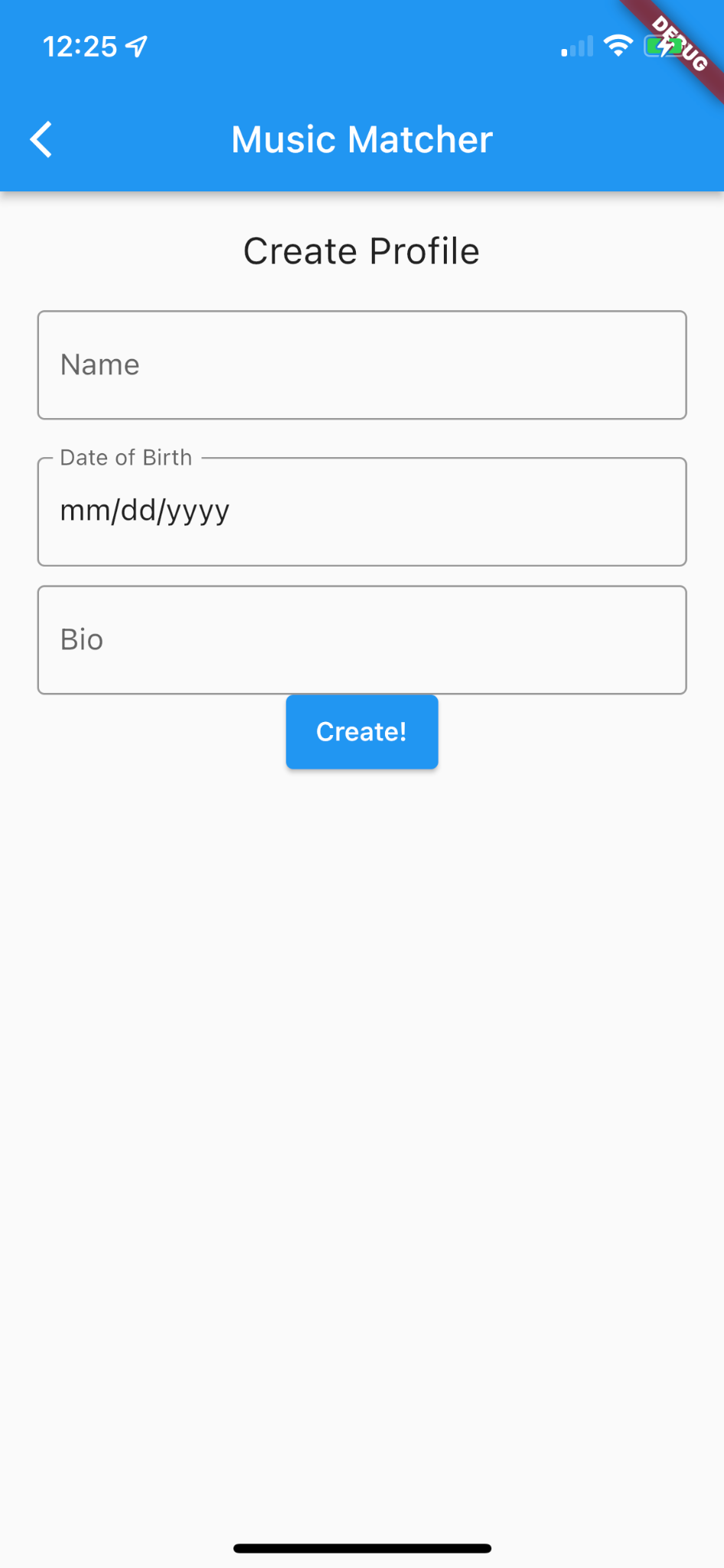
* 1. Spotify (iOS)



