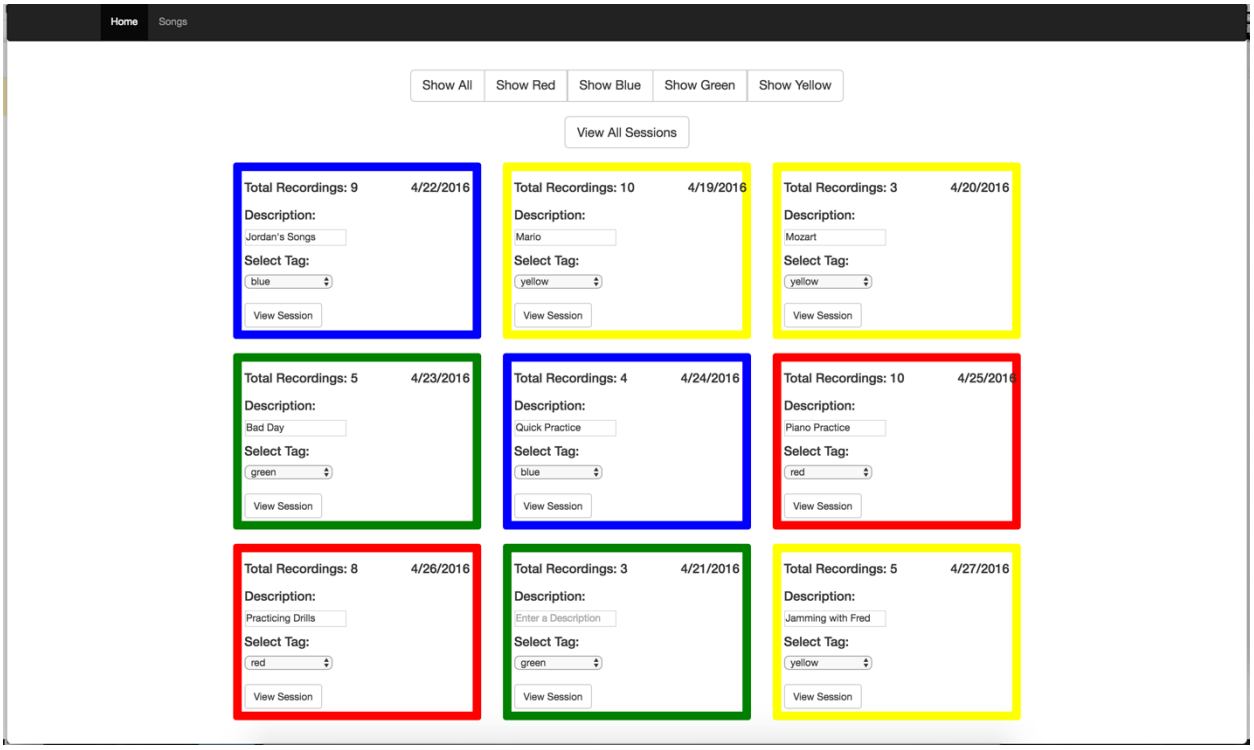
Senior Project Report

MusicTrakr



By Benjamin Lin

Advised by Andrew Danowitz

CPE Department

California Polytechnic State University, San Luis Obispo

June 2016

Purpose

The purpose of MusicTrakr is to provide musicians with a better way to track their practice time and record their playing sessions. With the growth of cloud storage and wireless technology, it is now possible to wirelessly record everything a musician plays. With MusicTrakr, a musician will be able to see how their total practice time has grown over time instead of guessing how many hours they practiced. In addition, a musician will be able to play back recordings from the past to see how much they have improved since then.

Goals

The goals of the MusicTrakr project were to create a wireless hardware device that records sound files to a cloud database and to create a website that allows the user to play back previous recordings and view graphs logging their practice times.

Overview

MusicTrackr is a recording device and accompanying website that allows musicians to record, keep track, and analyze everything that they play. The recording device, referred to as "The Recorder", is a wireless device that records raw sound and stores it in a cloud database. The Recorder accomplishes this with a start and stop button that the user may press. The accompanying website, referred to as "The MusicTrakr Website", displays all of the recordings stored in the database and organizes them by date. These groups of recordings by date are referred to as sessions. The user may choose a specific session to view in detail. After picking a specific session, the user can view graphs analyzing the practice time as well play back any of the recordings during the session. The user may also add comments to specific recordings and mark them as favorites.

