UNIVERSITY OF TWENTE.

Design Project
SonicViz

Authors:
Lieuwe van den Berg – s2962667
Wouter ten Brinke – s2924471
Tom van Rijn – s3002284
Lilya Saliba – s2846039
Wessel Witteveen – s3004346

Supervisor:
dr.ir. M. Zangiabady

Group:
18

April 16, 2025

Abstract

This report provides an application that is designed to capture, process, and visualize audio in real time, making sound more interactive and engaging. The app is designed with educational goals in mind, targeting Dutch high school students to spark their interest in STEM (Science, Technology, Engineering, and Mathematics) by making sound more interactive, prompting the students to think deeper about the possibilities this field has to offer. Users can record their own audio, explore preloaded samples, and experiment with a variety of visualizations such as waveforms, spectrograms, and frequency charts. Built using modern web technologies including Ionic, React, and the Web Audio API, the app supports a smooth and responsive experience across different platforms. In addition to its core recording and visualization features, SonicViz also includes customizable settings, interactive games focused on pitch and note recognition, and an accessible interface that encourages creative exploration.

Contents

Chapter 1.

Introduction

In Dutch high schools, students need to deliver their own final project at the end of their study. They are allowed to pick a subject on which they will do their project, which is often related to which study they are interested in or planning to do. Subjects related to STEM fields can be intimidating or they may seem uneventful to some students. In order to attract more students towards this area, they should be informed of all the possibilities the studies in STEM offer. This can be achieved by providing projects that are both enjoyable and interesting. SonicViz is a perfect example of combining these technical subjects with creativity, where students can experiment with sound visualization and learn about its fundamentals or the underlying processes.

1.1 Background

The visualization of audio poses as the main component of this project. Many forms of visualization exist and there are various approaches to this topic as well as similar ones. These existing methods and approaches of audio visualization will be discussed below.

First, we will briefly mention the classical forms of visualization. Sheet music is a very traditional method of musical notation, which is also a type of visualization: simply using paper and ink to write down the musical notes which indicate what sounds should be played. However, there are also applications like MuseScore[1] that contain playable sheet music of certain songs or allow you to create and upload your own. Sheet music is often used by musicians to know how to play certain songs with their instruments. When looking at an orchestra, a large group composed of musicians with various instruments, they do not only use sheet music to create the music. Orchestras are often directed by a conductor who sets the tempo and can control the way the musical piece will be played by the musicians. In a way, this expression of movements can also be considered as a form of visualization of music.

Moving on to the more modern implementations, there are several already existing platforms that can be used for audio visualization of music. Simple media players sometimes have certain types of visualization for audio files. Users can use it to explore these visualizations on their own. Another example is Specterr[2], an audio visualization software which provides an online platform where users can select or create their own music visualizer. Another well known platform is Adobe After Effects[3], an application with which users can create animations and thus also use it to visualize songs with their own designs. Platforms like Specterr and Adobe After Effects are primarily used for recreation and creative purposes where the user can experiment with the software to design their own visualization types.

[1] https://musescore.com/
[2] https://specterr.com/music-visualizer/
[3] https://www.adobe.com/products/aftereffects.html

1.2 Objectives

The objective of this project is to design, develop, and implement an audio visualization app specifically tailored for Dutch high school students. The primary aim is to create an interactive and engaging tool that combines music and visual elements to introduce students to various concepts within the STEM fields. By leveraging audio data, the app will provide captivating visualizations that aim to stimulate students' interest in technology, sound analysis, and creative design.

The project will focus on making the user experience intuitive and accessible, ensuring that students with little to no prior knowledge of STEM concepts can still enjoy the app and explore its features. Through dynamic visualizations, students will be encouraged to experiment with different audio inputs and observe how sound data translates into visual representation, fostering curiosity and inspiring them to explore the technical aspects of sound and digital media. The ultimate goal is not only to engage students in a fun and educational way but also to motivate them to pursue further learning in the fields of science and technology.

Chapter 2.

Requirements

This chapter outlines the essential requirements for the audio visualization app. The requirements are structured using the MoSCoW method, which categorizes them into four priority levels:

- **Must have** – Critical requirements that the app cannot function without.
- **Should have** – Important features that significantly enhance usability and experience but are not fundamental.
- **Could have** – Additional functionalities that would be beneficial but are not crucial for the initial release.
- **Will not have** – Features that are explicitly excluded from the scope of this version.

The requirements are further classified into functional and non-functional categories. Functional requirements define the core operations of the app, while non-functional requirements describe quality attributes such as performance and usability.

2.1 Functional requirements

These requirements ensure that the functionality of the app is clearly laid out.

2.1.1 Must have

The following features are essential for the app to fulfill its core purpose of enabling interactive and educational audio visualization:

- The app must be able to record audio from the microphone and visualize it in real-time.
  - Real-time visualization helps users immediately see how their sound is being interpreted.
  - This encourages playful experimentation, aligning with the goal of engaging students.
- The app must display real-time feedback when recording, such as a live waveform preview.
- The app must provide multiple visualization styles, including waveforms, spectrograms, and abstract animations.
  - This flexibility makes the app more inclusive and engaging.

2.1.2 Should have

These features add value and improve user satisfaction, particularly in educational and demo contexts:

- The app should allow users to record and store audio for later visualization.
  - Useful in scenarios where users want to compare different recordings.

- The app should include a set of preloaded audio samples for demonstration purposes.
  - Supports instant usage in environments where live recording is impractical.
  - Helps showcase different types of sound without needing external input.
- Users should be able to select different visualization types dynamically.
  - Allows users to interactively explore how different visuals represent the same sound.
  - Enhances educational impact by supporting experimentation and comparison.
- The app should allow users to adjust visualization settings (e.g., color themes, intensity).
  - Personalization increases user engagement.
  - Adjusting visuals may be useful for accessibility or aesthetic preference.

2.1.3 Could have

The following features are not critical for launch but would improve engagement and user depth:

- The app could support streaming and processing short audio clips (e.g., 30-second previews from Apple Music, Deezer, or SoundCloud).
  - Offers a wider range of audio sources.
  - Encourages users to explore sound beyond their own recordings.
- The app could include noise reduction for recorded audio to improve clarity.
  - Makes visualizations more accurate, especially in noisy environments.
- Small gamification features could be added to increase engagement (e.g., challenges or achievements related to visualization usage).
  - Promotes repeated use and learning through play.
- The app could provide explanations of how data is being processed, converted, and visualized.
  - Helps users connect visual output to the underlying technology.
- The app could support zooming in and out on the visualization to analyze details.
  - Enables deeper inspection of waveforms and frequency patterns.

2.1.4 Will not have

These features are intentionally left out of this version to keep the scope focused:

- The app will not have an authentication system, as user accounts are not required for core functionality.
- The app will not include a full-fledged audio editing suite.
  - The focus is on visualization and exploration of high school students, not detailed editing.
- The app will not support real-time collaboration between multiple users.
  - Collaboration would require networking features and synchronization, which are beyond current scope.

2.2 Non-functional requirements

These requirements ensure that the app is usable, performant, and aligned with modern expectations of quality:

- **Low latency:**
  - The app must process and visualize audio with minimal delay.
  - This is critical for real-time feedback and interactive learning.

- **Attractive UI/UX:**
  - The interface should be modern, visually appealing, and consistent.
  - A polished look and feel makes the app more enjoyable.
- **Simplicity and intuitiveness:**
  - Users should be able to use the app with little to no instruction.
  - A clear layout and intuitive controls support accessibility for younger students or those new to STEM.
  - At less easy to understand parts of the app, information should be available.
- **Mobile compatibility:**
  - The app must be available as an Android application (iOS application distibution requires being part of the Apple Developer Program which has a yearly subscription fee [1]).
  - Most high school students use mobile devices on a daily basis.
- **Smooth animations:**
  - Visual transitions and animations should run at consistent high frame rates.
  - Smooth visuals are more immersive and reduce fatigue.
- **Offline functionality:**
  - Users should be able to use key features (e.g., loading local files, viewing past recordings) without internet access.
  - This is essential for use in classrooms or workshops without reliable connectivity.

[1] Apple

Chapter 3.

Risk analysis

Creating an audio visualization app comes with several potential challenges. To keep development on track and ensure the app has educational value, it is important to identify and address these risks early. This chapter presents an overview of key risks related to technical development, usability, educational effectiveness, and ethical considerations. Each risk is categorized by severity and how difficult it is to mitigate, using the following color code:

Red	High severity, difficult to mitigate	Yellow	Medium severity
Orange	High severity, easier to mitigate	Green	Low severity

3.1 Technical risks

Developing an app that processes audio and renders visualizations in real time can introduce a range of technical challenges. This section outlines potential technical risks and how they can be mitigated.

Risk	Mitigation Strategy
Performance issues on older phones	Optimize both rendering and audio processing. Test the app on a variety of devices to ensure stable performance across different hardware.
Difficulty with real-time visualization	Use efficient visualization techniques. Make sure to only draw new data, for example, and optimize rendering in that way.
Audio input or recording bugs	Perform extensive testing of microphone features on multiple devices. Provide fallback audio samples to ensure the app remains functional if live recording fails.

Table 3.1: Technical Risks

3.2 Usability risks

This section covers risks related to user experience and ease of use. which is incredibly important to consider when creating an app.

Risk	Mitigation Strategy
Students do not understand the app	Implement clear onboarding elements such as tooltips, short animations, and a help screen with examples written in accessible language.
User interface feels overwhelming	Design the UI to be minimal and structured, using recognizable icons and limiting the number of elements on screen.
Poor gesture support on mobile devices	Test the app on different mobile devices with various users to ensure basic gestures work as expected and feel intuitive.
Lack of personalization options	Add optional features such as theme selection or customizable visual effects to increase user engagement.

Table 3.2: Usability Risks

3.3 Educational engagement risks

Even a well-designed and functional app may fail to achieve its educational goals if it does not succeed in engaging students or sparking curiosity about STEM. The following risks address how the app's content and design can influence it's effectiveness.

Risk	Mitigation Strategy
App does not succeed in sparking interest in STEM	Include educational content that explains how sound works and relates to STEM topics. Add interactive elements or small challenges to promote exploration.
App is perceived as purely entertainment	Emphasize educational features in both design and content. Make the purpose of the app clear from the start.
Low long-term engagement	Introduce optional progress tracking, achievements, or other gamification elements to encourage repeat use.

Table 3.3: Educational Engagement Risks

3.4 Ethical and privacy risks

it is important to consider how data is handled and how the app might be used in unintended ways. This section identifies potential ethical and privacy-related concerns.

Risk	Mitigation Strategy
Recording of copyrighted or inappropriate content	Display a warning before recording starts and prevent users from exporting or sharing visualizations made from such content.
Unclear handling of user data	Clearly state in the app that all audio remains local and that no data is uploaded or stored without explicit consent.
App is used for non-educational purposes	Keep the focus of the app on learning and creativity, both in how it is presented and in its core features.

Table 3.4: Ethical and Privacy Risks

By identifying and addressing these risks early in the project, the overall quality, reliability, and educational impact of the app can be improved. This structured approach to risk analysis also helps ensure that the final product meets its intended goals while remaining practical to develop within the available time and resources.

Chapter 4.

Audio fundamentals

The current chapter will explain some theoretical background to the audio component of this project. Knowledge about how sound works as well as how it can be digitally modified is beneficial to the project. This is used for the audio visualizations as well as the filters. To further elaborate, the different kinds of visualizations that have been used are also explained in this chapter in more depth. It will conclude with various methods in which audio visualization can be applied.

4.1 Sound waves

A sound source create pressure changes, which travel through a medium such as air, water, or solids as longitudinal waves. These pressure changes cause the particles in the medium to compress and stretch, creating alternating areas of high pressure and low pressure. This motion allows the sound wave to carry energy from the source to our ears.

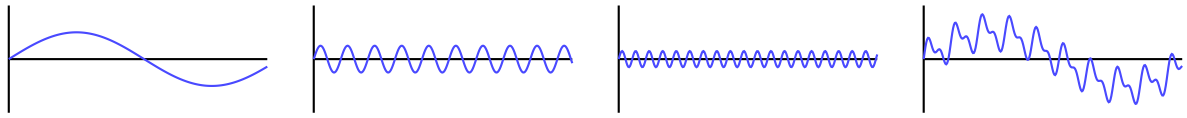
When talking about sound waves, two key characteristics come up. The first one is amplitude, this describes the height of the sound waves peaks. When the amplitude increases the loudness of the sound increases, and the other way around when the amplitude decreases. Apart from amplitude there is frequency, this refers to how often a wave vibrates per second. High-frequency waves sound high-pitched; low-frequency waves sound deep or low-pitched.

4.1.1 Hearing

Sound waves make the eardrum vibrate, and these vibrations pass through three tiny bones (the malleus, incus, and stapes) to the oval window. The oval window then sends these vibrations into the fluid-filled, spiral-shaped cochlea. Different areas of the cochlea respond to different pitches: high pitches near the base, and low pitches toward the tip. As the fluid moves, tiny hair cells bend and turn the movement into electrical signals, which travel through the auditory nerve to the brain for us to hear and understand (van den Boomgaard, 2022).

