Technical Guide

Project name: Sham Shazam
Student 1: Georgina Scanlon ID Number: 19392373
Student 2: Rhea Varkey ID Number: 19452962

Date submitted: 06/05/2023

Table of Contents

Motivation

Our motivation for this project is to help record labels and artists keep track of the royalties they are owed when their music is used by content creators, radio, media, and other platforms. It's a significant problem in the music industry, with artists and record labels losing billions of dollars in revenue each year due to unlicensed use of their music. This project aims to address this issue by providing a tool that helps users keep track of the songs they are playing and the royalties they may owe to prevent copyright infringement issues.

As music enthusiasts, we were keen to incorporate our passion for music into our final year computer science project. With the rise of platforms like Spotify and YouTube, we realised the potential for a project that could analyse and identify songs from audio recordings.

Once we got our idea for "Sham Shazam," we decided to use this as our project to showcase our UI and web app design skills and learn new skills like React, machine learning methods and single processing methods. The project could also be used in real-life scenarios where music streaming is involved, whether for personal or business purposes.

In summary, the motivation for our project is to address a real problem in the music industry and provide a useful tool for music enthusiasts and businesses alike.

Research

To start off our project, we first did extensive research on various topics related to music identification, audio processing, and machine learning. We started by researching existing APIs and tools that could classify songs based on audio recordings, and ultimately chose to use the API from AudD. We found it had detailed, easy to understand documentation and the most extensive database of songs. It also seemed to be widely used in a lot of other projects, which gave us confidence that we would be unlikely to run into issues using it.

In addition to the API research, we also explored techniques for denoising audio recordings to improve song identification accuracy. We studied various sound processing methods and evaluated their effectiveness in removing background noise from audio recordings and be able to pick the music in a noisy environment eg. shops and bars.

Furthermore, we reviewed research papers and open source repos related to machine learning algorithms and techniques for audio classification. We also experimented with different machine learning models and techniques to find the most accurate approach for our project.

Overall, our research efforts helped us gain a deeper understanding of the technical aspects of music identification and audio processing, and informed the development of our project.

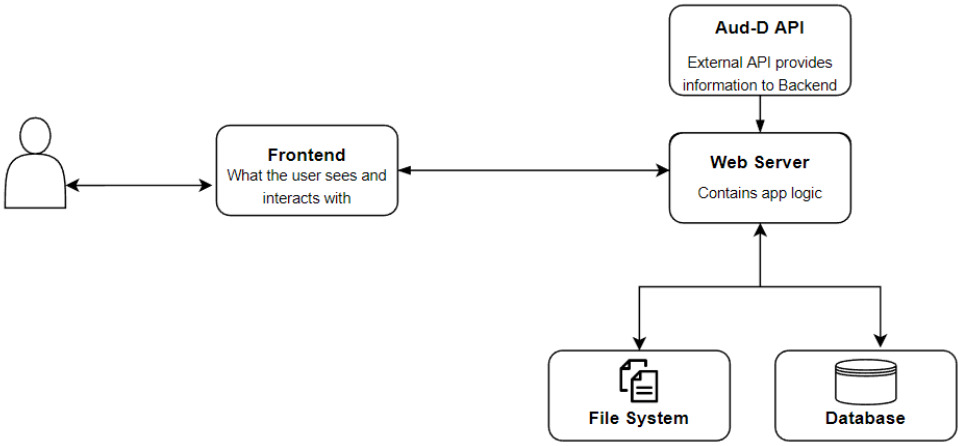
Design

The "Sham Shazam" system is designed to accurately identify songs from audio recordings and provide the user with detailed information about the song, including the artist, track name, and record label as well as the location where the app was used to classify the songs using GPS and Google's geolocation API. The system is built using React and a music recognition API from AudD, to classify songs based on audio recordings. In the backend, we also use Python to denoise the audio.

