

Music-Recognition-Tab

Conor Murphy

N00180141

Supervisor: John Dempsey

Second Reader: Sue Reardon

Year 4 2021-22

DL836 BSc (Hons) in Creative Computing

Abstract

The aim of this project was to create a music recognition program that would use a sample recording from a user of a song they wanted to identify, the program will then find a match and then display it to the user with the artist and song title, as well it will return link lessons on how to play the song on guitar from YouTube and UlitmatGuitar. This application enables users to try and find songs they would like to identify and learn on guitar or whatever instrument they play. Research was carried out for this project which included research on how applications like Shazam can identify songs. The process on how audio is identified is explained in this report, along with API's that could help develop the application. The final application is a web application that was created using Python and Flask, with only some bugs that were discovered while testing the application. Testing of the application was done during and after the implementation, the accuracy of the model was tested which found the recording format made it so the model would not give an accurate prediction. The database was also tested, this led to a discovery that the database could only contain 6 songs, any more than 6 would affect the accuracy of the model.

Acknowledgements

I would like to thank my supervisor and second reader, John Dempsey, and Sue Reardon, for their help and guidance throughout the project, and suggesting other ways to overcome challenges that occurred. For example, when trying to create a frontend for the application John suggested to use Flask which was something I had not thought of and made implementing the frontend easier. Also, when an error occurred with the application to do with the audio file not having a Riff ID, John put me in contact with some of his colleagues who specialize in audio files.

I would also like to thank my classmates and my parents who took their time in completing the survey for my project and being involved in the testing of the application.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Conor Murphy

Signed Conor Murphy

Failure to complete and submit this form may lead to an investigation into your work.

Contents

| | |
|--|----|
| Contents..... | 5 |
| 1. Introduction | 7 |
| 2. Research..... | 9 |
| 2.1 AI Audio Content-Based Music Retrieval..... | 9 |
| 2.1.1 Audio Identification..... | 9 |
| 2.1.2 Audio Matching..... | 11 |
| 2.2 Shazam | 12 |
| 2.2.1 History..... | 12 |
| 2.3 Music Identification & Retrieval Libraires/API's | 13 |
| 2.3.1 Shazamkit..... | 13 |
| 2.3.2 ACRCLOUD..... | 13 |
| 2.4 Flask | 14 |
| 2.5 Conclusion..... | 15 |
| 3. Requirements..... | 16 |
| 3.1 Requirements gathering | 16 |
| 3.1.1 Popular application - Shazam..... | 16 |
| 3.1.2 Survey..... | 18 |
| 3.2 Requirements modelling..... | 21 |
| 3.2.1 Persona | 22 |
| 3.2.2 Functional requirements..... | 22 |
| 3.2.3 Non-functional requirements | 22 |
| 3.2.4 Use Case Diagrams..... | 22 |
| 3.3 Feasibility | 23 |
| 3.4 Conclusion..... | 23 |
| 4. Design..... | 24 |
| 4.1 System Architecture..... | 24 |
| 4.2 Application Design | 24 |
| 4.2.1 Technologies | 24 |
| 4.2.2 Backend..... | 25 |
| 4.2.3 Frontend..... | 25 |
| 4.3 User Interface Design..... | 25 |
| 4.3.1 Wireframe | 25 |
| 4.3.2 Users Flowchart | 26 |

| | | |
|-------|--|----|
| 4.3.3 | Style Guide | 27 |
| 4.4 | Conclusion..... | 27 |
| 5. | Implementation | 28 |
| 5.1 | Development Environment..... | 28 |
| 5.2 | Database | 29 |
| 5.3 | Backend..... | 30 |
| 5.4 | Frontend..... | 39 |
| 5.5 | Conclusion..... | 48 |
| 6. | Testing..... | 49 |
| 6.1 | Usability Testing..... | 49 |
| 6.2 | Unit/Integration/Feature Testing | 53 |
| 6.3 | Algorithm Testing..... | 55 |
| 6.4 | Conclusion..... | 56 |
| 7. | Project Management | 57 |
| 8. | Future Development..... | 58 |
| 9. | Conclusion..... | 59 |
| 9.1 | Project summary | 59 |
| 9.2 | Limitations..... | 59 |
| 9.3 | Learning Outcomes | 60 |
| 9.4 | Final Words | 60 |
| 10. | References | 61 |

1. Introduction

This project is about music recognition, where a user using the application will use a sample recording of a song, the application will then try to identify the song using other songs stored in a database and show a result that is the closest match to the sample recording. The application will also show the user links that are used as lessons to play the song on guitar from YouTube and UltimateGuitar.

One of the application target markets is guitarists, it enables them to identify an unknown song they would like to learn to play. From the lessons on YouTube and UltimateGuitar website the user has a couple of options to choose from, YouTube has many video lessons to learn from that explain chords they can use, tips on playing the song and more. On the UltimateGuitar website it contains different versions of lessons including chords or tabs, which is a method for guitarist to learn songs quickly and easily by showing the guitarist the notes to play, for how long and what technique/strumming pattern to use.

The lessons to choose from both on YouTube and UltimateGuitar have rating systems that display how useful they are, on YouTube the user can see if a lesson is good by the views and likes on the video. While on UltimateGuitar has a 5-star rating system, it displays the number of people that have rated the lesson. Both YouTube and UltimateGuitar also have a comments section where feedback is given, this can contain someone else giving advice/tips on playing the song.

This report is structured as so, firstly the research chapter is examined. This will explain how music recognition works, including the theory and details of the process music recognition goes through when trying to identify a song. Within the research chapter audio identification is discussed, along with audio matching, the history of Shazam, a music recognition application, and Flask which was used for developing the application. The next chapter in the report is the requirements chapter, which layouts what is required for the development, here similar or popular application advantages and disadvantages are examined. Results from a survey is carried out to see what people would be interested in for an application like this and using this information a persona is created for a type of user who would use this application.

The design chapter is next, and this chapter is used to examine the application system architecture, the application design which includes the database design and technology used for development, and the user interface design which examines wireframes and style guides for the application. The chapter that follows the design chapter is implementation, here the development environment is discussed, and then the database, backend, and frontend are explained they were implemented into the application. Next is the testing chapter, the first thing discussed here is the usability testing which involved having users testing the application by giving them tasks to complete, and after they completed the tasks, they were asked to fill out a survey about their experience. In this chapter unit/feature testing and algorithm testing are also completed, this involves testing to see if all features the application set out to do work as desired, and the algorithm test is used to test to see if it is accurate.

The remaining chapters include project management which examines how the project was prioritised and where the code was stored. The next chapter is future development, this chapter contains future features that could be added to the application or what bugs could be fixed in the future to improve the application. The last chapter is a conclusion about the project, this includes the limitations the application has, the learning outcome and final thoughts about the project.

2. Research

This chapter of the report contains research that was investigated and gathered for the understanding of this project. How audio content-based music retrieval is the first topic examined, this includes information about audio identification and how it works, also audio matching is discussed too. Next services that could help with creating an audio identification application was researched, examples of applications that could help with the development were Shazamkit and ACRCLOUD which are discussed. Lastly Flask was researched for the project, here it is explained what Flask is, how it works, how to setup a Flask application and how a Flask application folder is structured.

2.1 AI Audio Content-Based Music Retrieval

Audio content-based music retrieval is using audio to retrieve information about music that matches the audio, applications like Shazam use this by using audio identification and audio matching. Audio identification is gathering hash data from an audio file or recording and searching a database to find a match. Audio matching is similar but uses peaks in the audio file or recording to find a match.

2.1.1 Audio Identification

According to Grosche, Müller, and Serra in one document and Jovanovic in another, audio identification has received the most interest and is widely used in commercial applications. With the process of identification, the audio material is compared to an audio fingerprint, these are compacted audio signatures from recordings. With real world applications fingerprints are needed to complete certain requirements in these applications, these fingerprints can capture specific characteristics of an audio fragments/recording that can be reliably identified and distinguished from millions of other songs. With real world scenarios these audio fragments and signals deal with exposure from distortions from microphones and other devices. Noises, pitch shifting, lossy audio and dynamics compression, equalization or time scaling are more likely to affect the audio signal. For identifications to be more reliable, fingerprints must show robustness against these noises and distortion. For all content-based retrieval applications an important issue is scalability.

Stated by Grosche, Müller, and Serra in their document and Jovanovic in his, for an audio identification system to be reliable these applications must have the entire digital music catalogued. Making fingerprints compact and efficiently computable would help minimize storage requirements and transmission delays. Importantly the application requires efficient retrieval strategies, to facilitate the database searches which are fast. Large-scale audio identification systems requirements are a crucial importance for the design of these application, however for all these requirements to be satisfied, there must be a face trade-off between contradicting principles.

Fingerprints can be designed and computed in various ways, one group type of fingerprints is built up of short sequences of framed-based feature vectors like MFCC, known as Mel-Frequency Cepstral Coefficients, a set of low-level descriptors or Back-scale spectrograms. For obtaining the representations for the required robustness thresholding techniques, vector quantization, or temporal statistics are required. The other group of fingerprints is made up of spectral peaks, which is a sparse set of characteristic points, or a characteristic wavelet coefficient, an example of a peak-

based fingerprint system is the Shazam Music identification service, which is stated by Grosche, Müller, and Serra in their document.

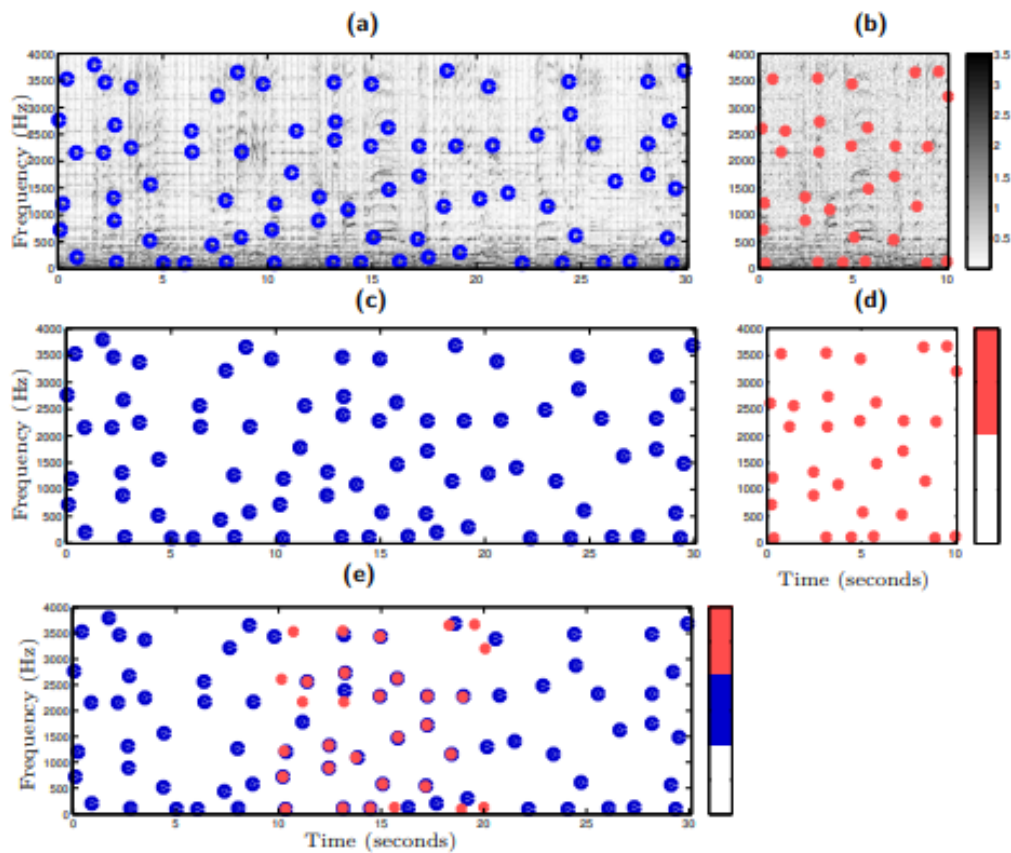


Figure 1: Shazam audio identification graph of "Act Naturally" by The Beatles, Grosche, P., Müller, M., & Serra, J. (2012)

In Figure 1 above from Grosche, Müller, and Serra document, shows an illustration of The Beatles song "Act Naturally" captured by Shazam, graph A displays the recording from the database which contains extracted peak fingerprinting to identify the song. Graph B is also using extracted peak fingerprinting from a recording sample for 10 seconds. Graph C displays a constellation map using the database recording while graph D also uses a constellation map too but using the sample recording. Lastly graph E displays a superposition which uses both the database fingerprint and sample recording fingerprints to try and match the two.

Shazam is available for smartphones and tablet applications and allows the users to record a short audio fragment sample of an unknown song which is picked up from the devices microphone. The application then sends out the audio fingerprint to a server, so the fingerprint is searched for through a database of songs. When the song is identified the retrieval, result is then sent to the user's device and displays the song information, for example the song title, the artist and album it's from. Grosche, Müller, and Serra stated in their document, that from this approach of music retrieval first the application must compute a spectrogram using the audio fragment sample and using a short time Fourier transfer, which is the tool that breaks waveforms by using sine and cosine to break the waveform into an alternate representation, Fourier transform proves that any

waveform may be rewritten as the sum of sinusoidal functions. After this step the application then decides to apply a peak-picking strategy to the process which extracts local maxima from the magnitude spectrogram which is a time-frequency point which are locally predominated. The peaks that are extracted superimposed to the spectrogram, then the peak-picking process is used so the complex spectrogram is reduced, changing it to a constellation map, which is a low-dimensional representation of which is the original signal that are displayed as small sets of time-frequency points, as seen in the graphs C and D in Figure 1. These peaks are highly characteristic, reproduceable, and robust against distortion signals, the only way a peak is defined is by its time and frequency values, the values of magnitude are no longer considered.

Grosche, Müller, and Serra in their document, declared that the database look-up strategy works by the sample recording and the database recording in the constellation maps being locally compared to each by counting the matching peaks that occur in both the constellation maps. If there is a high count of peaks this indicates the database fragment is most likely to be the correct match, this can be seen in graph E in Figure 1.

Shazam's has a high identification rate with identifying unknown songs while searching through a large database system. A weakness that the application has is that it is not able to handle a time scale modification of audio which is occurring frequently in broadcasting monitoring. This can occur due to time scale modifications leading to frequency shifts and recording samples hash values being completely changed, which is backed up by Grosche, Müller, and Serra research.

2.1.2 Audio Matching

For the tasks of audio matching there is a required suitable descriptor that is needed, from the recording sample the task will capture the characteristics of the song. A chroma-based audio feature, which is also known as a pitch class profile is used for analysing tonal music and is suitable for the mid-level representation of the retrieval context, which is stated by Grosche, Müller, and Serra.

