



# E-Music Box

Jack Madden

G00338579

BEng (Hons) Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

## Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

## Acknowledgments

I would like to thank Paul Lennon, my project supervisor, for providing me with guidance and assistance during this project.

I would also like to thank John Lawless, who helped learn how to use MongoDB.

## Table of Contents

Declaration.....	2
Acknowledgments.....	3
Glossary.....	5
Introduction .....	6
Project Goals .....	7
Structure of Report .....	7
Project Architecture Diagram .....	8
Block Diagram .....	8
Expanded Architecture Diagram .....	9
Software .....	10
Introduction .....	10
IDE .....	10
Front end.....	11
Sign Up .....	12
Log In.....	12
View All .....	13
Playback .....	14
Upload.....	15
Delete .....	15
Back end.....	16
Server .....	16
Database .....	18
Audio Storage.....	19
Conclusion.....	21
References .....	23
Additional Learning Resources.....	23

## Glossary

API: Application Programming Interface

CSS: Cascading Style Sheets

EJS: Embedded JavaScript

FLAC: Free Lossless Audio Codec

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IDE: Integrated Development Environment

JS: JavaScript

JSON: JavaScript Object Notation

NPM: Node Package Manager

SQL: Structured Query Language

TLD: Top Level Domain

URL: Uniform Resource Locator

WAV: Waveform Audio

## Introduction

The intent of my project was to develop a web API for free music upload, playback, and download.

A fully working implementation of this concept could be very useful in the world of video making. It could especially benefit makers of YouTube videos who would like to insert background music into their videos. Such people typically have neither the time nor the technology to write their own music for every video they make. Using copyrighted audio is also out of the question because YouTube does not provide any way for creators to acquire a license from record labels to use their music. Moreover, the website's Content ID system is quick to block or remove videos that contain any copyrighted content[1][2].

The only option left in this situation is to use royalty-free music; however, this also comes with its own issues. For starters, YouTube's own audio library is not unlimited, and therefore many of the same pieces of music appear in countless different videos, which can come across as lazy to perceptive viewers. It is possible to search the Internet for more music; however, while this is royalty-free, it commonly requires paying money in order to access the library. A product such as this would save them from dealing with these problems.

### Project Goals

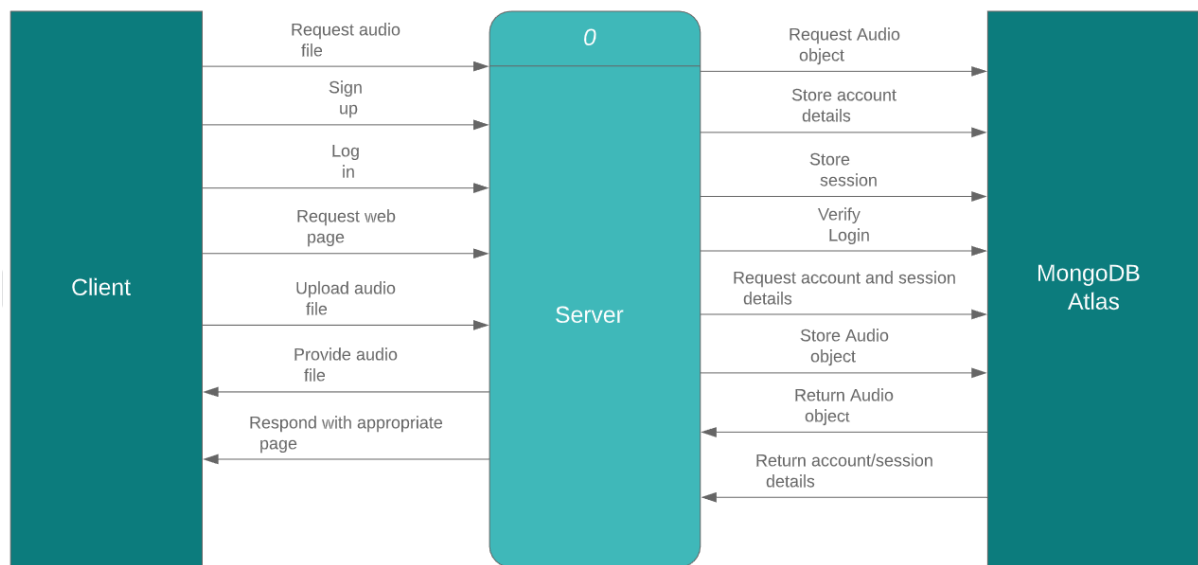
- Create a page with working audio playback
- Develop an uploading system with restrictions on file format and required filename and creator metadata
- Implement a signup/login system and limit uploading to logged in users
- Add a file deletion feature and protect files from deletion by anyone but their creator