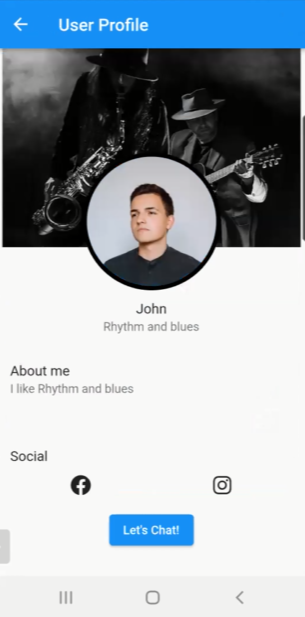
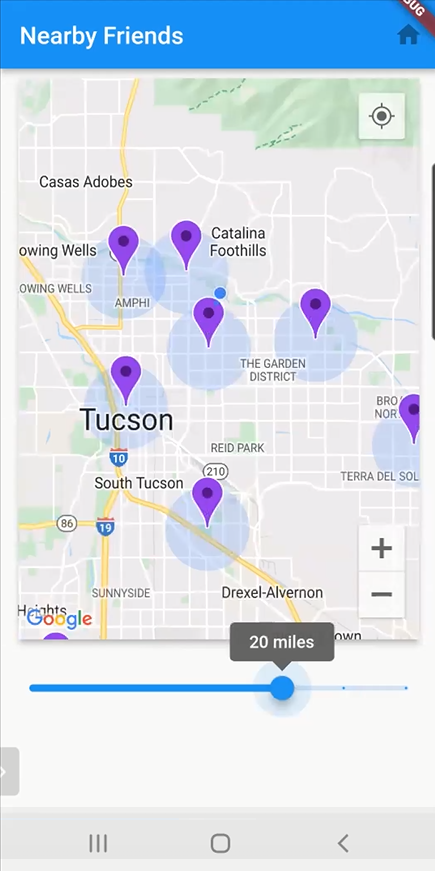
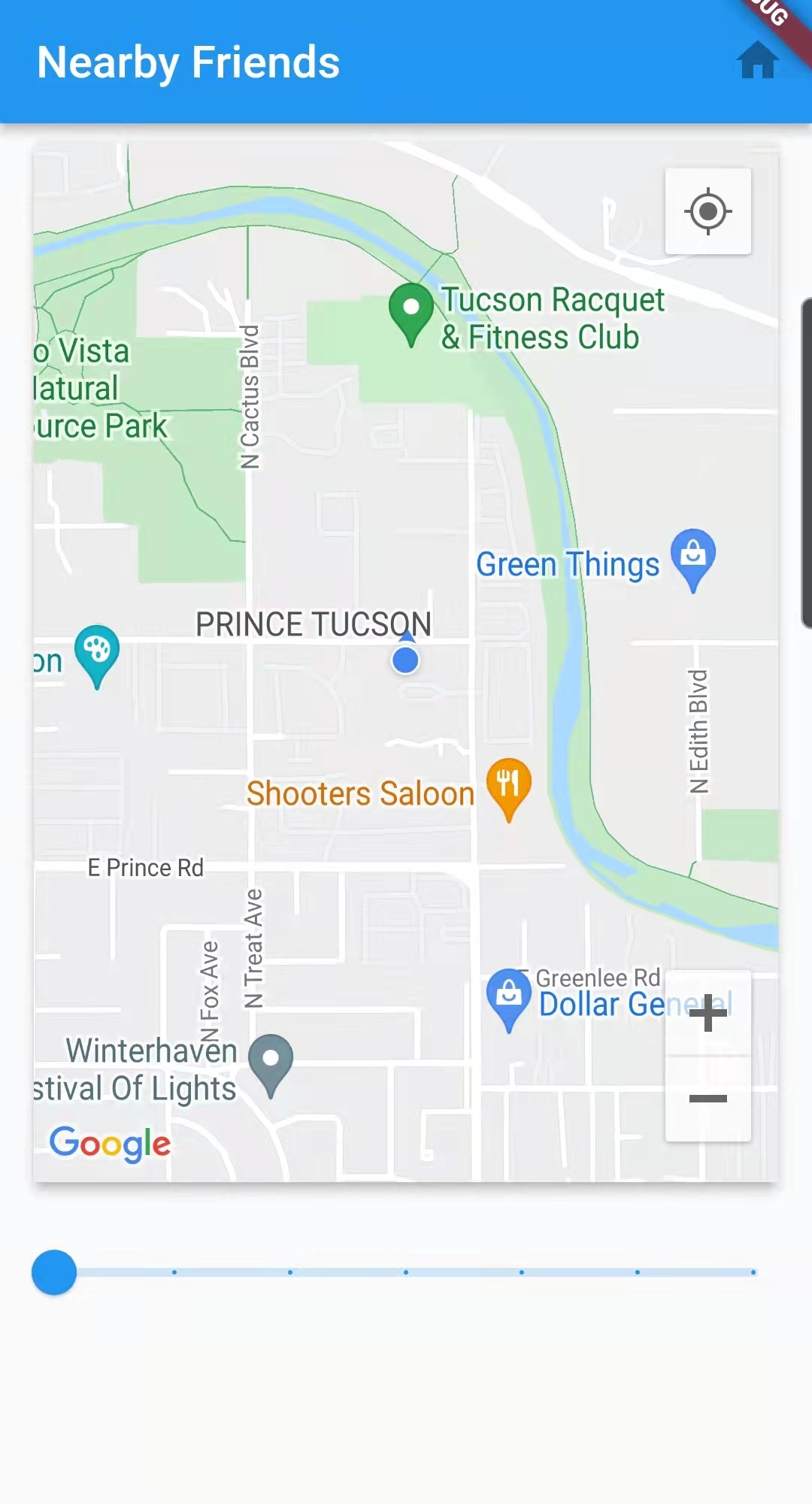
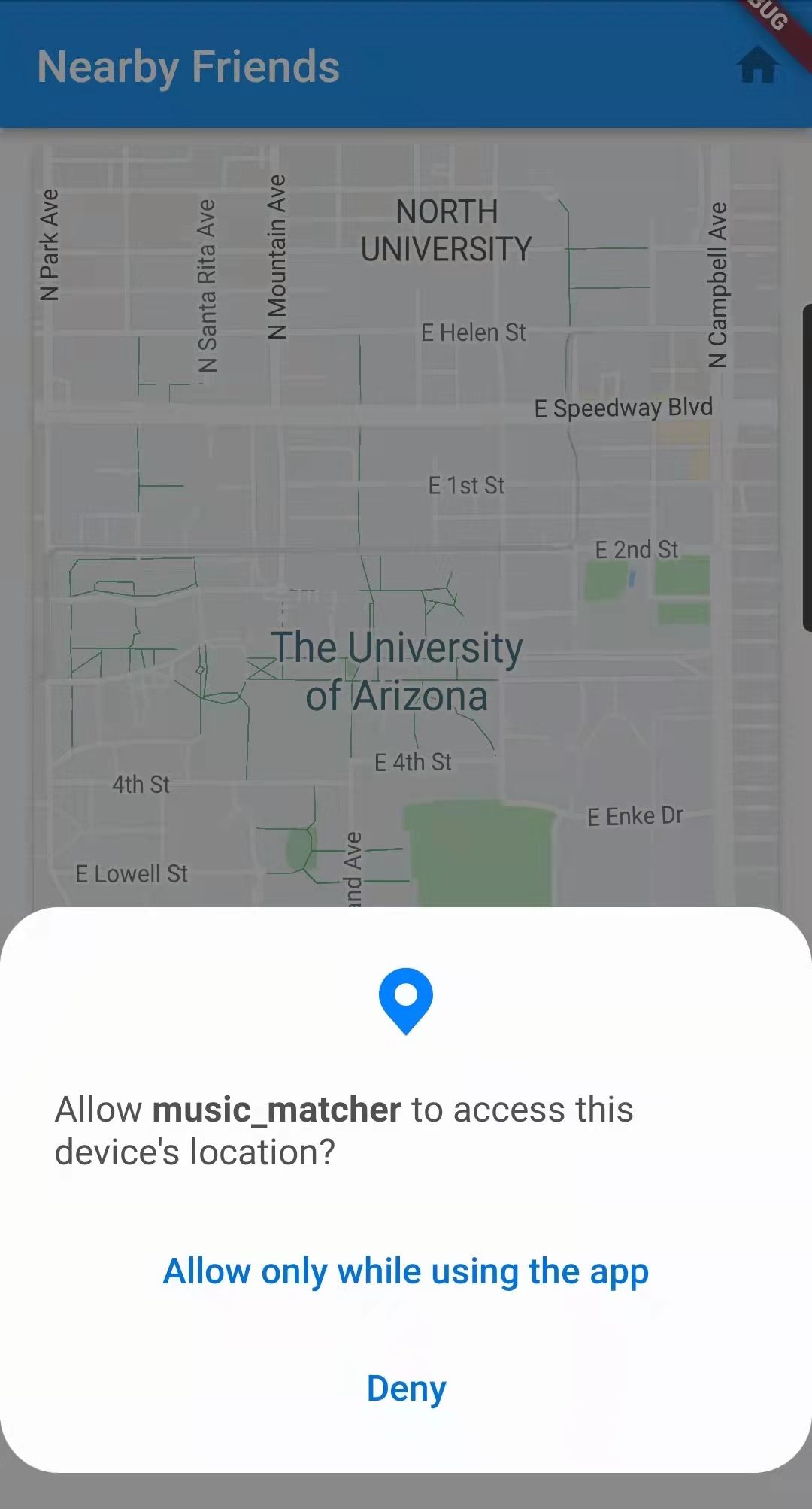
* 1. Apple Music (iOS)



