

NORTHWEST NAZARENE UNIVERSITY

Creation of a Web-Based Audio Recorder Utilizing a
USB Foot Pedal and External Microphone.

THESIS

Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements for the degree of
BACHELOR OF ARTS

Isaac L. Kronz

2019

THESIS

Submitted to the Department of Mathematics and Computer Science

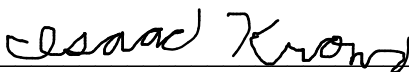
in partial fulfillment of the requirements for the degree of


BACHELOR OF ARTS


Isaac L. Kronz

2019

Creation of a Web-Based Audio Recorder Utilizing a USB Foot Pedal and External Microphone.

Author: 
Isaac Kronz

Approved: 
Barry L. Myers, Ph.D., Chair, Department of Mathematics & Computer Science
Faculty Advisor

Approved: 
Stephen P. Riley, Ph.D., Department of Theology and Christian Ministry, Second
Reader

Approved: 
Barry L. Myers, Ph.D., Chair, Department of Mathematics & Computer Science

Abstract

Creation of a Web-Based Audio Recorder Utilizing a USB Foot Pedal and External Microphone.

KRONZ, ISAAC (Department of Mathematics and Computer Science),
MYERS, DR. BARRY (Department of Mathematics and Computer Science).

Recording via the browser is not a new technology by any means, but this project aims to do much more. The goal of this project is to create a web-based audio recorder that takes input from a foot pedal, gives an option as to what audio input to choose, and uploads the encoded audio to a server for playing, renaming, or deleting. A key objective of this project is also to minimize the amount of browser delay there is between recordings. This delay could be potentially halved by splitting up the audio recordings by increments of time, encoding them in the background, and then sending those chunks to the server. Another potential solution to the delay is to send a binary stream of the audio directly to the server and have it encoded on that end. The result of this project should be a baseline product that can be improved for medical dictation in pathology, but can be extended to any similar purpose.

Acknowledgements

I would like to thank my parents, who supported me monetarily, mentally, and emotionally through the wild ride that college has been. I would also like to thank all of the professors here at NNU that have not taught me so well, especially Dr. Myers and Dr. Hamilton. Also, thanks to my uncle, Jason Kronz, for giving me the idea and push to do this project. Finally, a special thanks goes to my classmates Ryan, Zach, Nick, and Hanna for being there for me since day one for questions, laughs, and being willing to be a second eye for my projects.

Table of Contents

Title Page	i
Signature Page	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
Table of Figures	vii
Overview	1
Background	1
Terms	2
Implementation	3
Results	13
Conclusion	18
References	20
Appendix A	21
back.php	21
backend.php	21
index.html	23
launch.sh	24
php.ini	24
README.md	25
style.css	25
js	28

app.js	28
Appendix B	36
app.js	36
package.json	37
README.md	38
<i>public</i>	38
recorder.js	38
<i>tpl</i>	40
index.jade	40
style.css	41

Table of Figures

Figure 1	3
Figure 2	4
Figure 3	5
Figure 4	6
Figure 5	6
Figure 6	7
Figure 7	7
Figure 8	8
Figure 9	9
Figure 10	9
Figure 11	10
Figure 12	11
Figure 13	12
Figure 14	13
Figure 15	14
Figure 16	14
Figure 17	15
Figure 18	15
Figure 19	16
Figure 20	16
Figure 21	17

Overview

The goal of this project was to create a proof-of-concept of a web-based dictation software. Not only would it need to record audio and upload it to a server, it has to be able to take input from any audio source, record at the push of a foot pedal, and have minimum to no delay between recordings. There were a few challenges that had to be overcome when developing this software. First, most foot pedals are proprietary, meaning they only work with specific native software solutions, and do not give normal keyboard-like inputs to the PC like most similar devices do. Another challenge was dealing with the delays caused by encoding on the client-side, and deciding how to remedy that problem. Finally, an overarching problem was learning to interact with audio inputs and managing that data in the browser, which is still pretty recent technology.

Background

This project began as a solution to a problem of Joseph Kronz M.D., who is a local pathologist. Pathologists look at slides of tissue under a microscope in order to determine if the tissue sample has evidence of being cancerous or not. In order for this to be done correctly and efficiently, pathologists dictate their findings on a microphone, which is then played back by an assistant who transcribes the dictation. While many may use proprietary native software for this dictation, Dr. Kronz and his brother, Jason Kronz, want to incorporate dictation into a web-based software they are already developing for pathology. This project would hopefully be something they could drop into their existing product, and easily add features to the existing barebones set.

Terms

This project required learning and using many languages, technologies, and frameworks that the reader may not have come in contact with before. These need to be defined in order for any reader to have a full understanding of the project.

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the basic building blocks of websites, creating the text, form, color, and styling of the site. Javascript is a web programming language that allows a website to become interactive. Whether sending information to a server or accessing the microphone, it is an extremely versatile, one of the most used programming languages in the world, and the cornerstone of this project. PHP (Personal Home Page) is an older programming language built primarily for web servers, and is used as such in this project.

Node.js is one of the newer and faster server environments on the scene, written in Javascript. It allows for web hosting and expansion with many Javascript frameworks. The second implementation of my project utilizes Node.js with a few other frameworks in order to allow for binary streaming. These include the Express.js library, which is one of the most popular web frameworks for Node. Next is Binary.js, which is a binary streaming library for Node/Javascript. Binary.js allows for direct streams of information from one endpoint to another. Jade is what is referred to as a template engine, which basically allows for the writing of more efficient HTML in Javascript.

Outside of the programming languages and frameworks, there are a few other various definitions that will aid in understanding of the project. WAV (Waveform Audio File Format) is an audio format for storing an audio bitstream. MP3 (Moving Picture Experts Group-1 Audio Layer-3) is an encoded and compressed audio format that retains the audio of the original source

but dramatically reduces the file size. OGG is a fairly common audio format made by Xiph.Org Foundation, and unlike its peers in this terms section, actually is not an acronym. VIM (Visual Instrument iMproved) is an old command-line based text editor. HID (Human Interface Device) refers to peripheral devices that give the computer inputs from humans, like keyboards and mice.

Implementation

This project began with a large amount of research being done on Google for Javascript-based encoding software. This resulted in coming across WebAudioRecorder.js (Higuma, 2016), a Javascript library for encoding. However, no successful implementations were immediately shown, until the following demo was found (“Addpipe”, 2018). This is what provided the foundation for this project.

To begin, VIM was decided upon as the text editor, both to familiarize the author with it more, and for ease-of-use since command prompts were used for launching and keeping track of multiple servers already.

```
!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Isaac Audio Recorder</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <h1>Isaac's Audio Recorder</h1>
    <p>Made as my senior project for NNU<br><a href="https://math.nnu.edu" target="_blank">NNU CS</a></p>
    <!-- Selection of audio encoding type -->
    <p>Conversion encoding:<br>
    <select id="encodingTypeSelect">
      <option value="wav">Waveform Audio (.wav)</option>
      <option value="mp3">MP3 (MPEG-1 Audio Layer III) (.mp3)</option>
      <option value="ogg">Ogg Vorbis (.ogg)</option>
    </select>
    </p>
    <!-- Input section -->
    <div id="controls">
      <button id="recordButton">Record</button>
    </div>
  </body>
</html>
-- INSERT --
```

Figure 1 - HTML for the first project

In order to set up a quick server on localhost to test everything to see if it was functional, the following script was run.

```
^CMacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ php -S localhost:8000
PHP 5.6.30 Development Server started at Tue Jan 29 14:58:30 2019
Listening on http://localhost:8000
Document root is /Users/ikronz/Desktop/Senior Project/simple-web-audio-recorder-demo-master
Press Ctrl-C to quit.
[Tue Jan 29 14:58:33 2019] ::1:53817 [200]: /
[Tue Jan 29 14:58:33 2019] ::1:53818 [200]: /style.css
[Tue Jan 29 14:58:33 2019] ::1:53819 [200]: /js/WebAudioRecorder.min.js
[Tue Jan 29 14:58:33 2019] ::1:53820 [200]: /js/app.js
[Tue Jan 29 14:58:34 2019] ::1:53821 [200]: /js/WebAudioRecorderWav.min.js
```

Figure 2 - Localhost server

The site started up, and the recording and playback functioned. First, the HTML and CSS were re-formatted to be better named, commented, formatted, and organized. Next, the Javascript was examined on the app and commented through to understand what was happening behind the scenes. The following are screenshots of the app.js file.

```

function startRecording() {
    console.log("startRecording() called");

    /*
     * Simple constraints object, for more advanced features see
     * https://addpipe.com/blog/audio-constraints-getusermedia/
     */

    var constraints = {audio: true, video: false}

    navigator.mediaDevices.getUserMedia(constraints).then(function(stream) {
        theLog("Succeeded in accessing user media");

        // Create an audio context after getUserMedia is called

        audioContext = new AudioContext();

        //update the format
        /* Unnecessary
        document.getElementById("formats").innerHTML="Format: 2 channel "+encodingTypeSelect.op
        tions[encodingTypeSelect.selectedIndex].value+" @ "+audioContext.sampleRate/1000+"kHz"
        */
        //assign to gumStream for later use
        gumStream = stream;

        /* use the stream */
        input = audioContext.createMediaStreamSource(stream);

        //stop the input from playing back through the speakers
        //input.connect(audioContext.destination)
        //get the encoding
        encodingType = encodingTypeSelect.options[encodingTypeSelect.selectedIndex].value;

        //disable the encoding selector
        encodingTypeSelect.disabled = true;

        recorder = new WebAudioRecorder(input, {
            workerDir: "js/", // must end with slash
            encoding: encodingType,
            numChannels:2, //2 is the default, mp3 encoding supports only 2
            onEncoderLoading: function(recorder, encoding) {
                // show "loading encoder..." display
                //theLog("Loading "+encoding+" encoder...");
            },
            onEncoderLoaded: function(recorder, encoding) {
                // hide "loading encoder..." display
                theLog(encoding+" encoder ready");
            }
        });

        recorder.onComplete = function(recorder, blob) {
-- INSERT --

```

Figure 3 - Javascript for audio recording

The above section creates an audioContext in order to start the recording. Then it creates a new instance of the WebAudioRecorder, specifies the encoding, and begins the recording process.

```

var url = URL.createObjectURL(blob);
var au = document.createElement('audio');
var li = document.createElement('li');
var link = document.createElement('a');

// Add controls to the audio thing
au.controls = true;
au.src = url;

// Create link a + blob
link.href = url;
link.download = new Date().toISOString() + '.'+encoding;
link.innerHTML = link.download;

// Add the new audio and a elements to the list
li.appendChild(au);
li.appendChild(link);

// Add the li element to the ordered list
recordingsList.appendChild(li);

```

Figure 4 - Javascript for audio blob creation

The above code snippet displays the creation of a blob and its output to the list. In the same function as the code shown above, Javascript that sent the blob to the server was added.

```

const theurl = 'backend.php';
var fd = new FormData();
//fd.append('fname', 'test.wav');
fd.append('data', blob);

// This may be where the issue lies

fetch(theurl, {
  method: 'POST',
  body: fd
}).then(response => {
  console.log(response);
});

```

Figure 5 - POST method to backend.php

This uses the POST method to hit the backend PHP script which does the final step of bringing the file into the system on the server side.

```

<?php
/* need to lookup isset */
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_FILES['files'])) {
        $errors = [];
        $path = 'uploads/';
        $extensions = ['jpg', 'jpeg', 'png', 'gif', 'ogg', 'wav', 'mp3', 'blob'];
    /*
        $upload_max_filesize = 10M;
        $post_max_size = 10M;
    */
        $all_files = count($_FILES['files']['tmp_name']);

        for ($i = 0; $i < $all_files; $i++) {
            $file_name = $_FILES['files']['name'][$i];
            $file_tmp = $_FILES['files']['tmp_name'][$i];
            $file_type = $_FILES['files']['type'][$i];
            $file_size = $_FILES['files']['size'][$i];
            $file_ext = strtolower(end(explode('.', $_FILES['files']['name'][$i])));

            $file = $path . $file_name;

            if (!in_array($file_ext, $extensions)) {
                $errors[] = 'Extension not allowed: ' . $file_name . ' ' . $file_type;
            }
    /*
            if ($file_size > 2097152) {
                $errors[] = 'File size exceeds limit: ' . $file_name . ' ' . $file_type;
            }
    */
            if (empty($errors)) {
                move_uploaded_file($file_tmp, $file);
            }
        }

        if ($errors) print_r($errors);
    }
}

```

Figure 6 - Backend PHP

The above PHP gets the file, checks for the filetype, gives it a name, and moves it into the /uploads folder. There was one glaring issue when dealing with this, which was that the maximum file size only allowed for uploads of less than 2MB, basically impossible for audio uploads (this is why image files are allowed through the backend.php, they were used for testing). After a little searching, the solution was found in editing a PHP config file for my laptop, PHP.ini, in /private/etc/. The following two lines were changed, thus allowing for much larger files to be uploaded.

```

upload_max_filesize = 10M;
post_max_size = 10M;

```

Figure 7 - PHP.ini upload size

After all the above was completed, the web app succeeded in doing everything expected of it. Requests can be seen to main page and its styling, as well as the Javascript files when recording. As shown in the figure below, after a recording had taken place, the backend PHP script was hit with the uploaded audio file.

```
[Mon Feb 4 13:16:26 2019] ::1:56453 [200] : /js/app.js
[Mon Feb 4 13:16:27 2019] ::1:56462 [200] : /js/WebAudioRecorderWav.min.js
[Mon Feb 4 13:16:27 2019] ::1:56463 [200] : /js/WebAudioRecorderWav.min.js
[Mon Feb 4 13:16:40 2019] ::1:56516 [200] : /backend.php
[Mon Feb 4 13:16:53 2019] ::1:56567 [200] : /
[Mon Feb 4 13:16:53 2019] ::1:56568 [200] : /style.css
[Mon Feb 4 13:16:53 2019] ::1:56569 [200] : /js/WebAudioRecorder.min.js
[Mon Feb 4 13:16:53 2019] ::1:56570 [200] : /js/app.js
[Mon Feb 4 13:17:55 2019] ::1:56824 [200] : /
[Mon Feb 4 13:17:55 2019] ::1:56825 [200] : /style.css
[Mon Feb 4 13:17:55 2019] ::1:56826 [200] : /js/WebAudioRecorder.min.js
[Mon Feb 4 13:17:55 2019] ::1:56827 [200] : /js/app.js
[Mon Feb 4 13:24:03 2019] ::1:57763 [200] : /js/WebAudioRecorderWav.min.js
[Mon Feb 4 13:24:04 2019] ::1:57764 [200] : /js/WebAudioRecorderWav.min.js
[Mon Feb 4 13:24:16 2019] ::1:57817 [200] : /backend.php
[Mon Feb 4 13:26:04 2019] ::1:58247 [200] : /backend.php
[Mon Feb 4 13:33:59 2019] ::1:60160 [200] : /
[Mon Feb 4 13:33:59 2019] ::1:60161 [200] : /style.css
[Mon Feb 4 13:33:59 2019] ::1:60162 [200] : /js/WebAudioRecorder.min.js
[Mon Feb 4 13:33:59 2019] ::1:60163 [200] : /js/app.js
[Tue Feb 19 09:50:34 2019] ::1:52759 [200] : /
[Tue Feb 19 09:50:34 2019] ::1:52766 [200] : /style.css
[Tue Feb 19 09:50:34 2019] ::1:52767 [200] : /js/WebAudioRecorder.min.js
[Tue Feb 19 09:50:34 2019] ::1:52774 [200] : /js/app.js
[Tue Mar 19 21:30:33 2019] ::1:55208 [200] : /js/WebAudioRecorderWav.min.js
[Tue Mar 19 21:30:34 2019] ::1:55209 [200] : /backend.php
[Tue Mar 19 21:30:38 2019] ::1:55210 [200] : /
[Tue Mar 19 21:30:38 2019] ::1:55211 [200] : /style.css
[Tue Mar 19 21:30:38 2019] ::1:55212 [200] : /js/WebAudioRecorder.min.js
[Tue Mar 19 21:30:38 2019] ::1:55213 [200] : /js/app.js
[Tue Mar 19 21:31:32 2019] ::1:55214 [200] : /
[Tue Mar 19 21:31:32 2019] ::1:55215 [200] : /style.css
[Tue Mar 19 21:31:32 2019] ::1:55216 [200] : /js/WebAudioRecorder.min.js
[Tue Mar 19 21:31:32 2019] ::1:55217 [200] : /js/app.js
[Tue Mar 19 21:32:07 2019] ::1:55218 [200] : /js/WebAudioRecorderWav.min.js
[Tue Mar 19 21:32:11 2019] ::1:55219 [200] : /backend.php
[Tue Mar 19 21:32:24 2019] ::1:55220 [200] : /
[Tue Mar 19 21:32:24 2019] ::1:55221 [200] : /style.css
[Tue Mar 19 21:32:24 2019] ::1:55222 [200] : /js/WebAudioRecorder.min.js
[Tue Mar 19 21:32:24 2019] ::1:55223 [200] : /js/app.js
[Tue Mar 19 21:32:29 2019] ::1:55225 [200] : /js/WebAudioRecorderMp3.min.js
[Tue Mar 19 21:32:29 2019] ::1:55226 [200] : /js/Mp3LameEncoder.min.js.mem
[Tue Mar 19 21:32:34 2019] ::1:55227 [200] : /backend.php
[Tue Mar 19 22:10:45 2019] ::1:55325 [200] : /js/WebAudioRecorderMp3.min.js
[Tue Mar 19 22:10:45 2019] ::1:55326 [200] : /js/Mp3LameEncoder.min.js.mem
[Tue Mar 19 22:10:54 2019] ::1:55329 [200] : /backend.php
```

Figure 8 - Server log

After setting up the PHP server that simply placed the uploaded files into a folder, a graphical solution was needed for viewing and interacting with those files. A plug-and-play solution was decided upon in order to not expand the scope to custom-writing an interface for it. After some research was done, a program called FileBrowser (“Filebrowser,” 2019) was decided upon to be a web front-end to the filesystem. Due to its easy-to-use yet powerful tools (user creation, authorization levels, etc.), it was a simple choice. It was extremely simple to get up and

running after installation, simply running the following command in the folder of choice would start up a server.

```
MacBook-Pro:uploads ikronz$ filebrowser
2019/01/30 18:17:36 No config file used
2019/01/30 18:17:36 Listening on 127.0.0.1:8080
2019/01/30 18:17:39 /api/renew: 403 127.0.0.1:65117 <nil>
2019/01/31 10:20:02 /api/renew: 403 127.0.0.1:51899 <nil>
2019/02/03 13:52:35 /api/renew: 403 127.0.0.1:62683 <nil>
2019/02/04 13:17:00 /api/renew: 403 127.0.0.1:56602 <nil>
2019/02/19 09:50:35 /api/renew: 403 127.0.0.1:52761 <nil>
2019/03/19 23:29:03 /api/renew: 403 127.0.0.1:55717 <nil>
2019/03/20 10:16:05 /api/renew: 403 127.0.0.1:56405 <nil>
```

Figure 9 - FileBrowser server log

After everything else had been set up, the final piece of the project was incorporating the foot pedal. The pedal that is used by Dr. Kronz is the Infinity IN-USB-2 (“Infinity Foot Pedals,” n.d.).



Figure 10 - Infinity foot pedal

After 5-10 hours of research and testing, there was no way of collecting the inputs from the foot pedal into the computer without proprietary software, let alone the browser. Meeting with Jason Kronz, a co-sponsor of the project and an electrical engineer, he decided the best course of action would be to change some of the insides. Using a USB Rubber Ducky and some

soldering, he took apart the inner embedded computer and replaced it with the Ducky, changing the outputs into normal keyboard outputs. This caused the foot pedal's outputs to act as if it was an HID device, readable by Javascript.

After the hacking of the foot pedal, time was spent investigating what input it sent, and how to read it best. The following is the code for getting the input from the foot pedal.

```
//webkitURL is deprecated but nevertheless
URL = window.URL || window.webkitURL;

// Pedal Down

document.addEventListener('keyup', function(event) {
  if (event.code == 'KeyM' && event.code == 'KeyM') {
    //alert('Undo!')
    console.log("hi");
    startRecording();
  }
});

// Pedal Up

document.addEventListener('keyup', function(event) {
  if (event.code == 'KeyN' && event.code == 'KeyN') {
    //alert('Undo!')
    console.log("bye");
    stopRecording();
  }
});
```

Figure 11 - Javascript for pedal inputs

Originally, the event.keyCode was used, but it was faulty and outdated, so the simpler, event.code was used instead. The foot pedal continuously sent the code “MM” twice to signal the pedal being held down, and it sent the code “NN” once when it was lifted off. These codes would be used to call the startRecording() and stopRecording() functions for each press and lift, respectively. In the future, the key codes for the other two pedals could be used for other functions, such as skipping ahead or resetting the recording.

There will not be future work on this project, due to the fact that it is functional and complete as it is, and the delay issues are solved in the second project. This project cannot very

well implement the binary streaming of the audio with its current PHP backend. There will be future work in the second project, discussed below.

The second project is not nearly as advanced and complete as the first. The reason for creating an entirely different project is that the solution requires streaming the actual binary from the web browser to the server and encoding on the server side. After hours of searching, only a couple solutions were found, these being ScriptProcessorNode (Khan, 2019) and RecordRTC (“ScriptProcessorNode,” 2019). Only one actual implementation of one of these was found, becoming the building blocks for this project (Poca, 2016). After running, however, it was apparent that there were some issues.

After awhile of troubleshooting, the solution to getting it to begin running was to update Node:

```
MacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ man node
2019-02-26 14:10:52.217 xcodebuild[40264:2763796] [MT] PluginLoading: Required plug-in compatibility UUID DFFB3951-EB9A-4C09-9DAC-5F2D28CC839C for plug-in at path '~/Library/Application Support/Developer/Shared/Xcode/Plug-ins/Unity4XC.xcplugin' not present in DVTPluginCompatibilityUUIDs
2019-02-26 14:10:52.220 xcodebuild[40264:2763796] Failed to load plugin at: /Users/ikronz/Library/Application Support/Developer/Shared/Xcode/Plug-ins/Unity4XC.xcplugin, skipping. Reason: [for failure: *** -[__NSPlaceholderDictionary initWithObjects:forKeys:count:]: attempt to insert nil object from objects[0]]
MacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ node -v
v8.9.4
MacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ node --update
node: bad option: --update
MacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ npm -v
5.7.1

Update available 5.7.1 → 6.4.1
Run npm i -g npm to update

MacBook-Pro:simple-web-audio-recorder-demo-master ikronz$ npm i -g npm
^C
) ? loadRequestedDeps: still install loadAllDepsIntoIdealTree

Update available 5.7.1 → 6.8.0
Run npm i -g npm to update

npm ERR! cb() never called!
npm ERR! This is an error with npm itself. Please report this error at:
npm ERR! <https://github.com/npm/npm/issues>

npm ERR! A complete log of this run can be found in:
npm ERR! /Users/ikronz/.npm/_logs/2019-02-26T21_12_10_954Z-debug.log
```

Figure 12 - Node updates

Updating Node was not the only thing necessary to get it up and running. There were many missing modules that needed to be installed. An example of erroring and installing a missing module are shown below.

```
MacBook-Pro:browser-pcm-stream-master ikronz$ node app.js
module.js:540
  throw err;
  ^

Error: Cannot find module 'express'
    at Function.Module._resolveFilename (module.js:538:15)
    at Function.Module._load (module.js:468:25)
    at Module.require (module.js:587:17)
    at require (internal/module.js:11:18)
    at Object.<anonymous> (/Users/ikronz/Desktop/Senior Project/browser-pcm-stream-master/app.js:1:77)
    at Module._compile (module.js:643:30)
    at Object.Module._extensions..js (module.js:654:10)
    at Module.load (module.js:556:32)
    at tryModuleLoad (module.js:499:12)
    at Function.Module._load (module.js:491:3)
MacBook-Pro:browser-pcm-stream-master ikronz$ ls
LICENSE.TXT  app.js      public
README.md   package.json  tpl
MacBook-Pro:browser-pcm-stream-master ikronz$ npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN MicStreamToWav@0.0.0 No repository field.
[npm WARN MicStreamToWav@0.0.0 No license field.

+ express@4.16.4
added 48 packages from 36 contributors in 4.175s
MacBook-Pro:browser-pcm-stream-master ikronz$ node app.js
```

Figure 13 - NPM debug console output

Using this process, the Express, Binary.js, WAV, and Jade modules were installed. This finally ended with the successful launch of the app, displayed in the results section. After some research in Jade and CSS, the app was changed to make it a little more personable.

Future work on this area basically means adding all of the features that project one had. This includes foot pedal support, playback on the same window, and a playback from the uploaded server (via Filebrowser). These features should be completed by the next version of this thesis.

Results

As discussed previously, this project ended up with two separate implementations. The first was a more finished product written with a PHP backend, and the second utilizing Node.js with various other supporting JavaScript libraries. The first codebase, while successfully interfacing with the keyboard and mic and having a prettier user interface, suffers from the few second delays caused by client-side encoding. The second codebase, while only a skeleton of a project, successfully does away entirely with client-side encoding, streaming the binary to a Node server, removing any client delay. The results section will begin by displaying the first, more completed product.

The following figure is a screenshot of the finished initial web page.

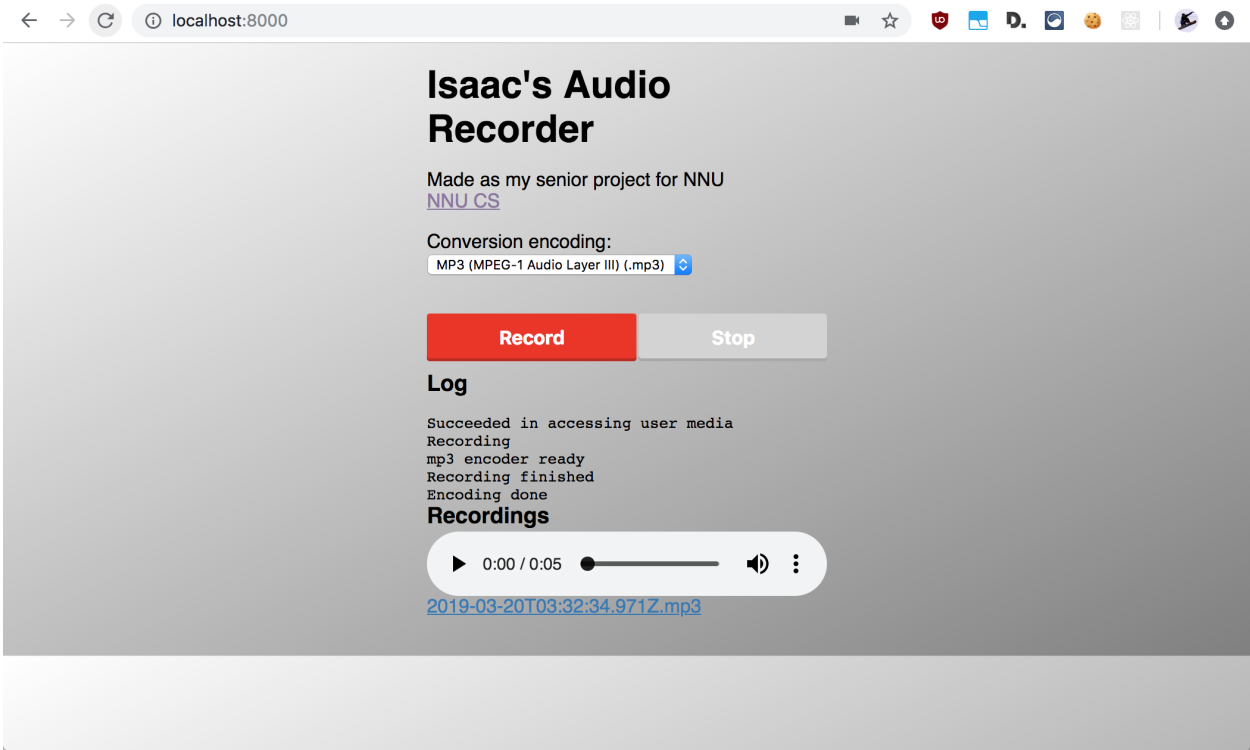


Figure 14 - First project web page

The background is a CSS linear gradient from white in the upper left to grey in the lower right, in order to look perhaps a bit more modern than a plain white or something more complex. To begin, the user would first begin by selecting one of the three options for audio formats.

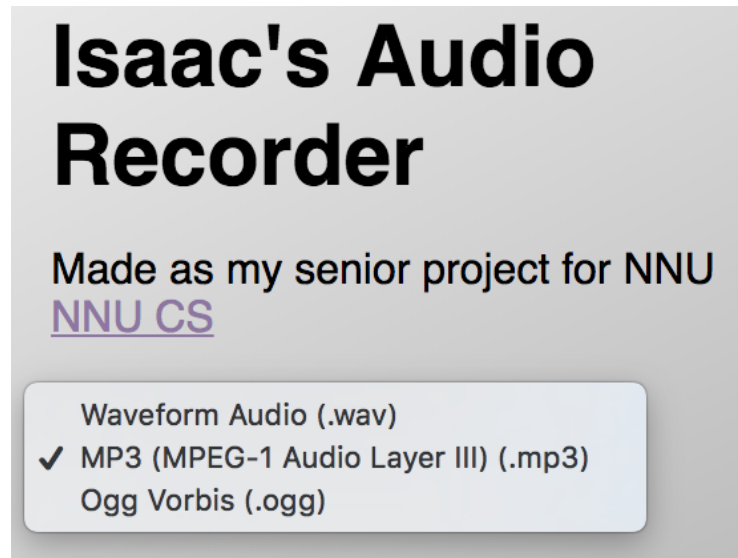


Figure 15 - Encoding selection

For the purposes of experimentation three options, WAV, MP3, and OGG are given for output, even though the OGG format ends up with the smallest files, and therefore is the clear choice. If the user has an external mic, they can change the mic input up in the URL bar.

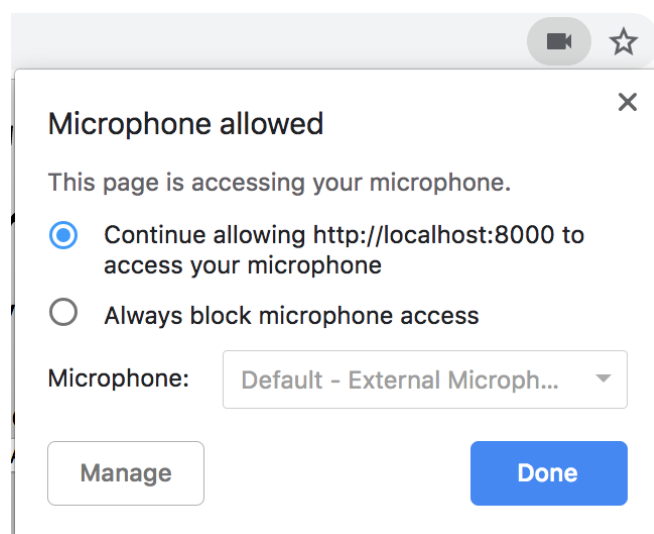


Figure 16 - Microphone selection

Once the user is ready for recording, they will either press the record button with their mouse or using a USB foot pedal. The record button, once pressed, will become grey, and the stop button will become red, until it is pressed. It does this same change once the foot pedal is pressed or released. This allows for more clarity and a nicer feeling user interface.



Figure 17 - Record and stop buttons

Once the recording has begun, either error messages or the normal success messages will be output in the Log section.

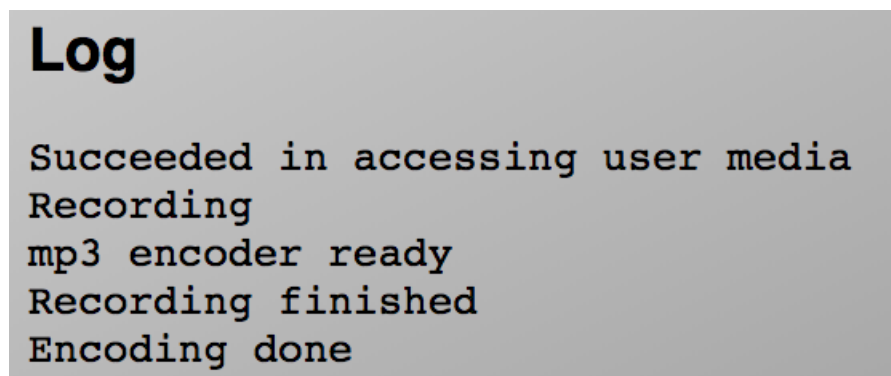


Figure 18 - Web log

The first three lines in this figure display a successful recording begun, and the last two lines are printed to the log if a successful recording is completed. Once the recording is complete, two things take place. The audio blob is uploaded to the PHP server, and an entry appears in the Recordings section.

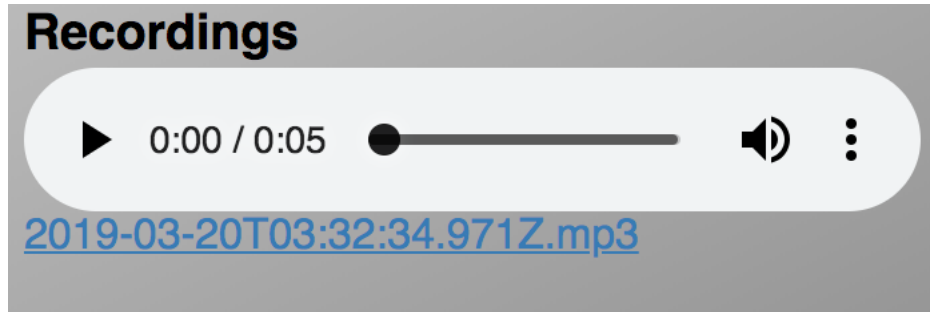


Figure 19 - Recorded audio player

A user can adjust value, play, and pause the recent recording, as well as download via the link generated beneath the player. Once the user has done their recording, the transcriber has a second interface in order to listen, move, play, pause, delete, or organize the data.

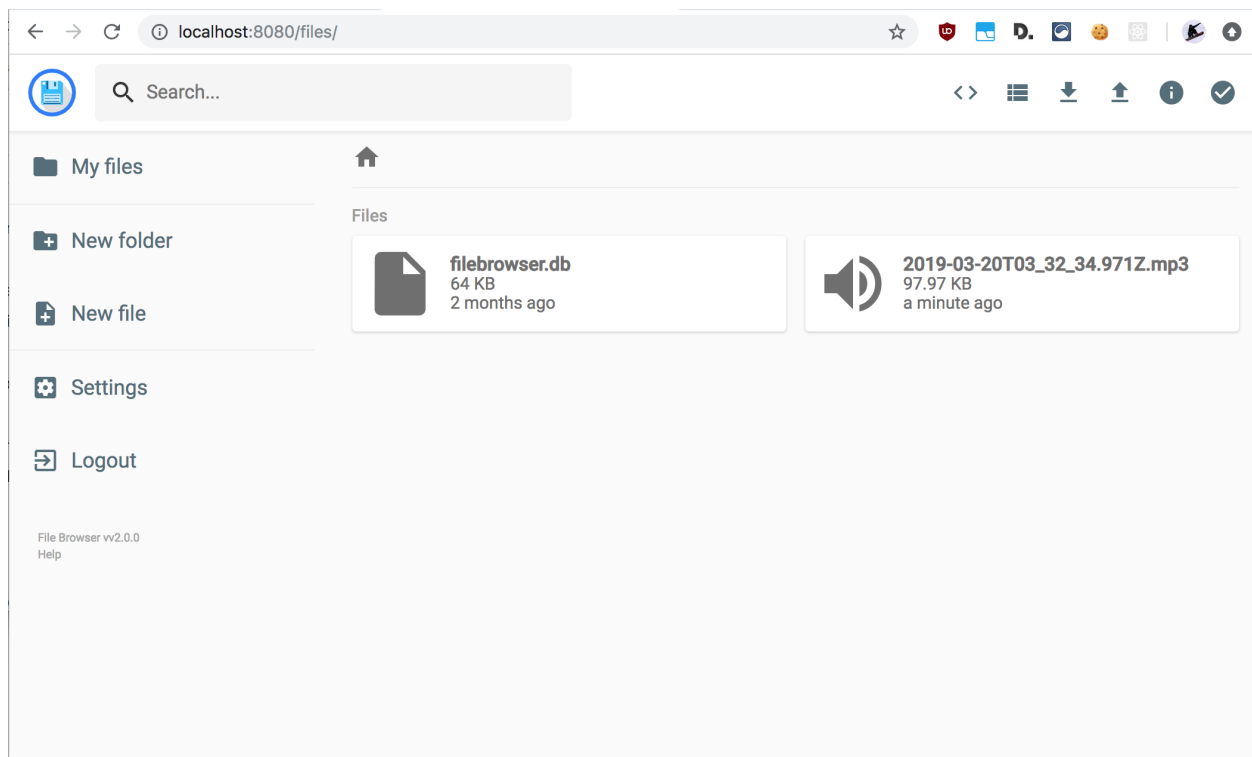


Figure 20 - FileBrowser web interface

Filebrowser hosts a local server that gives a Google Drive-like interface. Simple design allows for an easy interface for anyone on the transcribing end to use.

This system works very well, except for the encoding and uploading delay. This delay is roughly three seconds of encoding and uploading to the server per minute of audio recorded. This amount of delay does not allow the user to hit record until after it is finished encoding, and results in lower productivity, which is unacceptable to pathologists. The current delay on encoding is a problem, and is the reason that work began on a separate, second project.

The second project is not nearly as advanced and complete as the first. As of present time, it is very barebones and has yet to have the features of the first project added. Currently, it lacks the features of having a well-designed user interface, working with a foot pedal, or having a server-side interface for playback. It does, however, solve the delay problem.



Figure 21 - Second version web page

Once the Record button is pressed, it begins a binary stream to the Express server, and on that side it collects the stream until the Stop button is pressed. After that, it encodes the collected stream into an MP3 file on the server side, therefore eliminating encoding delays on the client

side. Future work should see the features of the first project incorporated into the second project, with redesigned UI, foot pedal support, server side playback, and perhaps a few other features.

Conclusion

I enjoyed working on this project because it felt, in a way, that I was breaking some new ground. Plenty of web-based audio recorders have been made before me, and a couple tying in with keyboard HID devices, but none that I found had implemented all those features with streaming the audio directly to the server, causing no delay. The summer before my senior year I had spent doing web development, so it felt nice to be able to work in an area I was a little more familiar with.

My uncle, Jason, is an electrical engineer and the president of BuildingReports, so this also allowed me to get a taste of the higher expectations of independent, contract-like work. A little more pressure can make things a bit more interesting for sure. Joseph and Jason Kronz's push and requests for features gave me a scope and a time I had to work.

This project almost more than any other showed me a lot about other's work. Both of my projects were built on top tons of work others have done, from Node/BinaryJS/Express to simple Javascript examples of implementing audio recording. This gave me both examples of good coding practices and extremely lazy (yet efficient) programming. Utilizing a lot of this code also gave me practice putting different code bases, with my own changes, together. This is a skill I have to become a master at for my future work in web development, and something I definitely experienced every day at my internship this last summer.

In conclusion, I feel like I learned a lot about working with USB's and audio in the web browser. I hope browsers continue working on pushing the boundaries of what Javascript is

capable of, while still keeping security a focus. I would have liked to add some more features, but I feel what I got completed was adequate, and even though it is not a fully-featured product, it is a solid proof-of-concept and something that can easily be improved upon.

References

1. Naicu, O. (2018, July 13). Simple-web-audio-recorder-demo. Retrieved from <https://github.com/addpipe/simple-web-audio-recorder-demo>
2. Altoedge. (n.d.). Infinity Foot Pedals. Retrieved from <http://www.altoedge.com/pedals/vec-infinity-foot-pedals.html>
3. Filebrowser. (2019, February 28). Filebrowser. Retrieved from <https://github.com/filebrowser/filebrowser>
4. Higuma, Y. (2016, June 11). Web-audio-recorder-js. Retrieved from <https://github.com/higuma/web-audio-recorder-js>
5. Khan, M. (2019, April 08). Muaz-khan/WebRTC-Experiment. Retrieved from <https://github.com/muaz-khan/WebRTC-Experiment/tree/master/RecordRTC>
6. Poca, G. (2016, September 17). Browser-pcm-stream. Retrieved from <https://github.com/gabrielpoca/browser-pcm-stream>
7. ScriptProcessorNode. (2019, March 23). Retrieved from <https://developer.mozilla.org/en-US/docs/Web/API/ScriptProcessorNode>

Appendix A - Version 1.0

root

back.php

```
<?php

// Created by Isaac Kronz -
// Experimental (and functional) backend written the day before
presentation
// ^ previous backend.php is deprecated but is kept because it
may be useful

if (isset($_FILES['thing']))
{

    $uploads_dir = 'uploads/';
    $tmp_name = $_FILES["thing"]["tmp_name"];

    $name = basename($_FILES["thing"]["name"]);
    move_uploaded_file($tmp_name, "$uploads_dir$name");
}
?>
```

backend.php

```
<?php

// Currently deprecated but still has useful functionality to
carry over
// to back.php

echo "Hello World";
```

```

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_FILES['thing'])) {
        $errors = [];
        $path = 'uploads/';
        $extensions = ['ogg', 'wav', 'mp3', 'blob'];
/*
        $upload_max_filesize = 10M;
        $post_max_size = 10M;
*/

        $all_files = count($_FILES['thing']['tmp_name']);

        for ($i = 0; $i < $all_files; $i++) {
            $file_name = $_FILES['thing']['name'][$i];
            $file_tmp = $_FILES['thing']['tmp_name'][$i];
            $file_type = $_FILES['thing']['type'][$i];
            $file_size = $_FILES['thing']['size'][$i];
            $file_ext = strtolower(end(explode('.',
$_FILES['thing']['name'][$i])));

            $file = $path . $file_name;

            if (!in_array($file_ext, $extensions)) {
                $errors[] = 'Extension not allowed: ' .
$file_name . ' ' . $file_type;
            }

/*
            if ($file_size > 2097152) {
                $errors[] = 'File size exceeds limit: ' .
$file_name . ' ' . $file_type;
            }

```

```

        */
        if (empty($errors)) {
            move_uploaded_file($file_tmp, $file);
        }
    }

    if ($errors) print_r($errors);
}
}

```

index.html

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Isaac Audio Recorder</title>
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <h1>Isaac's Audio Recorder</h1>
    <p>Made as my senior project for NNU<br><a
href="https://math.nnu.edu" target="_blank">NNU CS</a></p>
    <!-- Selection of audio encoding type -->
    <p>Conversion encoding:<br>
    <select id="encodingTypeSelect">
        <option value="wav">Waveform Audio (.wav)</option>
        <option value="mp3">MP3 (MPEG-1 Audio Layer III)
(.mp3)</option>
        <option value="ogg">Ogg Vorbis (.ogg)</option>

```

```

</select>
</p>
<!-- Input section -->
<div id="controls">
  <button id="recordButton">Record</button>
  <button id="stopButton" disabled>Stop</button>
</div>
<div id="formats"></div>
<h3>Log</h3>
<pre id="log"></pre>

<h3>Recordings</h3>
<ol id="recordingsList"></ol>
  <!-- inserting these scripts at the end to be able to use
all the elements in the DOM -->
  <script src="js/WebAudioRecorder.min.js"></script>
  <script src="js/app.js"></script>
</body>
</html>

```

launch.sh

```

#!/bin/bash
# Beginning of a script to launch the entire project
# as opposed to multiple manual php servers being created
php -S localhost:1000

```

php.ini

```

upload_max_filesize = 10M;
post_max_size = 10M;

```

README.md

```
# Functional WebAudioRecorder/Uploader using  
WebAudioRecorder.js
```

A web audio recorder that supports multiple forms of mics and a foot pedal.

After recording and encoding the audio, it uploads to a server.

Utilizes [[WebAudioRecorder.js](https://github.com/higuma/web-audio-recorder-js)] (<https://github.com/higuma/web-audio-recorder-js>) to record mp3, wav and Vorbis audio on a web page.

style.css

```
html {  
    background-image: linear-gradient(to bottom right, white,  
grey);  
}  
* {  
    padding: 0;  
    margin: 0;  
}  
a {  
    color: #337ab7;  
}  
p {  
    margin-top: 1rem;  
}  
a:hover {  
    color:#23527c;  
}  
a:visited {
```



```

    color: #8d75a3;
}
body {
    margin: 1rem;
    padding: 1rem;
    font-family: sans-serif;
    max-width: 21rem;
    margin: 0 auto;
    position: relative;
}
#controls {
    display: flex;
    margin-top: 2rem;
}
/*
#recordButton {
    background: green;
}
*/
button {
    flex-grow: 1;
    height: 2.5rem;
    /*min-width: 1rem;*/
    border: none;
    border-radius: 0.15rem;
    background: #ed341d;
    margin-left: 2px;
    box-shadow: inset 0 -0.15rem 0 rgba(0, 0, 0, 0.2);
    cursor: pointer;
    display: flex;
    justify-content: center;
    align-items: center;
}

```

```

    color:#ffffff;
    font-weight: bold;
    font-size: 1rem;
}
button:hover, button:focus {
    outline: none;
    background: #c72d1c;
}
button::-moz-focus-inner {
    border: 0;
}
button:active {
    box-shadow: inset 0 1px 0 rgba(0, 0, 0, 0.2);
    line-height: 3rem;
}
button:disabled {
    pointer-events: none;
    background: lightgray;
}
button:first-child {
    margin-left: 0;
}
audio {
    display: block;
    width: 100%;
    margin-top: 0.2rem;
}
li {
    list-style: none;
    margin-bottom: 1rem;
}
#formats {

```

```
margin-top: 0.5rem;
font-size: 80%;
}
```

js

app.js

```
// Written by Isaac Kronz and code from the Addpipe demo (link
in README.md)
```

```
//webkitURL is deprecated but nevertheless
URL = window.URL || window.webkitURL;
```

```
// Pedal Down
```

```
document.addEventListener('keyup', function(event) {
  if (event.code == 'KeyM' && event.code == 'KeyM') {
    //alert('Undo!')
    console.log("hi");
    startRecording();
  }
});
```

```
// Pedal Up
```

```
document.addEventListener('keyup', function(event) {
  if (event.code == 'KeyN' && event.code == 'KeyN') {
    //alert('Undo!')
    console.log("bye");
    stopRecording();
  }
});
```

```

/*
var code = "";
window.addEventListener("keyup",function(e) {
    code = (code+String.fromCharCode(e.keyCode ||
e.which)).substr(-11);
    if( code == "mm") {
        window.removeEventListener("keyup",arguments.callee);
        // do stuff here
    }
},false);
*/
// \// this is doing it on any key being pressed. I don't know
exactly why.
/*
document.onkeypress = function(e) {
    e = e || window.event;
    console.log(e);
    var charCode = (typeof e.which == "number") ? e.which :
e.keyCode;
    if(charCode = "m") {
        startRecording();
    }
    else {
        stopRecording();
    }
    //if(e = "[n^n]" {
    // stopRecording();
    //}
    //if (charCode) {
    // alert("Character typed: " +
String.fromCharCode(charCode));

```

```

    //}
};
*/
var gumStream;          //stream from getUserMedia()
var recorder;          //WebAudioRecorder object
var input;             //MediaStreamAudioSourceNode we'll be recording
var encodingType;     //holds selected encoding for resulting
audio (file)
var encodeAfterRecord = true;          // when to encode

var AudioContext = window.AudioContext ||
window.webkitAudioContext;
var audioContext; //new audio context to help us record

var encodingTypeSelect =
document.getElementById("encodingTypeSelect");
var recordButton = document.getElementById("recordButton");
var stopButton = document.getElementById("stopButton");

// Add events to the buttons
recordButton.addEventListener("click", startRecording);
stopButton.addEventListener("click", stopRecording);

function startRecording() {
    console.log("startRecording() called");

    /*
        Simple constraints object, for more advanced features
see
        https://addpipe.com/blog/audio-constraints-getusermedia/
    */
}

```

```

*/

    var constraints = {audio: true, video: false}

navigator.mediaDevices.getUserMedia(constraints).then(function(s
tream) {
    thelog("Succeeded in accessing user media");

    // Create a audio context after getUserMedia is called

    audioContext = new AudioContext();

    //update the format
    /* Unnecessary
    document.getElementById("formats").innerHTML="Format: 2
channel
"+encodingTypeSelect.options[encodingTypeSelect.selectedIndex].v
alue+" @ "+audioContext.sampleRate/1000+"kHz"
*/
    //assign to gumStream for later use
    gumStream = stream;