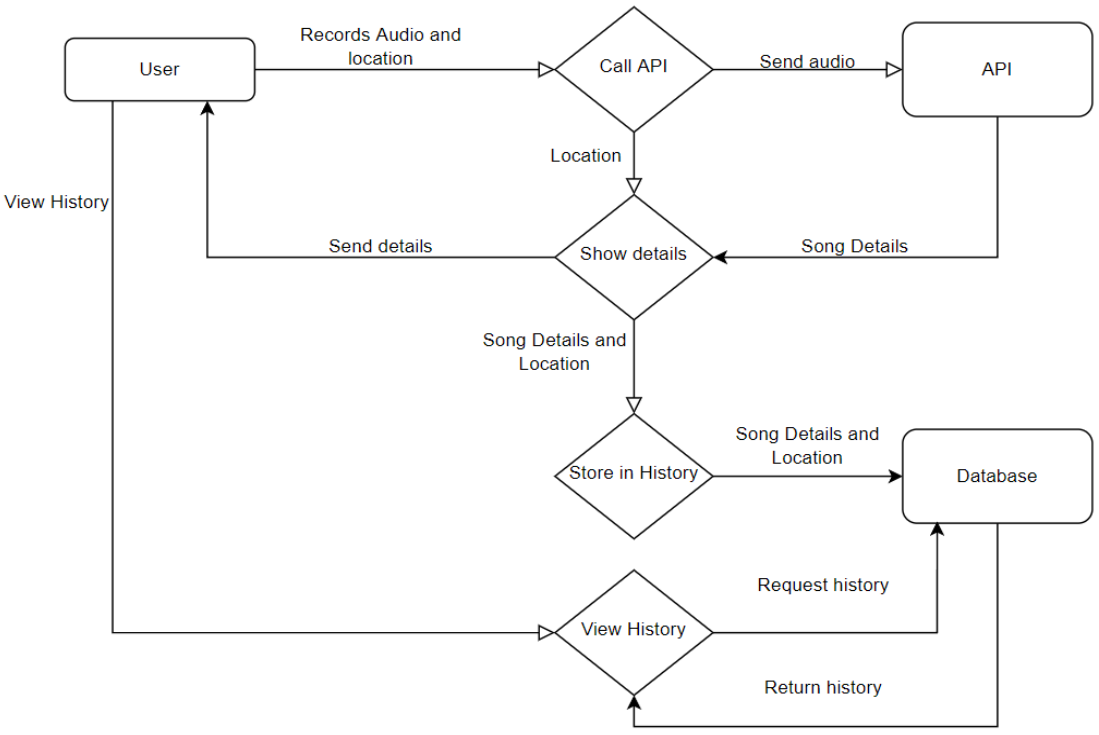
The system architecture consists of several components, including a signup/login component, a recording audio component, and a song history page on the frontend. We use Express JS for the backend and MongoDB for our database. In the backend, we define our database models and handle the audio files recorded on the frontend, in order to identify the song playing.

Login/Signup System:
Our application included a user authentication system to store song histories for each user and ensure that only logged in users could access the application. If a user is not authenticated on the frontend, they will be redirected to a login/signup page. There, they can either make a new account or login to an existing one. When signing up, we have a few checks to ensure that the same email or username isn't used multiple times. We also use a module called bcrypt to ensure that their password is adequately encrypted when stored in our database.
When a user is logged in they are assigned a JSON web token which is used in API requests and by the frontend to prove that they are authenticated. These web tokens also allow users to stay authenticated for a longer period of time, without having to constantly login again.

Identification requests:
In order to identify a song, an identification request sent from the user would be sent from the React frontend into the backend server. The server will take the audio provided, and temporarily store it in the server's file system. A request will be sent to the external AudD API with the audio file. If the API is unable to identify the song, the audio is then sent to our denoising algorithm. This algorithm uses a method called spectral gating to attempt to make the audio signal more clear. It works by using a 'noise' file to estimate which parts of the signal are noisy and removes it. The denoised audio file will then be sent in a new request to the API. The resulting response will then be checked. If the song title in the response matches the most recent song the user saved, then it will be ignored and the previous entry in the database will be returned. This is so that we don't end up with duplicate entries in the database for the same song. If the titles are different, a new entry will be created in the database with the relevant song information and the user and location data sent from the frontend. The new song object will then be sent back to the frontend and the song information will be displayed to the user. If the user does not wish to have their location data saved, it is replaced with a set value. After the song request is complete, the temporary audio file is then deleted.



Song history:
In order to get their song history, a user would send a request from the frontend to the backend server. The server will use the user's id to get all of the songs that were saved by that user and send this back to the frontend. This information is then displayed to the user along with the location data and timestamps which were saved along with the song information. A link to available streaming services for the song is also attached which can help the user to get more information about a song.



