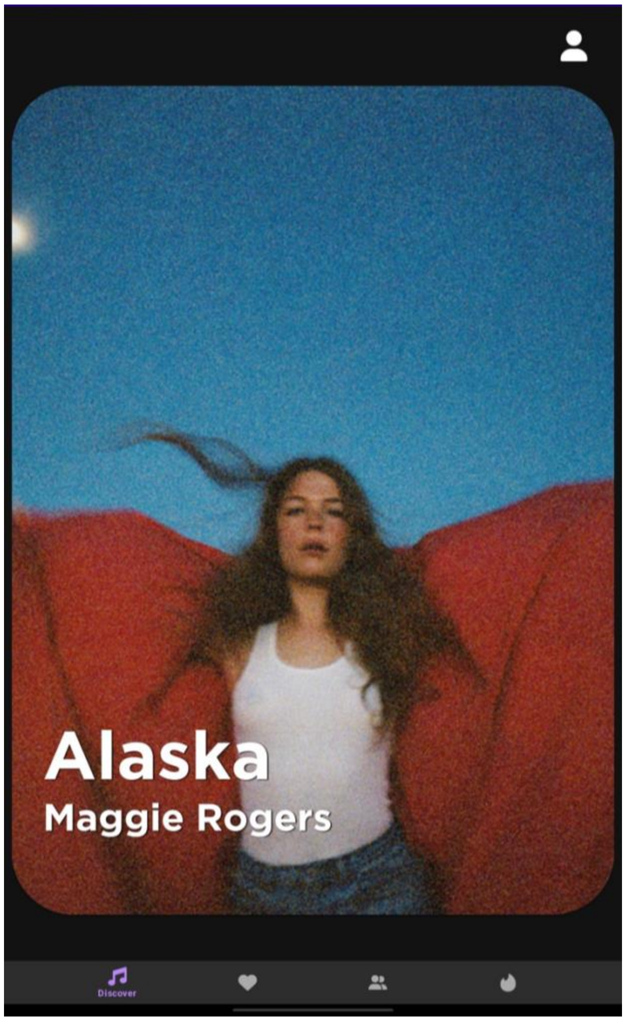
Tuneder

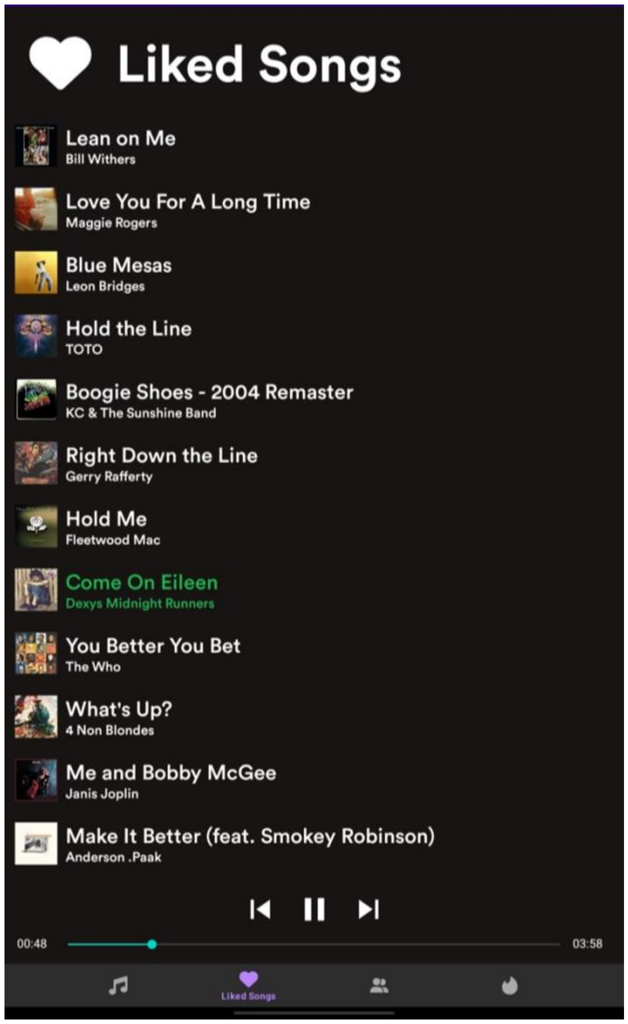
Created by Hannabeth Medina

A music discovery app that feels like a dating app. Swipe on songs and generate recommended playlists based on your own likes or a mash-up with your friends.

