Playlist Archiver
Final Presentation

Presentation Contents

Overview

Prototype

Format of Archive Files

Primary Features

Secondary Features

Tech Stack / Software Requirements

Accessibility / Hardware Requirements

Project Timeline

Dropped Features

Codebase + Accessing The Web App

Overview

The Playlist Archiver is React-based web app that will allow you to input a link for a Spotify playlist and in turn be able to download an archive file containing all the relevant information of each specific song within the playlist in json (Javascript Object Notation) format. Since playlists are often deleted or lost by the creator, it seemed like a helpful tool for those who wish to hold on the content of playlists they did not create or exist on accounts they no longer use.

When the user provides the web app with a link to any public Spotify playlist, this link is then used to retrieve all of the relevant playlist data from the Spotify API. From there the data is processed using different data structures and then formatted into a json file, which is made available for download on the website. This allows users to save the data of any given playlist to their own machine, which could be useful for people who use streaming services other than Spotify, or people who would like to have a permanent copy of the playlist without fear of it being deleted or made private by its original creator.

Prototype

The final version of the Playlist Archiver will be accessible via a URL like any other website, but to validate the idea before creating a frontend, a prototype was created which operated locally from the command line.

```
nayatrov@ariahas-MBP naya-utils % node sp-pl-archiver.js
Enter the Spotify playlist URL or ID: https://open.spotify.com/playlist/034XigRF
v5CSbUpNccqcqY?si=825bcbc463a4449a
Playlist data saved to Halcyon Days-playlist.json
```

As you can see in the sample output above, the prototype was a single javascript file and run via node.js similar to how python files are ran from the command line. After running the file it prompted the user for a link, and then after successfully saving the file it outputs the success message seen at the last line of the sample output. “Halcyon Days” was the name of the playlist, which is appended to the beginning of the output file name.

Format of Archive Files

To the right you can see the resulting json file from the code ran in the last slide. Each individual song within the playlist is stored as its own Javascript object with the four properties of title, album, artist, and spotifyLink. They are stored within the json file in the same order they appear within the playlist.

{} Halcyon Days-playlist.json > ...


```
[
  {
    "title": "I Need A Forest Fire",
    "album": "The Colour In Anything",
    "artist": "James Blake, Bon Iver",
    "spotifyLink": "https://open.spotify.com/track/0TLAptKgYxe7F0KewWH6X6"
  },
  {
    "title": "Avril 14th",
    "album": "Drukqs",
    "artist": "Aphex Twin",
    "spotifyLink": "https://open.spotify.com/track/1uaGSDFsLdReQgg8p7Obwh"
  },
  {
    "title": "Reality Surf",
    "album": "333",
    "artist": "Bladee",
    "spotifyLink": "https://open.spotify.com/track/6HJszgJO19tAKff7X40ggp"
  },
  {
    "title": "MOJABI GHOST",
    "album": "DATA",
    "artist": "Tainy, Bad Bunny",
    "spotifyLink": "https://open.spotify.com/track/4eMKD8MRroxCqugpsxCCNb"
  },
```

Primary Features

01 Text box which accepts a URL as input in order to pass it onto the Spotify API endpoint.

02 “Archive Playlist” Button which uses the text from the input box to fetch relevant data about the playlist located at the URL via the Spotify API. The fetched data is then automatically formatted into a json file. From there the Dropbox API is called to write the created json file to an account where it will be readily available for download.

03 “Download Playlist” Button that uses the Dropbox API to retrieve their archive file and download it to their personal device.

Secondary Features

01 Both of my presentations for this project are hosted within the web app and readily accessible from the front page of the site.

02 Created a component that links to my Github account, portfolio website, and my email address to place at the bottom of the front page.

03 Implemented a payment form that allowed me to charge users for each file downloaded via the Square API. (Not present in the current version)

Tech Stack / Software Requirements

Backend

Node.js and Javascript are being used on the server-side (backend) for retrieving data from the Spotify API, processing it, and then sending it to Dropbox via their API, where it's made available for download on the frontend.

Frontend

For the website portion (frontend) I am using React for the foundation, and CSS to style it. You can think of React as a Javascript framework that is essentially HTML with much more functionality. I am also using Next.js for better SEO (search engine optimization) thanks to it providing server-side rendering and static website generation.