The user interface provides a simple and intuitive experience.Users can also view a list of previously identified songs and access detailed information about each song, including the artist, track name, and record label. When storing song details, we also store the GPS location which allows users to see from where the app was used to classify the song(s).

In terms of user capabilities, "Sham Shazam" is designed to be used by both music enthusiasts and businesses. Individual users can identify songs for personal use, while businesses can use the system to track and pay royalties to the correct artists and record labels.

Implementation

We chose to use the ReactJS library for our frontend for a number of reasons. We wanted to challenge ourselves to learn a new technology, and neither of us had used React. It's also an extremely popular web development framework at the moment, so we were hopeful that would mean there would be a lot of resources available to learn from and that we could see a lot of benefits from learning how to use it. React is a very flexible and powerful framework, so we were confident that it would fit our needs.

For the backend, we chose to use the ExpressJS framework. From our research we found that it would perfectly complement the React frontend, as we would only need to use 1 language for the majority of our code. As well as that, it's a very straightforward framework and it fit our intended use case. In our backend we also have a python script which is called by the ExpressJS framework to denoise audio files in the case that they cannot be identified by the AudD API.

For our database, we chose to use MongoDB. MongoDB is a NoSQL database management system which stores data in a form similar to JSON instead of tables of rows and columns like SQL databases. This allows for a lot of flexibility and it's easy to use. It fit all our needs for the data we would need to store and as we would be handling JSON data from the AudD API, using a NoSQL database would simplify data handling.

Sample Code

Getting Location Data:

```
try {
    // Get location data and store town and city names in 'location' variable as string
    try {
    const position = await new Promise((resolve, reject) => {
        navigator.geolocation.getCurrentPosition(resolve, reject);
    });
    const lat = position.coords.latitude;
    const lng = position.coords.longitude;

    const coords = await Geocode.fromLatLng(lat, lng);
    const address = coords.results[0].address_components;

    var location = address[2].long_name + ', ' + address[3].long_name;
}  catch (error) {
    // If there's any issue getting location (i.e gps disabled) set to none
    var location = 'None'
}
```

The above code shows how we got the location of the user. We used a react package called Geocode which easily helped us to use the Google Geolocation API. First, we got the current GPS location of the user and attached their coordinates to the variables ‘lat’ and ‘lng’. We then make a request to the API use the ‘fromLatLng’ method and those coordinate values. The response gives us a full address and from that we extract the town and district area and save that into the variable ‘location’. (e.g for somewhere in Dublin, our location variable might be ‘Tallaght, Dublin 24’)
If there are any issues getting the location, most notably the user has their gps location disabled, we set the location to static ‘None’ string.

Song identification frontend API request:

```
    // Make call to API to get song data
    const formData = new FormData();
    formData.append('audio', audioFile);
    formData.append('location', location);

    const response = await fetch('/api/song', {
        method: 'POST',
        headers: { 'authorisation': `Bearer ${user.token}` },
        body: formData,
    });

    const json = await response.json()

    if (response.ok) {
        setSongData(json);
    } else {
        console.log(response)
    }
} catch (error) {
    console.error('Error getting song:', error)
}
```

The above code shows how the song identification request is made from the frontend to the backend server. We create a FormData object with the audio file and the user's location as a string. This is passed into a request to the backend which will have an authorisation header, to ensure the user is logged in and so that the song can be saved for the correct user. We wait for a response, and save the resulting data to a songData state so that it can be rendered for the user.

Call to AudD API:

```
async function getSongDetails (req, res) {
    console.log("song request received")
    const audioFile = req.file;
    const user_id = req.user.id

    const previous_song = await Song.findOne({user_id}).sort({createdAt: -1})

    var data = {
      'api_token': 'aff02a6c3bf3342de6aae2ec83f07d16',
      'file': fs.createReadStream(audioFile.path),
      'return': 'apple_music,spotify',
    };

    var response = await axios({
      method: 'post',
      url: 'https://api.audd.io/',
      data: data,
      headers: { 'Content-Type': 'multipart/form-data' },
    });
```

```
try {
  const { title, artist, song_link } = response.data.result;
  const { userEmail, location } = req.body;

  if (previous_song.title == title) {
    res.status(200).json(previous_song)

  } else {

  const song = await Song.create({
      title: title,
      artist: artist,
      location: location,
      user: user_id,
      song_link: song_link
  });
  res.status(200).json(song) }
} catch (error) {
  res.status(400).json({error: error.message})
}

// Delete file after use
fs.unlink(audioFile.path, (err) => {
  if (err) {
    console.error(err)
    return
  }
})
```

