BCS DIGITAL INDUSTRIES APPRENTICESHIP SYNOPTIC PROJECT

SOFTWARE DEVELOPMENT - MUSIC PLAYER

Wentao Shum

Definition,Investigation and Analysis

Definition - Music player

The definition of a music player is a device or application for playing digital audio files 1. This could relate to mp3 players, webpages that have a music player within, phone apps that play music. Digital audio files can consist of .mp3's, AAC's, wav's, FLAC's, ALAC's and DSD's 2. the most common of these mp3's and this should be the main focus of the format of the music player.

Key Features and additional information

According to the specifications the music player that I am creating must allow for must user control over playback, the music player must be able to go into an idle or power saving mode if no user interaction is recorded for over a 30 second period,the music player must have a search option for the file within the media database, the music player must list display options by song track or album, the music player must allow for a creation of a song playlist. Additional information is that Rebmem is looking to expand to supply interactive goods, they produce portable storage devices and have worked on a new device that will allow the storage and playback of music. They would like a simple interface to interact with the device to allow the continuation of development. My role is to take on the development of the interface ensuring that you meet the first phase requirements, with any additional elements you can include will help but deadline is 4.00pm 05/11/2021, the hardware is still being tested so changes will be required as it is developed. Ensuring that I document each step to allow any changes later.

Potential Problems

A potential problem I could face is when a user tries to upload a file which is not an audio file, I would need to make sure that the music player either ignores the file or lets the user know that the file within the folder is not an applicable audio file.

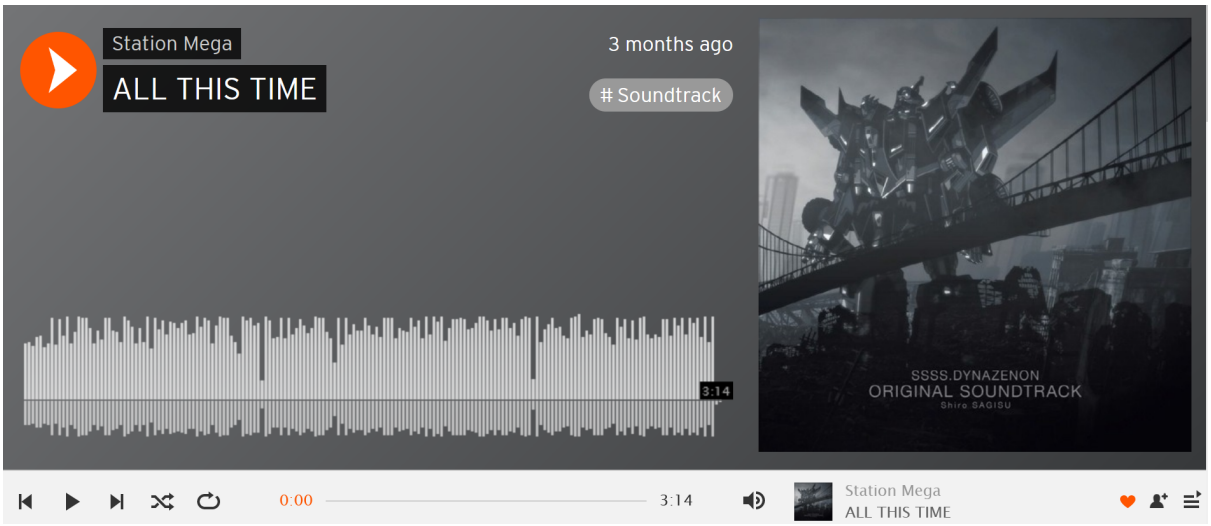
Stakeholders

My stakeholders would be the people that are using the music player on the new device developed, this is a wide range of users, men or woman, from young ages to old ages, experienced with technology or new to technology however to that the UI needs to be simple according to the specification, with such a wide range of users I would need to make a ergonomically pleasing music player.

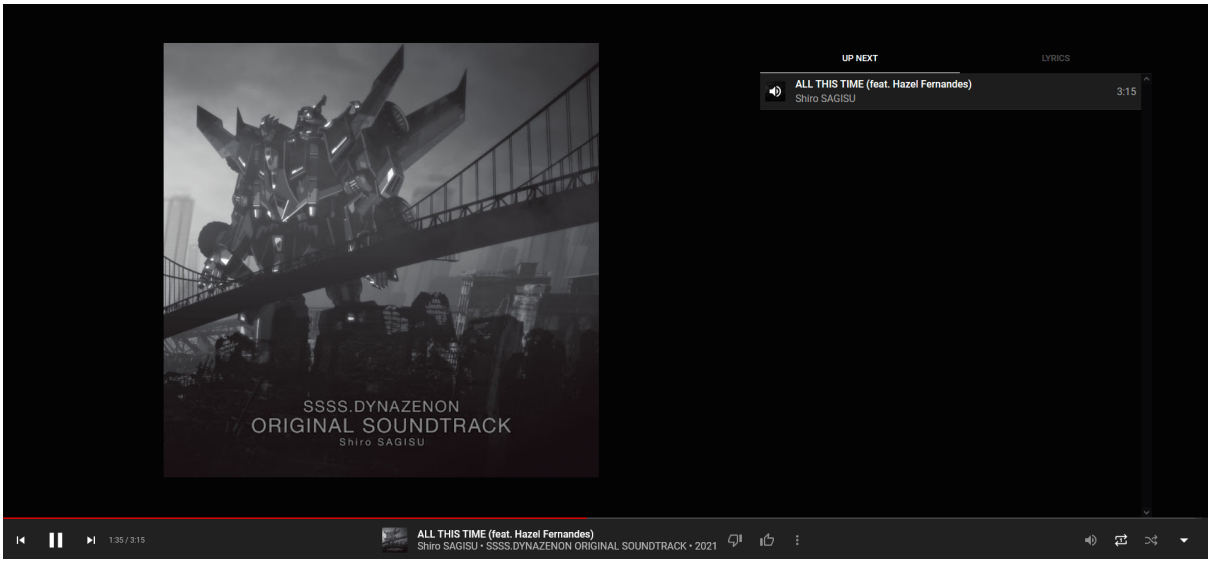
Research of Existing

This is where I've researched existing music players. I believe in order to make a successful music player I should look at existing music players. The two music players I have decided to take a look at are Soundcloud 3. and Youtube Music 4.. These are similarly popular, soundcloud has been around since 2007 and has been overshadowed, it's one that I use personally and it's quite an easy way to listen to music for free without ads.Youtube music is

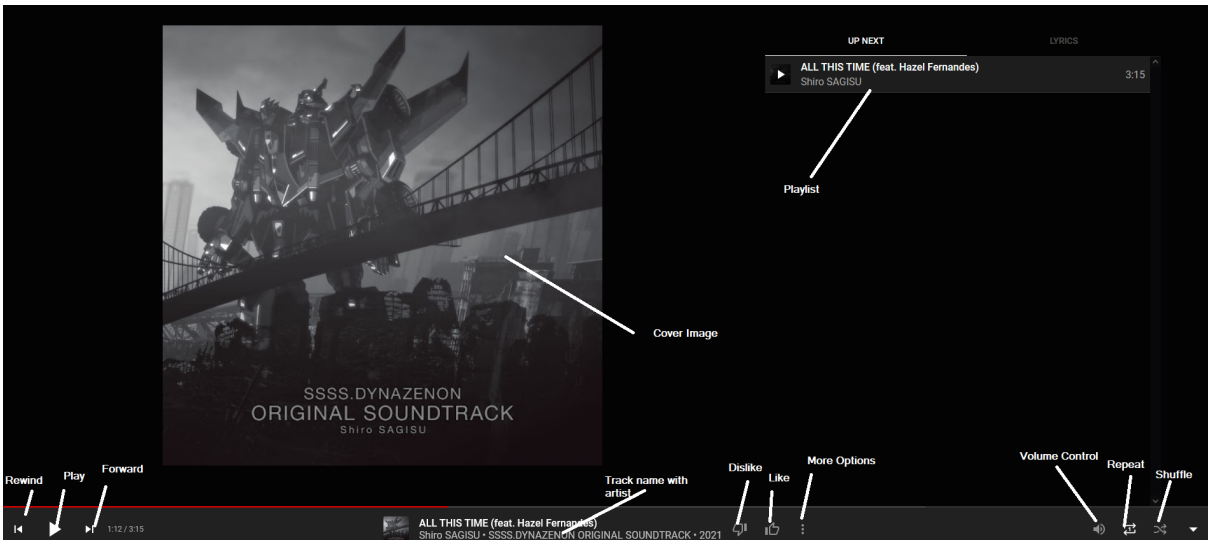
one of the newer music players being released in 2015, and mainly youtube as a platform which plays videos and not music.



Soundcloud has its colour scheme of orange, white and black in the music player to match it's same logo being orange and white. There is a big play button at the top in order to play the song, and it shows the song name with the uploader of the song and when it was uploaded. At the bottom of the page there is a more detailed look with a bar to indicate the progress of the song and there are buttons for pause/play, forward, back, shuffle and replay, there's also a volume button which when hovered gives you a volume slider to adjust the volume. The most unique feature on Soundcloud is that it displays the music in a wave format.The song cover is also displayed on the right. Another thing to mention is there is a visual aid to show how far in the song you are.



