GitHub Username: mikepalarz

JammyJamz

Description

This app is intended to allow users to share their musical interests with other users. It will include a news feed interface which shows what music they're currently listening to, along with their peers' current musical interests. 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:

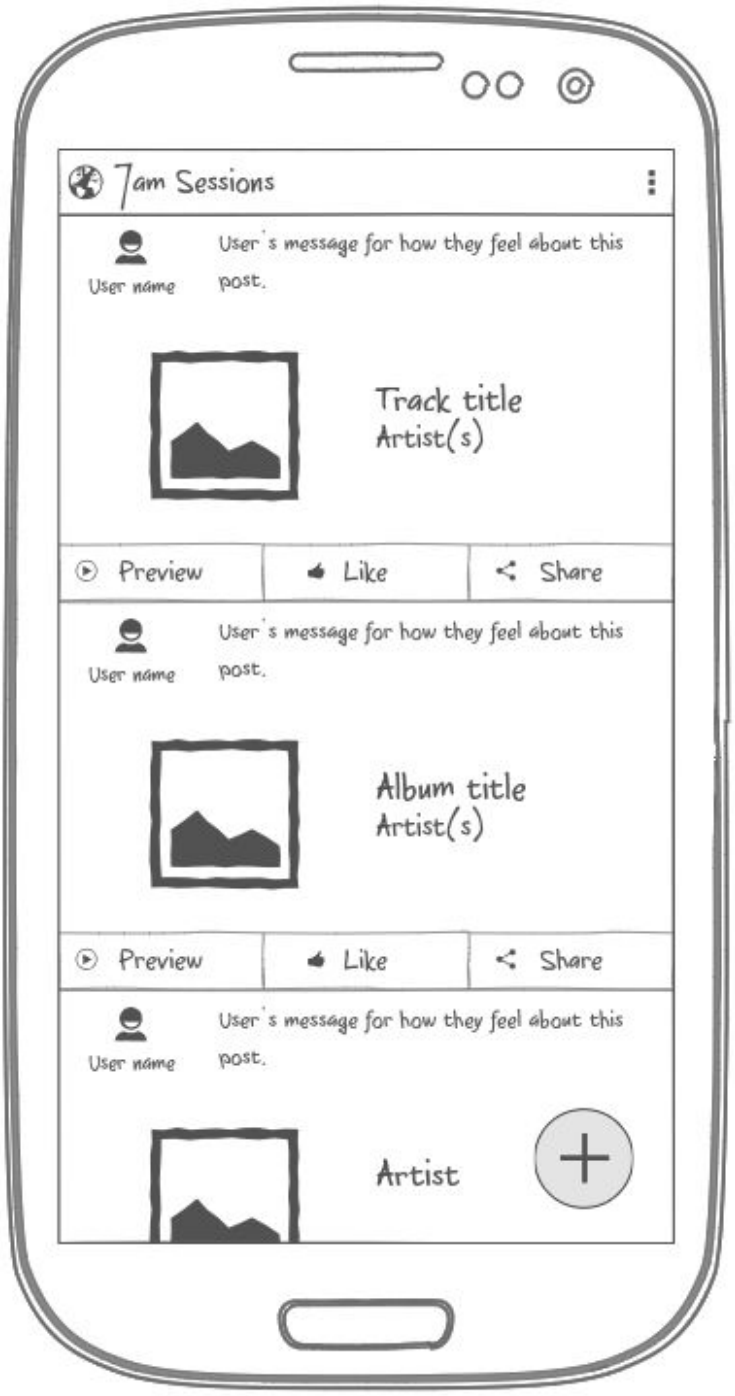
- News feed interface which shows all registered users' current musical interests (tracks, albums, and/or artists).
- Users can contribute to the news feed by searching for tracks/albums/artists and creating a post.
- Previews of tracks/albums/artists.
    - If the user has the Spotify app installed, 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 post with them.
- App will be solely written in the Java language.
- All of the strings within the app will be stored within a *strings.xml* file, making localization easily possible.
- Easily accessible for RTL languages by providing appropriate layout features - *start* and *end* XML attributes will be used throughout the app.