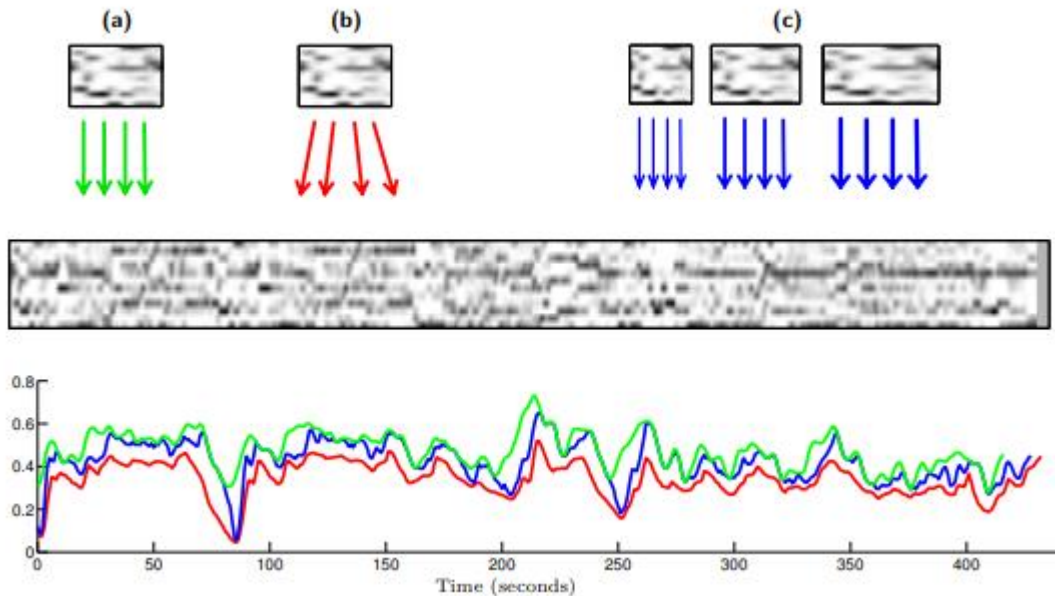


Figure 2: Audio Matching Procedure using Query Fragment on Beethoven's Opus 67, Grosche, P., Müller, M., & Serra, J. (2012)

A subsequence search is used, instead of a sparse peak representation with chroma features, a query chromagram is then compared to all subsequence of database chromagrams. A matching curve is obtained as a result, which can be seen above in Figure 2, the small values indicate that this position is where the subsequence database starts and is similar to the sample recording sequence, stated by Grosche, Müller, and Serra. They also stated that for the best match is found at the minimum of the matching curve, meaning a distance measure is applied that will deal with the tempo differences between versions. Like the dynamic time warping (DTW) which is used for comparing similarity, the distance that is between two arrays, and different length time series. The Smith-Waterman algorithm is also used, this uses a technique called dynamic programming that takes any length of an alignment at any sequence and location, an optimal alignment is determined and founded then. According to Grosche, Müller, and Serra to linearly scale the recording sample is an alternative approach which simulates different tempi, the distance is then minimized for all scaled variants. Strict subsequence matching, DTW and a multiple recording sample strategy were used in Figure 2 above which shows matching curves that are different.

2.2 Shazam

This section of the research examines the history of Shazam, it discusses how different the application was before it became what it is today, this includes how the application was used and the limitations of the application. Back in the early versions of the application when the song was identified it gave the user the option to purchase the album of the song and other merchandise of the artist.

2.2.1 History

Wang documented in their research, along with and Hussain, Almazini, Almazini, and Mkpojiogu in theirs, that in 1999 a music identifier was created called StarCD, where users with mobile phones

were able to identify songs that were playing on the radio. Users would identify the unknown song by calling the StarCD service and enter the radio station call letters with the keyboard on the phone. StarCD automatically would use a third-party database playlist to find out the song playing on the given radio station at the time the call was made. This made it so the StarCD system was limited to the songs that were being played on radio stations.

In Wang's research they state that QBE is a music recognition system that is more flexible, instead of requiring the radio station information it uses a sample recording of a song. In 2000 Shazam aimed to develop a QBE music recognition service that would be commercially offered on mobile phones. When a user captured an audio sample, they would call the service using a short dial code, when the server is finished sampling the audio the server would then hang up and the identification results would be returned to the user through a text message.

A newer version of the QBE music recognition was introduced in 2004 called Song Identity which provided a more interactive interface. Users would download an app onto their mobile phones before using Song Identity, when they wanted to identify a song, they would launch the app which recorded 10 seconds of the audio and sends an audio file to Shazam or it performs a feature extraction to generate a small signature file on the phone, which is stated by Wang in their research. The signature file is sent to a central server which performs a search for the song title, artist, and album, which then returns the information to the app and gives the users the option to download the song for a ringtone if it was available. Future versions of this app allowed the users to purchase the full song, CD, or concert tickets, which is documented in both Wang research and Hussain, Almazini, Almazini, and Mkpojiogu research. Song Identity and other commercial QBE music recognition systems must overcome technical challenges like noise, distortion, and database management, which was discussed earlier.

2.3 Music Identification & Retrieval Libraries/API's

Applications that were discovered and examined to help create the audio identification and retrieval of songs were Shazamkit and ACRCLOUD, these are libraries and API's that can be used. They have a database online that stores huge amounts of songs and information related to it, including title, artist, and the audio fingerprints.

2.3.1 Shazamkit

Shazamkit was released by Apple in the summer of 2021, it allows developers to use Shazam features in their own app. Shazamkit was created to be used on mobile applications, so it is implemented into applications with Swift for Apple products, or Kotlin for Android. With some investigation, this API requires an Apple developer token, which they do not give out to developers who are not a part of a company.

2.3.2 ACRCLOUD

ACRCLOUD is similar to Shazamkit, but instead of just being able to use music recognition this API can use humming recognition, where users will hum a tune and the API will find a matching song to the hum. They also have other APIs for broadcast monitoring, live channel detection, a copyright API, and so on. ACRCLOUD allows developers to use music services like YouTube and Spotify so their links

are displayed with the results to the user. With some investigation it was discovered the default database size only allow for two songs to be added, also the database would only work for 14 days then after that time payment was needed.

2.4 Flask

According to pythonbasic website, Flask is a Python web framework module that allows developers to create applications for the web easily using Python. Both the pythonbasic website and the Great Learning Team website, state Flask was created by a Python team called POCO which was led by Armin Ronacher. They based Flask on other projects they created such as Werkzeug WSGI toolkit and the Jinja2 template engine. The Werkzeug WSGI toolkits allows developers to implement requests, response objects and utility functions, the Jinja2 allows developers to render a dynamic web page and pass Python variables into a HTML template.

Stated by the pythonbasic website Flask is described as a microframework, meaning it has been designed applications to be simple and scalable. On the Great Learning Team website, they describe a microframework as a minimalistic framework where developers do not need to set up as many things as an enterprise framework to have the application hosted and running. Which is useful for small web applications as for not using an enterprise framework they can save lots of development time and money.

Some advantages the Great Learning Team declare on using Flask are that it is easy for developers to setup and use, also it allows developers the freedom of the web application structure. With a Flask app when its complexity increases the foundation is the structure of the application.

When setting up a Flask application first the user must make sure their device is has the most recent Python version, after they have the most up to date version they must setup the virtual environment by using the command `python -m venv .` in a terminal. After the virtual environment is setup the next step is to navigate to the scripts and activate the virtual environment by using the command `.\venv\scripts\activate`. When in the virtual environment the user must install Flask by `python -m pip install flask`, when flask is finish install the user can start development on their application and when they wish to run the application, they run the command `flask run` and click the link that appears in the terminal window. When the user wants to shut down the application in the terminal, they run the command `deactivate`.

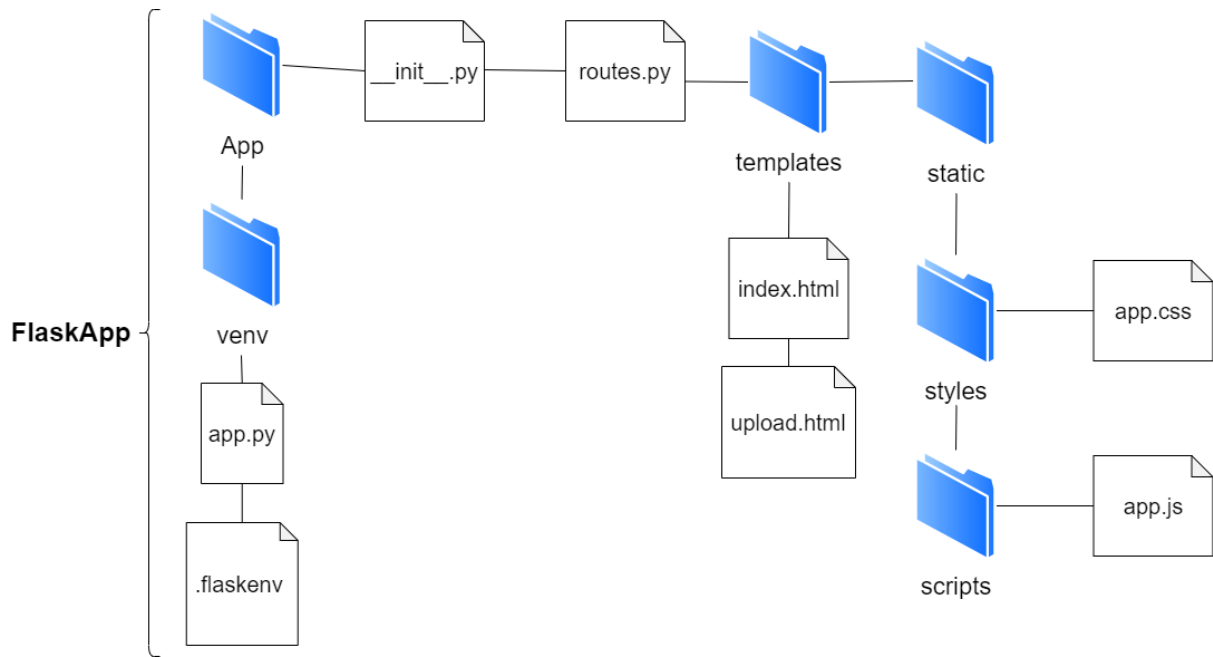


Figure 3: Default Flask folder structure

The above Figure 3 displays a default Flask application folder structure, the venv folder is added to the folder when the virtual environment is setup, this contains the scripts and libraries need for the application and runs Flask. The `__init__.py` file setups Flask for the application by importing it, it also tells the application from the app folder import the routes file. This file contains the routes used for the application and can use templates from other folders. The templates are stored in a folder called templates and if the templates require CSS or JavaScript the template tells the application to use them from the static folder. The `app.py` file is used to define the Flask application, the `.flaskenv` file is used to tell Flask how it is imported by using `FLASK_APP=app.py`.

2.5 Conclusion

In conclusion, this chapter examined the audio identification in audio content-based music retrieval dealt with the process of identification using audio fragments and fingerprinting from a sample audio fragment. The audio matching investigated the chroma-based audio features which contain pitch class profile that are used for analysing music. In this chapter the history of Shazam was also discussed, where older approaches of music retrieval were examined including StarCD, and QBE music recognition was discussed. The final area of this report used a study that was conducted to find the usability of Shazam, here the feedback of the study was positive, with the app Shazam scoring high in usability and being compared to other similar applications that were previously tested.

3. Requirements

The requirements section was used to figure out what the functions and features the application should be able to do, and to understand what a user who might use an application like this would like the application to be able to accomplish. This is done by doing interviews and surveys asking what type of features they would like to see on the application. Also, in this section of the report, other applications that are similar to this one is discussed, this includes its features, the advantages and disadvantages of the application.

3.1 Requirements gathering

Requirements gathering includes examining applications that are similar or related to this project, Shazam is the most popular music identifier and retrieval application out there, so it was discussed. Also, requirements gathering includes asking people to complete a survey about if they enjoy music, do they play and instrument, have they ever used Shazam, ect.

3.1.1 Popular application - Shazam

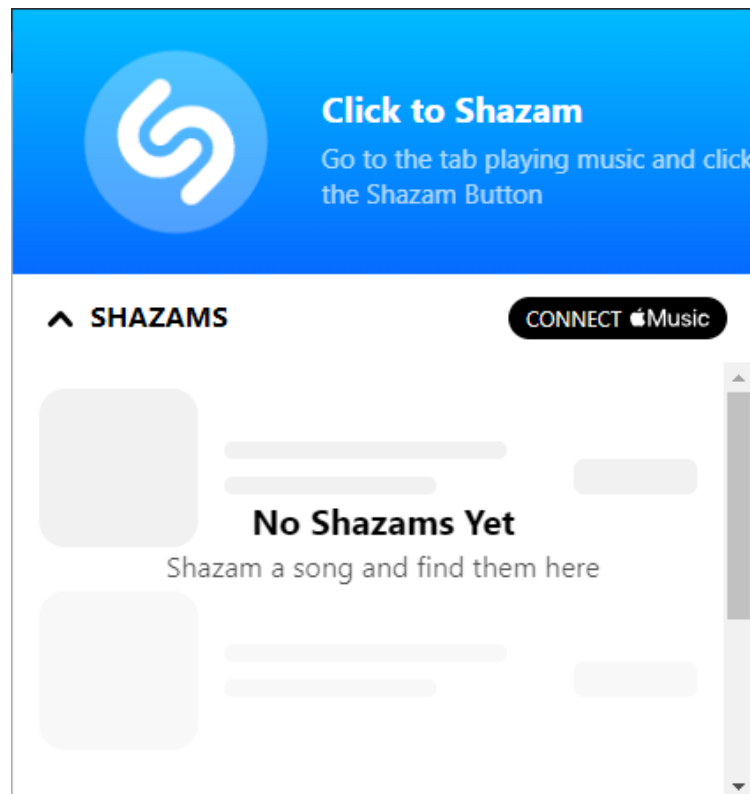


Figure 4: Shazam Page

Shazam has mobile apps and a Google Chrome extension; with the Google Chrome extension app the Shazam identifier is available on any website a user is on. When the user is on a page that contains a song, they wish to Shazam, they can access the Shazam Chrome expansion and a little box appears on the screen that has a button the user can press when they want to Shazam the song, there is also a drop-down menu that contains the history of Shazam's made by the user on the device, as seen in Figure 4 above.

When the user clicks the Shazam button the AI in the back end works to find the fingerprint and audio fragment from a database that is related to the sample of the song and then will display the title of the song and the artist. Figure 6 below shows Shazam trying to retrieve the song, while Figure 5 shows that Shazam has retrieved the song and is displaying its information and a link to play the full song.