In-App Screenshots

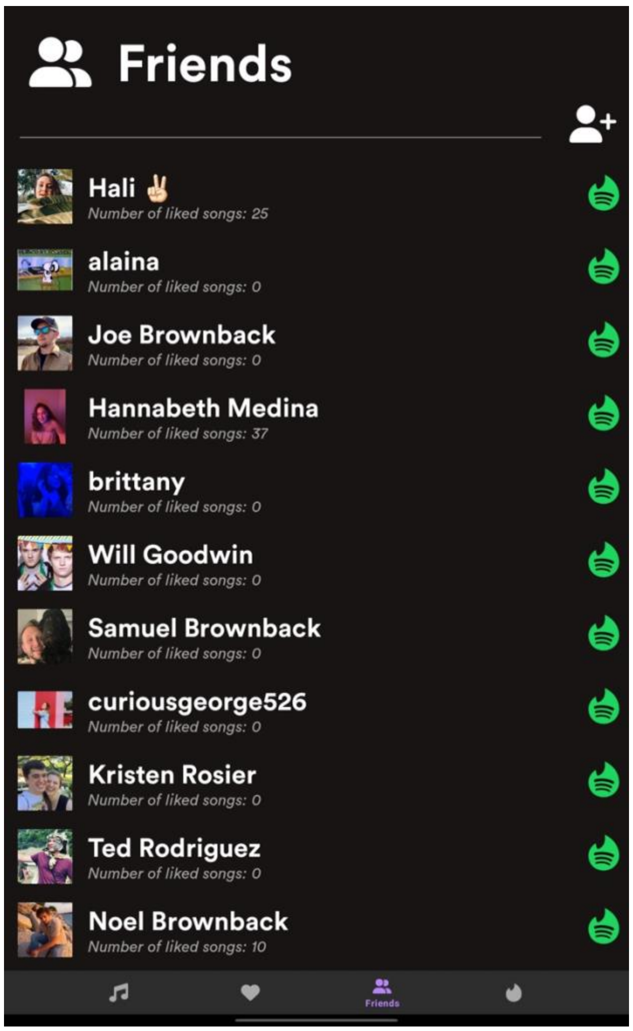


Home page: Discover new music



Lean on Me
Bill Withers
Love You For A Long Time
Maggie Rogers
Blue Mesas
Leon Bridges
Hold the Line
TOTO
Boogie Shoes - 2004 Remaster
KC & The Sunshine Band
Right Down the Line
Gerry Rafferty
Hold Me
Fleetwood Mac
Come On Eileen
Dexys Midnight Runners
You Better You Bet
The Who
What's Up?
4 Non Blondes
Me and Bobby McGee
Janis Joplin
Make It Better (feat. Smokey Robinson)
Anderson .Paak

