GitHub Username: mikepalarz

JammyJamz

Description

This app is intended to allow users to share what current music they’re listening to with a newsfeed-style interface. It is meant to allow end users to express their current musical interests as well as be inspired by the musical interests of their friends.

Intended User

The app is intended for music enthusiasts or for anyone who is comfortable with sharing their musical interests. It is meant to capture those moments in a user’s day when they’re listening to a particular song and can’t help but think to themselves “Man, this song is awesome! I’ve **gotta** share it with my friends!”. Substitute “awesome” with “a jammy jam” if the song is totally off the hook.

The focus wouldn’t be strictly on individual songs/tracks. Users will also be able to share albums or artists if they feel it is more appropriate for their current musical mood. For example, an end user could express that they’re really enjoying *The Wall* by Pink Floyd as a whole, instead of a single track. Another instance could be that the user in fact simply enjoys listening to Pink Floyd instead of a particular song or album. The key concept is that the user can express their musical interests in multiple ways.

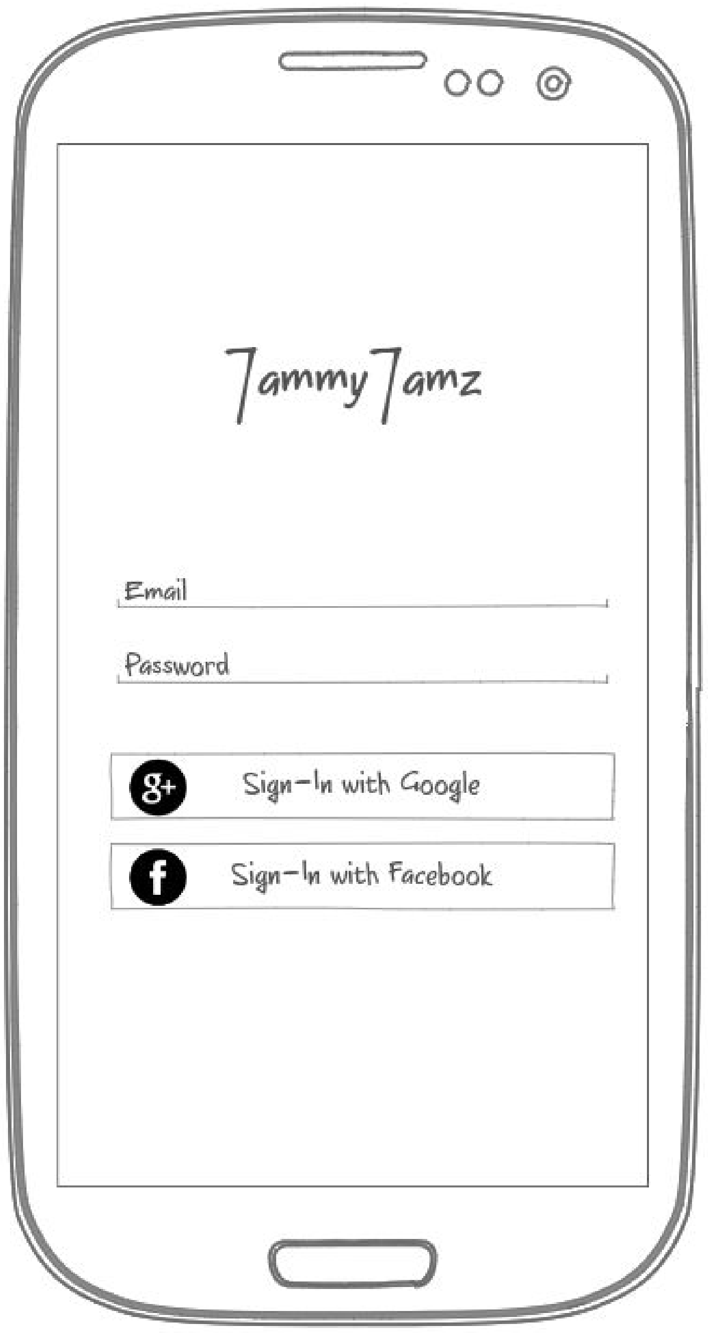
Features

Main features of the app:

- News feed interface which shows all registered users’ current musical interests (tracks, albums, and/or artists).
- Users can contribute to the news feed by search for tracks/albums/artists and creating a post.
- Previews of tracks/albums/artists.
    - If the user has the Spotify app installed, they’ll be taken directly to that particular track/album/artist within Spotify.
    - If they do not have Spotify installed, then they’ll be taken to the Spotify website for that track/album/artist that has 30s previews.
- Users can like posts from other users to express their interests of each other’s musical taste.
- Allows users to share music with each other
    - Users will receive a notification when another user shares a musical interest with them.

