CS 35L Full-Stack Project: MyTerrarium

Angela Tan, Cyrus Asasi, Helen Feng, Ryan Chaiyakul, Ryan Oh

Purpose

Our project is inspired by our own experiences searching for the perfect studying environment on campus as many areas may be physically ideal, but have distracting ambient sounds. While students already wear noise-canceling earbuds with music, there still exists a niche in which music is too distracting or a different kind of white noise is preferable. By integrating a variety of audio sources into a single web application, we hope to simplify the process of mixing your own soundtrack for your specific occasion.

Github: https://github.com/ryanchaiyakul/cs35l_project

1. Technological Overview

Our web application is designed as a simple client server design where users interact with the application through their browser and HTTP requests.

The server itself was implemented with React, Quart, and PostgreSQL for the frontend, backend, and database respectively.

1.2. Frontend

The frontend was developed in React as this was the framework our members were most familiar with.

Components of our application were developed in individual component files. These component functions were exported and imported in their respective files in which they were rendered. App.js was the primary file which rendered the main pages of the website using React Router.

Two major libraries that were used included styled-components, which allowed smooth organization of CSS, and Axios, which enabled fetches to back-end endpoints.

1.2. Backend

In the current implementation, only the backend server runs and serves the prebuilt React application (built by "npm run build") in order to consolidate our application and to streamline the authentication process. This allowed any authentication request to be handled in one place rather than deal with many redirects from the frontend.

Quart was chosen as the framework of choice as our member in charge of the backend was most familiar with Flask, another Python HTTP framework, and Quart is an attempt at creating an asynchronous Flask framework.

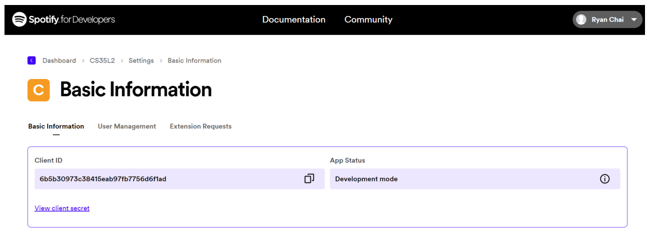
In order to simplify the prototyping process, the server is hosted locally on "http://localhost:4000" which was the arbitrary port provided in the suggested config.py. Therefore, this project is currently restricted to a single device, but steps to expand the project to accept clients from other network connected devices is explored in a later section.

For our application, the backend was in charge of two primary tasks: authenticating and requesting information from the Spotify API and storing and retrieving uploaded MP3 files.

1.1.1. Spotify API

In order to interact with the Spotify API, an authentication token must be requested for every user who wants to use the application. However, there were many complications that arose from this necessary task.

Spotify requires applications to be registered and the default "developer mode" only allows the user associated with the registered application itself to interact with the API. Therefore, every member of the project has to register their own personal Spotify application and use their respective client ID and secret when configuring the server.



Basic Information

Client ID: 6b5b30973c38415eab97fb7756d6f1ad

App Status: Development mode

Fig 1. Basic information tab where client ID and secret are location of registered Spotify application for user "Ryan Chai"

For every stored authentication token, the backend must check the validity of them before making requests and perform the proper steps to refresh them if they have expired.

An extensive caching system was implemented as a necessity as, through testing, we realized that the Spotify API aggressively rate limited developer mode applications which meant that requests had to be cached for them to be returned in a timely manner and avoid rate limiting.

To add onto the user experience, we fetched a list of playlists and their corresponding ID's from the user's Spotify account using Spotify API. This was then parsed and handled in the front end to present a list of playlists that the user could switch between, which would replace the Spotify Embed on the homepage with one of the user's selected playlist. This was done using an inline frame that pulled from a Spotify page generated by the playlist ID chosen by the user.

To personalize the data received from the Spotify API, the server places a "user_id" cookie which is referenced by the frontend when making HTTP requests to the backend endpoints to isolate the authentication and direct Spotify requests to the backend only.