User Interface Mocks

Screen 1

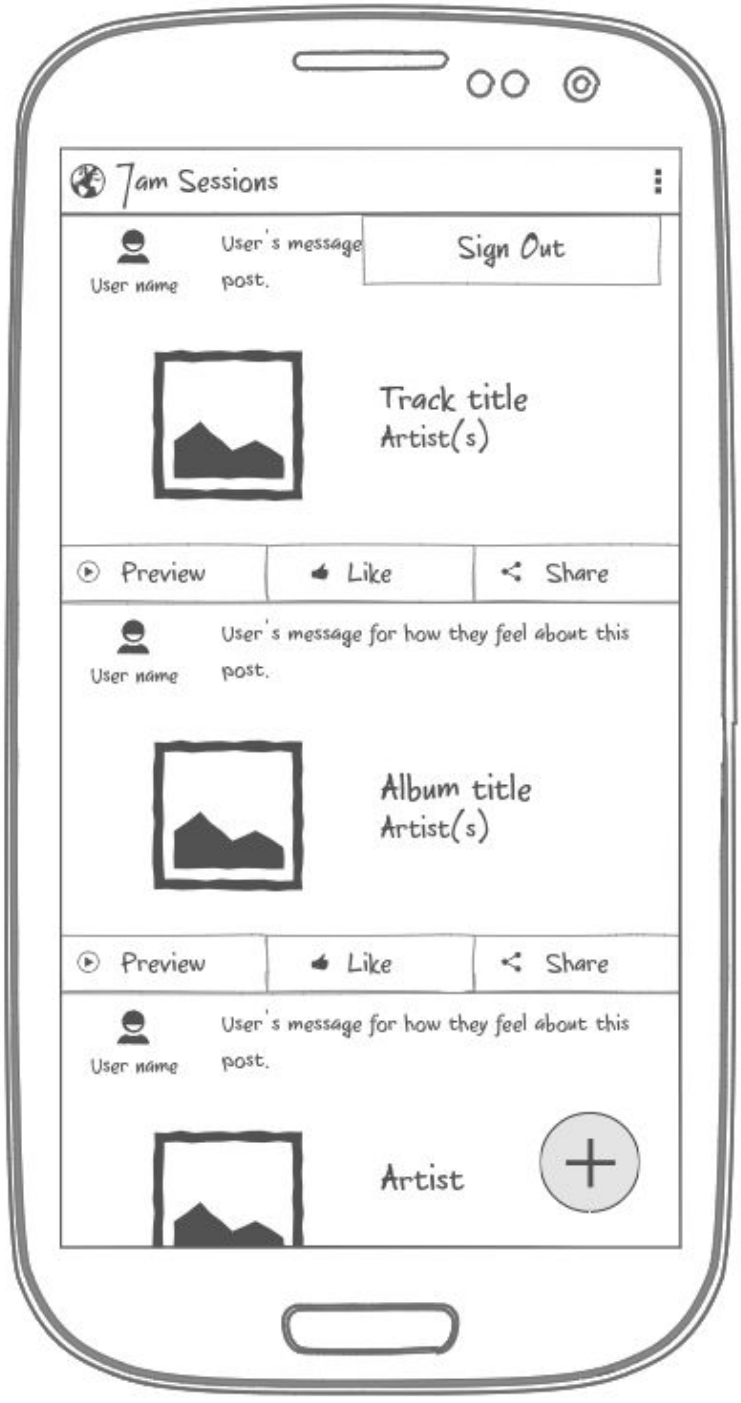
Sign-in screen for JammyJamz. The Google and Facebook sign-ins will be implemented by using FirebaseUI

