



# Intelligent Instrument Interface

Paul Doyle

N00193346

Supervisor: Timm Jeschawitz

Second Reader: Mohammed Cherbatji

Year 4 2022-23

DL836 BSc (Hons) in Creative Computing

## Abstract

This report will comprehensively cover the research, design, and development of the implementation of an AI audio classification model into a video call application. The application allows two users, a professional music/guitar instructor and student, to connect to each other via a peer-to-peer connection to exchange both video and audio data. The purpose of these connections will be to conduct music lessons online, eliminating the distance and allowing users to both learn and teach from the comfort of their own home. Both users will have the capability to send two audio tracks, one from a microphone and the second from a guitar or any other instrument that may be connected to their computer via an audio interface. Users are able to alter the sound of their musical instrument in real time. To further aid in the online lessons, an AI audio classification model has been implemented into the application. This model will take the signal from the instructor's instrument, process it, and display to the student what chord the instructor is playing.

The purpose of this application is to create an alternative video calling application for the teaching and learning of musical instrument online. Taking into account the strengths and weakness of other alternative services to create a peer-to-peer video application with features not found in these services/applications.

Further work that could be carried out include creating a way to integrate third party VSTs and audio software plugins to be integrated into the application so a user may further alter their sound. The model may also be improved on in the future, as at the time of writing it is trained to detect basic guitar chords, while in the future it could be trained to detect more complex chord voicings.

## Acknowledgements

I would like to thank my supervisor, Timm Jeschawitz for his advice and guidance throughout the duration of this project. I would like to thank Mohammed Cherbatji for his input to the development of the project idea and his valuable suggestions in the early stages of development. I would like to thank my friends and classmates Jake Black and Sean Durack Monks, for their feedback and support not just through this development but over the past four years spent together at IADT. I would like to thank any and all who participated in the user testing and those who submitted a response the surveys. I would like to thank my sister Jennifer for her feedback on the report. Most importantly I would like to thank my mother, Pauline, who has supported me and my decision to return to college as a mature student.

**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 leave copies of your own files on a hard disk where they can be accessed by others. 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: Paul Doyle

Signed

A handwritten signature in black ink, appearing to read "Paul Doyle". It is written in a cursive style with a clear 'P' at the beginning and a 'D' at the end.

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

# Table of Contents

1	Introduction .....	1
2	Research.....	2
2.1	Introduction .....	2
2.2	WebRTC.....	3
2.3	Note Detection.....	5
2.3.1	Purpose of Note detection.....	5
2.3.2	Machine Learning.....	5
2.3.3	Neural Networks .....	7
2.3.4	Machine Learning Audio .....	10
2.4	React .....	12
2.4.1	What is React? .....	12
2.4.2	Why React? .....	13
2.5	Conclusion.....	14
3	Requirements.....	15
3.1	Introduction .....	15
3.2	Requirements gathering .....	16
3.2.1	Similar applications .....	16
3.2.2	Survey.....	19
3.3	Requirements modelling.....	19
3.3.1	Personas.....	19
3.3.2	Functional requirements.....	22
3.3.3	Non-functional requirements .....	22
3.3.4	Use Case Diagrams.....	23
3.4	Feasibility .....	24
3.5	Conclusion.....	25
4	Design.....	26
4.1	Introduction .....	26
4.2	Program Design.....	27
4.2.1	Technologies .....	27
4.2.2	Structure of React .....	28
4.2.3	Design Patterns.....	30
4.2.4	Application architecture .....	31
4.2.5	Database Design.....	35
4.2.6	Process design.....	37

4.3	User interface design .....	39
4.3.1	Wireframe .....	39
4.4	Conclusion.....	41
5	Implementation .....	42
5.1	Frontend.....	42
5.1.1	React Redux .....	42
5.1.2	Redux Thunk.....	43
5.1.3	Simple-Peer.....	43
5.1.4	Spectrogram.....	43
5.1.5	Firebase.....	43
5.2	Backend.....	45
5.2.1	Express.js.....	45
5.2.2	Joi .....	45
5.2.3	MongoDB/Mongoose.....	45
5.2.4	Bcrypt.js.....	45
5.2.5	JSON Web Token.....	45
5.2.6	Socket.IO .....	45
5.2.7	Heroku.....	46
5.3	Application Functionality .....	47
5.3.1	Login/Register .....	47
5.3.2	Friend Functionality .....	49
5.3.3	Direct Messaging.....	53
5.3.4	Rooms .....	55
5.3.5	Peer Connections .....	61
5.3.6	Chord Detection.....	67
5.4	Development environment.....	72
5.5	Project Management - Scrum Methodology .....	73
5.6	Sprint 1.....	74
5.6.1	Research & Literature Review.....	74
5.6.2	Paper Prototype .....	74
5.7	Sprint 2 .....	75
5.7.1	Goal .....	75
5.7.2	Wireframes .....	75
5.7.3	Hifi Prototype.....	76
5.7.4	Survey.....	76
5.8	Sprint 3.....	77

5.8.1	Goal .....	77
5.8.2	Basic Video Call Application .....	77
5.8.3	Hosted Machine Model.....	78
5.9	Sprint 4 .....	80
5.9.1	Web Audio API .....	80
5.9.2	Screen Sharing Audio .....	81
5.9.3	Dual Audio Signals.....	82
5.9.4	Microsoft Azure Custom Vision.....	83
5.9.5	Guitar Spectrogram Generation .....	83
5.10	Sprint 5 .....	85
5.10.1	Interim Presentation.....	85
5.10.2	Backend Server .....	85
5.10.3	Frontend UI .....	86
5.10.4	Application Restructure .....	88
5.11	Sprint 6 .....	89
5.11.1	Backend Server .....	89
5.11.2	React Redux .....	89
5.11.3	Dashboard UI .....	91
5.11.4	Socket.io.....	91
5.11.5	Realtime Chat.....	94
5.11.6	Video Chat.....	95
5.11.7	Adding Components from previous versions.....	96
5.12	Sprint 7 .....	97
5.12.1	Combining Chord Detection & Guitar to Spectrogram Components .....	97
5.12.2	Displaying Chord in Video Call .....	97
5.12.3	Amp Component UI .....	98
5.12.4	User Testing – Video Call Functionality.....	98
5.13	Sprint 8 .....	100
5.13.1	User Testing - Chord Detection.....	100
5.13.2	Emit Only to Users in the Room.....	100
5.13.3	Resize Room.....	101
5.13.4	Implementing User Feedback .....	102
6	Testing.....	103
6.1	Introduction .....	103
6.2	Functional Testing.....	103
6.2.1	Auth.....	104

6.2.2	Guitar Testing.....	105
6.2.3	Rooms .....	106
6.2.4	Chord Detection.....	107
6.2.5	Discussion of Functional Testing Results .....	108
6.3	User Testing .....	109
6.3.1	User Testing Results.....	110
6.4	Conclusion.....	112
7	Project Management .....	113
7.1	Project Phases.....	113
7.1.1	Proposal .....	113
7.1.2	Requirements.....	113
7.1.3	Design.....	114
7.1.4	Implementation .....	114
7.1.5	Testing.....	115
7.2	SCRUM Methodology.....	116
7.3	Project Management Tools.....	116
7.3.1	GitHub .....	116
7.3.2	Trello .....	116
7.4	Reflection .....	117
7.4.1	Your Views on the Project.....	117
7.4.2	Completing a Large Software Development Project.....	117
7.4.3	Working With a Supervisor .....	117
7.4.4	Technical Skills .....	118
7.4.5	Further Competencies and Skills.....	118
7.5	Conclusion.....	118
8	Conclusion.....	119
8.1	Summary of Chapters .....	119
8.2	Future improvements .....	120
8.3	Personal takeaway .....	121
9	Bibliography .....	122
10	Appendices.....	124
10.1	Appendix A – Requirement Gathering Survey .....	124
10.2	Appendix B – Figma: Lo-fi Prototypes.....	124
10.3	Appendix C – Figma: Hi-fi Prototypes .....	124
10.4	Appendix D – User Testing: Video Call Application.....	124
10.5	Appendix E – User Testing: Chord Recognition.....	124

10.6	Appendix F – Hosted Site .....	124
10.7	Appendix G – Hosted Server .....	124

## 1 Introduction

In today's digital era, technology has transformed various aspects of our lives, including the field of education and music. With the increasing demand for remote learning and online interactions, there is a growing need for innovative solutions that facilitate effective communication and collaboration among musicians, particularly in the context of music education. Traditional music lessons require physical presence, limiting accessibility for learners and instructors alike. However, advancements in peer-to-peer video call technologies present new opportunities to bridge this gap and create a platform that enables musicians to teach and reach a wider audience.

This project aims to design and develop an AI aided video guitar lesson application aimed at both new musicians and instructors. The application will be a platform for music instructors to teach students without the need for traveling or renting a dedicated space. Users will be able to connect two audio inputs, a microphone and guitar allowing the other user to hear both their voice and instrument in real time during a video. Additionally, the application will contain an AI component to further ease the difficulties that come with teaching in an online environment. A machine learning model that is capable of detecting guitar chords in real time will be implemented into the model, adding additional tools for teaching new musicians. The application will run entirely in the browser to allow easy access for all who wish to use it. It is recommended to gain all functionalities of the application, the user taking role as the instructor has access to an audio interface to allow a guitar signal to be transmitted directly to the browser, as the AI model used will be trained on direct audio signals. However, the application will still be accessible with a single audio input.

The report first goes into detail researching technologies and similar applications along with their strengths and weaknesses, as well as researching the technology needed to create the application. Once the research is complete, it discusses how the application is designed, both functionally and stylistically. It goes into detail about the different technologies needed to create the application and why they were chosen. The report discusses the implementation of these designs and goes into great detail of how each component was created, and how each component's functionality integrates with one another. It individually discusses the front-end client, the back-end server, and the configuration of the database. Finally, the report discusses the functional testing of the components, the errors that were discovered and how they were resolved. User testing will also be discussed, including feedback from the user's and descriptions into how the feedback was addressed.

## 2 Research

### 2.1 Introduction

Since the COVID-19 pandemic, video calls have become a much more common means of communication in both people's work and social lives. A report released by Cisco (VNI, 2021), stated that IP video traffic was 82% of all consumer internet traffic for 2021. This growth was caused by necessity, allowing employees to work from home while still being able to keep in touch with their peers, as well as family and friends being able to keep in touch with others still isolating from the virus. Many employers have allowed the new work from home positions to stay, long after the COVID-19 safety restrictions have eased. Although online video calls have become more standardized, are they as beneficial as face-to-face conversations in terms of communication and mental health? "Video chats mean we need to work harder to process non-verbal cues like facial expressions, the tone and pitch of the voice, and body language; paying more attention to these consumes a lot of energy" - (Jiang, 2020).

Another aspect of life greatly affected by the COVID-19 pandemic is live music. Due to social distancing and the closure of many venues, many musicians struggled financially as for many it would be their main source of income. According to the Irish Music Rights Organisation's Annual Report and Accounts 2020 (IMRO, 2020), revenue in the music industry was down 31% compared to the previous year. To aid in their financial struggles, many musicians would begin to offer one on one lessons on their preferred instrument through video calls. However, there is a lack of options offered to a user when using typical video calling software, which in turn makes teaching online far more complicated than a face-to-face lesson. According to a study into how effective it is to teach guitar online during the pandemic (Zahal & Ayyıldız, 2022), it was found that music instructors had many difficulties conducting lessons online beyond connection and internet issues.

The focus of this research chapter is to understand both the benefits and the common frustrations and issues with the most popular video calling programs and propose suggestions to create a new alternative program. This chapter will examine the technology required to create such an application and if it is possible to alleviate the current issues with popular video calling software. To begin, research must be conducted on a technology that can connect users over the internet. Once such technology is WebRTC.

## 2.2 WebRTC

WebRTC is a collection of JavaScript API's that allow users to establish a peer-to-peer connection between two browsers, allowing users to exchange audio and visual data in real time (García , Gallego, Gortázar, & Bertolino, 2018). It has grown in popularity due to the fact that many browsers support it. WebRTC is similar to WebSockets, a computer communication protocol that allows real time communication between two browsers. However, WebSockets is real time communication through a server as opposed to WebRTC's peer to peer. In WebSocket's case, when a message or audio/visual signal is being transmitted from user to user, the data will need to first be sent to the server, then to the recipient which will create latency. This latency would not be a problem for an application such as a text chat, but in the case of video calling, there must be as little latency as possible. Therefore, WebRTC is the better choice for developing the video call application.

WebRTC uses User Datagram Protocol (UDP), a communications protocol which is used to establish low latency and loss-tolerating connections between two peers. While UDP is fast, it is not a reliable protocol for transferring important data as it never validates if the data was received. This is suitable for video calls as losing a few frames from the video signal will not drastically impact the quality of the call. However, if UDP was used for sending files and the files loses some of its data, the file itself will become corrupt. WebRTC has no built in signaling, meaning WebRTC by itself cannot be used to make the initial connection between the two clients (García , Gallego, Gortázar, & Bertolino, 2018). This is why WebSockets are used to signal the two peers to establish the initial connection, which is then taken over by WebRTC.

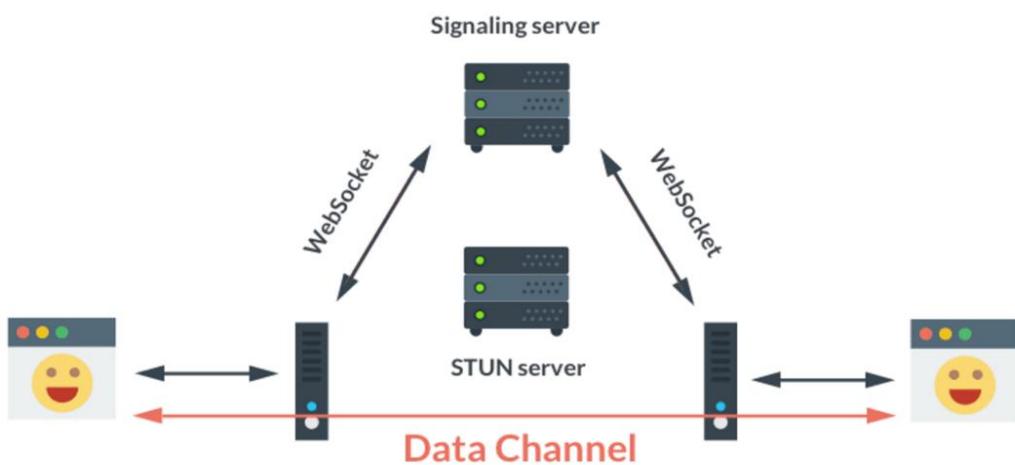


Figure 1: WebRTC Example (Rossouw, 2020)

As illustrated in Figure 1, the two main components being sent between the two peers are a Session Description Protocol (SDP) and some ICE Candidates (Manson, 2013). An SDP is an object containing information about the session connect, such as the codec, address, media type, etc. The peer initializing the connect will send one SDP in the form of an offer, then the receipt will return another SDP in an answer. An ICE Candidate is a public IP address and port that could potentially be used as an address to receive data (Manson, 2013). The connection between the two peers will start by the first peer sending their SDP to the second peer, which will be relayed through the signaling server using WebSocket. The second peer will receive the SDP and will return their own SDP back through the server to the first peer. After the two SDP's have been exchanged, the two peers can connect, but data cannot yet be exchanged. For data to be exchanged, the two peers need to share their public IP addresses. To do this, Interactive Connectivity Establishment (ICE) is used, which will make requests to a Session Traversal Utilities for NAT (STUN) server (Manson, 2013). STUN servers allow clients to find their public IP addresses. Once the peer receives the ICE Candidates from the STUN server, it will send these candidates to the other peer. Once both peers have received each other's ICE Candidates, they will be able to transfer data between one another.

WebRTC's peer to peer and UDP connections make it perfect for developing video call software. However, WebRTC covers the connecting the two users together and sending the data, it does not contain a user interface for the user to adjust settings and add additional inputs, nor does it offer any additional features that are missing from typical video calling software. To develop these features, there must be additional technology included in the application.

## 2.3 Note Detection

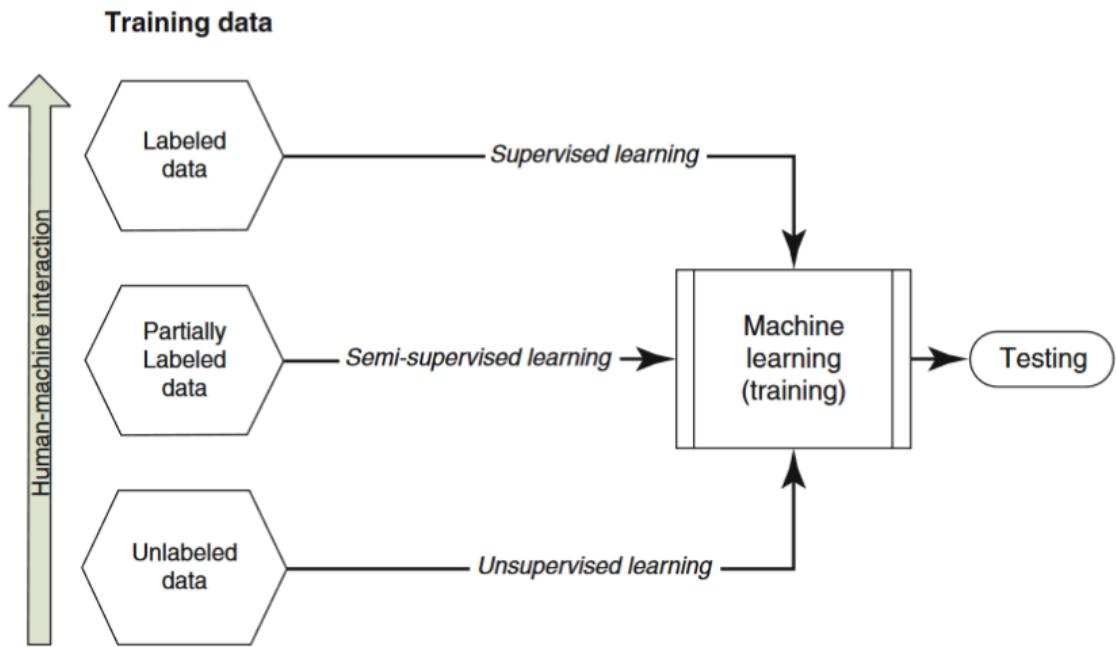
### 2.3.1 Purpose of Note detection

The main focus of the software being developed is to improve the teaching experience when teaching a musical instrument over video call. One of the main issues stated in the study by Zahal & Ayyıldız (2022), was that teaching beginners was much more difficult over video chat due to being unable to show correct positioning and technique. To rectify this issue, note detection will be implemented into the application.

The purpose of adding note detection to the application is to aid the student in learning their instrument. The teacher will play a chord, and the application will be able to recognize the note being played after first being trained. Once the application detects the note/chord being played, it will display to the student a chart displaying the chord being played and how to play it. This will help the student if their view of the teacher's hands is being obstructed in any way. This feature will be added to the application by using machine learning.

### 2.3.2 Machine Learning

Artificial Intelligence (AI) is a branch of computer science used to create intelligent computer programs that can learn and solve complex tasks that would typically involve human like intelligence. AI programs are capable of rationalizing and calculating actions that will help them achieve a specific goal. For this to be possible, these AI programs use machine learning. Machine learning gives computer programs the ability to learn and make decisions based on data (Murphy & Naqa, 2015). It uses algorithms to process large quantities of information to output a prediction. This is achieved by using statistics to find patterns in the data given to the program.



*Figure 2: Categories of Machine Learning (Murphy & Naqa, 2015)*

According to ‘What is Machine Learning?’ (Murphy & Naqa, 2015), there are three main types of machine learning, supervised, semi-supervised and unsupervised learning. In supervised learning, labelled data is given to the algorithm to help it determine the most accurate way to achieve its goal, be it image recognition or a recommender system. In the case of image recognition, this would mean that the labelled dataset would contain both an image and text telling the algorithm what is in the image. This is typically used to train a model or algorithm. If a model is given an unlabelled dataset, it will use unsupervised learning. Unsupervised learning involves the algorithm detecting and analysing patterns on its own. If it is given partially labelled data, it will use semi-supervised learning which is a mix of both supervised and unsupervised learning.

### 2.3.3 Neural Networks

A neural network is a type of machine learning in artificial intelligence, which uses a method to teach a computer how to process data similar to the human brain. It uses a series of interconnected nodes, like neurons in a brain, to take in multiple inputs and through a series of functions will find the best way to find the desired output (Chaturvedi, Titre , & Sondhiya, 2014).

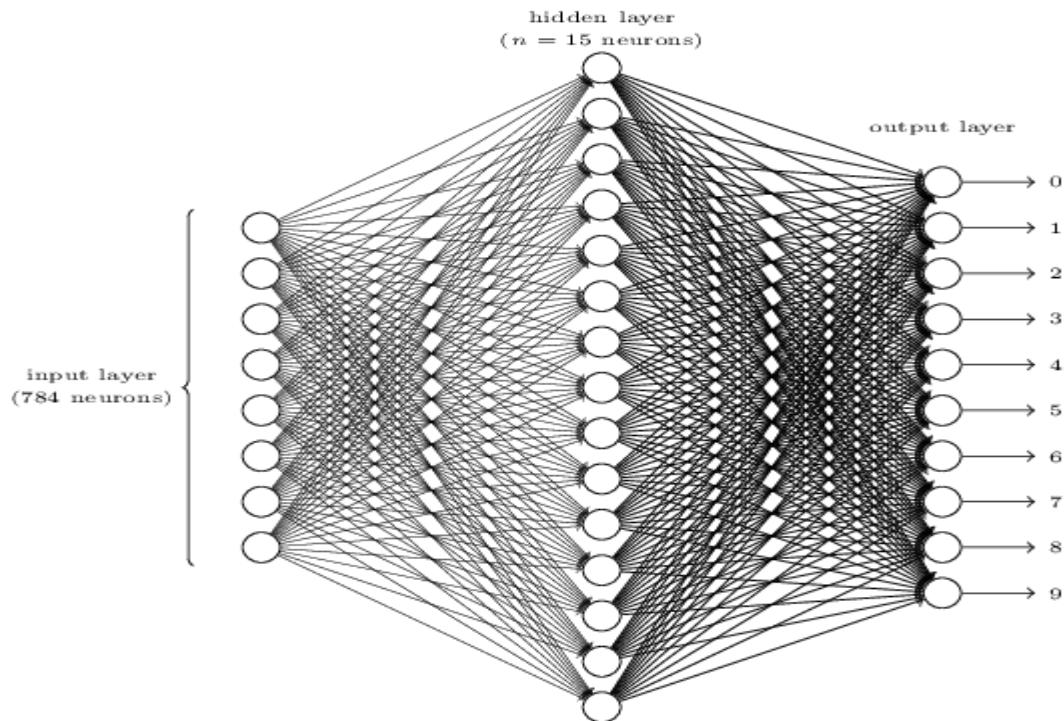
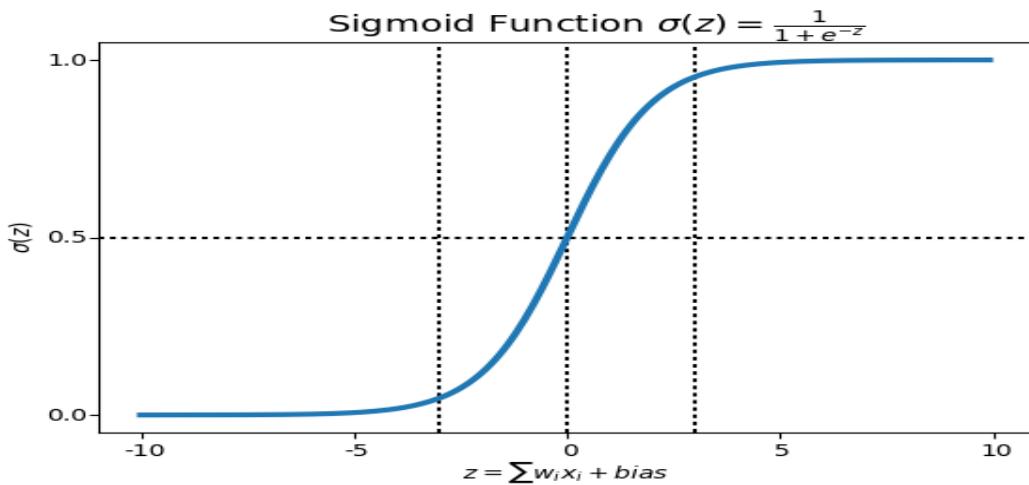


Figure 3: Neural Network Example (lee, 2018)

One of the basic examples of a neural network is one used to learn to recognise handwritten digits. A neural network has multiple layers, in this example in Figure 3, the first or input layer starts with 784 neurons. These neurons represent an input image that comprises of the length and width of pixels, so if there are 784 neurons, each image would contain  $28 \times 28$  pixels. Each neuron has a value between 0 and 1 which is called its activation. The activation represents a greyscale value for the corresponding pixel, so if a pixel's value is 0 it was be displayed as black, and a 1 will display as white. The last or output layer of the network has 10 neurons, each of which represents a number from 0 to 9. The activation in this layer represents how much the network thinks that is the number in the input image. For example, if the activation of the 7th neuron on the output layer is 0.8, the network has calculated that there is an 80 percent chance that is to detect edges and patterns that will help specify what a number look like (Krose & Smagt, 1996). For example, in the hidden layer there could be a neuron that detects if the base of the number has a straight vertical line, such as a 1 or 4. That would make the activation of these neuron close to 1, which will affect the activations of neurons in

the next layer. The lines connecting each of the layers are known as weights, which are values attached to inputs that convey the importance of that input in predicting the output by calculating the weighted sum (Krose & Smagt, 1996). the input number is a 7.

The layers in between the input and output layers are known as the hidden layers. While there is only one hidden layer in Figure 3, there is typically more than one hidden layer in a neural network. The activations in one layer will determine the activation in the next layer. The goal of hidden layers



*Figure 4: Sigmoid Function Graph (Amir, 2019)*

With the weighted sum, we use the sigmoid function to turn a real number line to a range between 0 and 1 (Jurafsky & Martin, 2021). With this function, we can easily see the activation of the neuron and measure how positive the relevant weighted sum is.

$$\frac{1}{1 + e^{-(w*x+b)}}$$

*Figure 5: Sigmoid Function (Jurafsky & Martin, 2021)*

If we only want the weighted sum to be active when above a certain number, we add a bias. The bias helps offset the result and shifts the activation to be more positive or negative (Jurafsky & Martin, 2021). This happens in each neuron in the hidden layers. Each neuron contains its own weight and bias, all of which can be changed to achieve different results. The job of the neural network is to

learn how to configure all of these different weights and biases so that it may achieve its task of properly recognising the numbers drawn.

For the network to start learning how to recognise the numbers, it is given training data. This data will contain handwritten digits as well as labels informing the network of what number the image is representing. The network will then use this data to adjust its weights and biases in each layer so that it will get better at detecting the different shapes and patterns that make up the numbers. After the network has been trained, it will be given test data to see how accurately it can detect numbers from hand drawn images that it has never seen before.

$$\frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

*Figure 6: Cost Function (S, 2021)*

When the network starts off with the testing data, its initial settings for the weights and biases are completely random (Günther & Fritsch, 2010). The network will then use a cost function to know the cost, it uses gradient descent (Bishop, 1993) to figure out how it should adjust its weights and biases in order to improve its efficiency and decrease the cost. It uses an algorithm called backpropagation to compute the gradient, then distributes it back to the hidden layers to adjust the weights to decide how accurately it predicted the numbers in the images. The cost is calculated by adding the squares of the differences between the output activation that the network predicted and the value that it is looking for. The sum of this number is small when the image is identified correctly and large when it is wrong. Then it retrieves the average cost of all the training examples to measure the efficiency of the network.

### 2.3.4 Machine Learning Audio

The reason for using image recognition as an example for machine learning and neural networks is that the method to detect audio is virtually the same bar one difference. Instead of using images to train the algorithm, spectrograms will be used.

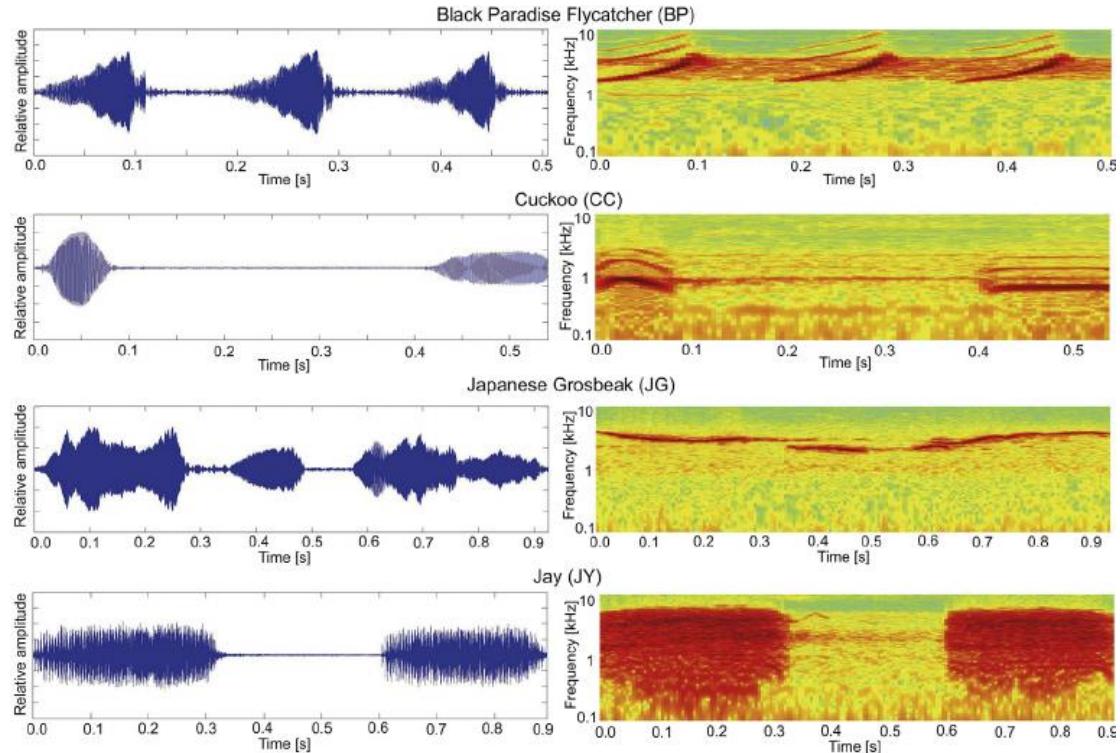


Figure 7: Waveforms & Spectrograms of Birdsongs (Soeta & Nakagawa, 2015)

A spectrogram, as seen on the right column in Figure 7, is a visual representation of all the frequencies of a waveform over time (Camastra & Vinciarelli, 2015). To a computer it is essentially a waveform represented as an image. These spectrograms are converted from waveforms of audio from different bird calls. We can apply computer vision techniques to the spectrograms, so a machine learning model can extract the dominant audio per timeframe by finding patterns in the spectrogram. From these patterns, the model can be trained to detect the different bird calls.