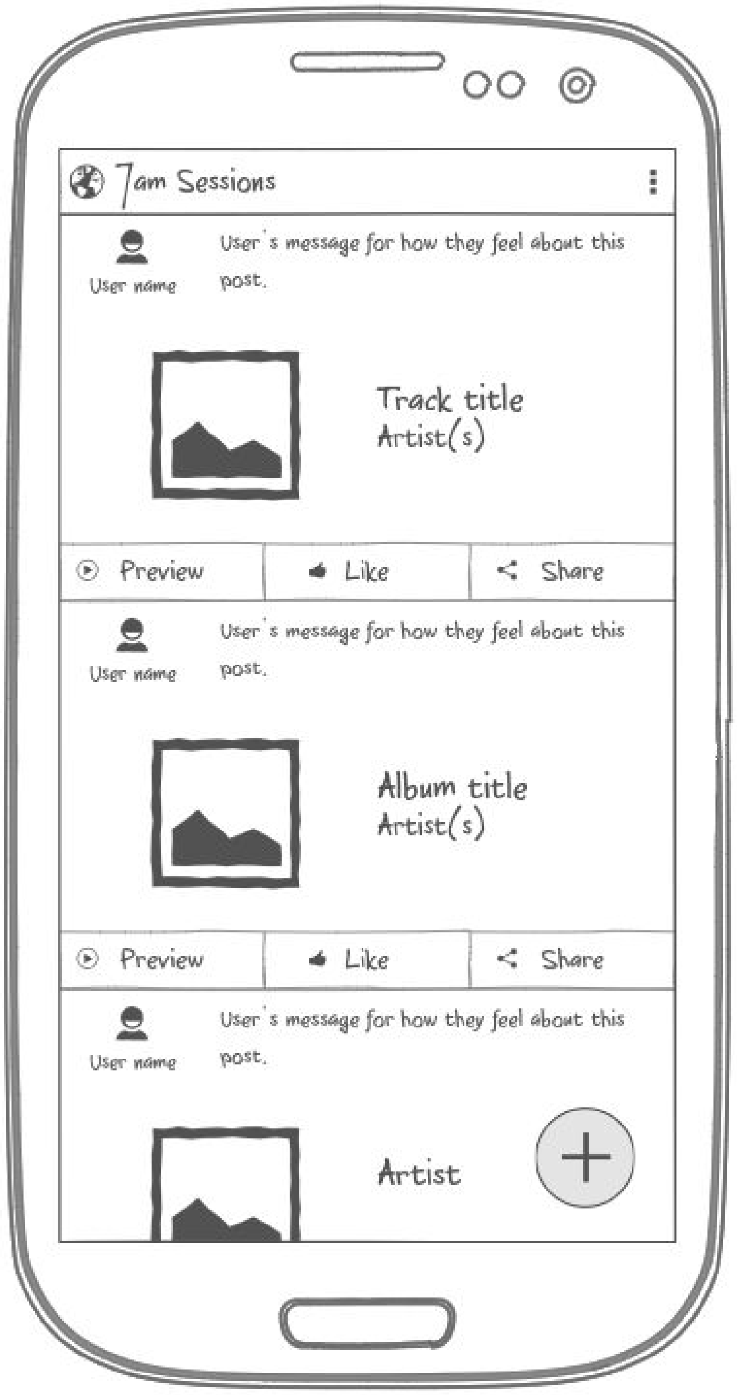
User Interface Mocks

Screen 1



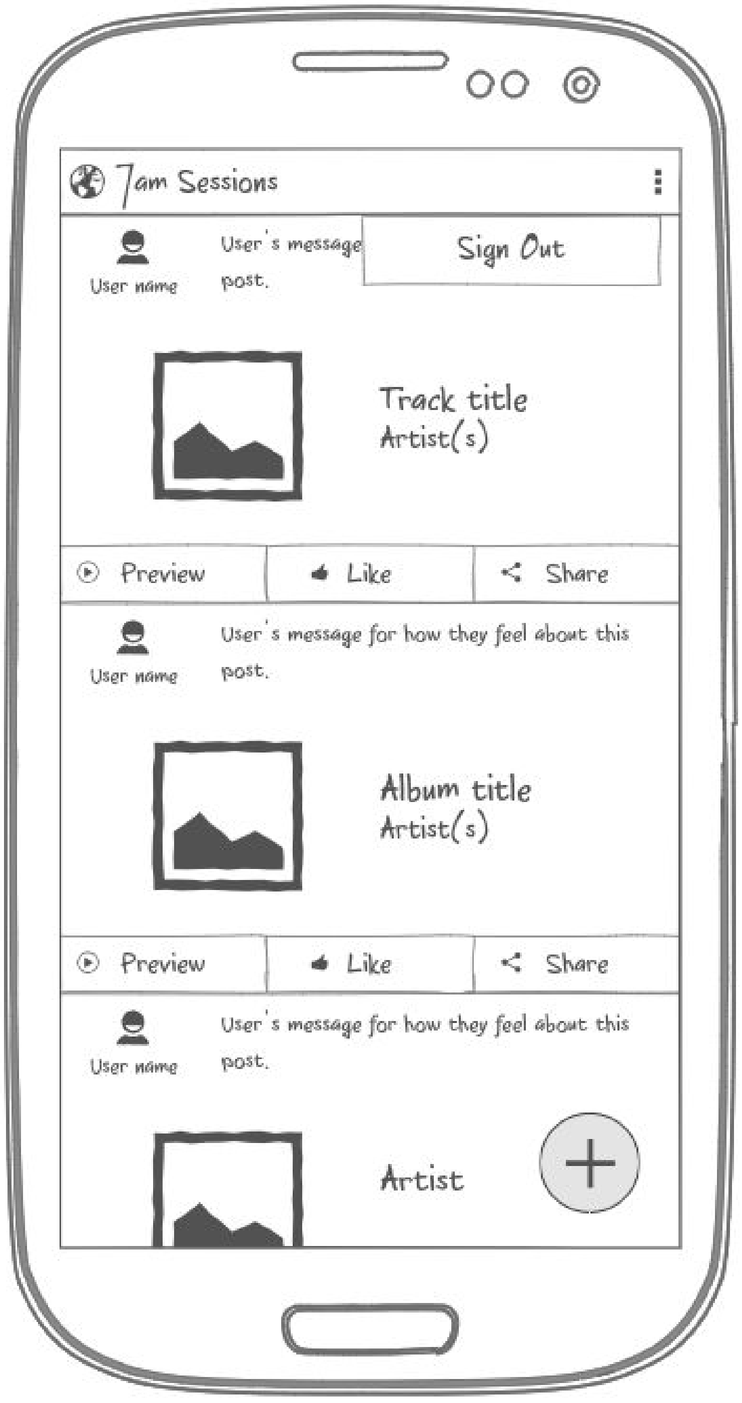
Sign-in screen for JammyJamz. The Google and Facebook sign-ins will be implemented by using FirebaseUI