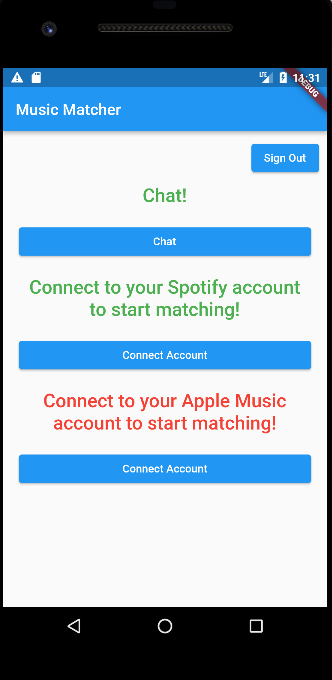
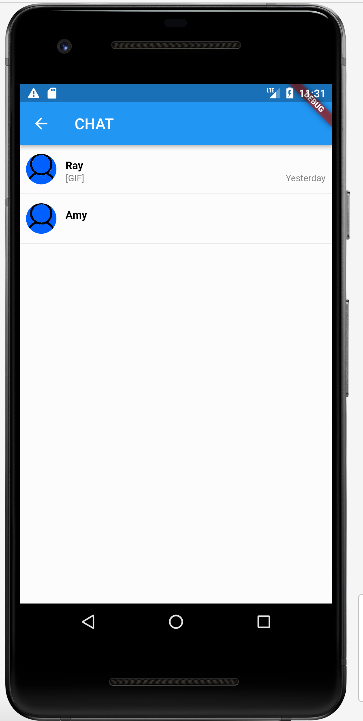
1. Profile Creation