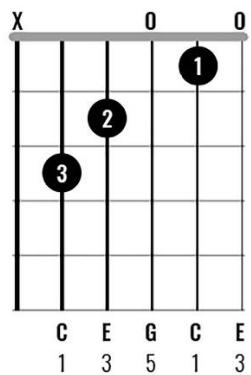


Figure 8: Guitar Diagram of 'C' Chord (Fogg, 2021)

This method of audio detection will be perfectly suited to the development of the note detection feature for the application. Instead of being trained on bird calls, the model will first be trained on basic guitar chords, each being played in various styles and with different effects and amplification methods to have a wide range of sample data. Once the model has been trained and is able to detect basic chords, it will be trained to detect single notes and more complicated chords. Once it has been trained, each note will be associated with an image file of the chord being played and will display this chord to the student.

## 2.4 React

Both WebRTC and machine learning models will be used for the core functionality of the application. However, the user will be unable to make use of this functionality without some form of user interface (UI). A UI is necessary for the application so the user may interact with the application and will allow them to initiate a video call, add their input devices or adjust their audio settings. Many of these UI elements could be created with plain HTML, CSS, and JavaScript, however, to make a cleaner and more efficient UI, it is common practice to use a JavaScript library.

### 2.4.1 What is React?

React is a JavaScript library used for building fast and interactive user interfaces for web and mobile applications. React was developed by Facebook and is currently the most popular JavaScript library for building user interfaces (Saks, 2019). It is a reusable component based front-end library, meaning each component of a page such as a navbar or search box, will be created separately and then imported to build a user interface. Every React application contains a root component, which represents the application itself. This root component will contain the child components that are used to build the application (Fedosejev, 2015).

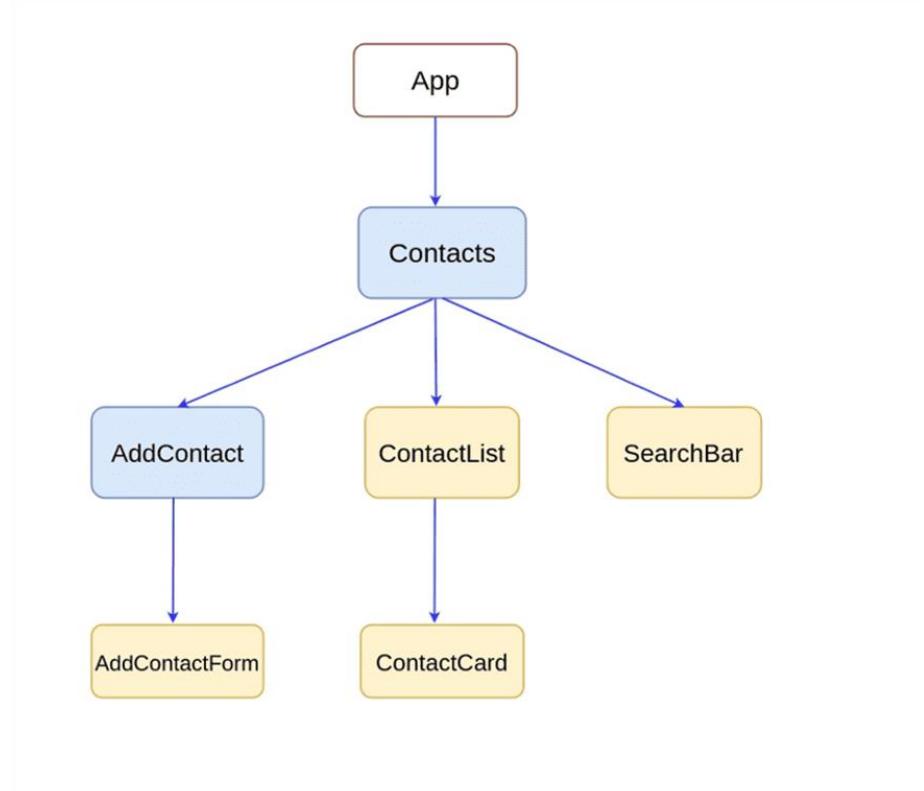


Figure 9: React App Structure (M, 2018)

A component is typically implemented as a JavaScript class that has a state and render method. The render method depicts what the component will look like when rendered on screen. The output of the render method is a React element, which is a JavaScript object that maps to a DOM (Document Object Model) element. The DOM represents the page so that programs can change the document structure, style, and content. It provides a way for programs to access and manipulate the content of a web page (Robie, n.d.). With the DOM, developers can use JavaScript to dynamically change the page's content and style, add or remove elements, and respond to user interactions. React keeps a lightweight representation of the DOM in memory known as the virtual DOM (Fedosejev, 2015). When the state of a component changes it will first change in the virtual DOM, as in virtual DOM is easier to create. When the virtual DOM changes, React will compare the virtual DOM to the real DOM to determine what component has changed and will only update that component so that it will be in sync with the real DOM. This means React does not have to work with the DOM API in browsers.

#### 2.4.2 Why React?

When determining what programming language should be used to develop the application, it was important that JavaScript be used, as the end users would be using the program through a browser. Rather than using vanilla JavaScript along HTML and CSS, it is far more efficient to use one of JavaScript's frameworks, Angular, React and Vue. From *JavaScript frameworks: Angular vs React vs Vue* (Saks, 2019), which studies these frameworks, it is possible to make an educated decision as to which framework should be used. One of the main benefits of React over the other two frameworks is the previously mentioned virtual DOM, which allows fast performance in both in development and deployment. This report also discusses how Angular is the most difficult to learn, due to it using TypeScript. Any additional difficulties should be avoided as the machine learning algorithms will add a great deal to the workload. Out of the three frameworks, React would be best suited to the application due to its performance and ease of use.

## 2.5 Conclusion

The information provided for this research has provided ample information for developing a new video calling application for guitar teaching with the integration of machine learning technologies. It is possible that the application may succumb to some of the typical drawbacks such as latency or poor connection because these are completely dependent on the user's hardware and connection state. However, the machine learning features will be a good feature to make these issues more bearable and will fix many of the issues stated in the study by Zahal & Ayyıldız (2022).

The application may also implement additional features in the future. It will be possible in the future to incorporate Virtual Studio Technology (VST) software, allowing users to alter their guitar sound to get the user more invested in the lesson. The application may also be able to improve on to allow more than two users connect via the application. This could allow virtual band practices, eliminating the cost of a rehearsal space and allowing musicians to play music with people from around the world.

### 3 Requirements

#### 3.1 Introduction

The application's purpose is to allow two users connect via a peer-to-peer connection which will allow them to exchange audio and video data. An AI model capable of detecting notes from an audio signal will also be implemented into the application. The application will be built by first creating two separate components, a video calling component and an AI machine building component. Both will be integrated together once they both are fully functional.

One of the components will be a video calling application. This will be a peer-to-peer video calling application, meaning no server will be required once the two users have established a connection between browsers. However, a server will still be required to establish this initial connection before data can be transferred between the two browsers. The video calling application will differ from similar applications by having the option to add multiple video and audio inputs. This will allow an instrument instructor to have several cameras at different angles as well as separate audio tracks for their voice and instrument. The parameters of these audio tracks such as gain and treble will also be adjustable.

The second component of the application will be a AI machine learning model that will be able to detect guitar chords. A sequential model will be trained on over a thousand files of different guitar chords. Once the model has been trained it will be tested to see if it can detect what note the chord is playing. If the model has a high accuracy of success, the model must then be configured to take in live data rather than audio files. To do this, it may be beneficial to convert the audio files to spectrograms before the audio goes through the model, as this will be less work for the model and may improve latency. Once the model is successfully taking in live audio and returning a note, it the model will be hosted online allowing it to be used by making a request to the model.

## 3.2 Requirements gathering

### 3.2.1 Similar applications

#### 3.2.1.1 Zoom



Figure 10: Zoom screenshot.

One of the most popular video calling applications is Zoom. Before the pandemic it was primarily used by businesses and large corporations due to being able to host very large online conferences. It grew in popularity during the pandemic as a user did not require an account to join a call, making it very accessible to the general public. It also has numerous features like screen sharing, virtual backgrounds, and chat functions that make it ideal for remote work or study. Its main benefit over other applications is its large audience hosting capabilities, free account service, Google Calendar support, Facebook integration and ease of use. However, it does also have many cons including too many subscriptions and add-ons, low quality picture (720p), and in the case of the guitar application proposed, no option for multiple audio and video inputs.

#### Advantages:

- Works on all platforms.
- Supports large audience.
- No account needed.
- Comprehensive feature set
- Easy to use.

#### Disadvantages:

- HD video is not standard.

- Too many subscriptions.
- Meetings can be unsecure due to lack of account needed.
- No option for multiple inputs

### 3.2.1.2 Teams



Figure 11: Microsoft Teams screenshot.

Teams is Microsoft's communications platform, allowing video calls as well as allowing access to Microsoft's other applications such as Word and Excel. A big benefit to Team's is allowing shared access to files so it may be used by multiple users. If a company already has a Microsoft 365 licence, Teams will be free to use for the company. Teams also has file storage for each channel, and easy to user search engine to find any file available to you. However, this storage could possibly be a security threat, as any guests may upload load malicious files.

### Advantages

- Free for Microsoft 365 users.
- Contains all of Microsoft's tools.
- Files search and storage.
- Multiple channels.

### Disadvantages

- Increased security risk
- Limited flexibility
- Insufficient notifications
- No option of multiple inputs

### 3.2.1.3 Discord

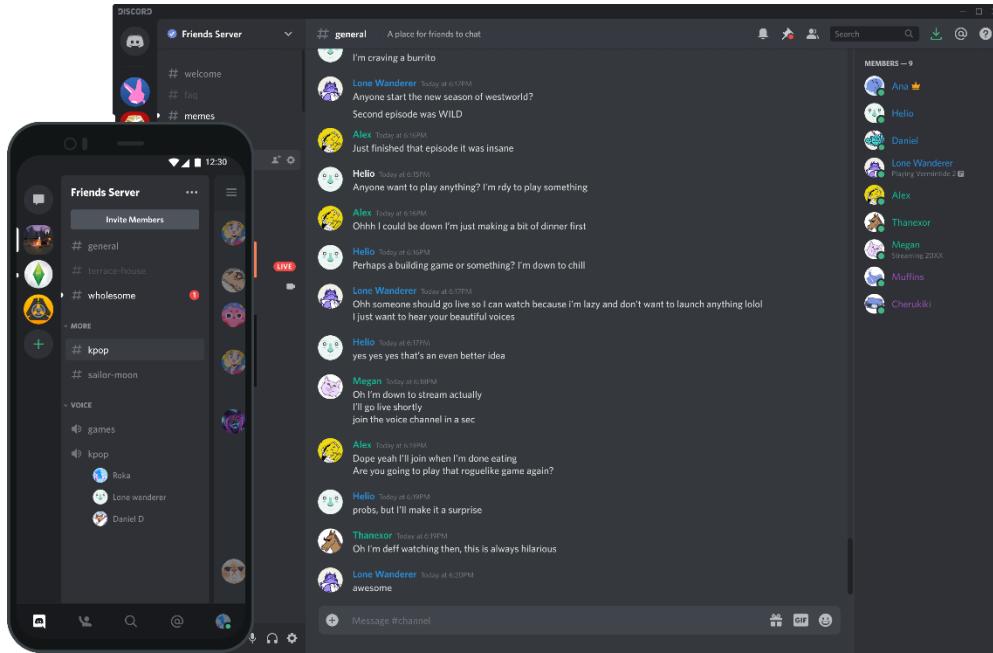


Figure 12: Discord

Discord is a communications application that offers voice, video and text chat with friends, communities and developers. Discord is most popular among video game enthusiasts and content creators, who will often use the platform to connect with their followers. Discord is free but also has several membership tiers.

#### Advantages:

- Voice, video, and text
- Can share screen to server participants.
- Can share files.
- Can create your own channels and servers.

#### Disadvantages:

- Video stream is capped at 720p at basic account.
- File size is limited to 8MB if at basic account.
- Server chat rooms are limited to 10 people if the server is free.
- No option of multiple inputs

### 3.2.2 Survey

A survey (see Appendix A) was conducted to gain more information on the application. This survey was shared among friends who play instruments, guitar instructors as well as within guitar communities on various forums and sites such as Reddit, Boards and Discord. Real world user feedback was needed on certain information such as what features would be expected of the application, as well as being able to determine what common shortcomings of other video chat are and how to combat them. The survey contained questions regarding instrument focused questions as well as questions regarding the user's experience with common video chat applications.

Is there a reason why you would not take lessons online?

Responses

ID ↑	Name	Responses
	anonymous	The teacher has to see where you are going wrong
!	anonymous	Tried it once didn't like it

Figure 13: Survey Responses

From the total survey results, three quarters of the respondents played musical instruments, which lead to the most surprising result of the survey, in that half of them stated that they had no interest in taking music lessons online. However, from other answers in the survey it was deduced that this was due to the fact that many of these respondents had tried music lessons already using video chat applications that did not have the features needed to properly conduct lessons.

Other information gained from the survey were the common issues most users found with video chat applications in general. Nearly all respondents said that they were unable to adjust the volume of individual participants in a call, and audio and video latency were the top issues for all respondents. It is important to take this information into account during the development of the application. WebRTC will help combat the issue with latency, as it is peer-to-peer and will continue with the call should there be a drop in audio or video.

## 3.3 Requirements modelling

### 3.3.1 Personas

The proposed application will focus on catering for two specific user profiles as its target audience. Both user's will be guitar players, however, they will be the opposite in terms of skill levels.

- The first user targeted will be beginner guitar players. These users may have busy lives and not be able to have the time to attend in person guitar lessons nor the time to investigate the necessary information to self-teach. It may also be possible that these beginner players simply prefer to be taught by a professional who will give them guidance and not overwhelm them with unnecessary information. Due to the interaction between teacher and student being online, it would be possible for the student to choose a teacher in a different country, so their chosen time for class may be more suited to a teacher in a different time zone.

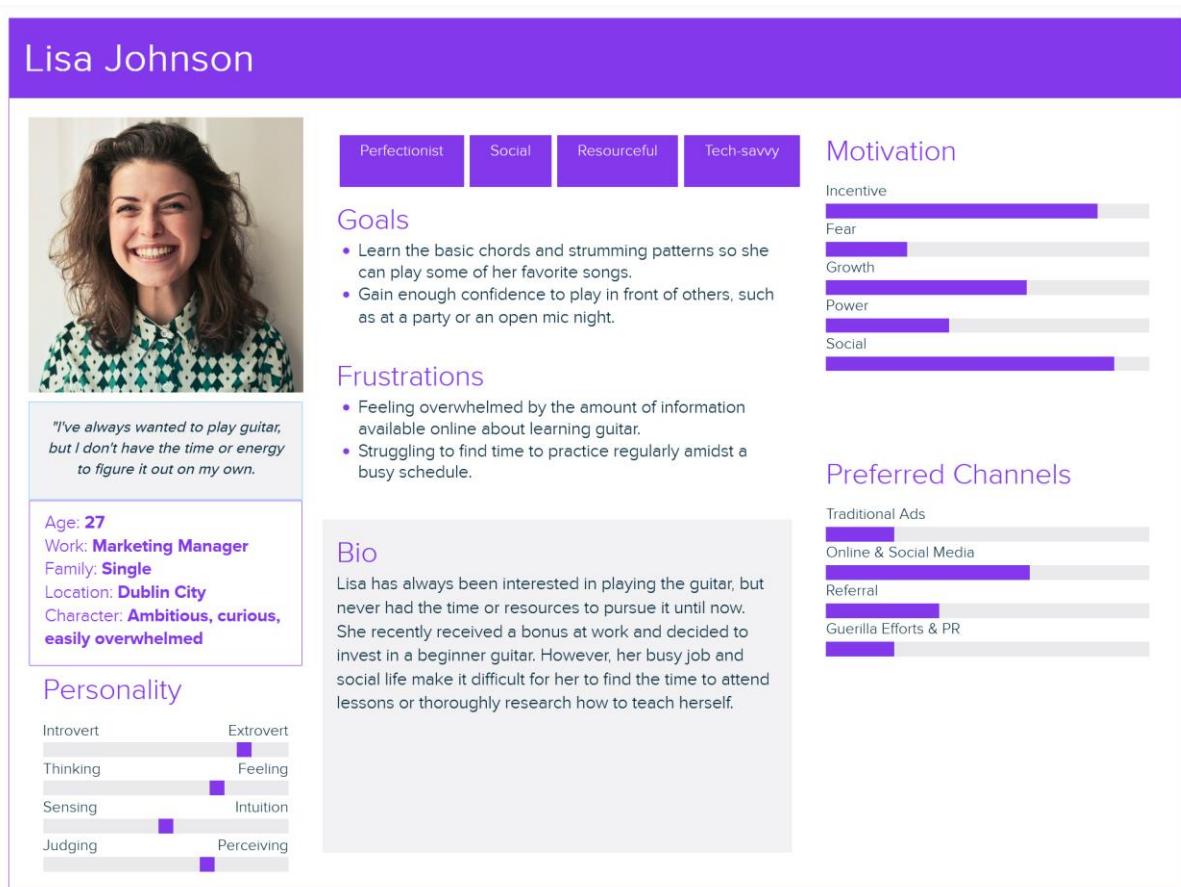


Figure 14: Student User Persona

The user persona in figure 14 describes an ideal user for the student role in the application. A person who is very interested in guitar and is tech-savvy enough to operate a computer any additional hardware required to use this application. They feel overwhelmed due to the vast amount of information online and would like some guidance. Their busy schedule stops them from conducting the research needed or making the time to travel to in person lessons.

- The second type of user targeted for the application are professional musicians that are qualified to teach their instrument. While the restrictions from the pandemic have eased, there are still many musicians struggling financially as live music has not fully recovered. Inflation has made touring much more expensive and increased ticket prices to combat these rising costs will see a drop in attendance for less known musicians. The proposed application would allow musicians to teach their instrument to guitarists around the world. The benefits of using the application over in person lessons are numerous and include not having to own a dedicated space or classroom for the lessons to take place, being able to attend lessons from their own home which eliminates extra travel costs, and being able to teach anyone with an internet connection will allow them to work within any time zone, so they may decide their own hours.

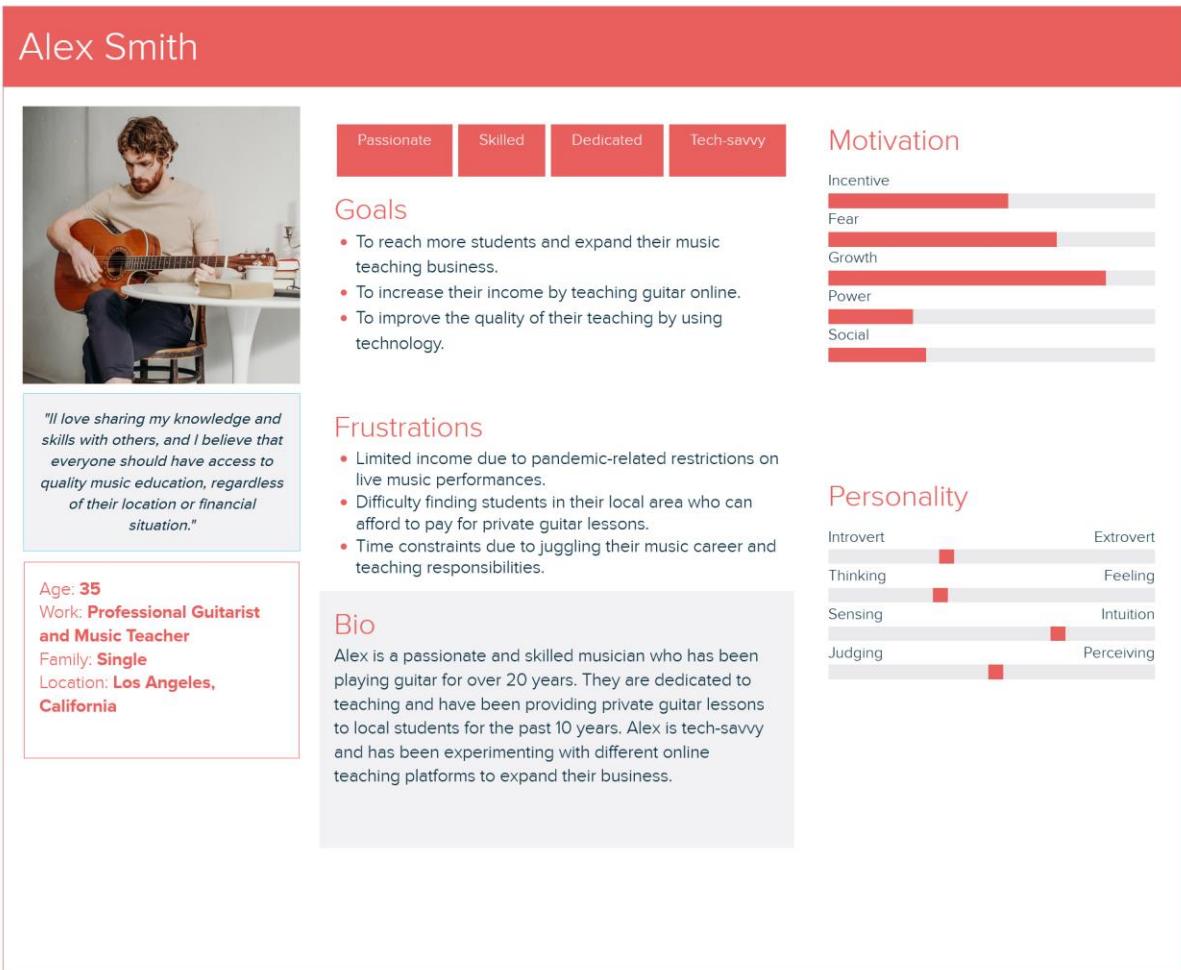


Figure 15: Instructor Persona

The user persona in figure 15 describes an ideal user for the instructor role in the application. A professional musician who is tech-savvy and looking to expand their business. Being tech-savvy, they would be interested in trying an online platform and have no trouble operating the application. As

they are struggling to find students locally, the application would be a perfect platform to expand their lessons to greater distances. The AI components of the application will also be able to improve the overall quality of their lessons.

### 3.3.2 Functional requirements

Functional requirements are the features and capabilities that an application must have to meet the needs of its users. They describe what the application should do, how it should behave, and what functions it should provide. The following functional requirements are what should be integrated into the proposed application:

1. **Login/Register:** a user needs to be able to register either as a student or instructor.
2. **Be able to establish a peer-to-peer connection:** A guitar instructor must be able to set up a time for a lesson, from which the student can join and connect to the instructor.
3. **Be able to transfer audio and visual data between two users:** Once a connection has been established, the user's must be able to see and hear each other.
4. **Use AI to detect a chord from the instructor's guitar signal:** When the instructor plays a chord on their guitar, a diagram of that chord must be displayed to the student to aid in their learning.

### 3.3.3 Non-functional requirements

The application should:

- Be Compatible with all web browsers.
- Ask the user's permission for use of their camera and microphone.
- Allow users to adjust all volume.
- Allow each user to connect multiple input sources.

### 3.3.4 Use Case Diagrams

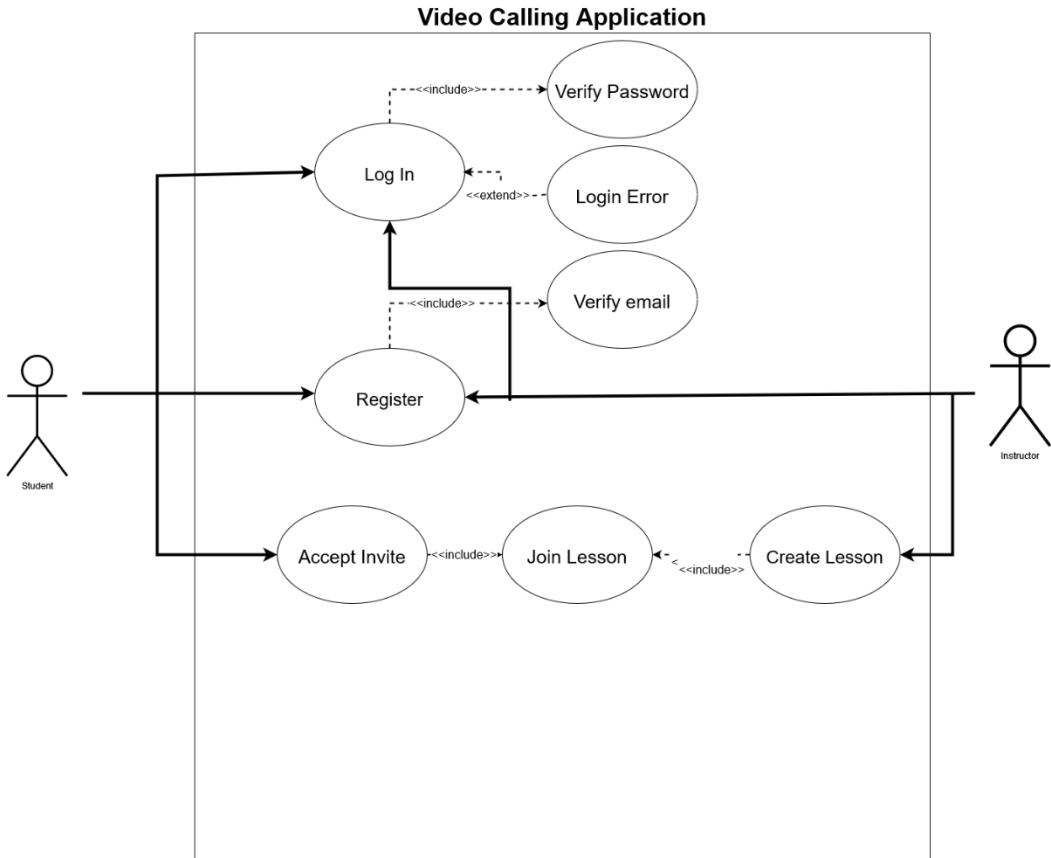


Figure 16: Use Case Diagram

There are two actors in the Use Case Diagram, the student and the music instructor. For both actors to be able to use the application, they must first register for an account. The register use case will include the verify email use case, meaning that for a user to register to the application, they will also have to verify their email address. Once a user has an account, they will use the log in use case, which includes the verify password use case, which ensures that the correct user is accessing their own account. If the password is entered incorrectly, the login error use case will be triggered, which is an extend use case if the log in use case, meaning it will only trigger if the verify password function fails. Once an instructor is logged in, they will be able to be able to create a lesson, which will then allow them to join said lesson. When a lesson is created, an invite is sent to the student who will be able to accept the invite and join the lesson.

### 3.4 Feasibility

The application will have two main components, the video calling component and the AI model that will detect the notes of the chords being played.

#### **Video Call Component:**

The UI (user interface) for the video call application will be developed with React, a JavaScript front end library. React is very popular for creating UI, so will be the perfect choice for developing the application due to its component-based architecture and virtual DOM. The back-end video calling functions will be handled by WebRTC. However, WebRTC cannot establish a connection between two clients on its own. To rectify this, Socket.IO will be used to establish the link between the two clients, and then WebRTC will take over. Socket.IO needs a server to handle the initial connection, so it will need to be hosted. Originally, Google's Firebase was planned to be used, as it offers free servers for small scale projects, and would be perfect for the initial testing phases, with the possibility for the application to upgrade the server in the future. However, during the original implementation, the Firebase server was too unreliable, and the connection was easily affected, causing the call to disconnect. For this reason, it was decided to develop a server, using node and express.

#### **AI Machine Learning Model:**

An AI model will be used as a way for the application to detect what chord is being played and display that chord to the user. First, a model must be trained on how to detect guitar chords. This will be done by having the model examine a large amount number of files containing recordings of different labelled chords being played. Once the model is trained it needs to be tested. If the model works, it will need to be altered so that it accepts live signal from a microphone. Once the model can detect a live signal, it will need to be hosted. Python programs can be hosted using AWS (Amazon Web Services). There are free tiers available, however it may be required to purchase a virtual computer above the free tier depending on the speed at which it is capable of processing requests. When the model is hosted, it will take an input request from the video call component and return a response that will contain a note which was detected within the signal. There will be a request made at least every 5 seconds, so the hosted model needs to be capable of receiving and returning requests at a high speed with as little latency as possible.

### 3.5 Conclusion

In conclusion, the proposed application aims to integrate a video calling component and an AI machine learning model for detecting guitar chords. The video calling component will allow multiple video and audio inputs and will be a peer-to-peer application, while the AI model will be trained on over a thousand files of different guitar chords to detect the note being played. The gathered requirements included an analysis of similar applications such as Zoom, Teams, and Discord, highlighting their advantages and disadvantages. Additionally, a survey was conducted to obtain real-world user feedback, and it revealed that half of the respondents who play musical instruments had no interest in taking music lessons online. Overall, the proposed application has the potential to provide an innovative and useful tool for guitar instructors and musicians.

## 4 Design

### 4.1 Introduction

As outlined in the previous chapters, the design of the application will involve two different components, a video calling component, and an AI model. The application is aimed at professional musicians looking for a platform to be able to teach their instrument, as well as beginner musicians looking to improve their skills.

The main component of the application will be the video calling component. A student and instructor will be connected to each other through a peer-to-peer connection via a browser and be able to transfer audio and video data so that they will be able to see and hear each other. One of main differences between this application and other video calling software is that both users will be able to use multiple video and audio input sources, allowing them to have separate audio channels for their voice and guitar, as well as allowing the instructor to have multiple cameras on their instrument to give the student a better view of the technique being showed by the instructor. Both instructor and student will be able to change the level of all audio sources, allowing them to configure audio levels to their preferred state.

The second component of the application will be the AI model which will be used to detect guitar chords. A machine learning model will be trained to listen to guitar chords and be able to detect the note that each chord is being played. Once a model has been trained, it will be hosted so that the video calling application may send requests to the model. The model will receive an audio signal from the instructor's guitar audio input, detect what note of is being played and then return that note to the application. The application will then display this note to the student.

## 4.2 Program Design

### 4.2.1 Technologies

**React:** React is a JavaScript framework primarily used to create user interfaces. After research was conducted into the various frameworks including Angular and Vue, it was clear that React would be best suited to this project, due its use of the virtual DOM and fast performance.

**Redux:** Redux is an open-source JavaScript library for managing application state. It provides a centralized store for all application state and provides a set of rules for how that state can be updated. These rules help ensure that the state is always predictable and that changes are handled consistently throughout the application.

**WebRTC:** WebRTC is a collection of API's that can be used to establish a peer-to-peer connection between two users, so that may transfer audio and video data between each other without the need of a server.

**Web Audio API:** Web Audio API is a JavaScript API that provides a set of functions and interfaces to allow developers to create and manipulate audio and music in real-time in a web browser (Web Audio API - Web APIs: MDN, 2023). The API supports a variety of audio formats and provides tools for audio processing and includes built in audio effects such as reverb and distortion and is supported by all major web browsers.

**STUN Sever:** A STUN (Session Traversal Utilities for NAT) server is a network protocol that enables devices on a private network to discover and communicate with other devices located on the public internet (Dutton, 2013). NAT (Network Address Translation) devices are used in networks to enable multiple devices to share a single IP address. However, because the NAT device assigns a private IP address to each device on the local network, the devices may have difficulty communicating with other devices on the internet. STUN servers will help these devices determine their public IP address to help them establish a direct connection with another device over the internet.