Screen 2



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

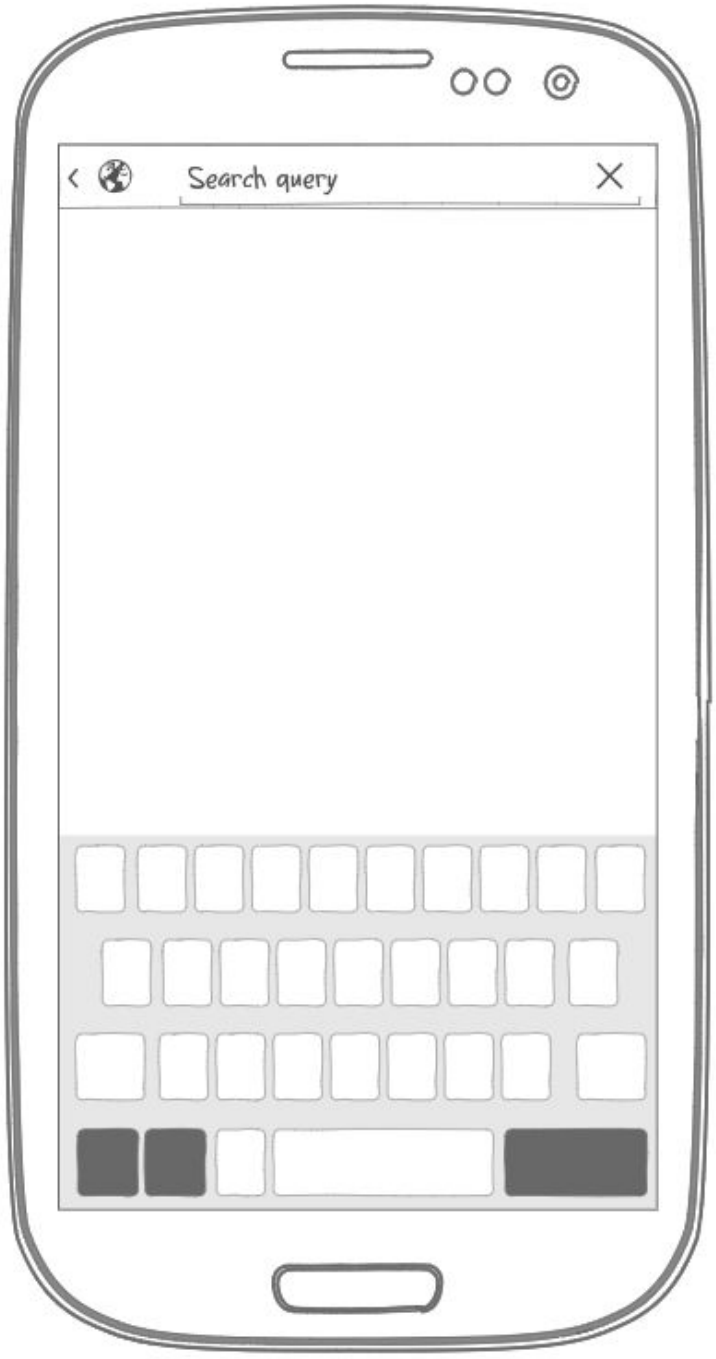


Under all circumstances (i.e., within all UI screens), the overflow menu will allow the user to sign out of JammyJamz.

Screen 4

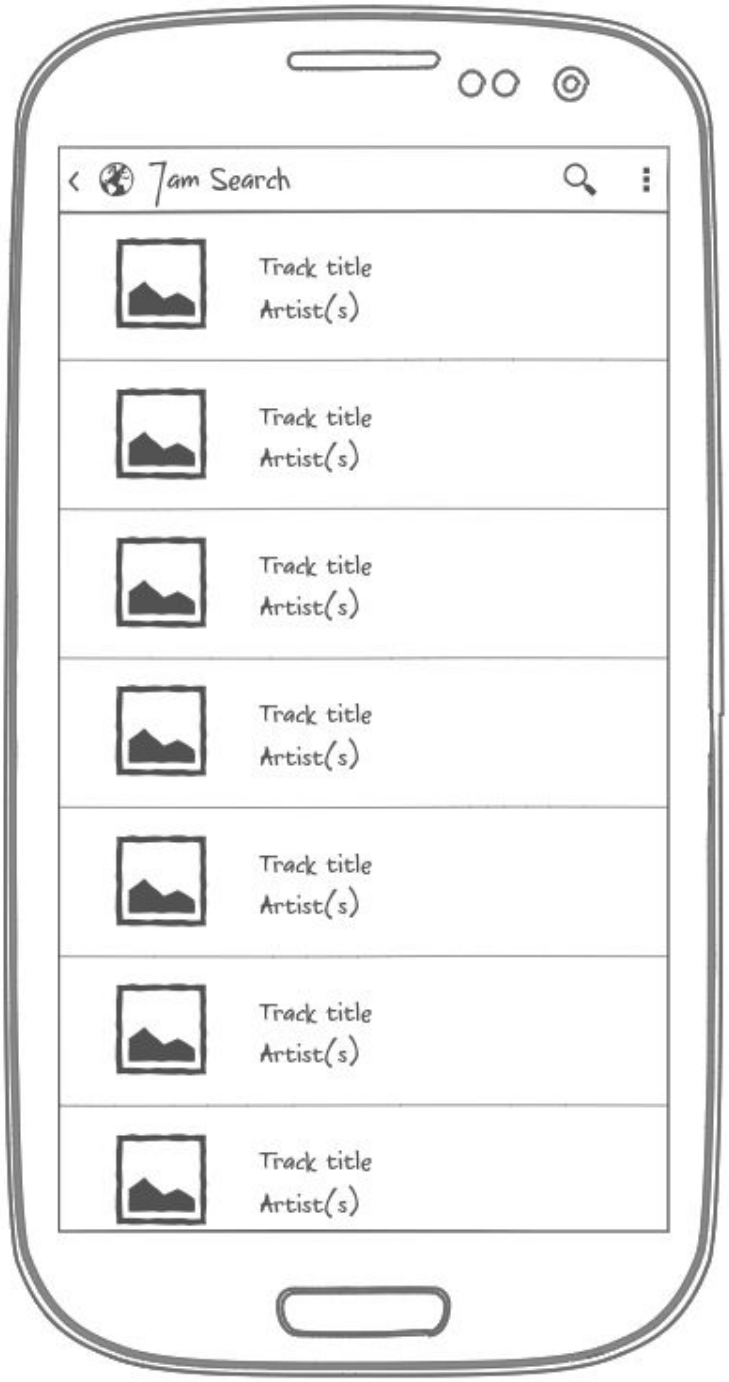
When the user clicks on the FAB, they will be presented with a dialog asking them what kind of post they'd like to make: track, album, or artist. If the user clicks "Cancel", then they will be brought back to the news feed. If they click "Ok", then they will be brought to the next screen.