Liked Songs page: Listen to the songs you've swiped right on



Friends

Hali ✌️
Number of liked songs: 25

alaina
Number of liked songs: 0

Joe Brownback
Number of liked songs: 0

Hannabeth Medina
Number of liked songs: 37

brittany
Number of liked songs: 0

Will Goodwin
Number of liked songs: 0

Samuel Brownback
Number of liked songs: 0

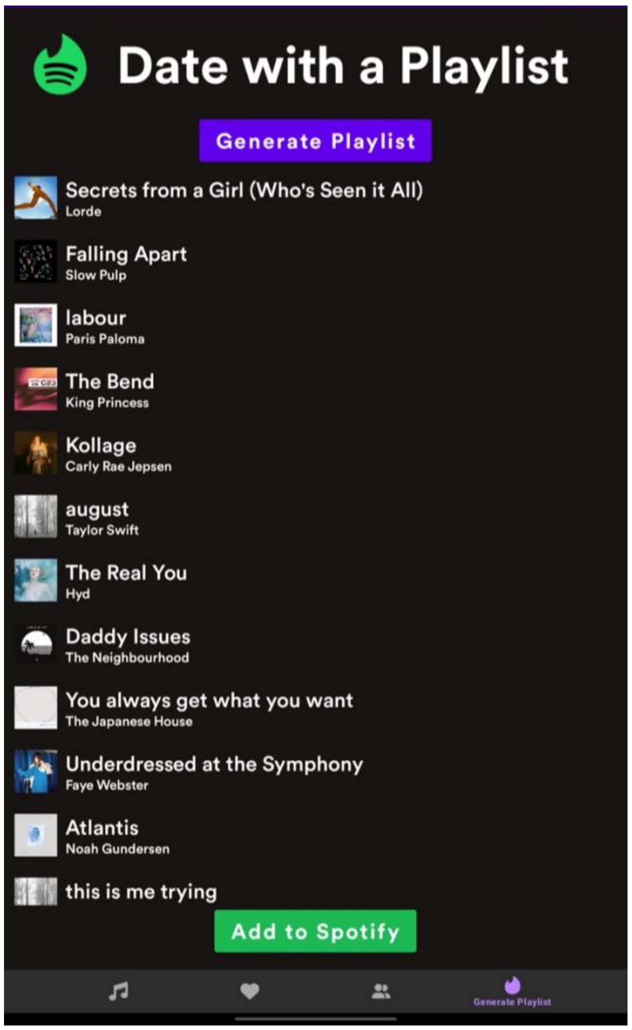
curiousgeorge526
Number of liked songs: 0

Kristen Rosier
Number of liked songs: 0

Ted Rodriguez
Number of liked songs: 0

Noel Brownback
Number of liked songs: 10

Friends page: Add new friends and generate mash-up playlists



Date with a Playlist

Secrets from a Girl (Who's Seen it All)
Lorde

Falling Apart
Slow Pulp

labour
Paris Paloma

The Bend
King Princess

Kollage
Carly Rae Jepsen

august
Taylor Swift

The Real You
Hyd

Daddy Issues
The Neighbourhood

You always get what you want
The Japanese House

Underdressed at the Symphony
Faye Webster

Atlantis
Noah Gundersen

this is me trying

Generate Playlist page: Generate a playlist of recommendations and add it to Spotify

Team Information

Name: Hannabeth Medina
EID: hrm747
Email: hannabethmedina@utexas.edu

APIs/Features

The APIs used in this project are listed below:

- Spotify Web API
- Cloud Firestore
- BottomNavigationView

Third-Party Libraries

Spotify Android SDK

The Spotify Android SDK consists of the following two libraries:

Authorization Library

This library is used to authorize the app and fetch an access token. This token is used to send requests to the Spotify Web API to retrieve data used for display and backend processing throughout the app e.g. user data, track audio analysis, etc.

App Remote Library

This library is used to manage playback in the Spotify app running in the background. This allows the user to listen to the collection of liked songs they've curated while swiping in the app.

The most challenging aspect for each of the libraries in this SDK was in integrating them. Spotify's documentation and tutorials, while fairly comprehensive, are out of date in significant areas. I relied heavily on online resources (Stack Overflow, ChatGPT, Medium) for the configuration of dependencies and initial implementation. Once past these struggles on setup, both libraries were extremely easy to use and essential in key functionality in the app.

CardStackView by Yuyakaido

This library (source found here) is used to create the card stack view used for the home (discovery) interface. It provides the framework for the swiping action used to like or dislike songs.