System Architecture

The system architecture consists of The Recorder, a recording server, a cloud database, a web server, and The MusikTrakr Website. A system architecture diagram is shown below in **Figure 1**. The system architecture begins with The Recorder. When the user presses the start button, The Recorder starts to record raw sound and streams it wirelessly to the recording server. The Recorder stops recording once the stop button is pressed. The recording server then compiles the sound data into a .wav file and then saves the .wav file onto the cloud database along with metadata relating to the recording. The cloud database is idle until the user interacts with The MusikTrakr Website. When a user opens up a specific recording session, The MusikTrakr

Website requests the recording metadata relating to that recording session from the web server. The web server then queries the cloud database to get the meta data and then sends that information to the requesting web client. This results in the the correct recordings being displayed on the MusikTrakr Website.

Two servers were used because the web server had to be hosted remotely while for this prototype the recording server had to be on the same local Wi-Fi network as the Particle Microcontroller. The recording server had to be on the same local Wi-Fi because the recording server specified the local IP address of the Particle when connecting and a remote server would end up connecting to a random IP on its remote network instead of the Particle. The system worked with a combined recording and web server when hosted entirely locally but real websites are usually accessible from other machines other than the hosting machine. Thus, the web server portion was hosted remotely through a service called Heroku while the recording server remained local on a MacBook Pro.