Screen 5



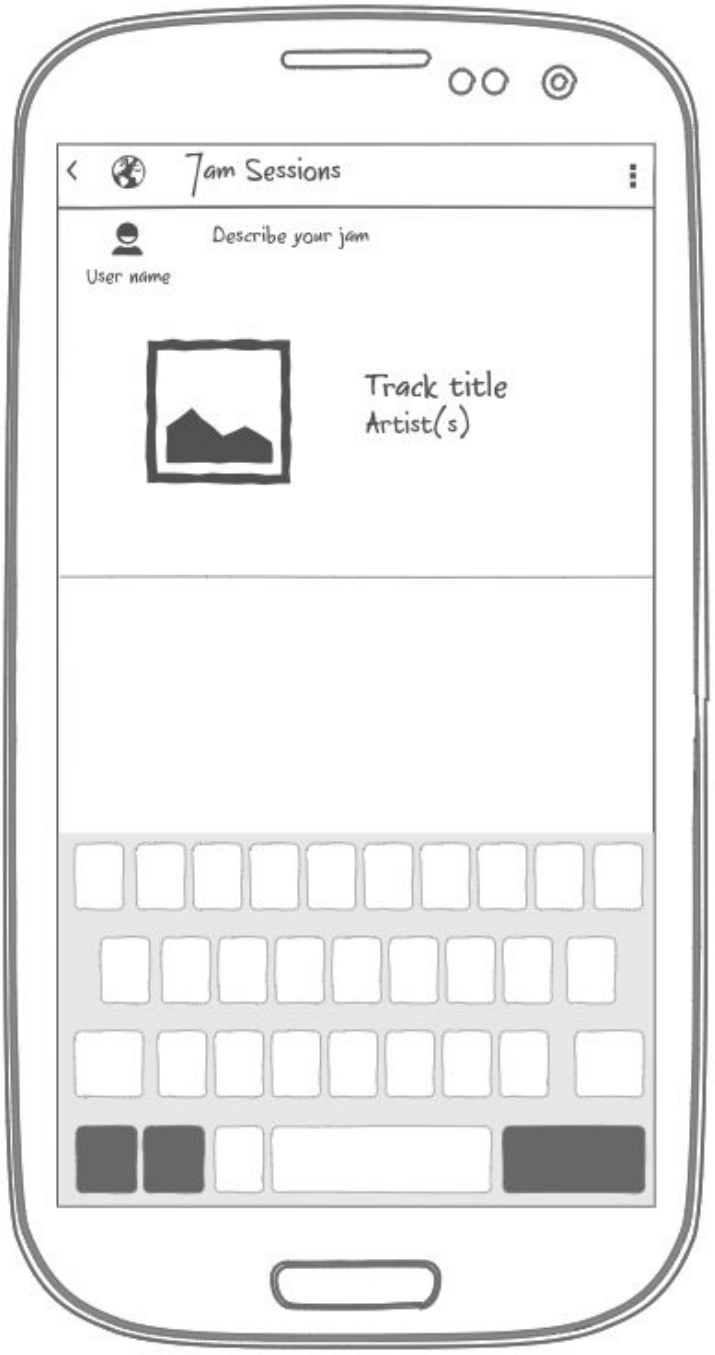
The user will then be prompted to search for the kind of material they are hoping to make a post about. Once the user hits “Enter” on the virtual keyboard, the query will be sent and they’ll be shown the next screen.