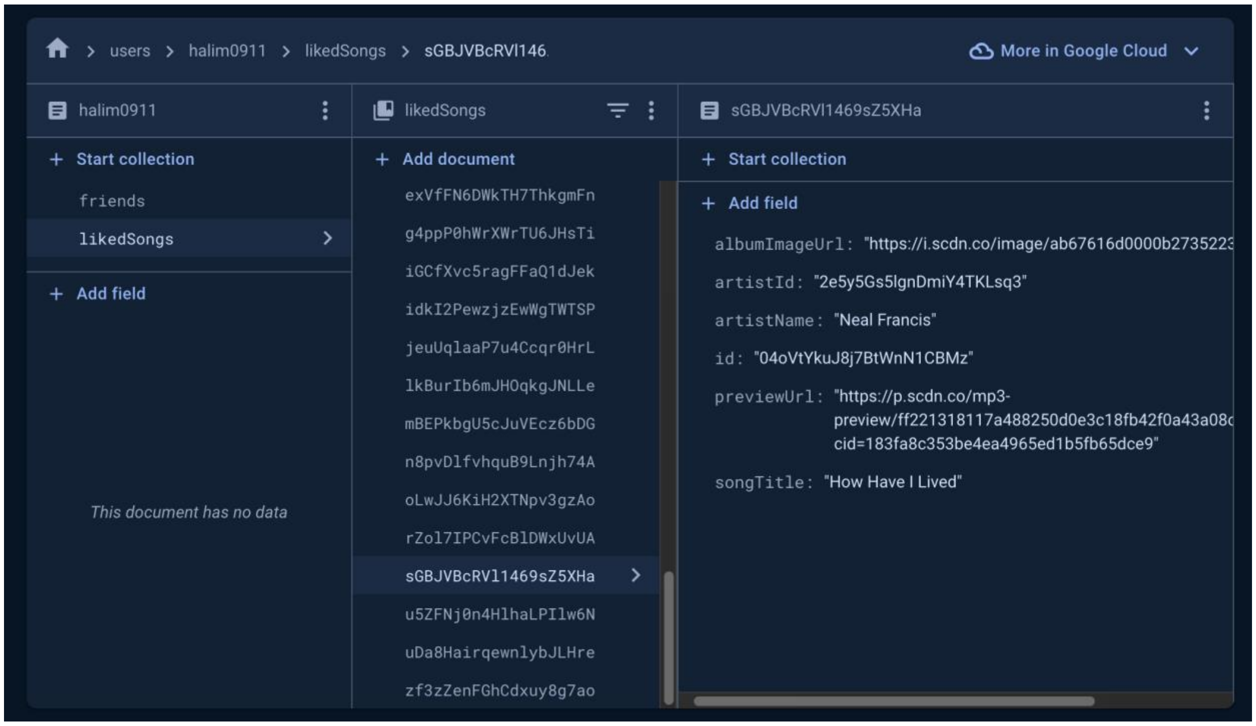
The most challenging aspect in using this library was in configuring the dependency. With a lack of explicit or thorough documentation and incompatibilities between dependencies in the gradle file, determining a combination of versions of the required libraries that worked together to produce a clean gradle build was a time-consuming process. This also required substantial help from online resources.

Third-Party Services

Firestore Database

This database service is used to store and sync data for the app. Users' friends and liked songs are stored here and are retrieved on demand. Users are also able to see a count of each of their friends' in-app liked songs that is synced in real-time using a snapshot listener.

User's Liked Songs



users > halim0911 > likedSongs > sGBJVBcRVl146.

More in Google Cloud

halim0911

+ Start collection

friends

likedSongs

+ Add field

This document has no data

likedSongs

+ Add document

```
exVfFN6DWkTH7ThkgmFn
g4ppP0hWrXWrTU6JHsTi
iGCfXvc5ragFFaQ1dJek
idkI2PewzjzEwWgTWTSP
jeuUqlaaP7u4Ccqr0HrL
lkBurIb6mJHOqkgJNLLe
mBEPkbgU5cJuVEcz6bDG
n8pvDlfvhquB9Lnjh74A
oLwJJ6KiH2XTNpv3gzAo
rZol7IPCvFcBlDWxUvUA
sGBJVBcRVl1469sZ5XHa
u5ZFNj0n4HlhaLPIlw6N
uDa8HairqewnlybJLHre
zf3zZenFGhCdxuy8g7ao
```