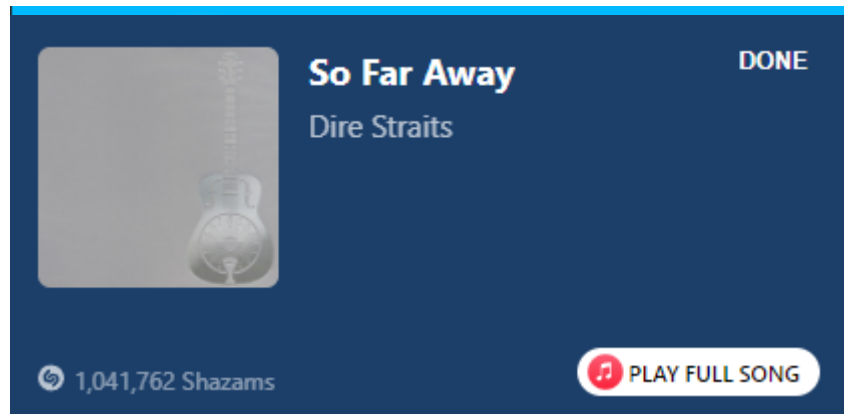


Figure 5: Shazam has found the song

The advantages of Shazam are:

- It has a huge database of songs to work from, even songs by not well-known bands like The Scratch can be scanned
- It's very accurate and quick with retrieving a song
- It's an easy-to-use application
- Can be used on several devices, i.e., Apple, Android, PC, Smart Watches
- Can see history of songs that were scanned
- Can connect Apple Music or Spotify account so the song can be added to a user's playlist
- Mobile versions are good at cancelling out background noises when scanning the song sample
- Very good at retrieving the correct version of the song, for example Whiskey in the Jar, when scanning the Thin Lizzy version it retrieves it, when the Dubliners version is scanned it works and with the Metallica one too.
- Most songs don't require lyrics for scanning the song, for example Wipe Out and Misirlou

The disadvantages of Shazam are:

- Some songs that are not as well known that do not have lyrics will not be retrieved when scanned
- The Shazam applications like the Chrome expansion, the Android Pop Up Shazam and both the Android and Apple notification areas, these applications are always on which could use up a lot of the device battery

3.1.2 Survey

A survey was conducted to see if people would be interested in using an application like this, the survey had thirteen responses with 8 questions. The first question on the survey was asking the participants how much they enjoyed music from a scale from 1-10, the average was nine people picking 10 meaning they loved music, one person chose 9 and 8 while two chose 7. From this question it shows that people enjoy music, which is displayed in Figure 6 below.

From a scale of 1-10, do you enjoy music?

13 responses

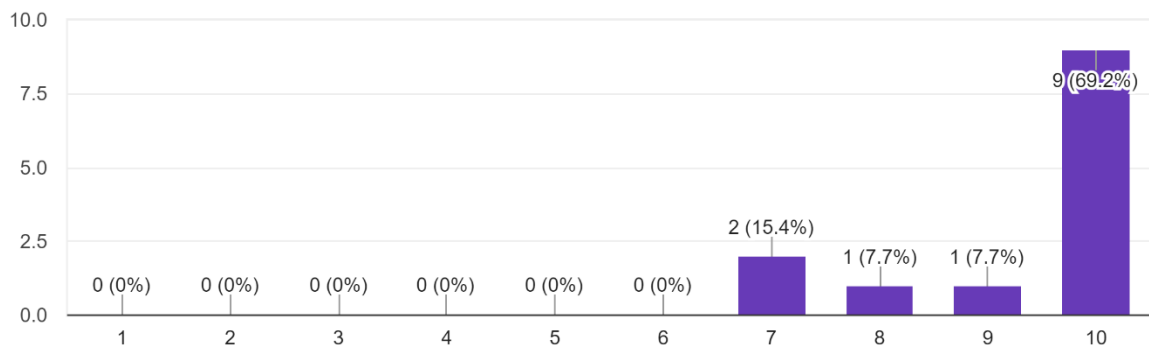


Figure 6: Survey scale question asking people if they enjoy music

The next question asks if the participants have ever been in the situation where they are watching a film, a TV show or outside and heard a song they liked but didn't know the song title or the artist, 76.9% of the participants said that it happens to them a lot, while 23.1% said only sometimes and 0% said no, which can be seen in Figure 7 below.

Have you ever watched a film, tv show or been out and heard a song you like but don't know the name of the song or the artist who plays it.

13 responses

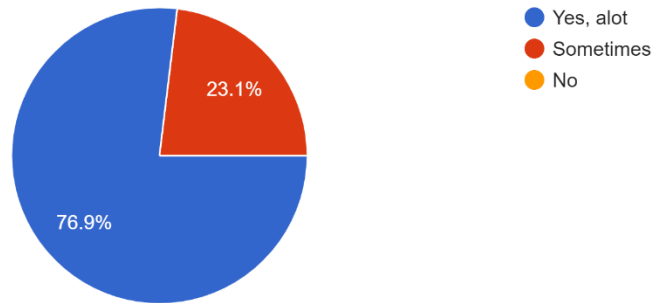


Figure 7: Survey Pie Chart of hearing a song not knowing the title or artist from a film or TV show

The next question is a continuation of the previous question, it asks if they have ever used the application Shazam to discover the title or artist of an unknown song like in the situation asked previously, 76.9% said they have used Shazam while 23.1% said no. This is displayed in Figure 8 below.

Have you ever used the app Shazam if you don't know a name of a song or the artist who plays it?

13 responses

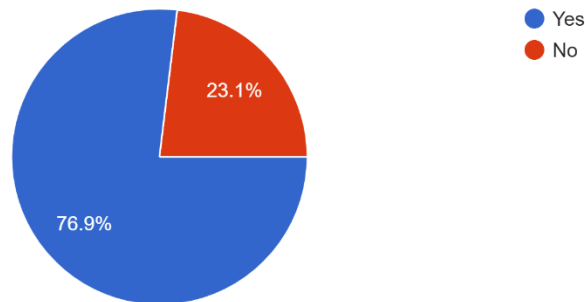


Figure 8: Survey Pie Chart of participants who have used Shazam

The question that was asked next was if they played an instrument, this had 30.8% of the participants saying they do play an instrument and 30.8% not playing one, while 38.5% said they do not play one but would like to learn, as shown in Figure 9 below.

Do you play an instrument?

13 responses

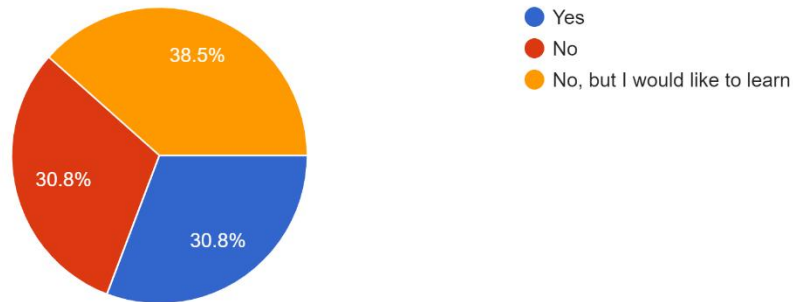


Figure 9: Survey Pie Chart of participants who play an instrument

This next question asks if the participants have ever used Shazam to find a song so they could search for lessons online for the song, only 23.1% said sometimes, 7.7% said they learn songs by ear, 69.2% said they don't play an instrument and 0% said they have done that before, which is shown in Figure 10 below.

If you play an instrument, have you ever heard a song while out you'd like to learn but have to Shazam the song to found out the name to find lessons online for it?

13 responses

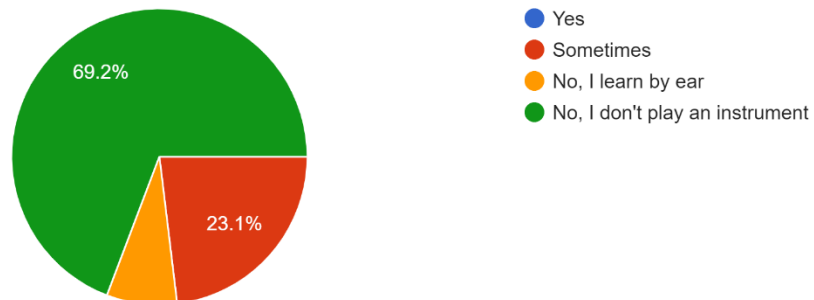


Figure 10: Survey Pie Chart of participants who has used Shazam to find a song they want to learn

The last remaining questions were optional to answer, this one asked if the participants answered to the previous question yes or sometimes, from a scale from 1-10 how interested who they be in using an application like this, only 5 participants answered, two answered 10 being really interested in the application, one said 7, 4 and 3, as shown in Figure 11 below.

If to the previous question you answered yes or sometimes, how interested would you be in an application that scans a sample of a song and retrieve...le, artist and a link for a lesson to play it online.
5 responses

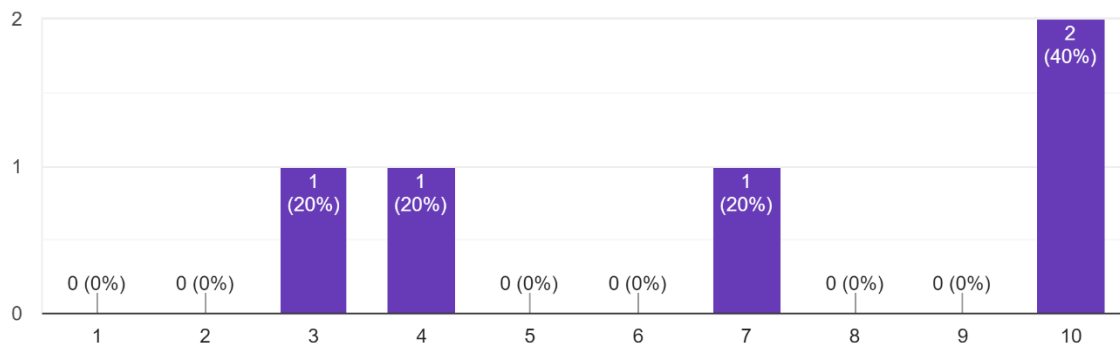


Figure 11: Survey scale question asking if participants would be interested in an application like this one

The next question as what other instrument would they like to see lessons for, only two answered saying drums and violin. The last question asks for any suggestions for the application, again two people answered saying having the ability to rate links of lessons so other users who are looking for the same song can see the best rating link. The YouTube link navigates the users to the search query of the song so users can see how much views and likes a lesson has received, as well the guitar tab link shows multiply ways of playing the song with ratings. The final suggestion was to have a link so the user can add the song to their Spotify account or playlist. All the suggestions are displayed in Figure 12 below.

If you have any suggestions or features you would like see in this application, please feel free to suggest one.

2 responses

Have the ability to rate links so people looking for the same song will see which link has a high rating


Link downloaded songs to Spotify account favourites or a play list.

Figure 12: Survey suggestion question

3.2 Requirements modelling

The requirements modelling includes using the information from the requirements gathering section and apply them to creating a persona for the project. Declaring what are the functional and non-functional requirements for the application, a use case diagram, and the feasibility of the project. Below is a persona that is made up which is the desired type of user to use this application, some information in the person is from the survey.

3.2.1 Persona

| User's Name | Key Demographic Details | Interests |
|--|---|--|
| <p>Billy</p>  | <p>Age: 19</p> <p>Gender: Male</p> <p>Education: IADT</p> <p>Location: Dublin, Ireland</p> <p>Employment: None</p> | <p>Enjoys music like Rock and Grunge. He has been playing the guitar for 2 years.</p> |
| Tasks | Future Features He wants | Goals |
| <p>Has heard a song from a film he was watching that he liked, he is interested in learning it on guitar. He uses the application to record a sample of the song to find out the song title, artist, and links to lessons.</p> | <p>He would like to see in future features a user dashboard where he can view the history of songs he has sent to be identified.</p> <p>He would also like there to be an option where he can add the song to his Spotify playlist.</p> | <p>He wants to be able to record a sample of an unknown song from a film and get the title and artist of the song, with links for lessons on how to play the song on guitar.</p> |

3.2.2 Functional requirements

Functional requirements are functionality of the application that are needed to run as intended, this includes the ability to use music recognition to gather information about a snippet of a song and use the title of the song and added to a String that is used to search on google to find a guitar lesson of the snippet song.

3.2.3 Non-functional requirements

Non-functional requirements are functionality that are not essential for the application to run but are a nice to have. These include:

- Asking the user for permission to use the device microphone
- Simple & easy to use
- Display results quickly
- Looks nice & easy to navigate

3.2.4 Use Case Diagrams

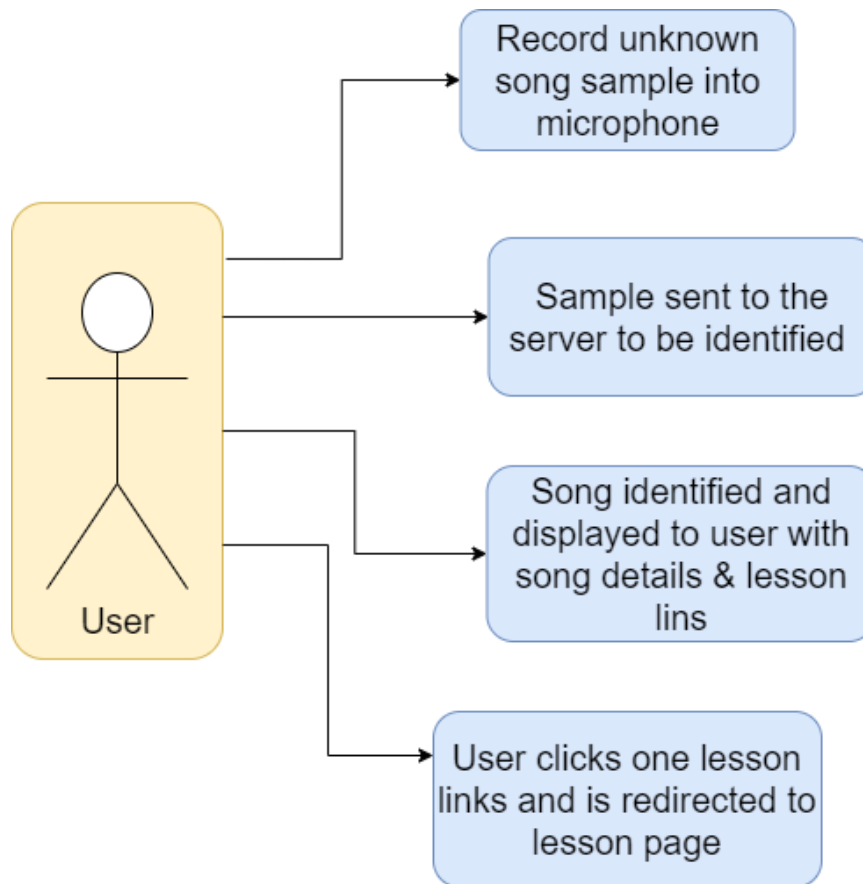


Figure 13: Use Case Diagram

3.3 Feasibility

Creating a music identifier and retrieval system will take up a lot of time to create, also having a database filled with song data will take up a lot of storage. So, the best option is to use a music identification API for the project or follow tutorials online on how to create one as this will save time.

Creating a microphone for the application that will record audio is doable using JavaScript, a tutorial will need to be followed to create one. Flask can be used to make the Python music identifier into a web application, for this some tutorials on how to setup and use Flask is needed to be examined.