Screen 6



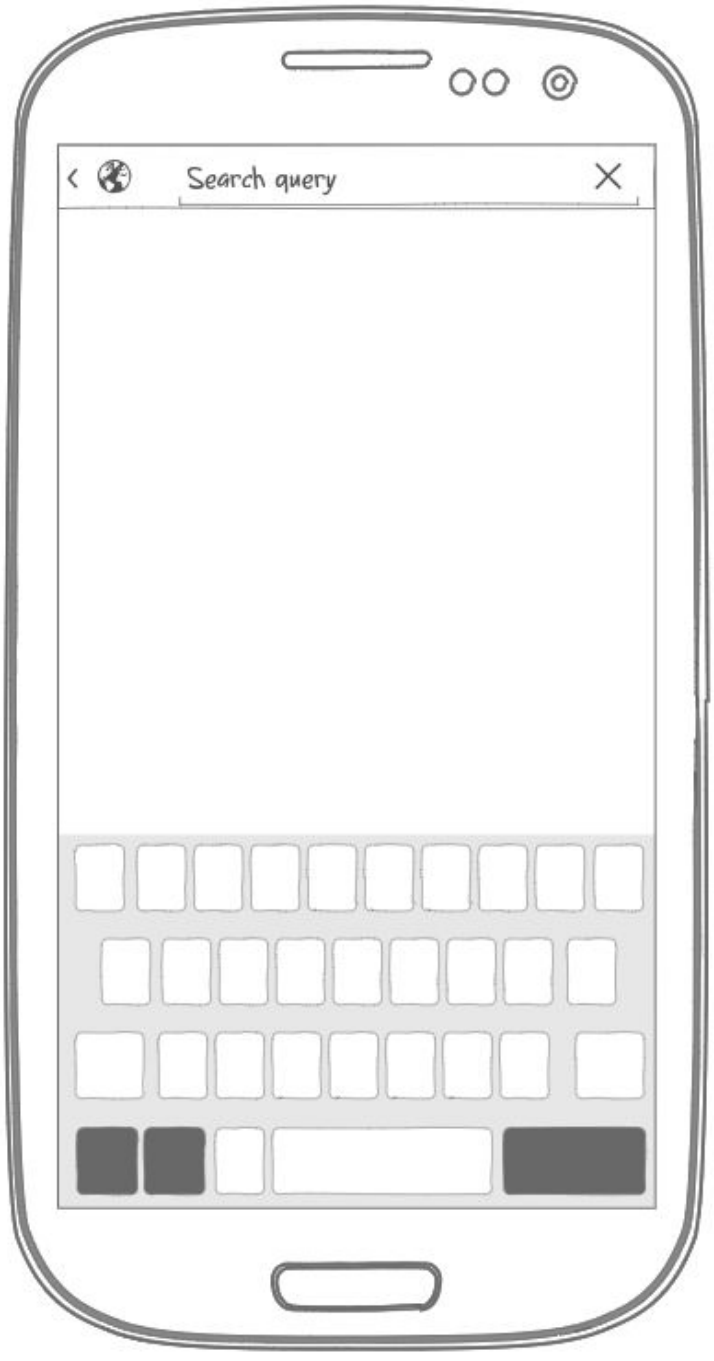
The user will be shown a list of search results from which they can choose to use for their post.