As for Youtube music the player is entirely on the bottom page with a player with the included pause/play, forward back on one side, on the other side there is volume with the same popout system and replay and shuffle. However the whole right side of the screen on youtube music has a playlist display.Another note as well with youtube music is that it matches the colour theme of your computer and due to me being in dark mode in the windows 10 settings the background here is also in darkmode, another feature is you can like and dislike songs, liked songs gets saved to a liked section where you can access all your liked songs meaning that if you discover a new track that you like you can come back to it and I'm assuming that disliked tracks remove similar tracks from your discovery list in the future. With soundcloud there is visual aid in order to show how far in the song you are.



Computational Methods

I will break down a music player into smaller problems and try to locate potential problems I will encounter within these smaller problems.

Audio:
This is the problem of making an audio player, being able to have audio play, user playback options (play/pause, forward/rewind and shuffle). I will also need to consider the audio files and the formats.

Visuals:
These are problems of how the audio player is being displayed, the placements of the music player within your screen and also the placement of the buttons, and where the playlist feature will go, I will also need to consider how power saving mode will look and how the power saving can be incorporated, also I need to think about how to visualise these features to the user either by drawing these symbols/buttons or using royalty free ones.

User Inputs:
This is the problem of how the user will interact with the music player, when a user presses a button, what is expected to happen, also something that can be considered is any keyboard shortcuts to control. Also I've only mentioned buttons at this point however maybe sliders or other input types will be more applicable. I need to find out how to detect an idle user in order to provide the power saving mode, when they are idle.

Rules:
This is the problem of when users try to break the boundaries of the music player, such instances could include a file which isn't an audio file, trying to break the search function to search for a song, by including unidentifiable characters.

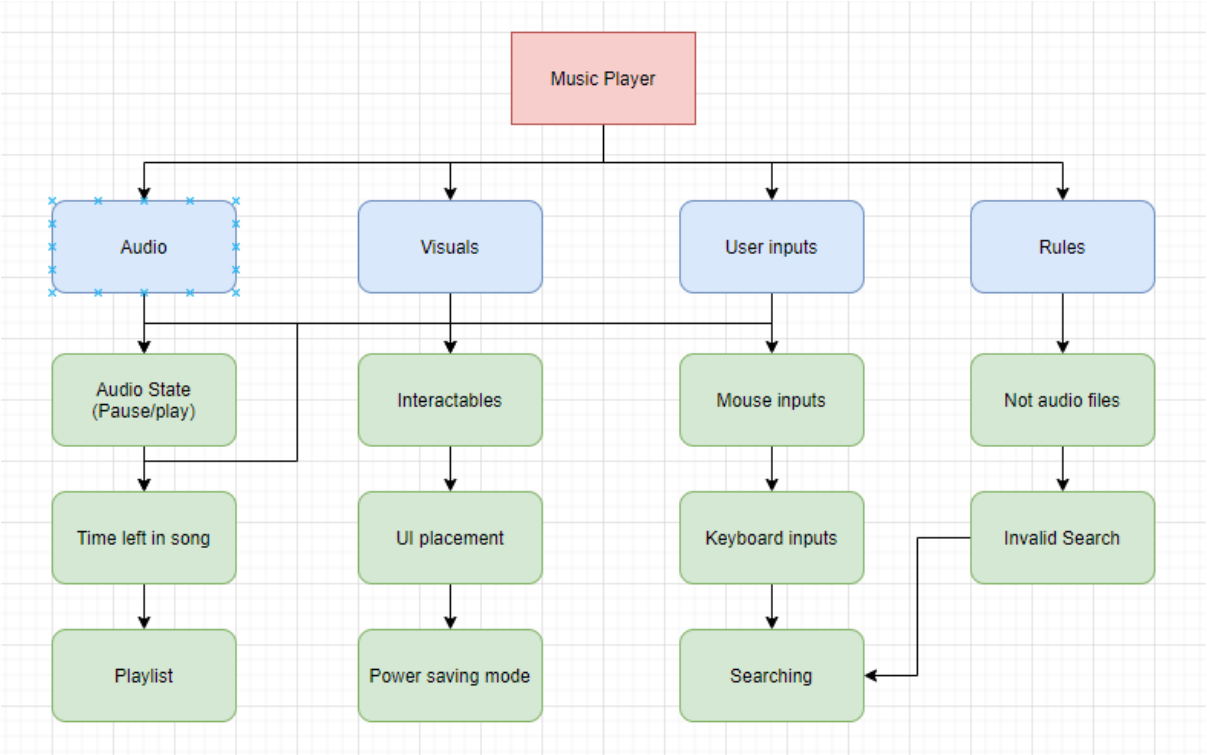
Successful Criteria

For this project to be successful I will be making a working music player and have it able to play audio, with a playlist feature, user playback functionality (including shuffle), search functionality, a power saving mode that turns on with 30 seconds of idle behaviour. I will have it to be able to run on a webpage, as I will be creating it using HTML 6., CSS 7., Javascript 8. (Node.js) 5..

Agile or Waterfall
Within this project I would like to be developing within an agile environment that gives the project lots of testing time and development, however due to how the project is structured and time restrictions I will be following a waterfall software development life cycle, as I have a clearly defined goal at the end.

Design

Structure for solution
I have broken the music player into 4 main problems that I will need to find work-arounds in order for me to be able to complete this project, I've also broken these problems into "mini-problems" that will help me when coming to design my project as these will indicate A step by step basis on what needs to be done.(File will be included)



Feature	Action to take / Possible Problem
Audio	This is where I need to take into account the whole point of the music player and think about the main specifics of audio, how I will get audio to play, if I'm going to do it through HTML, CSS and JS I will need to consider the <audio> tag within HTML 9..
Audio State	Audio state means I need the audio to know if it is in a paused or playing state, this also ties in with visuals and user inputs and what will the user pause/play and how will it look.
Time left in a song	Time left in a song means I need to be able to show how long is left within the song and what will happen when the time in the song is over. This also ties in with visuals as we need a way in order for the user to see the song is about to end.
Playlist	Playlist is how I will incorporate a playlist feature, a special list of specific songs which are able to be played one after another.

Visuals	This is where I need to take into account how everything will look and placement of elements within my page. I need to remember to make a “simple” UI that will also help with the usability and accessibility as a simple UI will give me a wide range of users who can use the product.
Interactables	I have used the word “interactables” which basically means anything that can be interacted with within the page. This is more of a visual issue than an user input issue in my opinion as the user needs to be able to see that they can interact with an element on the page, it’s useless to put user input on an element that a user doesn’t even know is interactable. So it’s about giving visual clarity in order to make the user understand that something can be interacted with. UI Placement - This is the sub-problem of how I can fit elements together on the page, I need to make it so the interactables are close enough together that the user can associate them with one another, however I need to make sure there isn’t too much going on in one area for the user meaning that an area is cluttered with information leading it to be frustrating.
Power saving mode	This is the problem of how I will incorporate a power saving mode, there isn’t really a proper way to include a power saving mode within a webpage however one way I thought of is with a “dark mode” within bringing down the brightness of the screen will make it more power saving, so after the 30 second idle period I can have it so that the page will darken up meaning for less brightness on the page, thus saving power
User Inputs	This is where I need to take into account how the user will act on the music player page, and limitations that will need to be provided in order for the user to not break anything, this slightly ties into rules however I see rules as an outside box and user inputs are what can be done within a box,user input is about the choice of choosing an action, when rules is about the

	limitations of an action. User inputs will need to be accounted for within the instructions/help document.
Mouse Inputs	Mouse inputs include how the mouse on a computer will interact with the interactable elements, either by a click, hold or a drag. I'll need to take into account what is to be done. Also mouse input ties into potentially touchscreen input too, I'll need to think about how mouse input can translate over into touch screen input.
Keyboard Inputs	This is the problem including how the keyboard on a computer interacts with the interactable elements, with keyboard shortcuts in order to give quick accessibility for users, I need to take into account any potential conflicting keyboard controls within default browsers such as arrows key can allow you to scroll up and down a page, so I will not be able to use those as shortcut keys.
Searching	Search is the sub-problem of how to include the searching, and how the searching will be done, this can be searching through a list or providing a typing area where you can search via name. I will most likely use the typing area to search as it's simple to use and universally known for usage of searching making it easier for users to understand.
Rules	This is where I will define and either give visual errors or ignore invalid inputs by the users. This is where I will set the boundaries of the music player and stop users from breaking and giving invalid inputs by taking into account these inputs first and letting them know what they are trying to do is incorrect and will not work.
Not Audio Files	This is the issue of how I will let the user know or how I will take into account a user trying to play an-audio file on the music player. I can do this by identifying the file extension of the file and checking if it is a .mp3, AAC, wav, FLAC, ALAC and DSD.

Invalid Search	Invalid search is how I will let the user know what they are searching for doesn't exist or is invalid as they inputted invalid characters. This can potentially be solved with a simple error message that just reads, "What you are searching for in invalid or doesn't exist please try again".

Proposed visuals/features

As for an example of visuals that can be used, I can use the FontAwesome library which uses a huge range of icons without needing to install or download anything. I just include the library tag and they're customizable 10..