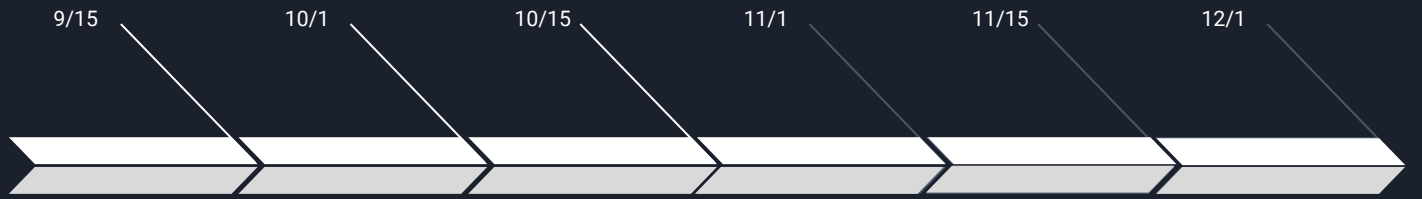
APIs / Node Modules

The Spotify API was originally the only API used for this project, but it was decided that using the Dropbox API was the best way to handle archiving the data and then making it easily downloadable at a set endpoint. The Square API was also used for processing microtransactions per each download, but this functionality was not included in the version of the web app being presented today. The only node module used was 'dotenv' which helps obscure sensitive data like API keys.

Accessibility / Hardware Requirements

This project will be accessible via any device that can connect to the internet and download .json files. All that is required is a link to a public Spotify playlist. The project is hosted on Vercel.

Project timeline



Topic Selection

Decided on my project topic and which language would be best to implement the project.

Prototype + Structuring

Created a prototype for the main project functionality, and then began working on structuring a frontend around it.

First Presentation

Preparing and giving the first presentation while also making progress on implementing the main functionality of the project.

Frontend + Adjustments

Reworked the initial prototype to fit our frontend, implemented the Dropbox API to make things simpler, and finished the structure and styling of the frontend.

Cleaning Up + Experimenting

The majority of the project was completed by this point, from here I was mostly fixing small issues and experimenting with added functionality like micropayments and integrations with other streaming services.

Final Presentation

Presenting the finished product in its entirety as well as turning in all documentation and the codebase. The project is now accessible via URL.

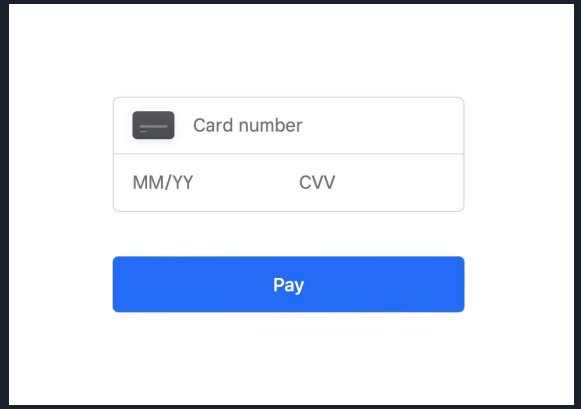
Dropped Features: Alternate Streaming Services

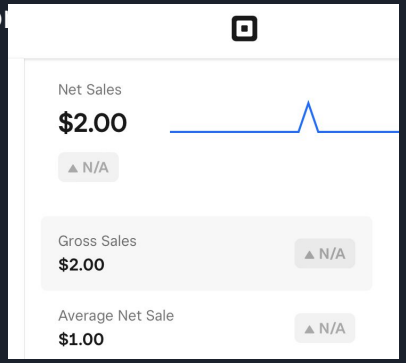
I had originally hoped to implement functionality to archive playlists from streaming services other than Spotify, such as Apple Music or Amazon Music, but this ended up being a far more difficult task than I was prepared for, and I eventually had to scrap the idea entirely.

Spotify's API is significantly older and much more well documented than any of the streaming alternatives, which means that any issue I ran into while implementing the Spotify API was relatively easy to solve since almost any issue out there had already been experienced by someone else and solved within their documentation or forums. It was the opposite when trying to implement Apple or Amazon. Both of their streaming services are relatively new, and not a big deal compared to the other departments of both of the very large companies. There was a miniscule amount of documentation or support for their streaming APIs compared to Spotify, and this is on top of their APIs already being far more complicated to begin with.

Dropped Features: Microtransactions

Once the project was pretty much finished I decided to see if I could somehow charge users per file downloaded. I settled on Square since it seemed to be the simplest, and came with pre-made React components like the one to the right, which popped up when the user clicked the “Download Playlist” button. Once the payment was processed and validated the download request would then retrieve the necessary file from the Dropbox API. Below is a screenshot of the dashboard of my Square account showing that at one point this feature was implemented, and that I paid a dollar fo separate downloads to test it.





Codebase + Accessing the Web App

The code I have so far can be found within the github repository at:

www.github.com/tenjotenge/playlist-archiver

I have deployed the web app using Vercel so you can access it via the link in the top right corner of the Github repository under “About.”