Screen 7



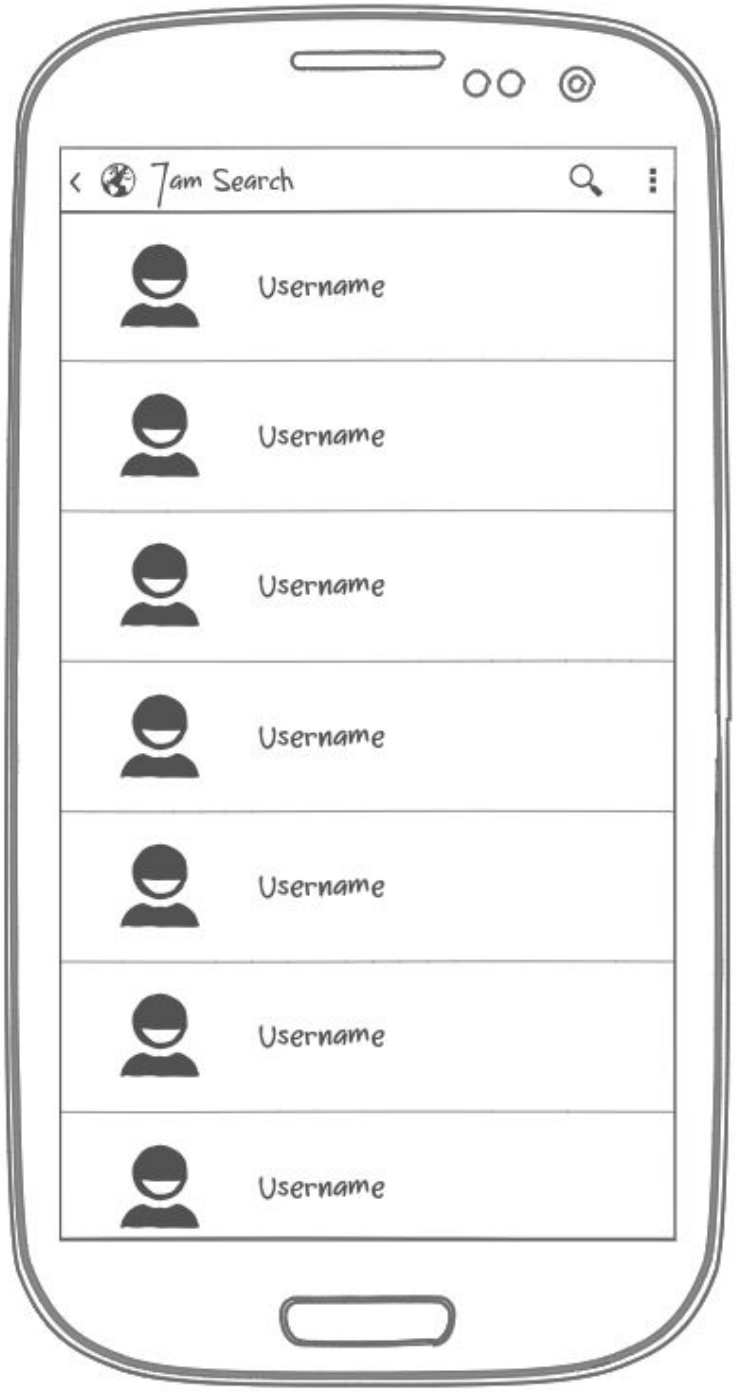
Finally, the user will be prompted to add an optional description of their post. This allows the user to further personalize how they feel about the given track/album/artist.

Screen 8



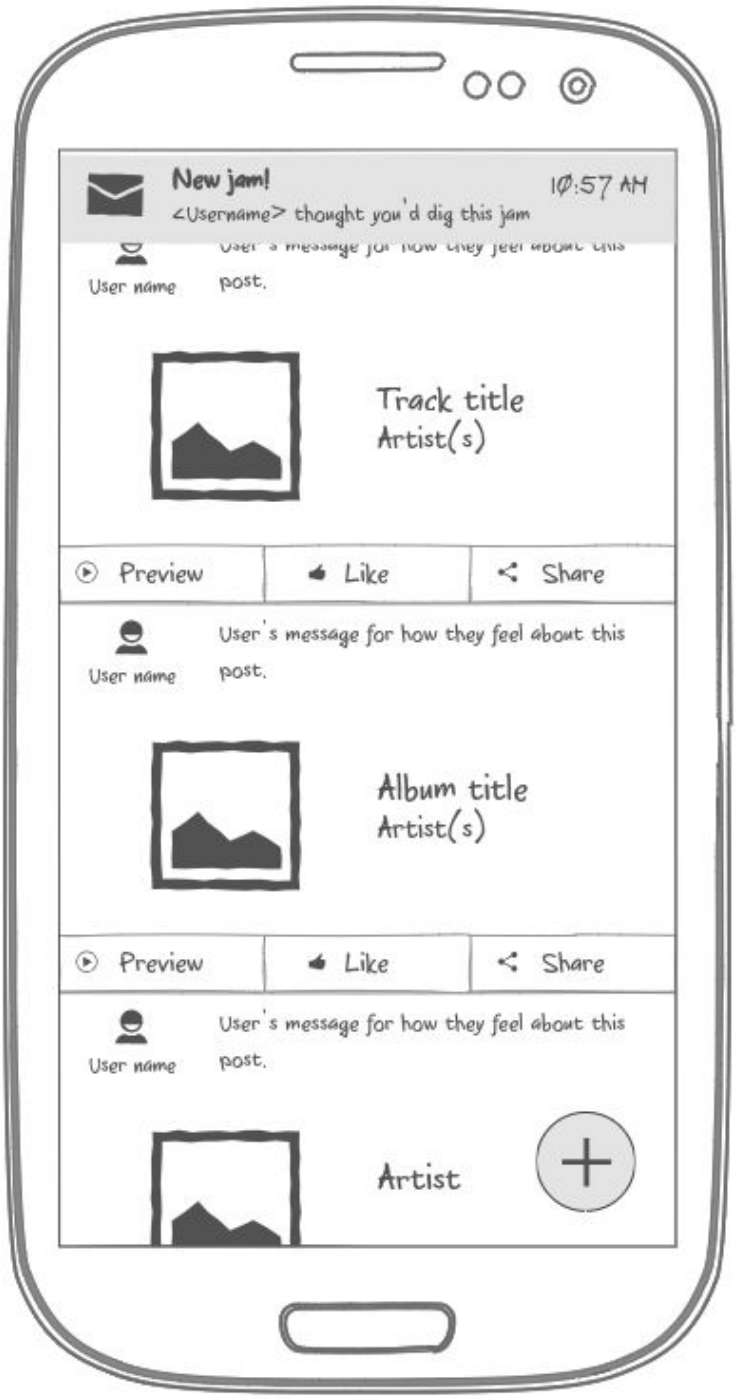
Users also have the option to share posts with other users. When a user clicks on the “Share” button for a particule post, they will first be shown a search UI, which allows them to search for the user they’d like to share the post with. The UI will be visually identical to search for a track/album/artist,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