4.2 Analog signal and filters

An analog signal is a continuous signal that represents physical measurements in this case, sound. Sound waves can be record with a microphone as an analog electrical signal, where the voltage varies over time, with the shape of the original sound wave. Filters are essential components in analog signal processing, particularly when working with audio signals. In this section, the most commonly used analog filters are described, each with a simple visualization showing their effect on an audio waveform. The audio waveform is a composition of three sinusoidal waveforms with three different frequencies, one in the low range Figure 4.1a, one in the medium range Figure 4.1b and one in the high range Figure 4.1c. These frequencies are combined into one signal Figure 4.1d and specifically chosen for these filters to clearly show their effects. We used Ling (2007) as a source for the following subsections explaining the pass and stop filters.



(a) Low-frequency. **(b)** Medium-frequency. **(c)** High-frequency. **(d)** Combined-frequency.

Figure 4.1: Overview of different signals.

4.2.1 Low-pass filter

A low-pass filter allows frequencies below a certain cutoff point to pass through while attenuating higher frequencies. This makes it especially useful for smoothing out signals, reducing high-frequency noise, and isolating slower-changing components of an audio signal. Figure 4.2 illustrates how a low-pass filter reduces the sharp fluctuations in a signal, resulting in a smoother waveform.

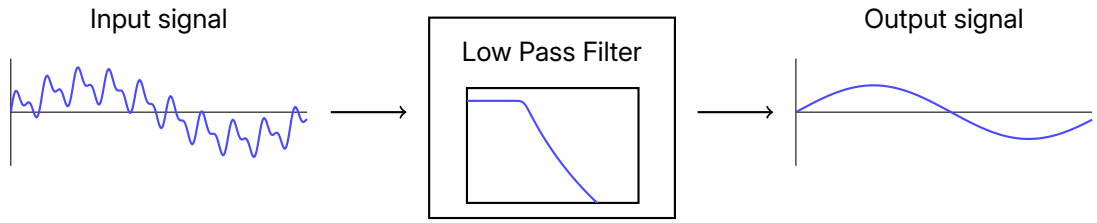


Figure 4.2: A simple visualization showing the before and after effect of a low-pass filter.

4.2.2 High-pass filter

In contrast to low-pass filters, high-pass filters remove low-frequency components from a signal while allowing higher frequencies to pass through. They are often used to eliminate low-frequency noise, such as rumble or hum, from audio recordings. As shown in Figure 4.3, the filter sharpens the signal by removing slower changes and preserving rapid fluctuations.

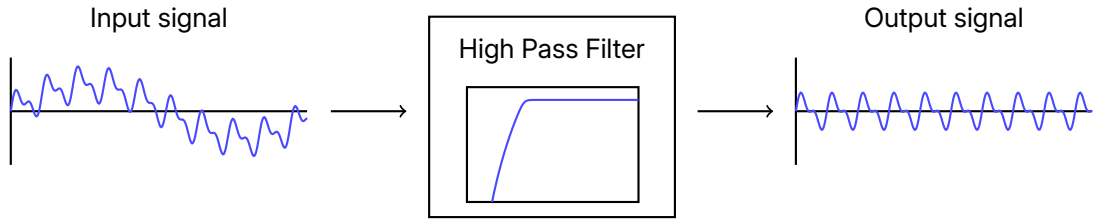


Figure 4.3: A simple visualization showing the before and after effect of a high-pass filter.

4.2.3 Band-pass filter

A band-pass filter allows a specific range of frequencies to pass through while attenuating those outside the range. This makes it ideal for isolating certain frequency bands, such as detecting a particular tone or tuning into a specific channel in radio communications. In audio processing, band-pass filters are used for tasks like instrument or vocal separation and audio feature extraction. Figure 4.4 shows how only the middle portion of the signal's frequency content is preserved.

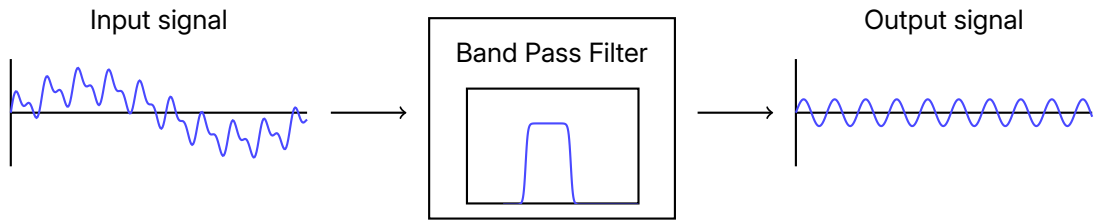


Figure 4.4: A simple visualization showing the before and after effect of a band-pass filter.

4.2.4 Band-stop filter

A band-stop filter, also known as a notch filter, is the opposite of a band-pass filter. It attenuates a specific frequency range while allowing frequencies outside that range to pass through. These filters are commonly used to eliminate unwanted frequencies such as electrical hum or feedback tones. In Figure 4.5, the filter removes a distinct part of the frequency spectrum, reducing the unwanted component while preserving the rest of the signal.

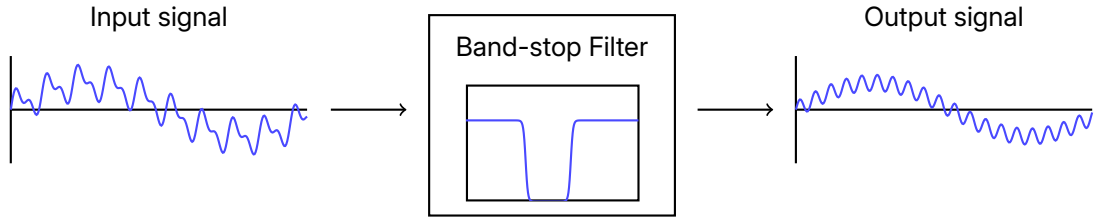


Figure 4.5: A simple visualization showing the before and after effect of a band-stop filter.

4.2.5 Gain filter

Unlike the other filters that focus on shaping the frequency content of a signal, a gain filter adjusts the overall amplitude of the signal. It can either amplify or attenuate the entire signal uniformly, which is useful for volume control, signal matching, or dynamic processing in audio systems. Gain filters are often applied before other filters to achieve the desired input level. Figure 4.6 demonstrates how the signal becomes stronger in amplitude, depending on the gain applied. (Connaghan, 2021)

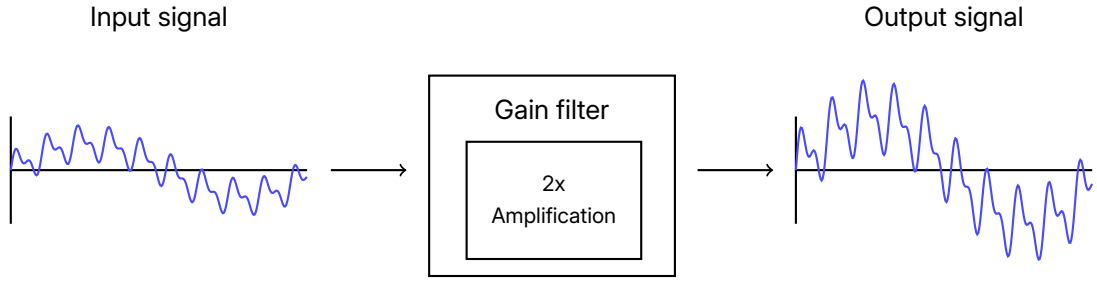


Figure 4.6: A simple visualization showing the before and after effect of a gain filter.

4.2.6 Reverb filter

The reverb filter performs a linear convolution with the input signal and a provided impulse response file. This impulse response file is a short audio file that describes how the input signal should be shaped after performing the convolution operation with the impulse response. The impulse response can therefore determine the sound of the output, simulating specific locations or speakers, for example. It is possible to stretch the input signal across a longer frame of time, essentially creating a reverb effect

in the audio. We can use an impulse response recording from a location with a lot of echo to simulate this effect (White, 2006).

4.3 Digital audio

Sound in the real world is an analog signal, meaning it varies continuously over time. To work with sound on a computer, it first needs to be transformed into a digital format. This transformation is done using a process called *analog-to-digital conversion* (ADC). During this process, a microphone or other input device captures the sound waves and converts them into an electrical signal. This signal is then sampled at regular intervals, producing a series of numerical values that represent the sound's amplitude at each point in time.

Once the sound is digitized, it can be stored in various file formats (such as WAV, MP3, or FLAC), transmitted over the internet, or manipulated using software for editing, analysis, or playback. Audio data can also be played back through digital-to-analog conversion (DAC), which turns the digital numbers back into a smooth electrical signal sent to speakers or headphones.

In the context of web applications, audio can be captured using APIs like the Web Audio API or MediaStream API, processed in real time, and visualized or streamed to other users.

4.3.1 Quantization

When converting an analog signal into its digital form, some information is lost due to a process called *quantization*. This loss occurs because the continuous analog signal is sampled at regular intervals. At each interval, the value of the analog signal is approximated by assigning it to the nearest discrete level or *bin*, a process that introduces a small error known as *quantization errors*.

Figure 4.7 illustrates a simplified example of quantization using a low sample rate and bit depth. Although such settings are far below those used in practical systems, the figure clearly demonstrates how a smooth waveform is approximated by discrete steps in a digital representation.

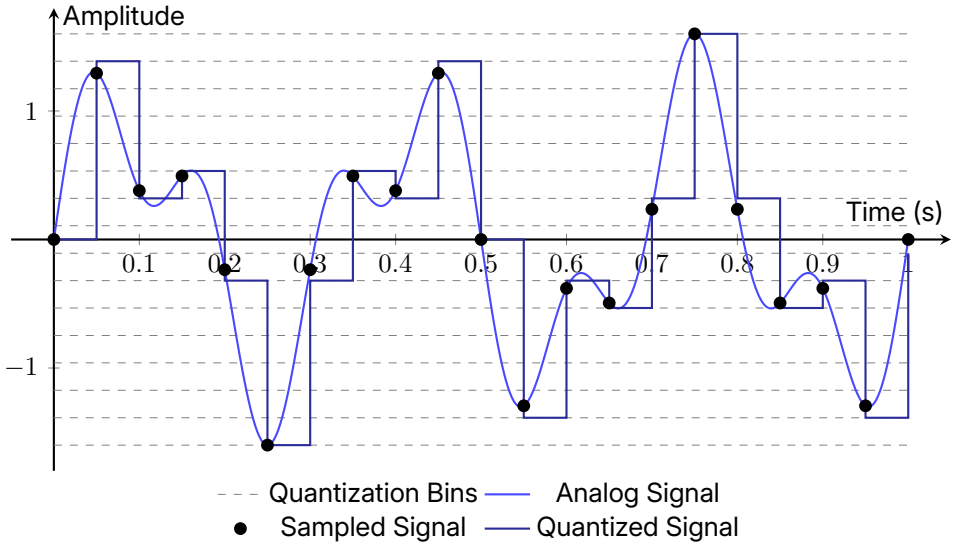


Figure 4.7: An example of quantizing an analog signal with a sample rate of 20 Hz and depth of 4-bit.

Two important parameters define the precision of this conversion: the `sample rate` and the `bit depth`. The `sample rate`, measured in Hz, defines how many samples of the signal are taken per second. The `bit depth` determines how many discrete levels each sample can be mapped to and is always a power of two (e.g., $2^8 = 256$ levels for 8-bit audio).

The number of quantization levels is determined by the bit depth of the audio signal. For instance, 8-bit audio can represent $2^8 = 256$ distinct amplitude values, while 16-bit audio can represent $2^{16} = 65{,}536$. A higher `bit depth` allows for finer resolution and reduces quantization error, which results in more accurate and natural-sounding audio.

4.3.2 Fourier transform

The Fourier transform is a mathematical technique that converts a signal from the time domain into the frequency domain. This is essential for analyzing the spectral components of a sound.

In digital signal processing, the Discrete Fourier Transform (DFT) is typically used, which is most commonly implemented using the Fast Fourier Transform (FFT) algorithm. The FFT enables real-time frequency analysis of sound by breaking it down into its frequencies. These can then be visualized using tools like spectrograms or frequency bars. (Müller, 2021)

Figure 4.8 illustrates this transformation: on the left, a waveform composed of multiple sinusoidal signals is shown. These underlying frequencies are then extracted and represented in the frequency domain on the right.

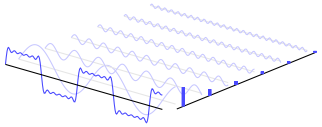


Figure 4.8: An example of a Fourier transform showing both the time and frequency domain. Adapted from Jake, 2013.

4.3.3 Digital filters

We do not go into depth about how digital filters work, but in essence, they produce the same kind of output as analog filters. Since our application runs on the web, we can make use of the Web Audio API, which provides powerful built-in digital filters and tools for working with sound directly in the browser. This allows us to implement a lot of audio transformations very easily [1]. See Figure 6.1 for a comprehensive list of all the audio API's that were used in this product.

4.4 Visualizing audio

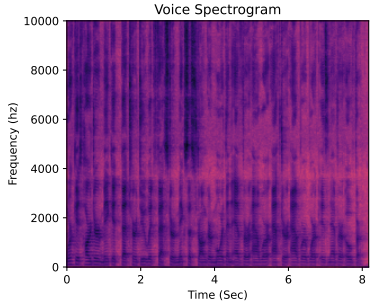
Audio visualization is the process of converting sound into images. This is used in fields like education, music, and art. Below are some common types, each with an image to show how they look.