**Socket.IO:** While WebRTC can transfer data between to user without a server, it is unable to make the initial connection by itself. For that reason, Socket.IO will be used to establish the initial connection before WebRTC takes over. Socket.IO allows the frontend of the application to communicate with the backend server, allowing the frontend to get access to the user details from the server.

**Machine Learning:** A machine learning model will be used to detect chords played by the instructor which will then displayed to the user. Originally, a machine learning model was going to be written in python, using a sequential neural network to train and test the model. Once the model has been

trained, it would be hosted via AWS. It would receive a request from the application containing spectrograms of an audio signal and will return to the application a response contain what note it thinks it's being played. However, due to complications, it was decided to use Microsoft's Custom vision. Microsoft's Custom Vision is a cloud-based service that allows users to easily train and deploy custom image recognition models (Gomez, 2023). Users can upload their own labelled images and use them to train a custom image recognition model, which can then be integrated into their own applications.

**MongoDB:** MongoDB is a cross-platform, document-oriented NoSQL database system that allows for flexible and scalable data storage. Data is stored in flexible, dynamic documents that can vary in structure from one to another. This allows for a more natural representation of data and makes it easier to store and query complex data structures.

#### 4.2.2 Structure of React

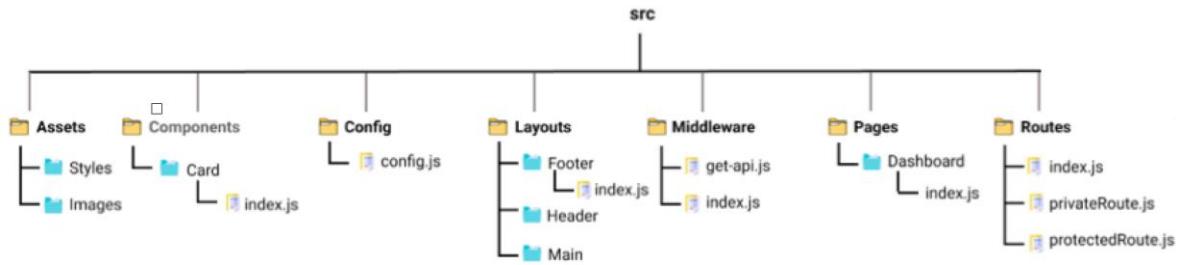


Figure 17: React File Structure

When a React application is first created, the folders in figure 14 are what a file structure for which the application composes of.

**Assets:** This folder contains the assets of the folder. Within the assets folder, there are sub folders to organise the various assets needed by the project if a project contains any images that need to be stored locally, they will typically be stored in the Images folder. If a developer is use any global styles, they will store them in the Styles folder.

**Components:** Components such as a buttons, cards and inputs are the building blocks of a React application. These components are typically created as a standalone component, that can then be used multiple times throughout the application. Once a component is created, it will be stored in this folder to be imported to any pages that requires it.

**Config:** This is where the configuration file is stored. The configuration file typically holds static variables that may need to be used throughout the application, variables such as an API key or server URL.

**Layouts:** The layouts folder stores the reusable layouts that are typically used on every page of the application, such as the header or footer.

**Middleware:** This folder contains files for any middleware used in the application. For example, it might contain a file containing various functions that can be used to make requests to APIs, which will then return some information to the application.

**Routes:** This is where the navigation routes are stored within the application. It will often store the various routes from which a user can travel to within the application while also storing the configuration as to what page a user can't visit depending on their role on the application.

In addition to the typical structure of React, the application will be using Redux. As previously discussed, Redux is a JavaScript library for managing the state of an application. It provides a predictable way to manage state in React applications by centralizing it in a Redux store and providing a set of tools for connecting components to that store and updating it with actions and reducers.

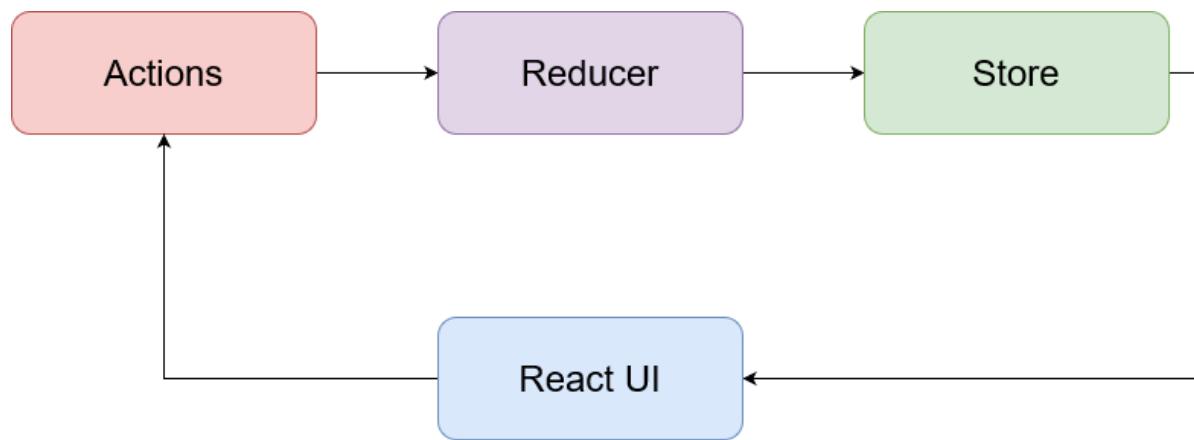


Figure 18: Redux Architecture

**Actions:** actions in React redux are used to trigger updates to the state in the Redux store. They are JavaScript objects that describe a change that has occurred in the applications. An action will typically have two properties. A 'type', which is a string that describes the action that has occurred. It is often represented as a constant value that is defined in a separate file to ensure consistency across the application. And a 'payload', which provides any additional data that needs to be passed along with the action.

**Reducer:** the reducer is a function responsible for handling actions and updating the state in the Redux store. The reducer will take the current state and an action object as inputs, and return a new state based on the action dispatched. The reducer will not modify the original state object but instead return a new state object that represents the new update state, to ensure that the state can be easily managed.

**Store:** the store in React redux is what is used the store and manage the state of the entire application. Instead of having to pass state down child components using props, all components can access the state of the store, making the development of applications with a large amount of components much easier.

#### 4.2.3 Design Patterns

A design pattern is a repeatable solution to common issues in web development. When discussing design patterns in React, they are generally used to solve issues in relation to rendering and passing props with ease. The following design patterns will be used to create the proposed application:

**Components:** Components are used as the core building blocks in React. They are reusable pieces of code that once created can be reused throughout the application. There are two types of components, functional components, and class components. A functional component is just a standard JavaScript function, which are capable of taking in parameters and returning an output depending on the context in which the function is used. Class components are JavaScript Object Oriented classes with functions that can be used to render components. The main advantage of class components were that they contain lifecycle methods that allowed the application to track state changes and update the global state. However, since the introduction of React Hooks, functional components may also track state changes making them near equivalent to class components. This is the reason that functional components are preferred when using React, and it is unlikely that class components will be used in the application.

**Hooks:** React hooks are a feature that allows functional components to use state management and lifecycle methods. They simplify code to make it more readable, as well as improve performance and reduce bugs caused by misuse of lifecycle methods. The two main hooks that will be used in the application are the useState and useEffect hooks. useState provides a way to declare and update state variables in functional components. UseEffect allows components to perform tasks such as fetching data or updating a text variable, after the component has already been rendered.

**Conditional Rendering:** Conditional rendering is a way of displaying different content within a react component based on certain conditions. Certain use cases may be displaying a loading icons while

data is being fetched from an API, or changing the content displayed to the user based on whether or not the user is logged in. It allows to create dynamic and responsive user interfaces by controlling what the user sees based on a wide range of conditions.

**Props:** Props are a way to pass data and behaviour from a parent component to a child component in a modular and flexible way. They are passed as read-only arguments to the child component that can be any value or type and are used to configure the component once created. Props can also be used to pass functions to allow for communication between the parent and child components.

#### 4.2.4 Application architecture

The applications will be created using the MERN stack. In web development, a web stack is a collection of software used to build a web application. A web stack typically consists of a frontend/client-side programming language, a backend/ server, and a database. The MERN stack consists of four technologies, MongoDB, ExpressJS, React, and NodeJs (Duggal, 2022).

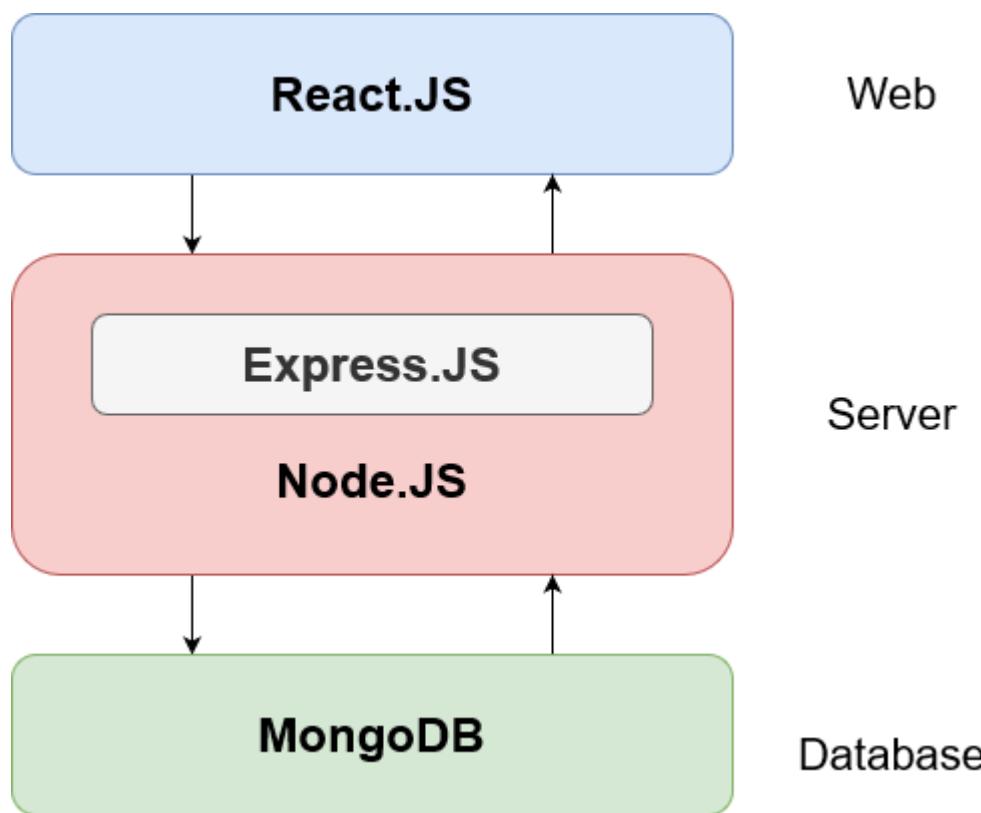


Figure 19: MERN Stack

**MongoDB:** will be used to store user profiles, room details and conversation history.

**ExpressJS:** is a server-side framework that will be used to create the backend server on top of NodeJS.

**ReactJS:** will be used to create the client UI.

**NodeJS:** is the runtime environment that will allow JavaScript code to run on the server.

Using MERN stack will allow the developer to use a single programming language, JavaScript, for both the front-end client and the back-end server. This can lead to a faster and more efficient development environment.

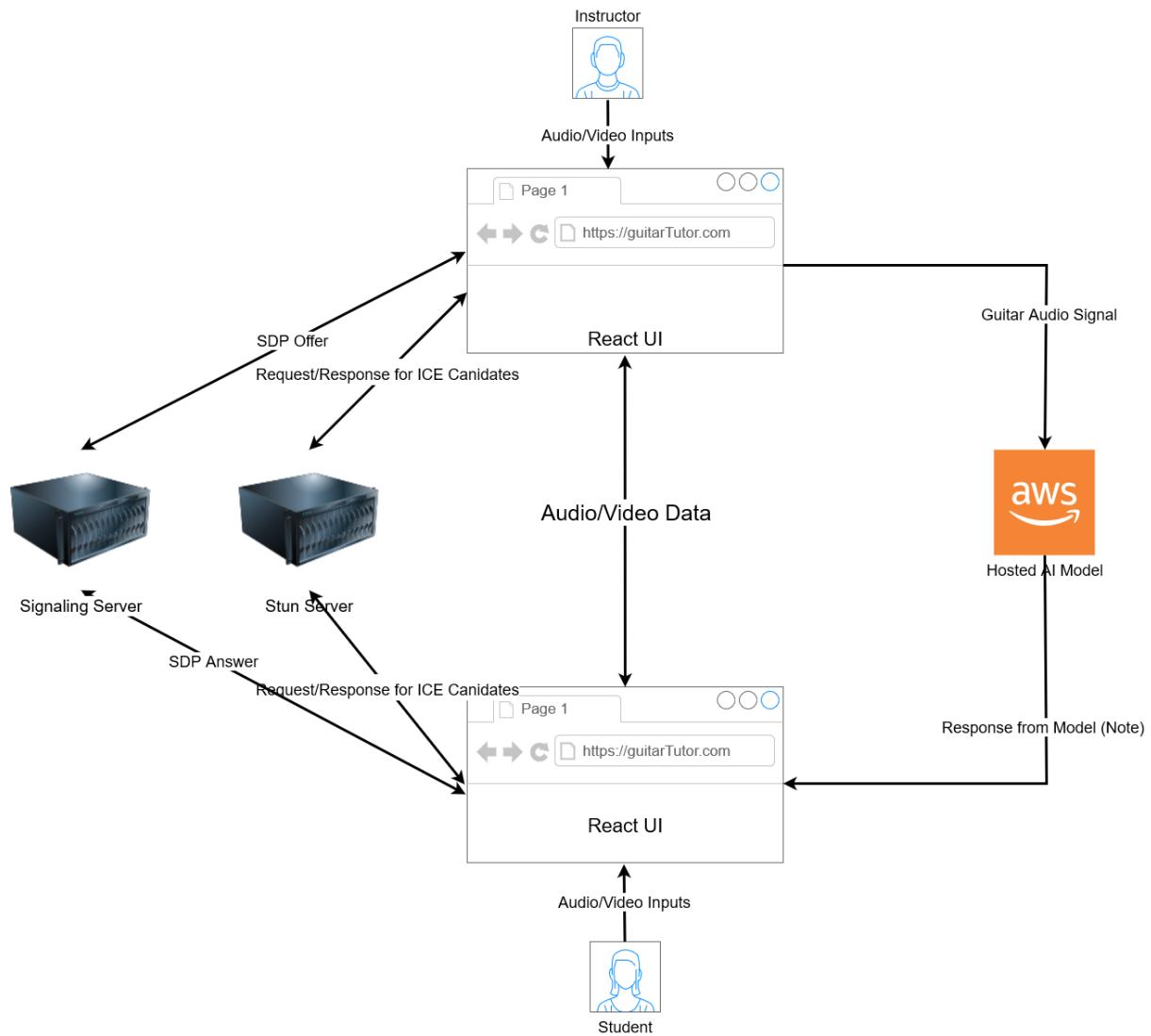


Figure 20: Original Application Architecture

The diagram in figure 20 gives an overview of the application's original architecture and details how each of the components would interact with each other. Both users would send their audio and video signals to the browser/React User Interface. From this interface, the instructor will be able to create an invite for the video call, which will send an SDP Offer to the server. The student will then accept the invite to join the call, which will create the SDP Answer to be sent to the signalling server. Both users will then make a request to a STUN server to receive ICE candidates, which will be used to

establish the peer-to-peer connection. Once the connection has been established, audio and video data will be exchanged between the two users. The diagram also shows how the instructors guitar signal will be sent from the react component to the hosted model, which will return a response to the student's react component.

The application architecture was redesigned during the implementation of the application due to complications with architecture and deployment of the hosted AI model. In the original implementation of the architecture, if an error occurred after the peer-to-peer connection, both users would be disconnected, and the application would close. If the users wanted to start the call again, the instructor would once again have to send another request for the student to join the call. There was also trouble with the AI model. The AI model was originally written in python and was planned to be hosted by Amazon Web Services. However, due to many issues with hosting the model, it was decided to take a different approach. To alleviate these problems, the architecture was redesigned.

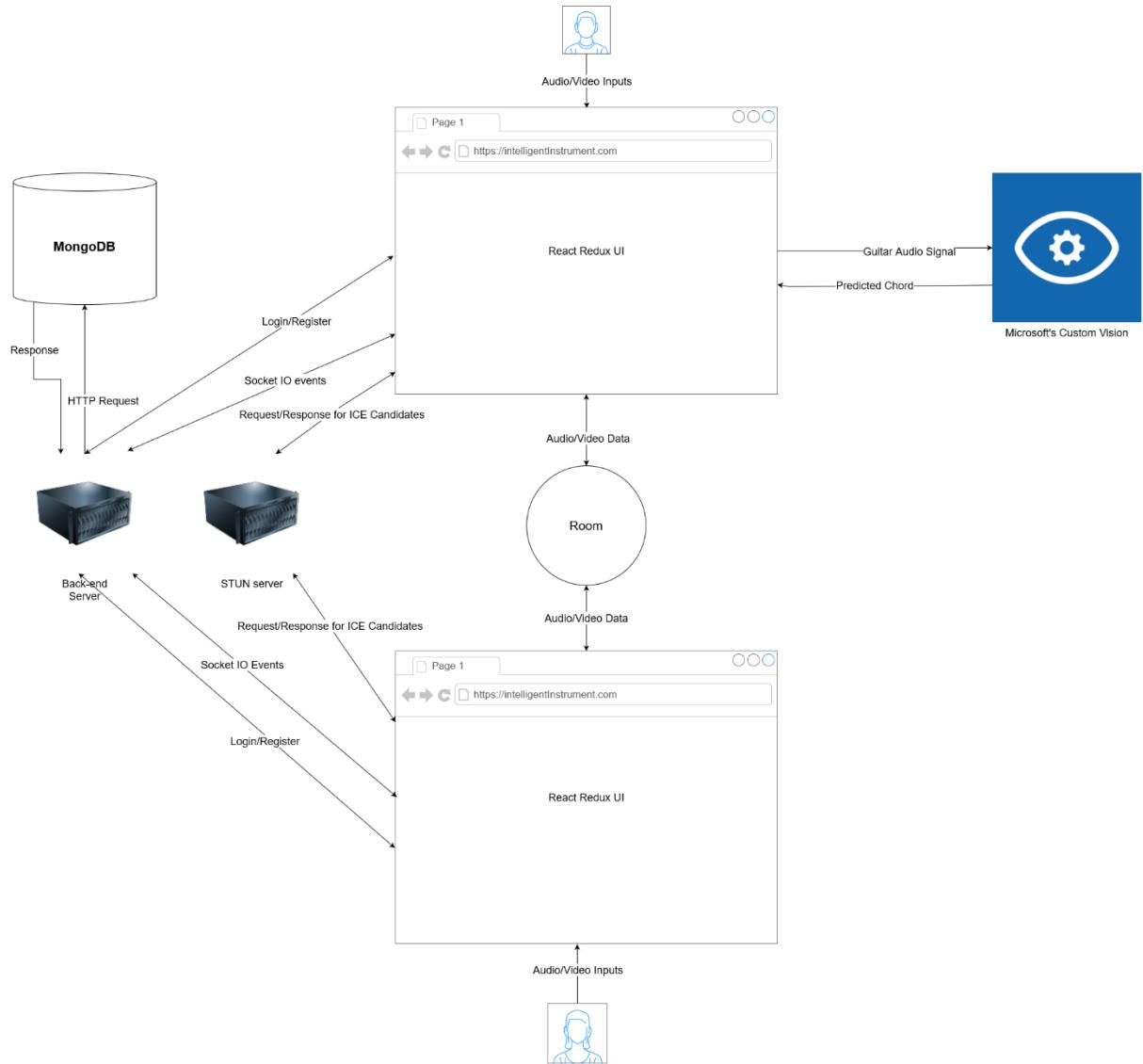


Figure 21: Application Architecture Redesign.

The diagram in figure 21 gives an overview of the redesigned architecture for the application. As before, both users will interact with a React UI, although now the application is using React Redux, making global state management much more efficient. Before using the application, the users will have to log in. If they do not have an account, they will first have to register. When the user logs in/creates an account, an API request will be sent to the server, containing the user's credentials. The server will then send a HTTP request to the MongoDB database to validate the user's credentials. The database will return a response, if the credentials match what is stored in the database, it will return a token to the client. If the credentials do not match the server will return an error. Once the users can log in, they will be able to add another user as friend. This is done through socket IO events and by storing pending friend requests in the database.

Once two users are friends, when one user creates a room, the other will be able to see the available room. Being able to create room removes the need for a user to send an invite directly to a user.

Once a user joins a room which another user is already in, the process of establishing a peer-to-peer connection will initiate. The user's will first get their ICE candidates from the STUN server, and then will exchange details through socket IO, which will allow them to connect and initiate exchange of video and audio data. The main benefit over this new architecture is a user is disconnected for any reason, instead of having to create a new invite, they can simply re-join the room.

The new architecture has also redesigned the way the audio classification works. The model now being used is a hosted Microsoft Custom vision model. Microsoft provides an easy to train model where users can simply upload their training photos, then are provided with a URL once training has completed. When the guitar signal from the instructor's guitar will now be sent to the model through the URL provided, and the model will return the predicted chord. This predicted chord will then be emitted through socket IO to all the user's who are in the same room as the instructor, which will then be displayed on the user's screen.

#### 4.2.5 Database Design

The database used in the application will be MongoDB, as it is part of the MERN stack, making it easy to implement with the React UI and Express/Node server. There will be a total of four tables in the database: users, messages, friend invitations and conversations.

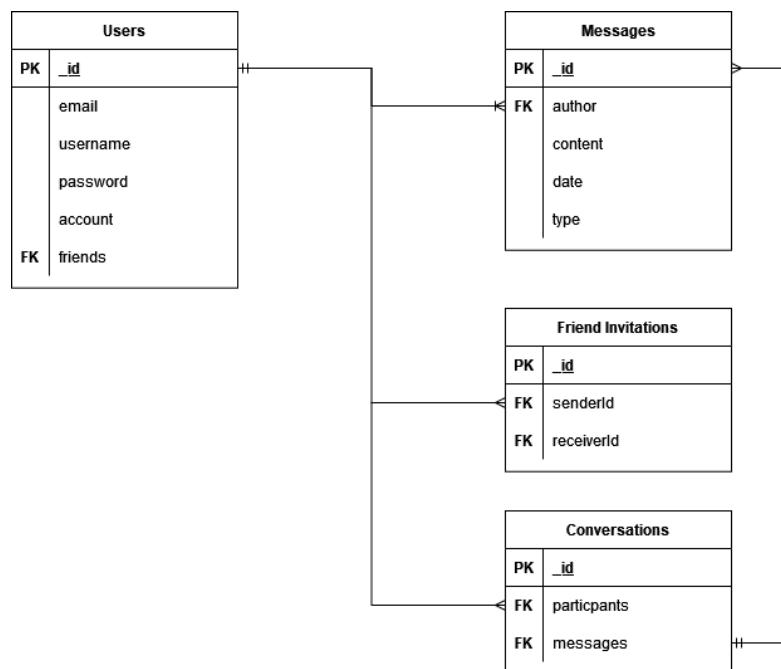


Figure 22: Database ERD

The ERD in figure 22 details the tables in the database as their relationship with each other. Each user will have an email, username, password which will be encrypted so the developer will not have access, their account type which will be either a student or instructor, and a friend's array. When the user has friends, they will be able to message that friend.

The messages in the database will contain the author, the contents of the message which will be encrypted so developers will not be able to see user messages, the date and type. The type refers to the message being either direct or group message. The implementation of the group messages will not be fundamental to the functionality of the application but is added in case the feature will be added in future iterations of the application.

The friend invitations table will contain two Ids, the Id of the user who sent the invitation and the Id of the user who will receive the request. Once the request is either accepted or denied, the object will be removed from the database. The final table is the conversations table. This will contain the participants, which will be an array of user Ids to represent the users taking place in the conversation, and the messages, which will be an array of Ids referencing messages from the messages table.

## 4.2.6 Process design

### 4.2.6.1 Pseudocode

Pseudocode is a simple, high-level description of a computer program or algorithm. It is a way of expressing the basic steps of a program using plain word statements that are easy to follow and understand (Ubah, 2021). It is written early on in development to plan the important functionalities and requirements of code. It allows developers to explain the process of a program to someone who might not be familiar with programming. The following is the pseudocode for the application for this report:

1. User will first create an account; they will choose between an instructor or student account.
2. The user's login details will be stored on a database, they will now be able to log in.
3. If the user is an instructor, they will now be able to create a lesson which is a signalling offer, this will create a session description protocol (SDP) object containing relevant information needed to establish a connection to the instructor.
4. The instructors SDP object will be saved in the signalling server.
5. A student will then answer the offer by creating their own SDP object.
6. Both instructor and student will then send requests to a STUN server, to receive their ICE candidates.
7. The server will then use both user's ICE candidates to create a peer-to-peer connection with WebRTC.
8. Once the connection has been established, the exchange of video and audio signals will begin.
9. The instructor's guitar audio signal will be split, one signal will be sent direct to student, while the other will be sent to the hosted AI model.
10. The AI model will take the input signal, run the model to determine what note it is, then send the result to the student.
11. The note will display on the student's screen.

#### 4.2.6.2 Flow Chart

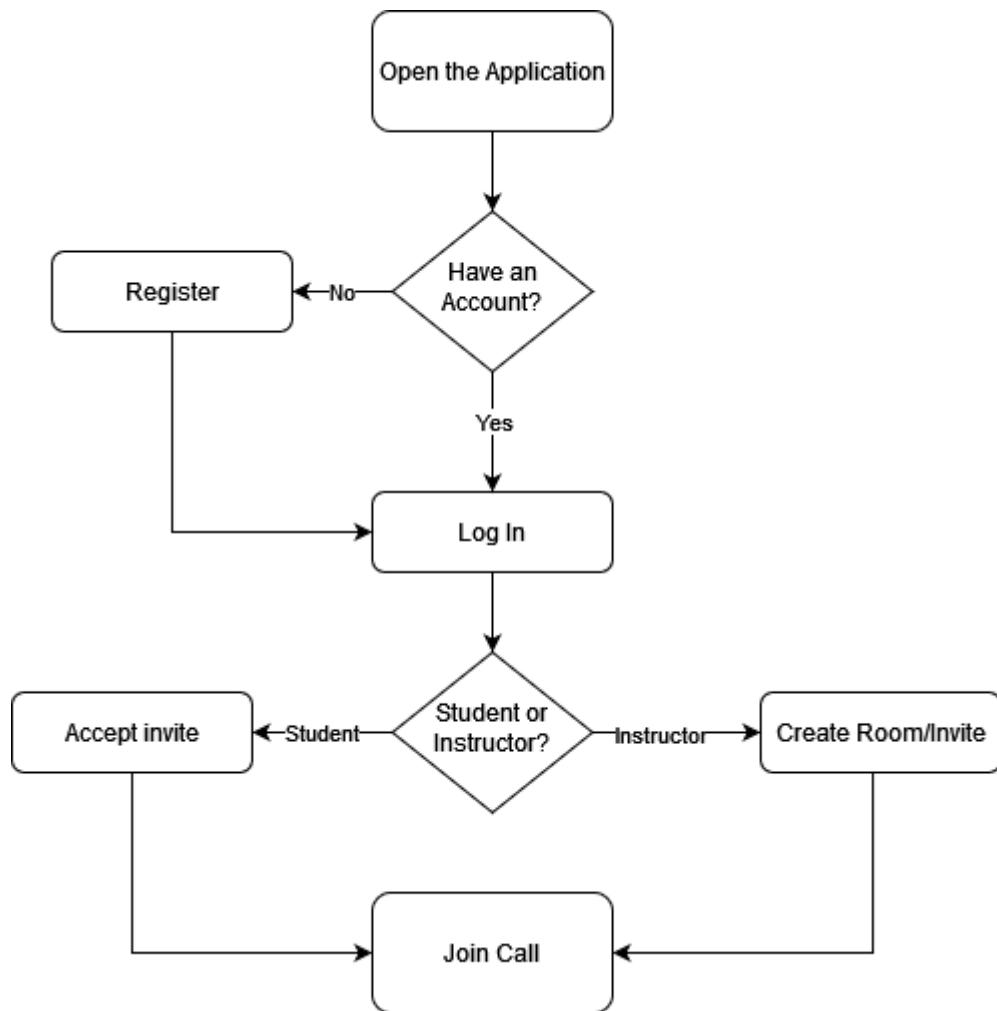


Figure 23: Flow Chart of User's activity

The flow chart in figure 23 details how a user will interact with the application. After opening the application, they must first register if they do not have an account. If they already have an account from previously using the application, they simply log in. The user's next step will depend on what type of user they are. If they are an instructor, they will create a room which will create an invite for the student. They will then join the call. A student will accept an invite from the instructor and then join the call.

## 4.3 User interface design

### 4.3.1 Wireframe

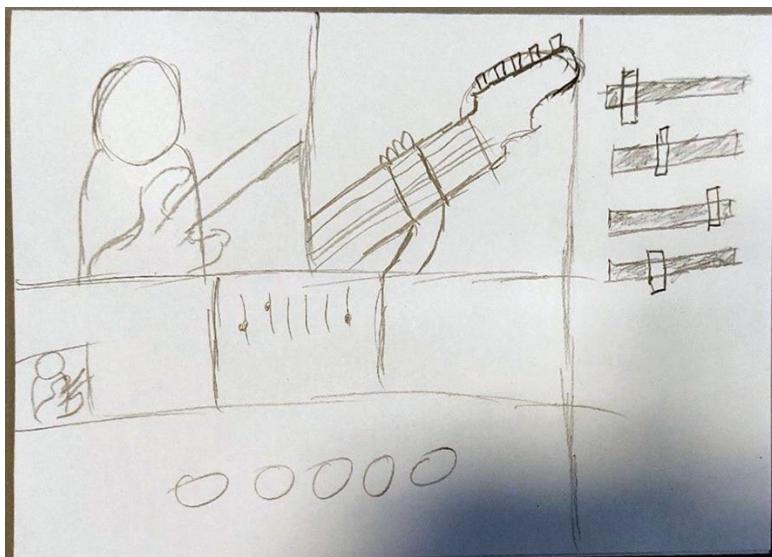


Figure 24: Paper Prototype of Video Call

Figure 24 shows the initial paper prototype drawing created during the early development stages. It was the first visualisation of the application and was used to further develop the design of the application.