### Structure of Report

- Project Architecture Diagram
- Software
- Conclusion
- References

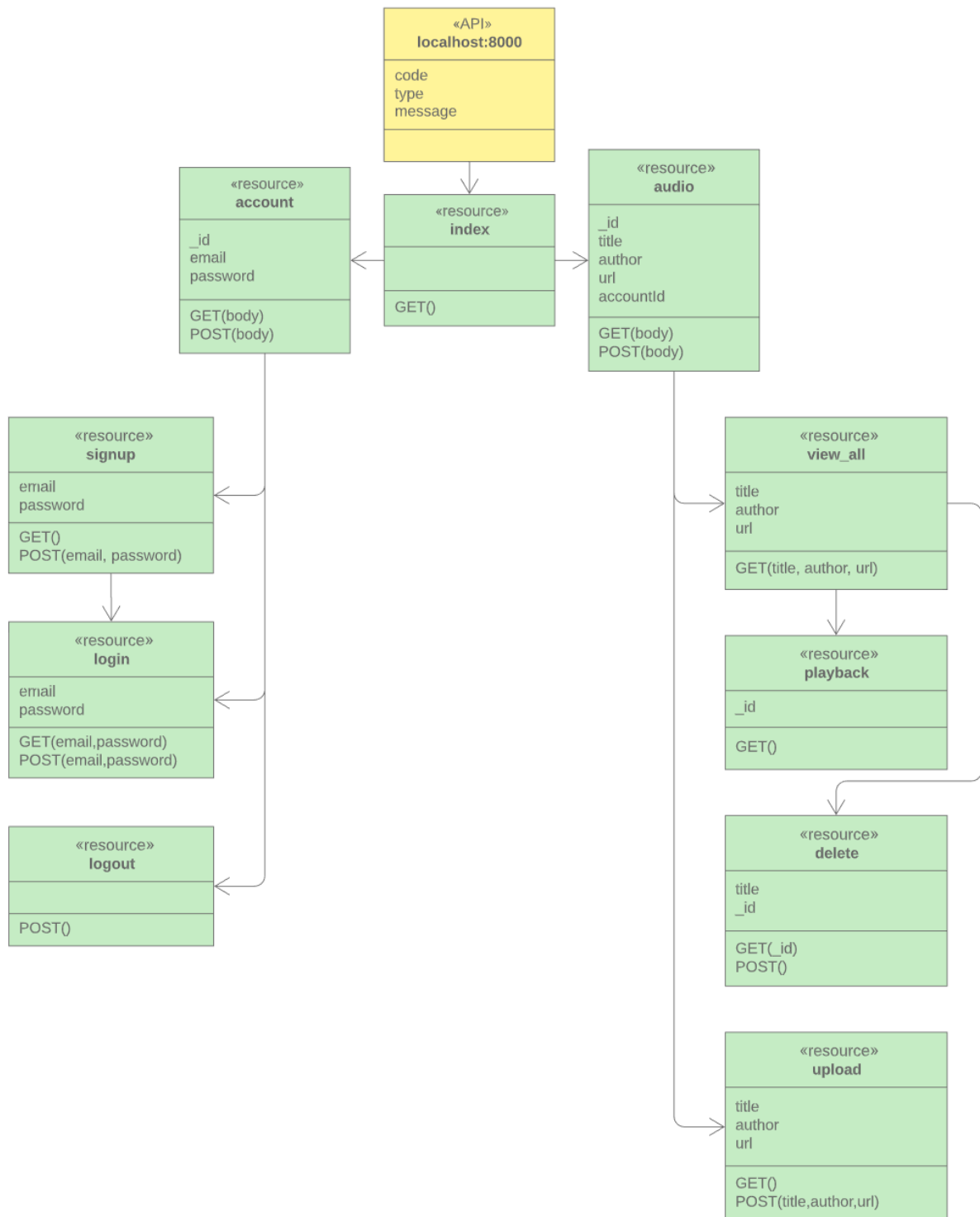
## Project Architecture Diagram

### Block Diagram





## Expanded