3.4 Conclusion

In conclusion this chapter examines the most popular application Shazam, which is similar to this application, also this chapter gathers information from a survey to investigate how interested people would be used an application like this and what features they would like to see. The functional and non-functional requirements are also examined for the application.

4. Design

In this section of the report the overall design for both the system and user interface of the application are discussed. The backend of the application will be created using Python, which will create the database and the model for audio identification, while the frontend of the application will be created with Flask.

4.1 System Architecture

System Architecture is the concept and fundamental of an applications system, it defines the structure, behavior, and the views of the application. As displayed in Figure 14 below, the application will have a page where a user will send an audio clip to the server, there the server will send the audio clip to a prediction model where it will get information from the audio clip, known as audio fingerprints. After the model gets the audio fingerprints it will send them to the database, which is in a form of pickle files, it will search the pickle files for matching audio fingerprints from songs stored on it that match the audio clips one. After a match is found the result is sent back to the server where the song title, artists, and lessons in forms of links are added to the response for the user, then the results are displayed to the user back on the web page.

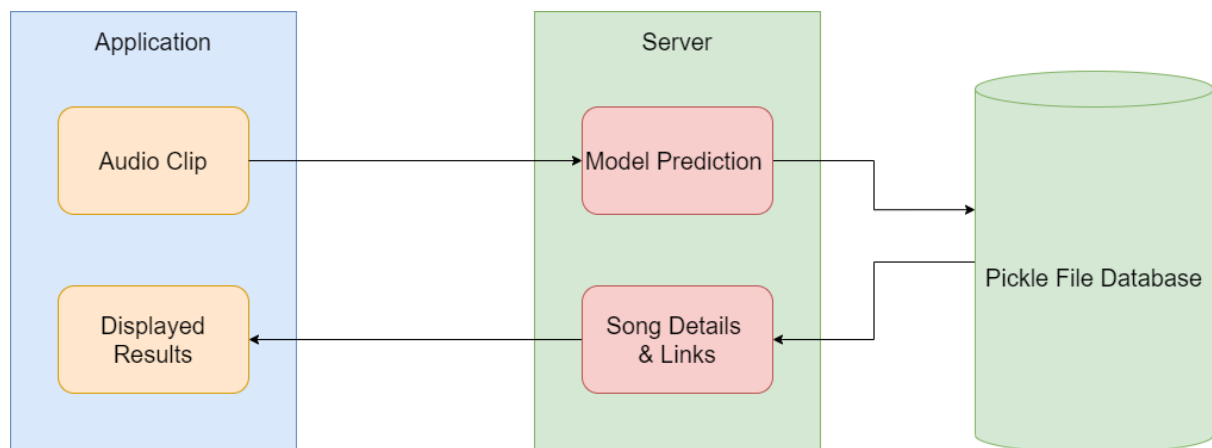


Figure 14: System Architecture diagram of the application

4.2 Application Design

The application design contains information about the technology used for the application, which are Python, Jupyter Notebook, Flask, and JavaScript. It also examines the design of the backend and frontend of the application.

4.2.1 Technologies

The technologies used for this application are Python for the backend, Jupyter Notebook was used to create the pickle file database for the application. After the pickle files are created, they are put into folder structure of the application to be used, and the model used for audio identification was created using Python. The frontend was created by using Flask, its setup the backend server of the application which stored the Python code, the Flask routes were told what to return to the user, this included web pages which for this application were created with HTML and designed with CSS. The

application used a recorder that would send the audio clip to the server so it could be identified, JavaScript was used to create this recorder, as well as sending the audio clip to the server and displaying the results of the prediction.

4.2.2 Backend

The backend contains the routes for the application, for example the default route / tells the application to load index.html page. The /upload_blob will receive the audio blob and send it through to the model to be identified and return the results in the form of JSON. The model accesses the pickle files and text file, which acts as the database, to find the matching result to the audio clip.

4.2.3 Frontend

The frontend of the application works with using Flask to display the application as a web application, the Flask routes are told where each route leads the user to, so the default page displays the home page to the user. Here the user can record an audio clip which is then sent to the server to be identified, the results are then displayed on the home page to the user which contains the title of the song, the artist, and links of lessons on how to play the song on guitar. JavaScript is used for the application in making the recorder work, sending the audio clip to the server, and displaying the results to the user.

4.3 User Interface Design

The user interface design discusses that the design for user will be, this includes a wireframe for the application, a style guide that the application will follow and user flowchart. The user flowchart contains a diagram that displays the process the user will take while using the application.

4.3.1 Wireframe

The below Figure 15 displays a simple wireframe prototype of the design of the application, the home page contains a record button and stop button, so the user can click the record button when they are ready to identify their unknown song. They can click the stop button if an error occurs with the recording and start again. When the recording has stopped, and the application has found a matching result it will display the results in a card below the buttons.

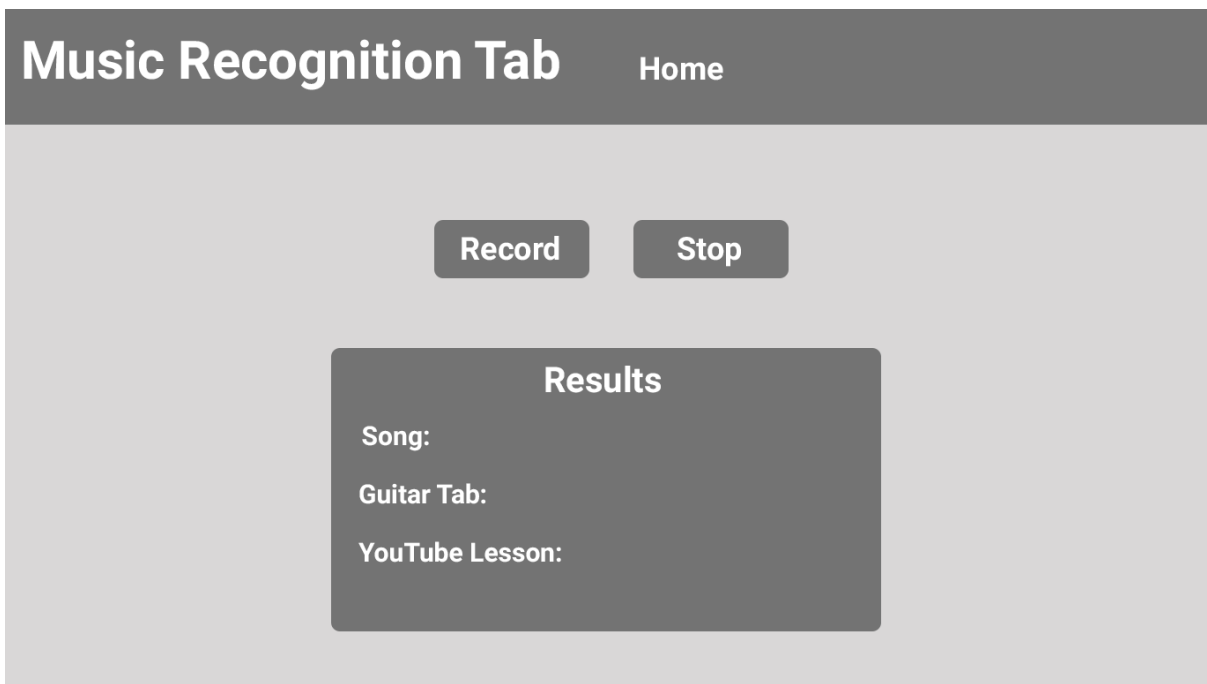


Figure 15: Desktop Prototype

4.3.2 Users Flowchart

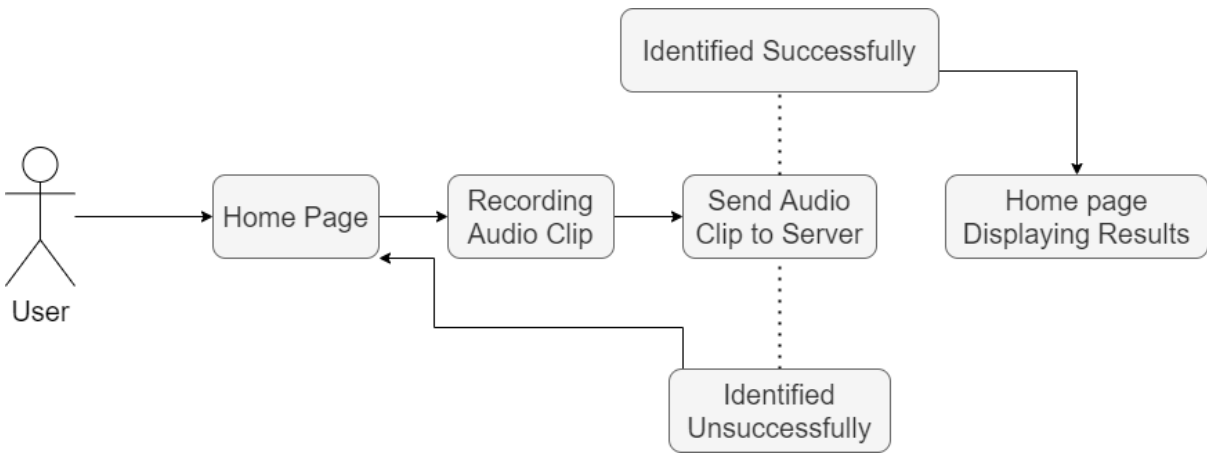


Figure 16: User Flowchart

The Figure 16 above displays a user flow diagram, the user will access the application and will be on the home page. From there the application will ask for permission to use the devices microphone, the user will then start playing a sample of a song into the devices microphone until the application has finished recording. The application will then send the audio clip to the server and look for a match, if the song has been identified successfully the user is redirected page to the home page but the results are display within the result card on the page. However, if the song could not be identified the user is still on the home page but are told no match could be found.

4.3.3 Style Guide

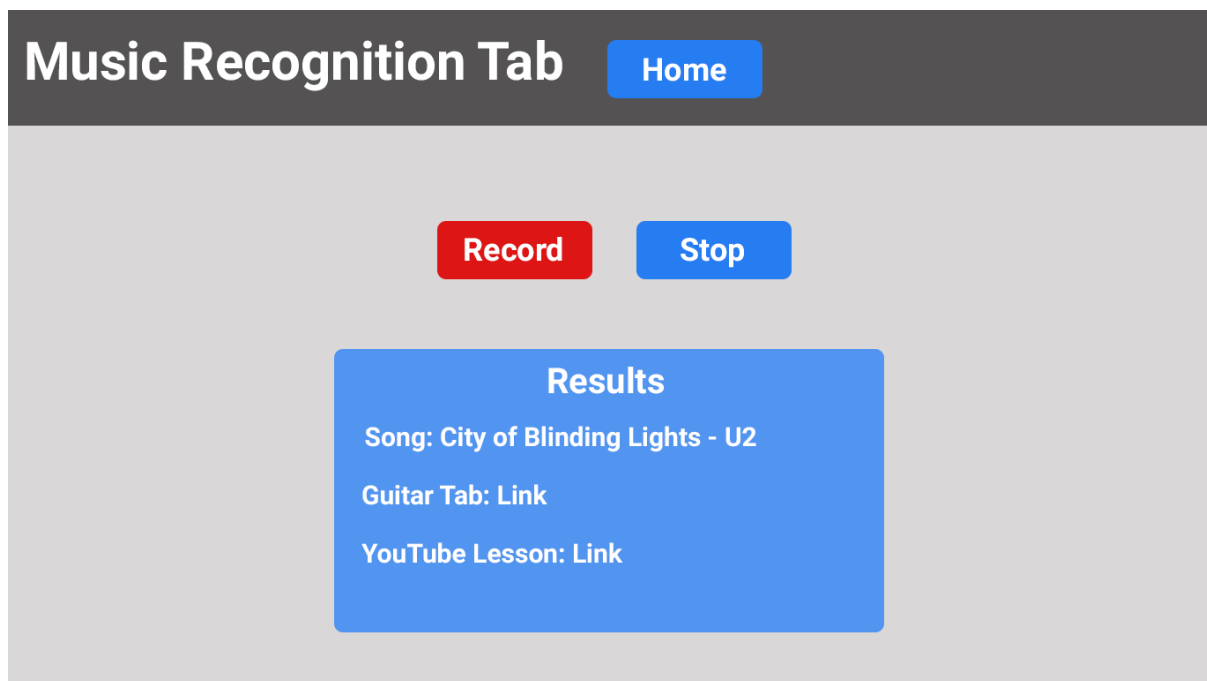


Figure 17: Detail Prototype

The above Figure 17 is similar to the previous prototype but is more detailed with the colour scheme the application will have. It shows the colours the buttons will have and the colour the card that contains the results will have.

4.4 Conclusion

This design chapter shows the system architecture of the application, showing the audio clip from the application being sent to the server which uses the model to identify the song, sending the results back to the application. Then the application design and user design are discussed, this includes discussing the technology the application uses, how the backend and frontend works, and wireframes of the application.

5. Implementation

This chapter of the report is about the implementation of the application, which contains the environment it was developed in, about the database and how both the backend and front end were developed and implemented into the application.

5.1 Development Environment

The development environment for the application includes python for the model that predicts the song, Flask was used for the backend of the application, pickle files and text files were used for containing database information. JavaScript was used to create the microphone features, formatting of the recording, converting the audio clip into a wav file, displaying the audio clip on the site, sending the audio clip to the server, and displaying the results on the page. The home page was created with HTML and designed with CSS. Figure 18 below displays the folder layout for the application.

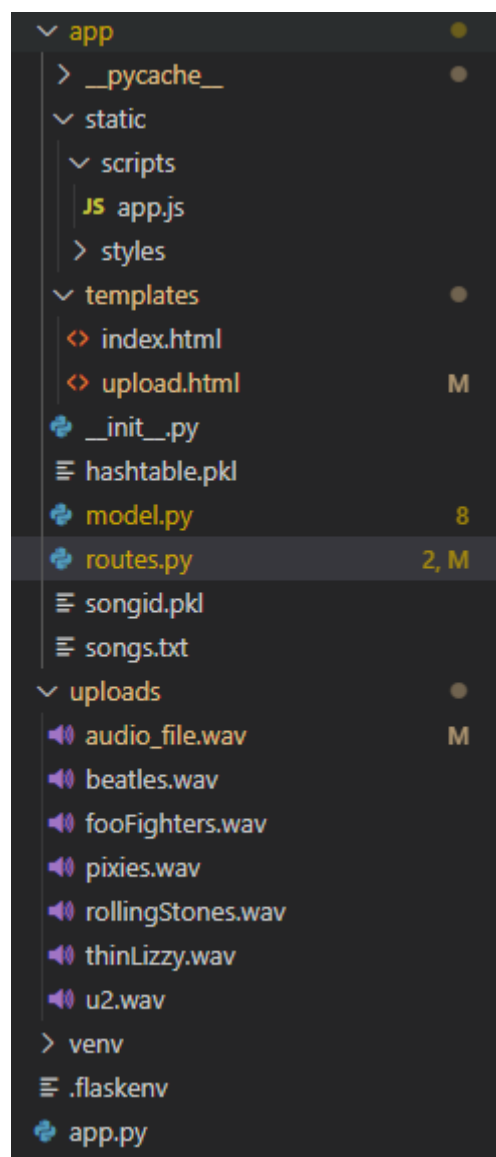


Figure 18: Development Environment layout

5.2 Database

Originally the application was going to be developed using Python for the backend and React for the frontend, though errors occurred when trying to implement the database for the Python backend. These errors included libraries that were required but were out of date and no longer working, also when trying to extract audio fingerprints from a song file the application would crash stating libraries were needed, so this approach was abandoned.