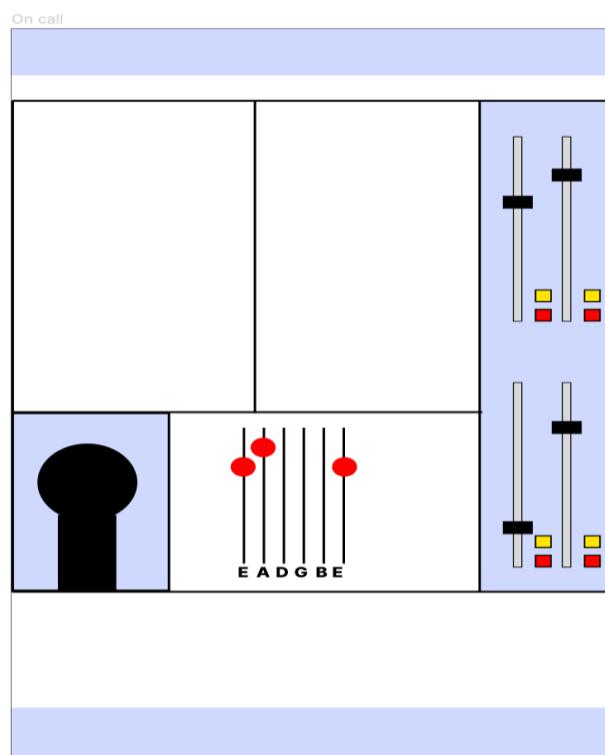


Figure 25: Wireframe for Video Call

Figure 25 shows the wireframe design for the students view when in a video call with an instructor. The two top rectangles will be for the instructors' cameras, while the bottom left section is for the student's webcam so they may monitor their own video feed if needed. Besides the student's feed will be the visualisation of the chord detection, which will show the student how to play a chord based on the note received from the hosted AI model. On the right side of the application is the mixer, where the student will be able to adjust both the instructor's and their own audio channels.

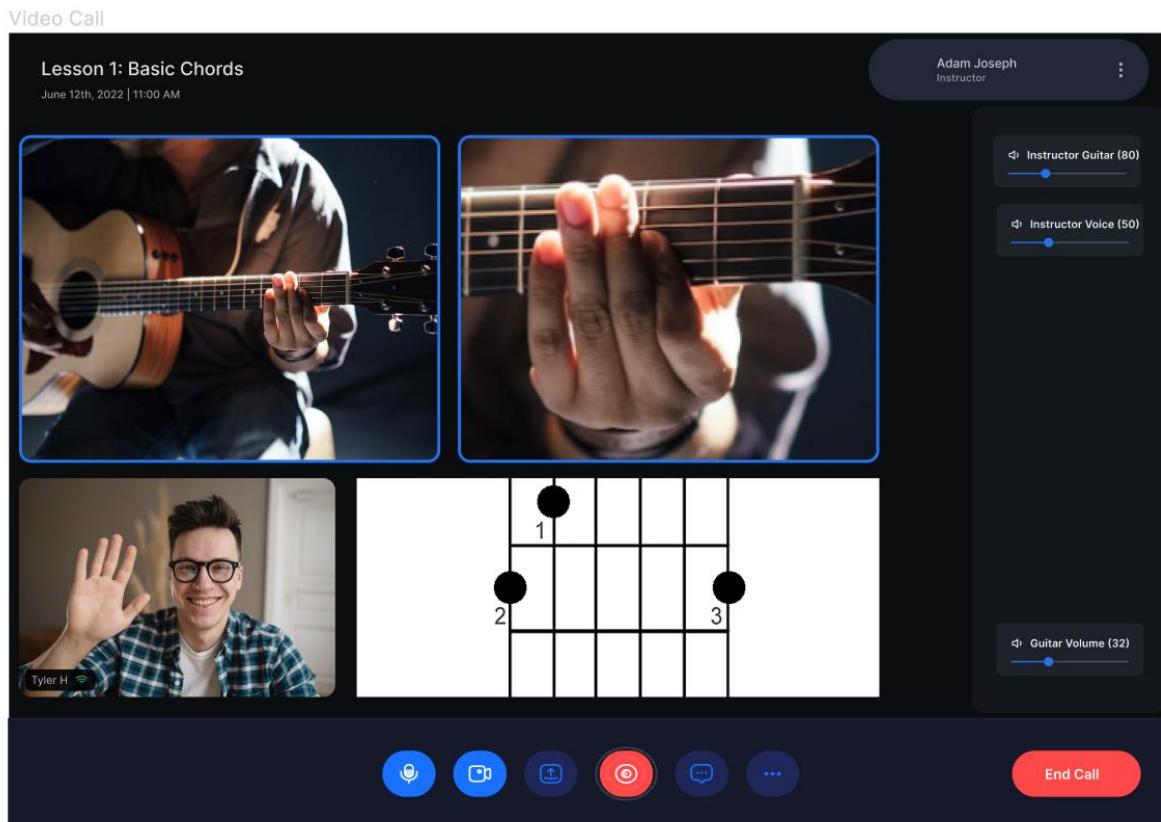


Figure 26: Hifi prototype of application

Figure 26 shows the Hifi prototype created in Figma. This closely resembles the intended design for the application and will be followed closely in order to develop a clear and concise UI for the final application.

#### 4.4 Conclusion

The design mentioned in this chapter is the original intention for the application. This design is subject to change depending on the implementation. For example, the application is currently designed for both users to be able to send two audio signals, one from a microphone and one from a guitar. While this functionality will remain, due to the complexity of sending two audio signals, it may be required to use third party library to merge the audio signals into one before sending it to the user. This will functionally be the same as both users will be able to hear both guitar and voice, but as there is only one audio source, the user's will not be able to manually control the separate audio tracks. This may also cause problems when sending a signal to the AI model, if the model is receiving an audio track containing background noise from a microphone, it will interfere with the model's accuracy of detecting the correct note.

## 5 Implementation

The application has many different components, all built separately then brought together to work as one. This chapter will discuss the development of each of these components and as well as the technologies used to develop them.

The components consist of both a frontend UI which allows the user to interact with the application, as well as a hosted backend server which is used to pass important data to the frontend as well as store user profile information.

### 5.1 Frontend

#### 5.1.1 React Redux

As discussed previously in this report, React is a frontend JavaScript library used for creating user interfaces, often creating each components of the interface separately, then bringing it all components together in a single file. React Redux includes the Redux library, which is used for managing state across the whole application. By using Redux, a component is able to access the state of any variable as long as it is stored correctly, which helps simplify the management of data flow.

```
export const setRoomDetails = (roomDetails) => {
  return {
    type: roomActions.SET_ROOM_DETAILS,
    roomDetails,
  };
};

export const setActiveRooms = (activeRooms) => {
  return {
    type: roomActions.SET_ACTIVE_ROOMS,
    activeRooms,
  };
};

export const setLocalStream = (localStream) => {
  return {
    type: roomActions.SET_LOCAL_STREAM,
    localStream,
  };
};
```

Figure 27: Redux Actions for handling video chat rooms

The decision to use Redux came during the restructure of the application's video call functionality. During the first build of the application, the video call was a simple WebRTC connection, none of the data of the call was stored and once the two users connected and disconnected the call, there would be no data leftover. While this implementation was functional, it had a fatal flaw in that if the

connection was interrupted for any reason, both users would be kicked from the call and would have to manually initiate another call. The video call implementation was restructured, by allowing a user to create a room, which other users could then join. This room would still exist if a user was to be accidentally disconnected, allowing the user to re-join if necessary, and the room would only be deleted if all users left the room. Using Redux, the application has the ability to store details about an active room, such as who created the room and how many participants are in the room, then this information can easily be passed to other users. Redux is also largely responsible for handling the data of a user's friends as well as handling the data for messages sent between users.

#### 5.1.2 Redux Thunk

Redux Thunk is middleware for Redux. It allows developers to use asynchronous actions that may return a function instead of an object, which in turn allows for state management which may not instantly return a value. For instance, in the application there is a send friend invitation function that is required to be asynchronous, as the friend request will not be answered instantly.

#### 5.1.3 Simple-Peer

Simple-Peer is a JavaScript library that provides a simplified version of the WebRTC API. It allows a peer-to-peer connection between two clients, allowing the transfer of video, audio, and data. More clients can be added to the video call using a mesh topology, in which all users are sending and receiving their data to each user on the call instead of sending their data to a server. However, due to the nature of peer-to-peer connections, there is a limit to how many clients may connect to a single connection.

#### 5.1.4 Spectrogram

Spectrogram is a JavaScript library that converts the live signal from an audio input such as a microphone to a Spectrogram. A spectrogram is a visual representation of the spectrum of frequencies within a signal. The AI model used with this application was trained on spectrograms, so by using this library to convert the signal to a spectrogram, the application can send an image representation of the guitar signal, so the model may identify the note and return it to the application.

#### 5.1.5 Firebase

Firebase is a Google owned mobile and web application development platform that offers a wide range of tools and services for developers. For the development of this application, the Firebase web

hosting services will be used to host the frontend of the application. Firebase provides free hosting for small projects with the option to upgrade should the application gain a lot of traffic in the future.

## 5.2 Backend

### 5.2.1 Express.js

Express is a framework for Node.js, a server-side JavaScript runtime environment. Using Express, a backend server for the application was created. This server would handle functionality such as creating and storing a user's profile, allowing users to send and receive friend invitations as well as handling the functionality of chat rooms and direct messages. The backend server also handles user authentication, restricting users from using the application if they were not registered or currently logged in.

### 5.2.2 Joi

Joi is library use for data validation within Node.js applications. Within the application, Joi is used to validate user information on both the login and register pages, restricting access if the inputs do not meet the requirements set by the library.

### 5.2.3 MongoDB/Mongoose

MongoDB is a database management system that allows users to store data in a NoSQL database. The information need for the application is stored in MongoDB. Mongoose is a JavaScript library that allows the user to connect to their MongoDB database and is used within the application to connect and send information to the database. The database stores the users, friend's invitations, conversation information as well as message information.

### 5.2.4 Bcrypt.js

Bcrypt.js is a JavaScript library that allows a user to convert plain text in an encrypted format. Within the application it is used when storing a user's password, as developers should not be able to see a user's password or other sensitive information within a database.

### 5.2.5 JSON Web Token

JSON Web Token is a JavaScript library used for generating web tokens used for authentication. When a user signs into the application, they are given a token, which allows them access onto otherwise restricted sections of the application.

### 5.2.6 Socket.IO

Socket.io is a JavaScript library built on the WebSockets API, that allows real-time communication between a web-client/frontend and a backend server. It establishes a persistent connection between the client and the server and listens for events to be triggered. When an event is triggered,

the user can choose what actions should take place, and can even use these events to emit data from the server to the client. One of the use cases within the application for Socket.io is the signalling of a new video chat room being created. Once the room has been created, an event will be emitted out containing the details of the room such as the creator of the room. If the current user is friends with user that created the room, that room will then appear on the user's profile, giving them the option to join it.

### 5.2.7 Heroku

Heroku is a cloud-based platform that enables developers to build and deploy applications. In the application, Heroku is used to host the backend server for the application. Initially Vercel was chosen to host the server, but unfortunately it does not support WebSockets natively, so it would be incapable of hosting the server containing Socket.io functionality. Once this was discovered, the server was deployed to Heroku.

## 5.3 Application Functionality

### 5.3.1 Login/Register

The first page the user sees when they open the application is the login page. From this page they can enter their login details or navigate to the register page if they do not have an account for the application. Both the login and register pages contain forms that contain error checking from self-made validator functions. These functions will not allow a password under or over a certain length, as these are the same rules for the passwords store in the database. Regex is also used in a function to determine if the input is an email in the email input. Once the information in the input box is validated, the button to either login or register will appear.

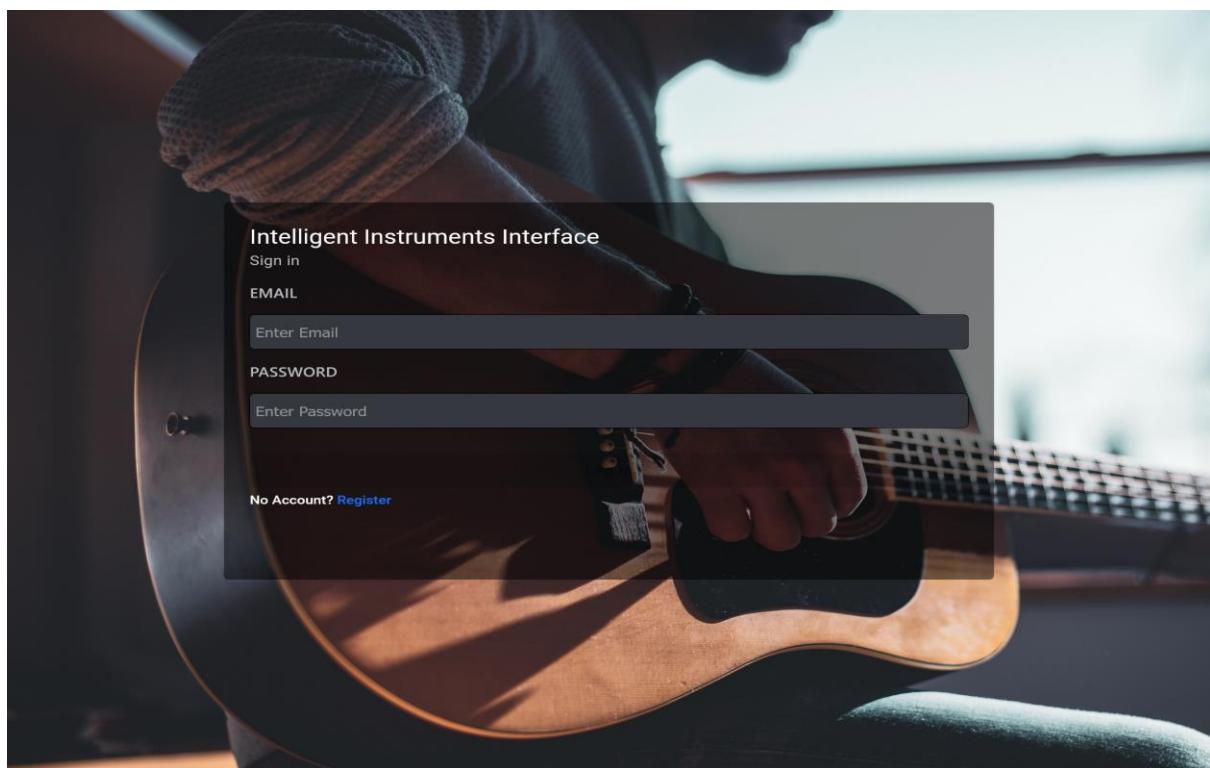


Figure 28: Login Page

On the login page, when the user presses the login button, the information will be sent to the backend server of the application to verify the information in the inputs. Once the information is retrieved by the server, it is validated by using the Joi library to ensure it once again is suitable for the database. Once validated, the email is then compared to the current emails in the database to find a match. Once a match is found, the passwords are then compared to ensure it is the correct account. If either the password is incorrect, or the email address is not found in the database, the server will return a 400 response, meaning the information submitted is incorrect. When both the email and password match, and token using the JsonWebToken library is created. The client will

then receive as response from the server, containing the user's email, token, username, id, and account. These details are then stored in the redux store so other components can make use of them, and then the user is navigated to the dashboard.

```
//Check if the user exists
const userExists = await User.exists({ email });

if (userExists) {
  return res.status(409).send("Email already in use");
}

//Encrypt password
const encryptedPassword = await bcrypt.hash(password, 10);

//Save user object in database
const user = await User.create({
  username,
  email: email.toLowerCase(),
  password: encryptedPassword,
  account,
});

//Create token
const token = jwt.sign(
{
  userId: user._id,
  email: user.email,
},
process.env.TOKEN_KEY,
{ expiresIn: "24h" }
);
```

Figure 29: Backend code for registering a User.

The register functionality has a lot of similarities with the login, the main difference being how the information is handled once it reaches the server. When the user sends the information from the form with the register button, the same validation functionality will take place on the frontend client and on the backend server with the Joi library. After validation, the server will then compare the email to all other emails stored in the database to ensure that the email is not already in use. If the email is already in the database, the server will return a 409 response, meaning the request could not be processed due to a conflict in the request. If the email does not exist in the database, the user's password is encrypted using the Bcrypt library, then the user's username, email, encrypted password, and account type are all stored in the database. A token will then be created and returned with the user's details. Same as the login, these details will then be stored in redux and the user is navigated to the dashboard.

## 5.3.2 Friend Functionality

### 5.3.2.1 Friend Invitation

Once a user has logged in for the first time, they will be unable to communicate with other users as they will have no friends in their friends list. To send messages or join a room created by another user, a user must either send or receive a friend request, and that request must be accepted. The user initiates the friend request by clicking the ‘Add Friend’ button. This will open a dialogue box with an input, form which the user will enter the email address of the user they would like to add. Once an email is entered, the user can press ‘Send’ to initiate the friend request. The API request is then sent to the backend server. All API requests within the application are made possible with the Axios library.

The server will receive data from the frontend client containing the target email the user would like to add, as well as the user’s id and email. Before the friend request can be sent, there are multiple layers of error checking. There is error checking to ensure the user email is not the same as the target email, and if it is the sender will receive a 409 response. The server will then check the database to see if the target email exists, and if it does exist, it will check that the invitation request does not exist in the database, as all friend requests are stored in the database. Finally, the server will check if the sender already has the target user’s id in the sender’s friends’ array on the database, meaning that the users are already friends. If all the error checking is passed, a new friend invitation object will be created, containing the sender’s user ID and the receiver’s user ID, and is stored in the database.

```
const updateFriendsPendingInvitations = async (userId) => {
  try {
    //Find friend invites where the userID is the receiver and populate the details
    const pendingInvitations = await FriendInvitation.find({
      receiverId: userId,
    }).populate("senderId", "_id username email");

    //Find if the user is active
    const receiverList = serverStore.getActiveConnections(userId);

    const io = serverStore.getSocketServerInstance();

    //For each socket ID, emit that user's friend invitations
    receiverList.forEach((receiverSocketId) => {
      io.to(receiverSocketId).emit("friends-invitations", {
        pendingInvitations: pendingInvitations ? pendingInvitations : [],
      });
    });
  } catch (err) {
    console.log(err);
  }
};
```

Figure 30: Sending friend requests to online users.

Now that the friend invitation exists, there must be a way for the receiver to know that they have been sent a friend request. As soon as the invite object is stored in the database, another function on the server is called. This function will first find all friend invitations in the database, where the user Id is the same as the receiver Id stored. It will then use a function in the server store to check all current online connections and put all the user socket Id into an array. It will then use socket io to emit on the client side to each online user if they have any pending friend requests and contain sender's data. Once the data from the socket event is received, it is stored in the redux store, and then used to display the information to the user.

### 5.3.2.2 Accepting/Denying Friend Invitations

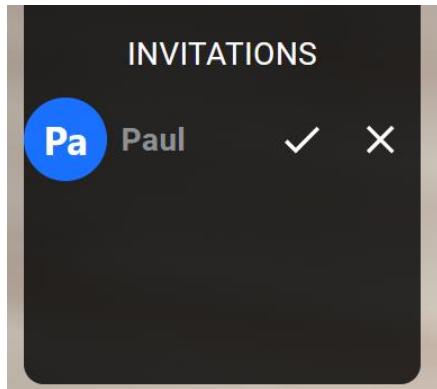


Figure 31: Friend Invitation

After receiving an invitation, the user will see two prompt buttons on the screen - one for accepting and one for denying the friend request. The functionality for these actions is similar, and the data indicating whether the request was accepted or denied will be sent to the server using a dispatch async function, which waits for a response. The redux thunk library enables the redux action to run asynchronously. The main difference between the two functions is how the data is handled on the server.

```

const { id } = req.body;

const invitation = await FriendInvitation.findById(id);

if (!invitation) {
  return res.status(401).send("Error occurred. Please try again");
}

const { senderId, receiverId } = invitation;

//Add both users to each other's friends array
const senderUser = await User.findById(senderId);
senderUser.friends = [...senderUser.friends, receiverId];

const receiverUser = await User.findById(receiverId);
receiverUser.friends = [...receiverUser.friends, senderId];

await senderUser.save();
await receiverUser.save();

//Delete the invitation from the database
await FriendInvitation.findByIdAndUpdate(id);

// Update list of the friends if the users are online
friendsUpdates.updateFriends(senderId.toString());
friendsUpdates.updateFriends(receiverId.toString());

//Update list of friends pending invitations
friendsUpdates.updateFriendsPendingInvitations(receiverId.toString());

return res.status(200).send("Friend successfully added");

```

*Figure 32: Accepting friend request.*

For accepting an invitation, the server receives the id of the friend invitation, then searches for the friend request object in the database. If it cannot find the friend request in the database, it will return a 401 error. Then it will retrieve the sender Id and the receiver Id from the friend request object, then it will add both users to each other's friend's array in the database. The friend invitation object will then be deleted from the database, and then the friends list of both the sender and receiver will be updated using a function from the socket handlers, which will emit this to each user if they are online. Once the friend list is updated, the name of the other user will now appear in their friend's list. The list of pending friend invitations is also updated, and a response of 200 is returned to the client. This will then dispatch an alert message using redux stating that the invitation has been accepted.

```

const { id } = req.body;
const { userId } = req.user;

// Find the invitation
const invitationExists = await FriendInvitation.exists({ _id: id });

//Remove the initiation
if (invitationExists) {
  await FriendInvitation.findByIdAndDelete(id);
}

//Update list of friends pending invitations
friendsUpdates.updateFriendsPendingInvitations(userId);

return res.status(200).send("Invitation successfully rejected");

```

Figure 33: Rejecting friend request.

If the invitation is rejected, the server will simply find the friend request on the database, then delete it. It will then update the pending invitations, return a response to the client which then will dispatch and alert message stating the request was denied.

#### 5.3.2.3 Online status

The application has the functionality to show users which of their friends are currently online. This functionality begins on the backend server, which contains a function that pushes all the socket IDs and user IDs of the users currently connected to the application into an array. This array is then emitted to the frontend client using socket io. This is repeated every couple of seconds, so the list of online users will be constantly updated.

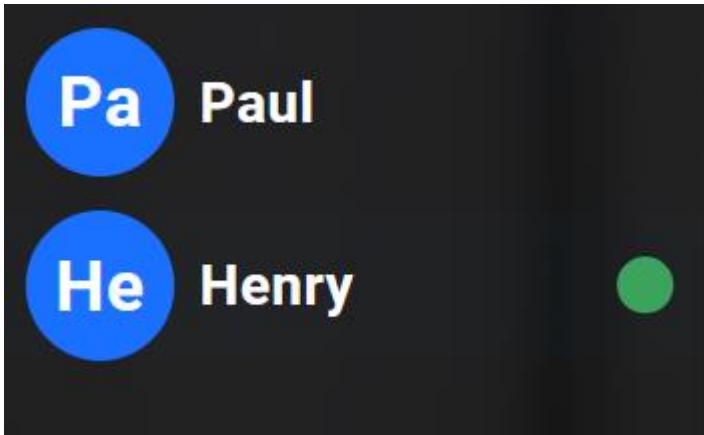


Figure 34: User online.

Once the client receives the list of online users, it will store the array of online users in the redux store. To then determine which of the user's friends are online, the user's friend's Ids are then compared to the user Id's in the online users array. If one of the Id's from the friend array is in the online users array, the 'isOnline' property in the friends list item component will be true, which will render the online indicator next to that user's name in the friend's list.

### 5.3.3 Direct Messaging

For direct messages to work on the application, the redux store had to be configured. In the redux store there is both chat actions which holds the various functions needed to send the messages and the chat reducer which stores the state of the messages. The chat messages were given two types, direct messages for messages to be exchanged between two users, and the second type which is not currently implemented is the group chat type. The group chat type as added so it will be easier to implement should group chats be added as a feature in future versions of the application.

```
const handleChooseActiveConversation = () => {
  setChosenChatDetails({ id: id, name: username }, chatTypes.DIRECT);
};

return (
  <Button
    onClick={handleChooseActiveConversation}
    style={[
      width: "100%",
      height: "42px",
      marginTop: "10px",
      display: "flex",
      alignItems: "center",
      justifyContent: "flex-start",
      textTransform: "none",
      color: "black",
      position: "relative",
    ]}
  >
  <Avatar username={username} />
  <Typography
    style={[
      marginLeft: "7px",
      fontWeight: 700,
      color: "#FFFFFF",
    ]}
    variant="subtitle1"
    align="left"
  >
    {username}
  </Typography>

```

Figure 35: Friends list item component.

The ‘setChosenChatDetails’ function will need the recipient user to receive a message. To set these details, the function is imported into the friends list item through redux. The friend list item will contain the user’s Id and username. From this component, the user can select the name of the user they would like to send a message to. When a user is selected, the selected user’s Id and username will be passed into the function alongside the user who selected Id, and the details of the chat are then saved in the redux store. Another component ‘MessengerContent’, will take in these chat details as a prop.

```
useEffect(() => {
  getDirectChatHistory({
    receiverUserId: chosenChatDetails.id,
  });
}, [chosenChatDetails]);
```

Figure 36: Messenger Content UseEffect.

It contains a use effect function, which will wait for the chat details to be updated in the redux store. Once the chat details are updated in the store, it will use the ‘getDirectChatHistory’ function. This function emits a socket io event containing the current user Id and the Id of the user they wish to message. The server will then use the Ids of both users to find a conversation on the database that contains both users as participants and as type ‘DIRECT’. In the database, there is a conversations object and an messages object. The messages will contain the id of the object, the id of the author or sender of the message, the content of the message, the date and time, and the type of message. The conversations object will contain the participants, and an array of message Ids. If the conversation is found it will be returned to the client. The message history is then rendered out to the user. If there is no chat history found between the two users a message is displayed instead.

```
// Create new message
const message = await Message.create({
  content: content,
  author: userId,
  date: new Date(),
  type: "DIRECT",
});

// Find if conversation exist with this two users - if not create new
const conversation = await Conversation.findOne({
  participants: { $all: [userId, receiverUserId] },
});

if (conversation) {
  conversation.messages.push(message._id);
  await conversation.save();

  //Update the sender and receiver if they are online
  chatUpdates.updateChatHistory(conversation._id.toString());
} else {
  // Create new conversation if one doesn't exists
  const newConversation = await Conversation.create({
    messages: [message._id],
    participants: [userId, receiverUserId],
  });

  //Update the sender and receiver if they are online
  chatUpdates.updateChatHistory(newConversation._id.toString());
}
```

Figure 37: Direct Message Handler.

If the current user wishes to send a direct message to the selected user, they can enter their message into the input box and press enter. This will emit a socket io event to the server containing the message object. Once the server receives the event, the data is passed to the message handler. The server will first create a message object to be stored in the database. It will then search the database to see if a conversation between the two users exists. If a conversation exists in the database, the message is added to the conversation object for the two users, and then the chat history is updated for the two participants if they are both online, so both users can now see the

chat updating in real time. If there is no conversation in the database, one is created with the new message Id and the Id's of both participants, and the chat history is updated.

### 5.3.4 Rooms

#### 5.3.4.1 Room Creation

In order for users to establish a peer-to-peer connection so they can begin to exchange video and audio data, a room must first be created. To create a room, a user can press the 'CreateRoomButton' component, which will call the 'createNewRoom/' function from the room handler. The 'createNewRoom' function will first dispatch the 'setOpenRoom' function, which assigns the user as the creator of the room and updates the store to say the user is currently in the room. It will then emit a socket io event to the server which calls the room create handler.

```
const roomCreateHandler = (socket) => {
  console.log("handling room create event");
  const socketId = socket.id;
  const userId = socket.user.userId;

  const roomDetails = serverStore.addNewActiveRoom(userId, socketId);

  socket.emit("room-create", {
    roomDetails,
  });

  roomsUpdates.updateRooms();
};

module.exports = roomCreateHandler;
```

Figure 38: Room Create Handler.

The handler will take the socket id and user id from the socket event and pass these details into the 'addNewActiveRoom' function from the server store. This function will assign the room creator, which is the user who initiated the function and add the user's Id and socket Id to an array of participants. A unique room Id will then be created with the `uuidv4` library. It will then add this new created active room the array of current active rooms. Once the room is created, the room details and emitted back to the client and the 'updateRooms' function will run. This function which will emit an event that contains all the current active rooms to all users. The client will receive the details of the new room through socket io and will then store the details of the room in the store.

#### 5.3.4.2 Displaying Active Rooms

```
export const updateActiveRooms = (data) => [
  const { activeRooms } = data;
  //Get friends of user
  const friends = store.getState().friends.friends;
  const rooms = [];

  //Get User Id
  const userId = store.getState().auth.userDetails?._id;

  //Go through all rooms
  activeRooms.forEach((room) => {
    //Check if user created the room
    const isRoomCreatedByMe = room.roomCreator.userId === userId;

    if (isRoomCreatedByMe) {
      //Show Room named as "Me" if user created
      rooms.push({ ...room, creatorUsername: "Me" });
    } else {
      //Show Room with friend name if friend created
      friends.forEach((f) => {
        if (f.id === room.roomCreator.userId) {
          rooms.push({ ...room, creatorUsername: f.username });
        }
      });
    }
  });

  store.dispatch(setActiveRooms(rooms));
];
```

Figure 39: Update Active Rooms

Even though the active rooms are emitted to all users through the ‘updateRooms’ function, only the rooms created by their friends will be visible to the user. The application ensures that only rooms created by the user’s friends are visible to the user in order to prevent them from joining the wrong room and potentially interrupting ongoing lessons. On the client side, there is a socket event listener named ‘active-rooms’ that triggers the ‘updateActiveRooms’ function. This function retrieves the array of active rooms from the socket event and fetches the user’s friend list and ID from the store. It then iterates through each room in the array, checking if the room was created by the user or their friends. If the room was created by the user, it is rendered as ‘Me’ in the sidebar. If the room was created by a friend, the friend’s name is listed on the join button. Any rooms not created by the user or their friends are ignored. Once the iteration is complete, the updated list of active rooms is saved in the Redux store. This list is what is used to display the available rooms to the user.

#### 5.3.4.3 Joining a Room

Once the active rooms are being displayed to the user, a user can join a room by clicking on the button related to that room. Once the user presses a button related to a room, the ‘joinRoom’ function from the room handler will initiate. It takes the Id of the room and saves it in the redux store. It will then use the ‘setOpenRoom’ function but unlike in ‘createNewFunction’, the ‘isUserCreator’ prop will be false as the user is joining an already active room. A ‘join-room’ socket event will then be emitted from the client containing the room Id.

```

const roomJoinHandler = (socket, data) => [
  //Get room Id from socket data
  const { roomId } = data;

  //Get user details from socket
  const participantDetails = {
    userId: socket.user.userId,
    socketId: socket.id,
  };

  //Get room details from server store
  const roomDetails = serverStore.getActiveRoom(roomId);
  serverStore.joinActiveRoom(roomId, participantDetails);

  //Send information to users in room that they should prepare for incoming connection
  roomDetails.participants.forEach((participant) => {
    if (participant.socketId !== participantDetails.socketId) {
      socket.to(participant.socketId).emit("conn-prepare", {
        connUserSocketId: participantDetails.socketId,
      });
    }
  });
  roomsUpdates.updateRooms();
];

```

Figure 40: Room Join Handler

On the server, there will be an event listener that will initiate the ‘roomJoinHandler’. This function will get the room Id from the data passed through socket io and get the user details from the socket connection. It will use the room Id to search for the active room on the server store, and then add the user to that room. Once the user has been added, the function will emit a ‘conn-prepare’ event to every user currently in the active room, to prepare the users for the peer connection. The ‘updateRooms’ will then run to update the details of the current active rooms.