As for a way to provide visual clarity and a bit of uniqueness to the project I wanted to include a animation, this can be done in CSS and just give that little bit of extra of "wow factor" for the user, similar to how soundcloud has the waveforms I want include a vinyl like cover of the image that spins when the song is playing. Hopefully the idea of vinyls isn't too abstract and out of date however this will be a nice reference for those who understand and a good visual indicator anyways to show if the song is in paused or playing state 11.

Above is a picture of a vinyl however I want it to be a cover image and spin when a song is playing.

Key Variables of structure

This is where I need to consider variables that I will implement in my program.

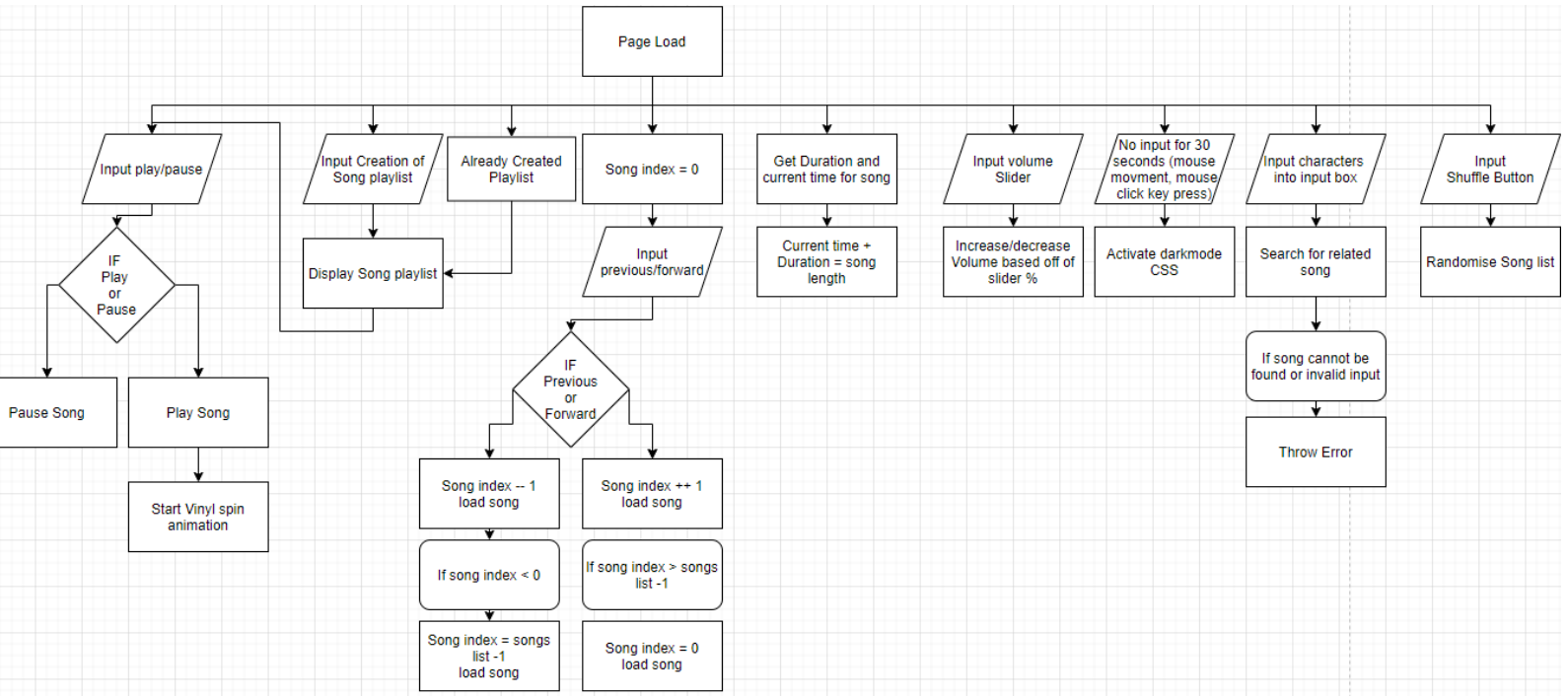
HTML interactable elements - these are the element within html which will require a javascript backend in order for them to be interactable with so I need to provide them with an appropriate ID name in order to for the javascript to find and select this element, using queryselector() 12..

HTML styled elements - these are the elements within html which will require a class or a specific ID for CSS styling, this includes the previously mentioned vinyl animation which will be done in CSS 13. This also includes creating the darkmode for when the user is idle for more than 30 seconds.

Variables used within functions - these are variables used within JS that will be included into functions one I can think of is a song index where you can identify a starting point for your song to begin from if I have a list of songs, this is one example I can think however there will be more of these variables.

Algorithms/Pseudocode

This is a flowchart for display of functionality/pseudocode for my music player. The squares show a change or an outcome, parallelograms show a user input (no input would still be an input as a user is inputting nothing), curved squares are conditions, If a condition is fulfilled then it will lead to an outcome. And diamonds are another form of condition which are either one condition or the other.



Pause and play are very similar and opposites they're like a state of a light switch where if one is active the other is inactive. In terms of logic gates this would be an XOR gate as only one can be active. The only major difference is once the play input has happened the vinyl animations need to start spinning.

Previous and Forward controls are very similar as well as they will rely on the song index and song index is a secret number that just keeps track of the song number in the array so the first song will always be 0 in the array. Moving the song into previous and forward either increments or decrements the song index. I needed to also consider what happened if the song index was 0 and if you would try and go to the previous song as I do not want the array

to underflow, you wrap back round and have the song index = to the number of songs in the list -1 and this will continually loop previous. And thus for forwarding to a song number that doesn't exist as you increment almost infinitely so if the song index is greater to the song list -1 then the song index will wrap round back to 0.

As for getting the duration and current time for the song, this is in order to find the length of the song out. If you take the current time and add it to the duration of the song you will get the current time the song will end, giving you the song length.

For the Volume input, it will be the basis of a range input and if the percentage of the range bar is full the sound will be louder and if the percentage of the range bar is low then sound will be quieter.

As for the power saving mode, when no mouse or keyboard inputs are done for 30 seconds, then the darkmode will overlay the css.

For the search function this will be done with an input and look for a match of characters within the files, if the files cannot be found or an invalid input then an error will be provided in order for the user to understand what they input was incorrected.

With playlists there are 2 states in order to be able to create a playlist or view an existing playlist both of them will lead to displaying the certain playlist selected and then leading to playing the song.

And for the shuffle button, when the shuffle button is pressed the song list will be randomised making the next song different as what would be the 2nd (1 in the array) would be replaced with the 4th song (5 in the array) would be 2nd (1 in the array).

Hardware

The hardware needed to use the music player is a mouse and keyboard to control certain features, a monitor is required to view the visuals and to see the music player. A computer with a basic component will be able to run the music player, with components such as CPU, Inbuilt graphics card etc.

Software

Software required to use the music player is an operating system preferably Windows 10 but overall I think the program will work on all versions of windows. Also a browser is required such as Microsoft Edge, Mozilla Firefox, Google Chrome as this will be integrated into a webpage.

Intended Draft Design

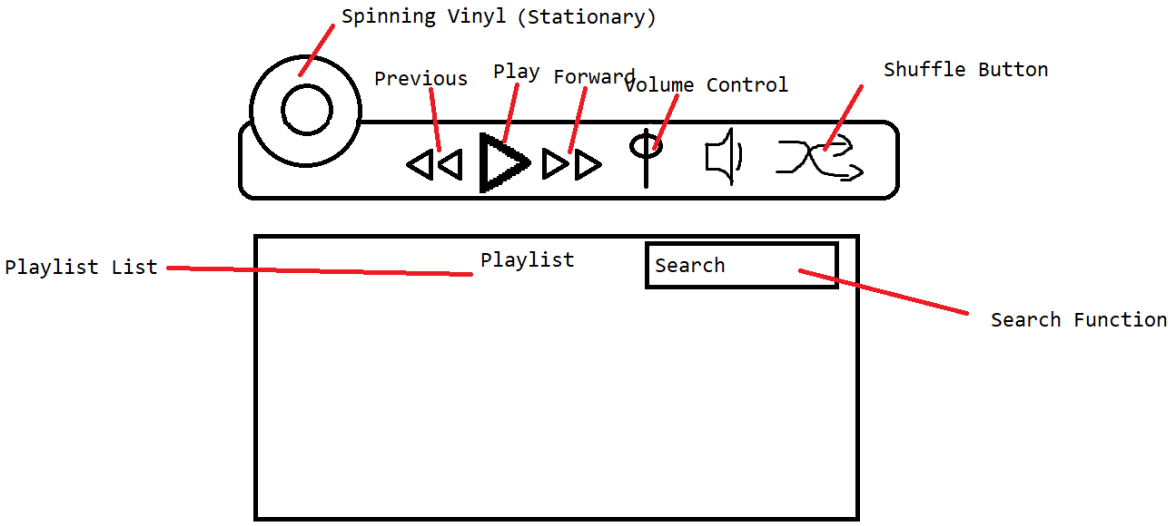
This is a draft of my Design, I will make it a page site into a light colour such as pink 14. as it's believed to be a colour relating to kindness or calming, which in my personal experience I listen to music to calm myself, such a colour will help.

I've included the controls close together so you can associate interaction along that container bar, the spinning vinyl is there too as it's interaction however with for eyes and not with user input.

