Spotify Party Radio

May-2020

Overview

Spotify Party Radio is an app that lets you listen to music with your friends. Every user has their own station, playing their choice of music. Your friends can listen to your station live with you. With your permission, they can even queue up songs for your station.

This document describes a minimal feature set and requirements to prove out the idea for the app.

Approach

The design is premised on users having Spotify premium access for its music catalog and playback capabilities.

Each user's station plays music for any listeners all the time. A user can DJ their station live, or have a robot DJ pick tracks from their favorites when they're not around, or give permission to others to play tracks.

The app is serverless, using a real-time shared cloud database like Firebase's Cloud Firestore with all responsibility for modifying shared objects distributed among clients.

The first client version is for the web, published as single-page Angular app. Later, the app could support other music services (like Apple Music) and clients (like native iOS and Android apps).

User Experience

We describe the user experience in four UI components. On mobile you'd navigate between then. On desktop web we can just show them all on the screen at once.

Auth / Sign in

Users authenticate with their Spotify account. When you authenticate for the firs time, your station is automatically created. The app should automatically renew its access tokens as needed.

Stations

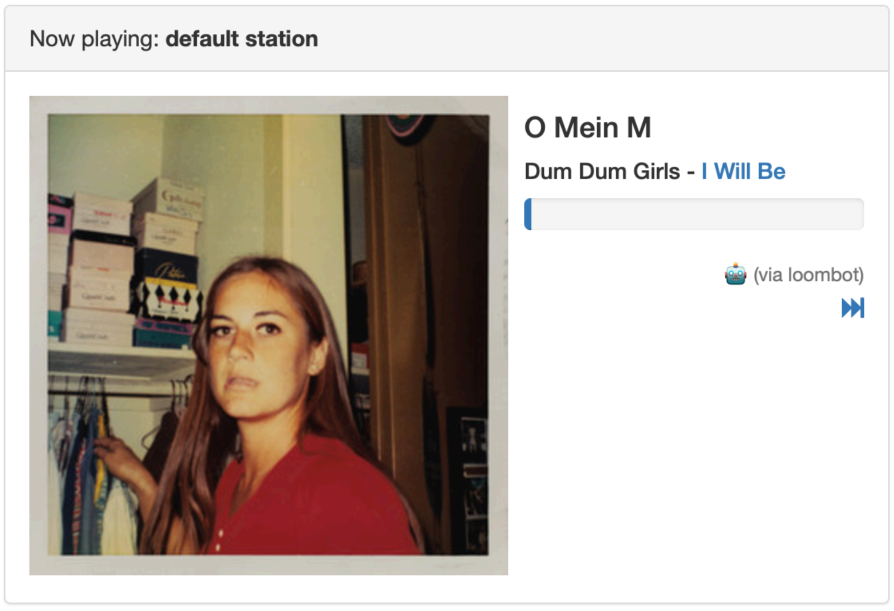
The user starts by picking a station. The app defaults to their own personal station, and makes available all the other stations the user could switch to (to start, this is just all stations, period).

The browser URL should represent the currently-selected station.

Now Playing

As soon as the app opens, music from the user's own station starts playing. The component should always be playing the first track in the queue. There is no "Pause" button (though there might be a "Mute" button, one day).

We show the user the album art and track info of the current track, plus a button to skip the track, if they have access to skip. We also show the user (or robot) who selected the track.

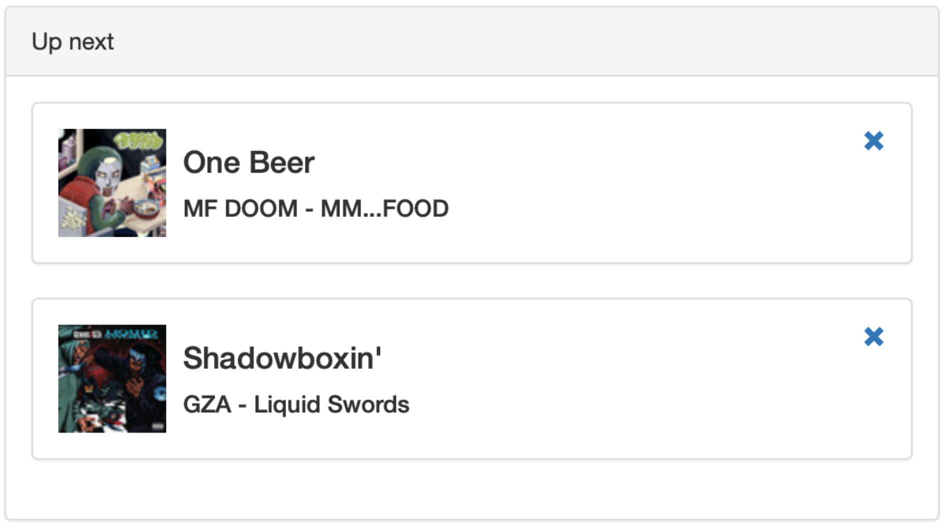


When a track finishes playing, it's removed from the top of the queue and added to the station's song pool (if skipped, it does not go into the song pool). The next track from the Up Next queue then begins playing.