Screen 2


Jam Sessions
User name
User's message for how they feel about this post.
Track title
Artist(s)
Preview
Like
Share
User name
User's message for how they feel about this post.
Album title
Artist(s)
Preview
Like
Share
User name
User's message for how they feel about this post.
Artist

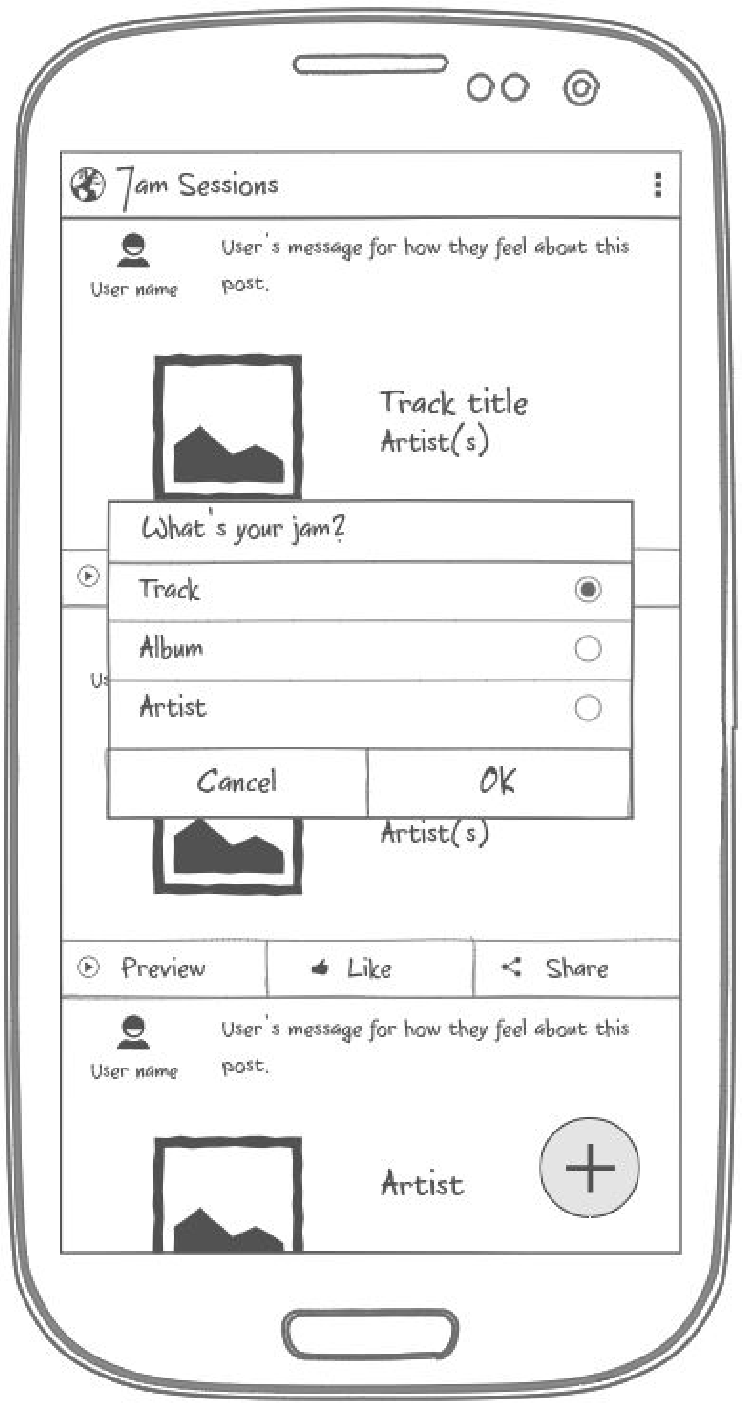
The news feed interface, which includes posts by all users. It also includes a FAB which allows the user to make a new post.