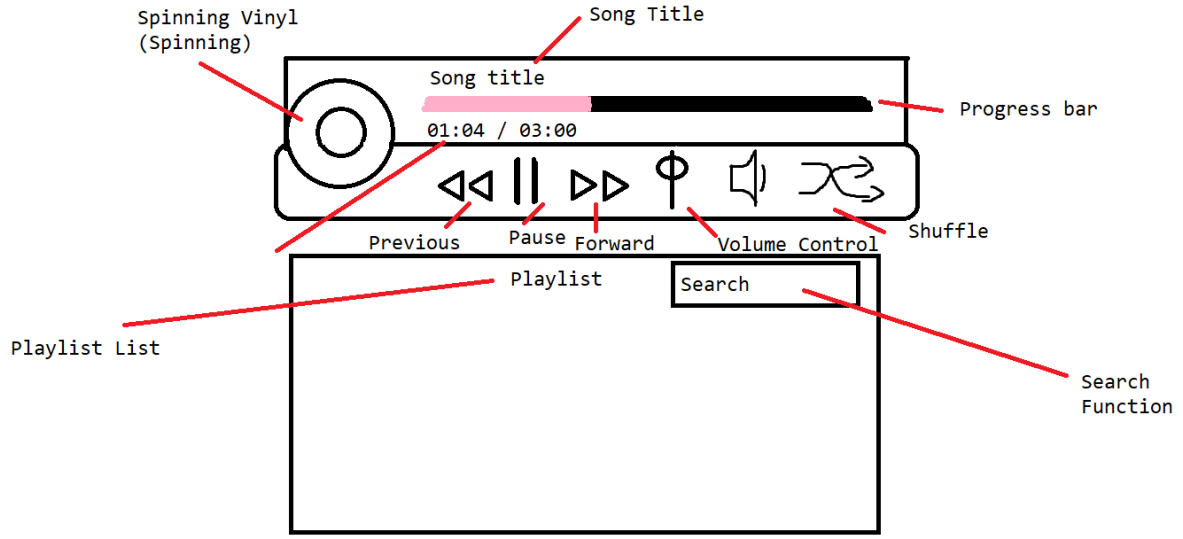
I've decided to also include the playlist list separately and left a gap inbetween with it to not make the space between controls and the playlist too cluttered.

Within the playlist box is the search function as you may want to search for a certain song in the playlist, the error message may be done through an alert or a modal.

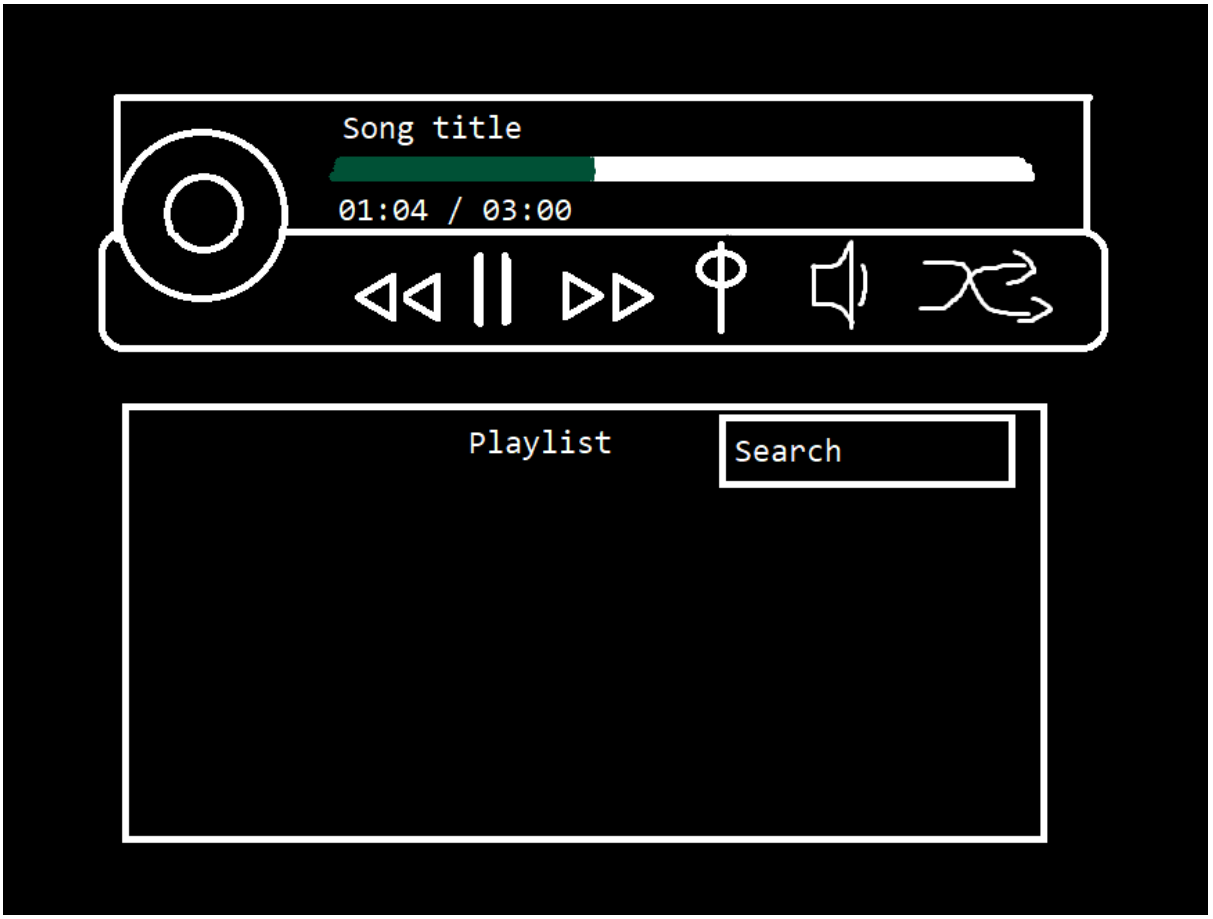
When playing I have decided to include another tab above the controls as this will contain all the song information, duration of the song, title a progress bar that's interactable with to skip to a part of the song.



Above is the paused view draft design with annotated parts.



Above is the playing view draft design with annotated parts.

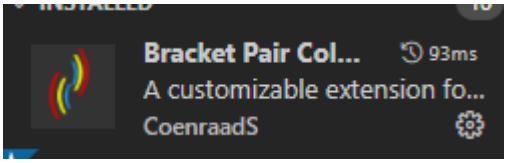


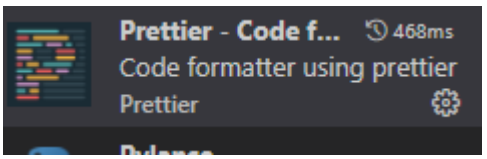
Above is an example of the darkmode, the page will almost invert colours so from everything being a bright colour it will dim down to a dark colour lowering the overall brightness of the page, saving power and this will happen after 30 seconds of idle time.

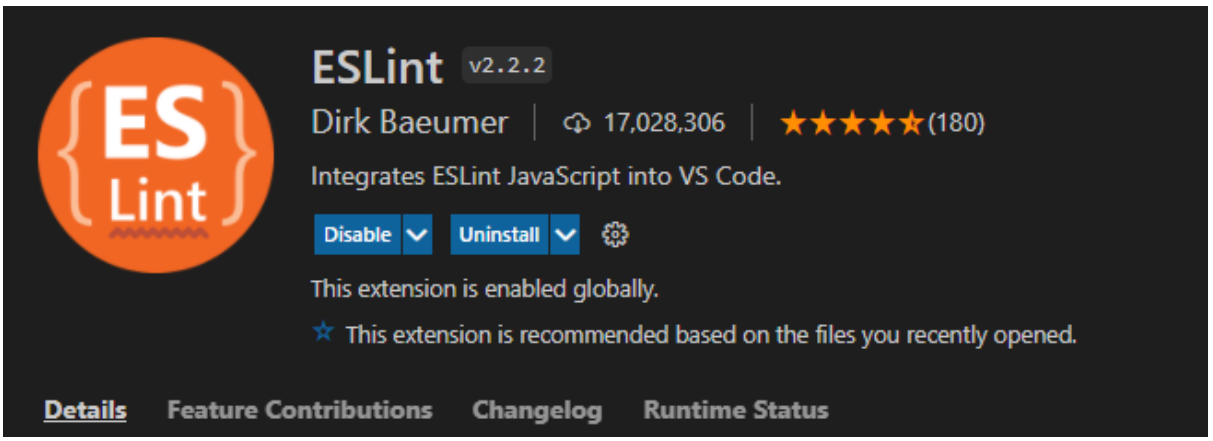
Software, Development, Testing, Implementation

Stage 0 preparation

I start off by choosing a IDE to work on I choose to do my development within VScode as it has useful features and extensions, such as Bracket Pair Colours (links 2 pairs of brackets with colour) And Prettier which is a code formatter helping me organising and formatting ESLint is used to analyse and how to have proper syntax as well.