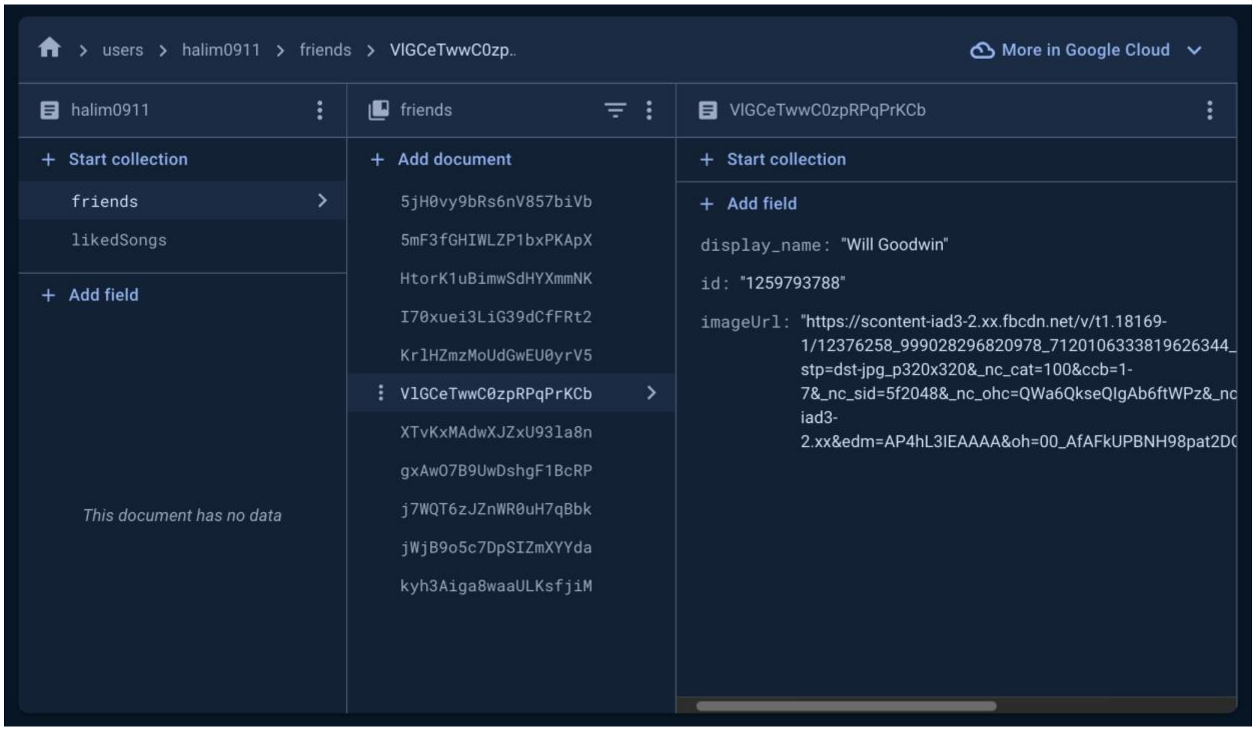
sGBJVBcRVl1469sZ5XHa

+ Start collection

+ Add field

```
albumImageUrl: "https://i.scdn.co/image/ab67616d0000b2735223
artistId: "2e5y5Gs5lgnDmiY4TKLsq3"
artistName: "Neal Francis"
id: "04oVtYkuJ8j7BtWnN1CBMz"
previewUrl: "https://p.scdn.co/mp3-
            preview/ff221318117a488250d0e3c18fb42f0a43a08c
            cid=183fa8c353be4ea4965ed1b5fb65dce9"
songTitle: "How Have I Lived"
```

User's Friends



users > halim0911 > friends > VlGCeTwwC0zp..

More in Google Cloud

halim0911

+ Start collection

friends

likedSongs

+ Add field

This document has no data

friends

+ Add document

```
5jH0vy9bRs6nV857biVb
5mF3fGHIWLZP1bxPKApX
HtorK1uBimwSdHYXmmNK
I70xuei3LiG39dCfFRt2
KrlHZmzMoUdGwEU0yrV5
VlGCeTwwC0zpRPqPrKCb
XTvKxMAdwXJZxU93la8n
gxAwO7B9UwDshgF1BcRP
j7WQT6zJZnWR0uH7qBbk
jWjB9o5c7DpSIZmXYYda
kyh3Aiga8waaULKsfjiM
```

VlGCeTwwC0zpRPqPrKCb

+ Start collection

+ Add field

```
display_name: "Will Goodwin"
id: "1259793788"
imageUrl: "https://scontent-iad3-2.xx.fbcdn.net/v/t1.18169-
          1/12376258_999028296820978_7120106333819626344_
          stp=dst-jpg_p320x320&_nc_cat=100&ccb=1-
          7&_nc_sid=5f2048&_nc_ohc=QWa6QkseQIgAb6ftWPz&_nc
          iad3-
          2.xx&edm=AP4hL3IEAAAA&oh=00_AfAFkUPBNH98pat2DC
```

AI Contribution

OpenAI's ChatGPT was utilized in debugging throughout this process, with varying degrees of success. Its most extensive usage was in configuring the third-party libraries and associated dependencies. An example prompt format: *In Android Studio Hedgehog 2023, how do I import a JAR/AAR Package as a new module in my project?*

Additionally, a notable contribution from ChatGPT was logic to determine the most frequent values of a field in a list of items. The function *findTopArtists()* uses this logic to find the "top" artists in the user's liked songs list.