Architecture Diagram



## Software

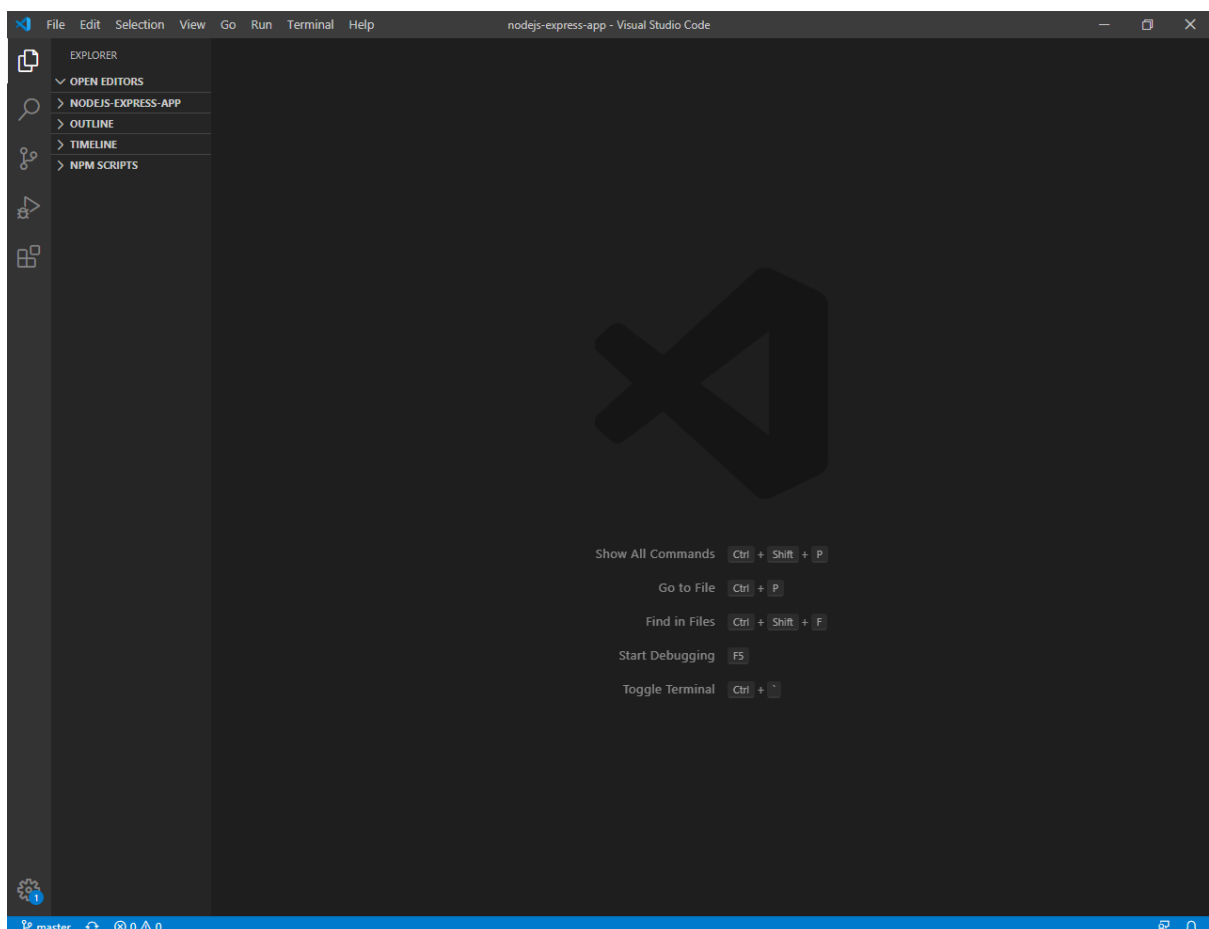
### Introduction

All the components of any web application fall into two main categories—the front end and the back end.