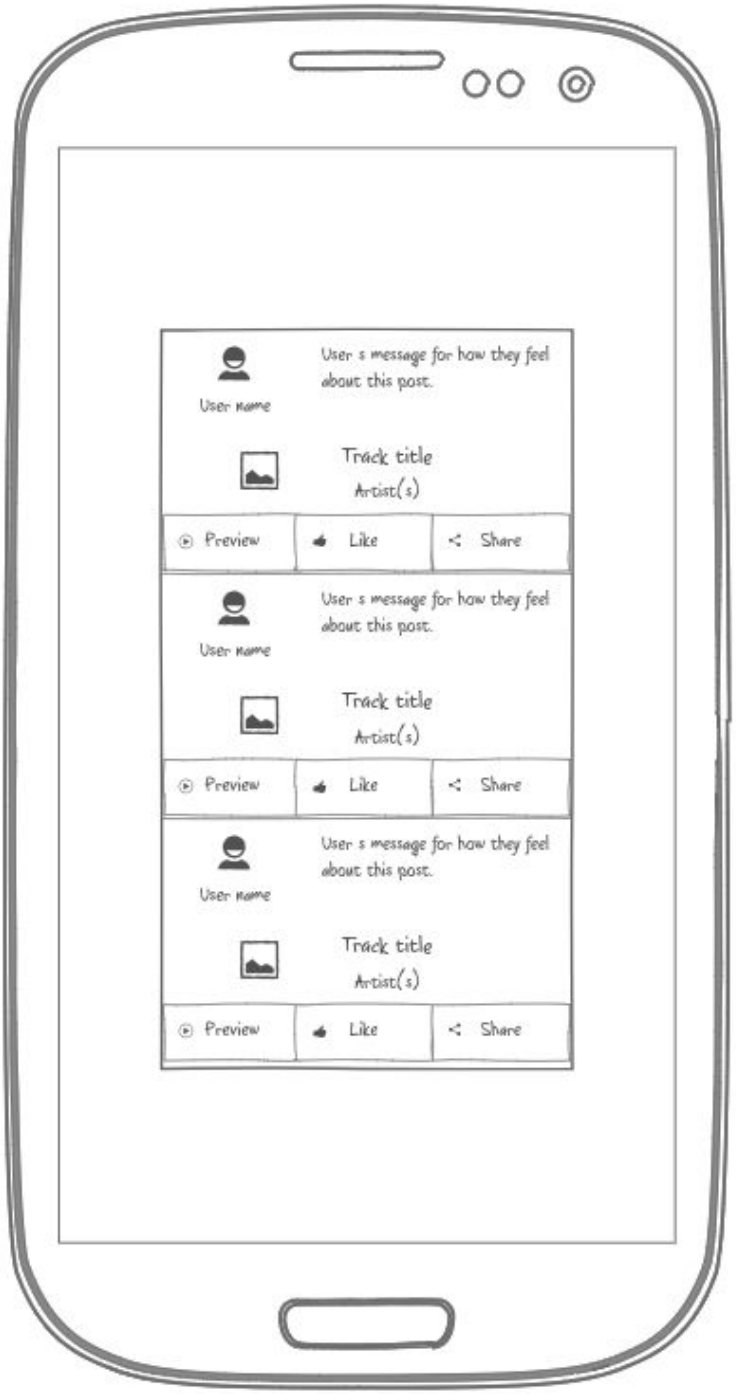


The user to whom the post was shared will receive a notification indicating that the other user shared a post with them. The user will click on the notification to be brought to the next screen.