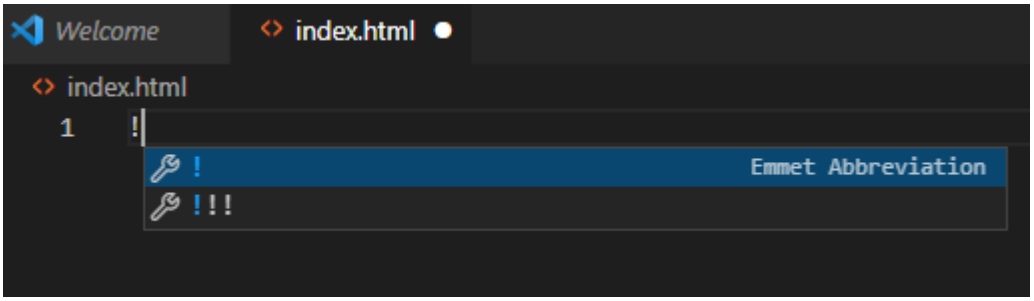
And to install it with the npm command which is used to manage javascript packages.

```
PS C:\Users\WentaoShum\Desktop\Music Player Project> npm install eslint
```

With that I will also install express.js22.. as we will need as it is a webframe work for node.js, these will be useful later when we work on backend

```
PS C:\Users\WentaoShum\Desktop\Music Player Project> npm install express --save npm install body-parser --save
[                ] / idealTree:Music Player Project: sill idealTree buildDeps
```

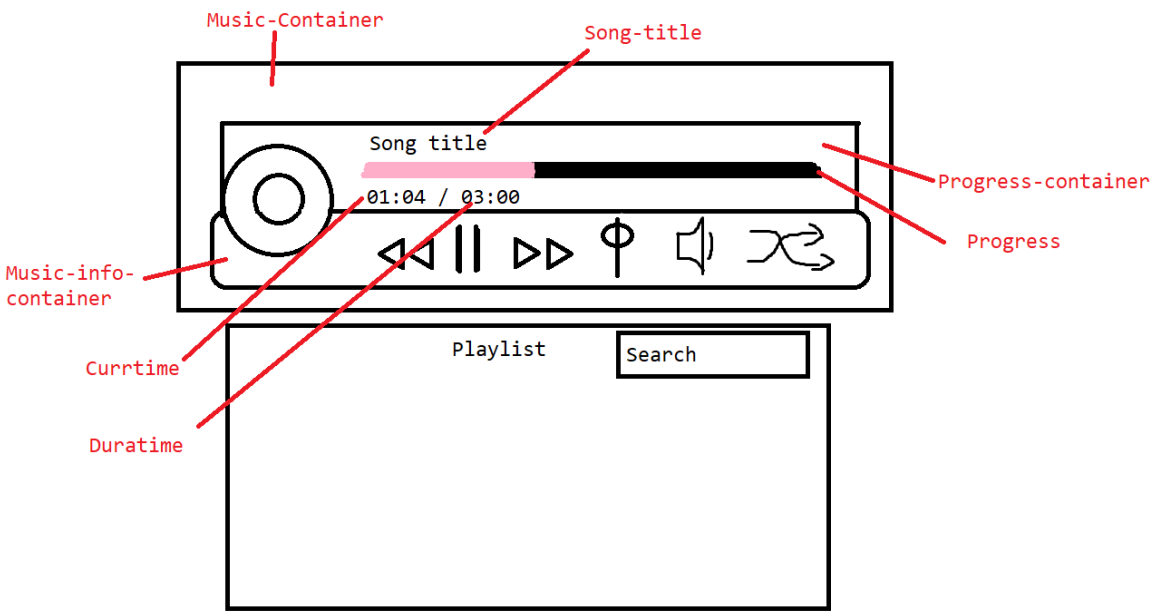
I start off as well by creating index.html (the html file for the music player) and use Emmet Abbreviation this loads a template of a HTML document for me



This is what happens after Emmet Abbreviation has processed it provides a basic template of HTML for me to use.

From this basic format I have changed the page title to “Music Player” and included CSS stylesheet and using cdnjs which lets you include libraries and as discussed before I will be looking to use FontAwesome so I link in the FontAwesome library (it is open source and free to use, so legality is not an issue) [15.]

To make it easier for myself I have decided to create another diagram of the draft I had created, however I split it into different containers in order to help me better visualise the elements and div’s within the HTML page, in preparation for the next stage.



Stage 1 Developing HTML

I start off by starting my development with HTML, as I want to add the editable elements first to give myself a base to work off of. I remember to comment throughout my code to give myself reminders of what needs to be done with certain parts and I gave certain parts ID too for later to use in JS and CSS.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./css/style.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta2/css/all.min.css" />
    <title>Music Player</title>
</head>

<body>
    <!--This is the music outside container containing all music elements-->
    <div class="music-container" id="music-container">
        <!-- This is the music information container which will display progress, song title and song times-->
        <div class="music-info-container">
            <h4 id="song-title"></h4>
            <div class="progress-container" id="progress-container">
                <div class="progress" id="progress"></div>
                <div class="title time" id="currTime"></div>
                <div class="title time" id="duraTime"></div>
            </div>
        </div>
        <!-- Audio which will play music however source will be removed and added dynamically in JS-->
        <audio id="audio" src="/music/dubstep.mp3"></audio>
        <!-- Vinyl container will be edited into CSS and JS -->
        <div class="img-container">
            <img src="images/dubstep.jpg"music-cover" id="vinyl" />
        </div>
```

I've also downloaded some royalty free music 16. and images 17. I've added the <audio> and <img> and coded the sources in at the moment however these will be removed later when we add in the JS. Here I've also added the fontawesome buttons as well along with the volume input slider.

```
    <!-- Audio which will play music however source will be removed and added dynamically in JS-->
    <audio src="music/dubstep.mp3"></audio>
    <!-- Vinyl container will be edited into CSS and JS -->
    <div class="img-container">
        <img src="images/dubstep.jpg" alt="music-cover" id="vinyl" />
    </div>
    <!-- control container-->
    <div class="control">
        <!-- Previous Button -->
        <button id="prev" class="control-btn">
            <i class="fas fa-backward"></i>
        </button>
        <!-- Play button will change with JS -->
        <button id="prev" class="control-btn control-btn-big">
            <i class="fas fa-play"></i>
        </button>
        <!-- Forward Button -->
        <button id="next" class="control-btn">
            <i class="fas fa-forward"></i>
        </button>
        <!-- Volume Slider -->
        <input type="range" id="volume-control">
        <i class="fas fa-volume-up"></i>
        <button id="random" class="control-btn">
            <i class="fa fa-random"></i>
        </button>
    </div>
    </div>
</body>
```

This is how the front end of the html page looks right now without any JS or any CSS styling.

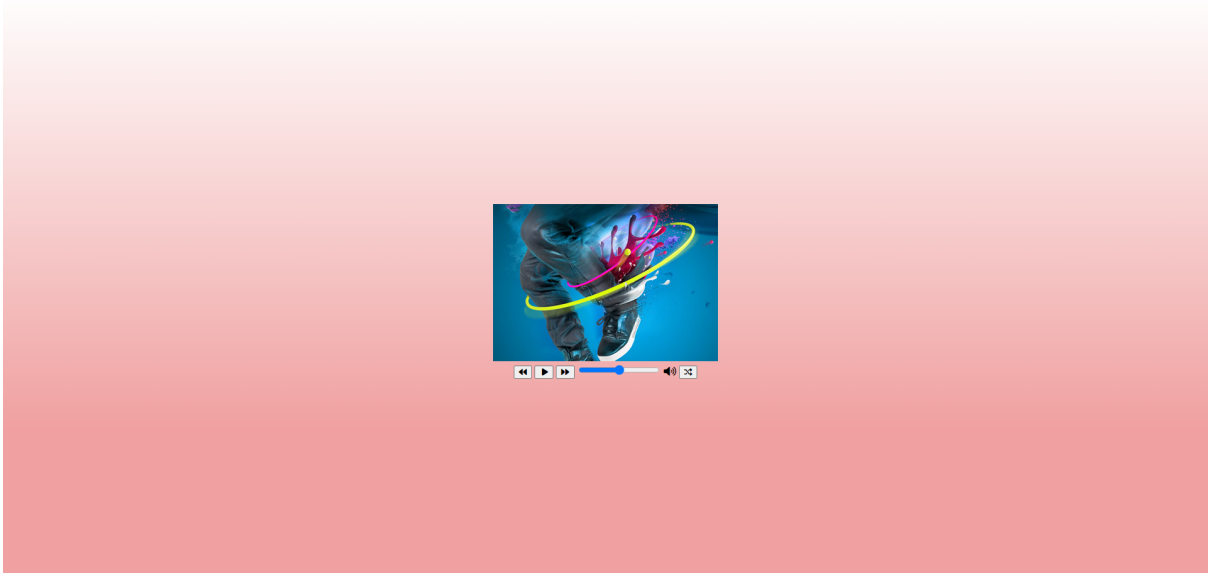
This will be the end of the first of the HTML however this doesn't mean that this isn't the end of adding or editing the HTML.

Stage 2 CSS Styling

I started off by adding styling the all the box sizing and the body of the page, this is like the base of styling as we work from the body and we can work our ways upwards to the elements on the page. Height, Margin, Font, Centering are all defined here. Also the back is a gradient this is done with linear gradient instead of just being on static colour.