The **front end** is the part of the application the client interacts with and which provides them access to the functions of the system. The **back end** consists of the server and database and is responsible for processing all incoming requests from the client. Communication between the two works via a series of HTTP GET and POST requests from the client and responses from the server. GET requests are used to load pages. POST requests send form data to the server.

### IDE

I chose to use Visual Studio Code as my IDE for this project.



## Front end

The front end consists of a collection of web pages that send the GET and POST requests to the server. Like all web pages, they are written using a mixture of HTML, CSS, and JavaScript. However, to improve productivity, Node.js offers a choice of several templating engines so that developers do not have to use the raw forms of all three languages.

My first choice of templating engine was Pug (formerly known as Jade.) This proved to be a mistake, as Pug has a very different syntax from HTML that I could not get used to. Later, I switched to EJS, which I found significantly more comfortable to use.

Compared with Pug, EJS has a syntax much more akin to standard HTML. What differentiates it from HTML, however, is the ability to insert JavaScript wherever needed more easily using the characters `<% %>`. Combined with a character next to the left percentage sign, this character sequence can serve three different functions:

- `<% %>`: Flow control, e.g. conditional statements (`<% if (...) { %> ... <% } else { %> ... <% } %>`) or loops (`<% for (...) { %> ... <% } %>`)
- `<%= %>`: Escaped output, e.g. variables (`Welcome <%= username %> !`)
- `<%- %>`: Unescaped output

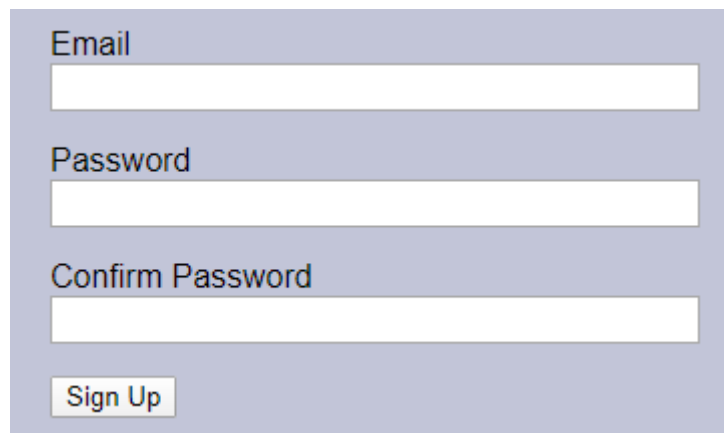
[3]

In addition to allowing the easy display of variables on web pages, the escaped output feature simplifies URL concatenation.

There are six features available to the client:

- Sign up
- Log in
- View all audio files
- Play an audio file
- Upload an audio file
- Delete an audio file

## Sign Up

A sign-up form with a light blue background. It contains three text input fields: 'Email', 'Password', and 'Confirm Password'. Below the fields is a 'Sign Up' button.

If a user wishes to sign up, they must provide an email address and password, and they must confirm their password. Not everything entered in the Email text field will be accepted as a valid email; it must end with “@<email>.<tld>” to be accepted.

Passwords also have limitations. They must be a minimum of 8 characters in length and can only contain alphanumeric characters.