1.1.2. MP3 Files

The backend has three endpoints related to handling MP3 files: "_upload_audio", "_get_audio_metadata", and "_get_audio_data".

In order to upload audio data from the frontend to the backend, a POST request is sent to "http://localhost:4000/_upload_audio" with an HTML form with the respective format: the MP3 file as "audio", the title as "title", the user_id cookie as "owner_id", and the tag (selected from the dropdown) as "tag". The backend receives, validates, and inserts the audio file with the respective metadata into the database.

To implement the search feature, the endpoint "http://localhost:4000/_get_audio_metadata" with the parameters "title" and "tag" allow the frontend search to receive the list of uploaded MP3 files that match the requested metadata.

To implement the playback feature, the endpoint "http://localhost:4000/_get_audio_data" returns the entire MP3 file with the "title" passed in.

1.3. Database

The preferred database that the backend uses to store the Spotify authentication tokens and uploaded MP3 files is PostgreSQL. However, due to the variety of operating systems, a JSON database with explicit file storage for MP3 files is available so that all developers could locally test the web application during development.

2. Features

Our application has two categories of features: audio and statistics.

2.1. Spotify Embedded Player

A Spotify Embedded Player is available to the user which allows them to select a playlist linked to their account to be played on the web application itself after authenticating.

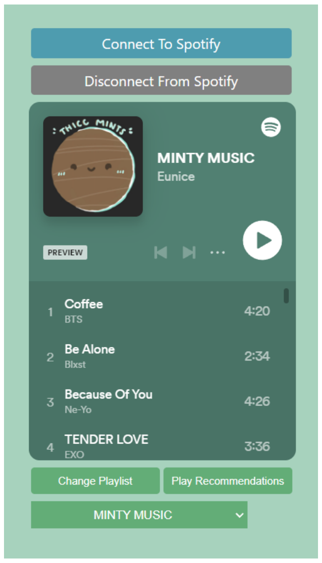


Fig 2. Spotify Embedded Player which is playing a personal playlist chosen from the authenticated user's library.

2.2. Recommendations

Closely linked with the Spotify Embedded Player, there is a button which creates a playlist of 10 songs that attempts to match your listening history and changes the player to the newly created playlist.

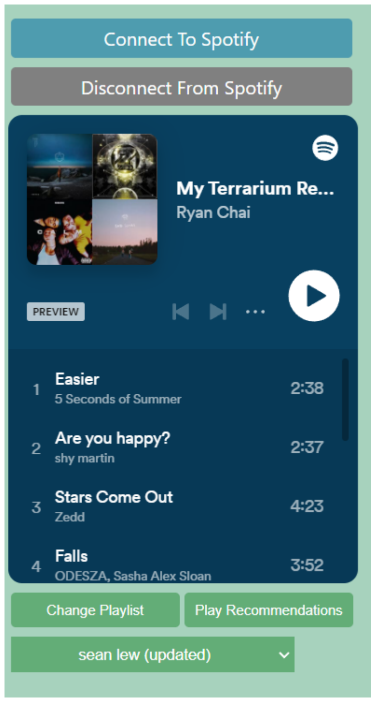

Connect To Spotify
Disconnect From Spotify
My Terrarium Re...
Ryan Chai
PREVIEW
1 Easier 5 Seconds of Summer 2:38
2 Are you happy? shy martin 2:37
3 Stars Come Out Zedd 4:23
4 Falls ODESZA, Sasha Alex Sloan 3:52
Change Playlist
Play Recommendations
sean lew (updated)

Fig 3. The Spotify Embedded Player displays a newly created playlist called My Terrarium Recommendations for the authenticated user.

2.3. Upload

The upload form is accessible from a button in the navigation bar which allows the user to upload a local MP3 file into the backend with a custom title and general tag. Errors and successes are displayed to the user after pressing upload.