The second approach examined for the database was to use an API that would store and retrieve the data about the songs, two APIs were examined, these were Shazamkit and ACRCLOUD. As these APIs are developed for mobile apps Kotlin was going to be used in Android Studios. Unfortunately, it was discovered that the Shazamkit API required an Apple developer token, which they only give out to companies, the ACRCLOUD had a limit on its usage and required users to pay for its service. Due to a different approach was examined.

The type of database that was used for the final application was a local host database, when a user sends the audio clip to the server the application opens the pickle files and loads the information in them into the model, the same is done with the song list text file. The model then checks the audio fingerprint of the audio clip and the match zone target to the ones from the pickle files to find a match, after the match is found the information of the predicted song is returned.

While developing the database it was discovered that no more than 6 songs could be added, any more than 6 would affect the accuracy of the application. For example, if the database had 7 songs, when testing the 7th song, the application would either no be able to find a match, or it would return a response saying one of the other songs were a match. Also, if there were more than 6 songs in the database sometimes the application would have a problem trying to finds a match for one of the original 6 songs.

5.3 Backend

The backend of the application was made with Python and Flask as the server of the application, it was originally made in Jupyter Notebook, there a file called server-side.ipynb which contains code that reads the song list text file, it then navigates to the folder where the song files are stored and will match the song file name from the song list text file, for example it will match the file Jailbreak – Thin Lizzy.wav to the same exact same song name meaning in the song list .wav is need. Then the audio fingerprints are created and dumped into two pickle files called hashtable and songid, as displayed in both Figure 19 and Figure 20 below.

```
In [3]: song_dict = {}
hashtable = {}

file = open('songs.txt','r')#list of song files
allsongs = file.readlines()
allsongs = [a[:-1] for a in allsongs]
print(len(allsongs))
num = 0

for a in allsongs:
    x = song("./Songs_Wav/"+a, num)
    song_dict[num] = a
    num+=1
    print(x.title)
    x.find_key()
    x.cal_address()
    for key in x.addresses.keys():
        if key not in hashtable.keys():
            hashtable[key] = []
        for b in x.addresses[key]:
            hashtable[key].append(b)

6
<ipython-input-2-0d19580850c5>:32: DeprecationWarning: frombuffer instead
a = np.fromstring(data, dtype='<%s%d' % (dt_ch,
Thin Lizzy - Jailbreak.wav
<ipython-input-2-0d19580850c5>:66: RuntimeWarning:
self.specgram = 10*np.log10(self.specgram)
Foo Fighters - Everlong.wav
Pixies - Hey.wav
The Rolling Stones - Monkey Man.wav
U2 - City Of Blinding Lights.wav
The Beatles - Hey Jude.wav
```

Figure 19: Server-Side.ipynb file creating audio fingerprints

```
In [4]: with open("hashtable.pkl", "wb") as output:
        pickle.dump(hashtable, output, -1)
        with open("songid.pkl", "wb") as output:
            pickle.dump(song_dict, output, -1)
```

Figure 20: Server-Side.ipynb file dumping audio fingerprint information in pickle files

With Flask the application has a routes file that contains the routes used for the application, they can tell the application what each route used can accomplish and what file they can use, for example telling the default route to use the index.html file. In my Flask route file as in seen in Figure 21 below, a folder is created called uploads which is where the audio clip is stored, and the extension wav is the allowed file in this folder. The default route for the application is / and returns the index.html file by using render_template which goes to the template folder and finds the file named index.html which can be seen in Figure 22 below.

```
8  UPLOAD_FOLDER = 'uploads'
9  ALLOWED_EXTENSIONS = {'wav'}
10
11 def allowed_file(filename):
12     return '.' in filename and \
13         filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Figure 21: Route.py file allowing file uploads

```
80 @app.route("/", methods=['POST', 'GET'])
81 def index():
82     return render_template('index.html')
```

Figure 22: Route File default route

The next route is used when the user sends the audio clip to the server called upload_blob, first the route checks to see if the request method is a POST, if the request is a POST request, then it continues with two messages displaying in the console saying the application has received the audio clip and the location of the audio file. The next thing the route does is check if audio_data is located in the upload folder and if it is not then an error message is displayed saying no file part. Then in the console a message is displayed saying the file has been upload successfully, along with the uploaded files name, then filepath is created which stores the path to the file along with the file name. This is used by the model, a prediction variable is made which says to use the model and use the function predict and that function should use the filepath, next jsonify is used on the prediction and then is the return response, which is displayed in the below Figure 23.

```

48 @app.route('/upload_blob', methods=['GET', 'POST'])
49 def upload_blob():
50     print("request")
51     if request.method == 'POST':
52         print("Recieved Audio File")
53         print(request.files)
54         if 'audio_data' not in request.files:
55             return jsonify({"error" : "No file part"})
56
57         file = request.files['audio_data']
58
59         if file.filename == '':
60             return jsonify({"error" : "No selected file"})
61
62         if file and not allowed_file(file.filename):
63             return jsonify({"error" : "No selected file"})
64         print('file uploaded successfully')
65
66         print("Upload filename = " + file.filename)
67         filename = secure_filename(file.filename)
68         filepath = os.path.join(UPLOAD_FOLDER, filename)
69         file.save(filepath)
70
71         prediction = model.predict(filepath)
72         return jsonify(prediction)

```

Figure 23: Route File upload_blob route

After all the imports are added in the model the pickle files and song list are opened and loaded into the model to be used as the database as seen in Figure 24 below. Then a spectrogram graph function is created as a layout, after that a function which converts byte strings into numpy array is created, which is displayed in Figure 25 below. A function is created that converts stereo to mono as seen in Figure 26 below. All these functions will be used later in the model.


```

1  import wave
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.mlab as mlab
5  from scipy import signal
6  from scipy.io import wavfile
7  import pylab
8  import matplotlib.mlab as ml
9  from scipy.fftpack import fft
10 import struct
11 import pickle
12 from sys import byteorder
13 from array import array
14 from struct import pack
15
16 #pickle files containing the songID and audio fingerprints are loaded
17 with open("app/hashtable.pkl", "rb") as input_file:
18     hashtable = pickle.load( input_file)
19 with open("app/songid.pkl", "rb") as input_file:
20     song_dict = pickle.load(input_file)
21
22 file = open('app/songs.txt','r')#list of songs to use
23 allsongs = file.readlines()
24 allsongs = [a[:-1] for a in allsongs]
25 numOfSongs = len(allsongs)

```

Figure 24: Model file imports and loading pickle files

```

33 def graph_spectrogram(sound_info, frame_rate):
34     pylab.figure(num=None, figsize=(19, 12))
35     pylab.subplot(111)
36     pylab.specgram(sound_info, Fs=frame_rate)
37     pylab.savefig('spectrogram.png')
38
39     """
40     Function that converts a byte string into a numpy array
41     """
42     def _wav2array(nchannels, sampwidth, data):
43         num_samples, remainder = divmod(len(data), sampwidth * nchannels)
44         if remainder > 0:
45             raise ValueError('The length of data is not a multiple of '
46                               'sampwidth * num_channels.')
47         if sampwidth > 4:
48             raise ValueError("sampwidth must not be greater than 4.")
49
50         if sampwidth == 3:
51             a = np.empty((num_samples, nchannels, 4), dtype=np.uint8)
52             raw_bytes = np.fromstring(data, dtype=np.uint8)
53             a[:, :, :sampwidth] = raw_bytes.reshape(-1, nchannels, sampwidth)
54             a[:, :, sampwidth:] = (a[:, :, sampwidth - 1:sampwidth] >> 7) * 255
55             result = a.view('<i4').reshape(a.shape[:-1])
56         else:
57             dt_char = 'u' if sampwidth == 1 else 'i'
58             a = np.fromstring(data, dtype='<%s%d' % (dt_char, sampwidth))
59             result = a.reshape(-1, nchannels)
60         return result

```

Figure 25: Model file byte to numpy array

```

66 def stereo2mono(audiodata, nchannels):
67     #     if nchannels==1:
68     #         return audiodata.astype(int)
69     audiodata = audiodata.astype(float)
70     d = audiodata.sum(axis=1) / 2
71     return d.astype(int)

```

Figure 26: Model file stereo to mono function

After all these functions have been declared in the model a class called song is created, here all the information made in the functions above are used inside this class. The first function in the class is an init function which uses the data from the song_id pickle file, it reads the rate of the song, the channel, frames and so on. It also uses the function stereo2mono that was declared earlier. The next function used is the spectrogram which will use the data from the song and convert it into a spectrogram, all of this is displayed in Figure 27 below. The function that is used next in the song class is the find_key function, this function creates the data for the spectrogram which is shown in the below Figure 28.

```

78 class song:
79     def __init__(self, file, songid):
80         wav = wave.open(file)
81         self.song_id = songid
82         self.title = file.split("/")[-1]
83         self.rate = wav.getframerate()
84         self.nchannels = wav.getnchannels()
85         self.sampwidth = wav.getsampwidth()
86         self.nframes = wav.getnframes()
87         self.data = wav.readframes(self.nframes)
88         self.array = stereo2mono(_wav2array(self.nchannels, self.sampwidth, self.data), self.nchannels)
89         wav.close()
90     def spectrogram(self):
91         self.specgram, self.frequencies, self.times = m1.specgram(self.array, Fs=self.rate, NFFT = 4096,
92             window = m1.window_hanning, noverlap = int(4096 * 0.5), mode='magnitude')
93         self.specgram = 10*np.log10(self.specgram)
94         self.specgram[self.specgram== -np.inf] = 0

```

Figure 27: Model file song class, init and spectrogram function

```

95     def find_key(self):
96         self.spectrogram()
97         all_times = self.specgram.transpose()
98         bands = []
99         count = 0
100        for a in all_times:
101            l = []
102            x = max(a[0:10])
103            l.append((x, [list(a[0:10]).index(x),self.times[count]]))
104            x = max(a[10:20])
105            l.append((x, [list(a[10:20]).index(x)+10,self.times[count]]))
106            x = max(a[20:40])
107            l.append((x, [list(a[20:40]).index(x)+20,self.times[count]]))
108            x = max(a[40:80])
109            l.append((x, [list(a[40:80]).index(x)+40,self.times[count]]))
110            x = max(a[80:160])
111            l.append((x, [list(a[80:160]).index(x)+80,self.times[count]]))
112            x = max(a[160:510])
113            l.append((x, [list(a[160:510]).index(x)+160,self.times[count]]))
114            bands.append(l)
115            count+=1
116        l = []
117        for a in bands:
118            for b in a:
119                l.append(b[0])
120        mean = .1*np.mean(l)
121        new_bands = []
122        for i in range(0, len(bands)):
123            a = bands[i]
124            m = [t[1] for t in a if t[0]>mean]
125            if len(m)!=0:
126                new_bands.append(m)
127        self.bands = new_bands

```

Figure 28: Model file find_key function

The function that comes next is the `cal_address` function, which creates `target_zones` from the previous `find_key` function, these are used in the search function to find a match for the song, which can be seen in Figure 29 below. The search function creates an array for `match_couples`, the function uses the key to see if it is in the hashtable pickle file and if it is the key from the hashtable is added to the `matched_couple` array. Next `target_zone_keys` are created which contains data from the `matched_couple` array, after that a `matched_target_zone` array is created this see if the data from the `target_zone_keys` and matches it and prints out the data of `matched_target_zone` which is displayed in the below Figure 30.

```
143     def cal_address(self):
144         new_bands = []
145         for ele in self.bands:
146             for sub in ele:
147                 new_bands.append(sub)
148         self.addresses = {}
149         target_zone = 0
150         for i in range(3, len(new_bands)-4):
151             anchor_point = new_bands[i-3]
152             for ele in new_bands[i:i+5]:
153                 diff = float("%.2f"%(ele[1] - anchor_point[1]))
154                 val = float("%.2f"%(anchor_point[1]))
155                 if (anchor_point[0], ele[0], diff) not in self.addresses.keys():
156                     self.addresses[anchor_point[0], ele[0], diff] = []
157                     self.addresses[anchor_point[0], ele[0], diff].append([val, target_zone])
158             target_zone+=1
159         print(target_zone)
```

Figure 29: Model file `cal_address` function

```

160 def search(self):|
161     matched_couples = []
162     for key in self.addresses:
163         if key in hashtable.keys():
164             matched_couples.append(hashtable[key])
165
166     count = [0 for i in range(numOfSongs)]
167     target_zone_keys = {}
168     for ele in matched_couples:
169         for subele in ele:
170             count[subele[1]]+=1
171             tup = (subele[1],subele[2])
172             if tup not in target_zone_keys.keys():
173                 target_zone_keys[tup] = 0
174             target_zone_keys[tup]+=1
175
176     matched_target_zone = [0 for i in range(numOfSongs)]
177     for key in target_zone_keys:
178         if target_zone_keys[key]==5:
179             matched_target_zone[key[0]]+=1
180     print('matched target zone ',matched_target_zone)
181     return matched_target_zone.index(max(matched_target_zone))

```

Figure 30: Model file search function

The final part of the model is the predict function, which is called in the upload_blob route, here the model gets the filepath information and retrieves the file, then it uses the find_key function, cal_address function and the search function. When the search is being carried out on line 192, the model is being told to remove the .wav from the result, then afterwards the predicted song is told to replace every space with %20, this is done so it can be used for the link results. After this a YouTube link variable is created which contains the first part of a YouTube link, this is then but into a join statement which adds the youtube_link to the predicted song, the same is done for the guitar tab. After this these results are places into a return which is watch the response used by the upload_blob route as displayed in the Figure 31 below.