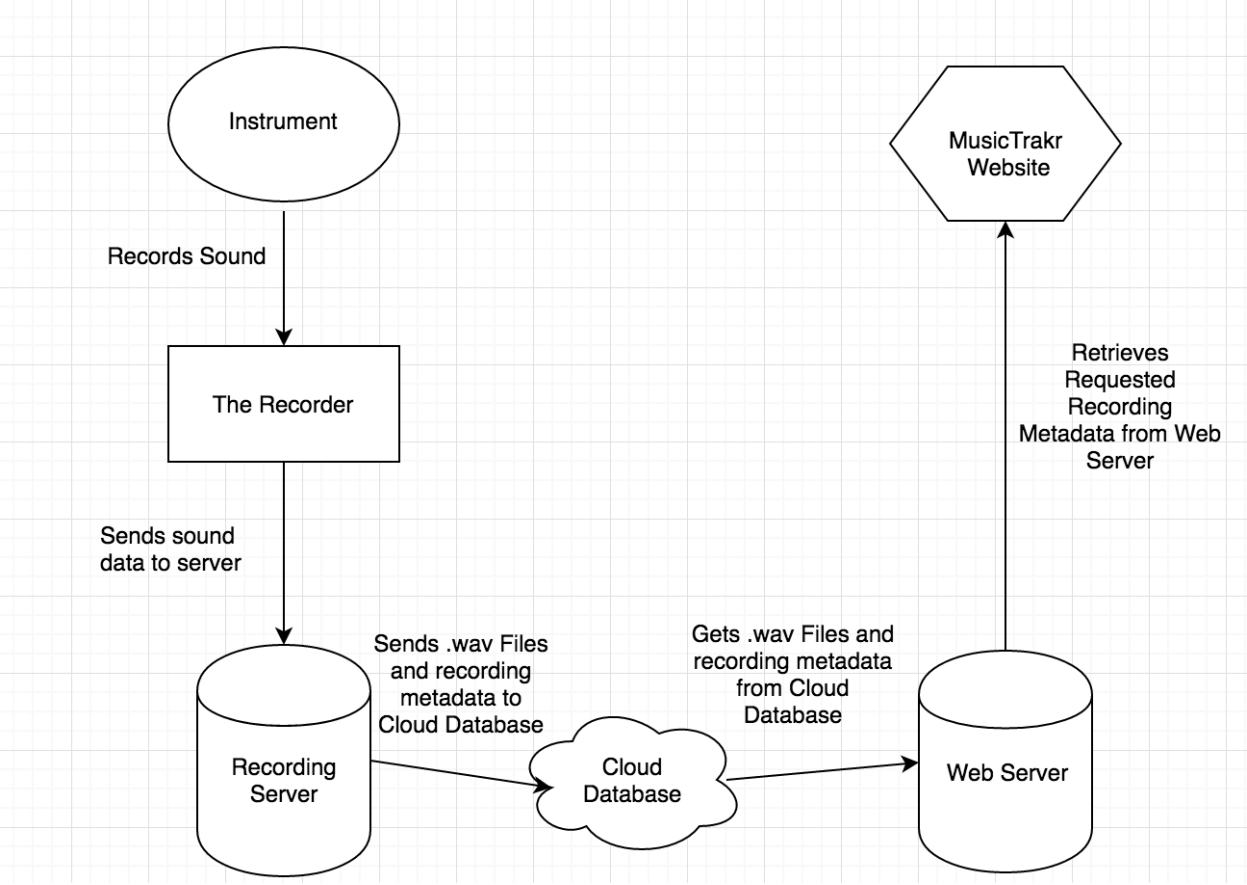


Figure 1: System Architecture Diagram

The Recorder

The recorder is a hardware device that consists of a Particle microcontroller, two push-button-switches and a microphone. A diagram of The Recorder is shown below in **Figure 2**. The two buttons are hooked up to digital IO pins on the Particle microcontroller and start and stop the recording functionality. The microphone outputs voltages related to the incoming sound intensity and is hooked up to an ADC on the Particle microcontroller. When the start button is pressed, the Particle microcontroller reads the value on the ADC, scales it, and then sends it wirelessly to the recording server using its built in wireless communication functions. When the stop button is pressed, the Particle microcontroller stops sending data. The Recorder is able to reach a sample rate of 11kHz.

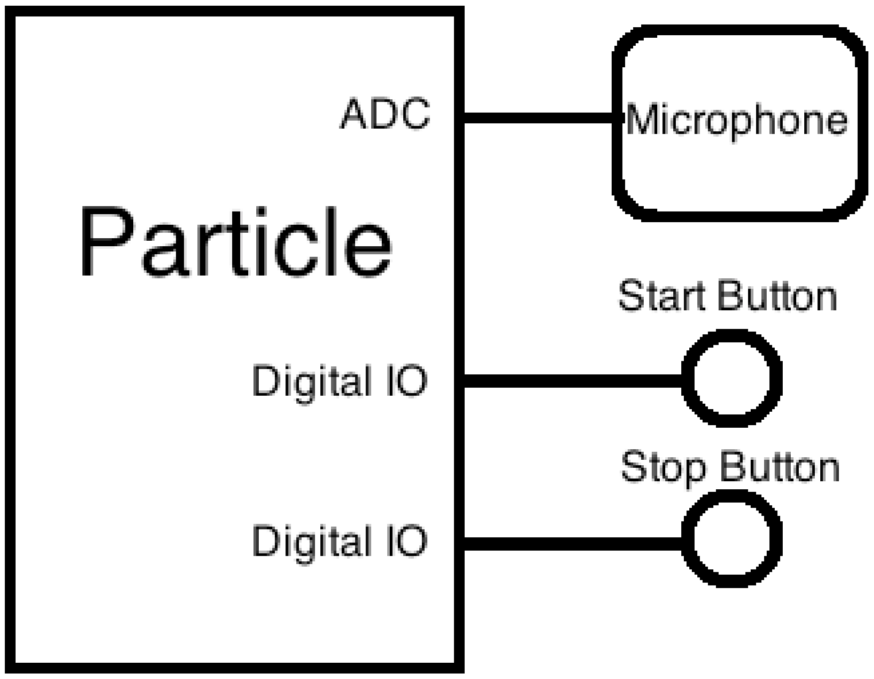


Figure 2: Diagram of The Recorder

Recording Server

The recording server was built using Node.JS and is responsible for receiving the incoming sound data from The Recorder and packaging it into a .wav file. The recording server must be on the same local Wi-Fi network as The Recorder in order for it to receive the sound data. The .wav format was chosen because it is a raw and uncompressed format that is simply a 44 byte header file on top of the raw sound data and thus easy to implement. After creating the .wav file, the recording server saves the .wav file to the cloud database along with metadata relating to the sound recording. The metadata includes the date at which the recording was recorded, the duration of the recording, and the name of the .wav file.

Cloud Database

The cloud database is a MongoDB database that is hosted on a Heroku server.