When the queue is empty, a Robot DJ picks a track from the song pool and adds it to the Up Next queue.

Up Next

Shows a queue of all the tracks scheduled to play, in order, after the track now playing. Tracks can be removed from the queue if you have access to do so. Tracks are always appended to the end of the station's Up Next queue.



Up next

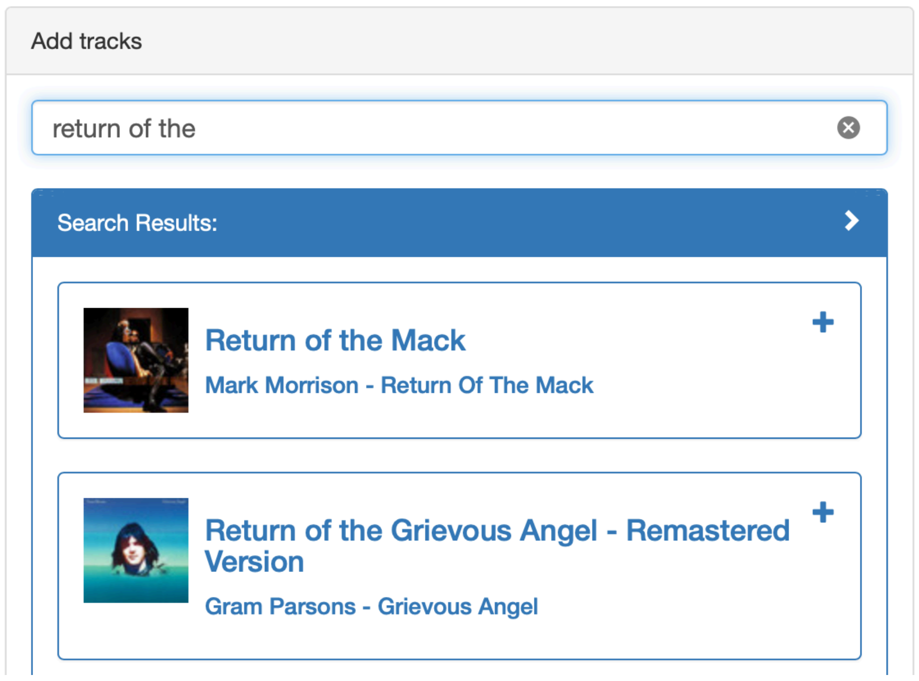
One Beer
MF DOOM - MM...FOOD

Shadowboxin'
GZA - Liquid Swords

Add Tracks

Users always have permission to add tracks to the queue for their own station. They may also have permission to add tracks to other stations' Up Next queues.

The Add Tracks component has a simple search feature that lets you append to the queue from search results.



Add tracks

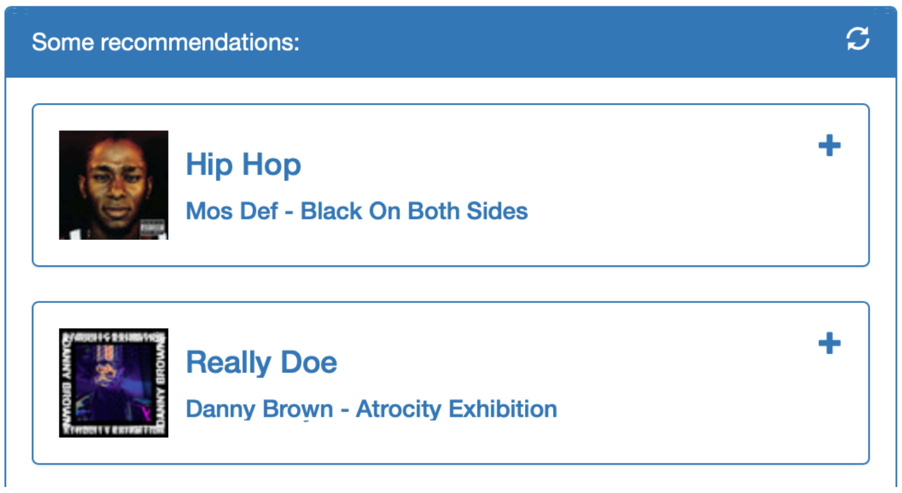
return of the

Search Results:

Return of the Mack
Mark Morrison - Return Of The Mack

Return of the Grievous Angel - Remastered Version
Gram Parsons - Grievous Angel

The component could also let you add songs from other lists: the station's song pool, recommendations based on the current queue, or your "On Repeat" playlist, for example.


