song-rec.me

song-rec.me is a non-algorithmic way to getting spotify song recommendations (although it will never come close to the feeling of a friend recommending a song in person, or discovering a song from the radio or in a tv show/movie). But the true purpose of this app is for myself to develop a full stack app and deploy it and then to have a proper application in my portfolio.

It's a web application that allows Spotify users to create a playlist, which generates an accompanying link. Users can then share that link with friends or place it in their social media bios, etc. Their friends can click that link, search for a song, and send that song to the user's playlist along with an accompanying message. The playlist owner, if notifications are enabled, will receive an email saying, "User sent you a song: <song info>" with a link to listen to the track on Spotify, as well as a link to their song-rec.me page.

Through the song-rec.me site, the owner can click on their playlist and see all the tracks in it along with who sent them and their messages. They can also react to recommendations, and the person who recommended the song will be notified by email if there is a reaction.

Isn't this just collaborative playlists? not really, with collaborative playlists, anyone in the world (or anyone in the playlist if private) can see who added which songs, but with this it will appear as just the owner of the playlist is adding the songs, so only the only and the sender will know, to encourage people to send recommendations. The sender can add a message with the recommendation. The owner can react to the recommendation. And the owner can get notified when a song is sent to them, and collab playlists don't have this feature.

Enginnering Requirements

- Develop a full stack app, using the MEAN stack

- Deploy the app to the world
- Become more familiar with the MEAN stack (mainly the MEN part)
- Document my development journey, in this journal/doc
- No use of ai coding tools, like copilot and whatever, I will be allowed to consult with an AI (I've only ever used chatGPT) for best practices when it comes to it (e.g are these database queries fully sanitized from exploits etc)
- DO NOT FALL VICTIM TO FEATURE CREEP
- ACTUALLY COMMIT AND FINISH THE PROJECT

Application Requirements

- Users login with spotify
- Users create a playlist which generates a shareable url (maybe even a QR code)
- Users can click a link search a song and attach a message and send it to the playlist owner (sends the song to the playlist and the owner can view the recommendation in the song-rec site)
- User can turn on notifications, so they receive emails when they receive recommendations
- Users can react to recommendations

Tech Stack & Deployment

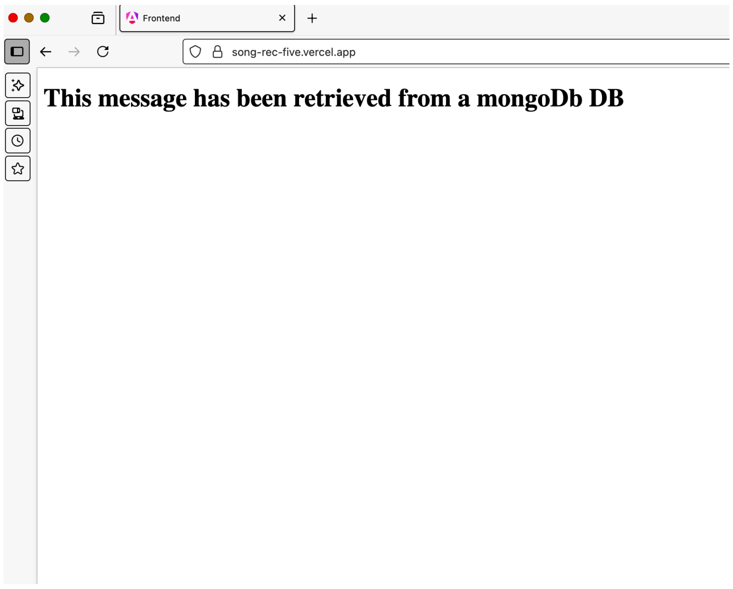
- **Frontend:** Angular, and host the repo on GitHub. Then link my GitHub to Vercel, and Vercel will deploy the frontend easily.
- **Backend:** Node.js (& Express) backend, and again host the repo on GitHub. Then link the GitHub to Heroku, and Heroku will deploy the backend easily.
- **Database:** MongoDB, and use MongoDB Atlas to host the data.
- **Spotify API:** Spotify API is well-documented, and all the features I want to provide are possible with the Spotify API.
- **Mailer:** Use Resend to send the emails.
- **Domain:** Buy song-rec.me on Namecheap, so I can have a memorable and intuitive name. This domain can also be used for sending mail, e.g., `noreply@song-rec.me`.