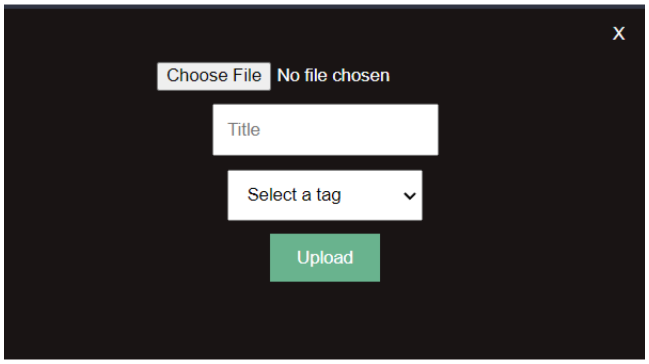
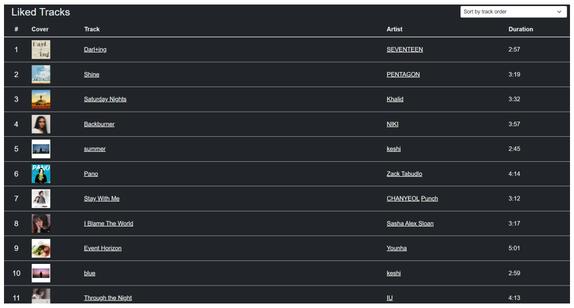


Fig 4. Upload form displayed to the user on the homescreen.

2.3. Liked Tracks

After authenticating with the Spotify API, there is a button in the Navigation bar which brings up a sortable list of your liked tracks with an audio preview button to the left of each track.



Liked Tracks

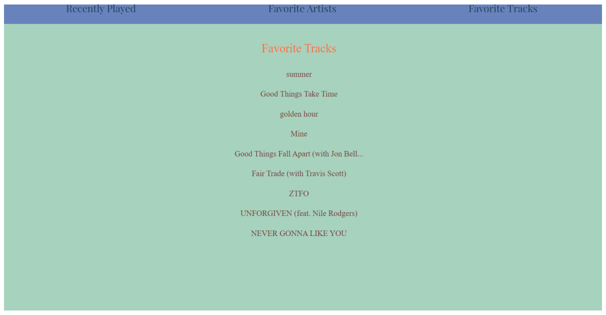
Sort by track order

#	Cover	Track	Artist	Duration
1		Darl+ing	SEVENTEEN	2:57
2		Shine	PENTAGON	3:19
3		Saturday Nights	Khalid	3:32
4		Backburner	NIKI	3:57
5		summer	keshi	2:45
6		Pano	Zack Tabudlo	4:14
7		Stay With Me	CHANYEOL Punch	3:12
8		I Blame The World	Sasha Alex Sloan	3:17
9		Event Horizon	Younha	5:01
10		blue	keshi	2:59
11		Through the Night	IU	4:13

Fig 5. Liked tracks of the authenticated user.

2.3. My Stats

Like the previous feature, this feature is selectable from the Navigation bar and presents other relevant information from Spotify such as “Recent Tracks” or “Favorite Artists”.



Favorite Tracks

summer

Good Things Take Time

golden hour

Mine

Good Things Fall Apart (with Jon Bell...

Fair Trade (with Travis Scott)

ZTFO

UNFORGIVEN (feat. Nile Rodgers)

NEVER GONNA LIKE YOU

Fig 6. My stats of the authenticated user.

2.3. Search

A search feature is hidden in a hamburger menu accessible in the bottom left corner which allows searching for uploaded tracks by title or tags. When a user is interested in a track, they can press the plus button to add it to their current “playlist” which allows them to play it from the home screen after closing the menu.

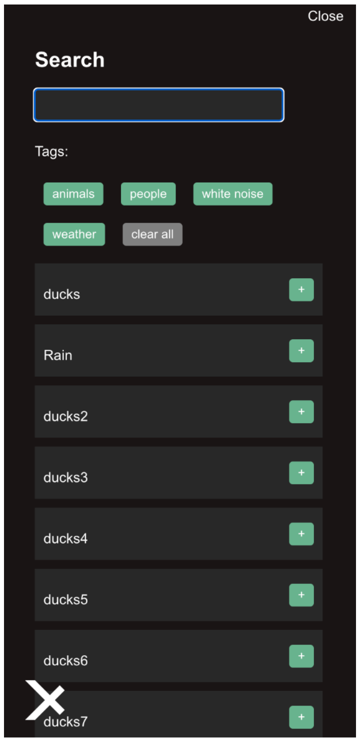


Fig 7. Example search bar of a user with a lot of duck MP3 files.