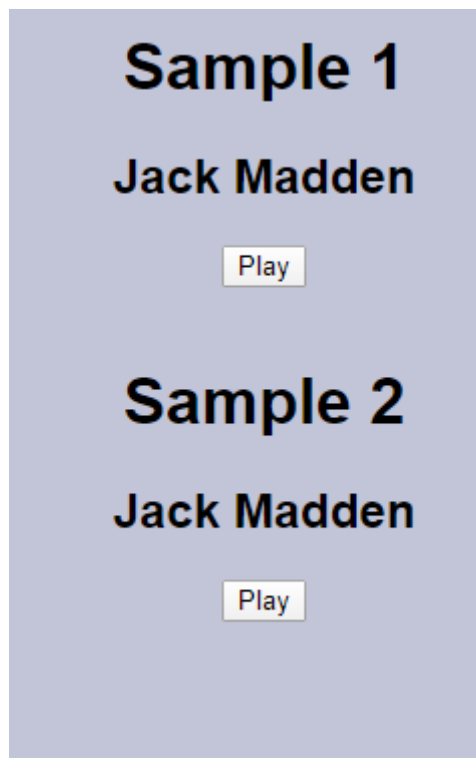
## Log In

A log-in form with a light blue background. It contains two text input fields: 'Email' and 'Password'. Below the fields is a 'Log In' button.

Once a user has registered an email, they may log in. They can also log out when they are done.

Logging in is required to upload and delete files. However, viewing, playing, and downloading existing files is free for anyone to do.

[View All](#)

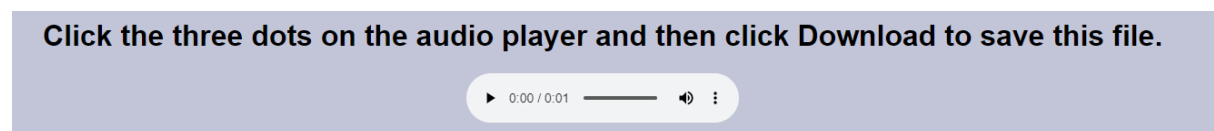


On the View All page, the user is shown a list of all available audio files. They are displayed vertically in order of upload, starting with the oldest. Every file on this page is rendered as title, author, link to playback page. To make sure all files appeared properly, I used a schema so that every audio file would have the same fields.



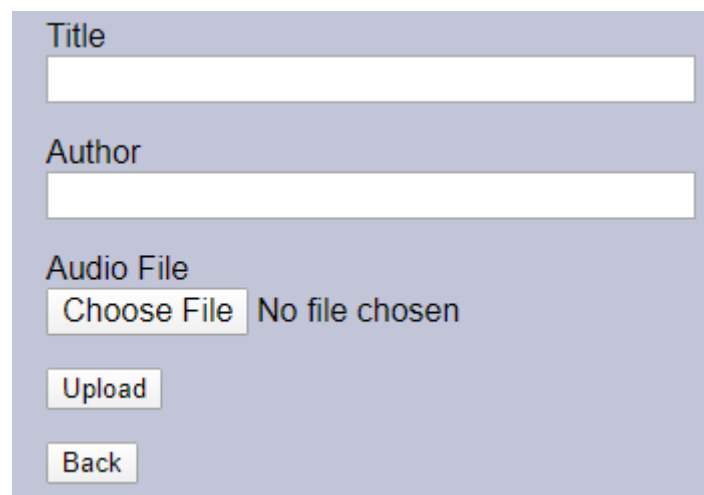
Logged-in users will also see the option to delete files. While the button is visible for all files, it only works for the account that uploaded it. For everyone else, the button does nothing.

### Playback



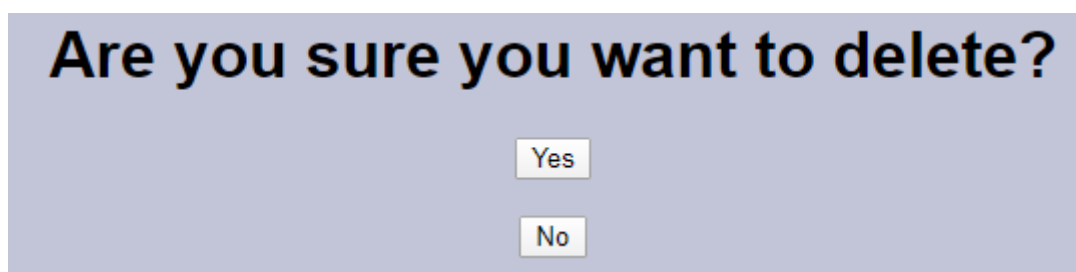
On the playback page, the user can press play to preview a piece of music. If they like it, they can then download it and use it themselves.