Screen 3



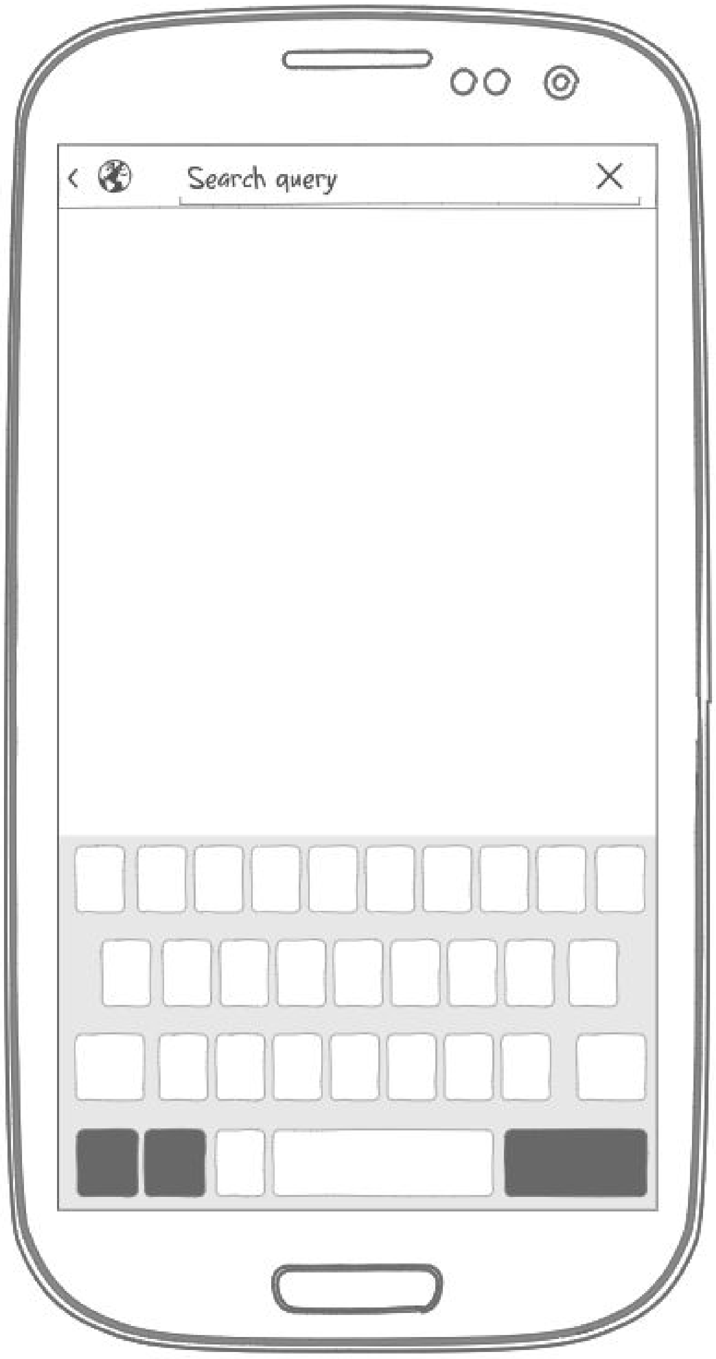
Under all circumstances (i.e., within all UI screens), the overflow menu will allow the user to sign out of JammyJamz.

Screen 4



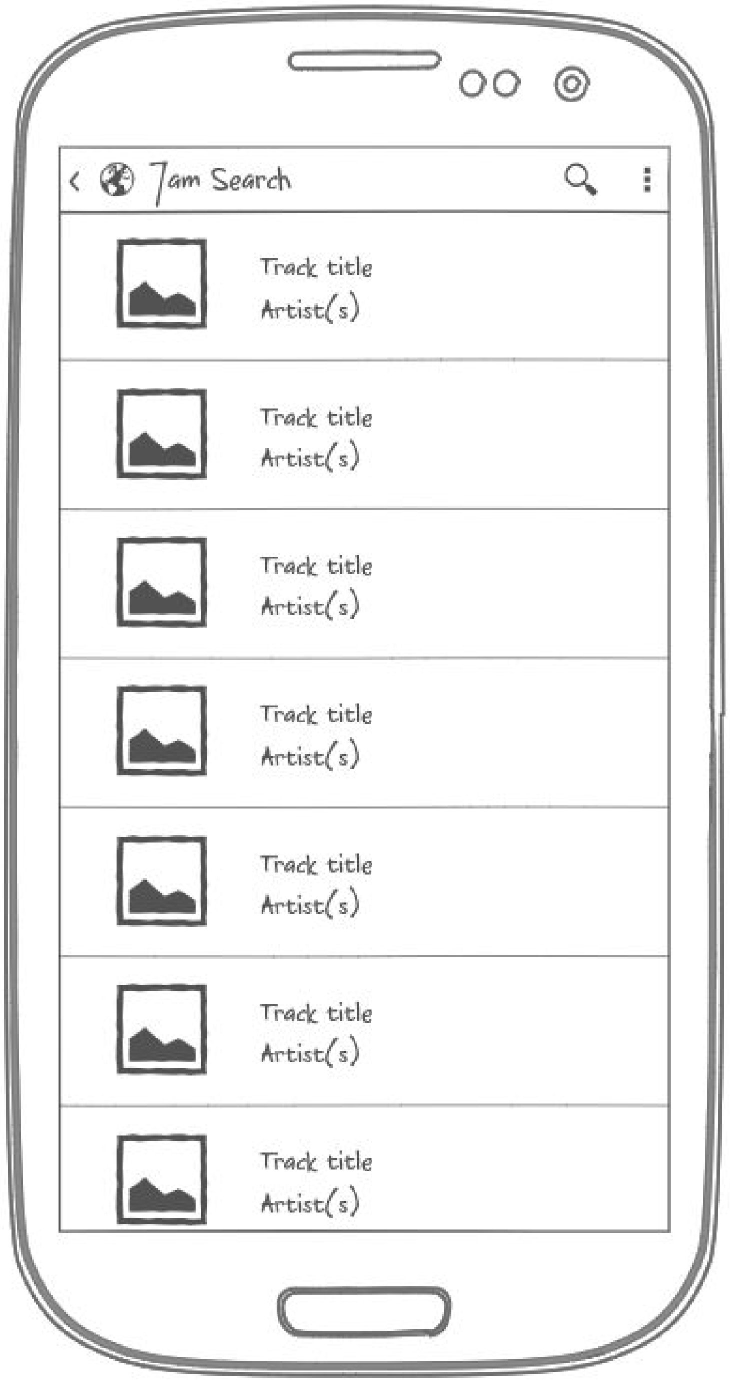
When the user clicks on the FAB, they will be presented with a dialog asking them what kind of post they'd like to make: track, album, or artist. If the user clicks "Cancel", then they will be brought back to the news feed. If they click "Ok", then they will be brought to the next screen.