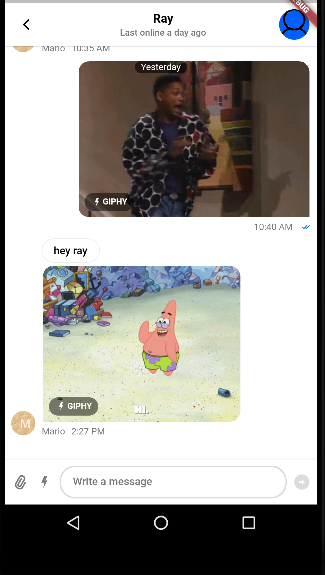
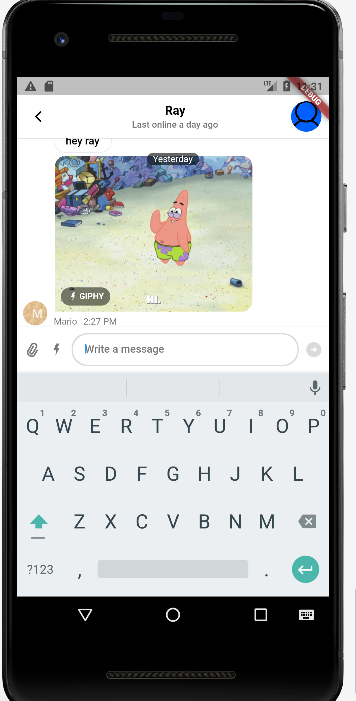


1. Seeing Nearby Users (asking for permission, my location, see nearby users by adjusting distance slider, see user’s profile)