#### 5.3.4.4 Resize Room

When the user is active in the room, they have the option to minimize the room, should they want to check their messages or perform other activities during a call while still being able to have a clear view of the video call. To minimize the video call, the user can press the icon in the bottom right-hand corner of the room window. The resize button will call a function that simply change the style of the ‘MainContainer’ in the ‘Room’ component. Instead of the container taking up the height and width of the screen, with the minimized style it will only take up a small section in the bottom right-hand corner of the overall application.

```

const VideoWrapper = styled("div")(({ isRoomMinimized }) => ({
  position: "absolute",
  left: isRoomMinimized ? "0px" : "30vw",
  top: isRoomMinimized ? "0" : "10vh",
  right: isRoomMinimized ? "0px" : "0",
  width: isRoomMinimized ? "200%" : "initial",
  height: isRoomMinimized ? "90%" : "initial",
}));

const LocalVideoWrapper = styled("div")(({ isRoomMinimized }) => ({
  position: "absolute",
  bottom: "5%",
  left: 20,
  zIndex: 1,
  visibility: isRoomMinimized ? "hidden" : "visible",
}));

```

*Figure 41: Styling Room Minimized.*

However, this alone would not be enough for the functionality, as the ‘VideoContainer’ is inside the ‘MainContainer’ which will severely affect the style of the videos inside the ‘VideoContainer’. To rectify this, the ‘isRoomMinimized’ prop is passed down the ‘VideoContainer’ component, which will pass the information to the component as to whether or not the room has minimized. Conditional statements are then used in the styled components ‘VideoWrapper’, ‘LocalVideoWrapper’ and ‘ChordDisplayWrapper’ as show in figure 39. If ‘isRoomMinimized’ is true, ‘VideoWrapper’ will be resized to fit the size of the minimized window so the user can still see the other participant of the call. The other two components will have their visibility set to hidden, as while all three components could fit in the window, they would be far too small to see clearly. If the user would like to resize the window back to normal, then can click the new resize button which replaced the original when the room size changed.

#### 5.3.4.5 Leaving a Room

When a user is in a room, they have an option to leave the room by pressing the disconnect button.

This button will run the ‘leaveRoom’ function from the ‘roomHandler’ on the client side.

```
export const leaveRoom = () => {
  //Get room Id from redux
  const roomId = store.getState().room.roomDetails.roomId;

  //Check if user has a local stream
  const localStream = store.getState().room.localStream;
  if (localStream) {
    //Stop all tracks
    localStream.getTracks().forEach((track) => track.stop());
    //Set to null
    store.dispatch(setLocalStream(null));
  }

  //Check if user sharing screen
  const screenSharingStream = store.getState().room.screenSharingStream;
  if (screenSharingStream) {
    //Stop all tracks
    screenSharingStream.getTracks().forEach((track) => track.stop());
    store.dispatch(setScreenSharingStream(null));
  }

  // Set remote streams to empty array
  store.dispatch(setRemoteStreams([]));
  // Close peer connections
  webRTCHandler.closeAllConnections();

  //Leave room
  socketConnection.leaveRoom({ roomId });
  //Set room details to null
  store.dispatch(setRoomDetails(null));
  //Remove user from room
  store.dispatch(setOpenRoom(false, false));
};

];
```

Figure 42: Leave Room Function

This function will first get the room Id from the redux store. It will then check if the user’s local stream (active camera or microphone) or screen sharing is live, and if they are they will stop all the tracks and set them to null in the redux store. The function will then set the remote streams array, which contains information about the peers to which they may be connected to null. The ‘WebRTCHandler’ will then close all peer connections. Socket io will then emit a ‘leave-room’ event along with the room Id, and then the room details in the redux store will be set to null, and then the user will be removed from the room.

The ‘leave-room’ event will initiate the ‘roomLeaveHandler’ on the server. This will use the room Id passed from the client to remove the user’s socket Id from the active room in the server store. It will then check if there are any users left in the active room. If there are no users left in the room, it will be removed from the active rooms. If there are users left in the room, an ‘room-participant-left’

event will be emitted to each user still in the room so they may close their peer connection with that user. The ‘updateRooms’ will then run to update the details of the current active rooms.

### 5.3.5 Peer Connections

#### 5.3.5.1 Local Stream

When the user joins a room the ‘getLocalStreamPreview’ function will run. This function will run before the functionality explained in the ‘Join a room’ section. This function will ask the user for their permission to use their audio and video inputs, and if they have multiple inputs devices, they will be able to choose which devices they would like to use. If the user declines both inputs or there is an error in retrieving the data from the devices, ‘getLocalStreamPreview’ will return an error which not allow the ‘joinRoom’ function to run, which in turn will not allow the viewer to join a room.

```
//Get user Devices
navigator.mediaDevices
  //Check constraints: audio only?
  .getUserMedia(constraints)
  .then((stream) => {
    //If there is a media/guitar stream, add it
    if (mediaStream) {
      stream.addTrack(mediaStream.getAudioTracks()[0]);
    }
    //Set local stream in redux store
    store.dispatch(setLocalStream(stream));
    callbackFunc();
  })
  .catch((err) => [
    console.log(err);
    console.log("Cannot get an access to local stream");
  ]);
}
```

Figure 43: Get Local Stream Preview

The function will first ask the user for their input devices, checking the constraints. The user has the option to select an audio only option when joining the room, however they must select it before initiating the local preview. Once it receives the devices it will then check if there is a media stream. This media stream will be the guitar audio coming from another component, that will be stored in the redux store. If the media stream exists, it will be added as an audio track to the local stream. The local stream is then saved, and it will run any call-back function passed into the function as a prop, the call-back being used in the application is the ‘join-room’ function.

```

// Checking if there are at least two cameras available
if (numCams >= 2) {
  // Enumerating all media devices
  navigator.mediaDevices
    .enumerateDevices()
    .then((devices) => {
      console.log("Enumerated devices:", devices);

      // Filtering video input devices
      const videoDevices = devices.filter(
        (device) => device.kind === "videoinput"
      );
      // Setting constraints for the first camera
      const constraints1 = {
        video: { deviceId: videoDevices[0].deviceId },
        audio: true,
      };
      // Setting constraints for the second camera
      const constraints2 = { video: { deviceId: videoDevices[1].deviceId } };
      // Getting media stream for the first and second camera
      Promise.all([
        navigator.mediaDevices.getUserMedia(constraints1),
        navigator.mediaDevices.getUserMedia(constraints2),
      ])
        .then(([stream1, stream2]) => [
          console.log("Stream 1:", stream1);
          console.log("Stream 2:", stream2);
          // Creating a new MediaStream object
          const localStream = new MediaStream();
          //Adding tracks to media stream
          localStream.addTrack(stream1.getVideoTracks()[0]);
          localStream.addTrack(stream2.getVideoTracks()[0]);
          localStream.addTrack(stream1.getAudioTracks()[0]);

          if (mediaStream) {
            localStream.addTrack(mediaStream.getAudioTracks()[0]);
          }

          console.log("Local stream:", localStream);

          store.dispatch(setLocalStream(localStream));
          callbackFunc();
        ])
    });
}

```

Figure 44: Get Local Stream Preview 2 Cameras

If the user opts to use a single camera, the local stream operates as previously described. However, there is an option for the user to utilize a second camera, albeit with some unresolved bugs that currently hinder the reception of audio by peers. When the user selects the second camera, the 'getLocalStreamPreview' function employs 'enumerateDevices' instead of 'getUserMedia'. This function retrieves a list of all available devices, eliminating the need for the user to manually pick a specific device. Subsequently, the list of devices is filtered to include only video input devices. Two constraints are then created: one for the first video input device that includes audio from the first camera in the list, and another for the second video input device that uses only the video from the second camera in the list. This approach prevents the occurrence of two microphone inputs from a single user.

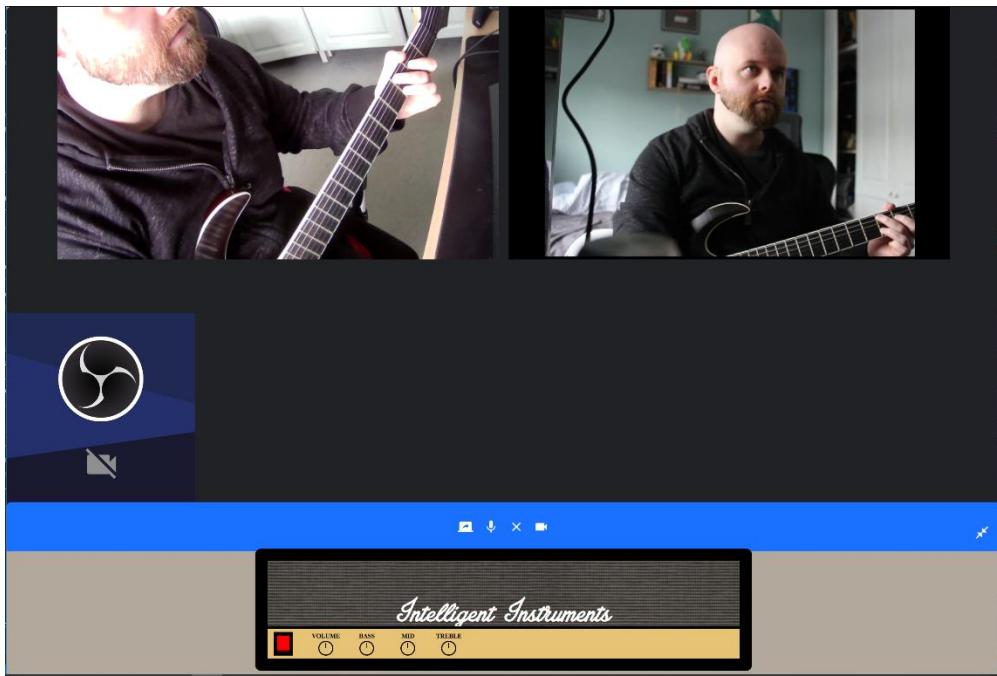


Figure 45: Testing Video call with 2 cameras

The 'getUserMedia' function is then used to retrieve media streams from the selected devices based on the created constraints. A new MediaStream object is created, and the video and audio tracks from the devices are added to it. If a guitar stream is available, it is also added to the local stream. The resulting media stream object is then stored in the Redux store. However, currently, it is unclear why this function is not fully operational at the time of writing. When a user who is creating a room uses two cameras, they can see both camera feeds in their local stream, but their peers are unable to see both video streams. However, they can hear the microphone and guitar audio tracks. On the other hand, if a user who is using two cameras joins a room after it has been created, peers can see the video feeds but are unable to hear any audio tracks from the user.

Once the user has their local stream, it will be passed into the 'Video' component inside the 'VideosContainer' component. The 'Video' component takes in two props, 'stream' which is the object containing the video element and 'isLocalStream' which is a Boolean indicating if the video stream is local. The 'useRef' hook is used to create a reference to the video element. The 'useEffect' hook is used to update the video element with the provided stream object as the video source and play it when metadata is loaded. The video element is rendered and is muted if the 'isLocalStream' is set to true. In the 'VideoContainer', the 'LocalVideoWrapper' is used to position the local stream in the bottom left corner of the room.

### 5.3.5.2 Remote Streams

To allow users to join a room and establish a connection with one another, the application uses the WebRTC Mesh Architecture previously discussed in this report. It uses the Simple Peer package to simplify the WebRTC functionality of the application. As discussed in the ‘Joining a Room’ section, when the user joins a room, a socket event called ‘conn-prepare’ is emitted to all the users currently in the room. When the client receives this event, it will use the ‘prepareNewPeerConnection’ in the WebRTC Handler. The ‘prepareNewPeerConnection’ will take in the user’s socket Id and a Boolean stating if the user is the initiator of the connection. If the user is receiving the ‘conn-prepare’ event they are not the initiator.

```
//Get Local Stream
const localStream = store.getState().room.localStream;
//Get guitar Stream
const guitar = store.getState().room.guitarStream;

// Turn local stream into combined tracks
const combinedTracks = [...localStream.getTracks()];

// If guitar stream isn't empty, push the tracks to combined tracks
if (guitar && guitar.mediaStream) {
  combinedTracks.push(...guitar.mediaStream.getTracks());
}

//Create a new media stream from combined tracks
const combinedStream = new MediaStream(combinedTracks);

// Log whether the current user is the initiator of the connection or not
if (isInitiator) {
  console.log("preparing new peer connection as initiator");
} else {
  console.log("preparing new peer connection as not initiator");
}

// Create a new Peer object and add it to the peers object using the socket ID as the key
peers[connUserSocketId] = new Peer({
  initiator: isInitiator,
  config: getConfiguration(),
  stream: combinedStream,
});
```

Figure 46: Prepare New Peer Connection

The function will first get the local stream and the guitar stream to combine them into one media stream object. It will then create a new Peer object using the simple peer library. The peer object can take in different configurations. If the initiator property is true, it will automatically try to connect to another user, if it is false it will wait for a connection. The configuration takes in the ICE candidates from the stun server as discussed previously in the report. Finally, the stream property will take in the local stream.

```

// When the Peer object emits a 'signal' event, send the signal data to the server
peers[connUserSocketId].on("signal", (data) => {
  const signalData = {
    signal: data,
    connUserSocketId: connUserSocketId,
  };
  socketConnection.signalPeerData(signalData);
});

// When the Peer object emits a 'stream' event, add the new remote stream to the Redux store
peers[connUserSocketId].on("stream", (remoteStream) => {
  console.log("remote stream came from other user");
  console.log("direct connection has been established");
  remoteStream.connUserSocketId = connUserSocketId;
  addNewRemoteStream(remoteStream);
});
});

```

Figure 47: Prepare New Peer Connection pt.2

The function then adds an event listener for the ‘signal’ event, which will pass the user’s signalling data. It will then use the ‘signalPeerData’ function, which emits the “conn-signal” event to the server so the user can pass their signalling data to other users. The second event listener is for the “stream” event which will occur if two users establish a connection. The listener will pass the remote stream to the ‘addNewRemoteStream’ function, which will save the incoming remote stream into the redux store.

```

const roomInitializeConnectionHandler = (socket, data) => {
  const { connUserSocketId } = data;

  const initData = { connUserSocketId: socket.id };
  socket.to(connUserSocketId).emit("conn-init", initData);
};

module.exports = roomInitializeConnectionHandler;

```

Figure 48: Room Initialize Connection Handler

When the “conn-prepare” event happens, alongside the “prepareNewPeerConnection” function, a “conn-init” function will be emitted to the server. On this event, the server will use the “roomInitializeConnectionHandler” to take in the socket Id of the user and the socket Id of the user they would like to connect to. It then emits the “conn-init” event directly to that user, to prepare for the peer connection. On the client side, the "conn-init" event is used to trigger the “prepareNewPeerConnection” function with the “connUserSocketId” received from the server, indicating that a new peer connection is being initiated on the client-side for a specific user identified by the “connUserSocketId”.

On the “conn-signal” event, the server will use the “roomSignalingDataHandler”. This handler will send the signalling data to the socket Id of the user to connect with, as well as containing the socket Id of the user that is sending the signalling data through a socket event called “conn-signal”. On this event in the client, the “handleSignalingData” function will initiate. This will then add the signalling data to the peer object with the socket Id, allowing the users to exchange SDP and ice candidate information to establish the peer connection. This will complete the “signaPeerData” function which will allow the users to exchange remote streams and allow the flow of video and audio data. The remote streams can now be rendered through the same “Video” component used so both users may see and hear each other’s streams.

### 5.3.6 Chord Detection

#### 5.3.6.1 Guitar Playback

For the chord detection to work, the user must first turn on guitar playback, which will allow the user to hear their instrument being played back to them, as well as adding the track created to local and remote streams so other users may hear their instrument.

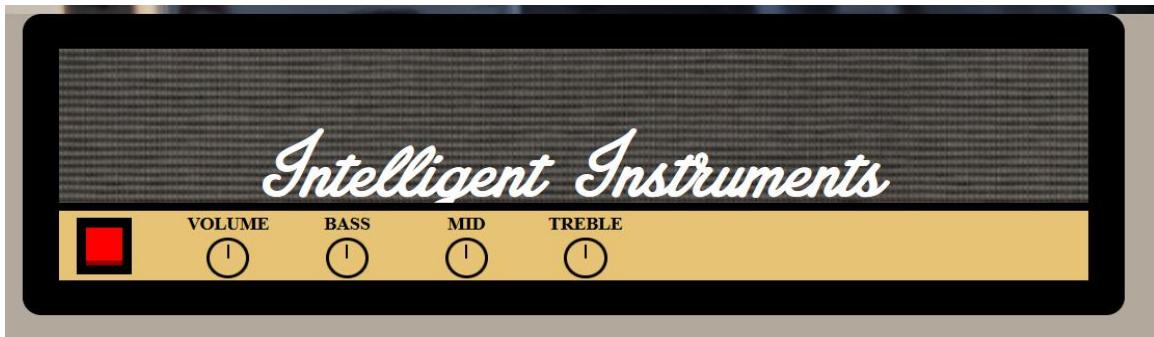


Figure 49: Amplifier UI

The user controls the guitar playback through the 'Amp' component. To initiate playback, the user can press the red power button on the UI, which triggers the 'setUpContext' function within the component. This function first calls the 'setUpGuitar' function, which prompts the user to select the audio device for playback with the "Amp" component. The 'getUserMedia' function is used, but with specific parameters to enable audio only and disable certain settings that are typically used for improving voice quality through a microphone. Enabling these parameters could negatively impact the guitar signal from the user. After the input device has been selected, and media stream source will be created from the input device. The source node is connected to the 'gainNode' using the connect method, which establishes an audio connection between two nodes in the audio graph. This means that the audio data from the guitar media stream will flow into the 'gainNode', allowing the user to control the volume of the audio before it is played back to the user. The source node is then stored in the redux store so it may be used elsewhere in the application, such as when adding it to the peer connection. If the user turning on the 'Amp' is an instructor, the component that handles the displaying of the predicted chords will also be turned on.

Once the user can hear their instrument being played back, they will be able to adjust the volume and other parameters of the signal. This is possible through the 'RangeInput' component, which is displayed as knobs on the Amplifier UI. The component will take the props of a label and a set state function from the "Amp" component, which will allow the 'RangeInput' component to change the values of states inside the "Amp" component. It maintains the current value of the input as "value" state and the rotation degree of the knob as 'deg' state. The 'useEffect' hook is used to update the

'deg' state based on the 'value' state. It calculates the rotation degree of the knob based on the current value and updates the 'deg' state accordingly. The component also defines event handlers for mouse events like 'handleMouseDown', 'handleMouseMove', and 'handleMouseUp' to handle user interactions with the knob. When the user clicks and drags the knob, the 'handleMouseDown' event is triggered, which calculates the new value based on the mouse movement and updates the "value" state accordingly. When the user releases the mouse button, the '#handleMouseUp' event is triggered, which removes the event listeners for mouse movement, ending the interaction.

#### 5.3.6.2 *Guitar To Spectrogram*

A spectrogram is a visual representation of the spectrum of frequencies in a signal over time. It is commonly used in signal processing and audio analysis to analyse the frequency content of a signal, such as sound waves or other types of signals. The AI model that will be used to detect the guitar chords from the application was trained on spectrograms, so to get a prediction from the model, the guitar signal must first be converted into a spectrogram. This is done by using the "Spectrogram" package.

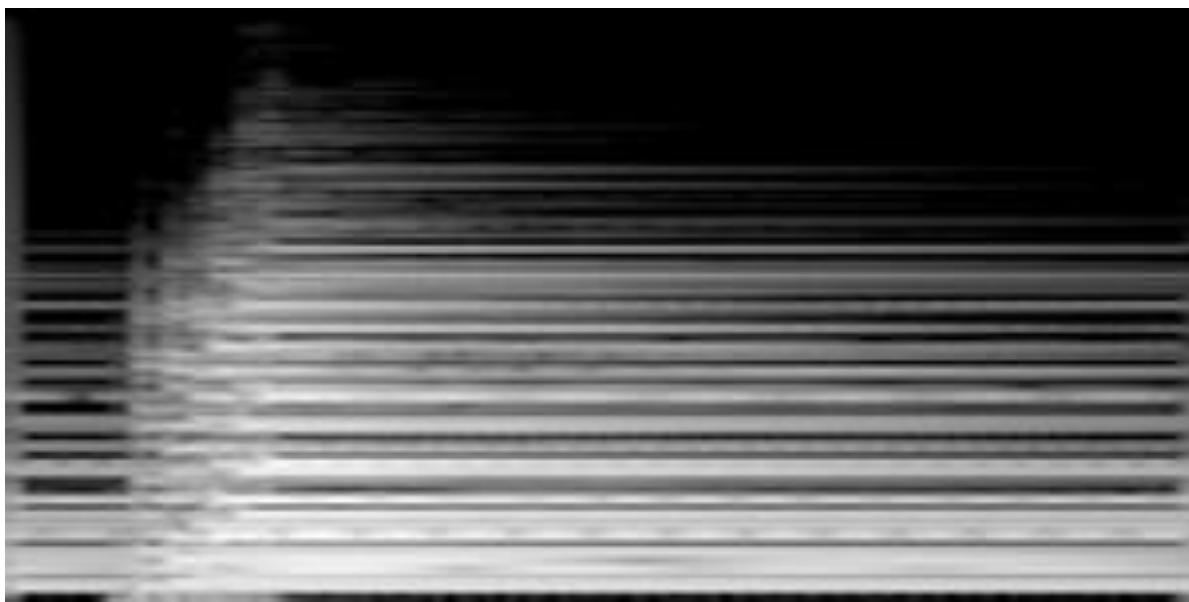


Figure 50: Guitar Chord Spectrogram

The Spectrogram package operates by taking an audio signal and generating a visual representation of that signal in the form of a spectrogram. In the 'ChordDetect' component, a prediction state is initialized using the `useEffect` hook, which is triggered by the value of the 'onChord' property passed down from the 'Amp' component. Once the 'setUpContext' function in the 'Amp' completes and sets the 'onChord' to true, the `useEffect` function is executed. It starts by initializing a canvas element and the Spectrogram library, which takes the canvas as input. Then, an audio context is created from the guitar stream state stored in the Redux store, and it is set up as the input for the spectrogram to

produce a signal. A screenshot of the canvas is taken every 4 seconds and passed to the submit function so it may be sent to the model for prediction. Although the spectrogram visualization is visible during the development of this functionality, it is hidden from the user in the completed application.

The submit function captures a screenshot as a Blob, which is a Binary Large Object representing raw data. Subsequently, the Blob file is converted into a jpeg image file. This image file is then transmitted to the Microsoft Custom Vision model, an AI model. Upon receiving a response from the model, the predicted chord is added to the 'setPrediction' and 'receiveChord' functions. The 'setPrediction' function is utilized to update the local state, which displays the predicted chord to the user. Although originally used for development purposes to showcase the predicted chord, the functionality remains in the chord for potential future debugging needs.

#### 5.3.6.3 Chord Display

### Cmaj Variation 1

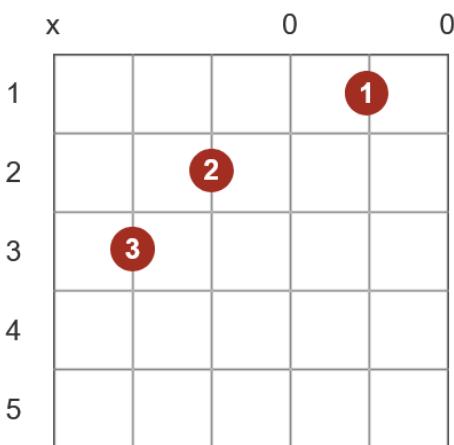


Figure 51: Image of predicted chord

When the predicted chord is passed to the 'receiveChord' function, it triggers a socket.io event and stores the prediction in the Redux store. The 'ChordDisplay' component then automatically updates based on the state of the store, eliminating the need for the 'getState' function. This is achieved through the use of 'useEffect' to monitor the current value of the predicted chord. If a chord is predicted, the corresponding guitar chord image is displayed to the user. However, if the prediction is negative, no chord image will be shown.

### 5.3.6.4 Microsoft's Custom Vision Model

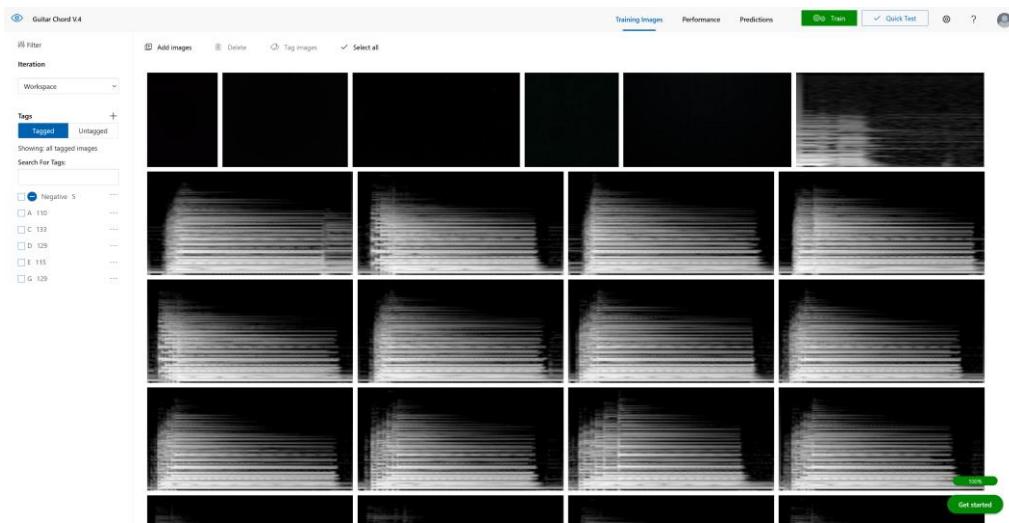


Figure 52: Custom Vision UI

Microsoft Custom Vision is a cloud-based machine learning service that allows users to create and train their own custom image recognition models without requiring extensive machine learning expertise. It is part of Microsoft's Azure Cognitive Services, a suite of APIs and tools that enable developers to build applications with AI capabilities. As previously described in Research section of the report, as audio classification is similar to image classification by converting audio into images, it was possible to use converted audio files converted to spectrograms to train a model.

```
for filename in os.listdir(audio_folder_path):
    if filename.endswith('.wav'): # Only process WAV files
        # Load the audio file
        audio_path = os.path.join(audio_folder_path, filename)
        y, sr = librosa.load(audio_path)

        # Create the spectrogram
        spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

        # Convert the power spectrogram to dB scale
        spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)

        # Save the spectrogram as a PNG with size 300 x 150 pixels
        output_path = os.path.join(output_folder_path, os.path.splitext(filename)[0] + '_spectrogram.png')
        plt.imsave(output_path, spectrogram_db, cmap='gray', dpi=100)
        img = Image.open(output_path)
        img = img.transpose(Image.FLIP_TOP_BOTTOM) # Flip vertically
        img = img.resize((300, 150), Image.ANTIALIAS)
        img.save(output_path)
```

Figure 53: Python Converter

The model was trained using spectrograms generated from guitar recordings in Audacity. The guitar played chords at varying tempos and styles to improve the model's accuracy. The audio files were split into individual chord samples and organized by note in folders. A Python program, shown in Figure 46, was used to convert these audio files into spectrograms of a specific size. Initially, the spectrograms produced by the program did not yield satisfactory results when compared to the

spectrograms from the library used in the application. It was later discovered that the spectrograms had a different orientation, so a line to flip the spectrograms in the Python program was added.

After converting all the audio files, the resulting images were uploaded to Microsoft's Custom Vision platform. Tags were added to each folder of spectrograms during the upload process, enabling the model to classify the images for training. Despite the model claiming high accuracy, its actual performance fell short of expectations. This is because the model was trained on full images of chords, but the application only captures spectrograms at intervals of 4 seconds, potentially resulting in incomplete chord representations. This limitation should be considered for future development. However, the model performs well when there is a continuous audio source in the spectrogram, such as a sustained chord strum.

## 5.4 Development environment

An IDE (Integrated Development Environment) is a software application that provides comprehensive tools and features for software developers to create, edit, compile, run, and debug code in a single, unified environment. VSC (Visual Studio Code) was the IDE selected to develop the application due to its low resource usage and fast performance, while also being free to download and use. VSC has extensibility allowing for easy to install extensions to further improve upon the coding environment and ease some of the minor inconveniences that come with coding. For example, the ‘Prettier’ extension was installed during development, so that when the code is saved it is formatted and indented in a clear and concise fashion. VSC also has a built-in terminal, providing a seamless workflow by being able to execute common development tasks directly from the IDE.

During the testing of the backend server functionality Insomnia was used. Insomnia is a RESTful API client that allows developers to interact with APIs, send HTTP requests, inspect responses, and debug API calls. This allows the developer to check if the server is functioning as intended. For example, the developer can send a HTTP request with login information using Insomnia to test if the server will return a response containing a token needed for the user to be authorised to use the application. It is much easier to test and debug the server functionality using Insomnia before developing the frontend UI, so the developer can be certain that any errors they may discovered during development will not be caused from the server.

Git is a widely used distributed version control system (VCS) that is commonly used for software development. Git allows multiple developers to work collaboratively on a project, tracking changes to files and coordinating updates. It provides a way to manage source code, track revisions, and merge changes made by different developers. Git was used during the development of the application to track changes and development progress. A repository with all of the development code was stored on Github, a web-based hosting service for Git repositories. From this repository, the developer would be able to note progress and plan future development and functionality of the application. A new branch would be made during development when a new feature was being developed in case the new feature would negatively affect the functionality of other components within the application. Once a new feature or component was developed and working without any negative effects, the branch would be merged into the main branch and deleted.

## 5.5 Project Management - Scrum Methodology

Scrum is a popular Agile framework for managing and completing complex projects. It is an iterative and incremental approach to software development that focuses on delivering value to the customer through short development cycles called sprints (Peek, 2023). These sprints are a set amount of time when the development team works on the tasks agreed upon during the sprint planning, with a goal of completing the work by the end of the sprint. While scrum is typically used in a large team with tasks divided out between the members of the team, the methodology was still used with a single developer during the development of the application as a tool to manage the project.

The implementation phase was composed of 7 sprints that started in January, each lasting for 2 weeks. The sprints consisted of a general overall task, such as creating the video call functionality, which is then further subdivided into smaller tasks to achieve the main objective. After each sprint, a meeting with the supervisor was held to evaluate the sprint's success, review completed tasks, and identify tasks that required completion in the following sprint. If something went wrong during a sprint, such as not enough time given to a specific task or development not working out as planned, the supervisor would offer advice on whether to continue with the original plan for the sprint or allow some extra development time for the task at hand. The process was effective in determining important tasks that required completion within a limited timeframe and prioritizing them. A comprehensive list of each sprint's activities is detailed in the follow chapters.

## 5.6 Sprint 1

### 5.6.1 Research & Literature Review.

For the research conducted for this project, it was deemed important to investigate the potential appeal of an application like this to the general public. Studies on the effects of prolonged video call usage during the pandemic and common issues faced by users of video call applications were reviewed. One particularly useful paper was a study conducted by university students in Turkey (Zahal & Ayyıldız, 2022), which explored the experiences of students and lecturers in an online guitar lesson setting at a music college. This study provided valuable insights into the technologies that could be researched to address common problems.

In addition to user experience with video calls, research was also conducted on the technologies utilized in the development of these applications. Based on the findings, React and WebRTC were selected for creating the application, and the understanding of Artificial Intelligence and Machine Learning algorithms was utilized to develop a custom model for integration within the application.