```
style.css > body
/* Universal selctor adding in a box sizing of border box */

* {
    box-sizing: border-box;
}


/* body styling with linear gradient background instead of a still colour */

body {
    height: 100vh;
    margin: 0;
    font-family: Arial, Helvetica, sans-serif;
    background-image: linear-gradient(0deg, rgb(240, 160, 160)25%, rgb(255, 255, 255)100%);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    z-index: -1;
}
```

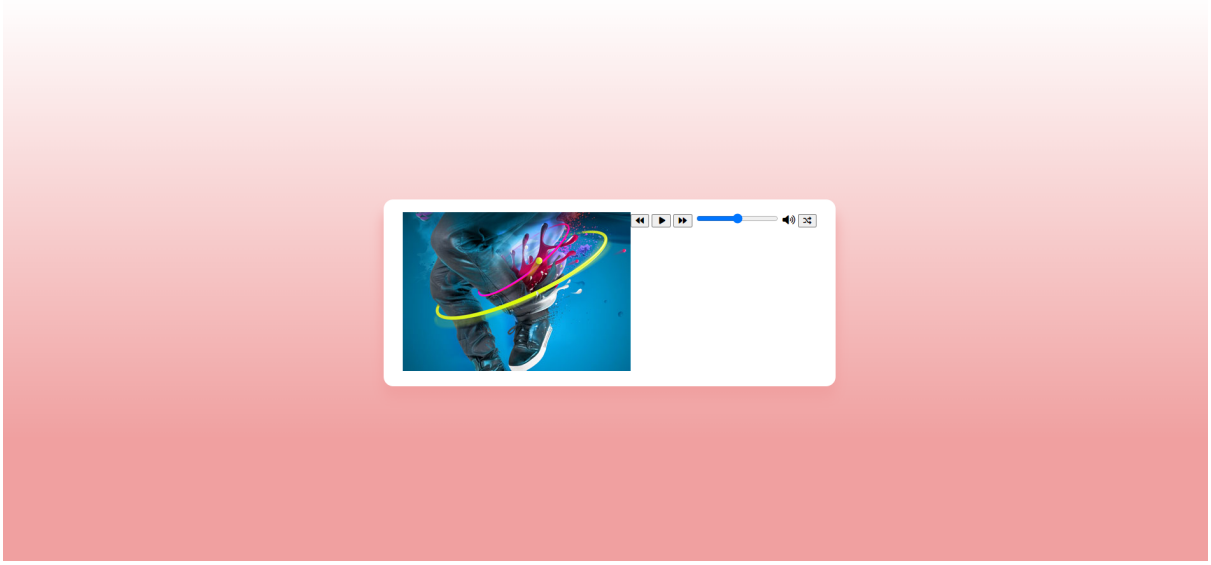
This is how the page looks after the body styling.



This is where I started to add the styling for the music container (the overall container) including a margin background colour and box shadow to hopefully make it not blend in the background as much and to make it pop.

```css
.music-container {
    display: flex;
    position: relative;
    margin: 100px;
    padding: 20px 30px;
    border-radius: 15px;
    background-color: white;
    box-shadow: 0 20px 20px 0 rgb(240, 160, 160);
    z-index: 10;
}
```

This is how the front end with the music container CSS styling looks like.



The styling for the image container is a bit complicated as we are trying to get the image to become this vinyl record, the image needs to have a border radius of 50% making it a circle and be placed correctly within the image container.

```css
/* img-container and image section */

.img-container {
    width: 110px;
    position: relative;
}

.img-container img {
    width: inherit;
    object-fit: cover;
    position: absolute;
    bottom: 0;
    left: 0;
    height: 110px;
    border-radius: 50%;
}
```

This is how the image container looks with the image inside currently I have the “dubstep.jpg” however this will be able to be changed when I add in the JS side.

Image-container after is a way to add in a white circle to make the cover image look like more of a vinyl, usually after is used for content and needs to have “content” however we

can bypass this as just leave it empty.

```css
/* Using the after pseudo selector (requires content property however we are just
using it as an element) this is to creating the empty space in the vinyl */
.img-container::after {
    content: '';
    height: 20px;
    width: 20px;
    left: 50%;
    bottom: 100%;
    border-radius: 50%;
    background-color: white;
    transform: translate(-50%, 50%);
    position: absolute;
}
```

This is how the vinyl looks with the white dot added to circle cover image, make it more vinyl like.

I've added the animation here at the bottom of the image where it will rotate for 3 seconds and forever until tolled stopped and it's starting state will be paused.

```css
.img-container img {
    width: inherit;
    object-fit: cover;
    position: absolute;
    bottom: 0;
    left: 0;
    height: 110px;
    border-radius: 50%;

    /* Added in the CSS anmation */
    animation:rotate 3s linear infinite;
    animation-play-state:paused;
}
```

Music-container.play will be used in JS later and will apply the and change the animation state of the image container. Also here is where the animation is set just, from being static to rotating 360 degrees like a spinning vinyl.

```css
/* Used to make the animation class start and stop this class will be applied in JS*/
.music-container.play .img-container img{
    animation-play-state: running;
}

/* This is how the vinyl record animation is made */
@keyframes rotate {
    from{
        transform: rotate(0deg);
    }
    to {
        transform: rotate(360deg);
    }
}
```

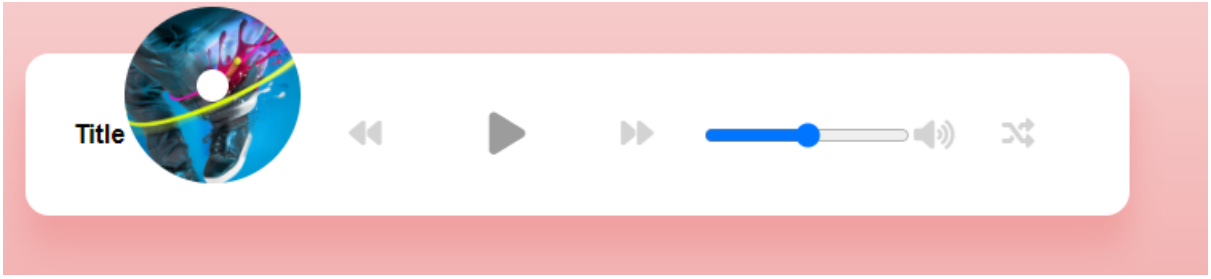
The Controls sections contain all the styling for the buttons and favicons, making the icons stand out by coloring them a light gray colour, volume up is a special case and pressing the volume up icon wouldn't do anything so I've taken the cursor pointer off of that element. The elements with cursor pointer are there to show that they are able to be interacted with.

```css
/* Controls Section */


/* Controls container */

.control {
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 1;
}


/* Control buttons styling */

.control-btn,
.fa-volume-up {
    background-color: white;
    border: 0;
    color: lightgray;
    font-size: 20px;
    cursor: pointer;
    padding: 10px;
    margin: 0 20px;
}


/* Volume up doesn't want to be clicked on so no cursor hover */

.fa-volume-up {
    padding: 0%;
    margin: 0%;
    cursor: default;
}
```

For the Pause/Play button it needs be slightly bigger and to make it pop and stand out a bit more I've used a darker gray. Also I've removed the button outlines so it looks like you're pressing the symbols.

```css
/* Makes the play/pause button in the middle slightly bigger
(meaning more user focus) */

.control,
.control-btn-big {
    color: rgb(156, 156, 156);
    font-size: 30px;
}


/* Removes the outline from the buttons */

.control-btn:focus {
    outline: 0;
}
```

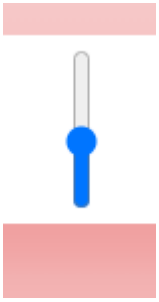
Here is how the buttons look I've also added a title to the just check how that looks, this will be more relevant for the next section in regards to music-info-container however the buttons are looking.



I said the buttons were looking okay however I did realise that I've forgotten about the volume slider. It makes the container longer than I'd like. Also in my draft diagram I've included it being vertical. I've had to rotate it to 270deg as opposed to 90deg due to orientation of the slider, 90deg would make 0% at the top and 270deg makes 100% at the top.

```css
/* Volume controls rotate 270 so
100 is at the top and 0 is at the bottom */
#volume-control {
    transform: rotate(270deg);
    width: 80px;
    cursor: pointer;
}
```

The volume slider is now correctly orientated.



The music info container I want to popup when music starts to play and for it to get hidden away when music is not playing. I've done this by adding the translateY transformations that will allow it to pop up out from under the controls. This isn't visible yet as it there is not music playing however I'll be able to see when the JS is added in.

```css
/*  Music Info container section */


/* Music information container has an animation
for popping up when music is playing however is hidden when music isn't playing */

.music-info-container {
    position: absolute;
    top: 0;
    left: 20px;
    width: 90%;
    padding: 10px 10px 10px 150px;
    opacity: 0;
    transform: translateY(0%);
    transition: transform 0.3s ease-in, opacity 0.3s ease-in;
    z-index: 0;
    background-color: rgba(255, 255, 255, 0.5);
    border-radius: 15px 15px 0 0;
}


/* When the song is playing the info container will appear above */

.music-container.play .music-info-container {
    opacity: 1;
    transform: translateY(-100%);
}


/* Song Title text no margin */

.music-info h4 {
    margin: 0;
}
```

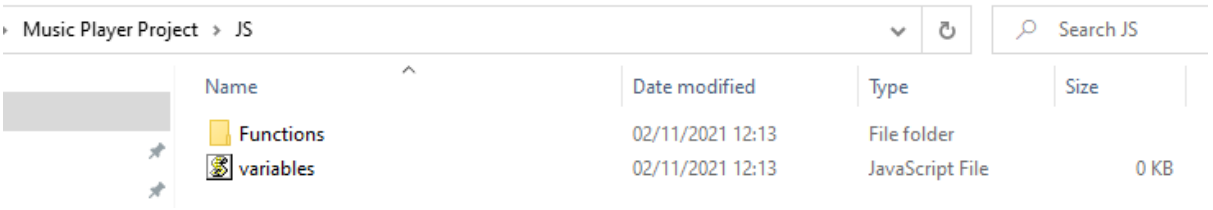
The progress container is where the progress bar will be placed.I've made the whole container have a cursor pointer to show that it can be interacted with. And the progress bar will be pink matching the rest of the website. The time text as well has been styled here as well where it will display how long is left within a song.

