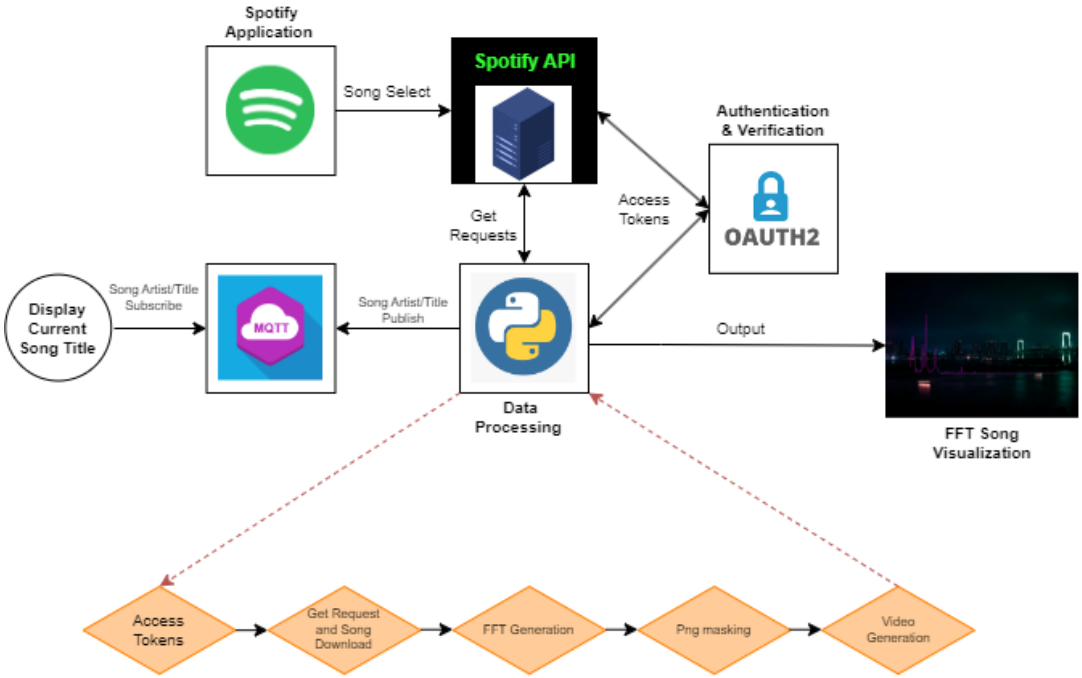
Audio Visualization

By Elliott Meeks

Description:

In my IOT project I utilized the spotify API to create an audio visualization of the current song playing from my spotify account. Additionally, it implemented an MQTT server to publish the selected songs artist and title to the server. This is then accessed by an external device, by subscribing to the server. It can then display this data however it wishes.

Block Diagram



Detailed Description

Spotify Access Tokens

Spotify uses the OAUTH2 workflow to authenticate and verify users attempting to make get requests to the API. This is done in a few of steps

1. Registering your application with Spotify.
2. Obtain authorization
3. Exchange authorization code for access token
4. Handle token expiration with refresh tokens

Get Requests and Song Download

A get request is sent to the Spotify API using the generated access token, the request is returned in the form of a JSON file. I then parsed this file for the songs title and artist. Because I cannot

download the song directly from the JSON file I had to get creative. Using the artist and song title I generated a search phrase and used python to download the song from youtube.

FFT Generation

After getting the audio data, I generated discrete FFT's at 0.1s intervals.

PNG Masking

Every good audio visualizer has a cool background image, however matplotlib, the library used to plot the FFT's, does not support background images. To get around this I implemented blue screen masking. I used the FFT generated PNG to create a mask over my selected background PNG and overlaid the FFT over the top of it (actually pretty complicated and annoying).

Video Generation

I compiled all the PNG's using the Movie library in python and generated the final audio visualization.

MQTT Server

Kind of just a through in but implemented an MQTT server to publish the songs title and artist. This is accessed and displayed by an external node

Reflection

Overall I am pretty happy with what I was able to produce. The highlights were figuring out how to download any song's audio using the Spotify API, and implementing the image masking to get a good background photo. One major downside to my program is run time. There is a lot of image processing and a 60s song takes about 60s to process. This could be improved in many ways. Better code structure and algorithms for one. But one thing we learned about is parallel processing. For one my laptop has multiple cores which I am not utilizing. Additionally, I could send the audio data to another machine for processing. Furthermore, we learned that run time improvement is dependent on the number of parallelizable processes(Amdahl's Law). The cause of the long run time is largely due to the FFT diagram and Image masking. This same process is performed numerous times (>1000) and could easily be performed in parallel.