Things to Factor in

- Of course rate limiting & security
- what if a user clicks on the link, but they haven't logged into spotify yet, they must log into spotify and then the redirect should take them to the playlist that they initially clicked on.
- Auth guards too

Friday July 4th

- Initialised the angular app and connected it to vercel and deploy it (as a test to see how to deploy the frontend)
- Initialised a node js app and connected it to heroku and deployed it
- Set up a mongoDb database and stored a test text message
- The node.js backend is connected to mongoDb and set up a route that retrieves the message from the database and then returns the message in its response
- The frontend sends a request to the backend and then displays the message that was stored in the database

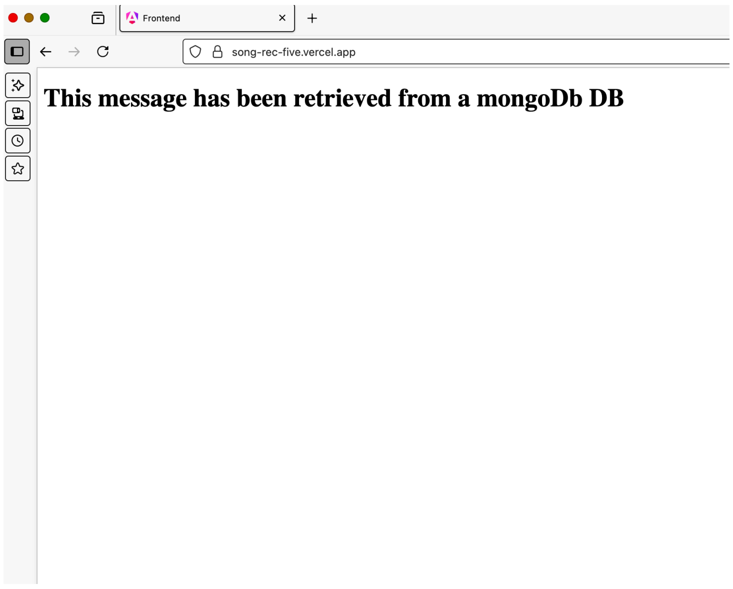

Frontend
song-rec-five.vercel.app

This message has been retrieved from a mongoDb DB

Deployed frontend that displays a message retrieved from the backend which was retrieved from a database

Saturday July 5th

- Purchased the song-rec domain from namecheap
- Setup a Resend (got API KEYS) and connected my domain to it, so I can now have an automated email sender


Frontend
song-rec-five.vercel.app

This message has been retrieved from a mongoDb DB

Sent an automated email from my backend (& domain) to my personal email

- Probably going to use JWT's `https://www.youtube.com/watch?v=7Q17ubqLfaM&ab_channel=WebDevSimplified`

  A JWT is a signed token that verifies a user's identity. When a user logs in, I retrieve their Spotify ID and sign it into a JWT using a secure 32-character secret. This JWT isn't encrypted, but it's signed, meaning it can't be tampered with—if someone modifies it, the signature won't match.

  I send this JWT back to the client in a secure HTTP-only cookie. On every request, the client automatically sends the cookie, and the backend decodes the JWT using the secret. This way, I can identify which user is making the request, and I can trust the data because the token is cryptographically signed.

- to get the users access and refresh tokens, need to redirect the user to the spotify auth page
- Then add a middleware to routes that verifies if the token attached with the requests is legit

Monday July 7th

- Search feature added, where we query Spotify's search endpoint with the user's access token in the header, then in the query params a query term (the song the user is searching for). I'm also attaching type="track" so the search result returned is only tracks (songs).

Tuesday July 8th

- Added sending a recommendation: post a request that searches the owner of the playlist in my MongoDB, gets their access token, then adds the sent track to the playlist using the owner's access token. Then generate a random recommendation ID, and store that in a recommendations playlist, with the rec ID, sender ID (sender's Spotify ID), playlist ID, and message (the message the user attached with the recommendation).
- Get your recommendations: simply look up in the recommendations table all the rows where sender ID is your ID.
- Get your recommendations on a specific playlist: simply look up in the recommendations playlist all the instances of that specific playlist ID.
- Add reactions: for a recommendation ID, send it with an emoji, then in recommendations insert into the recommendation ID entry the reaction.

TO DO ON BACKEND

- Email Sender: sender email when a recommendation or reaction is sent
- Refresh token: In all instances where an access token is used, check if the roken needs to be refreshed and if so refresh it.