## Upload

A screenshot of a web form titled 'Upload'. The form has a light blue background. It contains three input fields: 'Title', 'Author', and 'Audio File'. The 'Title' and 'Author' fields are empty text boxes. The 'Audio File' field is a file selection button labeled 'Choose File' with the text 'No file chosen' next to it. Below the 'Audio File' field are two buttons: 'Upload' and 'Back'.

On the Upload page, the user must give a title and author and attach the file they wish to upload. If either the title or author fields are left blank, the upload will be rejected. There are also limitations on what file formats can be uploaded to block files that are not audio (see more in Audio section.)

## Delete

A screenshot of a confirmation dialog box with a light blue background. The text 'Are you sure you want to delete?' is displayed in a large, bold, black font. Below the text are two buttons: 'Yes' and 'No'.

Since the Delete button is very close to the Play button when looking at View All, it would be very easy to accidentally delete files if this button simply sent a POST request to delete. Therefore, this button instead sends a GET request for a separate Delete page, and this is where the relevant file may be deleted. Upon deletion, the server reloads the View All page, where the file is now gone. Alternatively, the user can go back to View All without deleting if they change their mind.

## Back end

### Server

The server runs on **Node.js**, a runtime environment designed to allow the use of server-side JavaScript. I used the **Express** framework to avoid having to write all the server code from scratch. The most important files in Node.js are *app.js* and *package.json*. *app.js* is responsible for:

- Connecting to MongoDB
- Session creation
- Error handling
- Blocking uploads of Audio objects with invalid file formats
- Controlling audio file storage
- Setting the port number

Meanwhile, *package.json* contains a list of all the dependencies used in the project and their versions. These dependencies are all installed in the *node\_modules* folder.

The *package.json* file is extremely powerful for one reason: if a developer makes a copy of someone else's Node.js application, but is lacking the necessary packages to run it, they can just run the `npm install` command in the terminal of their IDE and NPM will install every dependency listed in *package.json*.

This functionality is immensely useful for developers who publish their source code to online repositories on services such as GitHub. *node\_modules* is enormous in size relative to the rest of the code in a Node application; in my case, by the final version of my project, it was 66.6 MB, compared to my own code which was 239 kB—a difference of 285×.

Without *package.json*, developers would need to commit and push dependencies along with their own source code, which would be both time-consuming and incredibly wasteful of server memory resources. With it, they need only push their source code (a *.gitignore* file is used to let the local repository know not to track *node\_modules*), and for people cloning or downloading from the repository, it will do the heavy lifting on their end.

The rest of my code is stored in folders. Some of these folders include *controllers*, *routes*, and *views*.



The controllers folder contains JS files with the names and logic of all GET and POST methods. It would be possible to place it all in one file, but the result would be very large and difficult to understand. Therefore, methods are split into two files, *audio.js* and *login.js*, depending on whether they relate to the handling of audio files or to signup and login.

In the audio controller, the `getDelete` method performs a check before it will load the delete page. Every Audio object has an `accountId` key that is automatically set upon creation and is taken from the `_id` of whichever account performed the upload. If the `accountId` of the file to be deleted matches the `_id` of the account making the GET request. If they match, the user will be taken to the delete page. If they do not, they will instead be redirected back to the View All page.

The routes folder contains one JS file, *index*. This file works as an extension of the controllers files and assigns each method from them a URL on which they can be accessed on the client side. For uploading and deletion, this is also where access is restricted to users who are logged in. Furthermore, it contains the controller for the index page, which has no functionality for either audio files or accounts.

The GET methods for the deletion and playback pages have a variable attached to their end: `audioid`. This means that each audio file gets its own URL for deletion and playback, but all these pages look like each other, similarly to how every YouTube video, Twitter account, or Wikipedia article has the same page layout.

The views folder has a collection of EJS files. These are the pages that get loaded in the browser. The appearance of some pages changes depending on whether the user is signed in or not.