The above code is used to make a call to the AudD API using the data sent by the user. The request requires an audio file and for the user to be authenticated, so that it can get the user's id. It makes a call to the AudD API with the audio file. If the request is successful, we check the most recent song for the corresponding user and ensure that the title does not match the APIs response. If it does, we return the data that's already stored in the database. Otherwise, we create a new Song object with the data received from the API and return that to the user. After we have responded to the user and stored the new Song object, we delete the audio file which was temporarily stored.

Denoising:

```
if (response.data.result == null) {
  // If the API doesn't find a match, run denois script to attempt to denoise the audio file and try again
  const { spawn } = require('child_process');

  const pythonProcess = spawn('python', ['../denoise.py', audioFile.path]);

  pythonProcess.stdout.on('data', (data) => {
    var modifiedAudioPath = data.toString().trim();

    // Read modified audio from temporary file
    var modifiedAudioBuffer = fs.readFileSync(modifiedAudioPath);
    // Delete temporary file
    fs.unlinkSync(modifiedAudioPath);
  });

  var data = {
    'api_token': 'aff02a6c3bf3342de6aae2ec83f07d16',
    'file': modifiedAudioBuffer,
    'return': 'apple_music,spotify',
  };

  var response = await axios({
    method: 'post',
    url: 'https://api.audd.io/',
    data: data,
    headers: { 'Content-Type': 'multipart/form-data' },
  });
}
```

```python
import IPython
from scipy.io import wavfile
from scipy.io.wavfile import write
import sys
import scipy.signal
import matplotlib.pyplot as plt
import librosa
import numpy as np
import noisereduce as nr

file = sys.argv[1]
sr = 16000
# Load audio file
y1,sr = librosa.load(file, mono=True)
#Load noise file
noise1, sr = librosa.load("storage/noise.wav", mono=False, sr=sr, offset=0, duration=60)

# Reduce the noise on the audio file, using the noise file
yg1 = nr.reduce_noise(y=y1, y_noise=noise1, sr=sr)

# Write the new file to storage.
temp_file_path = '/storage/modified_audio.mp3'
modified_audio.export(temp_file_path, format='mp3')

# Return path of temporary file to ExpressJS server
print(temp_file_path)
```

In the case that the AudD API is unable to identify the song from the audio file, we run a python script to attempt to denoise the audio. The 1st code sample is from the SongController file within the getSongDetails method. We check if the result from the API is null and if it is we create a new python process, passing in the python script and our audio file.The python file in image 2 is then run to create a new modified audio file, which it passes back to the SongController method. Then we make a new call to the API with the modified file.

The python script uses a module called ‘noisereduce’. This module uses spectral gating to attempt to denoise the audio using a noise file to estimate the noise threshold. It will gate noise below this threshold which should hopefully remove background noise.

Problems Solved

Recording and interval times

We wanted to differentiate our project from similar applications such as Shazam by focusing on long term use for the application. We wanted to allow users to be able to leave the application running for a longer period of time to keep track of all the music playing during that period, rather than only using the application to identify individual songs. To allow this, we had to have a system for recording the songs on a timer with an interval between recordings when the start button is pressed.

We had to decide how long the recording and interval times should be for the best results. On the one hand, we wanted to ensure that the recording was long enough to increase the chance of the API being able to identify the song but on the other hand, we didn't want too much data being sent as we were conscious about the privacy of our users. To decide what would be the best timings to use, we ran some experiments to test the performance of the API and how effectively it could identify clips depending on the length.

We put together a set of music clips from a 'Top hits' spotify playlist. We hoped that the songs on this playlist would cover our use case, as we were most interested in being able to identify current pop music. The AudD API recommended a maximum clip length of 20 seconds, so for our experiment we started with 15 second clips. Then we ran experiments with the same clips shortened to 12, 10, 7, 5 and 3 seconds to see how they performed with the API.