As mentioned above, the utility of ChatGPT varied by topic and use-case. While straightforward in the usage for generating a piece of logic or in decoding a cryptic error message, it became less useful when dealing with abstract issues or crashes without verbose error logging.

UI/UX Discussion

CardView Swipe Interface

The key UI element of Tuneder is the swipe interface on the Discovery screen that allows users to listen to a quick preview of a song and decide on whether they like (swipe right) or dislike (left swipe) it. This was implemented using CardViews and a CardStackView by Yuyakaido. The thought behind this design was to provide the user with a fun and interactive way to discover music using a popular swiping method that's common amongst current dating apps and familiar to most users.

BottomNavigationView

A BottomNavigationView serves as the primary way to navigate between the different fragments within the app.

The only two exceptions to this method of navigation are:

- Jumping back to the Discovery screen from the User Profile
- Jumping to the Generated Playlist screen after generating a mash-up playlist from the Friends screen.

Thoughtful Style Decisions

Throughout the app, the style (fonts and color palette) used was intentionally designed after Spotify to create a cohesive feel with the app. The design for Tuneder's logo (displayed throughout the app and at the top of this report) is meant as a play on the two apps that this app was inspired from – Tinder and Spotify.

Backend Discussion

Spotify Web API

Much of the core functionality throughout Tuneder is made possible with the Spotify Web API. These are all the endpoints used and their functionality:

- **Song Recommendations (/v1/recommendations):** This request returns recommended songs based on provided seeds (artists, tracks, and genres) and optional feature targets. This is used both to fetch new songs on the Discovery screen and to generate playlists on the Generated Playlist (and Friends) screen.
  - When used for the Discovery screen, the API call includes an artist seed with the user's top two artists based on their liked songs.
  - When used for the Generated Playlist screen, the API call includes additional parameters. For each song in the user's liked songs list, audio features for the songs such as danceability, valence, speechiness, etc. are retrieved. These values are then averaged across all the songs and passed into the API call as target values for the features.
- **Audio Features(/v1/audio-features/<track-id>):** This request returns an audio feature analysis of a provided track ID. It is used to retrieve the features that are used for playlist generation, as described above.
- **User (v1/users/<username>):** This request returns user profile information based on a provided username. This is used to fetch users on the Friends screen.
- **Current User (/v1/me):** This request returns user profile information on the current user. This is used to retrieve the current user's info used throughout the app.
- **Create New Playlist (/v1/users/<username>/playlists):** This request allows us to create a playlist in Spotify given the user's username and a specified name for the playlist. This is used in the Generated Playlist screen to create the playlist in Spotify.
- **Add Songs to Playlist (/v1/playlists/<playlist-id>/tracks):** This request allows us to add songs to a playlist using a provided playlist ID and a list of songs URIs. This is used in combination with the call to create the new playlist in the Generated Playlist screen to populate the newly created playlist with the recommended songs.

Difference in Generating Playlists

There are two methods of generating a playlist of recommended songs: on the Generated Playlist screen and on the Friends screen. The key difference in the playlist generated is the pool of liked songs that are used to determine parameters that are passed into the API request to retrieve recommended songs back. When triggered in the Generated Playlist screen, the current user's liked songs are used, and when triggered on a friend in the Friends screen, both the current user's and the selected friend's liked songs are used.

Friends Liked Songs Count

On the Friends screen, the count of liked songs (stored in Firestore Database) is displayed for each friend. This count is updated in real-time using snapshot listeners, so the values displayed on each friend are live and consistent for Tuneder' users.