### 5.6.2 Paper Prototype

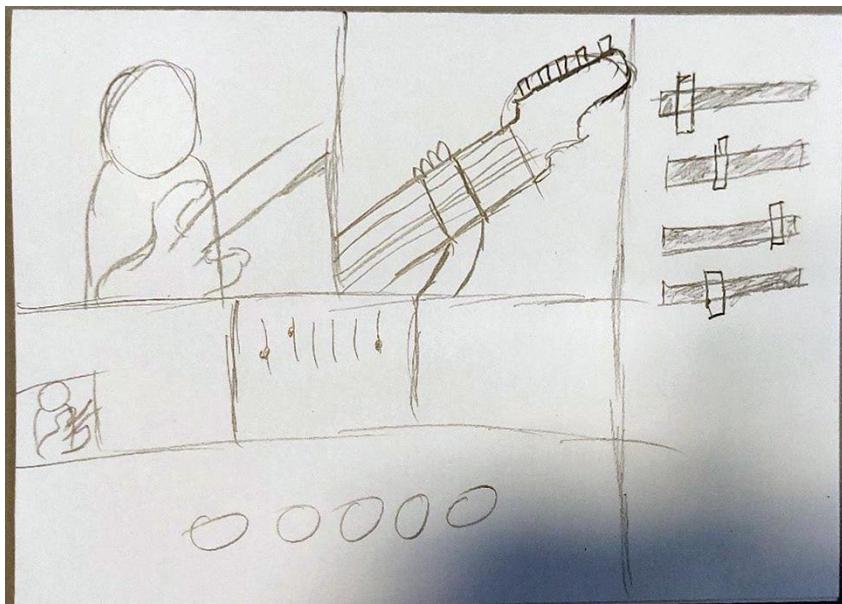


Figure 54: Paper Prototype of video call

To initiate the visualization of the application, preliminary rough designs were created to assess the essential requirements for the application to function. While these drawings were basic in nature, they served as a beneficial exercise to stimulate thought on the necessary features to be incorporated into the application.

## 5.7 Sprint 2

### 5.7.1 Goal

The objective for sprint two was to enhance the visualization of the application by creating wireframes and a high-fidelity (HIFI) prototype. This would aid in the development process by providing a visual reference for creating the necessary components and understanding the UI elements required to facilitate user interaction.

### 5.7.2 Wireframes

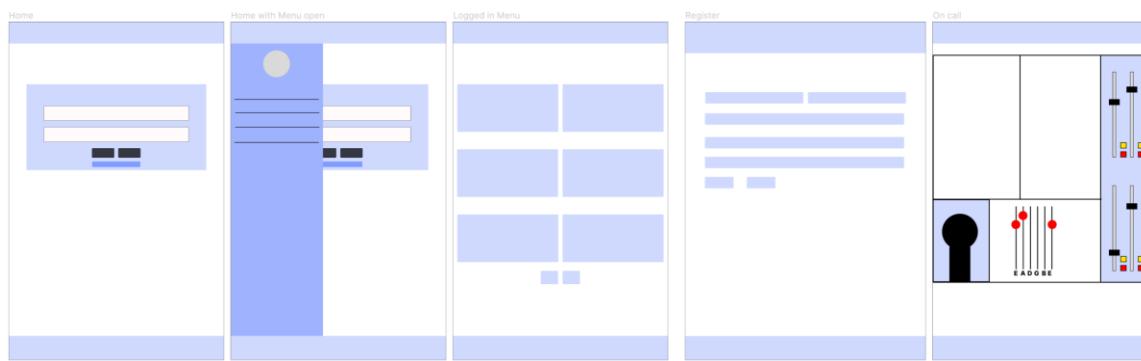


Figure 55: Wireframes

The wireframes, as shown in figure 55, were designed for multiple pages of the application, including login, meeting menu, and video call UI. At this stage of development, it is uncertain whether user profiles will be implemented, as the video call and AI model components are substantial tasks. However, efforts will be made to establish some form of signalling within the application to avoid dependence on third-party apps for establishing connections.

### 5.7.3 Hifi Prototype

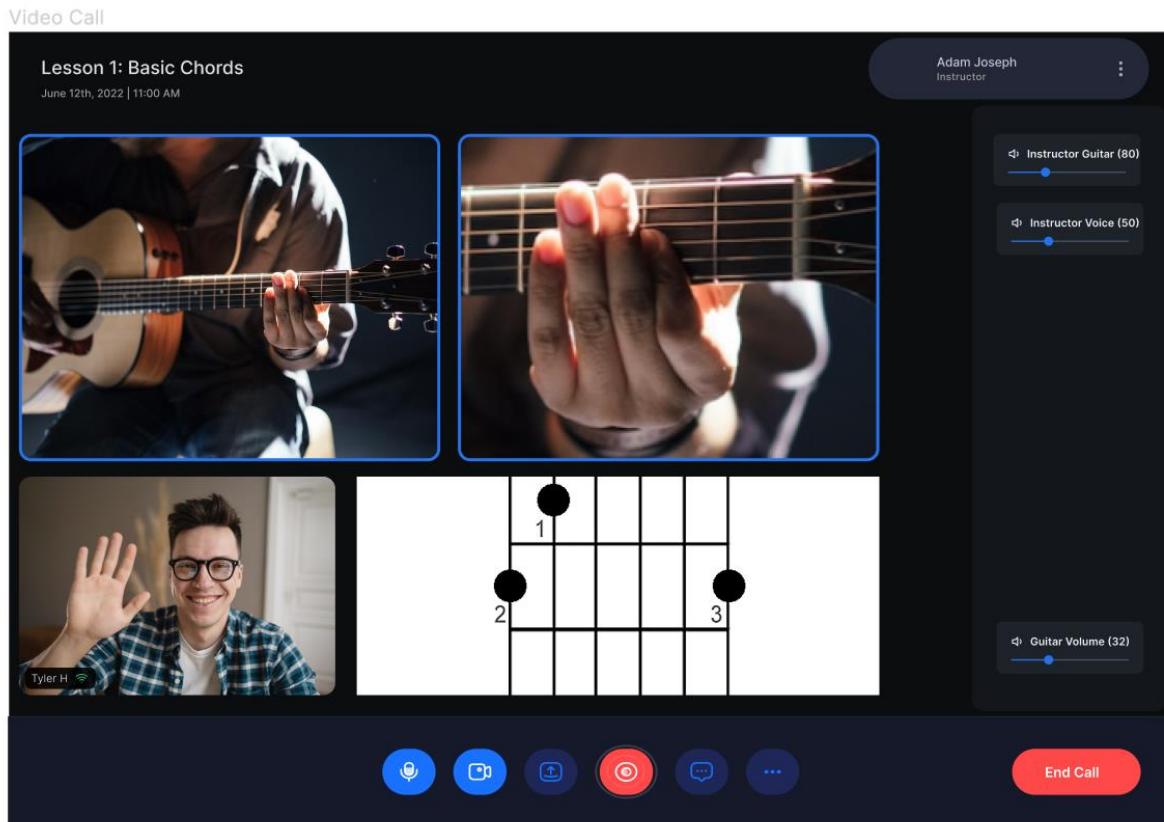


Figure 56: Hifi Prototype

The HIFI prototype depicted in figure 56 was developed using Figma. The prototype closely aligns with the desired appearance of the final application. While there may be some adjustments to colours for improved accessibility, the placement of the cameras, chord display, and volume mixer are positioned optimally according to the intended design.

### 5.7.4 Survey

In this sprint, a survey was developed and distributed among classmates, guitar forums, and LinkedIn to gather feedback. The survey was utilized in the requirements gathering phase of the project to assess the level of interest in an application like this, and to identify common issues faced by users of similar applications. The findings from the survey, combined with independent research, were used to inform the development of the application.

## 5.8 Sprint 3

### 5.8.1 Goal

The goal for the sprint is to initiate the development of the application. The focus is on creating a minimal version of the application with basic functions. This phase of development will help determine the feasibility of different aspects of the application and identify any features that may need to be revised or removed.

### 5.8.2 Basic Video Call Application

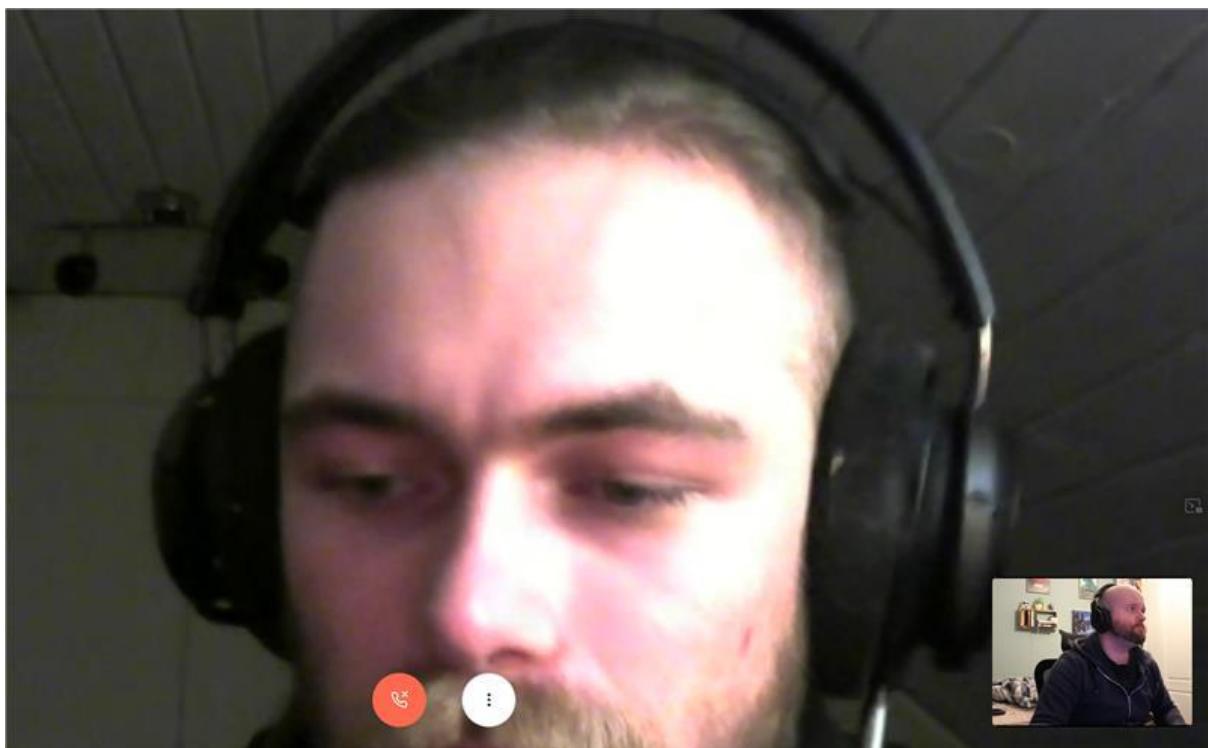


Figure 57: Basic Video Call

During this sprint, a basic video call application was created using React and WebRTC, hosted on Google Firebase with a signalling server. Google STUN servers were utilized to generate the necessary ICE candidates for connecting users. Currently, users can only connect by exchanging a code through a third-party service. Working on this code was challenging and required reliance on tutorials from the WebRTC documentation. However, with the basic prototype functioning it creates a solid base for the full application. Figure 57 shows the basic video call application in action.

```

//Get user's camera and audio from two separate audio input devices, if available
const audioDevices = await navigator.mediaDevices
  .enumerateDevices()
  .then(devices) =>
  devices.filter((device) => device.kind === "audioinput")
);

let audioConstraints;

if (audioDevices.length >= 2) {
  // use two separate audio input devices
  const audio1 = audioDevices[0].deviceId;
  const audio2 = audioDevices[1].deviceId;

  audioConstraints = {
    audio: [{ deviceId: audio1 }, { deviceId: audio2 }],
  };
} else if (audioDevices.length === 1) {
  // use the single available audio input device
  audioConstraints = {
    audio: [
      { deviceId: audioDevices[0].deviceId },
    ],
  };
} else {
  console.error("Could not find any audio input devices.");
  return;
}

```

Figure 58: Code snippet of attempt at a second audio input

During this sprint, efforts were made to enhance the application by adding support for a second audio input. The enumerateDevices function from the WebRTC documentation was utilized to detect multiple audio devices on the user's computer, as shown in Figure 26. However, although the application was able to detect and allow selection of a second audio input, the recipient could not hear both audio sources. Further attempts will be made to rectify this issue, with the goal of allowing users to adjust the volume of each audio input. However, research revealed a library that can mix two audio inputs and send them as a single audio track through the data channel, allowing the recipient to hear both audio sources, but with a shared volume control. While this solution is not ideal, it may be considered as a viable option. It should be noted that this solution may pose challenges in the future, particularly when sending the guitar signal to the hosted model.

### 5.8.3 Hosted Machine Model

During this sprint, research was conducted on hosting the created guitar chord detection model. Various Cloud services were explored, and Google Cloud was chosen due to familiarity with Firebase, which is part of Google's hosting services. However, encountering difficulties, the model was also

attempted to be hosted using Tensorflow.js, which converts the model and its weights into local files for local execution. Despite extensive experimentation, the desired outcome could not be achieved. There is now uncertainty on whether to persist with the current model or explore alternative hosting services.

## 5.9 Sprint 4

### 5.9.1 Web Audio API

To address the issue of implementing the second audio input, a brainstorming session was conducted, leading to the idea of sharing the user's screen audio as a potential solution. Research indicated that this could be a more feasible option. However, it was noted that there needs to be audio playing on the user's desktop for it to be audible during the call. As a first step towards this solution, an "amp" component was created, which seeks permission to access the user's microphone or audio signal and allows the user to adjust various parameters of the signal. The Web Audio API was utilized to build this component, as it is natively supported in modern browsers, without the need for external libraries to be downloaded, similar to WebRTC.

```
// Getting the guitar input
const setupGuitar = () => {
  return navigator.mediaDevices.getUserMedia({
    audio: {
      echoCancellation: false,
      autoGainControl: false,
      noiseSuppression: false,
      latency: 0,
    },
  });
};

const setupContext = async () => {
  const guitar = await setupGuitar();
  if (context.state === "suspended") {
    await context.resume();
  }
  const source = context.createMediaStreamSource(guitar);
  source.connect(gainNode);
  source.connect(bassEQ);
  source.connect(midEQ);
  source.connect(trebleEQ);
};
```

Figure 59: Code snippet of setting up guitar context.

The implementation of the component begins with the prompt for the user to select their audio input device. Following the user's device selection, an audio context is created to handle audio processing and node creation. The audio input is used to create a media stream source, which is then assigned to the context for playback to the user. The context can be modified to include effects and parameter adjustments, such as volume, bass, middle, and treble EQ settings, allowing the user to customize the sound of their instrument.

## 5.9.2 Screen Sharing Audio

After successfully implementing the amp component, the next step was to create a screen sharing function that would allow the audio from the user's desktop to be added to the peer connection for others to hear.

```
//Get audio from user's desktop
const desktopStream = await navigator.mediaDevices.getDisplayMedia({
  video: true,
  audio: true,
});
```

Figure 60: Code snippet of retrieving user's desktop audio.

This functionality was easily achieved using the built-in 'getDisplayMedia' function, which retrieves the desktop video and audio once approved by the user and sources are selected. The retrieved audio and video were then added to the peer connection. However, during testing, it was discovered that while this functionality worked fine in Chrome, it did not work in Firefox due to Firefox not supporting audio sharing when screens are shared in the browser. As a result, the decision was made to primarily focus on using Chrome.

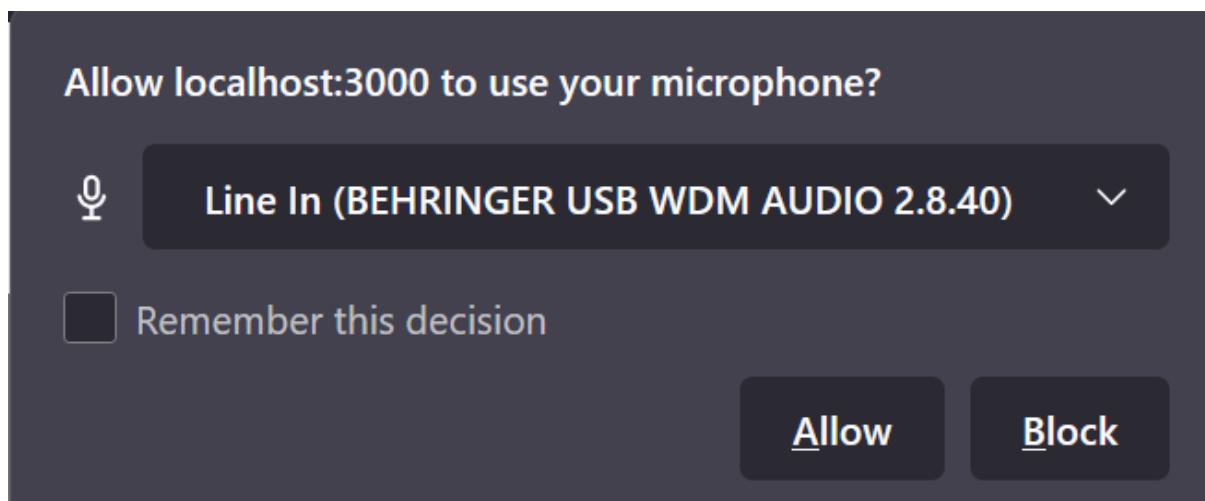


Figure 61: Firefox input prompt

With the screen audio sharing now functional, the next step was to combine it with the amp component so that users could hear both audio signals from each other. However, using Chrome posed another challenge as it does not allow more than one audio input to be used simultaneously. Unlike Firefox, which prompts the user to select an audio input from a list, Chrome automatically

uses the default audio device, and only one device can be set in the settings. Despite researching for a possible workaround, it was determined that overriding this setting was not feasible.

### 5.9.3 Dual Audio Signals

Unable to get the screen sharing audio functioning, the integration of two audio sources into the peer connection had to be reconsidered. Various npm libraries were attempted, including one recommended by WebRTC that involved mixing the two audio signals into one before adding them to the peer connection, but all of them were found to be deprecated. Based on research, it appeared feasible to add two different audio sources to the peer connection, prompting a decision to retry the original plan.

```
//Function to set up video stream
const setupSources = async () => {
    //Get user's camera and microphone
    const localStream = await navigator.mediaDevices.getUserMedia({
        video: true,
        audio: true,
    });

    var ctx = new AudioContext();
    var source = ctx.createMediaStreamSource(guitarStream);
    var gainNode = ctx.createGain();
    gainNode.gain.value = 0.5;
    source.connect(gainNode);
    source.connect(ctx.destination);

    // create a new MediaStream object
    const newStream = new MediaStream();

    // add the guitarStream to the new MediaStream object
    guitarStream.getAudioTracks().forEach((track) => {
        newStream.addTrack(track);
    });
}
```

Figure 62: Setting up dual audio inputs

With the microphone audio input functioning, instead of retrieving another device, which may cause issues in the browser by having to use the ‘getUserMedia’ function twice in a row, a decision was made to set up the guitar input in a similar manner as the ‘Amp’ component. Upon loading the page, the browser would ask for permission to use the audio input for the guitar. When the user pressed ‘start’ to initiate the stream, the guitar input would be placed into an audio context and added to the peer connection alongside the microphone audio input. This approach proved successful, enabling each user to hear each other’s instrument and microphone input simultaneously. However, it should

be noted that this application would not be compatible with Chrome, as there was no option to select different audio input devices.

#### 5.9.4 Microsoft Azure Custom Vision

After encountering difficulties hosting a personal model, alternative options were sought for the AI model. Microsoft Azure's Custom Vision was chosen due to its provision of \$100 free credit for students. A successful creation of an image recognition model was achieved using Custom Vision.

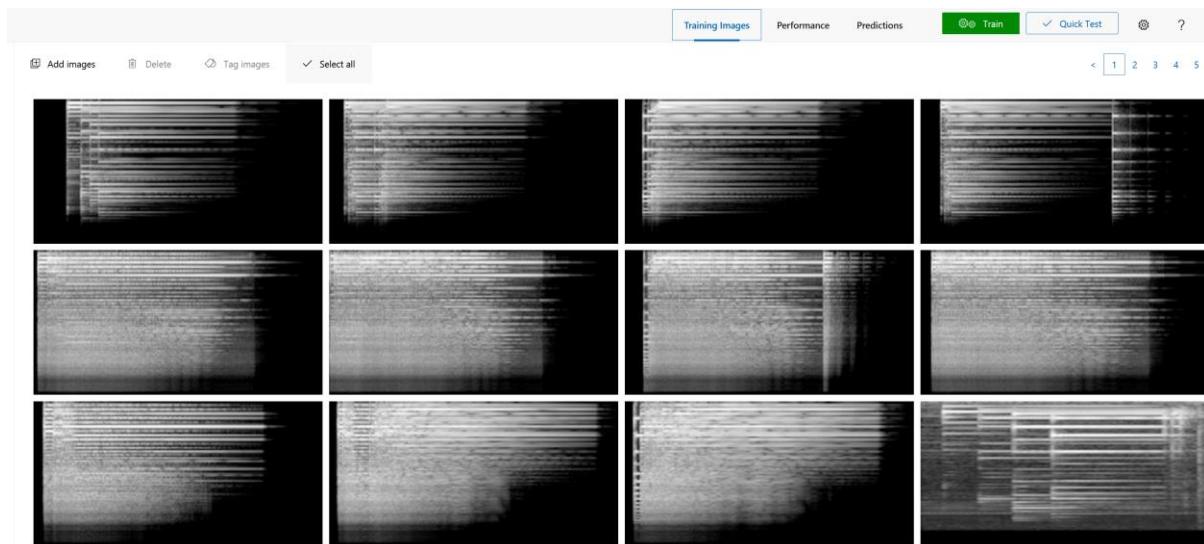
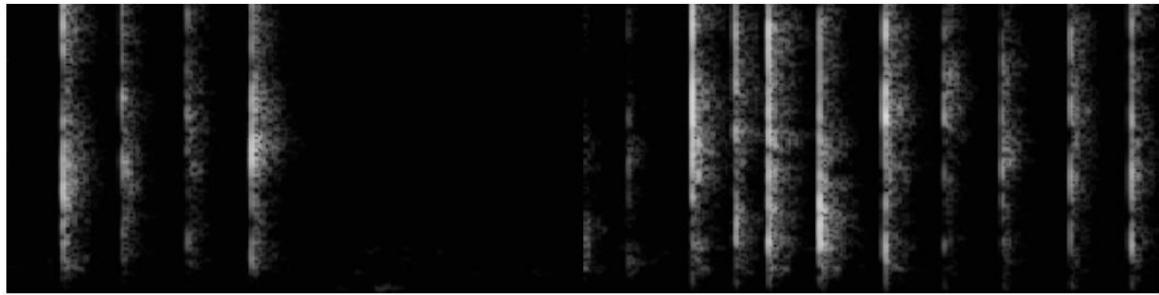


Figure 63: Images used to train model in Custom Vision.

A Python program was developed to train the model, which involved converting the audio files from the previous model into spectrograms. Approximately 200 images were generated for each chord after the conversion of all audio files. These images were then uploaded to Custom Vision with appropriate tags for each chord. Paid advanced training was utilized with some of the available free credits to train the model. Once trained, the model was tested using spectrograms that were not used during training. Although the model was found to be less accurate than the Python-written one, it was hosted and provided an API endpoint for easy browser requests. To further test the model, a simple HTML program was created to allow users to upload an image and receive a response from the model.

#### 5.9.5 Guitar Spectrogram Generation

Now that a model had been trained on spectrograms, a method of generating spectrograms from an audio source needed to be developed. To generate spectrograms from live audio signals, the npm spectrogram library was utilized. This library allowed for the conversion of an audio context into a continuous spectrogram, which could be displayed to the user as needed.



*Figure 64: Spectrogram generated from microphone.*

A screenshot function was created to capture the live stream of the spectrogram every 10 seconds, which was then sent to the model for analysis and a corresponding value was returned. The screenshot function was adjusted as needed to improve its functionality. However, the model was not trained to detect the absence of chords, leading to predictions even when no chords were present. While the program functioned correctly in terms of generating spectrograms from guitar audio, sending them to the model for analysis, and returning a value, further training of the model and adjustments to the parameters used for spectrogram generation may be necessary. Nonetheless, this effort provides a strong proof of concept.

## 5.10 Sprint 5

### 5.10.1 Interim Presentation

The sprint began with an overview presentation of the application to the supervisor and second reader, including an update on the progress of development. The presentation included information from the report, as well as a live demonstration of the application. Currently, the components of the application are only working separately and not functioning as a prototype. Feedback from the presentation highlighted the need for a more user-friendly UI, including the ability for users to create their own profiles and call each other within the app without having to use a third-party app to copy and paste an ID.

### 5.10.2 Backend Server

Prior to developing the frontend UI for the website, the decision was made to create a backend server to handle the application's functionality. The main objectives included implementing CRUD functionality for user profiles, as well as for lessons/rooms.

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();
const port = 3000;
require("dotenv").config();

require("./utils/db.js")();

app.use(express.json());
app.use(express.static("public"));

app.use((req, res, next) => {
  if (req.headers?.authorization?.split(" ")[0] === "Bearer") {
    jwt.verify(
      req.headers.authorization.split(" ")[1],
      process.env.APP_KEY,
      (err, decoded) => {
        if (err) err.user = undefined;

        req.user = decoded;
        next();
      }
    );
  } else {
    req.user = undefined;
    next();
  }
});
```

Figure 65: Backend Server Code

The development process started with the creation of the server.js file, utilizing Express and Node to handle backend events. This file facilitated the connection of the server to the Mongo database. Next, models were created to define the properties of objects on the backend. Two models were

implemented, one for lessons and one for users, with the possibility of additional models in the future. Following the creation of the models, controllers were developed to handle CRUD functionality for both models, enabling users to create, update, or delete profiles, lessons, or rooms. Once the controllers were established, routes were created and added to the server file. Finally, thorough testing of all endpoints was conducted using tools such as Insomnia, to ensure proper sending and receiving of information with the Mongo database, while running the server locally.

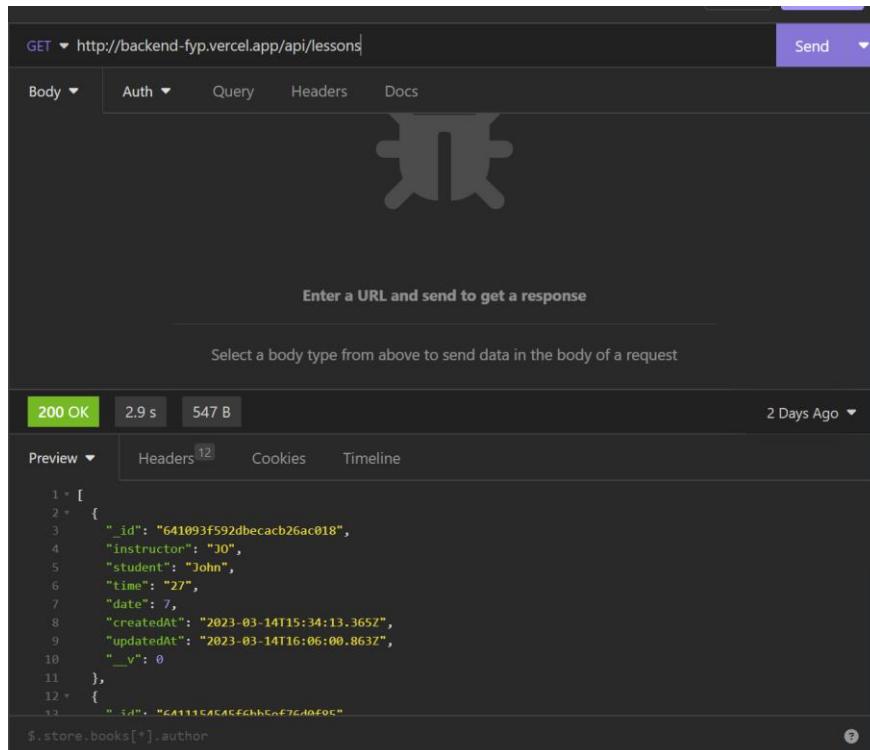


Figure 66: Testing hosted endpoint

After addressing and resolving any issues, and ensuring that all endpoints were functioning as intended, the backend was uploaded to Vercel for remote hosting. Further testing of the endpoints was conducted using the hosted URLs provided by Vercel and verified using Insomnia to confirm proper functionality.

### 5.10.3 Frontend UI

Once the backend was confirmed to be functioning as intended, the development process shifted to building the frontend UI for the application. To minimize potential errors during the initial building process, a new React application was created. Given that the previously developed functional components were also built with React, copying these components to the new UI once satisfied with the UI design was straightforward.

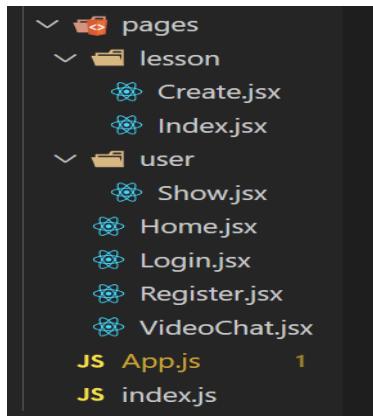


Figure 67: Initial Front End Pages

The initial steps in building the frontend UI involved downloading the necessary libraries, such as Axios for making server requests and React-Router-DOM for handling application navigation. Additionally, several Material-UI (mui) packages were downloaded to provide styling options. All the required components/pages were imported, even though they were not yet created, as a visual aid to outline the needed components for the frontend. Empty components were then created for each page, serving as placeholders to establish the necessary routes for navigation.

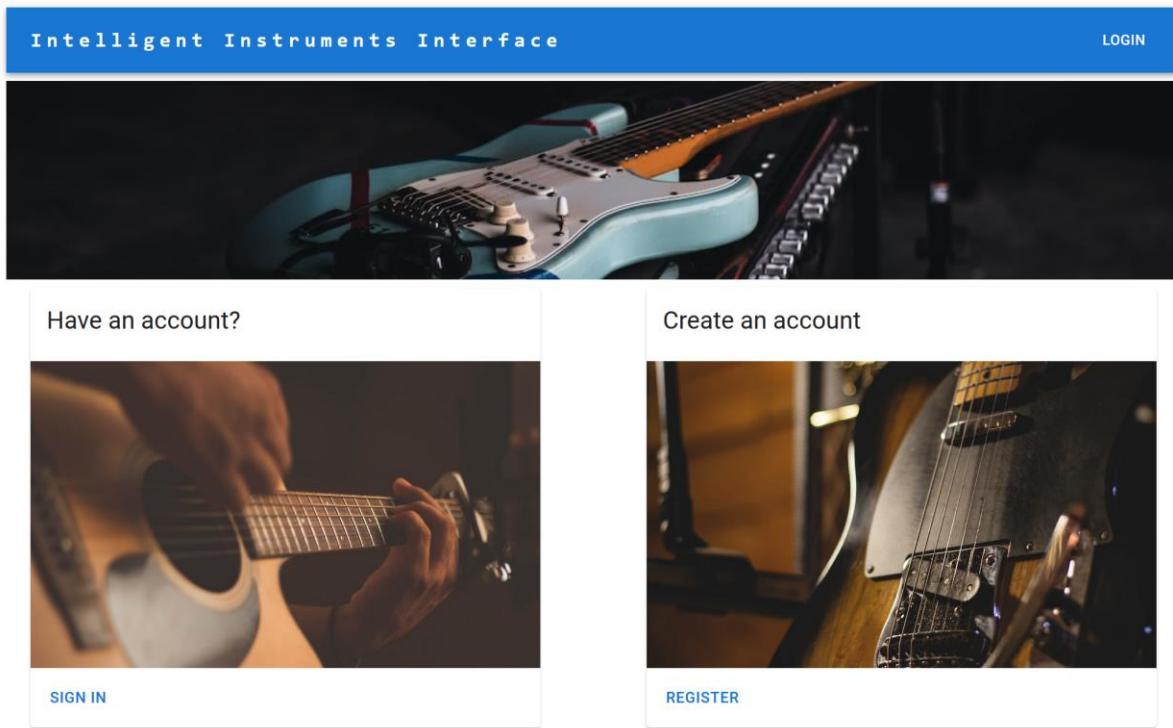


Figure 68: Application Homepage

Components/pages were created with a focus on functionality and a simple layout. The home page directs users to sign in or register. Upon registration, users can choose between a student or instructor account. Registered user details are stored in the mongo database for login. After login,

users are taken to a profile page. Future plans include adding functionality for users to edit or delete their accounts, though not currently required. Additionally, functionality was added to detect whether a user is an instructor or student, and if they are an instructor, they can view all lessons and create new lessons. Lessons include the instructor's name, student's name, and the time and date. In the future, the plan is to implement a room ID for lessons that both users can use for conducting lessons.

#### 5.10.4 Application Restructure

The application's video functionality has a working prototype, but it requires restructuring for improved user-friendliness. Currently, users need to share a link via a third-party application to initiate the data stream. The goal is to integrate this process within the application itself using socket.io for handling the signalling between clients. The plan is to create a room where users can join, instead of establishing a direct call between two users. Although the room could technically accommodate more than two users, it will be intended for only two users. This approach will allow users to re-join the call if they get disconnected for any reason, instead of the current behaviour where the application stops running when the peer connection is lost.

During research on socket.io, PeerJs was discovered as a WebRTC framework that simplifies and improves efficiency. Research and learning about this framework are currently underway, with the possibility of using it for rebuilding the application's functionality. However, the restructuring of the application has not been completed yet, as the technologies are still being learned.

## 5.11 Sprint 6

### 5.11.1 Backend Server

In the last sprint, the focus was on rebuilding the application with a new structure. A basic backend Node/Express server was set up, including the folder structure, necessary packages such as Express, Socket.io, and Bcrypt.js, and routes for functionality. Testing was done using Insomnia to verify the routes. Models/schemas were created for the application, and basic user authentication was implemented, including register and login functions. The validation process was made simpler with the use of Joi package. Token generation was also implemented using the JsonWebToken package, and testing was done using Insomnia to ensure token generation was working. Middleware functionality was added to validate the token stored in the user's local storage, enhancing the security of the application. This would require users to sign in again if the token was invalid or expired.