```css
/* Progress section for the progress bar */


/* Progress container styling */

.progress-container {
    background: white;
    border-radius: 5px;
    margin: 10px 0;
    height: 10px;
    width: 100%;
    cursor: pointer;
}

/* This is the styling for the progress bar */

.progress {
    background-color: pink;
    border-radius: 5px;
    height: 100%;
    width: 0%;
    transition: width 0.1s linear;
}


/* Styles for the time text */

.time {
    width: 50%;
    display: inline;
    position: relative;
}
```

Additional worth mentioning this is for CSS for now however this will be revisited in for the darkmode option, however I would like to get the function for it to work first before implementing the CSS in.

Stage 3 Javascript Variables and Change of Plans

First of all I'll show my file structure I'm doing to include all my JS into sections in regards to variables I'll have a seperate file for this import variables for functions that are needed,the variables.js is more like a big library of variables and then functions can import them when needed. I've done this in order to show a clear file structure and this will make it easier for debugging and locating specific issues.



For the HTML to use the different JS files they need to be included within the body and I need to specify the type for the variables, as it is a module.

```
    </div>
  </div>
  <script type="module" src="JS/variables.js"></script>
  <script src="JS/Functions/playPause.js"></script>
/body>
```

This is the variables I've included so far, I've taken all the ID's from the html and created the new variables in JS and exported them for later use in the other functions.

```
const audio = document.getElementById("audio");
const progress = document.getElementById("progress");
const progressContainer = document.getElementById("progress-container");
const title = document.getElementById("song-title");
const vinyl = document.getElementById("vinyl");
const currTime = document.querySelector("#currTime")
const duraTime = document.querySelector("#duraTime")

const musicContainer = document.getElementById("music-container");
const playButton = document.getElementById("play");
const prevButton = document.getElementById("prev");
const nextButton = document.getElementById("next");

//These are the title names of the songs
const songs = ["dubstep", "betterdays", "sunny"];

//This is the starting point within the song array
let songIndex = 0;

export {
    musicContainer,
    playButton,
    prevButton,
    nextButton,
    audio,
    progress,
    progressContainer,
    title,
    vinyl,
    currTime,
    duraTime,
    songs,
    songIndex,
};
```

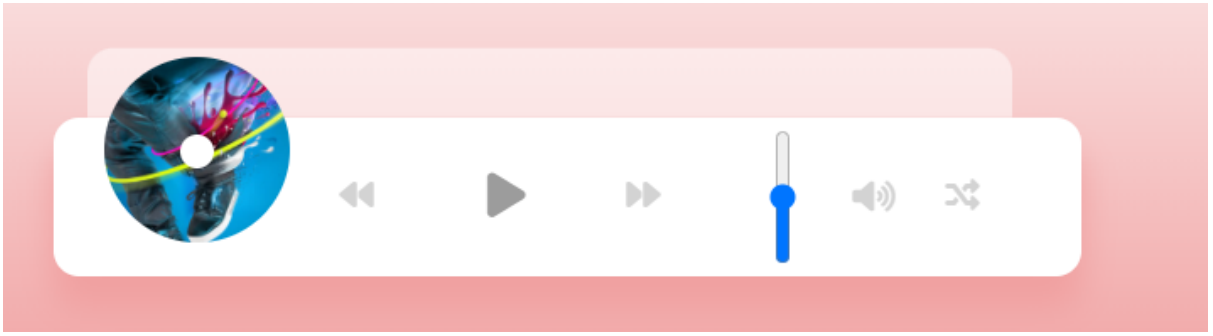
I started working on importing the variables that I needed to include for the loadsong function which will be included into playPause.js, making the loading of songs dynamic.

public > JS > Functions > JS playPause.js > loadSong

```
1   import { audio, songs, title, songIndex, playButton, vinyl } from "../variables.js";
2
3   // Loads the song DOM information
4   loadSong(songs[songIndex])
5
6   function loadSong(song) {
7       title.innerText = song
8       audio.src = `./music/${song}.mp3`;
9       vinyl.src = `./images/${song}.jpg`;
10  }
```

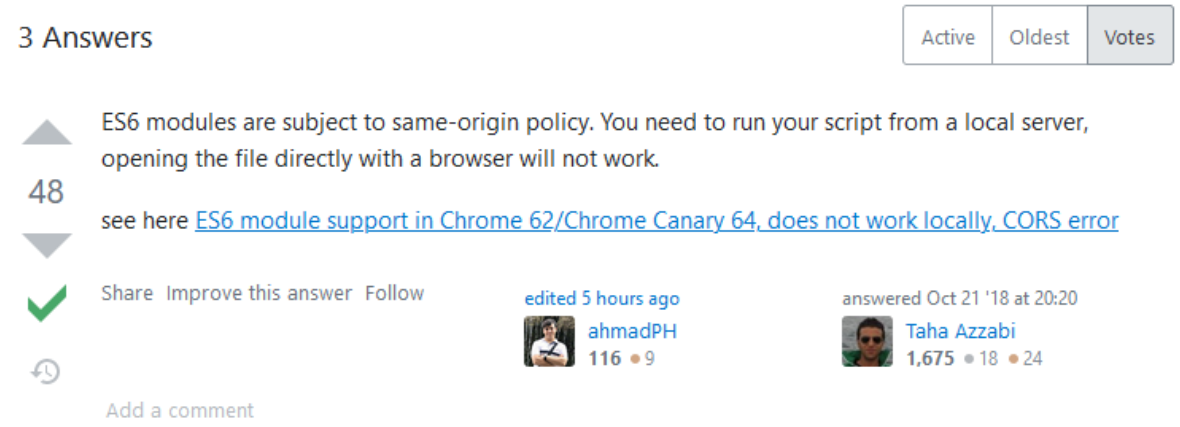
However when I tested my code by adding the “play” class to the music container it should display the song name within the title and include the music source, it doesn’t though and was empty

```
<div class="music-info-container play">
    <h4 id="song-title"></h4>
    <div class="progress" id="progress">
        <div class="title time" id="currTime"></div>
        <div class="title time" id="duraTime"></div>
    </div>
</div>
```



I came across the error below, I looked it up and it means that you cannot use the file directory and you need to run the script from a local server.

Access to script at 'file:///C:/Users/WentaoShum/Desktop/Music%20Player%20Project/JS/variables.js' from origin 'null' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: http, data, chrome, chrome-extension, chrome-untrusted, https.



19.

For a change of plans I will create my local server now instead of later. I was hoping to create all the variables and functions first and then create the local server to run it off of but doing it this way means I can test my changes whilst I code the functions.

Stage 3.1 Local server setup and Bug fixing

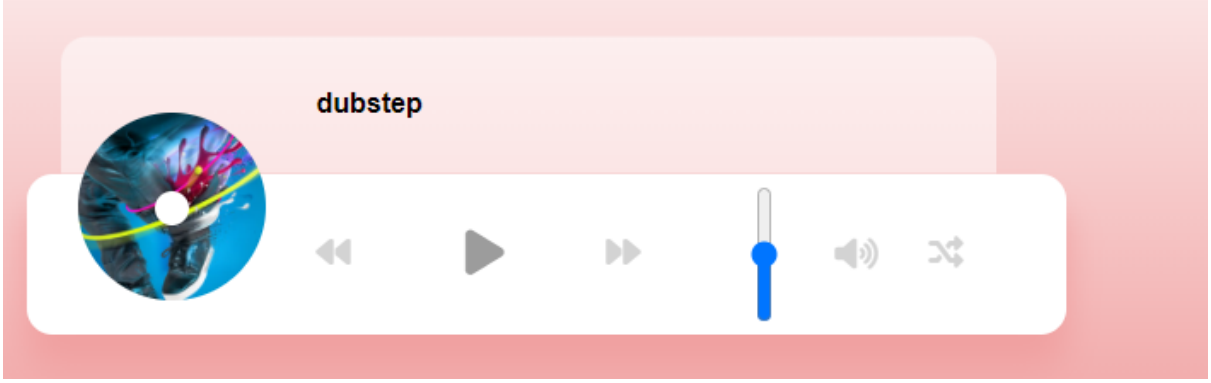
With the events of the previous stage, I've had to adapt and setup the local server now. The setup uses express.js npm install that we installed earlier, the folder structure as well JS, CSS, Images and music need to be within a "public" directory folder. res.sendFile sends the file to the directory. Var server = app.listen runs the server address at the port of 8081, meaning to access the site it's at "localhost:8081".