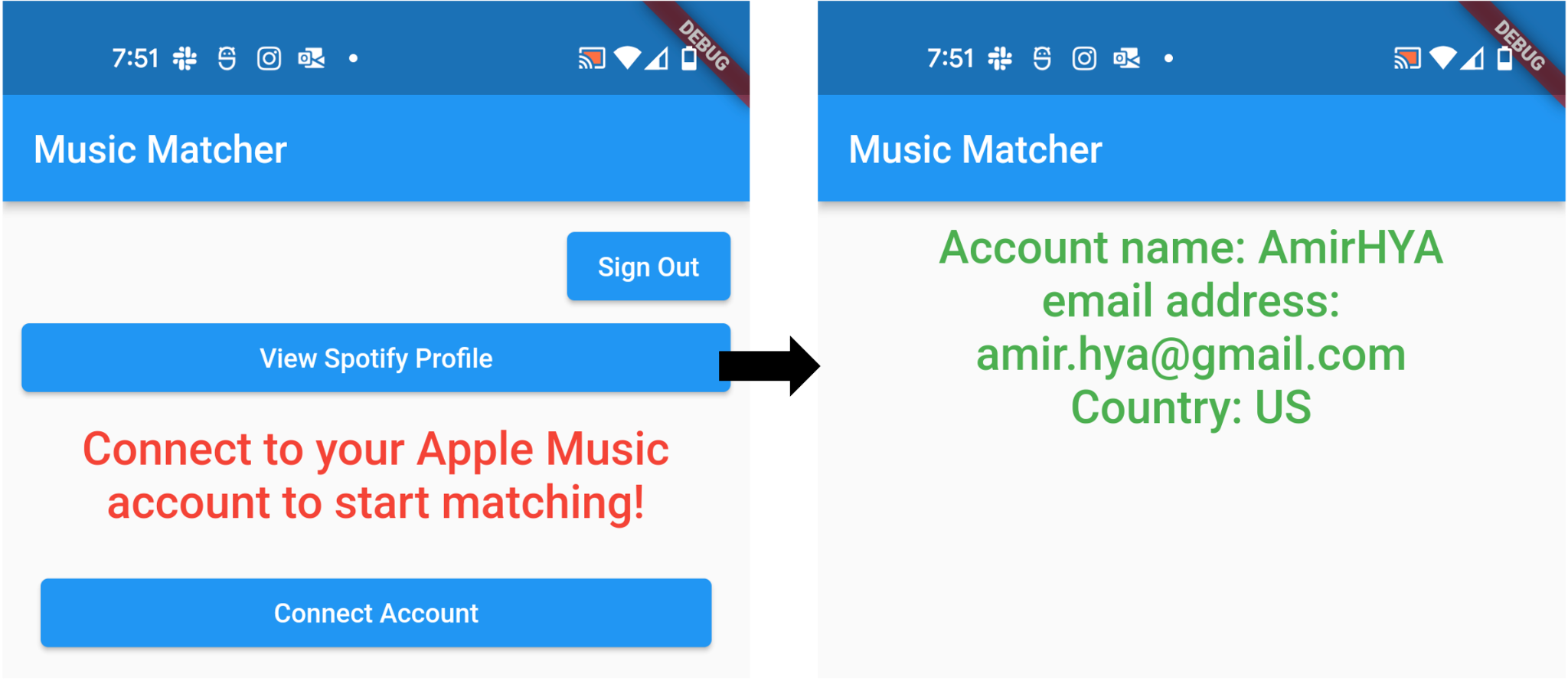


1. Chatting with Users

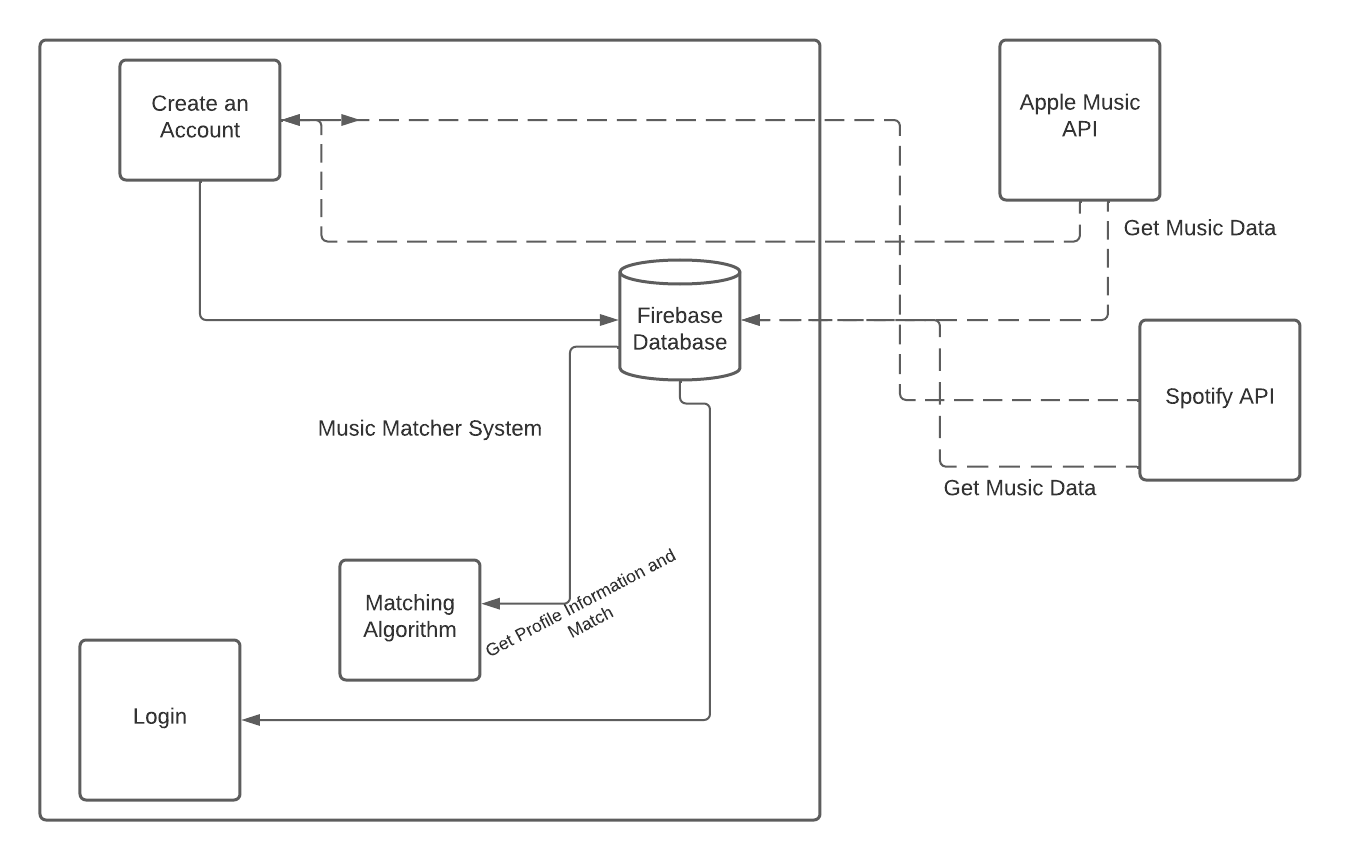
 →

→

1. Showing Spotify user profile details (Android)



Block Diagram:



The block diagram above shows our proposed final system. As of now, all parts and integrations have been completed aside from the matching algorithm.

We have many fewer challenges than the last sprint, admittedly. The systems have been working properly and development has gone much more smoothly. However, there has been a challenge in getting the Apple Music authentication to work as intended on Android devices. We believe we have found a workable solution, but as of now this remains a problem. We have manually tested each feature and had outside users test the functionality. More unit and integration tests are planned as a major part of the next sprint. As of the end of the sprint, we have written approximately 2,000 lines of code, collectively.

6. Project Management

*Changelog link:*

<https://docs.google.com/spreadsheets/d/1H8TjAgGXaTTaaAIvyOuuYWwSE5_7cy0iwTG5hW02yTw/edit?usp=sharing>

Our next goal is integrating our code, as some of it was written separately and modularly in order to facilitate development. This is a critical goal that will be accomplished ASAP. Other goals for the next iteration include much more testing, getting and storing more data from user music accounts, and allowing users to match with each other along with miscellaneous changes that will be reflected in the changelog as they occur. The plan for the rest of the semester is to finish up the features as soon as possible, write as many unit and integration tests as necessary, and have more external users (friends and family, as well as potentially others) use our app and give us feedback.