Screen 5



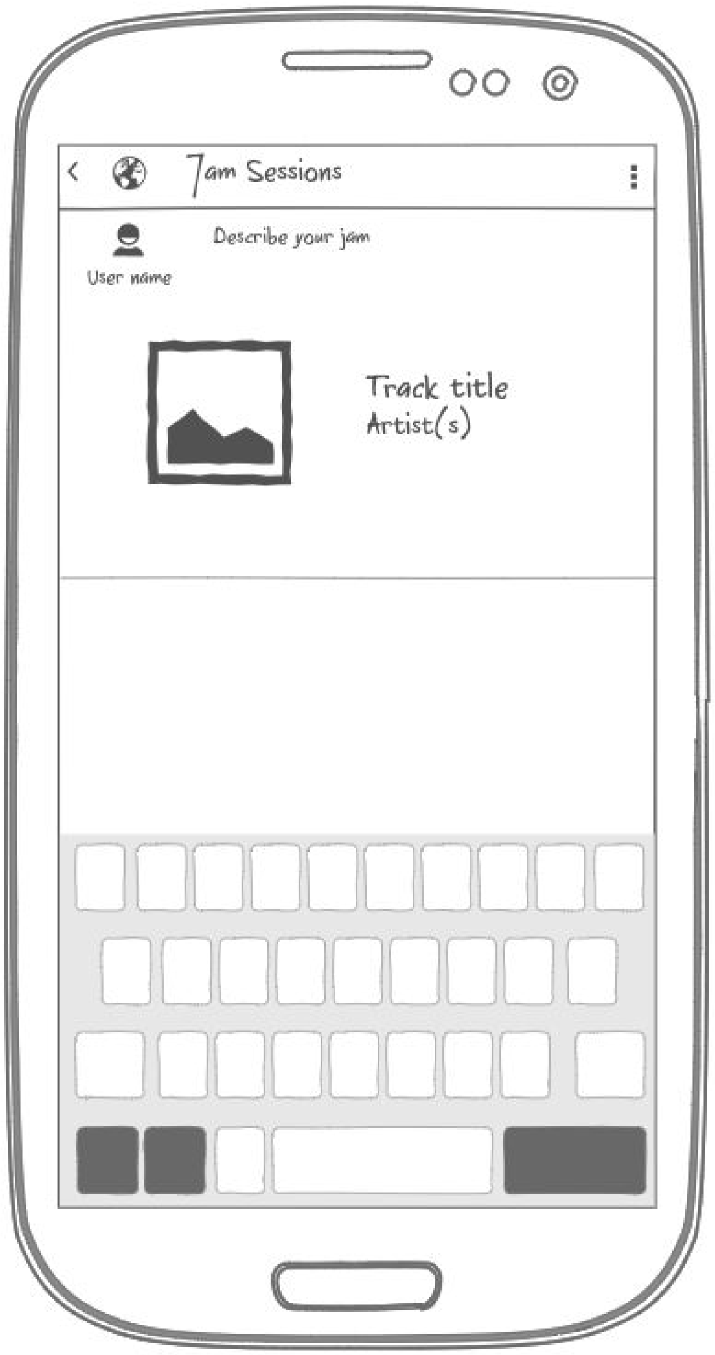
The user will then be prompted to search for the kind of material they are hoping to make a post about. Once the user hits “Enter” on the virtual keyboard, the query will be sent and they’ll be shown the next screen.

Screen 6



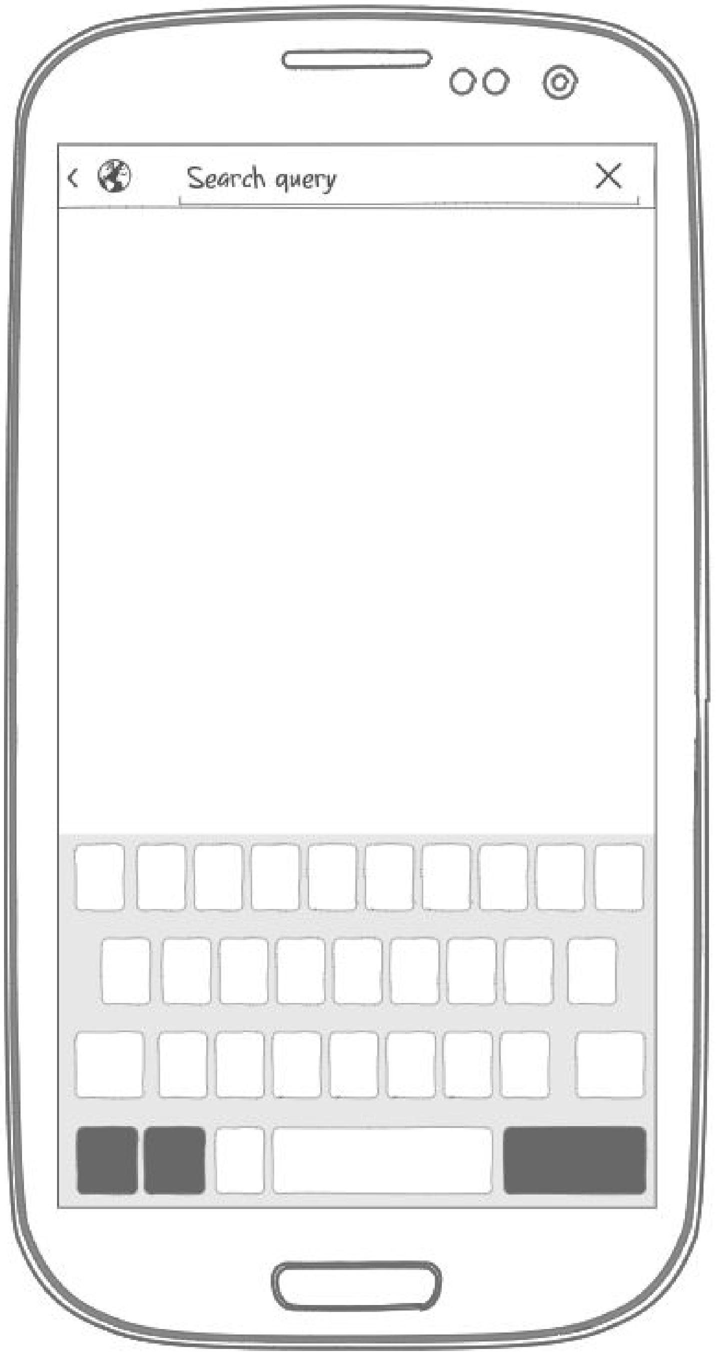
The user will be shown a list of search results from which they can choose to use for their post.