4.4.1 Waveform

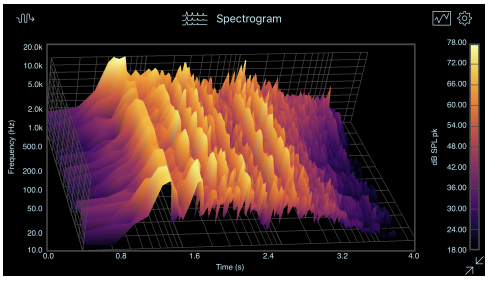
Waveform visualization displays the amplitude of a sound signal over time, providing a direct representation of its volume changes and rhythm. This type of visualization is particularly useful for identifying patterns such as beats, silences, or sudden changes in loudness. It is the same type of waveform that was used in Section 4.2 to demonstrate the effects of different types of filters. Waveforms are commonly used in audio editing software, where they allow users to precisely cut, align, or manipulate audio tracks based on their visual representation.

4.4.2 Spectrogram

A spectrogram is a visual representation of the frequency content of a sound over time. It uses a heatmap where the x-axis represents time, the y-axis represents frequency, and the color intensity indicates the amplitude or energy of the sound at a given frequency and time.



(a) A basic 2D spectrogram visualization adapted from Tralie, 2021.



(b) A basic 3D spectrogram visualization adapted from LLC, 2020.

Figure 4.9: Comparison of 2D and 3D spectrogram visualizations.

[1] See the Mozilla documentation for more information.

The 2D spectrogram (Figure 4.9a) provides a straightforward view, making it easy to analyze patterns in speech, music, or other complex sounds. On the other hand, the 3D spectrogram (Figure 4.9b) adds depth by visualizing the same data in three dimensions, offering a more dynamic perspective that can help in understanding the relationships between time, frequency, and amplitude more intuitively. Both formats are useful, with the choice depending on the specific application or user preference.

4.4.3 Bar visualizer

The bar visualizer shows how sound is made up of different frequencies by turning audio into moving bars. It uses Fast Fourier Transform (FFT) to split the sound into frequency bands and shows their strength as the height of each bar. This makes it easier to see how the sound changes over time and helps users understand the structure of audio in a visual way, as seen in Figure 4.10.

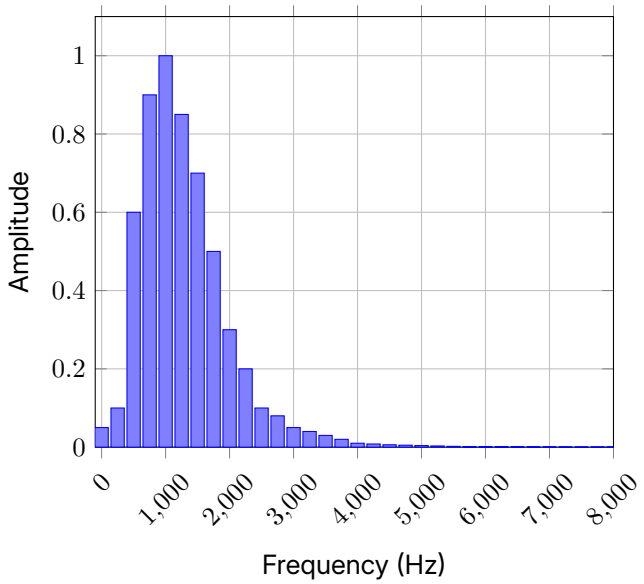


Figure 4.10: A typical bar visualizer with different frequency bands.

4.5 Existing audio visualization applications

Audio visualization is widely used in various domains, ranging from artistic performances to educational tools and commercial software. For instance, Adiletta and Thomas (2020) explores expressive and abstract visualizations that reflect the feeling of sound rather than just its structure, showcasing how audio can be transformed into visually engaging art.

Beyond artistic applications, audio visualizations are commonly found in music players, such as the visualizers in media software like VLC or Winamp, which display waveforms or frequency bars in real time. In education, spectrograms and waveforms are used to teach concepts in acoustics and signal processing. Additionally, interactive installations in museums often use audio visualization to engage visitors by allowing them to see how their voices or sounds interact with visual elements. Aside from that, it is used as a tool in the work field. One example could be sound designers that need to edit sounds using their waveforms. They do this for movies, games or artists' songs. Another field of employment that works with audio visualization is data science. The data from the audio can be used for speech analysis or for measurements in either a natural or an urban environment.

While this project focuses on education and inspiring high school students to explore STEM, these existing examples demonstrate how visualizations can be both functional and emotionally engaging. Our goal is to combine artistic approaches with educational content to create a tool that is not only informative but also fun and interactive.

Chapter 5. Design

The design of the app was created using Figma. Each page was constructed as part of a low fidelity prototype at the start of the project before implementation. This design posed as a guide to how the app's screens should be structured and connected to each other. This design phase also helped identify the key components of the project. This chapter presents the design and explains the process behind it.

5.1 Navigation

After we finalized which pages we wanted to have in the app, we started designing the navigation. The navigation was made to be as simple as possible to increase usability, so it is easy to understand for the main user. The navigation bar, placed at the bottom of the screen, allows users to switch between the home page, audio page, record modal, games page, and profile page. The finalized design is shown in Figure 5.1a.

5.2 Header

The header design, shown in Figure 5.1b, provides a consistent and user-friendly interface across the app. On the left side, the app's name is displayed, ensuring easy identification. On the right side, an information icon is available, offering help. When clicking on the info icon, information about the page will be displayed or, when on the home page, a walkthrough of the app will be started.

(a) Bottom navigation bar. **(b)** Top header.

Figure 5.1: Static app navigation.

5.3 Home page

The home page is the first screen the users will see when they open the *SonicViz* app. Its goal is to provide a welcoming introduction and an easy guide for users through the core features of the app.
At the top of the screen, a welcome message introduces the user and briefly describes the app, which includes making, recording, and visualizing audio. Below the message, the page visually outlines the three main steps:

1. **Step 1: Make music**
   Users can begin by either selecting from a variety of audio samples or creating their own music.
2. **Step 2: Record your music**
   A large, centrally placed red microphone button invites users to record their own audio input.
3. **Step 3: Visualize your music**
   Once audio is selected or recorded, users can generate waveform visualizations using the app's built-in tools.

This design ensures that the home page is not only visually appealing but also functional, guiding the users through the app's main features, as shown in Figure 5.2.

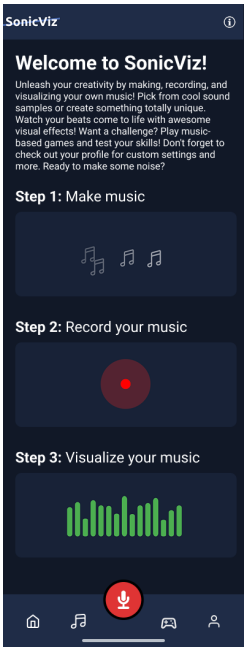


Figure 5.2: Home page.

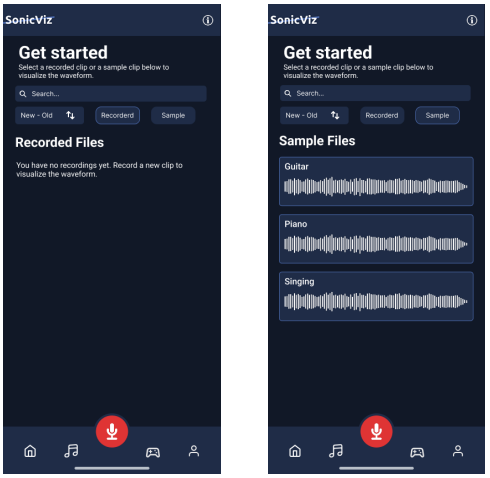
5.4 Audio page

The Audio Page is a key page of the *SonicViz* application, designed to let users easily access and manage their recording or sample recordings. The interfaces in Figure 5.4 show a clean, minimalistic design, optimized for usability on mobile devices. The page consists of two main views selectable by toggle buttons, **Recorded** and **Sample**:

- **Recorded**: This view shown in Figure 5.3a, displays the user's own recorded audio files. The User can select a recording to visualize it or delete it.
- **Sample**: This view shown in Figure 5.3b, displays a selection of sample audio files provided by the app. The user can select a sample to visualize it.

Both views maintain a consistent layout, including:

- A search bar at the top for easy navigation through the audio files.
- A sort option to arrange the audio files by name or date.
- A toggle button to switch between the recorded and sample audio views.
- A list of audio files, with the option to select and visualize it.



(a) Recorded audios selected.

(b) Sample audios selected.

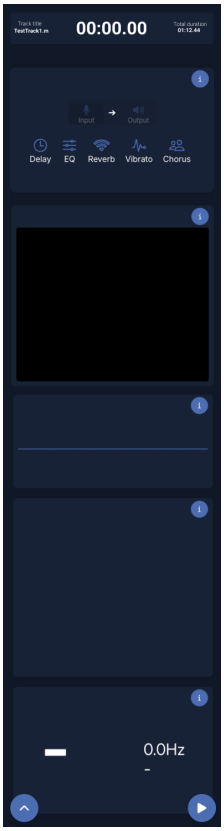
Figure 5.4: Audio page with with recordings and samples.

5.4.1 Visualization modal

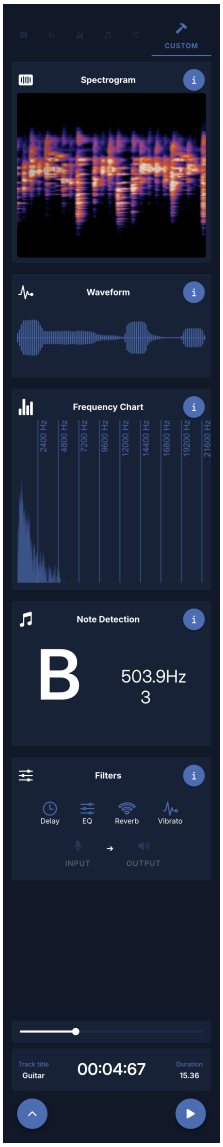
The visualization modal is a crucial component of the app, allowing users to visualize their audio recordings. It provides a range of visualizations and filters to enhance the audio experience. The old design of the visualization modal is shown in Figure 5.5a and the new design in Figure 5.5b. The visualization modal is designed to provide the users the visuals for the audio and filters to modify the audio, shown in Figure 5.5a.
At the top of the screen the name of the audio file, the timestamp and the total duration of the audio are displayed. Below there is a bar in which you can scroll to a different timestamps.
All components within the modal are draggable, and can also be added and removed, this allows the user to customize the screen to their preferences. Every graph or filter has info button on the top right corner. Clicking this button opens a modal with information about the graph or filter. At the left bottom there is a floating action button (FAB), when it is clicked it opens a two or three buttons modal, depending on the audio file.

- **Back to audio files:** This button takes the user back to the audio page.
- **Change layout:** This button opens a modal with the option to change the layout of the visualization modal.
- **Delete audio:** This button deletes the audio file if possible. This button is only available when the user is in the recorded audio view. If the user is in the sample audio view, this button will not be displayed.

Lastly, at the bottom-right corner, there is a play/pause button for controlling audio playback.



(a) Old design.



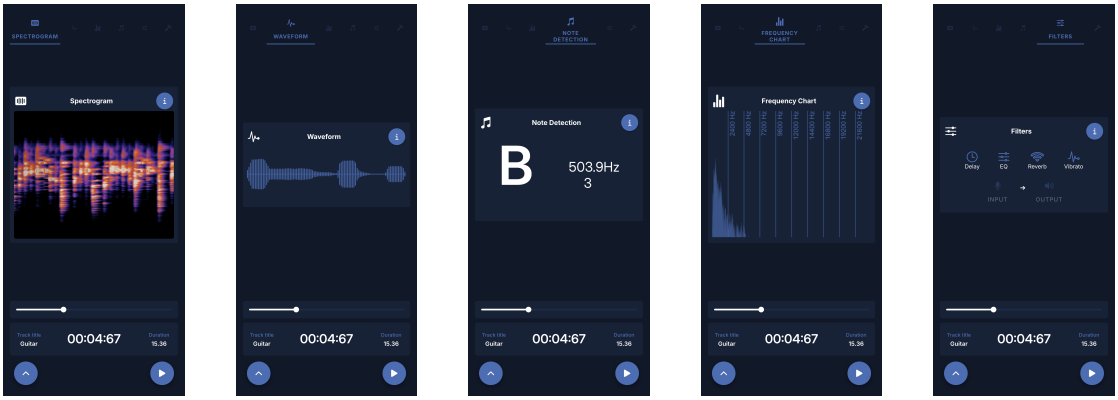
(b) New design.

Figure 5.5: The designs of the visualization modal.

5.4.2 Updated visualization modal

After implementing the visualization modal, we discovered that the design was not optimal for the user experience and usability. The initial design was too cluttered because of the amount of things that were displayed at once. There was an option to customize this but at the first glance this was not clear to the user. The new design makes use of tabs to display the different visualizations and filters. This allows the user to select the visualization they want to see and the filters they want to use. The tabs are displayed at the top of the screen and the user can switch between them by clicking on the tab or swiping left or right. The tabs are:

- **Spectrogram:** This tab displays the spectrogram of the audio file, as shown in Figure 5.6a.
- **Waveform:** This tab displays the waveform of the audio file, as shown in Figure 5.6b.
- **Note:** This tab displays the note being played in audio file, as shown in Figure 5.6c.
- **Frequency chart:** This tab displays a frequency chart of the audio file, as shown in Figure 5.6d.
- **Filters:** This tab displays a list of filters that can be applied to the audio file. The user can select the filter to use and adjust its parameters. this tab is shown in Figure 5.6e.
- **Custom:** This tab displays similar to the old modal design but with a cleaner look. This is the last tab so it is more of a option for the user to use if they want to. This tab is shown in Figure 5.5b.