Screen 11

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

Screen 12



A widget will also be available to the user so that they can quickly view the latest jamz, directly from their home screen.

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.

In addition, caching will be implemented through both Firebase and Picasso. Firebase offers offline capabilities while Picasso implements caching whenever downloading an image, which is particularly useful for scrolling through list views.

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:

Library Name	Description	Version Number
Spotify Web API	Searching for tracks, albums, and artists; it will also be used to allow users to either listen to or preview music.	N/A
Retrofit	Make and receive HTML requests through the Spotify API	2.4.0
Picasso	Image loading and caching	2.71828

Butterknife	Binding XML views to *View* objects within Java	8.8.1
Firebase	Realtime Database for the news feed, Authentication for user sign-in, Cloud Messaging for notifications	16.0.1
Timber	Better control over error logging	4.7.1
Espresso	UI testing	3.0.2

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 the 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 a 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: Structure of Firebase Realtime Database

The Firebase Realtime Database should have a child node that contains all of the posts within the news feed interface. In addition, each post will also contain key/value pairs which characterize it.

- Create a child node, named *posts*, within the Realtime Database which will contain all of the posts within the news feed.
- Determine what key/value pairs are essential for describing a post within the news feed. 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
    - *usersLiked:* A child of *posts* that includes key.value pairs for the users that liked the post
        - *username:* A user that has liked this post
    - *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: 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 first need to search for the content of the post and

then 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 correctly structure the HTML request for the appropriate content type (track, album, or artist). This will be useful for the next sub-task.
- 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 a *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 (screen 2).

Task 5: User Sign-In

Implement user authentication to allow users to create an account and sign in/out of JammyJamz (i.e., to become a Jammer). This would be accomplished via the following:

- Provide sign-in UI within screen 1 by using FirebaseUI
- Customize the sign-in flow and UI of the sign-in screen
    - Add Facebook and Google sign-in options
    - Adjust the color scheme of the sign-in UI to match the colors used for JammyJamz
- Link the sign-out button within the overflow menu to a function which actually signs out the user

Task 6: Widget

Users will be able to get a quick snapshot of the news feed through a widget on their home screen.

- Create the UI for the widget
- Adjust the *AppWidgetProviderInfo* XML so that the widget updates at a reasonable rate.
- Link the UI within the widget to the Firebase Realtime Database

Task 7: Liking a Post

The “Like” button will update a count within the UI that displays the total number of likes that the post has. When a user clicks on the “Like” button, the count should be incremented. A user should only be able to like a post once and they should also have the ability to “unlike” a post (remove their “like”).

- Implement *onClick()* of the like button
    - If the user is not included within *usersLiked* of the Realtime Database (refer to task 3), then the user will be able to like the post.
        - The like count will be incremented by 1.
        - *usersLiked* will be updated to include the new user.
        - The like button within the post will change color to be indicative to the user that they’ve already liked the post.
    - If the user is already included in *usersLiked*, then clicking the like button will “unlike” the post.
        - The like count will be decremented by 1.
        - *usersLiked* will be updated by removing the user.
        - The like button will be reverted back to its default color to indicate to the user that they’ve removed their like.

Task 8: Previewing a Post’s Content

When the user clicks on the preview button of a post, they will be offered a preview of the post’s content. Regardless of the type of post (track, album, or artist), the user can preview the content as long as the Spotify API offers a URL for a preview.

- Implement *onClick()* of the preview button
    - Within *onClick()*, an implicit intent will be created. The URL for the preview of the post will be included as an extra to the intent.
    - The preview URL will be sent to the user’s web browser on their device.
- The user will then be able to preview the content in one of two ways:
    - If the user has Spotify also installed on their device, then the URL will cause the device to open Spotify. The Spotify app will be redirected to the content they were hoping to preview.
    - If the user does not have the Spotify app installed, then they will be taken to a web page on the Spotify website which includes 30s previews of the content.

Task 9: Sharing a Post

Users can share a jam with other Jammerz if they feel another user would enjoy the post's content.

- Connect the share button to the activity where users can search for other Jammerz (AKA Jammerz search, screen 8).
- Implement the Jammerz search functionality
    - When a username is entered into the search bar of Jammerz search, this creates an HTML request to the Realtime Database for a list of users which match the search request.
    - The UI will then be updated with a list of users that matched the search request (screen 9).
- Implement *onClick()* for the *ViewHolder* of the search results to launch a notification to the user that's to receive the post.
    - This would be accomplished through Firebase Cloud Messaging
- Adjust the app so that when a pending intent is received from this type of notification, then the single post activity is displayed to the user (screen 11).