Screen 7



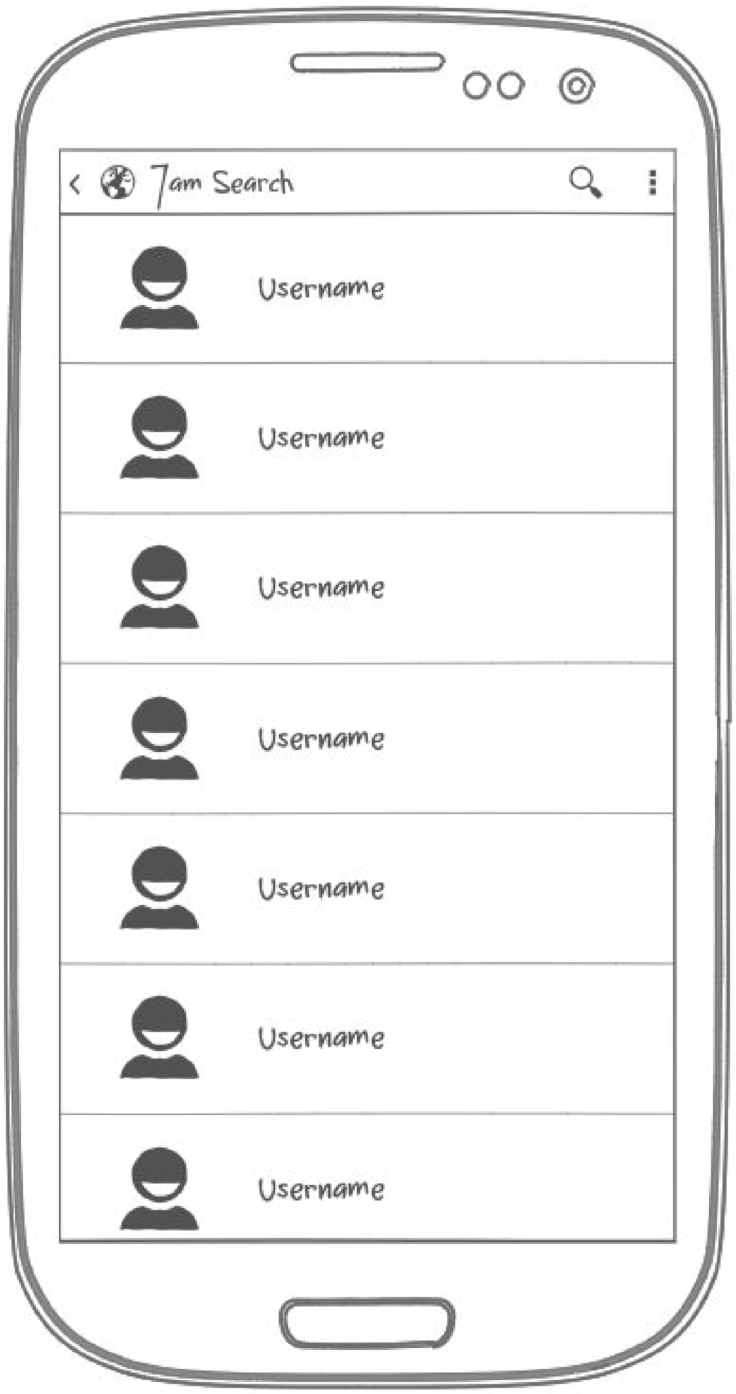
Finally, the user will be prompted to add an optional description of their post. This allows the user to further personalize how they feel for the given track/album/artist.