(a) Spectrogram tab. **(b)** Waveform tab. **(c)** Note tab. **(d)** Frequency chart tab. **(e)** Filters tab.

Figure 5.6: Visualization modal tabs.

5.5 Record modal

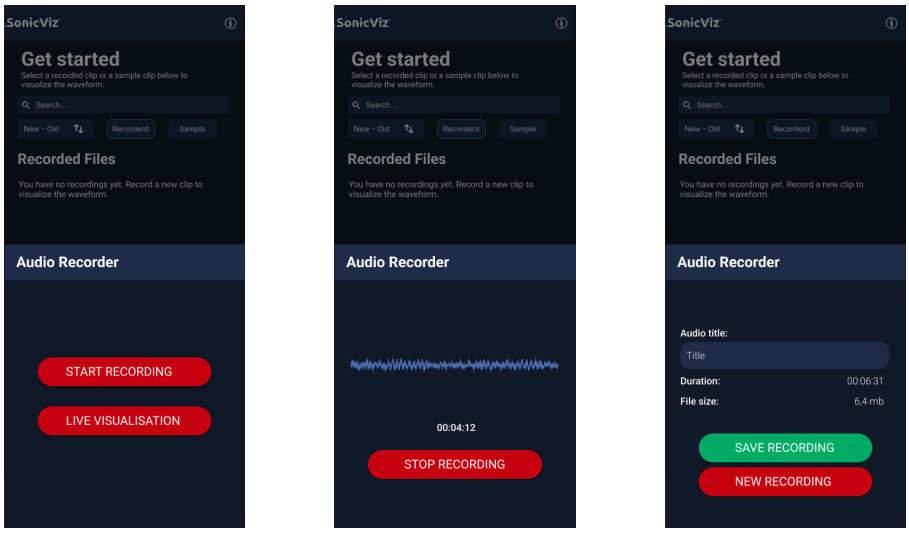
The Record Modal provides users with a simple and intuitive interface to either record or visualize live audio. It consists of three main screens, shown in Figure 5.7:

5.6 Record modal

The Record modal is designed to provide users with a simple and intuitive interface to record or visualize live audio. The record modal is divided into three main sections:

1. **Start Screen:** The first screen (Figure 5.7a) allows users to choose between two options:
   - **Start New Recording** — Initiates a new audio recording.
   - **Live Visualization** — Displays a live visualization of the audio input without recording.
2. **Recording Screen:** If the user selects **Start New Recording**, the modal transitions to the recording screen (Figure 5.7b) and immediately begins recording. This screen includes:
   - A waveform visualization of the live audio input.
   - A timer showing the duration of the recording.
   - A stop button to end the recording.
3. **Save Screen:** After stopping the recording, the modal transitions to a save screen (Figure 5.7c), which includes:

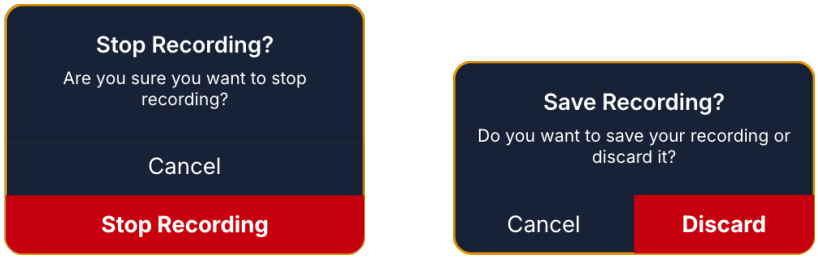
- An input field to name the recording.
- A display of the audio duration and file size.
- A save button to store the recording.
- A new recording button to delete the current recording and return to the start screen.



(a) Start screen. **(b)** Recording screen. **(c)** Save screen.

Figure 5.7: All screens of the record modal.

There are two different alerts used in this modal, these are designed to check if the user really wants to this action. These alerts open when trying to close the modal when the user is in a procedure. The first alert is to ask the user if it really wants to stop their recording and so delete their recording (Figure 5.8a). The discard alert is to ask the user if they want to delete discard their recording (Figure 5.8b).



(a) Stop alert. **(b)** Discard alert.

Figure 5.8: All app alerts.

5.7 Games page

The Games Page allows users to select a game to play from a list of available options. Each game is represented by a card displaying the game's name, a short description, and the user's high score for that game. The layout of the Games Page is shown in Figure 5.9a.

Currently, two games are available: *Guess the Note* and *Pitch Perfect*. Each game consists of two screens: an explanation screen and the actual game screen.

5.7.1 Guess the Note

Guess the Note is designed to helps users develop their ability to recognize musical notes by ear in an intuitive and engaging way.

The first screen provides a brief explanation of the game, as shown in Figure 5.9b. Users can start the game by clicking the "Start" button.

The second screen is the main gameplay screen (Figure 5.9c) and includes the following components:

- **Score display** — Shows the current score and high score. The high score increases when multiple notes are guessed correctly in a row.
- **Instrument selector** — Allows users to choose the instrument on which the notes are played.
- **Play button** — Plays the note to be guessed.
- **Guesses left indicator** — Shows the number of remaining guesses for the current note (maximum of three guesses per note).
- **Note counter** — Displays how many notes have been attempted.
- **Note grid** — A grid of possible notes. Users select a note to make a guess. Correct guesses are marked green, while incorrect ones are marked red. Upon a correct guess, a modal appears asking if the user wants to proceed to the next note.

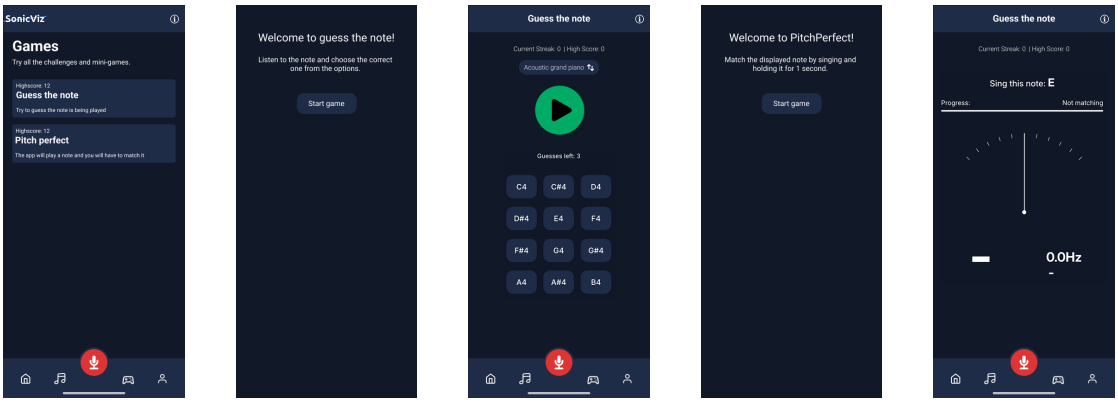
5.7.2 Pitch Perfect

Pitch Perfect is designed to help users learn how to sing musical notes accurately.

The first screen explains the rules and objectives of the game (Figure 5.9d). Users can start the game by clicking the "Start" button.

The second screen is the gameplay interface (Figure 5.9e) and includes the following components:

- **Score display** — Shows the current score and high score. Like in the previous game, the high score increases with consecutive correct performances.
- **Target note** — Displays the note that the user needs to sing.
- **Progress bar** — Indicates how close the user's pitch is to the target note. To complete a challenge, the user must match the pitch for a total of one second.
- **Pitch meter** — Shows the note the user is currently singing.
- **Note counter** — Tracks how many notes have been attempted.



(a) Game page. **(b)** Guess the note explanation page. **(c)** Guess the note game page. **(d)** Pitch perfect explanation page. **(e)** Pitch perfect game page.

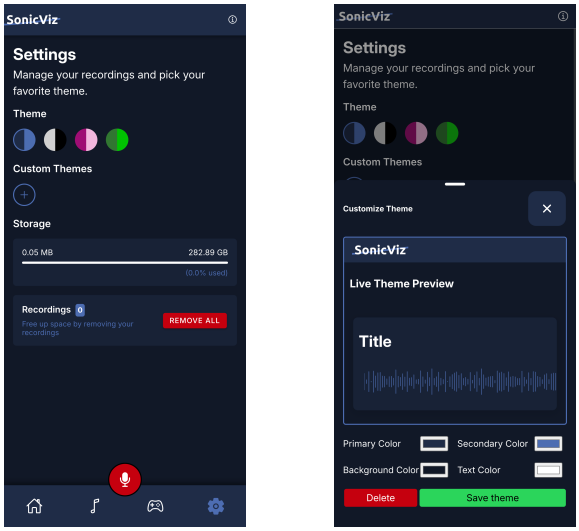
Figure 5.9: Game page and game screens.

5.8 Settings page

The settings page enables users to switch between themes and create their own custom themes (Figure 5.10a). There are four built-in themes available:

- **Dark Theme**: The default theme of the app, featuring dark blue tones with white text.
- **Light Theme**: A light theme with a white background and dark text.
- **Pink Theme**: A theme with a pink background and white text.
- **Green Theme**: A theme with a green background and white text.

Users can also create and manage up to four custom themes. When creating a new theme, a modal appears allowing users to customize their color preferences. Once created, users can switch between any of their custom themes.
Additionally, the settings page displays storage usage, showing both the total storage available and the amount used by the app. It also includes a button to delete all stored recordings.



(a) Settings page.

(b) Custom theme modal.

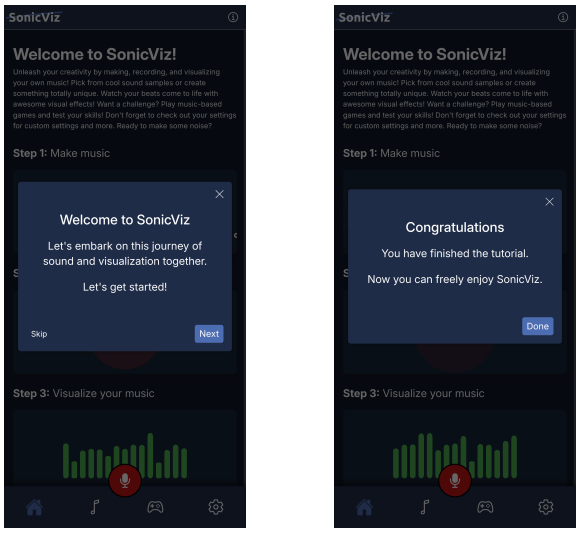
Figure 5.10: All screens of the settings page.

5.8.1 Theme edit modal

The theme edit modal allows users to create and preview a personalized theme, as shown in Figure 5.10b. Within this modal, users can choose their preferred primary, secondary, background, and text colors. Any changes made are reflected in a live preview, giving immediate feedback on the appearance of the newly selected theme.

5.9 Walkthrough

The walkthrough is designed to guide users through the app's features and functionalities.This is especially useful for new users who may not be familiar with the app's layout and options. The walkthrough is automatically initiated with opening the app for the first time, if the user wants to do it for a second time it can be done by clicking on the info button on the home page. The user can then click "Next" to proceed through the walkthrough, which consists of several screens highlighting different features of the app. Each screen includes a short description and a highlight of some part of the screen, illustrating the feature. The walkthrough start screen is shown in Figure 5.11a and the walkthrough end screen in Figure 5.11b.



(a) Walkthrough start screen.

(b) Walkthrough end screen.

Figure 5.11: All the walkthrough screens.

Chapter 6.

Implementation

When implementing the design, we faced several problems to solve and decisions to make. In this chapter, we will detail what issues we ran into and the solution we chose. Additionally, we will go over key choices we made regarding implementation.

6.1 Mobile app development

One of the non-functional requirements is to create a mobile app. It has to be compatible with both Android and iOS. We assumed our app will not have large performance requirements, and therefore we chose the Ionic framework. Ionic bundles web apps as mobile apps, and therefore allows you to use one code base for multiple platforms. This allows us to realistically build the app for two platforms and also give the option to deploy on the web.

6.1.1 Deployment

This app can be deployed in two ways at the moment, which can depend on the target platform. Firstly, it is possible to deploy SonicViz as a Progressive Web App (PWA)[1] allowing any device to access the website version of the app on their preferred browser. Aside from the PWA, the app can also be shared through a pre-compiled `.apk` file, so the user can download the application from this file. This, however, is not possible for iOS users as iOS does not support installing `.apk` files. Ionic does support deploying the app into a format that can be compiled by XCode (the development platform of iOS and other apple related products). This means that it is possible to deploy the app on the app store but it does require enrollement in the Apple Developer Program[2] which costs 99 USD at the time of writing.

6.2 Development tools

In addition to Ionic, we use React with TypeScript. React is one of the options for a default template for Ionic, which we used. The template also comes with the bundler Vite[3], which adds a bunch of features like Hot Module Replacement to make the web development experience much nicer and bundles our entire app into a smaller package by using tree-shaking, which is important considering the substantial amount of packages that are installed. We use plain CSS for styling our website.