We ultimately found that 7 second clips would be the best to use. The vast majority of these clips could be identified, with the set having approximately 98% success rate out of the 70 clips. With the 5 and 3 second clips we observed a significantly higher number of cases where the song could not be identified by the API and so we decided these lengths would not be suitable. The 10, 12 and 15 second clip sets were all 100% successful at identifying the songs, but we decided that the performance of the 7 second clips was good enough and that this length would better suit our application.

When deciding the interval time, we wanted to ensure that our application would have multiple chances to identify a song in case the first recorded clip could not be identified, due to background noise or simply a hard to identify section of the song being recorded. We decided that we should try to give the application at least 3 attempts at identifying a song, assuming the song is allowed to play from start to finish. We made the assumption that the majority of songs are at least 2 minutes long and from that, we estimated that our interval should be 40 seconds to ensure the application was robust enough.

Denoising

Denoising proved to be a really tough challenge and we weren't fully satisfied with our final implementation. To figure out how to denoise the audio clips, we started with regular signal processing methods. This included using spectral gating to compute a spectrogram of the

audio signal and estimate a noise threshold. Then it uses that to gate any noise below the threshold. We weren't fully satisfied with this method as, while it did slightly improve identification accuracy, we were hoping for better results.

We then looked into Machine Learning methods and were excited at some of the possibilities we found. We were particularly interested in a 2-stage U-Net model we found (Google Colab link) which has been used to denoise historical music recordings to enhance the quality. We had wanted to see about implementing this model into our application but ran into a few problems. The first was that we would likely need to retrain the model. Given the model was designed for use with historical recordings and specifically for removing disturbances from old analog discs it didn't fit our use case which was more geared towards current pop music and removing noise like background chatter that might be present in a public place. This was something we were willing to attempt but we ran into a second problem. Unfortunately, the model took quite long to denoise the audio files. From our testing with 7 second long clips, it took approximately 3-5 minutes for the clips to be denoised. This was way too long for our application, as we were hoping to provide results to users within a few seconds. Having to wait multiple minutes, likely wouldn't be worth it for the user.

Due to these problems, we returned to the spectral gating method and implemented a version of that which used a noise file to better calculate the noise threshold. We used a noise file with some generic background chatter which we hoped would adequately cover the majority of cases. This was an improvement over our original attempts at using spectral gating without providing a noise file.

While our final implementation of denoising the audio files was adequate, given more time, I think we would look into possibly creating a Machine Learning model from scratch to perhaps fit our needs more. Unfortunately with the limited time frame and the limited experience we began with in the field of Machine Learning, this wouldn't have been possible for this project. Despite that, we were satisfied with the spectral gating method and we both still learned a lot about the field of audio processing, denoising and machine learning.

Results

Overall we were quite pleased with how our project turned out. We would've liked to have a more robust denoising system in place using some of the Machine-learning methods we had researched. If we could start again, we would've focused more on implementing the denoising system from the beginning to perhaps have enough time to create our own Machine Learning model. As we only started researching denoising methods a few months into the project, we found that we would have to rely on regular signal processing methods, due to the limited time and the issues we encountered with the Machine Learning methods.

Besides that, we were really happy with how everything else turned out. We were happy with our implementation of the API calls and the interval and recording timings. We were also satisfied with the overall usability of the application and feel that we were able to keep the design quite intuitive.

We learned a lot during the project. We learned how to use the ReactJS library and would now be more confident using it for future projects. We also learned a lot about signal processing and machine learning methods in relation to denoising audio clips. We are both hopeful that we can apply what we learned about Machine Learning in future, whether to make improvements to this project or in something new.

Future Work

In terms of our project, something we really would like to do is to improve the denoising functionality to better identify music in a noisy environment. We found a lot of promising open source software dealing with speech recognition in noisy environments, so it would be interesting to see if we could do something similar to specifically recognise and enhance music playing.

Another key area of future development is integrating the system into existing music streaming platforms, such as Spotify and YouTube, to provide accurate song identification and royalty tracking for users and content creators. This would require further research and development to ensure compatibility with different platforms, but would greatly enhance the usefulness of the system for a wide range of users.

In addition, we plan to use the skills and knowledge gained through the development of "Sham Shazam" to create similar apps that address other services and industries. For example, we are interested in exploring the potential for similar systems in the field of speech recognition or environmental monitoring or even AI chatbots.