Screen 8



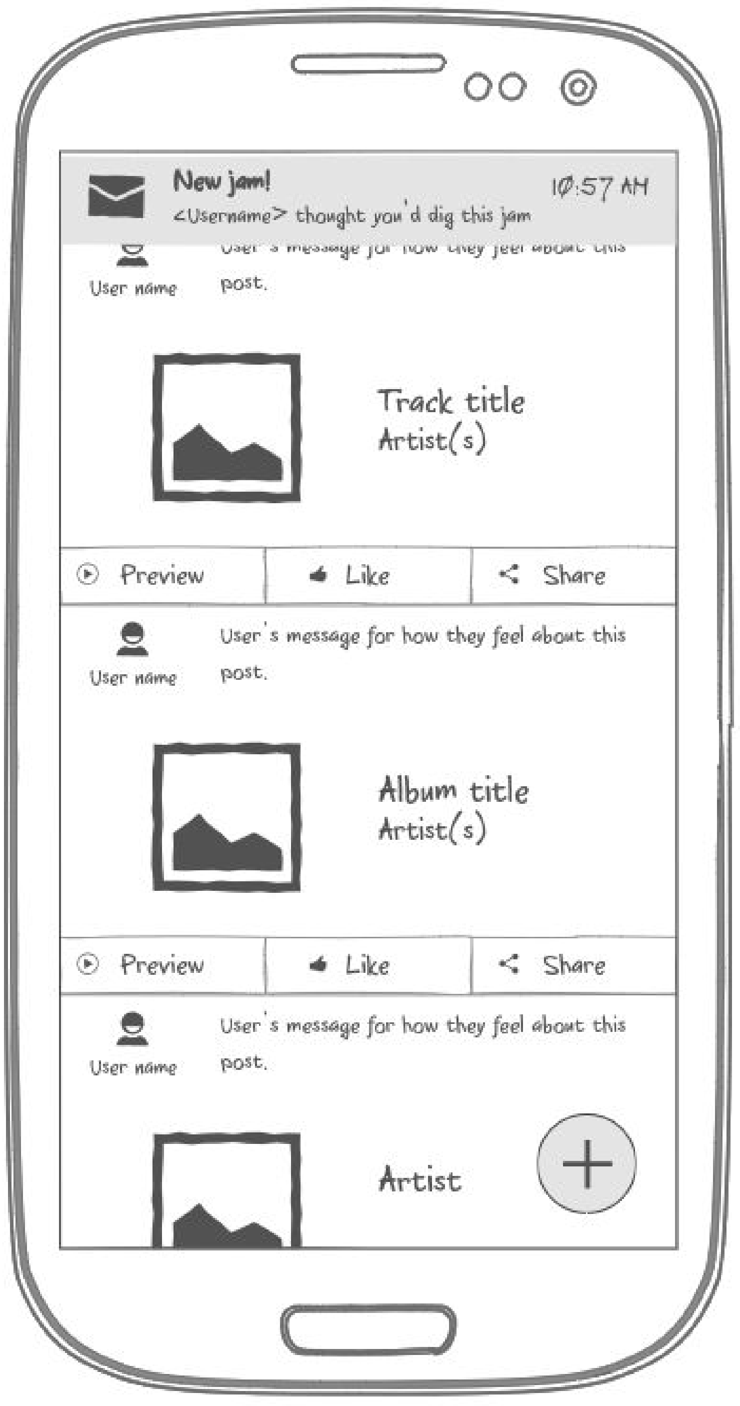
Users also have the option to share posts with other users. When a user clicks on the “Share” button for a particule post, they will first be shown a search UI, which allows them to search for the user they’d like to share the post with. The UI will be visually identical to search for a track/album/artist, but the query will be very different.

Screen 9



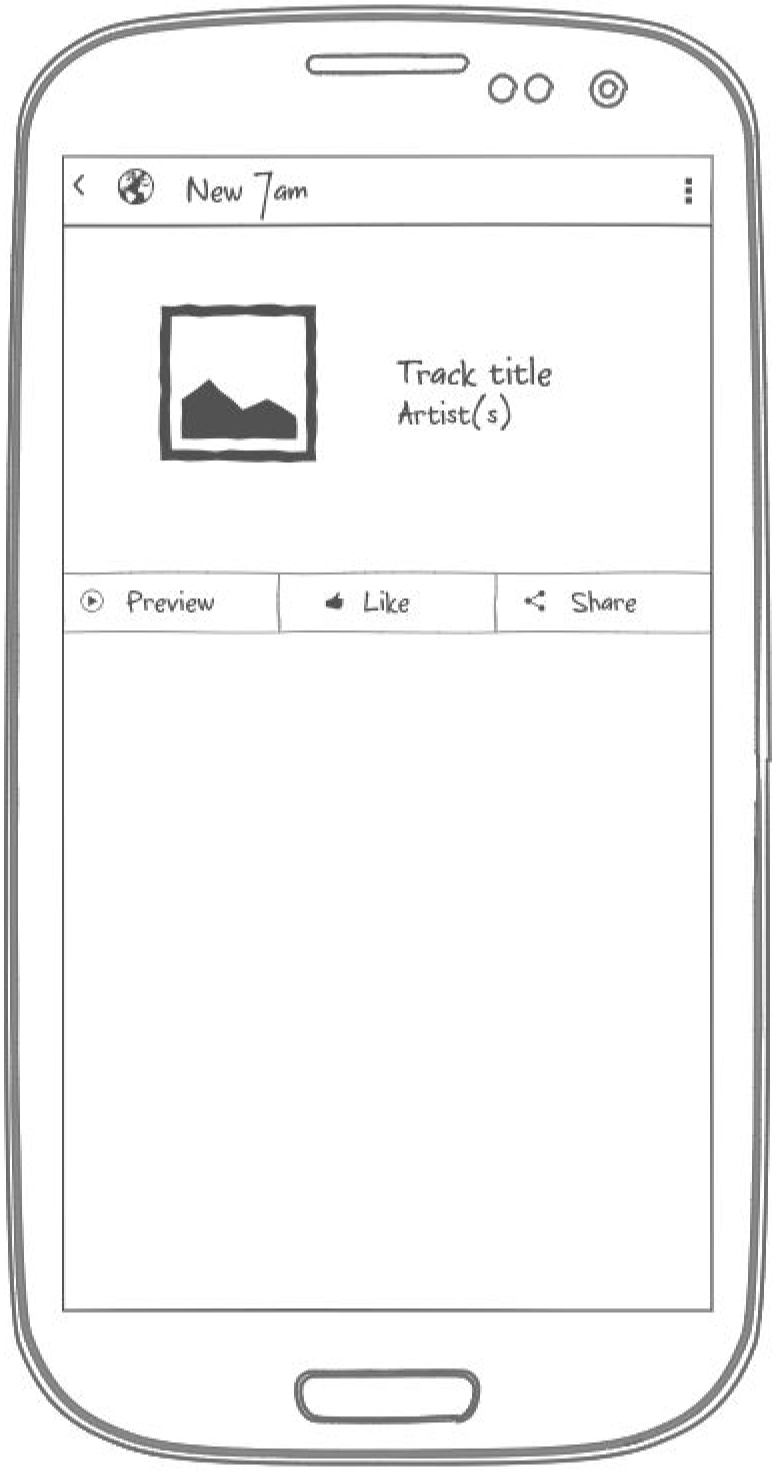
Once the users provides a search query and hits the “Enter” key on the virtual keyboard, the will be shown a list of JammyJamz users (AKA Jammerz) that potentially match the user name. The end user will select a user from the list in order to share the post.