6.3 Web deployment

The website is deployed on Cloudflare Pages[4]. This is a static file host that is free to use, like GitHub Pages. We scrambled together money for a domain, and since we are with five people we thought this

[1] More info about PWA's can be found on the Mozzila web docs.
[2] More information about the program can be found on the Apple Developer website.
[3] More info about Vite can be found on vite.dev.
[4] More info about Cloudflare Pages can be found on pages.cloudflare.com.

was feasible. In the end, we ended up spending a whole €0.01 for https://sonicviz.nl. The domain's nameservers are Cloudflare's, and it integrates well with Cloudflare Pages as it automatically adds the necessary DNS entry for us. Their system automatically builds the latest commits from our production branch, prod. This means we only have to update prod with the latest master branch on the GitHub repository and Cloudflare Pages builds and deploys our website automatically within minutes.

6.4 Web technologies

This project relies on modern web-based audio APIs to facilitate recording, playback, real-time audio processing, and visualization. Figure 6.1 is an overview of the main technologies used and their respective roles in the project. Each API is linked to its respective Mozilla MDN documentation article.

API	Purpose
AudioContext [MDN]	Core object that manages audio operations. It serves as the root of the audio processing graph and is responsible for creating and controlling audio nodes.
AnalyserNode [MDN]	Provides real-time frequency and time-domain analysis used for waveform and pitch visualization.
DelayNode [MDN]	Introduces delays into the audio signal path. Used to create the vibrato effect.
GainNode [MDN]	Controls volume levels.
ConvolverNode [MDN]	Applies impulse responses to simulate realistic reverb effects.
BiquadFilterNode [MDN]	Implements frequency filtering for equalizer effects.
AudioBuffer [MDN]	Holds audio data in memory, enabling playback or signal analysis. Also used for decoding audio files and reverb responses.
decodeAudioData [MDN]	Converts binary audio file data into an AudioBuffer.
getUserMedia [MDN]	Accesses the user's microphone for capturing live audio input. Allows for real-time analysis and recording.
MediaRecorder [MDN]	Records audio from a MediaStream into a series of Blob objects. Enables saving or playback of user recordings.
Blob [MDN]	Stores binary audio data, such as recorded audio. Can be converted to base64 or streamed to a playback component.

Figure 6.1: List of all web APIs used.

6.5 Libraries

In addition to the standard Web APIs, we used several open-source libraries that can be found in Figure 6.2. These libraries implemented (parts of) functionality that we wanted to implement or helped us solve some problems. Each library is linked to its repository or website. Some libraries were standard with Ionic and (such as the native mobile integrations from Ionic Capacitor) and are not listed here.

Library	Purpose
Ionic Storage [GitHub]	An official Ionic library. This wraps IndexedDB and localStorage and automatically chooses the best option available. Since we use this to store recorded audio files locally, Ionic Storage will use IndexedDB since localStorage does not allow a lot of data to be stored. Essentially, this is an IndexedDB wrapper.
Ionicons [ionicons.com]	An official Ionic library. This library contains many icons in various styles and is licensed under the permissive MIT license. We use these icons all over the app.
pitchy [GitHub]	pitchy is a pitch detection library that we use to detect the current pitch in Hz of the input audio. We can transform this pitch to a note as well that we show or use in various places (visualization, Pitch perfect).
React Joyride [GitHub]	React Joyride is used for the guided tour throughout various pages of our app.
smplr [GitHub]	smplr is used to generate audio based on a given note for the Guess the note game. It includes multiple instruments that are selectable in the game, such as a piano, various guitars, etc.
dnd kit [dndkit.com]	dnd kit provides drag and drop functionality for our app. This is used for adding or removing filters or reordering visualizations in the Custom tab.
Motion [motion.dev]	Motion is an animation library for web apps. We use it in the Guess the note game and to animate the sliding notes on the homepage.
tiny-invariant [GitHub]	tiny-invariant is a small library used for falsy (false, undefined, null) checks. If a value is falsy, it throws and stops further code execution. It makes the code a bit cleaner for us by not having many if statements.
standardized-audio-context [GitHub]	This library provides a cross-browser AudioContext interface. Some browsers, namely Safari, have some small issues or mismatches with the AudioContext used by Chromium-based browsers. This library handles this for us.

Figure 6.2: List of the libraries we used.

6.6 Visualizing

Since most of our visualizations are made with HTML Canvas, we need to draw our data in multiple different places. To do this, we use a single AudioContext and AnalyserNode that passes their data on to functions provided by the visualizations. These functions are executed all at the same time before the browser repaints by using the requestAnimationFrame function and passing a callback. The browser will call the callback function, which gathers data from the AnalyserNode that is then passed on to the drawing function of each visualization. The drawing functions are made to be efficient, for example the spectrogram only draws the input data (one time unit) instead of redrawing the entire spectrogram, which would be considerably slower. When all the drawing functions are run, another animation frame is requested and the cycle repeats.

6.7 Permissions

To use the microphone on mobile devices, the app needs to request permissions. These permissions need to be added to a manifest file for Android platforms, called AndroidManifest.xml. At first, we used only Listing 6.1.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Listing 6.1: Android record audio permission

However, this did not work. The phone asked for permission but it did not allow us to get the microphone input. After some research, we found that we also need another permission Listing 6.2.

```
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

Listing 6.2: Android modify audio settings permission

For iOS, we need to add a key Listing 6.3 to the `Info.plist` file. This key is used to inform the user why the app needs access to the microphone.

```
<key>NSMicrophoneUsageDescription</key>
<string>Yes</string>
```

Listing 6.3: iOS modify record audio permission

6.8 Limitations

During the development of SonicViz, a few limitations were discovered regarding our implementation. These limitations prevent reaching optimal results or can even completely rule out certain possibilities. These limitations are currently obstructing the development and will remain. It is important to acknowledge these obstacles in order to make progress.

6.8.1 Concurrent microphone access

In mobile operating systems, only one app can use microphone input simultaneously. This means that you cannot be on a voice call (e.g., screen sharing SonicViz while in a call) and simultaneously use the app. This is not a problem we can solve. Only certain privileged or built-in accessibility services such as Google Assistant or voice commands can use the microphone at the same time as another app.

6.8.2 Platform differences

Different platforms will handle the same application in different ways when it comes to aspects like permissions, optimization, design and more. Similarly, do various web browsers have such differences on these areas. This can cause inconsistencies in how the user experiences their use of the application. The most prominent difference is the changes in design, button shapes or notifications will look different for Apple devices compared to Android. Another dissimilarity is that there are limitations on how many audio output streams a device can handle depending on the hardware or operating system (Android Open Source Project, 2025).

6.8.3 Device optimization

Compared to other applications, SonicViz does not require much processing power from the device. Even so, there can be noticeable changes in performance when using a less suitable device. Older devices are generally slower and have less capabilities. When it comes to live visualizations, any delay will negatively impact the user's experience. In order to achieve real-time feedback in the visualization, optimization is certainly required, especially for older devices. Extending the scope of device compatibility to make this app suitable for as many devices as possible is a desired prospect. Unfortunately, it needs to be addressed that not all devices will be able to achieve optimal results.

6.8.4 Dutch smartphone use in high schools

As of 01-01-2024, the use of smartphones and other non-educational smart devices during school hours is no longer permitted in Dutch high schools. However, this regulation makes an explicit exception for devices used for educational purposes. (Rijksoverheid, n.d.) This creates an important consideration for our application. While the technical implementation and the intended educational value are our key design points we have not yet thought of a clear strategy to position the app as an educational tool within the school environment. It will be necessary to come up with arguments for its educational relevance. These arguments should be clearly communicated to teachers and school administrators so that they can opt to use of the app. Although this was outside the scope of our current project, we identify it as a crucial point for future work.

Chapter 7.

Testing and validation

This chapter outlines the testing strategies used in the development of the audio visualization app. The goal of the testing process is to ensure that the application functions correctly across various environments and maintains stability throughout its development cycle. We employed a combination of automated and manual testing methods to validate the app's functionality and compatibility with different operating systems and browsers.

7.1 Automated testing

Testing is performed using a combination of End-to-End (End-to-End) and component tests introduced by the Cypress testing framework. These tests are continuously developed following the Test-Driven Development (TDD) strategy. Due to the application being frontend heavy, the testing plan integrated frontend testing. This frontend testing verifies that the user interface behaves correctly, which is increasingly more important when using a library like React, which introduces more complex state logic in the frontend.

7.1.1 Cypress

For frontend testing, Cypress was used for its End-to-End testing capabilities and simple setup process. Cypress focuses on frontend testing and comes with extensive tools. The End-to-End tests are a form of functional tests where actions are performed on the site like a real user would. Using a set of commands, you can guide the test to click UI elements, validate that content is in view or that the state of the website is correct. The tests run isolated in a headless browser, making them accurate and fast. Cypress also allows for browser selection during the tests, as well as specific configurations such as the viewport. This enables highly specific testing environments to help that the application works consistently across different devices.

Graphical interface

Additionally, Cypress comes with a graphical user interface which provides a real-time view of the tests being performed, as well as the capabilities to take screenshots and recordings during failed tests. These screenshots and recordings can be accessed after the tests have run, both on GitHub and when run locally. This eases the testing process greatly, since the exact performed steps can be retraced, easing the process of altering the tests or discovering when the application behaves unexpectedly. This can be especially useful when running the tests in an environment such as Github Actions, allowed for an earlier discovery of the limitations of Cypress.

Custom commands

Cypress allows for the creation of custom commands. These commands are a set of actions that can be called from any End-to-End tests. This is useful for actions which need to be done during many tests, as it will reduce code duplication without reducing the testing capabilities.

```
Cypress.Commands.add("record", (recordingName: string = "test-recording") => {
    cy.get('[data-test-id="record-button"]').click();
    cy.get('[data-test-id="start-recording-button"]').click();
    cy.wait(500);
    cy.get('[data-test-id="stop-recording-button"]').click();
    cy.get('[data-test-id="recording-title-input"]').should("have.text", "");
    cy.get('[data-test-id="recording-title-input"]').type(recordingName);
    cy.get('[data-test-id="save-recording-button"]').click();
    cy.wait(300).then(() => {
        cy.get("body").click(0, 0);
    });
});
```

Listing 7.1: Custom command to perform a recording.

```
it("records audio", () => {
    cy.record("Recording-name");
});
```

Listing 7.2: Using the command in a test.

In the end, a wide range of custom commands were created for actions such as opening a visualization modal, selecting options from dialogs, and starting a game. These commands allowed the core actions to be tested across many different configurations.

Stubbing objects

For tests to be fair and consistent certain methods and objects need to be stubbed. Stubbing involves encapsulating an object or class and overriding its functions with a standardized output. This was necessary for, for example the Guess the note game. Normally the note is randomized, but by stubbing the global Random object, as showcased in Listing 7.3, the ”random” note can be selected as such that it is always the same note, making the environment predictable and testable.

```
Cypress.Commands.add("stubCorrectNote", note => {
    const NOTES = ["C4", "C#4", "D4", "D#4", "E4", "F4",
                   "F#4", "G4", "G#4", "A4", "A#4", "B4"];
    const noteIndex = NOTES.indexOf(note);
    assert(noteIndex !== -1, `Note ${note} not found in the list of notes`);