```

183 def predict(filepath):
184
185     #read file contents
186     total = song(filepath, 0)
187     total.find_key()
188     total.cal_address()
189     idx = total.search()
190
191     #The .wav is removed from the song in the songs.txt file,
192     #and spaces are replaced with %20 so the repsonse can be used as a link
193     remove_wav = (allsongs[idx]).replace('.wav', '')
194     predicted_song = remove_wav.replace(' ', '%20')
195
196     youtube_link = "https://www.youtube.com/results?search_query="
197     youtube_song_link = "".join((youtube_link, predicted_song, "%20guitar%20lesson"))
198
199     guitar_tab_link = "https://www.ultimate-guitar.com/search.php?search_type=title&value="
200     guitar_tab_song_link = "".join((guitar_tab_link, predicted_song))
201
202     return {
203         "Song Prediction" : remove_wav,
204         "YouTube Link" : youtube_song_link,
205         "UltimateGuitar Link" : guitar_tab_song_link
206     }

```

Figure 31: Model file predict function

Due to how the audio recorder is formatted the user will not get an accurate prediction, for the user to get an accurate prediction they will need to upload an audio clip from the Jupyter Notebook recorder. In the Flask route file there is a route called /upload which is similar to the /upload_blob route. If the file selected is not the correct file type an error message is displayed in the saying no file part and the user is redirect to the /upload page. If the user clicks upload without uploading a file, the browser will submit an empty file without a file name and a message is displayed saying no file was selected and the user is redirected. If everything is successful, the user has uploaded a file and the file is the correct file type then similar to the /upload_blob route the file is added to the filepath and is sent to the model and return as a json response, as displayed in the below Figure 32.

```

15 @app.route('/upload', methods=['GET', 'POST'])
16 def upload_file():
17     if request.method == 'POST':
18         # check if the post request has the file part
19         if 'file' not in request.files:
20             flash('No file part')
21             return redirect(request.url)
22         file = request.files['file']
23         # If the user does not select a file, the browser submits an
24         # empty file without a filename.
25         if file.filename == '':
26             flash('No selected file')
27             return redirect(request.url)
28         if file and allowed_file(file.filename):
29             print("Upload filename = " + file.filename)
30             filename = secure_filename(file.filename)
31             filepath = os.path.join(UPLOAD_FOLDER, filename)
32             file.save(filepath)
33
34             prediction = model.predict([filepath])
35             print(prediction)
36             return jsonify(prediction)
37     return render_template('upload.html')

```

Figure 32: Flask Route file /upload route

5.4 Frontend

As stated previously, with Flask to display the frontend `render_template` is used with the HTML file name that is to be used for a route, which can be seen in the above Figure 15 on line 37. After the route has been told what HTML file to use the application goes to the template folder and goes to the give HTML file. With Flask for the HTML file to use a CSS or JavaScript file in the href and src an `url_for` has to be used to link the files to the HTML, inside the `url_for` for the application has to state to look for the files in the static folder, then the filename is given, as displayed in Figure 33 below for the CSS file the application is told to look in the styles folder and use the `app.css` file and to use the JavaScript file look in the scripts folder and use the `app.js` file.

```

6 <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='styles/app.css') }}">
7 <script type="text/javascript" src="{{url_for('static', filename='scripts/app.js')}}"></script>

```

Figure 33: Using CSS and JavaScript files with Flask

With the `index.html` file, which is the default route, the page contains a div for the navbar which navigates to the home page and the upload page for uploading an audio file. After this the buttons for controlling the recorder are implemented, both the pause and stop buttons are disabled by default, under the buttons there is a div for displaying the format of the recording, before the recording is started there is a message stating the format will appear when recording starts, after the recording has started this message will be replaced with the formatting details. Next after the

recording has completed the recorded audio clip will be displayed where the user can play back the recording, download the audio clip locally or upload the clip to the server. The user can make as many new recordings as they want, and the previous recording will still be displayed on the page. When the user sends the audio clip to the server and after the server has made a prediction the results are displayed in the results div. The below Figures 34 and 35 display the websites default page before recording has started and after, Figure 36 shows the code for displaying this.

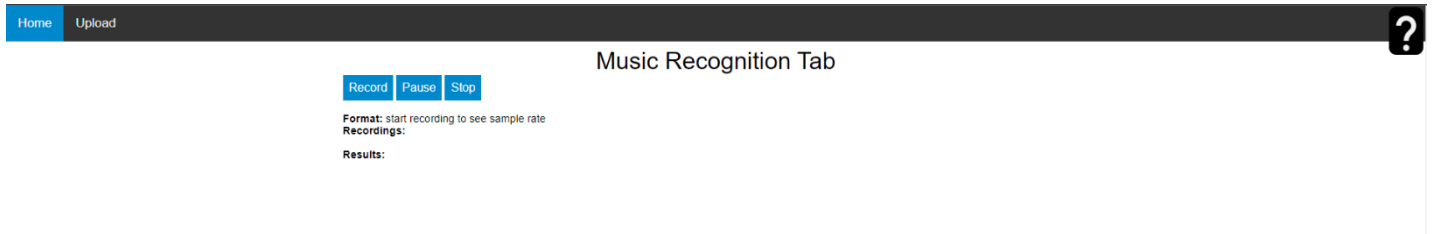


Figure 34: Home page before recording

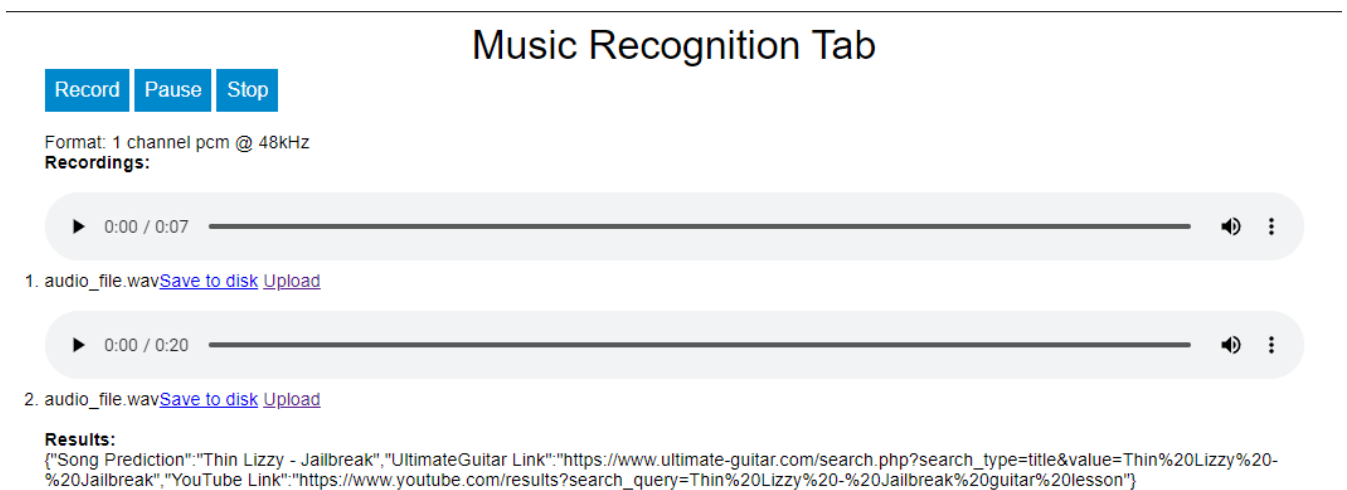


Figure 35: Home page after recording


```

<div class="topnav">
  <a class="active" href="/">Home</a>
  <a href="/upload">Upload</a>
</div>

<div class="wrapper">
  <h1>Music Recognition Tab</h1>

  <div id="controls">
    <button id="recordButton">Record</button>
    <button id="pauseButton" disabled>Pause</button>
    <button id="stopButton" disabled>Stop</button>
  </div>
  <br>
  <div id="formats"><p>
    <strong>Format:</strong> start recording to see sample rate</p>
  </div>
  <p><strong>Recordings:</strong></p>
  <ol id="recordingsList"></ol>
  <div>
    <br>
    <p><strong>Results:</strong></p>
    <p id="result">{{ prediction }}</p>
  </div>
</div>

```

Figure 36: Code from index.html for displaying recording details and results

Last with the index.html page, a toggle button is created which is displayed in the top right corner of the page, when clicked it displays information about the application, how to use it, the songs available for testing and errors that occur with the application and explaining how to work around them, the information displaying on the page is displayed in Figure 37 below and in Figure 38 the code for it is displayed.

Information

This application uses the microphone and listens to a song for 20 seconds, then you can click the upload link to send the audio clip to the server to make a prediction. The application will search the database for a match. Along with the song result, links from YouTube and UltimateGuitar Tab are displayed so the user can learn the song on guitar. The application database only contains a few songs to test the prediction, they are Jailbreak by Thin Lizzy, City of Blinding Lights by U2, Hey by Pixies, Monkey Man by The Rolling Stones, Hey Jude by The Beatles and Everlong by Foo Fighters. Due to a problem with the recorder format Jailbreak by Thin Lizzy is the only response returned, even if there is no music playing, because of this the user has to go to the upload tab in the navbar and upload the audio clip from the python recorder to get a prediction. The python audio clips are found in the /uploads folder in the Flask App, the files are named after the band.

Figure 37: The home page toggle button information

```

35 <label for="toggle">?</label>
36 <input type="checkbox" id="toggle">
37 <aside>
38   <h2>Information</h2>
39
40   <p>This application uses the microphone and listens to a song for 20 seconds,
41     then you can click the upload link to send the audio clip to the server
42     to make a prediction. The application will search the database for a match.
43   </p>
44
45   <p>Along with the song result, links from YouTube and UltimateGuitar Tab are
46     displayed so the user can learn the song on guitar.</p>
47
48   <p>The application database only contains a few songs to test the prediction,
49     they are Jailbreak by Thin Lizzy, City of Blinding Lights by U2, Hey by
50     Pixies, Monkey Man by The Rolling Stones, Hey Jude by The Beatles and
51     Everlong by Foo Fighters.
52   </p>
53
54   <p>Due to a problem with the recorder format Jailbreak by Thin Lizzy is the
55     only response returned, even if there is no music playing, because
56     of this the user has to go to the upload tab in the navbar and upload the
57     audio clip from the python recorder to get a prediction.
58     The python audio clips are found in the /uploads folder in the Flask App,
59     the files are named after the band.
60   </p>
61 </aside>

```

Figure 38: Code from the index.html page for the toggle button

The last HTML file is the upload.html, similar to the index.html file it contains the same div for the navbar. For the uploading a file a form is used, declaring the method as a POST, and using the input type as file so when clicked the user can choose a file from their device. After the user has choose the file, they want to get a prediction for there is a submit button that sends the audio file to the server. The results are the only thing displayed to the user; the results displayed as JSON. Figure 39 shows the upload.html page, while Figure 40 displays the code for the file form on upload.html.

Upload File

Results

Figure 39: Upload page

```
16  ✓ <header>
17    | <h1>Upload File</h1>
18    | </header>
19    | <br>
20  ✓ <form method=post enctype=multipart/form-data class="form_wrapper">
21    | <input class="file" type=file name=file>
22    | <input class="upload" type=submit value=Upload id="filepath">
23    | </form>
24    | <br>
25  ✓ <div class="results">
26    | <h3>Results</h3>
27    | <p id="results"></p>
28    | </div>
```

Figure 40: Code from the upload.html file for the file form

The last area of the frontend is the JavaScript file which allows the application to record audio, send an audio blob to the server and displays the results on the index.html page. Two other methods of creating a recorder for the application were tried but didn't work, the first one used a JavaScript API called MediaRecorder. It worked well in creating an application that would record an audio clip and allowing users to playback the clip. However, it was discovered when the recording was completed, and the application tried to automatically send the audio blob to the server it would return a response saying the file had no Riff ID. Later it was discovered this error occurred because the MediaRecorder API didn't work with wav files only ogg files.

Then with the second method with which was similar to the first method, it created a recording application well, but when the audio blob was sent to the server it would return with the response No File Part. With some examination it was discovered that the recorder code was not actually saving the audio blob, so when the server would go to the upload folder to retrieve the audio file to identify it could not, giving the response No File Part.

With the working recording code in the app.js file, the first thing done is declaring the variables, these include setups for gumStream for getUserMedia, audioContext which helps record and the button variables are declared and given event listeners. This means an event has been added to the button, in this case if the user clicks the buttons the function it has been told to do will occur, for example the event listener for the recordButton has been told when it is clicked it will run the startRecording function, which can be seen on line 17 in the below Figure 41.

```
1 //webkitURL is deprecated but nevertheless
2 URL = window.URL || window.webkitURL;
3
4 var gumStream; //stream from getUserMedia()
5 var rec; //Recorder.js object
6 var input; //MediaStreamAudioSourceNode we'll be recording
7
8 // shim for AudioContext when it's not avb.
9 var AudioContext = window.AudioContext || window.webkitAudioContext;
10 var audioContext //audio context to help us record
11
12 var recordButton = document.getElementById("recordButton");
13 var stopButton = document.getElementById("stopButton");
14 var pauseButton = document.getElementById("pauseButton");
15
16 //Events have been given to these buttons
17 recordButton.addEventListener("click", startRecording);
18 stopButton.addEventListener("click", stopRecording);
19 pauseButton.addEventListener("click", pauseRecording);
```

Figure 41: JavaScript app.js file showing variables being declared and eventListeners added to buttons

Next in the app.js file a function is created for the pause recording, here the first thing the function does is display if the button was clicked in the console by saying pause button clicked equals true. Then an if else statement is used, if the pause button is clicked the recording will stop and the text in the pause button will change from Pause to Resume. The else statement will continue with the recording and make the pause button text return to Pause, in Figure 42 displayed below the pause button changing to the resume button and Figure 43 displays the code for the pauseRecording function.



Figure 42: Home page pause button changing to resume

The function after the pauseRecording is the stopRecording function, when the stop button is clicked in the console it is displayed that the stop button was clicked, just like with the pause, and start recording buttons. After it is displayed in the console the stop and pause buttons are disabled and start recording button is enabled, this is done by giving on line 39 stopButton.disabled = true. Then the function is told to stop the recording and the clearTimeout is used to stop the record timer. Next the gumStream stops access to the microphone, then on line 56 an audio blob is created of the recording and exported as a wav and sent to the function createDownloadLink. In the Figure 44 displayed below the code for the stopRecording is shown.

```
21 function pauseRecording(){
22     console.log("pauseButton clicked rec.recording=",rec.recording );
23     if (rec.recording){
24         //pause
25         rec.stop();
26         pauseButton.innerHTML="Resume";
27     }else{
28         //resume
29         rec.record()
30         pauseButton.innerHTML="Pause";
31     }
32 }
33 }
```

Figure 43: pauseRecording function from app.js

```
35 function stopRecording() {
36     console.log("stopButton clicked");
37
38     //Stop button is disabled, record button is enabled allowing new recordings
39     stopButton.disabled = true;
40     recordButton.disabled = false;
41     pauseButton.disabled = true;
42
43     //If the recording is paused the reset button can be pressed to rerecord
44     pauseButton.innerHTML="Pause";
45
46     //Recorder is told to stop
47     rec.stop();
48
49     //When stop button is pressed the limit timer is cleared
50     clearTimeout(recordTimeout);
51
52     //Microphone access is stopped
53     gumStream.getAudioTracks()[0].stop();
54
55     //Wav blob created and passed into createDownloadLink
56     rec.exportWAV(createDownloadLink);
57 }
```

Figure 44: stopRecording function from app.js

This next function is the startRecording function, and like the previous functions it starts with the console displaying the start recording button has been clicked, then an object constraint has been

declared giving only permission to be used for audio and do not use video. Then like the stopRecording function the record button is disabled while both the pause and stop button are enabled. Next the function uses the microphone from the device and uses the constraints that were created earlier and a then statement is used to use a function called stream. The format for the recording is then updated and stream has been assigned to gumStream to be used for the input audio context on line 76. The recorder object is then created and is configured to for recording mono sound, then the function is told to start the recording process, and, in the console, it is displayed that recording has started and a setTimeout is created which allows the recording function to continue for 20 seconds and after the time to run the stopRecording function. After all that the catch statement is then used and says if there is an error in getting a device for recording reset the buttons, making the recording button enabled and the pause and stop buttons disabled. This is all displayed in the Figure 45 below.