Screen 10


New jam!
<Username> thought you'd dig this jam
10:57 AM
User name
post.
Track title
Artist(s)
Preview
Like
Share
User name
User's message for how they feel about this post.
Album title
Artist(s)
Preview
Like
Share
User name
User's message for how they feel about this post.
Artist

The user to whom the post was shared with will receive a notification indicating that the other user shared a post with them. The user will click on the notification to be brought to the next screen.

Screen 11



This screen contains information pertaining to just the post that was shared with the user. The user can now like the post, preview it, or even share it with another Jammer!

Key Considerations

How will your app handle data persistence?

The app will use Firebase Realtime Database for data persistence. This is critical for the news feed as well as allowing users to share music with one another.

Describe any edge or corner cases in the UX.

The fact that I'm using the Spotify Web API can be considered an edge case as a whole. I'm assuming/expecting users to be comfortable with using Spotify, even if they have another, more preferred music streaming service. I did look into other popular music streaming services, such as Google Play Music, Amazon Prime Music, and Apple Music. However, they either did not have a RESTful API or they didn't offer the features that aligned with what I wanted to do for my project.

Describe any libraries you'll be using and share your reasoning for including them.

I plan on using the following libraries:

- Spotify Web API - Searching for tracks, albums, and artists; it will also be used to allow users to either listen to or preview music.
- Retrofit - Make and receive HTML requests through the Spotify API
- Picasso - image loading and caching
- Butterknife - binding XML views to View objects within Java
- Firebase - Realtime Database for the news feed, Authentication for user sign-in
- Timber - better control over error logging
- Espresso - UI testing

Describe how you will implement Google Play Services or other external services.

I will be using Firebase heavily for this project, especially the Realtime Database.

Next Steps: Required Tasks

Task 1: Project Setup

Before even diving into working on the project itself, the following tasks will be completed:

- Third-party libraries will be added as Gradle dependencies
- Firebase project will be created

Task 2: Implement UI for Each Activity

The following portions of the UI will be created:

- Build UI for the news feed
- Create overflow menu
    - It would only include sign-out button for the time being
    - This would ideally be as reusable as possible since every activity would have the same overflow menu
- Build UI for song search and user search
    - This could potentially be broken down into parent-child classes, where the parent would provide the same general UI elements and the children would simply provide their HTML queries
- Create custom dialog where the user selects what type of post they'd like to make (screen 4)
- Create UI for describing new post (screen 7)
- Create UIs for search results, both posts and users (screens 6 and 9)
- Build UI for viewing a single post (screen 11)
    - This would only be used for when the user views a shared post

Task 3: Create Structure for the Firebase Realtime Database

The Firebase Realtime Database should have a child node within that contains all of the posts within the news feed interface. In addition, each post will also contain key/value pairs which characterize it. This next task can be broken down into the following steps:

- Create a child node, possibly named *posts*, within the Realtime Database which will contain all of the posts within the news feed.
- Determine what key/value pairs are essentially for describing a post within the newsfeed. Some possible key/value pairs include:
    - *username:* Username of the person who made the post
    - *title:* Track/album title
    - *artist:* Artist(s) name
    - *coverArt:* URL for cover art
    - *preview:* URL for preview of the track/album/artist
    - *likes:* How many likes the post has

    - *type:* Describes if the post is for a track, album, or artist. Possible values would be 0, 1, and 2 which would be mapped to a track, album, or artist post respectively.
    - *message:* The message which the user added to describe the post
- Create a Java class that describes a post. Include appropriate getter/setter methods for all member variables.

Task 4: Implement Post Creation

This will likely be one of the more complex parts of the project. The user can make three different types of posts. In addition, they will need to search for the content of the post and add a message before the post is finally added to the Realtime Database. This task can be broken down as follows:

- Create an explicit intent to connect the FAB within the news feed to the dialog in screen 4
- Create an explicit intent to connect the "Ok" button within the dialog to the jam search activity
    - When the user hits the "Ok" button, add their jam type selection as an extra to the explicit intent.
    - Use this extra within the jam search activity to correct structure the HTML request for the appropriate content type (track, album, or artist)
- Create the search feature
    - Extract the content type as an extra from the explicit intent
    - Use the Spotify API, combined with Retrofit, to perform different types of search queries and display the results in a list.
- Create an explicit intent to connect the jam search results activity (screen 6) to the post description activity (screen 7)
    - This can be done via *onClick()* for the *ViewHolder* of the search results list
- Allow the user to describe the post
    - Add an editable text view to hold the user's description
    - Add a button which actually sends the post to the Realtime Database and brings the user back to the news feed.