    cy.window().then(win => {
        cy.stub(win.Math, "random").callsFake(() => noteIndex / NOTES.length);
    });
});
```

Listing 7.3: Stubbing the Math.Random object.

7.1.2 End-to-End testing

The majority of tests were done as End-to-End tests. End-to-End testing is ”an approach to testing that simulates real user experiences to validate the complete system” (Schmitt, 2024), meaning that all parts of the application are tested together.

A complete list of End-to-End test suites is described in Figure 7.1 along with their status. There were End-to-End test suites per page, and per action which is available across different pages. For example, the *Guess the note* End-to-End test plays the game in its entirely and navigates between pages to validate the state is updated accordingly. Each test suite contained a number of test cases which would test both the happy path but also complex actions to cover edge cases. Across the 9 test suites, a total of 65 test cases are tested.

Test	Purpose	Result
Core Tests	Verifies that all major pages (Home, Audio, Games, Settings) can be navigated to via the main navigation bar and that the correct URLs load as expected.	Passed
The Record component	Tests the full recording functionality across the app. Verifies that recordings can be made from any page, titles are required before saving, and proper dialogs are displayed. Also confirms that the live visualization modal opens correctly.	Passed
Walkthrough	Tests the in-app tutorial system powered by React Joyride. Verifies that the tutorial starts automatically on first visit, can be skipped or restarted, supports step navigation, and displays beacons correctly.	Passed
Visualization Modal	Validates the sample and recorded audio visualization modals. Tests playback controls, timeline display, segment navigation, and the ability to add or remove visualization cards. Also confirms editing mode behavior and filter-specific restrictions.	Passed
Guess the note	Ensures the game starts, displays correct information, and handles user interactions (correct/incorrect guesses, streak tracking, high score updates). Tests the ability to play the note and select different instruments. It also checks if the game state is saved and retrieved correctly.	Passed
Pitch perfect	Validates the functionality of the "Pitch Detector" game, including showing instructions, detecting correct and incorrect pitches, updating the score, maintaining high scores across sessions, generating new notes, and ensuring progress resets correctly when the pitch changes.	Passed
Games page	Verifies the core functionality of the Games page. Tests the visibility of the info popover and the ability to launch both the note detection and pitch detector games via their respective buttons.	Passed
Settings page	Validates the Settings page functionality. Tests theme selection, creation, editing, and deletion for both built-in and custom themes. Also verifies the storage section updates after saving or deleting recordings.	Passed
Audio page	Validates the audio recordings page. Confirms basic functionalities, sample recordings section, and handling of empty states. Verifies recording creation, search and sorting functionalities.	Passed

Figure 7.1: List of all the End-to-End test suites.

7.1.3 Component testing

Another functionality of Cypress is component testing, which involves writing smaller tests that focus on individual components. For a component to be tested, it needs to be mounted. However, most of the components created were using Ionic components, which require additional setup to render correctly. Additionally, these Ionic components needed to be configured as well to create isolated and clean tests.

While this would be a good approach to increase code coverage and have good test cases, the decision was made to primarily focus on writing End-to-End tests, as writing these test types would be faster.

7.1.4 Results

All test suites successfully passed on both Chrome and Firefox. Cypress was used to generate the code coverage report following the execution of all tests. The complete results can be found in Appendix B, with a summary of the key findings provided below.

Statistic	Result
Statements	78.4%
Branches	70.9%
Functions	82.6%
Lines	78.4%

Table 7.1: Summary of the code coverage report.

Overall the test results are on an acceptable level. Certain functionalities such as dragging and dropping are difficult to perform using Cypress, and could therefore not be automatically tested. Additionally there are some branches which are for error handling if the user is running the application on an unsupported browser. To further increase the test coverage component test would need to be written to test the components which are not UI heavy.

7.1.5 Github actions

To automate the testing process, GitHub actions were used to test branches automatically. A workflow (which can be found in Appendix C) was started after opening a pull request into the main branch. This workflow will attempt to build the project and run the tests on the current branch. Testing before merging into the main branch is crucial, especially with multiple team members working on different functionalities. By ensuring that the main branch always builds and passes tests, it remains a reliable source for merging, preventing one member's work from being blocked by another's issues.

The original workflow testing plan involved running Cypress tests across six configurations: the three major browsers (Chromium, Firefox, and WebKit), each tested on both desktop and mobile viewports. For each configuration, both End-to-End and component tests were executed. These configurations were run in parallel to reduce the overall runtime.

After the Cypress tests complete, the results are uploaded and accessible via the GitHub Actions page. If a test fails, Cypress provides a screenshot of the failed execution. While Cypress also supports video recording of test executions, this feature remained disabled to reduce overhead as the Cypress graphical interface provided the same insights. Once the workflow finishes, any failure screenshots can be downloaded to help diagnose the exact state of the application at the time of failure.

7.1.6 Limitations

While Cypress made writing functional tests easier, there were still limitations that came with using it. However, most of these limitations came from the browsers in which Cypress was operating. Besides the difference in browsers, there was also the difference in operating systems, which caused inconsistencies. Some members were running Windows x86, others Mac OS and the GitHub runner was using Linux. The combination of these variables, along with the frail nature of functional tests, made testing tricky.

Launch options

The heavy dependency on the microphone and the WebAudio API caused problems for both Firefox and WebKit. For chromium based browsers, a fake input device is created, which could be used during the tests. For Firefox this had to be enabled by setting launch options to start a fake stream used for input and the disabling of permissions.