Another folder in node is *public*. There is nothing in this besides a subfolder called *stylesheets*, which contains CSS files. Due to my limited knowledge of CSS, I did not make much use of it, only using it to centre text, set the background colour and font, and format the layout of form pages (sign up, log in, upload, delete).

## Database

For my database, I elected to use **MongoDB**, a NoSQL, non-relational database program that stores data in JSON-like documents.[4]

While MongoDB does not require that documents have a schema defined beforehand, I chose to use schemas to enforce uniformity within collections for Account and Audio objects. There are two main reasons for this. First, it greatly simplifies the process of displaying all available audio files, because the field names are identical across all Audio documents. This also makes login verification possible. Second, it allows me to mark fields as required and make the server reject documents that leave them empty.

To create the schemas, I imported the Schema constructor from Mongoose and set the key names and their data types and made them all required. The JS files containing the schema definitions were stored in the *models* folder in Node.js. To make them available in other files, I used `module.exports`.

## Audio Storage

Both Account and Audio documents are stored in MongoDB Atlas, but while accounts can be stored directly in the database, the same is not possible with audio files. The fundamental issue is size: audio files—even the smallest ones—are huge compared to database documents, and for most audio formats their size increases linearly with time, so that a two-minute file is twice as big as a one-minute file, assuming both files are of the same type and bit rate.

WAV audio, which is uncompressed, has a bit rate of more than 1,200 kilobits per second, meaning that a three-minute song in this format is about 30 MB. For comparison, most MongoDB documents are typically only a few hundred bytes in size. Lossy compression formats such as MP3 can reduce files to about 10 percent of their original size[5] (this is known as having a 10 percent compression ratio), but even with this, they are still thousands of times bigger than is practical to store in the database.

The solution is to not store them there at all. To demonstrate, here is a typical entry in the *accounts* collection:

```
_id: ObjectId("5ec07fc9a66b740e604a9171")
email: "johndoe@email.com"
password: "$2a$12$1bniswPlMph672Hg68nsfOvw1aQULkMcC88w2pJAwfpvX/orw0T0."
__v: 0
```

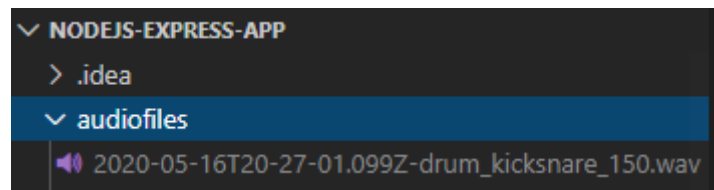
As expected, it is directly storing the email address and encrypted password.

Now here is an entry in the *audios* collection:

```
_id: ObjectId("5ec04c95eefcd73694129974")
title: "Sample File"
author: "Jack Madden"
url: "audiofiles\2020-05-16T20-27-01.099Z-drum_kicksnare_150.wav"
accountId: ObjectId("5ec04c6eeefcd73694129973")
__v: 0
```

There is no file here; instead, there is a `url` field. This is pointing to a folder in Node.js called *audiofiles*.

Looking at this folder in Node reveals the following:



This is the file that gets retrieved and played back. All MongoDB needs to do is tell the client where to look for it. All audio files are stored in this manner, which allows Audio objects to be fewer than 200 bytes on average.

Their size is unaffected by file size; an hour of WAV audio will need no more space in the database than a highly compressed, two second MP3.

As mentioned above in the Server section, app.js prevents uploads of invalid file types from being saved or having associated documents created. The server accepts three file formats:

- WAV (.wav)
- MP3 (.mp3)
- Vorbis (.ogg)

HTML5 audio supports only these file types.[6] I could allow the upload of any file, but this would be a waste of space since it could not be played back. In addition to blocking all non-audio file formats (e.g. PNG, GIF, ZIP), this also means that the upload mechanism is incompatible with FLAC audio. FLAC is a lossless compression format that has a compression ratio of 50–70 percent for most digital audio files.[7] It is inherently less effective than lossy compression, but the tradeoff is that on playback, this file is decompressed to its original bit rate with no loss in quality. Presumably, this is the reason why HTML5 audio does not support it, as the player would need a built-in codec, while files in other formats are simply read and played as they are.