7. Team

*Team roles:*

Ray - Scrum Master

Amy - Product Manager

*Contributions:*

Amy: Spotify and Apple Music authentication, creation of user app profiles.

Ray: Location feature to allow users to see nearby connections.

Amir: Creation of Spotify user profiles.

Mario: Chat feature to allow users to message nearby connections.

*Percentage contributions and constraints:*

Amy: 25%

Mario: 25%

Ray: 25%

Amir: 25%

8. Reflection

*Things that have gone well:*

1. No one was blocked due to the android emulator.
2. Sprint tasks were assigned and are either completed or very close to completion.

*Things that have not gone well:*

1. Hours allocated during sprint planning was hard to estimate due to the team’s lack of experience with the platform/api services.
2. Integration of every individual’s modules.
3. Apple Music authentication for Android.

*Issues and how to overcome them:*

The team members put a lot of effort into making every individual function modularized and making sure it runs smoothly before putting them together, but underestimated the time needed for the work in integration. In addition, we are currently working on a solution to the issue of Apple Music authentication for Android. To overcome the issues for next iteration, we plan to start early and specify the module interface specification with more details.

*What we will do differently for next iteration:*

1. Allotting more time for app setup was necessary since many unaccounted for errors came up.
2. Team could have been more vocal about being blocked, so the scrum master could get involved.