Web Server

The web server was built using Node.JS and is responsible for delivering HTML views to the MusicTrakr Website client based on the URLs requested as well as handling requests for the recording metadata. The web server is hosted remotely on a Heroku server. Whenever a new recording is added to the cloud database, the web server downloads the .wav file off the cloud database and stores it in its memory. This is done to allow the MusicTrakr Website to reference the .wav file from memory so it can be played by the user. When the MusicTrakr website requests recording metadata, the web server queries the cloud database and returns a JSON object containing the metadata to the client.

MusicTrakr Website

The MusicTrakr website is what the user interacts with when they want to playback their past recordings and view graphs logging their practice time. When the user clicks on a recording session, the MusicTrakr web client requests recording metadata from the web server. Then, using AngularJS, the web client binds the recording metadata to media elements and graphs on the UI so the user can begin to play back recording clips and view their practice graphs. When a user changes views, the web server sends a different view for the web client to display.

System Design

The system leveraged the Particle Microcontroller, the MEAN stack, Bootstrap, Google Charts, and Heroku in its design.

Particle Microcontroller

The Particle Microcontroller was used as the microcontroller in the hardware design because it only cost $20, had an Arduino-like coding language, and built in functions to send data wirelessly. The alternatives to the Particle Microcontroller were various Wi-Fi modules and other Wi-Fi integrated microcontrollers. In the end, Particle was chosen because it was the cheapest and most well-documented option. Particle can not connect to campus Wi-Fi because it requires a network name, log-in name, and password, but this should not be a problem for people who use home Wi-Fi that doesn't have a log-in name.

MEAN Stack

The MEAN stack consists of MongoDB, Express, AngularJS, and NodeJS. The MEAN Stack was mainly chosen because all of its components used JavaScript which made development easier since only one language needed to be learned. The MEAN Stack was chosen over the LAMP stack because the developer of this project already knew JavaScript and did not want to slow down development by learning PHP, Linux and Apache. In retrospect, MySQL might have been a better choice for the database portion of the stack. MongoDB was simple to use when querying all rows in a table with simple filters (e.g., find all recordings on a specific date) but aggregated results (e.g., return a table showing the distinct dates that had recordings and display the number of recordings in each of those) were very verbose with MongoDB. MySQL would have resulted in more readable query code.

Bootstrap

Bootstrap was chosen because the developer of the project had a limited knowledge of CSS and needed an external library to improve the UI of the website. Bootstrap is a styling library that makes HTML components look better by adding Bootstrap styling to them. In addition to that, Bootstrap makes the HTML components resize based on the screen size so the website would still look well-formatted on mobile, tablet and desktop. The alternative to Bootstrap was to learn CSS and code in all of the styling or to use a different styling library. Since the developer had previous experience in Bootstrap, Bootstrap was chosen.

Google Charts

Google Charts was chosen to be the charting API because it had a lot of documentation and examples. An alternative that was looked into was the D3.js library which can be used to build graphs in any shape or form. The problem was D3.js was that it was more difficult to learn and did not have any pre-built charts out of the box. Since MusikTrakr only needed simple bar graphs and line graphs, Google Charts was chosen.

Heroku

Heroku was chosen to be the hosting site because it was free and had support for hosting both a MongoDB cloud database and a Node.js web server. An alternative that was first looked into for hosting was OpenShift, but hours spent trying to figure out OpenShift resulted in little progress so Heroku was tried instead. Heroku had a quick example on how to host a Node.js web server that made more sense, so in the end Heroku was chosen.

Website Design

The website was designed to two main views. The first view is the home page, which displays all of the available recording sessions. The second view displays the graphs and recordings present in an individual recording session.