## Conclusion

As stated near the beginning of the report, the goals of my project were to get an audio playback page working, allow people to upload and check files before accepting them to prevent uploads of non-audio files, let users sign up and log in and hide uploading from logged out users, and allow them to delete previous uploads while protecting them from being deleted by others. I believed that such a product could be useful for people working in video editing and dealing with tight schedules and budgets that disallowed the composition of original music or use of licensed music.

I believe I have accomplished these goals. My project is successfully storing, retrieving, and playing uploaded audio, rejecting uploads with invalid file formats or empty metadata, creating and storing accounts, limiting upload and delete functions to signed in users, and preventing accounts from deleting other accounts' files.

Nevertheless, there is more I would like to add to this project in the future. For starters, right now, uploading is limited to single files at a time. Ideally, it could be expanded to allow uploading of folders or albums as well. A more sophisticated search function would also help the project.

Furthermore, for use in a production environment, a copyright system and more rigorous email verification would be necessary. Currently, it is not capable of automatically detecting stolen content, which would lead to rampant piracy if not changed. If the site were to grow big enough without addressing this, it could end up facing lawsuits and being shut down.

While the project does have email verification as it is, it only checks to see that an @, email service, and dot-TLD are present, not that the address is real. Upgrading this system would be vital for real-world deployment.

Finally, there is the issue of funding. No website is truly free; servers must be maintained round the clock and occasionally need repairs. There would be three main options for revenue:

- Direct charges/subscription fees
- Advertising
- Donations

The first option would go completely against my original goal of not making people pay up front for using the site. The second option is unreliable because of the prevalence of ad blockers on the modern internet. The third option is most suitable for my intentions and would keep the site from being beholden to advertisers with questionable business ethics.

Realistically, regardless of the choice for funding, the site would probably lose money for at least the first year.

Despite these issues, however, I am pleased with what I was able to build with the time and knowledge I had.

## References

- [1] J. Katzowitz, 'YouTube Stars Say Copyright Claims Are Unfairly Targeting Them'. [Online]. Available: <https://www.dailydot.com/upstream/youtube-copyright-claims-thefatrat/>.
- [2] C. Wood, 'YouTubers MxR and PotasticPanda face \$6,000 bill over a copyright breach - Business Insider'. [Online]. Available: <https://www.businessinsider.com/youtuber-mxr-potasticpanda-face-6000-bill-copyright-breach-2020-1?r=US&IR=T>. [Accessed: 17-May-2020].
- [3] 'ejs - npm'. [Online]. Available: <https://www.npmjs.com/package/ejs>.
- [4] 'What Is MongoDB? | MongoDB'. [Online]. Available: <https://www.mongodb.com/what-is-mongodb>.
- [5] 'MP3 (MPEG Layer III Audio Encoding)'. [Online]. Available: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000012.shtml>.
- [6] 'HTML Audio'. [Online]. Available: [https://www.w3schools.com/html/html5\\_audio.asp](https://www.w3schools.com/html/html5_audio.asp).
- [7] S. Pigeon, 'Looking At Flac Compression Ratios | Harder, Better, Faster, Stronger'. [Online]. Available: <https://hbfs.wordpress.com/2012/02/07/looking-at-flac-compression-ratios/>.

## Additional Learning Resources

Academind, 'Node.js Basics'. [Online] Available:

[https://www.youtube.com/playlist?list=PL55RiY5tL51oGJorjEgl6NVeDbx\\_fO5jR](https://www.youtube.com/playlist?list=PL55RiY5tL51oGJorjEgl6NVeDbx_fO5jR)

'HTML Tutorial'. [Online] Available: <https://www.w3schools.com/html/default.asp>

'CSS Tutorial'. [Online] Available: <https://www.w3schools.com/css/default.asp>

'JavaScript Tutorial'. [Online] Available: <https://www.w3schools.com/js/default.asp>