Important Takeaways

While most of the functionality in Tuneder is largely a culmination of what we've learned in assignments throughout the semester, putting all the puzzle pieces together to create a fully functioning app was the most valuable learning experience of this project. Having to figure out the beginning-to-end of authenticating and fetching data to storing, displaying, and persisting data across multiple users and multiple sessions – and everything else in between – was no simple feat. Additionally, learning to work with Spotify Web API and its limitations to develop features for the app with what information is available (instead of the other way around) was a great learning opportunity. Though the specific puzzle pieces that I use in my day to day may look different, learning how to manage a project with a large scope and integrate with third party APIs are skills that have helped me become a better developer.

Challenges

While a variety of challenges were encountered in this process, the most frustrating and exasperating aspect was the app's most essential element: Spotify Web API. The readability of Spotify's documentation and tools for this were incredibly helpful, however, the Web API was unreliable to develop with at best and became increasingly unpleasant in the last few days leading up to the assignment's due date.

A particularly hair-pulling incident occurred in these few days before the due date. Over the course of the preceding days, heavy implementing and testing on the app was being conducted. After a single small tweak to the UI (modification to a button element's left and right padding), the app was suddenly unable to run. Any attempts to run the debugger failed, and the app would crash before displaying the home/discovery screen. Even after backing out the previous UI modification that had seemingly caused the entire app to collapse (it was unrelated), the behavior didn't change. With absolutely no helpful messages in the debug log, I was completely lost. After my futile attempts to comment out any code that seemed suspicious and many attempts to restart devices and invalidate caches, I was finally able to deduce what was causing the crashing.

The home page makes an API call to retrieve a recommended song for the user. My API call, which had worked the numerous times in the previous weeks failed and crashed the app every time. Though Spotify's Web API documentation only lists a rate limit of 100 calls in 30 seconds, that is apparently not all the restrictions enforced. I found no official documentation or known related bugs listed but instead found extensive threads from users in situations similar to mine – API calls suddenly stopped working after too many calls in an unspecified period of time.

In looking at the Spotify Developer console, I found that I had hit several endpoints hundreds of times in single days. The most frustrating aspect of this is that there is no solution to this or no way to allot or purchase more API calls. My only options were to create different projects in the Spotify Developer console with other Spotify's user accounts (in my case, borrowing friends and family's accounts), or wait an unspecified "long" period of time (I experienced between 13 and 24 hours).

While that was challenging to deal with, especially with an approaching deadline, I learned a valuable lesson on working with components that are out of my control and finding patience and resilience to find a way to continue making progress.

How to Build and Run Tuneder

Instructions to build and run Tuneder are included below. They can also be found in the README in the project repository.

Prerequisites

1. On Spotify Web Developer Console, create a new account
2. Create a new app
   2.1 Connect to Android project (Package name and SHA1 fingerprint)
   2.2 Add Redirect URI: edu.utap.tuneder3://callback
   2.3 Enable Web API and Android SDK
3. Download Spotify App on testing device and log in under the same Spotify account

Build and Run

4. In the CLIENT_ID fields, replace the value with the Client ID retrieved from the field in the Spotify Developer Console
5. On initial run, you will be asked to grant Tuneder access (Web API access)
   5.1 You will also have to grant permission when you first navigate to Liked Songs screen (App Remote access)

GitHub Repository

Link to repository: https://github.com/ut-msco-s24/tuneder

Most regrettably, this entire project was written and stored locally. The few commits present in the repo represent the full codebase. As a developer, it is a chronic bad habit of mine to primarily work locally. I attribute it to an often-detrimental perfectionist instinct that prevents me from putting my name on something incomplete and "sending it out to the world". Though no major hardware crashes or loss of unsaved code were encountered in this project, I've experienced enough instances to dissuade me from this behavior, but it's still something I'm working on to become a better developer.

Code Line Count

```
    Language          files       blank      comment        code
-------------------------------------------------------------------------------
    XML                 42          81          50            14318
    Kotlin              24         346          78            1409
-------------------------------------------------------------------------------
    SUM:                66         427          128           15727
```

The following files' totals were excluded from the count:

- MainActivity.kt and activity_main.xml – mostly boilerplate
- SplashActivity.kt and UserService.kt – created following a tutorial (source documented in files)

Demo

The video demo can be viewed here:
https://drive.google.com/file/d/1k40FbJmD2J3S0dsROuR7E1zh72MRhJxZ/view?usp=sharing