Home Page

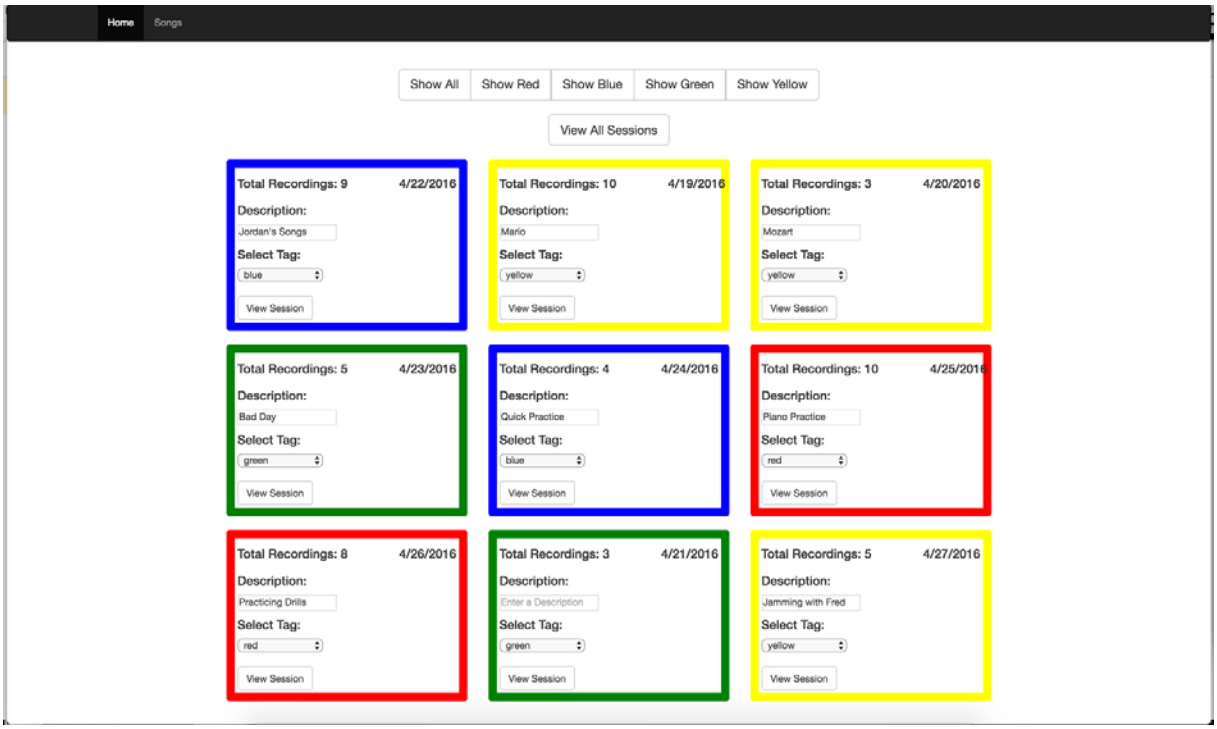


Figure 3: Home Page of MusicTrakr Website

The home page displays a thumbnail for each specific date in the database that had at least a single recording. An example of the Home page is shown above in **Figure 3**. The user can add a description to the date and also tag it with a specific color. Near the top of the screen, the user can filter the dates by the tag color. The user may also view all the recordings ever made by clicking the “View All Sessions” button. Lastly, at the top of the screen there is a navigation bar. The “Songs” tab has the same result as the “View All Sessions” button.