```
on("before:browser:launch", (browser = {} as Cypress.Browser, launchOptions) => {
    if (browser.name === "firefox") {
        // Set Firefox preferences for fake media stream
        launchOptions.preferences["media.navigator.streams.fake"] = true;
        launchOptions.preferences["media.navigator.permission.disabled"] = true;
    }
    return launchOptions;
});
return config;
```

Listing 7.4: Launch options for Firefox in Cypress config.

Testing on WebKit

For WebKit the tests were even more problematic, causing us to abandon WebKit testing entirely. The issue lies with WebKit not having launch options to disable the permissions needed. This causes any microphone related test to wait for permissions which are never given, causing it to fail. Due to WebKit being run via Playwright [1] within Cypress it did not allow complete control over the browser instance, limiting the configuration options.

7.2 Manual testing

Manual testing was conducted to verify the application's compatibility across various operating systems and browsers. The following configurations were tested[2]:

- MacOS 15.4
  - Safari
  - Google Chrome
- IOS 18.4
  - Safari
- Windows 11
  - Google Chrome
  - Firefox
- Android 8
  - Google Chrome
- Android 15
  - Google Chrome
  - Firefox

Manual testing was performed regularly to also quickly test new features on a more insubstantial level. This contributes to a trial and error approach we sometimes would use to find the best solutions or approaches. Thoroughly navigating and using the application has been done in order to determine any weak points or inconsistencies as well as test out the flow of actions the user would take.

[1] More info about Playwright can be found on the Playwright site.

[2] This does not imply that the application runs exclusively on these configurations. Instead, it ensures that the application functions perfectly with these combinations. Other configurations may also work, but due to time constraints, they were not tested

Chapter 8.

Discussion

This chapter, will reflect on the outcomes of our project and identify opportunities for improvement and possible further development. While the current version of the app successfully meets the requirements and goal of introducing high school students to computer science in an engaging and playful way, there is still room for expansion and refinement. We explore how the user experience, educational impact, and overall functionality could be enhanced. After which we will reflect on the project as a whole, including the process, team dynamics, and personal growth.

8.1 Future work

This project managed to incorporate all the important requirements in order to allow the high school students to explore the computer science field through entertainment. However there are still some points of improvement that can be worked on. In this section any future ideation to this project are presented.

8.1.1 Homepage enhancement

Currently, the homepage mainly contains static text. Improving the homepage by making it more dynamic and visually appealing could spark user curiosity and encourage exploration. Possible enhancements include interactive animations, clear and engaging feature demonstrations, or dynamic visual examples that clearly illustrate what the app can do. Such enhancements aim to make users immediately curious and eager to try out the app.

8.1.2 Additional mini-games

Currently, we created two example mini-games to use with the user. We have more ideas for mini-games, and of course, there exist even more possibilities. These mini-games could help make the app even more fun and interactive, while also encouraging users to explore the app in different ways.

Challenges

Challenges throughout the app to encourage users to explore different features. These could be small tasks or missions, like “use each visualizer once” or “change the background color 5 times,” with a reward or badge system to keep users engaged.

Voice-controlled

A voice-controlled version of the Chrome no-internet dinosaur game or Flappy Bird. The idea is to use pitch or loudness to make the character jump or move, turning it into a fun way to interact with the microphone and sound input system.

Guess the instrument

A sound clip plays, and users choose the correct instrument from a list. This could be expanded with levels of difficulty, hints, and more instrument types over time. This functionality can already be

implemented using only the smplr library mentioned in Figure 6.2, since this library can create tones from a huge range of instruments.

Rhythm-matching
A rhythm-matching game where users tap along to the beat of a song. The app would detect the timing of the taps and score the player based on accuracy, encouraging listening skills and rhythm recognition.

Sound memory
A sound memory game where users repeat sequences of sounds in the correct order. Similar to the classic "Simon Says" game, but with musical tones or real instrument sounds instead of lights, testing both memory and listening skills.

8.1.3 More analysis on music or sounds

The app could analyze the emotion, style, or tempo (BPM) of music using audio analysis libraries or machine learning models. Since we are working with Ionic and React, we would rely on JavaScript-compatible libraries and external APIs that can be integrated into the app. Some possible options include:

- `Meyda`[1] - A JavaScript audio feature extraction library compatible with the Web Audio API. Useful for extracting features like tempo.
- `TensorFlow.js`[2] - We can integrate pre-trained models or train custom models to detect emotion or style in music using spectrograms or audio features.

8.1.4 Input audio files

Users should be able to upload their own audio files (e.g., MP3, WAV) that were not recorded directly in the app. This opens up more flexibility and lets users analyze or play with any sound of their choice.

8.1.5 Instrument detection

To detect instruments in an audio clip, we can use machine learning models trained on labeled audio data. One practical approach involves using the Essentia.js[3] library, a JavaScript port of the Essentia audio analysis library, which provides tools for feature extraction that can be used for instrument classification. However, for more advanced and deep learning-based detection, it is common to use models trained on datasets such as the IRMAS[4] dataset, which contains labeled samples of various musical instruments.

8.1.6 Translation

Currently, the app is only available in English. Since the target audience includes Dutch high school students, translation features are important. Using internationalization (often abbreviated as i18n) libraries compatible with Ionic React, we could support Dutch and other languages in the future. A commonly used solution is `react-i18next`[5].

8.1.7 Field test

We expect the app could cause some disruption in a classroom due to sound and interaction. To address this, a test deployment in a real school environment could help reveal issues and areas for improvement.

- Testing in actual classroom environments can identify the strengths and weaknesses of the app in practice, such as whether noise becomes an issue.
- Interviews with high school teachers can offer valuable feedback on classroom management, app usability, and educational impact.

As also mentioned in Section 6.8.4, it is important to also take into account the Dutch law in this context.

[1] https://github.com/meyda/meyda
[2] https://www.tensorflow.org/js
[3] https://github.com/mtg/essentia.js
[4] https://www.upf.edu/web/mtg/irmas
[5] https://react.i18next.com

8.1.8 Playwright for testing

The current used testing framework is Cypress, and throughout the development phase the limitations of Cypress with regards to browser configuration caused us to deviate from our test plan. A good alternative is to use Playwright[6] as a testing tool instead, as it offers full cross-browser support, meaning that it should be possible to run tests on WebKit. Furthermore, Playwright supports advanced features such as device emulation, enabling the testing of possible future application files. And while the user interface is not as user-friendly as what Cypress delivers, it will enable for better and more consistent testing.

8.2 Reflection

At the start of the project, we had many ideas and possibilities. After setting up the requirements and defining the project goals, we quickly moved into the design phase. In hindsight, this transition may have been a bit rushed. Once we began implementation, we encountered some challenges that should have been thought through better during the design stage. As a result, we had to make some changes to both the design and the implementation, which required additional time and effort.

Despite these hurdles, we managed to get everything working, and we are proud of the final result. Throughout this process, we gained valuable experience in designing, developing, and refining a mobile application. This project has taught us a great deal about the full development cycle, and we're satisfied with what we've achieved.

Looking ahead, one clear lesson to consider is the importance of spending more time to the design phase. Carefully considering the technical implications of our ideas early on would likely have saved us time and reduced friction later in the development process.

We also learned a lot about team collaboration and communication. Our team dynamic was positive, and we enjoyed working together throughout the project. Active discussions were held weekly to keep everyone up-to-date and feedback was always taken into consideration.

The communication with our supervisor/client was also a strong point. We had weekly meetings where we received valuable feedback on our progress. Her insights helped our development and improve the quality of the app.

8.3 Contribution

Group member	Proposal	Research	Design	Implementation	Report
Lieuwe	✓	✓	×	✓	✓
Wouter	×	✓	×	✓	✓
Tom	✓	×	✓	✓	✓
Lilya	✓	×	×	✓	✓
Wessel	✓	×	×	✓	✓

[6] More info about Playwright can be found on the Playwright site.

Chapter 9.

Conclusion

In this report, we have presented the full development process of the SonicViz app, from the initial analysis of requirements to the final implementation and testing. We began project by setting clear set of functional and non-functional requirements, which guided the design and development phases. During implementation, we faced multiple challenges, such as integrating different features and ensuring the app's performance on mobile devices. Through iterative development of testing and improving, we were able to solve these problems.

We are happy to conclude that we successfully implemented all of the must-have and should-have requirements, as well as two of the could-have requirements. This shows that we have met the core goals of the project

Throughout the development, we also emphasized the importance of user experience, resulting in an app that is not only functional but also engaging for our target audience. While we have discussed some limitations and areas for future work, the current version of SonicViz already offers a solid foundation.

Overall, we believe that SonicViz is a valuable and fun educational tool that enables high school students to explore the world of sound through visual and interactive experiences.

Bibliography

Adiletta, M. J., & Thomas, O. (2020, December 15). An Artistic Visualization of Music Modeling a Synesthetic Experience [Version Number: 1]. https://doi.org/10.48550/ARXIV.2012.08034

Android Open Source Project. (2025, April 4). Audio device type limitations. https://source.android.com/docs/core/audio/device-type-limit

Connaghan, T. (2021, August 22). What is gain & how it differs from volume. Retrieved April 15, 2025, from https://emastered.com/blog/what-is-gain

Jake. (2013). Fourier transform - PGFplots.net [Licensed under CC-BY-SA 4.0]. Retrieved March 24, 2025, from https://pgfplots.net/fourier-transform/

Ling, W.-K. (2007). 2 - reviews (W.-K. Ling, Ed.). https://doi.org/https://doi.org/10.1016/B978-012372536-3/50002-8

LLC, F. A. (2020). Signalscope advanced. Retrieved April 6, 2025, from https://www.faberacoustical.com/apps/signalscope/signalscope_adv_2020

Müller, M. (2021, April 9). Fourier Analysis of Signals. In M. Müller (Ed.), *Fundamentals of Music Processing: Using Python and Jupyter Notebooks* (pp. 39–117). Springer International Publishing. https://doi.org/10.1007/978-3-030-69808-9_2

Rijksoverheid. (n.d.). Gebruik van mobiele telefoons niet toegestaan in de klas [use of mobile phones not allowed in the classroom] [Ministerie van Onderwijs, Cultuur en Wetenschap]. Retrieved March 28, 2025, from https://www.rijksoverheid.nl/onderwerpen/voortgezet-onderwijs/mobiele-apparaten-in-de-klas

Schmitt, J. (2024, December 23). What is E2E? A guide to end-to-end testing. https://circleci.com/blog/what-is-end-to-end-testing/

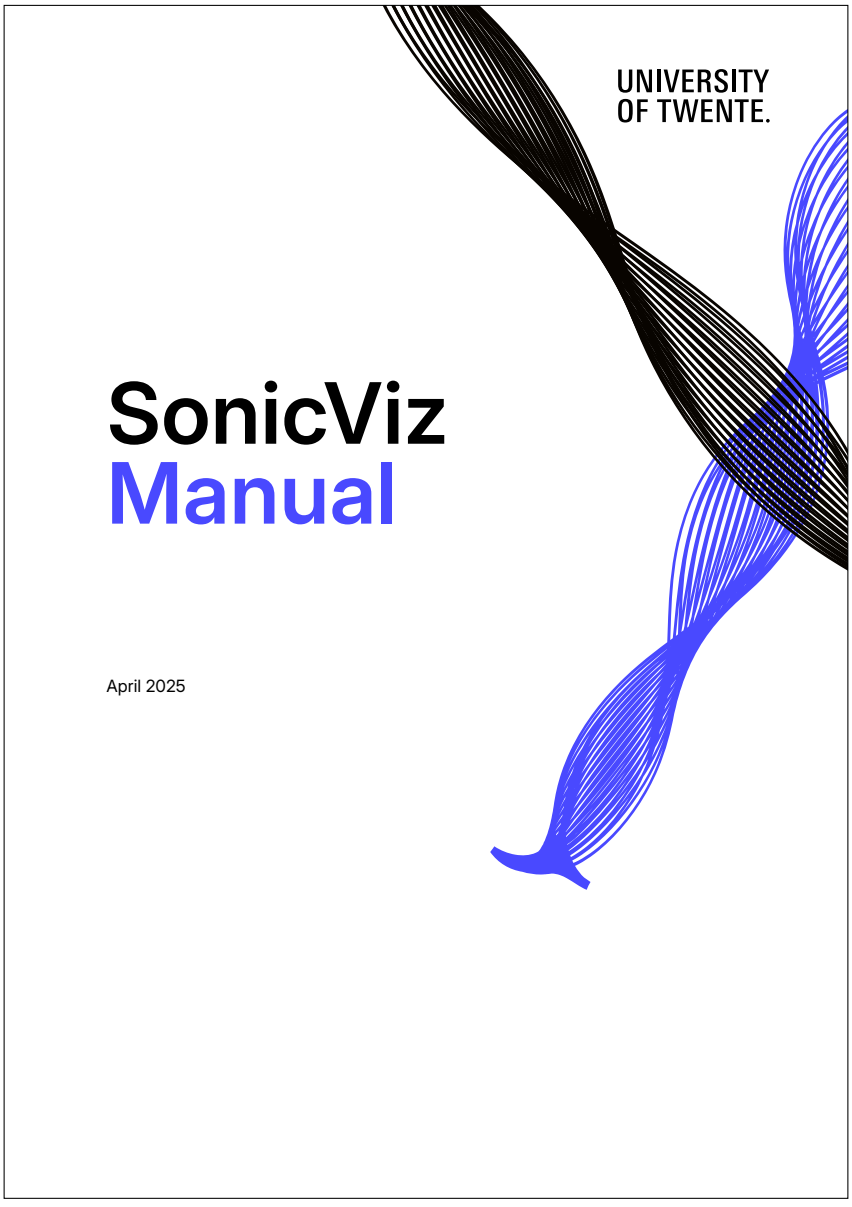
Tralie, C. (2021). Assignment 3: Vocoders and phase retrieval. Retrieved April 2, 2025, from https://ursinus-cs472a-s2021.github.io/CoursePage/Assignments/HW3_Vocoders

van den Boomgaard, R. (2022). Digital signal processing - lecture notes. Retrieved March 24, 2025, from https://staff.fnwi.uva.nl/r.vandenboomgaard/SignalProcessing/index.html

White, P. (2006, March). Choosing the right reverb. Retrieved April 8, 2025, from https://www.soundonsound.com/techniques/choosing-right-reverb

Appendix A

Manual



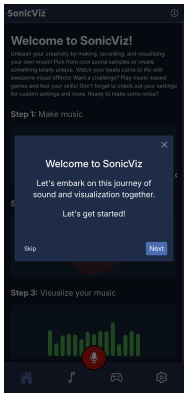
This is the manual for SonicViz. The manual details all the functionality that SonicViz offers.

To start, we will list all the tasks you can do. Every task contains a description of what it is and an instruction on how to use it. These tasks will essentially guide you through all the app has to offer. Each task is an entry in the table of contents.

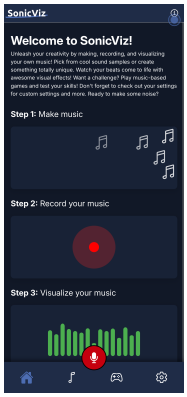
Contents

1 First launch

When you first launch the app or website, you will be presented with the home page. As this is your first time launching the app, it will start a tour. The tour is intended to show you a few of the main features of the app, as shown in Figure 1a. You can click Next on the dialog to follow the tour, click Back in order to see the previous step, if there is any, or click Skip to skip it. If you wish to temporarily pause the tour, you can press the close button (X). Then you can resume the tour by clicking the little dial, as shown in Figure 1b at the top right. At any time, you can click the information icon in the top-right corner of the page to start it again. The tour will restart if you finish or skip it unless the information icon is clicked. Other pages also contain a similar tour.



(a) Walkthrough start screen.



(b) Walkthrough end screen.

Figure 1: All the walkthrough screens.

2 Visualization

This section will explain the main functionality of the app: the visualizations.

2.1 Live visualization

The easiest way to start visualizing is to use the live visualization. From any page, you can click the red recording icon at the bottom. A modal will pop up and give you two options: Start Recording or Live Visualization. For now, we will focus on Live Visualization. Click the button "Live Visualization". Your phone or browser may ask for permission to use the microphone. Click "Always allow" or "Allow only this time", or similar, as long as permission is granted.

This will show the visualization dashboard based on the microphone input.

2.2 Visualization dashboard

The visualization dashboard contains multiple features to customize your experience and visualize the audio data.

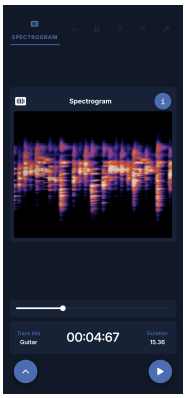


Figure 2: Visualization dashboard.

At the top of the dashboard, as you can see in Figure 2, there are various tabs containing various visualizations. Clicking through these tabs takes you to different graphs and the filters. Every item on a tab has an information icon in the top right. Clicking this will provide detailed explanation about the graph. The information icon in the Filters tab lists all filters as well with a short explanation about what they do. To close the dashboard, click the arrow button on the bottom left and then click the bottom left arrow button. Alternatively, you can also use your device's back button or gesture to close it.

Below, all six tabs, from left to right, are explained.

2.2.1 Spectrogram

The spectrogram shows three dimensions of the audio data: time on the x-axis, frequency on the y-axis, and amplitude as the color. A black color means there is little or nothing of that frequency in the audio input, and white means there is a lot. The frequencies in the spectrogram are logarithmic, as differences in low frequencies are more interesting (they relate to the human voice and general sounds, at very high frequencies audio becomes almost inaudible). The spectrogram will scroll from right to left infinitely for live visualizations and will reset when an audio clip is (re)started for recorded or sample audio clips.

2.2.2 Waveform

The waveform shows the amplitude of the audio data over time. The amplitude is the aggregated amplitude of all the frequencies. It moves right to left and is a measure of loudness of the audio. As with the spectrogram, it resets when the audio clip is started, and is infinite for live visualizations.

2.2.3 Frequency chart

The frequency chart shows the amplitude of all individual frequencies. There is no time component involved. At the top of the graph, there is vertical text with frequencies. The frequencies go from left to right, and the vertical text and the accompanying lines are markers for where that frequency is.

2.2.4 Current note

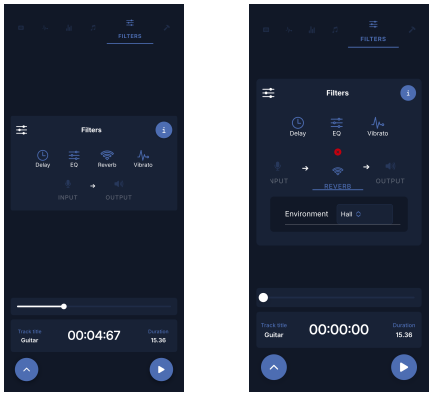
The current note tab shows the current detected note. It is based on identifying dominating frequencies in the audio data. It also shows the frequency of the detected note. For noisy audio, the note may jump around a lot.

2.2.5 Filters

The audio input stream can be modified using filters. These filters are described in the chapter Audio fundamentals. This section of the manual will only explain how to use them. The filters tab can be seen in Figure 3a. A filter can be applied by dragging it between the input and output below the list of filters. Once applied, a filter will have a red cross above it. Clicking this will remove it.

An applied filter has multiple options that are configurable. These options are shown at the bottom after clicking on the applied filter. For example, for the Reverb filter, you can choose what environment you want to use to simulate the reverb (Figure 3b).

You can apply multiple filters at once. They will be processed in order from input to output. The filters can be reordered by dragging them around.



(a) The filters tab. **(b)** The reverb filter.

Figure 3: View of filters visualization tab.

2.2.6 Custom tab

On the Custom tab, you can customize which visualizations you want to see. Initially, you will see all of them below each other. When you click the left arrow button in the bottom left, and then the top Edit button (indicated by a writing icon) also shown in Figure 4.
Clicking the Edit button will reveal additional buttons on the dashboard. Firstly, on each visualization, there are two new buttons: in the top left, a hamburger icon (3 stacked horizontal lines) and in the top right a delete button. The hamburger icon can be held to rearrange the order of the visualizations via drag and drop functionality. The delete button is used to delete visualizations from the Custom tab. Secondly, there is a green add button in the bottom right of the dashboard (indicated by a plus sign) that is used to add new visualizations to the dashboard. You are not restricted to which visualizations you add, you may have multiple of the same type, except for the filters.
Changes you make to the Custom tab are *not* currently saved when you close the dashboard.

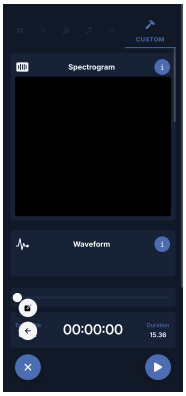


Figure 4: Customization of visuals page.

3 Recording

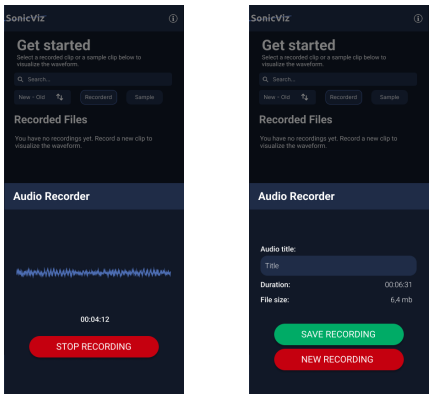
Now we will show how you can record audio and save it to visualize later. Just like live visualization you can click the red recording icon at the bottom from any page. A modal will pop up and give you two options: Start Recording or Live Visualization. This time, we will click the button "Start Recording". Your phone or browser may ask for permission to use the microphone if you haven't already done this. Click "Always allow" or "Allow only this time", or similar, as long as permission is granted.

3.1 Record

Then you will see a simple waveform that shows you the audio it detects, shown in Figure 5a. Below this waveform the current recording duration is shown. At the very bottom of the modal you can see a red button "stop recording" with which you can stop recording. If during recording you try to close the modal a notification will pop up confirming your choice. You can either click this new red "stop recording" button in order to confirm you want to discard the recording or click "cancel" to continue your recording.

3.2 Save recording

When you finish recording you will see the view of Figure 5b and must give a title to this file. You can also see the duration and size of the recording. After filling in a title, you can save the recording or with the green "save recording" button. If you do not want to save the recording you can select the red "new recording" button and record something else if you want, or close the modal which will then again confirm your choice with a notification.



(a) Modal during recording.

(b) Modal after finishing recording.

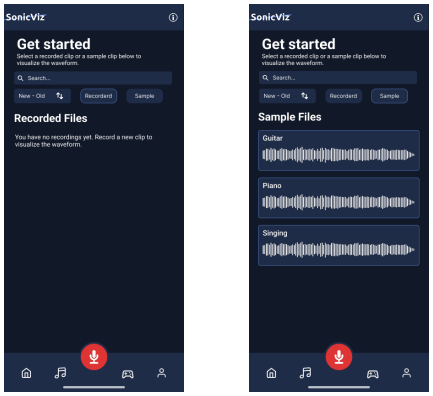
Figure 5: The recording process.

3.3 Audio page

On the audio page you can find your saved recordings as well as the sample audio files on the app. Below the search bar on the left you can switch between your recordings and the sample files, as can be found in Figure 6a and Figure 6b. To the right of the search bar you can sort the audio files by date or duration in both ascending and descending order. You can also search for a specific audio file with the search bar. If you don't have any recordings made yet, the "Recorded" section of the audio page will be empty.

Clicking a recording will reveal the visualization dashboard which is the same for live visualization, except there will now be playback controls. At the bottom of the page, there is a bar which shows the title of the audio being played on the left, the current elapsed time in the middle, and the total audio duration on the right. Above these, there is a track which the progress of the audio. You can drag the circular knob of this progress bar to scrub through the audio. There is also a play/pause button in the

bottom right of the dashboard. When an audio reaches the end, playback will stop. Pressing play will then restart the audio. Visualizations will reset when an audio is restarted in this way.



(a) Recorded audios selected.

(b) Sample audios selected.

Figure 6: Audio page with with recordings and samples.

4 Games

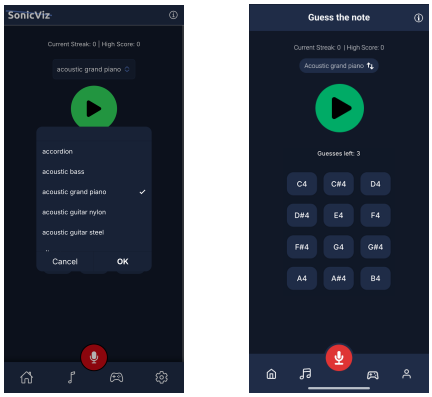
The games page holds some mini-games which can be played. A small description can be found about each game under the game's title. The high-score can be seen above the game's title which holds the highest score that was ever achieved. To play a game, its corresponding card must be clicked which will then open the start screen. The start screen will entail a short description of the game, which can be played after pressing the "Start Game" button.

4.1 Guess the note

The big round green play button can be pressed to generate a note. The generated note is then played by the selected instrument, which can be seen and changed above the play button. When selecting a new instrument, the choice must be confirmed by pressing "OK" to save the selection. You can see the instrument selection in Figure 7a.

In "Guess the note" the user must select which note is played by using the provided button panel. If the guess is wrong, the button will turn red and the number of remaining available guesses will decrease by one. Once all guesses have been used up, a screen will pop up showing the correct answer. Then a new note will be generated and the guesses reset.

If the guess is right, a screen will pop up, once again showing the correct answer. Then again new note will be generated and guesses reset, but the current streak is increased. The current streak will increase with consecutive right answers and reset with a wrong answer. See Figure 7b for the Guess the note game screen.



(a) Guess the note explanation page.

(b) Guess the note game page.

Figure 7: Game page and game screens.

4.2 Pitch perfect

In pitch perfect, the generated note will be shown which must then be played or sung back for one second. The progress bar will fill up in green while the correct note is played or turn red when the wrong note is played. It will also show in text at the right, whether the note is matching. The meter will move according to which notes it detects. Below the meter, the detected note as well as the frequency is shown. When the note was matched successfully, a screen will pop up. Then a new note is generated and the score will be increased.

5 Changing themes

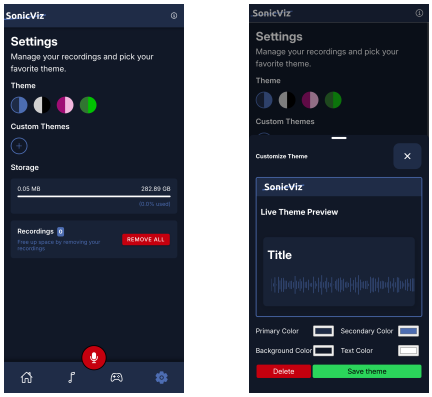
On the settings page, see Figure 8a, you can find a theme section. Here you can switch between the four default themes of the app.

5.1 Custom themes

It is also possible to add a custom theme by clicking the round add button in the custom theme section. This will reveal a new popup modal with the custom theme editor in the currently selected theme. This theme editor modal is shown in Figure 8b. The outlined box holds a preview of several components of the app in the currently selected theme. The colors can be changed below the preview, by clicking the smaller boxes next to each element. After selecting a color this will be updated in the preview, so you can see how your change affects the theme.

The theme can be discarded or deleted with the red "Delete" button or saved with the green "Save Theme" button. In case you try to close the editing modal, a notification will appear to confirm that you want to discard your changes.

When saving the theme, it will close the modal and automatically apply this new theme. Then you can find your new custom theme in the custom themes section. You can add another custom theme with the same add button or edit the currently selected custom theme with the brush button. A maximum of four custom themes can exist.



(a) The settings page.

(b) The theme editor modal.

Figure 8: Changing themes in the settings page.

6 Deleting recordings

Lastly, we will show you how to delete your recordings. The sample files cannot be deleted, only your own recordings.

6.1 Delete single recording

First, to delete a single recording, you must click on your recording on the audio page, so the visualization dashboard pops up. Then you click the round chevron button in the bottom left. This will expand a small menu. Now you can click the red button with a trash icon and a small notification will appear to confirm that you want to delete the recording. Clicking the red "delete" button will permanently delete the recording. If you don't wish to delete it you can click "cancel".

6.2 Delete all recordings

On the settings page you can delete all of your recordings at once. In the storage section the first bar shows you how much storage is currently being used by your recordings. Below this bar you can see how many recordings you have at the left and you can delete them with the red "remove all" button at the right. Once again a notification will pop up to confirm your choices where pressing "delete" will permanently remove all of your recordings.

Appendix B

Code coverage

All files

78.38% Statements `1615/2073` **70.93%** Branches `576/812` **82.61%** Functions `406/489` **78.4%** Lines `1449/1848`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	100%	9/9	100%	3/3	100%	2/2	100%	9/9
src/components/filters	92.85%	13/14	100%	4/4	100%	2/2	92.3%	12/13
src/components/filters/DelayFilter	71.05%	27/38	57.14%	4/7	61.53%	8/13	68.57%	24/35
src/components/filters/EQFilter	59.09%	26/44	60%	9/15	50%	8/16	62.16%	23/37
src/components/filters/FilterBar	61.29%	38/62	60.6%	20/33	70.58%	12/17	64.15%	34/53
src/components/filters/FilterDropzone	55.55%	5/9	57.14%	4/7	33.33%	1/3	80%	4/5
src/components/filters/FilterOption	100%	10/10	89.47%	17/19	100%	4/4	100%	6/6
src/components/filters/FilterSource	100%	9/9	80%	4/5	100%	4/4	100%	6/6
src/components/filters/ReverbFilter	72.41%	21/29	66.66%	4/6	70%	7/10	71.42%	20/28
src/components/filters/VibratoFilter	71.73%	33/46	57.14%	4/7	61.53%	8/13	70.45%	31/44
src/components/games/GameInstruction	100%	4/4	100%	3/3	100%	2/2	100%	3/3
src/components/games/Meter	100%	14/14	80%	4/5	100%	5/5	100%	11/11
src/components/games/ToneGenerator	81.81%	27/33	85.71%	6/7	80%	8/10	79.31%	23/29
src/components/general	100%	18/18	80%	4/5	100%	6/6	100%	17/17
src/components/general/InfoPopover	100%	5/5	100%	3/3	100%	2/2	100%	5/5
src/components/general/Logo	100%	26/26	100%	8/8	100%	5/5	100%	21/21
src/components/general/NavItem	93.33%	14/15	80%	8/10	100%	5/5	90.9%	10/11
src/components/general/Navbar	100%	4/4	100%	3/3	100%	2/2	100%	4/4
src/components/general/PageHeader	100%	5/5	100%	3/3	100%	2/2	100%	5/5
src/components/general/RecordingCard	97.82%	45/46	92.85%	26/28	100%	12/12	100%	42/42
src/components/general/ThemeEditorModal	77.27%	51/66	57.14%	12/21	68.42%	13/19	77.04%	47/61
src/components/general/ThemeSection	100%	37/37	91.3%	21/23	100%	14/14	100%	32/32
src/components/visualization/BinnedWaveform	90%	72/80	54.54%	12/22	87.5%	7/8	95.52%	64/67
src/components/visualization/CurrentNote	43.24%	16/37	38.46%	5/13	66.66%	4/6	41.93%	13/31
src/components/visualization/FrequencyChart	46%	23/50	35.71%	5/14	85.71%	6/7	43.47%	20/46
src/components/visualization/Note	100%	8/8	100%	5/5	100%	2/2	100%	6/6
src/components/visualization/ScrollingWaveform	47.61%	20/42	36.36%	4/11	75%	6/8	44.73%	17/38
src/components/visualization/Spectogram	48.57%	34/70	37.5%	9/24	87.5%	7/8	49.23%	32/65
src/components/visualization/WaveformVisualizer	96.22%	51/53	81.25%	13/16	100%	5/5	100%	47/47
src/components/visualizationModal/AddVisualizationSegmentSheet	100%	11/11	100%	6/6	100%	5/5	100%	10/10
src/components/visualizationModal/DeleteRecordingAlert	100%	4/4	100%	3/3	100%	2/2	100%	4/4
src/components/visualizationModal/InfoModal	87.5%	7/8	71.42%	5/7	66.66%	2/3	83.33%	5/6
src/components/visualizationModal/VisualizationCard	100%	9/9	78.57%	11/14	100%	3/3	100%	5/5
src/components/visualizationModal/VisualizationModalContent	77.45%	79/102	64.15%	34/53	82.14%	23/28	79.31%	69/87
src/components/visualizationModal/VisualizationModalFooter	37.77%	34/90	52.17%	24/46	37.5%	6/16	36.7%	29/79
src/components/visualizationModal/VisualizationSegmentContent	95.52%	64/67	82.92%	34/41	92%	23/25	95.65%	44/46
src/components/walkthrough/AudioWalkthrough	72.16%	70/97	59.45%	22/37	100%	16/16	69.31%	61/88
src/components/walkthrough/GamesWalkthrough	87.01%	67/77	87.5%	28/32	66.66%	6/9	86.3%	63/73
src/components/walkthrough/HomeWalkthrough	100%	35/35	95%	19/20	100%	5/5	100%	32/32
src/components/walkthrough/SettingsWalkthrough	86.3%	63/73	68.57%	24/35	92.3%	12/13	85.29%	58/68
src/hooks	62.5%	15/24	16.66%	1/6	100%	4/4	60.86%	14/23
src/pages	95.11%	214/225	82.71%	67/81	95.38%	62/65	95.6%	196/205
src/pages/games	92.85%	169/182	88.33%	53/60	91.66%	33/36	93.49%	158/169
src/providers/AudioContextProvider	36.66%	22/60	0%	0/14	44.44%	8/18	35.18%	19/54
src/providers/ThemeProvider	69.69%	46/66	66.66%	12/18	83.33%	10/12	69.84%	44/63
src/services	85%	51/60	66.66%	6/9	88.23%	15/17	84.74%	50/59

Code coverage generated by istanbul at 2025-04-15T22:47:57.363Z

Appendix C

Github workflow