2.3. Playback

After selecting songs from the search feature, these songs will show up under “My Audios” with a play button, volume control, and a way to remove the track from the playlist. Once started, the MP3 file will loop as it is intended to be an ambient sound.

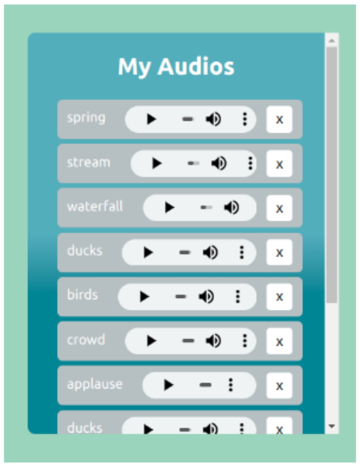


Fig 8. Example playlist of MP3 files that can be toggled and controlled individually.

3. Contributions

3.1. Angela Tan

- Frontend: Homescreen, upload form, search menu, styling

3.2. Cyrus Asasi

- Backend: Quart server, PostgreSQL, JSON database, Spotify data fetching, caching, User authentication, api endpoint implementation.
- Frontend: Connect and disconnect buttons, Liked Tracks (audio playback, sorting, and styling).
- Internal: Source code management

3.3. Helen Feng

- Frontend: Spotify embedded player, recommendations, styling

3.4. Ryan Chaiyakul

- Frontend: Audio playback and styling
- Internal: Bug fixing, source code management, presentation and report

3.5. Ryan Oh

- Frontend: My stats page (10 most played tracks, 10 most listened artists, 10 recently heard tracks), styling, fetch recommendations.

4. Challenges

4.1. Cookie and Browsers

After realizing that a browser cookie would be needed so that the frontend could convey the user associated with its session, there was difficulty using this cookie as different browsers had different caching behavior around the cookie. Also, we realized that the cookie was originally linked to /spotify/… instead of the base url itself which meant that the homescreen could not access the cookie to perform the necessary tasks.

To solve these issues, checks of the validity of the cookie were employed and endpoints were shifted so that the cookie was visible to the frontend (as it was integrated into the same port).

4.2. Serving The Frontend From The Backend

Initially, we planned to run our application on two separate ports, where the frontend would run on localhost:4000, while the backend would be served from localhost:3000. This brought many limitations including: not being able to efficiently redirect the user from the frontend to the backend endpoints “/connect” and “/disconnect”, and not being able to access cookies from the differing domains. We resolved this issue by serving the entire frontend from within quart, all on localhost:4000. This can be done by first running the frontend to produce an html file, then serving this file through quart using jinja templating.

4.3. Spotify Rate Limiting

Throughout the prototyping phase, we had frustrating experiences where the backend requests inexplicably failed. After a lot of debugging of browsers, operating systems, and database options, we realized that it was the Spotify API itself rate limiting our requests.

Like we mentioned before, we solved this issue by adding extensive caching to any direct Spotify API requests.

5. Additional Features

5.1. Exposing The Backend to The Internet

Currently, the project is only usable locally which is a crippling limitation in the usefulness of the program. In order to make this project more usable, it must be ported from localhost to being ideally served under some domain so that users can access the application easily.

5.2. Presets and Saved Combinations

Presets of specific Spotify playlists and audio files could be provided to the user to give an example of the power and a starting baseline to create their own personal amalgamations.

Also, once they perfect their environment, there currently is not a way of saving the exact state of the files, volumes, and Spotify playlist which would increase the quality of life when trying to set up the application.