Songs Page

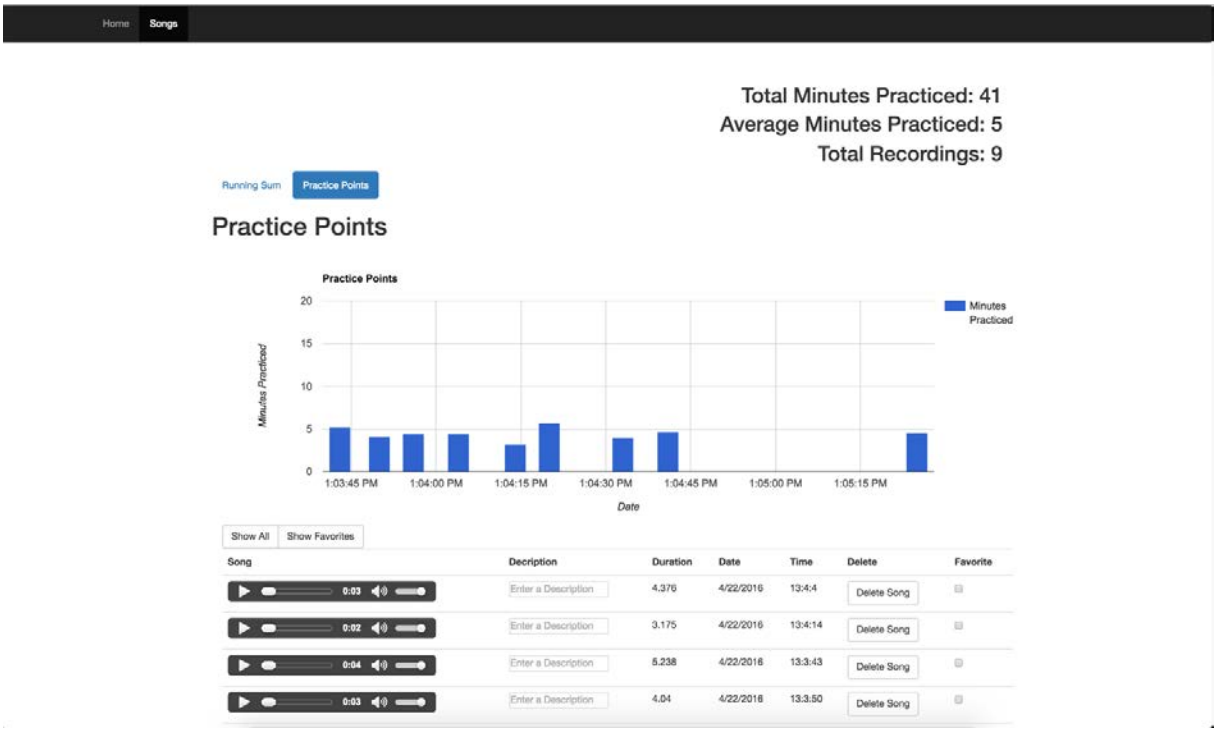


Figure 4: Song Page of MusicTrakr Website

The songs page can be accessed by clicking on a specific date, the “View All Sessions” page, or the “Songs” tab on the navigation bar. An example of the Song page is shown above in **Figure 4**. Clicking on a specific date will bring up sessions that were recorded on that date and charts that reflect that date. Clicking on the “View All Sessions” or “Songs” tab will bring up all recorded sessions ever made and charts that reflect every song recorded.

The Songs page displays the total amount of minutes practiced, average minutes practiced, and total songs recorded at the top of the screen. Below that, there is a chart that will either show a running sum of the total amount of minutes practiced over time or the lengths of individual recordings over time depending on whether the “Running Sum” or “Practice Points” option is selected.

At the bottom of the page there is a table that contains the song media, description, duration, date, and time as well as an option to delete the song from the database or favorite it. The description of the song can be edited by editing the text field. The user has the option of only displaying favorite songs by hitting the “Show Favorites” button. The “Show All” button will allow all of the songs to be shown.

The user may return to the home page by hitting the “Home” tab on the navigation bar.

Main Issues

Low Sampling Rate

Due to limitations in the microcontroller speed, a sampling rate of only 11kHz was received. Since humans can hear up to 20kHz, a sampling rate of 40kHz is needed to prevent any aliasing of the sound recordings. The limitations were due to the fact that the microcontroller is always connected to the internet. To stay connected to the internet, an interrupt is triggered every loop in the main loop function, which slows down the sampling rate of the ADC. This could be fixed by temporarily disabling the internet interrupt, recording the values, and then turning the internet back on. However, the microcontroller only has 128 KB of flash ram which can be filled up after recording about 64 seconds worth of data so the buffer would have to be flushed once every minute.

Hosting Remote Server and Connecting to Particle

As mentioned before, the recording server and web server were split because remote servers were unable to connect to the Particle's local IP address. This could be solved by having the Particle connect to the remote server instead. This was attempted during the project but the two-server method was chosen to keep the project moving forward to meet deadlines. If the two servers were merged and hosted remotely, the cloud database would only need to store the metadata and would no longer need to store the .wav files since the server would save the .wav file to memory directly. This would help keep the storage size down in the cloud database.

Conclusion

Overall, the project accomplished its initial goals of creating a wireless hardware device that records sound files to a cloud database and creating a website that allows the user to play back previous recordings and view graphs logging their practice times. However, in order to reach these goals, some compromises had to be made in the project quality and implementation. First, the project fell short in the quality of the sound play back due to The Recorder's sampling rate of only 11kHz. Second, the recording web server was unable to be hosted remotely which makes it impossible for users to record anything without a computer running the recording server somewhere nearby on the local network. If this project were to be continued, the first goal would be to change the design so the Particle can connect and send data to a remote server. The second goal after that would would be to improve the sampling rate to at least 40kHz. Some additional goals would be to create packaging for The Recorder and to conduct user studies to get feedback on the website UI.