```
59 function startRecording() {
60     console.log("recordButton clicked");
61     //Constraint object telling the recording to use audio not video
62     var constraints = { audio: true, video:false }
63     //Record button is disabled until getUserMedia() returns a success or fail response
64     recordButton.disabled = true;
65     stopButton.disabled = false;
66     pauseButton.disabled = false
67     navigator.mediaDevices.getUserMedia(constraints).then(function(stream) {
68         console.log("getUserMedia() success, stream created, initializing Recorder.js ...");
69         audioContext = new AudioContext();
70         //update the format
71         document.getElementById("formats").innerHTML="Format: 1 channel pcm @ "
72         |audioContext.sampleRate/1000+"kHz"
73         //gumStream is assigned for later use
74         gumStream = stream;
75         //Stream is used
76         input = audioContext.createMediaStreamSource(stream);
77         //Recorder object is created and configured to record mono sound, which is 1 channel.
78         //Recording 2 channels doubles the file size
79         rec = new Recorder(input,{numChannels:1})
80         //start the recording process
81         rec.record()
82         console.log("Recording started");
83         //Recording is set to stop after 20 seconds
84         recordTimeout = setTimeout(stopRecording, 21 * 1000);
85     }).catch(function(err) {
86         //If getUserMedia() fails the record button is enabled
87         recordButton.disabled = false;
88         stopButton.disabled = true;
89         pauseButton.disabled = true
90     });
91 }
```

Figure 45: startRecording function from app.js

The last function from the app.js file is the createDownloadLink function, displayed in Figure 29 below, which also sends the audio blob to the server and helps to display the results on the index.html page. The function is told to use the blob that was created earlier, the function firsts create variables, the first one created is an url variable which is an URL object used to make the blob a link, the next variable is an audio element, a list element is created, and a link element is created

as a variable. Lastly a filename variable is created which contains the name for the file which is `audio_file.wav`, after declaring the filename controls are added to the audio variables and save to disk link is created. The other variables are added to the list variable, so when the audio element is displayed on the application along with it the filename name will be displayed the link to save to disk will also be displayed next to the filename. The next part of the function is the upload link, an upload variable is created which uses a link element, a `# href` is used for the upload link on line 131, the link is called Upload and an event listener is added to the upload link. When the link is clicked it will use the function `event.onload` displays the response of the event listen, in the console the results are displayed of the song, the `getElementById` is used to display the results of the song on the `index.html` page. When the upload link is clicked it sends the blob as `audio_data` containing the filename on line 143 and sends the request as a POST to `/upload_blob`. The last bit of the function creates a space between the save to disk link and upload link and a list of the recordings is added to the list variable. The Figure 46 displayed below show the code for the `createDownloadLink` function.

```

112 function createDownloadLink(blob) {
113     var url = URL.createObjectURL(blob);
114     var au = document.createElement('audio');
115     var li = document.createElement('li');
116     var link = document.createElement('a');
117     //filename contains the .wav file that is used for upload and download
118     var filename = "audio_file.wav";
119     //Controls are added to the audio element
120     au.controls = true;
121     au.src = url;
122     //save to disk link
123     link.href = url;
124     link.download = filename; //Forces the browsers to download the file from filename
125     link.innerHTML = "Save to disk";
126     li.appendChild(au); //New audio element is added to li
127     li.appendChild(document.createTextNode(filename)) //filename is added to the li
128     li.appendChild(link); //Save to disk is added to the li
129     //upload link
130     var upload = document.createElement('a');
131     upload.href="#";
132     upload.innerHTML = "Upload";
133     upload.addEventListener("click", function(event){
134         var xhr=new XMLHttpRequest();
135         //Displays the reponse prediction from the model
136         xhr.onload=function(e) {
137             if(this.readyState === 4) {
138                 console.log("Server returned: ",e.target.responseText);
139                 document.getElementById("result").innerHTML = e.target.response;
140             }
141         };
142         var fd=new FormData();
143         fd.append("audio_data",blob, filename);
144         xhr.open("POST", "/upload_blob",true);
145         xhr.send(fd);
146     })
147     li.appendChild(document.createTextNode(" ")) //Add a space in between
148     li.appendChild(upload) //Add the upload link to li
149     recordingsList.appendChild(li); //add the li element to the ol
150 }

```

Figure 46: createDownloadLink function from app.js

5.5 Conclusion

The implementation chapter examines how the code and features of the application were implemented, the development environment and how the database was created. In this chapter the model that identifies the song is also examined and how the model identifies the songs are explained. Later in the chapter the JavaScript recorder is also explained how it works.

6. Testing

The testing chapter is used for testing the functionality/usability of the application with users and have them fill out a survey about their time using the application. The chapter also includes testing the algorithm of the application, to see if its accurate and works as intended, also the performance of the application is tested too.

6.1 Usability Testing

User testing was carried out with two users, they were given tasks to complete with the application. The first task the testers had to do was navigate to the application, then click the question mark button to read the information about the application. The second task the testers would complete is to click the record button and let the application record for 6 seconds then click the pause button for 6 seconds and resume the recording until the recording is completed. Then the tester would click the record button again and let the application record for 6 seconds again, then the tester has to click the stop button, then for the last time the tester clicks the record button and lets the application finish recording. The third task the testers would complete is to playback one of the recordings that was created, and then they clicked the upload link to view the results. Once the tester had the results, they would use one of the lesson links to navigate to a lesson. The last task the tester had to complete is to navigate to the upload page from the navbar, then choose a file to upload to get a prediction from and click the upload button, like with the previous task when the results show the tester has to navigate to one of the lesson links.

After the testers completed the tasks given to them, they were asked to complete a survey about their experience, the first question from the survey was to rate their experience from a scale of 1-10 on using the application, one gave a score of 9 and the other gave a 6, as displayed in Figure 47 below.

From a scale from 1-10 how did you find your experience using the application?

2 responses

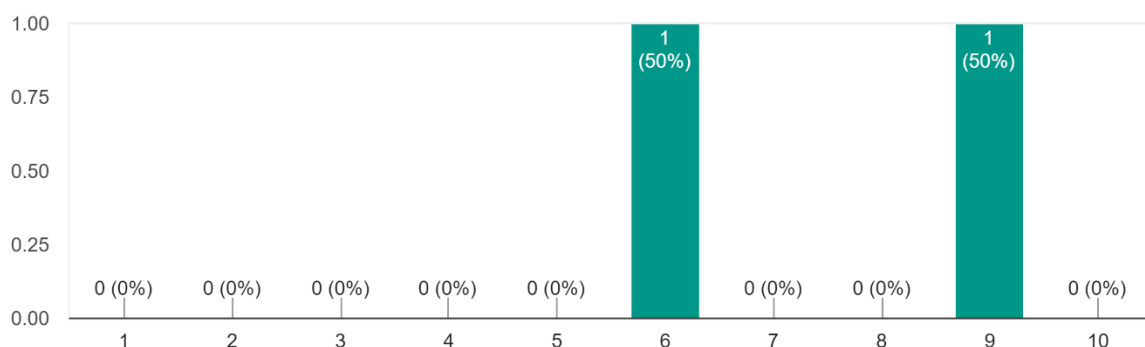


Figure 47: Post Testing survey about application experience

The second question the testes were asked was another scale from 1-10 question, which asked them how the found the usability of the application, they rated it a 9 and 8, as shown in Figure 48 below.

From a scale from 1-10 how did you find the usability of the application?

2 responses

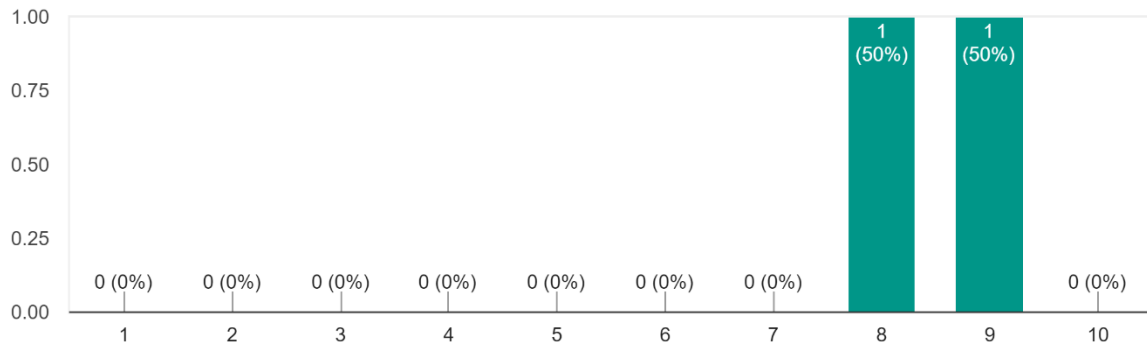


Figure 48: Post Testing Survey rating of usability of application

Next the testers were asked about the design and layout of the application, which is another rating question from a scale from 1-10, the testers rated it a 9 and a 7, which is displayed below in Figure 49.

From a scale from 1-10 how would you rate the design and layout of the application?

2 responses

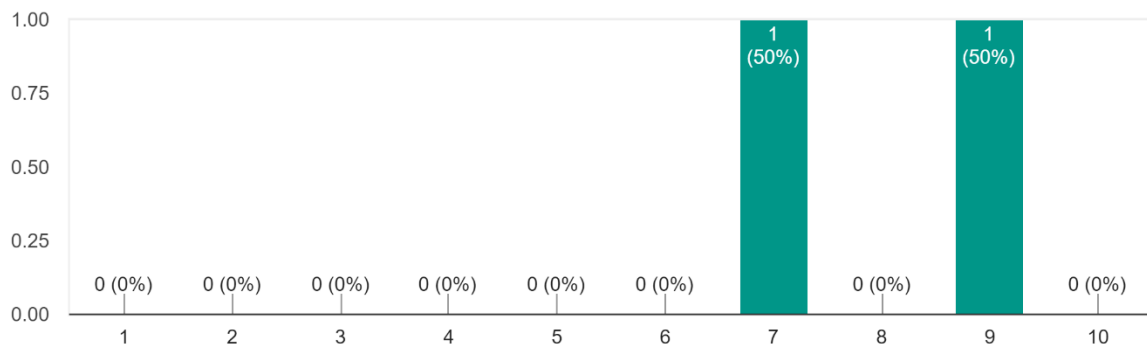


Figure 49: Post Testing Survey rating of design and layout of application

The follow up question asked the testers how they found the tasks that were given to them to complete, as shown in Figure 50 below, the pie chart shows that both the testers found the tasks simple to complete.

How did you find the tasks given to you for testing the application?

2 responses



Figure 50: Post Testing Survey pie chart of difficulty of tasks

The question the testers were asked next was to see what they liked about the application, one said they liked that the results linked to YouTube for lessons, the other tester like that the application was simple to use, these responses can be seen in Figure 51 below.

What did you like about the application?

2 responses

Good functionality linking to youtube

Simple to use

Figure 51: Post Testing Survey the things testers liked about the application

The next question is similar to the previous question, the testers were asked what they disliked about the application, one said they wanted a bigger database of music to choose from, while the other said the design of the application was simple and would prefer nicer colours and patterns, as shown in Figure 52 below.

What did you dislike about the application?

2 responses

Needs bigger database of music.

The design a little simple, could of done with some nice colour or pattern

Figure 52: Post Testing Survey what testers disliked about the application

The testers were then asked from a scale from 1-10 how likely they would be to use the application, they gave a rating of 9 and a 7, as shown from the bar chart in Figure 53 below.

From a scale from 1-10 how likely would you be to use this application?

2 responses

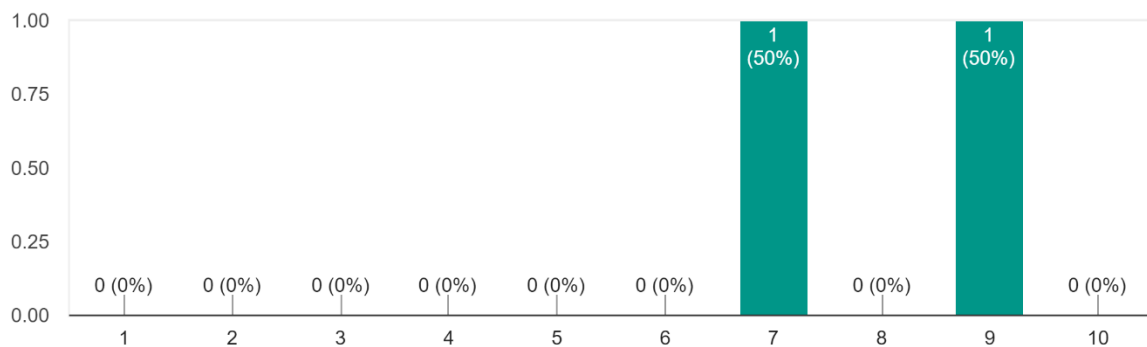


Figure 53: Post Testing Survey asking testers how likely they would use the application

Testers were then asked about how likely they would recommend the application to a person, the rating results is the same as the previous question, one gave a rating of 9 and the other gave a 7, as displayed in Figure 54 below.

From a scale from 1-10 how likely would you be to recommend this application to someone?

2 responses

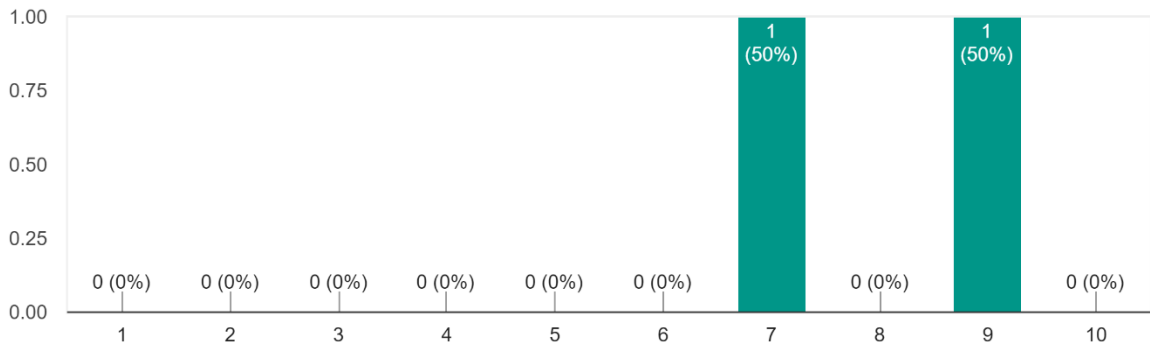


Figure 54: Post Testing Survey on recommending the application

The final question the testers were asked was if they had any suggestion for the application, one testers response was for a designed dashboard for the application, the other tester said a user page that had a better and interesting design, these responses are displayed in Figure 55 below.

If any, what would you recommend to be added, changed or removed for a better experience?

2 responses

- Designed front end dashboard.
- The user page with more interesting design

Figure 55: Post Testing Survey recommendation for improving the application

6.2 Unit/Integration/Feature Testing

This part of the application testing is to test the features of the application and to see if the pages of the application are working, in the below Figure 56 shows the get request of pages, scripts and CSS file used by the application. The 200 after the request means it is working, the only one that has a 404 meaning it is not working is favicon.ico.