```yaml
name: Build and Cypress Tests
on:
  workflow_dispatch:
  pull_request:
    branches:
      - master
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22.14.0'
      - name: Install dependencies
        run: npm ci
      - name: Build the project
        run: npm run build
      - name: Cache build artifacts
        uses: actions/cache@v3
        with:
          path: |
            node_modules
            dist
            ~/.cache/Cypress
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
    outputs:
      build_success: ${{ job.status }}

    desktop-tests:
    needs: build
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        browser: [chrome, firefox]
```

```yaml
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22.14.0'
      - name: Restore cached dependencies
        uses: actions/cache@v3
        with:
          path: |
            node_modules
            dist
            ~/.cache/Cypress
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}

      - name: Run desktop E2E tests on ${{ matrix.browser }}
        uses: cypress-io/github-action@v6
        with:
          install: false
          start: npm run preview
          config: '{"e2e":{"baseUrl":"http://localhost:4173"}}'
          wait-on: "http://localhost:4173"
          wait-on-timeout: 120
          browser: ${{ matrix.browser }}

      - name: Run desktop component tests on ${{ matrix.browser }}
        uses: cypress-io/github-action@v6
        with:
          start: npm run preview
          config: '{"e2e":{"baseUrl":"http://localhost:4173"}}'
          wait-on: "http://localhost:4173"
          wait-on-timeout: 120
          browser: ${{ matrix.browser }}
          component: true

      - name: Generate code coverage report
        run: npx nyc report --reporter=lcov

      - name: Upload E2E test results on failure
        if: failure()
        uses: actions/upload-artifact@v4
        with:
          name: cypress-desktop-results-${{ matrix.browser }}
          path: |
            cypress/videos
            cypress/screenshots
            coverage
    timeout-minutes: 30

  mobile-tests:
    needs: build
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        browser: [chrome, firefox]
    steps:
```

```yaml
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22.14.0'
      - name: Restore cached dependencies
        uses: actions/cache@v3
        with:
          path: |
            node_modules
            dist
            ~/.cache/Cypress
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}

      - name: Run mobile E2E tests on ${{ matrix.browser }}
        uses: cypress-io/github-action@v6
        with:
          config: '{"e2e":{"viewportWidth":375,"viewportHeight":667,
                           "baseUrl":"http://localhost:4173"}}'
          start: npm run preview
          wait-on: "http://localhost:4173"
          wait-on-timeout: 120
          browser: ${{ matrix.browser }}

      - name: Run mobile component tests on ${{ matrix.browser }}
        uses: cypress-io/github-action@v6
        with:
          config: '{"e2e":{"viewportWidth":375,"viewportHeight":667,
                           "baseUrl":"http://localhost:4173"}}'
          start: npm run preview
          wait-on: "http://localhost:4173"
          wait-on-timeout: 120
          browser: ${{ matrix.browser }}
          component: true

      - name: Generate code coverage report
        run: npx nyc report --reporter=lcov

      - name: Upload E2E test results on failure
        if: failure()
        uses: actions/upload-artifact@v4
        with:
          name: cypress-mobile-results-${{ matrix.browser }}
          path: |
            cypress/videos
            cypress/screenshots
            coverage
    timeout-minutes: 30
```