```
const registerSchema = Joi.object({
  username: Joi.string().min(3).max(12).required(),
  password: Joi.string().min(4).max(12).required(),
  email: Joi.string().email().required(),
});

const loginSchema = Joi.object({
  password: Joi.string().min(4).max(12).required(),
  email: Joi.string().email().required(),
});
```

Figure 69: Validation using Joi Package

### 5.11.2 React Redux

After setting up the basic backend server, the focus shifted to the frontend. React Redux was chosen for better state management and easy state sharing between components without heavy reliance on props from parent components. To configure Redux in the frontend, a store folder was created with necessary components, including the store component which holds all the states for the application. Folders for Redux actions and reducers were also created. In Redux, actions are used to change the state, and reducers take in the action and previous state to return a new instance of state. Once

Redux was configured, routes for the application were created to define the different pages and functionalities that users would interact with.

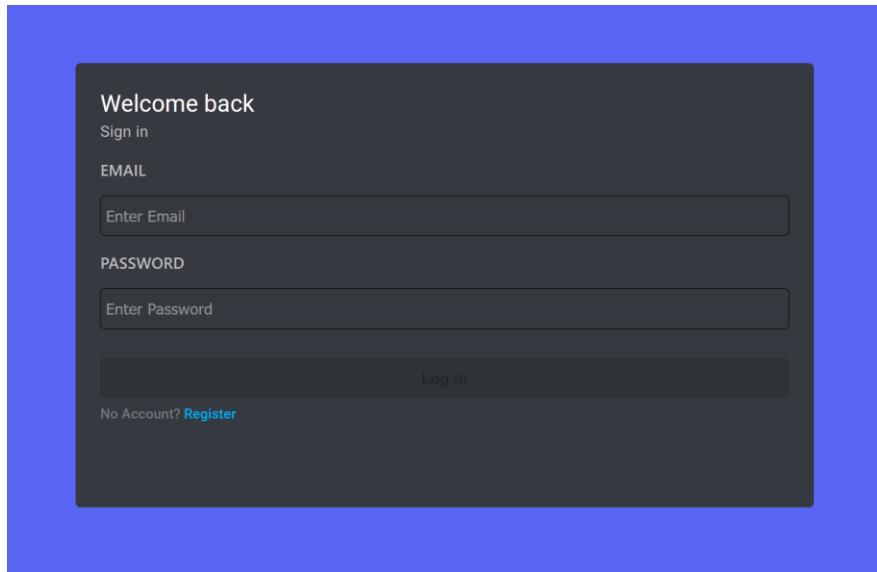


Figure 70: Original Login Page

The decision was made to simplify the application scope by keeping only three pages: login, register, and a dashboard page that would contain all relevant user information. Basic login and register pages were created with styling inspired by the Discord application for ease of testing. Functionality was added to allow users to register and login, and upon successful completion, a token would be generated for the user.

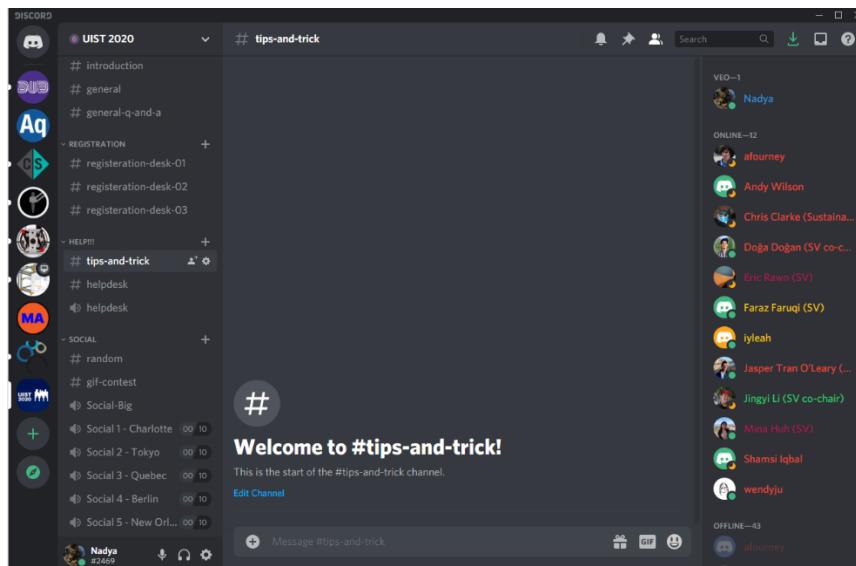


Figure 71: Discord UI

### 5.11.3 Dashboard UI

After ensuring that the register and login pages were functioning, the focus was shifted to working on the dashboard's UI. Inspiration was drawn from the UI of similar applications, specifically Discord, to determine the layout of features in the dashboard. Sections were created to house the various components. A component for rooms/lessons was placed on the left, followed by a user's friends list as there were no channels in the application like in Discord. A header component was created using Material UI's 'AppBar' component. Dummy data was added to the friends list component to work on styling for usernames and an online indicator component. An 'Add Friend' button, invitation list, and logout button were also added to the 'AppBar'. With basic UI styling in place, focus shifted to implementing functionality for the dashboard.

### 5.11.4 Socket.io

The first functionality incorporated into the application was the ability for users to send friend requests to each other. Upon becoming friends, users would be able to send direct messages and see if the other user created a room/lesson and join that room. To achieve this, socket.io was integrated into the application. On the backend server, socket.io was installed and a JavaScript file was created to handle socket.io functionality. Basic emitting events, such as logging the socket id when a user connected, were implemented for testing purposes. Subsequently, a JavaScript file was created on the frontend to handle socket events and tested the connection to the server. Once the frontend was able to communicate with the server through socket.io, the user's token was attached to socket requests for verification before completing the request.

```
const verifyTokenSocket = (socket, next) => {
    const token = socket.handshake.auth?.token;

    try {
        const decoded = jwt.verify(token, config.TOKEN_KEY);
        socket.user = decoded;
    } catch (err) {
        const socketError = new Error("NOT_AUTHORIZED");
        return next(socketError);
    }

    next();
};
```

Figure 72: Function to Verify Token

Upon establishing a successful connection, focus shifted to developing functionality for storing information about connected users on the server. A server store component was created on the backend to handle this functionality, including declaration of new user connections. Additionally, a socket handler folder was created to hold functionality for each of the socket objects. A new connections handler was developed within this folder to manage user details when they connect to the server, including passing their socket id and profile information. Once the new connection handler was created, a handler for user disconnections was also added.

```
const reducer = (state = initState, action) => {
  switch (action.type) {
    case friendsActions.SET_PENDING_FRIENDS_INVITATIONS:
      return {
        ...state,
        pendingFriendsInvitations: action.pendingFriendsInvitations,
      };
    case friendsActions.SET_FRIENDS:
      return {
        ...state,
        friends: action.friends,
      };
    case friendsActions.SET_ONLINE_USERS:
      return {
        ...state,
        onlineUsers: action.onlineUsers,
      };
    default:
      return state;
  }
};
```

Figure 73: Friends Reducer

The functionality to allow users to send friend invitations was then implemented. A friends reducer was created in the reducers folder to manage the state of the user's friends, pending friend invitations, and online friends. An action file for friends was also created to handle functions related to managing the user's friends, such as sending invitations and checking online status. Routes and models were created to support friend invitations, and the functionality was tested in the mongo database. Subsequently, a socket.io event listener was created on both the client and server sides to handle when a user invites another user to become friends. These socket.io events enabled real-time updates to the friends invitation list.

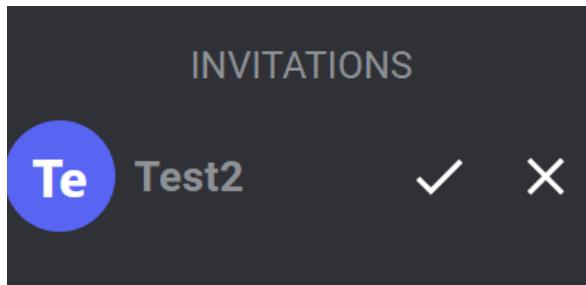


Figure 74: Testing Friend Invitations

Once the user's invitations were successfully displayed on the dashboard, the functionality to accept or reject friend requests was implemented. The frontend functionality was incorporated into the existing actions and reducers, while on the backend, a controller folder was created to specifically handle backend functionality. Each user object on the backend now includes a friend's property, which contains an array of other user's IDs if they are friends. When a user accepts a friend request, a function was created to update both users' information on the backend and add each other to their respective friends' arrays. If the request is rejected, it is simply deleted from the system.

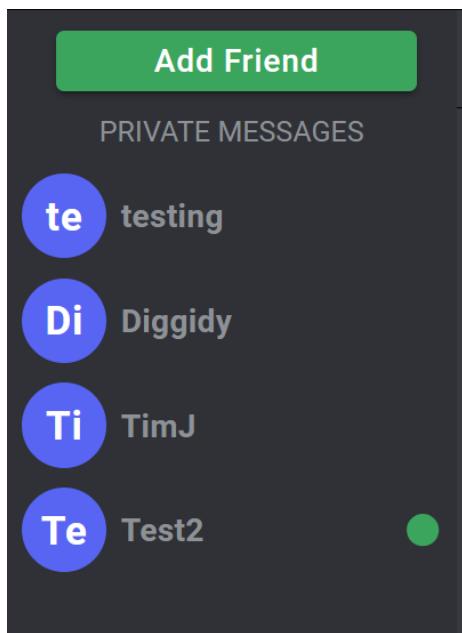


Figure 75: Working Online Status

Now that users could become friends the functionality to allow the user to see when their friends are online had to be implemented. A socket event listener was added to get the friends' data from the database. Necessary action and reducers were created for the friends' state in the frontend. The friends' list from the server was mapped through to display the friends' data on the screen. Once the friends' list was displayed, the functionality was implemented to show which friends are currently online. In the server store, a function was created that takes in a user's ID and returns whether the

user is online. This function was used in a socket.io emitting event that runs periodically and contains all the users currently online. On the frontend, this socket.io event was utilized to loop through the user's friends and compare the list to the online users, indicating which of the user's friends are currently online.

#### 5.11.5 Realtime Chat

While gaining a better understanding of socket.io, the decision was made to implement realtime chat functionality to the application for instructors to interact with students outside of lessons and organize future lessons. To implement this functionality, new files were created for chat actions and reducers, including initial states for chat types (direct to another user or a group message), the message content, and the chosen chat details (e.g., recipient information). Although group chat functionality was not initially planned, some basic functionality was implemented in case it is needed in the future. A function was created to pass the Id and name of the chosen friend for messaging, which was then applied to the friends list so that the Id would come from the friend clicked by the user. A basic messenger UI was created to allow users to type in an input field, and a component was developed to store all the messages exchanged between users, using dummy data for rendering purposes.

```
const Schema = mongoose.Schema;

const messageSchema = new Schema({
  author: {
    type: Schema.Types.ObjectId,
    ref: "User",
  },
  content: { type: String },
  date: { type: Date },
  type: { type: String },
});
```

Figure 76: Message model

After finalizing the UI, the focus shifted towards implementing the functionality of the messages. A socket event was created to emit when a direct message is sent, including the recipient and the content of the message. Similar to the connection handler, a server handler was created for direct messages, along with a model for messages exchanged between users. A socket.io handler was then developed to utilize this handler upon receiving the direct message event from the frontend. With the ability to technically send messages between users, real-time chat updates needed to be added.

A function was created in the socket handlers folder to take in the conversation Id and populate the messages from the conversation. Additionally, a new handler was created for chat history, which would search for a conversation based on the IDs of the two users. If a conversation containing the IDs of the two users was found, the previous function to populate messages would be utilized.

#### 5.11.6 Video Chat

Similar to the previous version of the application, the video chat functionality is handled by WebRTC. The implementation involves creating the necessary state variables and functions in the reducer and actions folder for the components. A UI button is then created to allow the user to create a room. In the room actions, a function is created to create a room when the button is pressed, set the user as the creator of the room, and put them in the room. A separate component is created for the UI of the room, which initially appears minimized in the corner of the dashboard when a user creates a room. Functionality is added to resize the room component to full screen, and UI buttons are created to control video functionality, such as muting the user's microphone or leaving the call.

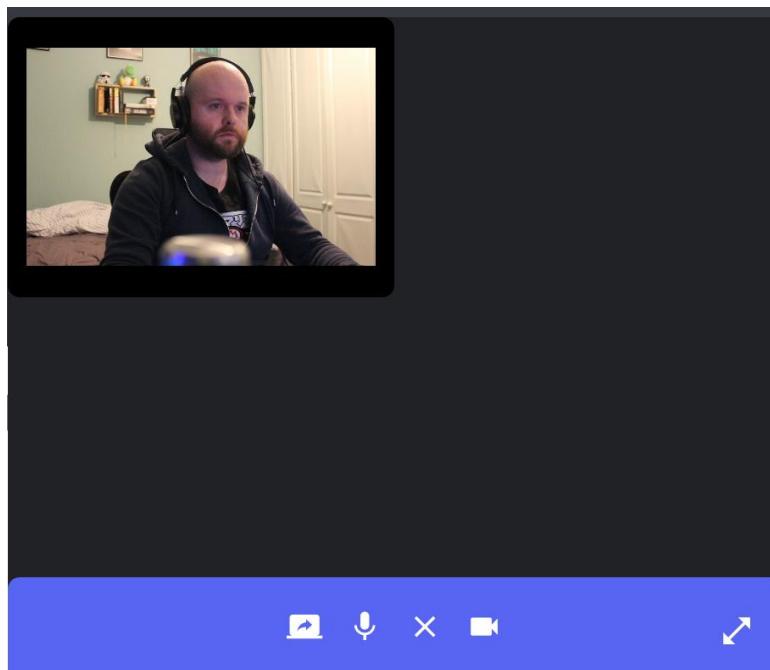


Figure 77: Room view upon creation

The server functionality of the room is implemented by creating a room handler file that takes in the user's Id and socket Id and uses socket.io to emit these details. In the server store, an 'add new room' function is created, which contains the creator and participants of the room, and uses a library called UUID to generate unique IDs for each created room. A listener is then created on the socket server to run the handler once the room is created on the front end.

After a room is created, the details of the active room are sent to other users who are friends with the user who created the room. This is implemented by creating a function to get all active rooms on the server, and using this function to check if the creator of the room is in the friends array of the user. If the users are friends, the room is displayed to that user. Handlers are also created to check if the number of participants in the room is less than one, and if so, remove the room from the active rooms in the server.

To add a video stream to the active room, access to the user's local stream is required, including permissions to use the user's video and microphone input. This process is similar to previous iterations of the application, but this time, since Redux is used, the local stream has reducers and actions for the state. A WebRTC handler file is created on the front end to handle getting the user's devices and dispatching them to the backend server. React ref is used to assign the user's local stream to the video preview in the room component. Additional functionality such as allowing the user to join with audio only is added, and the local stream is stopped when the user leaves the room.

To handle the WebRTC connection between users, the simple-peer library is used to simplify implementation. To implement this, an event is emitted whenever a new user joins the active room, containing the details of the user joining the room. These details are then used to create a peer connection between the user that joined and any users already in the room. As this application uses a mesh architecture, every user creates a peer connection with every other user in the room, instead of sending their video and audio data over a server. This limits the number of users allowed in a room, but this architecture is chosen because in previous versions of the application that only contain direct peer connections, if anything happens to the connection, the call ends for both users. Using rooms allows the user to join the call again if anything happens to the connection.

#### [5.11.7 Adding Components from previous versions](#)

With the ability for users to create rooms and join calls with each other, working components from previous versions of the application were implemented. The first component was the guitar amp, which allowed users to generate sound with an additional audio input. Redux made it easier to take the state of the guitar stream and pass it to the local stream for other users in the call to hear.

Next, the audio spectrogram converter was implemented, again leveraging Redux states for seamless integration. A component was added to send the spectrogram to the AI model. Currently, the first request to the model works, but the second request returns an error. It's unclear if this is due to the spectrogram screenshot not updating properly due to props. An attempt will be made to combine the two components into one to resolve this issue.

## 5.12 Sprint 7

### 5.12.1 Combining Chord Detection & Guitar to Spectrogram Components

As mentioned in the previous sprint, the 'ChordDetect' component and 'GuitarToSpec' component were partially functional but encountered issues after the initial screenshot was sent to the model. It was suspected that the issue was related to the screenshot prop not updating with the new screenshot after it had been set. To resolve this problem, the two components were combined into one. A submit function was implemented to take the last screenshot created and send it to the Microsoft Custom Vision model. Once a response was retrieved, it would be displayed to the user as a state, which could easily be updated with each new result from the model. The function was called inside a useEffect hook, which would wait for the 'isRunning' prop to be set to true. 'isRunning' was passed down as a prop when the user turns on the 'Amp' component. This would start the spectrogram canvas displaying the converted guitar signal, which now comes from the redux store. A screenshot would then be passed into the submit function at intervals of a few seconds for continuous processing.

### 5.12.2 Displaying Chord in Video Call

During testing, it was observed that the chord prediction was initially displayed in the 'Amp' component. However, to allow students to view the chord, it was necessary to display it in the 'Room' component. Consequently, a decision was made to develop a chord state in the redux store that would be updated whenever a new response was received from the model. Socket io was used to update the state by transmitting data to the backend server and then back to the client. The reason for transmitting the predicted chord to the server was to enable it to be stored in redux state and displayed locally, as well as emitted to other users in the lesson.

To display the chord charts, a new 'ChordDisplay' component was created, which imported images containing a chart of finger position for multiple chords. The connect method from Redux was utilized to connect the state to the component, making it possible to detect changes in the state without the need for a 'getState' function. An useEffect was implemented to monitor changes in the 'guitarChord' prop, which contained the chord prediction from the redux store. When a guitar chord was detected, the corresponding chord chart image was displayed on the screen. The 'ChordDetect' was ultimately displayed in the 'Room' component to enable both users in the room to see the chord being played by the instructor.

### 5.12.3 Amp Component UI

The application's functionality was complete, but the user interface needed further development for better presentation and clarity. The 'Amp' component lacked any style and had basic sliders to control parameters, so it required a redesign to make its functions more apparent. To achieve this, the component was designed to resemble a guitar amplifier from the well-known and largest guitar amplification manufacturers, Marshall. This design was chosen as it would be recognizable to most guitar players, including beginners, who may have encountered such amplifiers at some point.

The design process began with creating an amp container that utilized an image of the grille cloth material commonly found on guitar amplifiers. The Marshall font was downloaded to display the application name on the component, as would be seen on a real Marshall amplifier. A container was then created for the controls to adjust the various parameters of the guitar audio being played back to the user. Another container was made for the on/off switch, which was styled to resemble a standard red rectangle switch found on an amplifier, with slight changes in its appearance to indicate whether the amp was switched on or off.

The 'RangeInput' components were already functional, but they required restyling to match the new aesthetic of the component, as they were just standard input sliders. The component was redesigned as a circular knob, and the values were assigned to match the state they represented. For example, 360 degrees would correspond to 100%, and 0 degrees would represent 0%. To provide a more accurate representation, the starting position was marked at 7 o'clock, and the end position at 4 o'clock, as amplifier knobs typically do not rotate a full 360 degrees. A marker was placed inside the knob to show the user the parameter's value, and JavaScript mouse events were added to allow the user to adjust the parameter values by moving the mouse from left to right. The changed value would then be passed back up the 'Amp' component.

### 5.12.4 User Testing – Video Call Functionality

During the sprint, user testing commenced to evaluate the functional capabilities and usability of the application. A task list was formulated, consisting of the following questions:

1. Can you create an account?
2. Can you turn on the Amplifier to hear your instrument and adjust its volume?
3. Can you add a friend using the provided email?
4. Can you send a message to that friend?

5. If you are an instructor, can you edit the call settings?
6. If you are an instructor, can you create a room? If you are a student, can you join the available room?
7. Can you join and leave the call?

After each task, candidates were given an opportunity to provide feedback and rate the difficulty of each task. The overall testing results were positive in terms of functionality. Most users were able to create an account, hear their instrument being played back, add a friend, send a message, and join the room. However, some users initially encountered issues when choosing a second audio source for a microphone. It was later discovered that they were not using the Firefox browser.

The main feedback from the user testing was focused on the appearance of the buttons used to select a user for direct messaging and to join a room. Users reported that all buttons were the same colour, making it difficult to distinguish between interactive and non-interactive components. Further feedback was provided regarding the UI for the 'Amp' component, particularly the small size of the knobs for adjusting parameters, which made it challenging to make slight changes. There was also feedback for the layout of 'Room' component, feeling there was too much open space between the cameras and the chord chart.

## 5.13 Sprint 8

### 5.13.1 User Testing - Chord Detection

During this sprint the chord detection functionality was tested. It was decided to test this functionality separately from the video call functionality to avoid overwhelming candidates with too much information in a single session. While the testing video call testing had the user performing tasks, this testing would put the user in a more passive role, simply observing the functionality of the chord detection model, and providing feedback. The candidate would sit through a five-minute presentation of the chord detection model functioning in the application and when then given a series of statements to rate according to how well they felt the model functioned. These statements were as follow:

1. The chord being played by the instructor was clearly displayed.
2. The image of the chord was a good representation of how to play the chord.
3. I had enough time to view the chord to copy what the instructor was playing.
4. The placement of the chord was a good position to see the chord and the instructor.
5. The chord being shown was accurate to what the instructor was playing.
6. I believe this feature could help in an online teaching environment.

The candidates were also presented with the opportunity to give additional feedback. The overall feedback from the results of the testing were positive. All users felt that the implementation of these technology would improve the quality of a real-life lesson. There were no criticisms, mainly referring the size and positions of the chord chart being displayed to the user, but these criticisms can be easily addressed.

### 5.13.2 Emit Only to Users in the Room

In the application's current state, the instructor is able to turn on the 'Amp' component, which in turn will start the 'ChordDetect' component. Once the user plays a chord, a spectrogram will be taken from the guitars signal, sent to the Custom Vision model, then the prediction will be returned to the user. This prediction is then emitted through socket IO to all users, who can see the chord being displayed if they are in a room. While this is working as intended, the main problem is that the chord is emitted to every user, whether or not they are friends with the instructor who emitted the chord or are in a room with the instructor. This effectively will make two instructors unable to emit chords at the same time as they will interfere with one another, as both users will be emitting chords. To rectify this issue, the chords emitted from the instructor must only be emitted to users who are in the same room as the instructor.

```
export const receiveChord = (data, remoteUsers) => {
  socket.emit("receiveChord", { data, remoteUsers });
};
```

Figure 78: Receive Chord Function

The ‘receiveChord’ function was modified by updating it to include a variable that retrieves the state of ‘remoteStreams’. This state contains information about all users connected to the current user via a peer-to-peer connection. By iterating through the ‘remoteStreams’, the socket Id of each user was retrieved and passed as a parameter to the ‘receiveChord’ function, which is responsible for emitting chord predictions to the backend server. The predicted chord and the socket Ids of users in a video call with the instructor were then emitted by the function. Figure 78 illustrates this process.

```
const roomChordHandler = (socket, data, remoteUsers) => {
  remoteUsers.forEach((remoteUser) => {
    socket.to(remoteUser).emit("sendChord", data);
  });
  socket.emit("sendChord", data);
};
```

Figure 79: Room Chord Handler

A new handler for chords, ‘roomChordHandler’ was created on the backend server, as shown in figure 79. The handler function receives data containing the predicted chord and socket Ids of users. It then iterates through each socket Id and emits the ‘sendChord’ event. This event triggers the client to store the predicted chord in the redux store, which is subsequently displayed to the user. With the emit targeted only at socket Ids from remote streams, only users in a call with the instructor can see the chords. Additionally, the instructor receives the emit, allowing the chord to appear on their own client as well.

#### 5.13.3 Resize Room

While the resize room functionality was working as intended, it had an unintended effect of causing all elements inside the container to overlap. The solution was to display only the other user in the call when the room was minimized.