Some recommendations:

Hip Hop
Mos Def - Black On Both Sides

Really Doe
Danny Brown - Atrocity Exhibition

Data Model

A proposed approach pseudo-modeled in a conceptual hierarchy. Imagine this fitting into Cloud Firestore data structures.

```
▷ Users/Stations
 ▷ loombot
  ▷ loombot's song pool
   ▷ song 1
   ▷ song 2
   ▷ song 3
   ▷ song 4
  ▷ loombot's playlist
   ▷ 0: song 3
    ▷ song 3's expiration time
   ▷ 1: song 4
    ▷ song 4's expiration time
   ▷ 2: song 1
    ▷ song 1's expiration time
 ▷ pietro
  ▷ pietro's song pool
   ▷ song 2
   ▷ song 3
   ▷ song 4
  ▷ pietrorea's playlist
   ▷ 0: song 3
    ▷ song 3's expiration time
   ▷ 1: song 2
    ▷ song 2's expiration time
```

Most of these items should be self-explanatory, and the details of how they're modeled may vary depending on the data store that is chosen.

Some of the non-obvious elements are:

`user's playlist`

> A first-in first-out queue of songs. The DJ (human or robot) can append or remove songs. Playback automatically removes the first song when it finishes playing (passes its expiration time).

`song pool`

> The pool of songs that are available to add to the station by the robot DJ service. The pool is populated with any song that finishes playing in full. It should be pruned to a fixed maximum size by recency.

`song's expiration time`

> When a song is added to an empty playlist, we give it an expiration time, which is the current timestamp plus the song's duration. When a song is added to the end of a playlist, its expiration time is the expiration time of the song ahead of it in the queue plus its duration. Clients check this expiration time to decide whether to play the song or remove it from the playlist. See playback operations section of this document for more details.

More Implementation Details

All application state is managed directly through the data store. The advantage of this approach is that all clients can act as if they only need to observe changes or publish new changes to the local data store. The data store service handles publishing those changes to other clients and keeping them all in sync.

Authentication

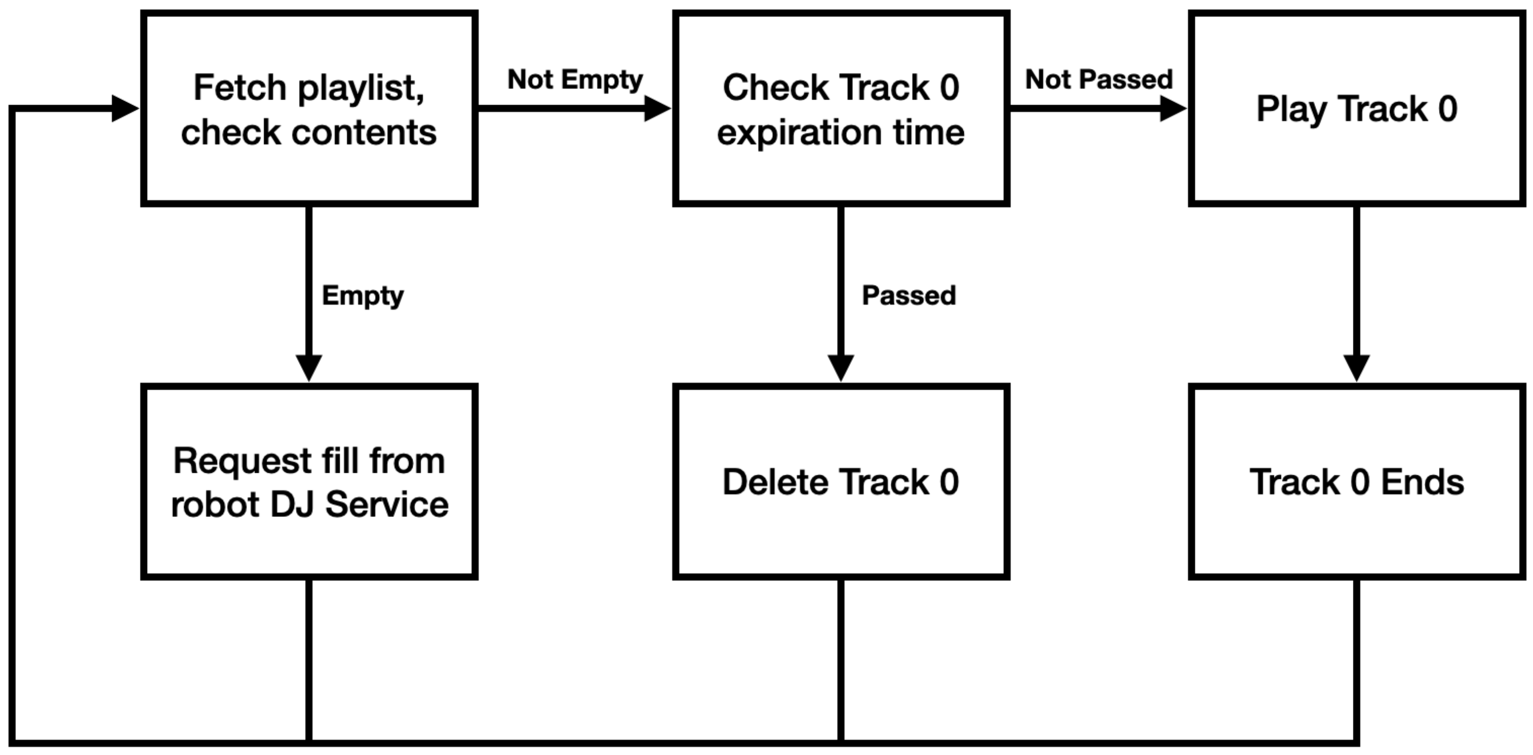
Spotify docs on authentication specify a few different auth methods. "Implicit Grant" is the easiest to implement and gets you access to the user's web player resources, but we need to use "Authorization Code" to allow for automatic token refreshes so the user isn't asked to sign in again every 45 minutes.