```
127.0.0.1 - - [16/Apr/2022 16:24:26] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [16/Apr/2022 16:24:26] "[37mGET /static/styles/app.css HTTP/1.1[0m" 200 -
127.0.0.1 - - [16/Apr/2022 16:24:26] "[37mGET /static/scripts/app.js HTTP/1.1[0m" 200 -
127.0.0.1 - - [16/Apr/2022 16:24:26] "[33mGET /favicon.ico HTTP/1.1[0m" 404 -
127.0.0.1 - - [16/Apr/2022 16:24:34] "[37mGET /upload HTTP/1.1[0m" 200 -
127.0.0.1 - - [16/Apr/2022 16:24:35] "[37mGET / HTTP/1.1[0m" 200 -
```

Figure 56: Unit Testing Get Requests

When testing the recording feature from the application the song Hey by Pixies was used, but due to the way the microphone is formatted in the app.js file the algorithm cannot make an accurate prediction of the song, after it gets the audio fingerprints the matched zone all are 0, and since the first song in the database is Jailbreak by Thin Lizzy, this is returned. This is the case even when there is no song playing into the microphone. The below Figure 57 shows the audio clip being sent to the server and no match zones being found.

```
Recieved Audio File
ImmutableMultiDict([('audio_data', <FileStorage: 'audio_file.wav' ('audio/wav')>)])
file uploaded successfully
Upload filename = audio_file.wav
1091
matched target zone [0, 0, 0, 0, 0, 0]
127.0.0.1 - - [16/Apr/2022 16:45:36] "[37mPOST /upload_blob HTTP/1.1[0m" 200 -
```

Figure 57: Recording testing Results

The next test carried out on the application was the pause/resume, stop and the upload to sever button. The Figure 58 shown below is the home page of the application, and shown on the page are the record, pause and stop buttons, as well as the format information of the recording, the recordings made with the option to save the file to disk and the upload to server link. Below the records the results of the prediction are displayed.

Music Recognition Tab

Record
Pause
Stop

Format: 1 channel pcm @ 48kHz

Recordings:

▶
0:00 / 0:07

🔊
⋮

1. audio_file.wav [Save to disk](#) [Upload](#)

▶
0:00 / 0:20

🔊
⋮

2. audio_file.wav [Save to disk](#) [Upload](#)

Results:

```
{
  "Song Prediction": "Thin Lizzy - Jailbreak",
  "UltimateGuitar Link": "https://www.ultimate-guitar.com/search.php?search_type=title&value=Thin%20Lizzy%20-%20Jailbreak",
  "YouTube Link": "https://www.youtube.com/results?search_query=Thin%20Lizzy%20-%20Jailbreak%20guitar%20lesson"
}
```

Figure 58: Application Home Page

```

recordButton clicked app.js:60
getUserMedia() success, stream created, initializing Recorder.js ... app.js:71
▲ ▶ [Deprecation] The ScriptProcessorNode is deprecated. Use AudioWorkletNode instead. (http recorder.js:57
s://bit.ly/audio-worklet)
Recording started app.js:99
pauseButton clicked rec.recording= true app.js:22
pauseButton clicked rec.recording= false app.js:22
stopButton clicked app.js:36
recordButton clicked app.js:60
getUserMedia() success, stream created, initializing Recorder.js ... app.js:71
▲ ▶ [Deprecation] The ScriptProcessorNode is deprecated. Use AudioWorkletNode instead. (http recorder.js:57
s://bit.ly/audio-worklet)
Recording started app.js:99
stopButton clicked app.js:36
Server returned: {"Song Prediction":"Thin Lizzy - Jailbreak","UltimateGuitar Link":"https://ww app.js:149
w.ultimate-guitar.com/search.php?search_type=title&value=Thin%20Lizzy%20-%20Jailbreak","YouTube Link":"http
s://www.youtube.com/results?search_query=Thin%20Lizzy%20-%20Jailbreak%20guitar%20lesson"}
> |

```

Figure 59: Application Home Page Console

The above Figure 59 is from the console of the home page of the application, it shows the record button being clicked and the getUserMedia being accepted so the recording can start. When the pause button is pressed the recording is pause and the pause button is changed to say resume, when the resume button is pressed the recording is continued as can be seen where pauseBtton clicked is true and false, then the record button is clicked so the recording is stopped. The next time the recording button is pressed no other button is clicked, the recording has a time limit of 20 seconds so after the 20 seconds the recorder is stopped. The last part in the console is the server response to the audio clip being sent to the server, the song prediction is sent, with YouTube link on how to play it on guitar and a guitar tab link too.

6.3 Algorithm Testing

The algorithm testing is done in Visual Studio Code's console, where after the audio clip has been sent to the server the results of the algorithm are shown on the website, and in the console of VSCode the audio clip name is shown, the audio fingerprint is shown, the matched target zones and the prediction details of the song, as seen in Figure 60 below. The Figure 60 below was tested using the file upload and choosing the Jupyter Notebook recording.

```

Upload filename = fooFighters.wav
2578
matched target zone [18, 1248, 350, 190, 48, 22]
{'Song Prediction': 'Foo Fighters - Everlong', 'YouTube Link': 'https://www.youtube.com/results?search
_query=Foo%20Fighters%20-%20Everlong%20guitar%20lesson', 'UltimateGuitar Link': 'https://www.ultimate-
guitar.com/search.php?search_type=title&value=Foo%20Fighters%20-%20Everlong'}
127.0.0.1 - - [16/Apr/2022 15:49:34] "[37mPOST /upload HTTP/1.1-[0m" 200 -

```

Figure 60: VSCode Console Algorithm results

When testing the algorithm to see if the database could have more than six songs, the newly added song would either be matched to one of the other songs in the database or it would give back an error message of no match found, due to this there is only six songs the algorithm can predict.

6.4 Conclusion

This testing chapter testes several things from the application, the first being usability which uses testers to test the usability and features of the application. They were given a set of tasks to complete and after they completed those tasks, they were asked to complete a survey about their experience and feedback about the application. Other testing completed during this chapter was unit/feature testing of the application and testing the algorithm.

7. Project Management

The project management chapter discusses how the project was managed throughout development. The project was carried out close to 5 months, from the start of January to start of May 2022. Deadlines were set out for the project, these included deadlines for report chapters, having the application built, adding features to the application, and testing the application.

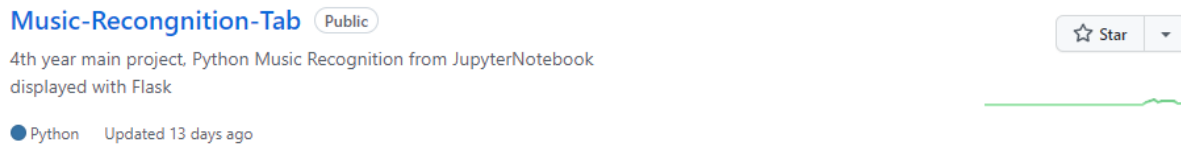


Figure 61: GitHub repository

A GitHub repository was used to store the code, whenever changes were made to the application it was pushed onto GitHub. With every push to GitHub a commit was added, this described what changes had been added to the application or removed from. Figure 61 above displays the GitHub repository used for the project.

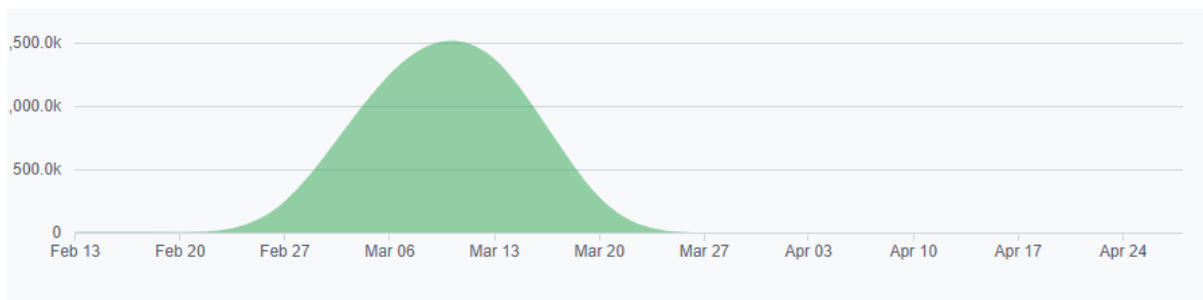


Figure 62: Graph of additions to the repository made from Feb 13th - April 28th, 2022

Figure 62 above displays a graph of additions made to the repository throughout the project. Other data can be displayed on the graph like commits that were made and the deletion of code from the project. It is better to show the addition data as there can be many commits with very little to no code added or removed from the project.

From the above Figure 62 little additions were made at the start of January and February as I was trying to make the application using a few different methods, so the code was stored on a different repository. The highest peak of additions was carried out between the 6th and 13th of March, with a slow decline of additions until the 27th of March.

8. Future Development

This chapter of the report examines any future developments or bug fixes for the application. When applications are built, they are never finished, this can include small fixes of bugs or new features that can be added to the application. In this case with the application some future development which could happen are to make the results display on the home page with a good design with the users being able to click on the links of the lessons.

The application could have a bigger selection of songs in the database for users to get results from. The users could also have a dashboard section, here they could view their history of their results of songs they scanned, as well as a favourites/watch later section where they can save lessons they want to learn later. The users could also have a link where they can add the songs, they scanned to their Spotify account or playlist.

Lastly the major future development for this application is to fix the formatting of the JavaScript recorder so when the wav file is created it has been formatted the same way as the recorder from the Jupyter Notebook. With that bug fixed that means when the user uploads the audio clip to the server it will return the correct prediction.

9. Conclusion

This last chapter of the report is used to summarise the projects development, this includes the strengths of the project, the limitations that occurred and the learning outcomes from the project. Then there is a section for final thoughts about the project.

9.1 Project summary

The aim of this project was to create an application that would use music recognition to identify a sample recording of a song and return the title and artist of the song, it also returns lesson links on how to play the song on guitar from YouTube and a guitar learning site.

The first objective of the project was to do research on music recognition and applications that uses music recognition, like Shazam. The first area of research examines how audio identification works in music recognition, this includes audio fingerprinting which is the process of compacting the audio signal and extracting any relevant features from the audio content. Next in the research audio matching is discussed which analyses the audio data and tries to find a matching result from a database. Then the history of Shazam is examined, this involves the first version of Shazam called CDStar. Lastly in the research chapter, how Flask works was also examined which includes the folder structure and how the server works.

The design chapter discussed the applications system architecture, the applications design which involves the technology used for development, the database design. Next the user interface is examined which examines a wireframe and style guides for the application. The information from the design chapter is then used for the development of the application.

If I could redo the project I would improve and fix several things about the application, firstly I would fix the formatting of the audio recorder so the recorded audio blob could get an accurate prediction. I would improve the database, so users can have more options of songs to get predictions from, I would also improve the design and layout of the site. For example, having the results of the prediction displayed neater and having the links being clickable.

9.2 Limitations

There are several limitations for the project, first being the formatting of the audio recorder. Due to the formatting being different to the one from Jupyter Notebook, when the user uploads the audio clip to the server no accurate match can be found, so instead the first song from the database is retrieved.

The next limitation is the size of the database for the application, it contains 6 songs users can try and get a prediction from. Whenever a 7th song was added to the database the song could not get an accurate reading, it would return one of the other songs from the database. Also, when there was a 7th song, whenever the user tried to get a prediction from one of the other songs it effected the accuracy of the prediction.

Another limitation of the application is when the recording is finished instead of the audio blob automatically being sent to the server to be identified, the user must click the upload link to send the audio blob.

The last limitation is when the result of the prediction is displayed it is displayed on the home page as a JSON, because of this the user can't click the links of the lessons and instead must copy and paste the links to view the lessons.

9.3 Learning Outcomes

The learning outcomes I learned from this project is how music recognition works, as we did not learn about it in classes. I also learned how to use Flask, as I have never used it before. From doing this project though I feel confident in creating another application/project using Flask.

9.4 Final Words

For this project I aimed to create an application that would identify a sample of an unknown song and display the results to users that included the song title, artist, and lessons on how to play the song on guitar. Even though there are some limitations with the application, it still works, and I believe I have accomplished what I set out to complete.

10. References

Bevelacqua, P., 2021. *Fourier Transform*. [online] Thefouriertransform.com

Computers and the Human Genome Project: Smith-Waterman Algorithm, from https://cs.stanford.edu/people/eroberts/courses/soco/projects/computers-and-the-hgp/smith_waterman.html

Everything you need to know about Flask for beginners. (2021), from <https://www.mygreatlearning.com/blog/everything-you-need-to-know-about-flask-for-beginners/>

GitHub - KavyaVarma/Music-Recognition: Algorithms project: Recognize music played from a clip. (2018). From <https://github.com/KavyaVarma/Music-Recognition>

Grosche, P., Müller, M., & Serra, J. (2012). Audio content-based music retrieval. In Dagstuhl Follow-Ups (Vol. 3). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, from <http://www.baskent.edu.tr/~hogul/RA5.pdf>

Intrasonics Team. (2021). How Fast Can Audio Matching Identify Audio Content? From <https://www.bmat.com/audio-fingerprinting-songs-identification/>

Jovanovic, J. (2015, February 2). How does Shazam work? Music Recognition Algorithms, Fingerprinting, and Processing. From <https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition>

Naicu, O. (2018). Using Recorder.js to capture WAV audio in HTML5 and upload it to your server or download locally. From <https://blog.addpipe.com/using-recorder-js-to-capture-wav-audio-in-your-html5-web-site/>

Philip, C., & G, S. (2020). Audio Fingerprinting- Understanding the Concept, Process & Application. From <https://www.pathpartnertech.com/audio-fingerprinting-understanding-the-concept-process-application/>

Using the MediaStream Recording API - Web APIs | MDN. From https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Using_the_MediaStream_Recording_API

Wang, A. (2006). The Shazam music recognition service. *Communications of the ACM*, 49(8), 44-48.
From <https://dl.acm.org/action/downloadSupplement?doi=10.1145%2F1145287.1145312&file=p44-wang.jp.pdf>

What is Flask Python - Python Tutorial. from <https://pythonbasics.org/what-is-flask-python/>

Zhang, J. (2020). Dynamic Time Warping. From <https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd>