# PROJECT REPORT

PETER FRANGATZIS – P3200234

MULTIMEDIA TECHNOLOGY

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

*ABSTRACT*

This document serves as a user guide, a technical report and high-level documentation on my **audio and music visualizer** project (*visualize.me*) *for* the class "Multimedia Technology" at AUEB. For a faster read, I recommend reading the project's README, which also contains an overview and instructions on installing and getting started with the application.

January 10th, 2025

# TABLE OF CONTENTS

# 1 USER GUIDE

If you have worked with Node.js projects before, the following instructions should be self-explanatory.

## 1.1 INSTALLATION

Firstly, you will need to install Node.js (http://nodejs.org). This will automatically install "npm" alongside it, Node's package manager. To install the required npm packages, navigate to the project's directory and do:

```
$ npm install
```

Confirm that the packages installed correctly with:

```
$ npm list
```

The output should look something like this:

```
<project_directory>@ <path_to_project>
├── express@version
├── multer@version
└── nodemon@version
```

The p5.js library is **not** cited as a dependency since it is used to do frontend stuff, like graphics and audio playback. It is fetched from a CDN (specifically http://cdnjs.org) in the index.html page inside the directory called 'public'.

## 1.2 USAGE

To start serving the application from your local machine, use:

```
$ npm run start
```

By doing so, an express.js server will start listening on port 9090 and you can start using the application! On your local machine, open your browser visit http://localhost:9090.

Currently tested in these web browsers:

- **Mozilla Firefox**.
  - 134.0 20250106205921 20250106205921 (Arch Linux)
  - 128.6.0esr-1~deb12u1 [Debian Bookworm (stable) 12.8]
- **Chromium** 131.0.6778.264 Arch Linux

# 2 DOCUMENTATION

Visualizing music has always fascinated me. It is a thing that has been around since before I was born, only I discovered it around 2012 with the rise of Electronic Dance Music (EDM) at the time (and a couple years before). YouTube channels of EDM labels like Monstercat, NCS and Trap Nation saw great popularity back then, not just for the music they released, but also by their distinct style of visualization.


*Figure 1: Monstercat's unique visualizer with vertical rectangles and particles in the background.*

I will split this part into 3 sections: general overview, backend and frontend. Most of the complexity happens at the frontend, since that is where the rendering happens.

## 2.1 OVERVIEW

*Visualize.me* is a very basic audio/music visualizer written in JavaScript using the p5.js framework and p5.sound, which is an official addon that extends p5 with more audio processing functionality. Since it only makes sense to view the visualization while listening to the audio, *visualize.me* also implements a music player. Audio playback can be controlled by right-clicking anywhere inside the browser window. The application comes with 4 (royalty-free) songs included for the user to try out, but also supports uploading audio files to visualize. There is (currently) support for 3 different visualization methods, each one focusing on a distinct feature of sound.

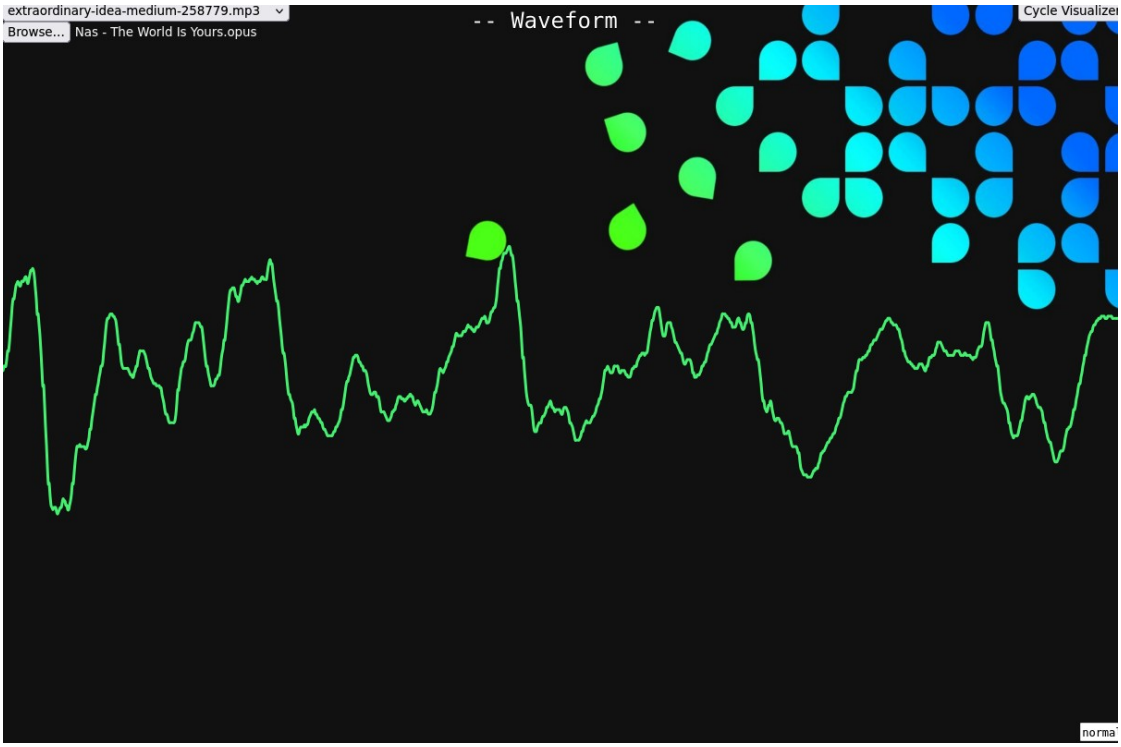


*Figure 2: Screenshot of visualize.me playing an uploaded song and drawing a linear interpolation of its waveform inside the browser window.*

## 2.2 BACKEND

When we start serving the application, Node.js runs the file index.js. In there, we configure and set up a very basic express server that listens on port 9090. The server uses the directory 'public' as the content root from which it sends requested files (index.html by default for a regular HTTP GET). We also enable parsing for JSON and URL-encoded data for communication on endpoints (for example '/uploads', which I will talk about next).

Supporting audio file uploads was challenging, but npm's 'multer' package made it a *lot* easier. Multer is middleware for handling multipart/fom-data. We set it up alongside express and tell it to save files at 'public/assets/uploads'. For a file to be processed and saved this way, an HTTP POST request has to be sent to '/uploads' from an HTML form's `<input>` element, that has `name="custom_audio"` and `type="file"`. The uploaded file gets saved at the specified location with a unique prefix added to it's name. Finally, an HTTP response is returned that contains the unique file name. The frontend needs this information in order to request it from the server and start it's playback.

## 2.3 FRONTEND

As mentioned previously, the express server sends index.html by default when receiving an HTTP GET request. Inside our HTML page we load the p5.js library and it's addon from a Content Delivery Network (CDN). We also fetch some stylesheets and, most importantly, our p5 sketch, called 'sketch.js'.

Most p5.js sketches have the same structure: you only need to override the two global functions `setup` and `draw`. There are more functions one can override, for example the handy

`windowResized`. Our sketch starts from the overriden `preload` function where we load our first song and a static image that is used for the background. Then we move to `setup` and then the `draw` loop starts.

```
function preload() {
     snd = loadSound("assets/songs/" + sndFiles[0]);
     imgBg = loadImage("assets/images/bg.jpg");
}
```

I will not go into much detail of how UI elements are organized/positioned or how a user's file ends up at the server etc. in order to keep this report high level. I will mostly focus on how we achieve the visualization part of the project. You can browse the source code at your own pace.

The main tool we use is `p5.FFT`, which is a wrapper object around common audio processing functions involving Fourier Transforms, like getting a representation of the frequency spectrum of an input signal in array form (`p5.FFT.analyze`). `p5.FFT`'s other advantage is that we can configure the length of the resulting array and even apply smoothing to the contained values. Another useful object is `p5.Amplitude`, that helps with measuring the amplitude of a sound waveform.

```
amp = new p5.Amplitude();
amp.setInput(snd);
amp.toggleNormalize();
fft = new p5.FFT(fftSmooth, fftSz);
fft.setInput(snd);
```

By using `toggleNormalize`, `p5.Amplitude` finds the difference between the loudest reading it has processed and the maximum amplitude of 1.0. It adds this difference to all values to produce results that reliably map between 0.0 and 1.0. However, if a louder moment occurs, the amount that normalization adds to all the values will change.

The first visualizer uses `p5.FFT.waveform`, which returns an array of sample values from the input audio. We can map the values we got to the canvas space by applying some tricks and drawing the corresponding points. P5.js has a dedicated mode for drawing shapes, where points are joined together. So we end up with a linear interpolation of the sample values.

```
/* Linear interpolation on sample values. */
let wav = fft.waveform();
stroke(colR, colG, colB);
strokeWeight(3);
noFill();  // Don't fill the area under the curve
beginShape();
for (let i = 0; i < width; ++i) {
     let idx = floor(map(i, 0, width, 0, wav.length));
     curveVertex(i, wav[idx] * 300 + height / 2);
}
endShape();
```

The second visualizer displays the frequency spectrum we get from `p5.FFT.analyze` as vertical lines, the number of which equals the FFT analysis size (`fftSz`). To fit each frequency on the canvas, we divide the canvas by the number of frequencies and give each line/rectangle that much width.

```
/* Frequency spectrum as vertical bars. */
let fspec = fft.analyze();
let colw = width / fftSz;
stroke(colR, colG, colB);
for (let i = 0; i < fspec.length; ++i) {
     let y = map(fspec[i], 0, 255, height, height/4);
     line(i * colw, height, i * colw, y);
}
```

The third one is the simplest. It draws a circle with radius equal to `p5.Amplitude.getLevel` in the middle of the canvas. In my experience, I have always had to scale the resulting amplitude level.

```
/* Amplitude as a circle. */
let scaling = 500;
let vol = map(amp.getLevel(), 0, 1, 0, scaling);
fill(colR, colG, colB);
ellipse(width / 2, height / 2, vol, vol);
```

# 3 CREDITS

## 3.1 SONGS

1. *Level VII Short*, by moodmode from Pixabay.
2. *Extraordinary Idea Medium*, by moodmode from Pixabay.
3. *Music for Arcade Style Game*, by lucadialessandro from Pixabay.
4. *Cool Driver*, by Emmraan from Pixabay.

## 3.2 IMAGES & ICONS

1. Graph icons created by Retinaicons - Flaticon
2. Background image by dr15