Playback Operations

Spotify's Player API is the quickest way to play music through the signed-in user's account. It can control any active instance of Spotify the user has active, such as the app on their desktop or on their phone. Separately, the Spotify Web Playback SDK could be implemented, which would remove the need for the user to have the Spotify app running elsewhere, and actually creates a new active instance of a Spotify player right within the browser.

Broadly, the player in the app is responsible for monitoring the queue for the current station, playing the top track in the queue (starting at the correct offset time), removing the top track when playback has finished (when the track's expiration time has passed), and adding tracks to the queue (via "Robot DJ") when it is empty.

Collectively, all clients running the playback service control the state of the playback queue. The basic playback loop revolves around inspection of the first track (track 0).


Fetch playlist, check contents
Not Empty
Check Track 0 expiration time
Not Passed
Play Track 0
Empty
Passed
Request fill from robot DJ Service
Delete Track 0
Track 0 Ends

In this flow, the client listens to all of track 0 once playback has started. After the track ends, the client evaluates the current "track 0" in the playlist, and either plays it or removes it from the front of the queue.

Robot DJ 🤖

A simple, hacky approach to stay serverless: When a client recognizes that the playlist is empty, it attempts to automatically add a new track to the queue. If the track it appends winds up in a position other than 0 (meaning another client go there first), it backs off the append transaction. If there are no active clients, no one is actually listening, and the queue stays empty until someone shows up.

A simple approach is that Robot DJ just picks a track randomly from the pool of previously played songs on the station. We need to find a way to trim that list so it doesn't grow out of control. In theory we should only keep a pool of the last 250 tracks played.

Helping the User Add Tracks

In addition to facilitating search, the app will need to present the user with suggested songs they might want to add to the Up Next queue. Some easy wins for suggestions come via these APIs:

- Search
- Seeded Recommendations based on the current playlist
- User's Top Tracks
- User's Playlists
- Song pool of the current station

Managing the Up Next Queue

Tracks are always appended to the end of the queue. Any newly added track is given an expiration time, which is when we presume playback has finished. The expiration time is dependent on the prior tracks in the queue.

Note that the skip and delete operations require updating expiration times on all subsequent items in the playlist.

Future Considerations and Notes

We could insert our own content between songs. Music service licenses would preclude paid ads but not other original content, but you could let users record their own short messages to play between songs.

Given a "Now Playing" API you could in theory sync the user's station to music they are currently listening to, even *outside the app*. So my regular Spotify sessions could become live broadcast sessions to any other user interested in hearing what I'm hearing.

Tracks can be correlated between Apple Music and Spotify, in theory, using the ISRC value, so the service could work across music services.

Feedback from listeners is probably critical, as simple as “Likes” for playing tracks or complex as station chat.

See Also

This app is inspired by the defunct turntable.fm.

Stationhead uses a very similar approach, and even includes a collaborative voice clip feature. They’ve gotten some press.

A Spotify engineer’s hack to deliver a collaborative (voting-based) listening experience.