    /* use the stream */
    input = audioContext.createMediaStreamSource(stream);

    //stop the input from playing back through the speakers
    //input.connect(audioContext.destination)
    //get the encoding

```

```

    encodingType =
encodingTypeSelect.options[encodingTypeSelect.selectedIndex].value;

    //disable the encoding selector
encodingTypeSelect.disabled = true;

recorder = new WebAudioRecorder(input, {
    workerDir: "js/", // must end with slash
    encoding: encodingType,
    numChannels:2, //2 is the default, mp3 encoding
supports only 2
    onEncoderLoading: function(recorder, encoding) {
        // show "loading encoder..." display
        //thelog("Loading "+encoding+" encoder...");
    },
    onEncoderLoaded: function(recorder, encoding) {
        // hide "loading encoder..." display
        thelog(encoding+" encoder ready");
    }
});

recorder.onComplete = function(recorder, blob) {
    thelog("Encoding done");
    createDownloadLink(blob, recorder.encoding);
    encodingTypeSelect.disabled = false;
}

recorder.setOptions({
    timeLimit:120,
    encodeAfterRecord:encodeAfterRecord,
    ogg: {quality: 0.5},

```

```

        mp3: {bitRate: 160}
    });

    //start the recording process
    recorder.startRecording();

    thelog("Recording");

    }).catch(function(err) {
        //enable the record button if getUserMedia() fails
        recordButton.disabled = false;
        stopButton.disabled = true;

    });

    //disable the record button
    recordButton.disabled = true;
    stopButton.disabled = false;
}

function stopRecording() {
    console.log("stopRecording() called");

    //stop microphone access
    gumStream.getAudioTracks()[0].stop();

    //disable the stop button
    stopButton.disabled = true;
    recordButton.disabled = false;

    //tell the recorder to finish the recording (stop recording
+ encode the recorded audio)

```



```

recorder.finishRecording();

thelog('Recording finished');
}

function createDownloadLink(blob,encoding) {

    //const theurl = 'backend.php';
    const theurl = 'back.php';
    const fd = new FormData();
    //fd.append('fname', 'test.wav');
    fd.append('thing', blob);
    console.log(fd.get('thing'));
    // This may be where the issue lies, but I'm 99% sure it's
the php

    fetch(theurl, {
        method: 'POST',
        body: fd
    }).then(response => {
        console.log(response);
    });

    var url = URL.createObjectURL(blob);
    var au = document.createElement('audio');
    var li = document.createElement('li');
    var link = document.createElement('a');

    // Add controls to the audio thing
    au.controls = true;
    au.src = url;

```

```

// Create link a + blob
link.href = url;
link.download = new Date().toISOString() + '.'+encoding;
link.innerHTML = link.download;

// Add the new audio and a elements to the list
li.appendChild(au);
li.appendChild(link);

// Add the li element to the ordered list
recordingsList.appendChild(li);
}

// This adds to the log and the list

function thelog(e, data) {
    log.innerHTML += "\n" + e + " " + (data || '');}

```

Appendix B - Version 2.0

root

app.js

```
// Written by Gabriel Poca, additions by Isaac Kronz
// Sets up Express and Binary.js
// Takes the binary stream as input and uses filewriter to
create a file

var express = require('express');
var BinaryServer = require('binaryjs').BinaryServer;
var fs = require('fs');
var wav = require('wav');

var port = 3700;
var outFile = 'demo.wav';
var app = express();

app.set('views', __dirname + '/tpl');
app.set('view engine', 'jade');
app.engine('jade', require('jade').__express);
app.use(express.static(__dirname + '/public'))

app.get('/', function(req, res){
  res.render('index');
});

app.listen(port);

console.log('server open on port ' + port);
```

```

binaryServer = BinaryServer({port: 9001});

binaryServer.on('connection', function(client) {
  console.log('new connection');

  var fileWriter = new wav.FileWriter(outFile, {
    channels: 1,
    sampleRate: 48000,
    bitDepth: 16
  });

  client.on('stream', function(stream, meta) {
    console.log('new stream');
    stream.pipe(fileWriter);

    stream.on('end', function() {
      fileWriter.end();
      console.log('wrote to file ' + outFile);
    });
  });
});

```

package.json

```

{
  "name": "MicStreamToWav",
  "version": "0.0.0",
  "description": "Stream microphone pcm data to server and save
on wav file.",
  "scripts": {
    "start": "node app.js"
  },

```

```
"dependencies": {
  "binaryjs": "^0.2.1",
  "express": "^4.16.4",
  "jade": "^1.11.0",
  "wav": "^1.0.2"
},
"author": "gabrielpoca"
}
```

README.md

Made by Isaac Kronz using a lot of code from Gabriel Poca's Github.

Instructions:

```
prompt@username~$ node app.js
```

Browser: <http://localhost:3700>

public

recorder.js

```
// Written by Gabriel Poca, with additions from Isaac Kronz
// Sets up the client side of the Javascript, creating a
binary.js tunnel
//

(function(window) {
  var client = new BinaryClient('ws://localhost:9001');

  client.on('open', function() {
```

```

window.Stream = client.createStream();

if (!navigator.getUserMedia)
    navigator.getUserMedia = navigator.getUserMedia ||
navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia || navigator.msGetUserMedia;

if (navigator.getUserMedia) {
    navigator.getUserMedia({audio:true}, success, function(e) {
        alert('Error capturing audio.');
```

 });
} else alert('getUserMedia not supported in this browser.');

```

var recording = false;

window.startRecording = function() {
    recording = true;
}

window.stopRecording = function() {
    recording = false;
    window.Stream.end();
}

function success(e) {
    audioContext = window.AudioContext ||
window.webkitAudioContext;
    context = new audioContext();

    // the sample rate is in context.sampleRate
    audioInput = context.createMediaStreamSource(e);
```

```

var bufferSize = 2048;
recorder = context.createScriptProcessor(bufferSize, 1, 1);

recorder.onaudioprocess = function(e) {
  if(!recording) return;
  console.log ('recording');
  var left = e.inputBuffer.getChannelData(0);
  window.Stream.write(convertoFloat32ToInt16(left));
}

audioInput.connect(recorder)
recorder.connect(context.destination);
}

function convertoFloat32ToInt16(buffer) {
  var l = buffer.length;
  var buf = new Int16Array(l)

  while (l--) {
    buf[l] = buffer[l]*0xFFFF;    //convert to 16 bit
  }
  return buf.buffer
}
});
})(this);

```

tpl

index.jade

```
doctype
```

```
html
```

```
  head
```

```

    title= "Isaac's Binary Stream"

script(src="https://cdn.jsdelivr.net/binaryjs/0.2.1/binary.min.js")

    style.
        html {
            background-image: linear-gradient(to bottom right, white, grey);
        }
        body {
            text-align: center;
            margin-top: 300px;
            margin-bottom: 300px;
        }
        p {
            font-size: 36;
        }
body
    div
        h1 Isaac's Recorder V2
        button(onclick="startRecording()") Record
        button(onclick="stopRecording()") Stop
        script(src="recorder.js")

```

style.css

```

html {
    background-image: linear-gradient(to bottom right, white, grey);
}

```