```

const VideoWrapper = styled("div")(({ isRoomMinimized }) => ({
  position: "absolute",
  left: isRoomMinimized ? "0px" : "30vw",
  top: isRoomMinimized ? "0" : "10vh",
  right: isRoomMinimized ? "0px" : "0",
  width: isRoomMinimized ? "200%" : "initial",
  height: isRoomMinimized ? "90%" : "initial",
}));

const LocalVideoWrapper = styled("div")(({ isRoomMinimized }) => ({
  position: "absolute",
  bottom: "5%",
  left: 20,
  zIndex: 1,
  visibility: isRoomMinimized ? "hidden" : "visible",
}));

```

Figure 80: Video Wrappers

To achieve this, the ‘isRoomMinimized’ state was passed down to the ‘VideoContainer’ as a prop, enabling the elements in the ‘VideoContainer’ to detect when the room was minimized. A conditional rendering was then created for the components of the ‘VideosContainer’ as illustrated in figure 80. If ‘isRoomMinimized’ was true, the visibility of the ‘LocalvideoWrapper’ and ‘ChordDisplayWrapper’ would be set to hidden, thereby hiding them from the user. The properties of the ‘VideoWrapper’ were also adjusted to fit the smaller container for the room.

#### 5.13.4 Implementing User Feedback

As discussed in the earlier sections of the sprints, there was some feedback received during the user testing of the application. Two main criticisms of the application from the testing of the video call functionality were the UI of the dashboard elements, and the UI of the Amp component. Originally, the room were represented by a Material UI icon in a button. This seemed to confuse some users, so it was replaced with a simple title in the same style as the other titles. The main criticisms of the Amp UI were that the controls for the different parameters were too small, so the container of the controls was made larger in order to enlarge the controls themselves, making them clearer and easier to control.

The chord detection functionality received criticism due to the unclear display of the chord chart at times. Rectifying this issue required increasing the size of the chord chart and positioning it closer to the video stream of the other user. The original layout of the components was reorganized, resulting in a less visually appealing design compared to the Hifi prototypes. However, the larger and clearer display of the chord chart made it easier for the user to see.

## 6 Testing

### 6.1 Introduction

This chapter describes the testing that has been undertaken for the application. This chapter is presented in two sections:

1. Functional Testing
2. User Testing

Functional testing is a type of software testing whereby the system is tested against the functional requirements. The app is tested by looking to see if the actual output for a given input corresponds with the expected output. The tests should be based on the requirements for the app. The results of functional testing can indicate if a piece of software is functional and working, but not if the software is easy to use or if it is clear what parts of the UI the user can interact with. Developers may often overlook these aspects due to spending a long time developing the functionality, for this reason User testing is also used. User testing will bring in outside participants fitting the target audience to complete a series of tasks that will test the functionality of the application but will also test the usability of the application. A usability test will consist of a series of tasks that participants need to complete, typically focused on an applications usability, navigation and overall satisfaction. From the results of these tests, changes will be made to the UI and application functionality if necessary.

### 6.2 Functional Testing

Functional testing took place throughout the development of the application. After a new component of the application was created, it would be tested once when it was initialised, and would repeatedly be tested throughout development in case any new components would interfere with its functionality. The frontend component would be tested by simply running the application and interacting with the component to see if a component was working as intended. When working with React, if there is an error with the code, the application will not run, and an error will be displayed instead of the application. This helps with the detection of bugs that may otherwise involve debugging to determine the route of the issue.

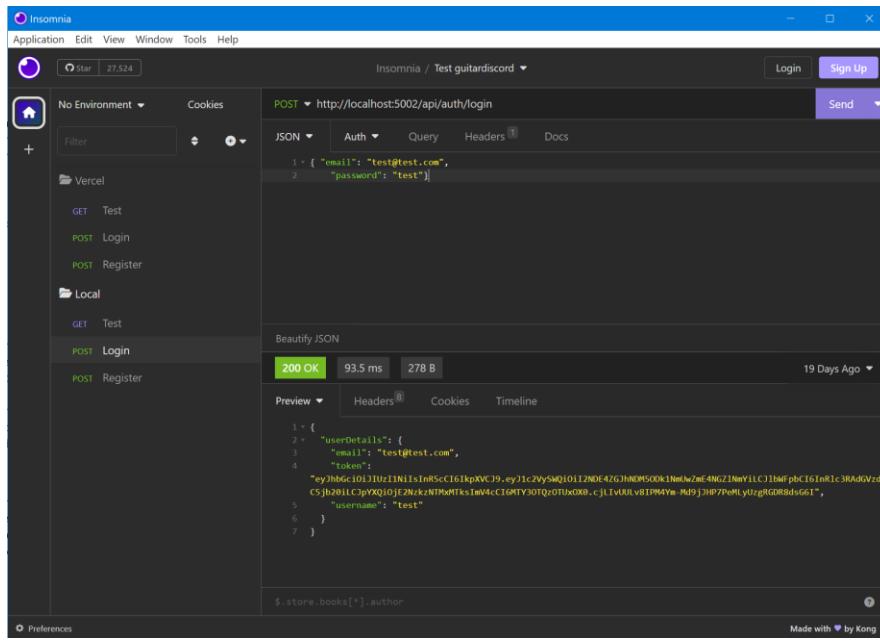


Figure 81: Testing Endpoints in Insomnia

However, this does not mean that debugging was not a part of the development process. When adding in backend servers and socket.io requests between the client and the server, debugging would be required if for instance, an event was emitted, but a listener for the event was not triggered. The Insomnia application was used during the development of the backend server, especially during the development of the different routes and sending and receiving data to and from the server. As shown in figure 71, different functionalities which required data being sent to and from a server like logging in and registering where tested. Before the server was deployed in Heroku, it was first run and tested locally to avoid having to redeploy the application each time an error was discovered. The following tables contain some of the results used for testing the application during development:

### 6.2.1 Auth

Test No	Description of test case	Input	Expected Output	Actual Output
1	Register on local server	No input	Error listing missing inputs	Error listing missing inputs
2	Register on local server	User details	Token	Error listing missing inputs
3	Login on local server	User details with error	Error	Error

4	Login on local server	User details	Token	Error
5	Register on Heroku	No input	Error listing missing inputs	Server error
6	Register on Heroku	No input	Error listing missing inputs	Error listing missing inputs
7	Register on Heroku	User details	Token	Token
8	Login on Heroku	User details	Token	Token
9	Sign out		Token removed and returned to login page	Token removed
10	Sign out		Token removed and returned to login page	Token removed and returned to login page

## 6.2.2 Guitar Testing

Test No	Description of test case	Input	Expected Output	Actual Output
1	Attempting to get guitar playback	Press on switch	Guitar audio	No output
2	Attempting to get guitar playback	Press on switch	Guitar audio	Select microphone prompt, but no audio
3	Attempting to get guitar playback	Press on switch	Guitar audio	Guitar audio
4	Control gain on playback	Move slider to control gain	Guitar audio gets louder/quieter	Application crashes
5	Control gain on playback	Move slider to control gain	Guitar audio gets louder/quieter	Slider moves, value changing in console but audio not changing volume
6	Control gain on playback	Move slider to control gain	Guitar audio gets louder/quieter	Guitar audio gets louder/quieter

### 6.2.3 Rooms

Test No	Description of test case	Input	Expected Output	Actual Output
1	Creating a room	Click room create button	Local stream and room open	Room opened but no local stream
2	Creating a room	Click room create button	Local stream and room open	Local stream and room open
3	Joining an already created room	Click room button	Join room and see/hear other user	Join room and see other user, no audio coming from other user
4	Joining an already created room	Click room button	Join room and see/hear other user	Join room and see other user, no audio coming from other user
5	All users leave room	Hangup button	Users should leave room and room should delete	Users left room but room did not delete
6	All users leave room	Hangup button	Users should leave room and room should delete	Users left room and room deleted
7	Testing two audio tracks from user	Turning on guitar before entering room	Users should be able to hear mic and guitar audio	Users can only hear guitar audio
8	Testing two audio tracks from user	Turning on guitar before entering room	Users should be able to hear mic and guitar audio	Users can join room but can't hear any audio
9	Testing two audio tracks from user	Turning on guitar before entering room	Users should be able to hear mic and guitar audio	Users can now hear both audio tracks when they join the call.

#### 6.2.4 Chord Detection

Test No	Description of test case	Input	Expected Output	Actual Output
1	Testing Spectrogram	Guitar signal	Spectrogram from guitar audio	Spectrogram from guitar audio
2	Screenshotting Spectrogram	Guitar signal	Display updating screenshot of spectrogram	Screenshot displayed but not updating
3	Screenshotting Spectrogram	Guitar signal	Display updating screenshot of spectrogram	Screenshot displayed and updating
4	Send screenshot to Custom Vision model	Spectrogram	Updating predicted note	Error with file format
5	Send screenshot to Custom Vision model	Spectrogram	Updating predicted note	First predicted note works, does not update
6	Send screenshot to Custom Vision model	Spectrogram	Updating predicted note	Updating predicted note
7	Predicted note in video call	Socket io event of note	Users in call should see chord chart	Users not receiving socket event.
8	Predicted note in video call	Socket io event of note	Users in call should see chord chart	Users receiving socket event, chord display not set up.
9	Predicted note in video call	Socket io event of note	Users in call should see chord chart	Users can now see chord in call, but display is too large
10	Predicted note in video call	Socket io event of note	Users in call should see chord chart	Users can now see resized chord chart

## 6.2.5 Discussion of Functional Testing Results

As mentioned before, functional testing was conducted while developing each component, and the provided results only represent a small, selected sample of the overall testing that took place during the development process. The testing and its outcomes were pivotal in comprehending the component's functionality entirely and guaranteeing that users wouldn't encounter any malfunctioning components. Authentication testing was especially vital in preventing unauthorized access to the app and ensuring that users were receiving the appropriate token and user details, which would be utilized by other components in the application. The tests conducted for guitar, rooms, and chord detection were considered the most significant, as these features are the primary attraction of the application. To realize the full scope of the application, it was essential to test these components separately and together.

### 6.3 User Testing

User testing commenced after the application was developed with all features fully implemented and functional. The purpose of the testing was to evaluate the application's user experience and usability. The testing was broken down into two sections: one for video call functionality and the other for AI model functionality. This decision was made as the functionality of these components were implemented at different stages and to avoid overwhelming the users during testing.

A series of applicants were chosen who matched the target audience of the application. To conduct user testing, a series of tasks were created for the usability testers to follow. For the testing of the video call component, there would be two sets of tasks, one set for the role of instructor and one set for the role of student. These roles would be given randomly to applicants so that there would be an equal number of results for both roles. The applicant would play their role while the developer would play the opposite role, as the application requires two users to take advantages of all features. The following are the list of tasks to be completed by the applicants:

1. Create an account.
2. Turn on the Amplifier so you may hear your instrument.
3. Adjust the volume of the Amplifier.
4. Add a friend with the email provided.
5. Send a message to that friend.
6. If your role is an instructor; edit the call settings.
7. If your role is an instructor; Create a room.  
If your role is a student; join the room available to you.
8. Once you have joined the call, leave the call.

The applicant will be given the opportunity to give feedback on each of these tasks, as well as the option to give general feedback on other aspects of the application.

For the testing of the AI model component, the focus would be on the student's experience. The user would join a room, where the tester would play a series of chords from their instrument which would be processed by the model and a chord would be played to the user. A series of questions would be asked to get feedback on the user's overall experience and comments. While it was not a necessity to use the same testers from the video call functionality, all previous testers agreed to take part. The following statements were given to the user, and they rated each statement how much they agreed with the statement:

1. The chord being played by the instructor was clearly displayed.
2. The image of the chord was a good representation of how to play the chord.
3. I had enough time to view the chord to copy what the instructor was playing.
4. The placement of the chord was a good position to see the chord and the instructor.
5. The chord being shown was accurate to what the instructor was playing.
6. I believe this feature could help in an online teaching environment.

The user was then asked to provide any additional feedback they may have on their overall experience.

### 6.3.1 User Testing Results

#### 6.3.1.1 Video Call Component

There was only a small amount of user testers as the criteria is very specific. As stated before, the developer would take the opposite role of the user tester during the testing. Testing results in general were positive. None of the testers had any problems with the functionality of the application, meaning that all components are working as intended. However, many of the testers had feedback relating to design of the UI.

## Responses

The dial is a bit small, but easy enough

Didn't realise the amp was functional, thought it was just decoration

Figure 82: Amp UI Feedback

As shown in the feedback in figure 82, there were many testers who felt that the UI for the 'Amp' component was not clear enough in its functionality, thinking that it was a part of the design. Other users stated that the controls for the different parameters were not large enough and had trouble interacting with them. Other feedback from users was mainly addressing the button in the dashboard and in the video call component. In the dashboard, all the buttons are the same colour as some components that are not interactable, which caused some confusion. Users that were not familiar with the 'Discord' app were not familiar with this sort of layout configuration and so initially had trouble identifying the purpose of each of the components.

### 6.3.1.2 *Guitar Chord Detection*

The same testers were used for the user testing of the guitar chord detection. Feedback of the functionality was positive, and users found the implementation to be a positive experience that could be of great benefit in a real-world scenario. There were minor complaints in relation to the position and scales of images which represented the guitar chords predicted from the model stating that they were too small and should be repositioned.

One tester mentioned the speed of which the image was displayed would be too quick to react to in the case of the instructor playing a song containing multiple quick chord changes. Their suggestion was to perhaps display all chords on the application and highlight the chord currently being played. This feedback will be taken into consideration for future developments of the application. At the time of writing, it would be too late to redesign and implement this feature into the application to test this against the current version of only displaying the chord being played.

### 6.3.1.3 *Implementation of feedback*

Based on feedback from user testing, the application's style underwent minor changes. One of the issues raised by users was the lack of clarity on which components were interactive, as some components had the same colour as the buttons used to add a friend or join a room. To address this concern, non-interactive components were replaced with text where necessary. For example, the room sidebar was previously represented by a Google Materials icon within a button, similar to the one used to join a room. The icon was replaced with a simple title, clarifying the purpose of the buttons beneath it.



Figure 83: Revised Amp Component

The most commonly raised issue was with the Amp UI, specifically, the controls were too small and challenging to operate. To remedy this, each control was enlarged for easier user interaction.

Additionally, the on switch was made larger as seen in figure 83, and mouse-over effects were added to both the switch and parameter controls to further indicate their interactive nature.

#### 6.4 Conclusion

In conclusion, this chapter has provided an overview of the testing process for the application, which was carried out in two phases: functional testing and user testing. Functional testing is a type of software testing that focuses on testing the system against functional requirements to check whether the actual output corresponds to the expected output. On the other hand, user testing involves bringing in outside participants fitting the target audience to test the application's functionality and usability. The testing for the application took place throughout the development process, and different tools were used for testing the frontend and backend components. The results of the tests were used to make necessary changes to the UI and application functionality. Finally, the results of different tests conducted during the development process were presented in tables.

## 7 Project Management

This chapter will discuss the management of the project. It will provide an overview of the project phases, starting with how the idea behind the application was conceived and how it was developed further throughout the development months.

### 7.1 Project Phases

#### 7.1.1 Proposal

The application proposal deadline was in October of the previous year. Although this gave ample time to prepare for the project once the idea was finalized, it was still early in the year to consider due to other classes and assignments occurring. Lecturers were consulted to generate a challenging idea utilizing an unfamiliar technology but not one that was impossible to complete. The initial idea was to create a shopping list application that compares prices of different supermarkets to inform the user of the cheapest place to shop. However, a lecturer suggested that the project would be too easy and encouraged taking on a more difficult task.

During the period leading up to submission date, an AI module was attended as part of the college curriculum, which sparked an interest in image and audio classification. This led to the development of an idea for a model that detects guitar chords, which could be integrated into an online video calling application to address a real-world problem. After discussing the proposal with a lecturer, it was determined that the project was feasible but also challenging enough to help acquire new skills and learn new technologies.

#### 7.1.2 Requirements

Once the project sprints began, I started researching technologies that could be used to create a video call application. Peer-to-peer technologies were identified as a means of avoiding audio and image stuttering issues that are often encountered with server-based video technology. To conduct this research, published papers and books were studied instead of relying solely on websites. Papers on the topic of music played online were also examined to determine if guitar playing could be facilitated over a video call. Studies on the impact of COVID-19 on online interactions were also reviewed. Based on this research, surveys were developed to gauge public interest in the proposed application.

Functional and non-functional requirements for the application were established during this period. These requirements were used to determine the necessary technology for developing the

application. User personas were created based on the survey results, providing additional information on the technology needed to create the application.

### 7.1.3 Design

During the design phase of the application is when the visual appearance and functionality of the different components of the application were visualised. The design process began with a simple paper prototype, which was a rough sketch created within five minutes to consider the application's appearance. This was followed by the development of low-fidelity wireframes and then a high-fidelity prototype to determine the layout of the user interface and what features should be available to users. It was decided that the application would be split into two components: the video call application and the AI chord detection model. These two components were developed separately before being merged together. In the technical aspect of the design, consideration was given to how components would interact within the application, how the server would connect to the client, and how the server would connect to the database. It was necessary to determine what would be stored in the database and what was required to develop both the server and the database.

### 7.1.4 Implementation

The implementation started by developing the two different components needed for the overall applications. Development between the two components was managed so each component would get sufficient time for completion. A basic prototype was built for the video call application, where a user could create an SDP offer and send that offer to their friend via a third-party application. Their friend could then answer the call and a peer-to-peer connection could be established, initiating the video call. This was good progress for the video call component and could be built upon.

The AI model was developed using Python and TensorFlow. It would take in a set of training data and use a CNN(Convolutional Neural Network) to create a model that could detect patterns in spectrograms of audio signals, and classify them to the correct note. This was good progress for the AI model, however there was a problem. The model would work fine locally, running in Jupyter Notebook, but there was no way for the model to take in a live audio signal, just an audio file. There would be more research needed to resolve this issue.

Doing further work on the video call component proved difficult, and not much progress was made. The difficult decision came to scrap all the code already produced and to start again from the beginning, redesigning the functionality and technology used. It was at this point where React Redux started to be used. The layout of the application also changed because of this; the application

changed to more of a social hub than a direct calling application. Development was now easing with the video application with the revised design and much more progress was being made.

While progress was being made with the video call application, there was not much progress being made with the AI model. Attempts were made to host the python model and to integrate it locally into a JavaScript application, but unfortunately it was not possible. Again, the decision was made to scrap the current version of AI model and find an alternative. This is when Microsoft's Custom Vision was discovered, which allowed users to create their own hosted AI model by simply uploading photos. It was decided to use this as an easier alternative, as while progress was being made on the video application, it was technically becoming increasingly difficult.

Towards the end of the final implementation sprints, a prototype of the application was in a functional state. This prototype had all the intended features of the original, reworked peer-to-peer connection now using room to connect instead of manually sending SDP offers and had the hosted Custom Vision model integrated. With this prototype, user testing could begin.

### 7.1.5 Testing

The testing phase was split into two sections, functional testing, and user testing. Functional testing generally took place during the development of a component, to make sure that component is both fully functioning and does not interfere with the functionality of other components. However, now that both elements of the application were put together, more testing was conducted to ensure that the functionality of each component did not interfere with one another. Functional testing was conducted for authentication, audio playback and more. As well as testing the functionality works, errors were also tested, to ensure the application behaves as expected and to avoid any user error.

Once the application was tested internally, user testing was conducted. Participants who suited the target audience of the application were chosen to take part. Participants were asked to complete a series of tasks and were asked to give feedback on each of the tasks, if they found them difficult and if they would change anything. These tests were conducted to see if the application worked as intended as well as to see if there was any aspect of the application overlooked, as sometimes it can be hard to tell after spending such a long time developing. There were no issues with any of the functional aspects of the application. Users mainly had issues with different elements of the UI, which are easily remedied.

## 7.2 SCRUM Methodology

The SCRUM methodology proved very useful during the whole development process. It was a good way to plan what aspect of the application should be focused on, and when that focus should change. The sprints were a good reminder on the time left in development and keeping informed on when it is time to move on. For example, in the Implementation chapter it was explained how the decision was made to redesign the application. One of the deciding factors on this decision was made from checking the sprints and realizing how little development time was left. There was no time for trying to fix broken code and it was time to redesign the code, or else too much time would be wasted.

## 7.3 Project Management Tools

### 7.3.1 GitHub

GitHub was used during the duration of the development of the application. GitHub was an effective tool for managing and keeping track of code. Using branches allowed for multiple iterations of the application to be worked on at once, which allowed testing of new ideas and attempts at integrating code which may not have worked initially. By using a branch, there would be a way to revert back to working code should something go wrong. GitHub commit history also proved useful, creating a project log in which progress for development could be recorded. GitHub also made working from more than one workspace much more efficient, allowing the user to download the most recent version of their code even when it is not stored locally.

### 7.3.2 Trello

During the beginning of the development of the application, Trello was used to track and easily manage tasks. It allows users to prioritize tasks and projects by creating lists such as “to-do” and “in progress”, so the user can easily track their progress. While this functionality is useful, it did not see much use. Due to the simply extra steps of having to open a browser a sign into the application, Trello was not used as much as intended. Instead, tasks were simply pinned to the desktop and written down on physical paper. While these methods are certainly not as well organized, they were much easier and quicker. This did however lead to some inorganization problems, which would have been easily rectified if Trello was used throughout the project. Trello would have more likely to be used if there was more than one developer working on the project.

## 7.4 Reflection

### 7.4.1 Your Views on the Project

Despite the complexity of managing the different sprints and requirements of the project, I found the development of the application to be an enjoyable learning experience. The opportunity to work with new technologies like WebRTC was particularly rewarding, and I now feel confident in my ability to apply these skills in future projects. As someone with a personal interest in music and guitar playing, I believe that the application I developed has practical value. In addition, the project provided an opportunity to explore Artificial Intelligence, machine learning, and neural networks, which are increasingly relevant in the industry.

Overall, the project has not only enhanced my technical skills in coding and development, but also improved my abilities in time management, research, and critical thinking. Through the process of developing the application and planning for testing and other project aspects, I was able to solve problems and visualize solutions that I would not have been able to in the past.

### 7.4.2 Completing a Large Software Development Project

While this was not my first time working on a large project such as this, it was the first time tackling such a large project alone. Having to manage each aspect of the project while still having to develop the application brings a different perspective and appreciation for working in a team and each member having their own role. While I struggled at the beginning to the development to keep track of progress, I slowly learned to follow a strict schedule to make sure only essential tasks were completed. As mentioned in the Trello section, the project could have been managed much more efficiently, but I also feel that this is the reason teams have a leader to oversee the schedule and make sure things are on track.

### 7.4.3 Working With a Supervisor

Having a supervisor during the development of the project as a great help. I met with my supervisor, Timm, once a week to discuss the progress I had made and what I planned to do for the next week. At the end of each sprint, we would revise the work that had been completed to determine how well the development was going. Timm provided reassurance at times when I felt overwhelmed, having me redirect my focus to the tasks at hand instead of worrying about future tasks. While Timm was not familiar with the technologies I was working with, he often had a good opinion and outsiders view to the development process, suggesting ideas that would not have occurred to me had I not been discussing the project with him. Having a supervisor kept me on track during sprints was one of

the main reasons I feel like the development process did not fall apart. Timm's contributions to the project were invaluable I feel the project would not have been successful without his input.

#### 7.4.4 Technical Skills

This project has significantly enhanced my technical skills in full stack development. Although I had some experience working on Express/Node servers, I had to create a server from scratch and develop it to handle a large amount of functionality, which vastly improved my skills. Additionally, I had no previous experience or knowledge of socket IO, but now I feel comfortable integrating it into any application. Connecting these technologies to a client and database has deepened my understanding of their relationship and will be advantageous in developing my skills further. Previously, I had no experience using React Redux, but learning how to use it has improved my skills and taught me how to use more advanced and efficient state management. I believe that combining my experience in all these technologies will help me achieve my goal of becoming a full stack developer.

#### 7.4.5 Further Competencies and Skills

As mentioned in the Technical Skills section, I believe the skills gained from developing a full stack application will provide me with a better chance of gaining a position in the workplace, as I had previously mainly only worked on front end applications. I also feel having a project working with AI may attract possible employers, as AI is currently very popular within the tech industry. In future developments, I plan to stick with an organization application more closely such as Trello, but perhaps rethink how I interact with it, as I tend to prefer the quicker option.

### 7.5 Conclusion

In conclusion, effective project management is crucial for the successful completion of any project. This chapter has provided an overview of the different phases of the project, from the initial proposal to the final testing phase. The proposal phase involved identifying an idea for the application and conducting research. The requirements phase involved establishing functional and non-functional requirements for the application and creating user personas. The design phase involved visualizing the appearance and functionality of the application and developing wireframes and prototypes. The implementation phase involved developing the different components of the application and integrating them to create a functional prototype. Finally, the testing phase involved conducting user testing to ensure that the application met the established requirements. Through effective but not perfect project management, I was able to overcome various challenges and produce a functional prototype that met the initial requirements.

## 8 Conclusion

The project aimed to develop an AI assisted video call application that facilitates easy connectivity between new guitarists and professional musicians/instructors. The application allows for the seamless exchange of audio and video data, addressing the challenges associated with meeting in person and the shortcomings of existing video call applications. The success of the project is evident in the application's ability to transmit both the microphone and guitar signals of the users while also displaying the chords played by the instructor. Additionally, the application includes additional features such as the option for two cameras, despite the existence of bugs in their current state, indicating the possibility of their implementation.

### 8.1 Summary of Chapters

The research chapter constituted a crucial aspect of this project, being the first and one of the most significant. The investigation of all available technologies was a crucial step in developing the application, requiring the determination of the most suitable technology for the project. The research focused on the practical aspects of these technologies and how they could be combined to develop an application that goes beyond the technology itself. The study of machine learning, neural networks, and audio classification was particularly important since it provided the necessary information for building the design using spectrograms from a converted guitar signal. This information was crucial because attempting to run the machine learning model locally would have imposed an overwhelming workload on the browser, rendering the application purposeless. From this research, the requirements of the application could be investigated.

The requirements chapter focused on the exploration of existing applications available to the public and the identification of the intended audience for the application. Awareness of the features offered by other applications was necessary to provide users with a compelling reason to choose this application over established alternatives. In addition, understanding the preferences and requirements of the target audience helped to determine the features that should be incorporated into the application. The survey conducted also generated feedback from the target audience, providing insights into their experiences using other available applications and gauging their interest in the proposed application. The information gained could then be used to work on the design of the application.

The knowledge obtained from the research and requirements was utilized to shape the design of the application. These chapters provided guidance on the overall architecture of the application,

including how features should be incorporated and how they would interact with each other. Design patterns were employed to structure the components of the application. The application architecture and flow chart were developed to provide a visual representation of how users would interact with the application. The creation of lofi and hifi prototypes allowed for visualization of the application, drawing inspiration from applications researched in the requirements chapter. This approach facilitated styling of the application during development, eliminating the need to revise the application after implementing the functionality.

With the completion of the design, the implementation of the application commenced. Leveraging the knowledge gained from previous chapters, the application could be developed. As stated in detail in the sprints, the AI and video call components were developed separately and subsequently merged to create the application. This approach proved effective, enabling the developer to switch between the two components when encountering difficulties and requiring additional research to complete the task at hand. By managing the development through SCRUM sprints, the developer could monitor progress and address any obstacles that arose during the development process. Once all the functionality was implemented, testing could begin.

Functional testing was conducted during the development of the application. Upon implementation of a specific functionality, it would be tested to validate its expected outcome and to test its limits to enable the developer to program error handling to prevent application crashes due to user errors. Once all functionality and styling were integrated, user testing commenced. Users who matched the target audience for the application were recruited and tasked with testing all the different functionalities of the application. They were provided with a list of tasks and an opportunity to provide feedback on usability, visual appeal, and overall impression. The feedback on functionality was positive, but several testers encountered visual issues with the application. These issues were addressed and resolved after testing.

## 8.2 Future improvements

Though the development of the application has been successful, there are features that could be added to further improve upon the application. Integration of VSTs and audio plugin software could enable users to manipulate the sound produced by their instruments, expanding the application's utility beyond a basic learning platform, and offering professional musicians the ability to practice and compose music collaboratively online, without needing to rent a rehearsal space. This could

facilitate online rehearsals for full bands and enable musicians from around the globe to connect with others they may not have encountered otherwise.

The model used to detect notes could be improved upon. It was only trained on basic guitar chords in order to demonstrate the potential of integrating an audio classification model into a video call application. The model could be trained further, using more complex chords and voicings, and potentially even be trained on every note capable of being played by a guitar. This would potentially allow the model to transcribe songs, and could offer an instructor an additional tool in an teaching environment.

Another potential feature that could facilitate learning is the ability to record lessons. This feature would allow students to review lessons with their instructors for a refresher on topics discussed during the lesson. It is important to acknowledge that while these features would enhance the application, they could also broaden its scope and complexity, potentially rendering it unsuitable for browser-based operation.

### 8.3 Personal takeaway

The project accomplished its objectives by developing a peer-to-peer video call application for guitarists and integrating an AI audio classification model to identify chords played by the instructor and present them to the student. The project provided an opportunity to learn new technologies such as React Redux and Socket IO that could be implemented into future projects. The project also provided insights into AI, machine learning, and neural networks, which could be relevant to future employment prospects.

The project was valuable in teaching how to manage the creation of a large-scale application from inception to completion. It provided insights into proper component management and efficient time management. Additionally, it emphasized the importance of adopting new approaches when original ideas do not work. The project management approach helped to learn the importance of setting deadlines and being able to discard components if they did not meet the objectives of the project.

## 9 Bibliography

- Amir, D. (2019, October 30). *Harnessing The Power Of Uncertainty*. Retrieved from Towards Data Science: <https://towardsdatascience.com/harnessing-the-power-of-uncertainty-3ad9431e595c>
- Bishop, C. M. (1995). *Neural Networks and Their Applications*. Birmingham: Department of Computer Science and Applied Mathematics.
- Camastra, F., & Vinciarelli, A. (2015). *Machine Learning for Audio, Image and Video Analysis*. London: Springer.
- Chaturvedi, S., Titre , R., & Sondhiya, N. (2014). *Review of Handwritten Pattern Recognition of Digits and Special characters using Feed Forward Neural Network and Izhikevich Neural Model*.
- Duggal, N. (2022, November 24). *All You Need To Know About MERN Stack*. Retrieved from SimpliLearn: <https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mern-stack-introduction-and-examples>
- Dutton, S. (2013, November 4). *WebRTC in the real world: STUN, TURN and signaling*. Retrieved from HTML5 Rocks: <https://www.hehuo168.com/hehuo/WebRTC%20in%20the%20real%20world%20STUN,%20TURN%20and%20Signaling.pdf>
- Fedosejev, A. (2015). *React.js Essentials*. Packt Publishing.
- Fogg, R. (2021, June 24). *Chord Clinic: Learn to play 10 interesting C major chord variations*. Retrieved from Guitar: <https://guitar.com/lessons/beginner/learn-to-play-10-interesting-c-major-chord-variations/>
- García , B., Gallego, M., Gortázar, F., & Bertolino, A. (2018). *Understanding and estimating quality of experience*.
- Gomez, D. (2023, January 24). *Object detection with Azure Custom Vision* . Retrieved from Dev: <https://dev.to/esdanielgomez/object-detection-with-azure-custom-vision-38ka>
- Günther, F., & Fritsch, S. (2010). *Neuralnet: Training of Neural Networks*.
- IMRO. (2020). *Annual Report and Accounts* . Irish Music Rights Organisation.
- Jiang, M. (2020, April 22). *Video chat is helping us stay employed and connected. But what makes it so tiring - and how can we reduce 'Zoom fatigue'*? Retrieved from BBC: <https://www.bbc.com/worklife/article/20200421-why-zoom-video-chats-are-so-exhausting>
- Jurafsky, D., & Martin, J. H. (2021). *Speech and Language Processing*.
- Kroese, B., & Smagt, P. v. (1996). *An Introduction to Neural Networks*.
- Kumar, A., & Singh, R. K. (n.d.). *Comparative Analysis of AngularJs and ReactJs*.
- Lee, T. B. (2018, December 18). *How computers got shockingly good at recognizing images*. Retrieved from Ars Technica: <https://arstechnica.com/science/2018/12/how-computers-got-shockingly-good-at-recognizing-images/>

- M, M. (2018, October 19). *React JS for WordPress Users: A Basic Introduction*. Retrieved from Elegant Themes: <https://www.elegantthemes.com/blog/wordpress/react-js-for-wordpress-users-a-basic-introduction>
- Manson, R. (2013). *Getting Started with WebRTC*. Packt Publishing.
- Murphy, M. J., & Naqvi, I. E. (2015). *What is Machine Learning?*
- Peek, S. (2023, March 20). *What Is Agile Scrum Methodology?* Retrieved from Buisness News Daily: <https://www.businessnewsdaily.com/4987-what-is-agile-scrum-methodology.html>
- Robie, J. (n.d.). *What is the Document Object Model?* Retrieved from W3: <https://www.w3.org/TR/REC-DOM-Level-1/introduction.html>
- Rossouw, F. J. (2020, April 30). *Introduction to WebRTC*. Retrieved from Medium: <https://medium.com/dvt-engineering/introduction-to-webrtc-cad0c6900b8e>
- S, A. (2021, January 8). *ML Fundamentals: What Is Cost Function?* Retrieved from Medium: <https://medium.com/swlh/ml-fundamentals-what-is-cost-function-d396129cc611>
- Saks, E. (2019). *JavaScript frameworks: Angular vs React vs Vue*.
- Soeta, Y., & Nakagawa, S. (2015). *BE(BirdMEG)*.
- Ubah, K. (2021, July 26). *What is Pseudocode? How to Use Pseudocode to Solve Coding Problems*. Retrieved from Free Coding Camp: <https://www.freecodecamp.org/news/what-is-pseudocode-in-programming/>
- VNI, C. (2021). *Forecast and Methodology, 2016–2021*.
- Web Audio API - Web APIs: MDN.* (2023, 25 January ). Retrieved from Web APIs | MDN: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Audio\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API)
- Zahal, O., & Ayyıldız, E. B. (2022). Instructor experiences with online guitar lessons during the Covid-19 pandemic in Turkey.

## 10 Appendices

### 10.1 Appendix A – Requirement Gathering Survey

[https://forms.office.com/Pages/ResponsePage.aspx?id=e5V92hEVQkqy9Xj4R\\_jlehDCSp7VCUFkmU-ZW51GmWJUOFdOMDNMUIZOVUlRFYyTTRBMk1QTUDWS4u](https://forms.office.com/Pages/ResponsePage.aspx?id=e5V92hEVQkqy9Xj4R_jlehDCSp7VCUFkmU-ZW51GmWJUOFdOMDNMUIZOVUlRFYyTTRBMk1QTUDWS4u)

### 10.2 Appendix B – Figma: Lo-fi Prototypes

[https://www.figma.com/file/Gb0qCwhtV8Ph8jZxN05TzD/FYP---Wireframes?node\\_id=0-1&t=cpUfNF3jKZoE3KwN-0](https://www.figma.com/file/Gb0qCwhtV8Ph8jZxN05TzD/FYP---Wireframes?node_id=0-1&t=cpUfNF3jKZoE3KwN-0)

### 10.3 Appendix C – Figma: Hi-fi Prototypes

<https://www.figma.com/file/IuYS7Y7X6qkH8QFVG1VWat/Hi-Fi-Prototype?t=cpUfNF3jKZoE3KwN-0>

### 10.4 Appendix D – User Testing: Video Call Application

<https://forms.office.com/e/EuU8HxAGsU>

### 10.5 Appendix E – User Testing: Chord Recognition

<https://forms.office.com/e/Z2v3967KWz>

### 10.6 Appendix F – Hosted Site

<https://intelligentinstrumenttest.web.app>

### 10.7 Appendix G – Hosted Server

<https://intelligenttest.herokuapp.com>