```
JS index.js > server
1   //requires the express.js npm install
2   var express = require('express');
3   var app = express();
4
5   //Used to serve static assests (using CSS Javascript Images with a public directory)
6   app.use(express.static("public"));
7   app.get("/", function(req, res) {
8       //Sends the file to the user domain
9       res.sendFile(__dirname + "/index.html");
10  })
11
12  var server = app.listen(8081, function() {
13      var host = server.address().address
14      var port = server.address().port
15      console.log("At http://%s:%s", host, port)
16  })
```

In order to run the server within the terminal (cmd) you need to go to the directory of index.js and run the command "node index.js" and this runs the server.

```
Node.js v17.0.1
PS C:\Users\WentaoShum\Desktop\Music Player Project> node index.js
At http://:::8081
```

And now the variables imported as expected when testing the code onto the local server directory.



There is another bug as well, within my loadsongs function the "audio.src" seems to be null

```
▶Uncaught TypeError: Cannot set properties of null (setting 'src')
    at loadSong (playPause.js:9)
    at playPause.js:4
```

I've added this console.log to check what the audio variable is.

```
import { audio, songs, title, songIndex, playButton, vinyl } from "../variables.js";

// Loads the song DOM information
loadSong(songs[songIndex])

function loadSong(song) {
    title.innerText = song;
    console.log(audio);
    audio.src = `./music/${song}.mp3`;
    vinyl.src = `./images/${song}.jpg`;
}
```

Within the console it shows that "audio" is null.

```
null
▶Uncaught TypeError: Cannot set properties of null (setting 'src')
    at loadSong (playPause.js:9)
    at playPause.js:4
>
```

Within variables it is defined by the id of "audio" from the HTML.

```
const audio = document.getElementById("audio");
```

And the issue was found as I didn't apply the "audio" id for the audio source.

```
<audio src="music/dubstep.mp3"></audio>
```

Now including the id,
Within the console it shows the element.

```
<audio id="audio" src="./music/dubstep.mp3"></audio>
```

With this I can remove the src hard coded within the HTML as now the JS loads it dynamically.

```
<audio id="audio"></audio>
<!-- Vinyl container will be edited into CSS and
<div class="img-container">
    <img alt="music-cover" id="vinyl" />
```

Stage 3.2 Back to Functions

Play/Pause

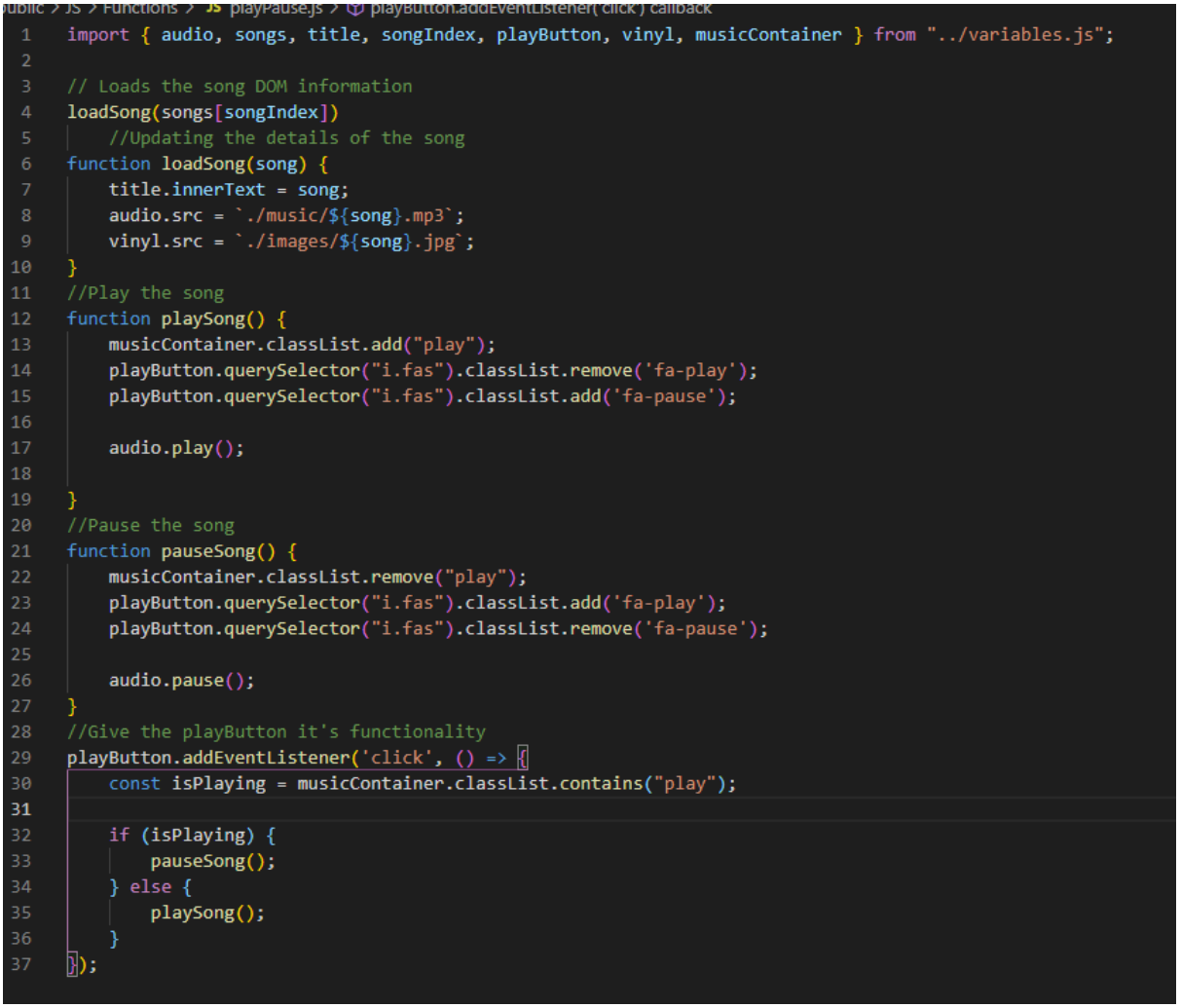
With the local server setup and the previous bug fixed I will carry on working on the playPause function file.

Continuing with the playPause function file. I've imported all variables that I would be using from the previous variable file loadSong, loads a song dynamically using the index number since it's 0 it will load the first song in the array shown previously in variables.

The playSong function adds the "play" class to the music container and removes the FontAwesome play button and adds in the FontAwesome pause button, it also plays the audio.

The pauseSong function does the opposite of the playSong function removes "play" class and adds FontAwesome play button and removes FontAwesome pause button, it also pauses the audio.

In order to get this button to exist I need to give it functionality, so I added an event listener when clicking on the playButton, it does a check if it's playing and if it is playing, pause the song and if it is playing the song, pause the song. And the result is this will be all the CSS animations too 18..



In order to be ready for the next function file I needed create exports of the previous functions as they will be used within the other functions (playSong,loadSong,pauseSong)

```
export function loadSong(song) {
    title.innerText = song;
    audio.src = `./music/${song}.mp3`;
    vinyl.src = `./images/${song}.jpg`;
}
//Play the song
export function playSong() {
    musicContainer.classList.add("play");
    playButton.querySelector("i.fas").classList.remove('fa-play');
    playButton.querySelector("i.fas").classList.add('fa-pause');

    audio.play();

}
//Pause the song
export function pauseSong() {
    musicContainer.classList.remove("play");
    playButton.querySelector("i.fas").classList.add('fa-play');
    playButton.querySelector("i.fas").classList.remove('fa-pause');

    audio.pause();
}
```

NextPrev Song

Below is the first draft of my NextPrev.js file which will control the next and previous buttons in theory it works, as it contains imported functions and variables from the previous PlayPause.js and variables.js, I've even included the exceptions of if the array would underflow or overflow into both nextSong function and prevSong function, it will load the song and play it.

```
> JS > Functions > JS NextPrev.js > prevSong
import { nextButton, prevButton, songIndex, songs } from "../variables.js";
import { loadSong, playSong } from "./playPause.js";

//Previous Song Function
function prevSong() {
    songIndex--;
    //Expection of if you are trying to under 0
    if (songIndex < 0) {
        songIndex = song.length - 1;
    }
    loadSong(songs[songIndex]);
    playSong()
}

//Forward Song Function
function nextSong() {
    songIndex++;
    //Expection of if you are trying to go over the max number
    if (songIndex > songs.length - 1) {
        songIndex = 0;
    };
    loadSong(songs[songIndex]);
    playSong()
}
prevButton.addEventListener('click', prevSong);
nextButton.addEventListener('click', nextSong);
```

Unfortunately this error came up in the console, which extremely confusing as I didn't set my songIndex to be a const value.

```
▸Uncaught TypeError: Assignment to constant variable.
    at HTMLButtonElement.prevSong (NextPrev.js:6)
```

songIndex is not a const value.

```
//This is the starting point within the song array
let songIndex = 0;
```

Only through reading I've come to find that when exporting variables they're only exported as "read only"'s which in all honestly makes sense in order to encapsulate and protect the variable. 20.

A way around this would be in variables.js I create a setter method and call on this method within the nextSong or prevSong function, I will also need to do this for the exceptions as well.

This is the setter method in variables.js

```
export function incrSongIndex() {
    songIndex++
    //Expection of if you are trying to under 0
    if (songIndex > songs.length - 1) {
        songIndex = 0;
    };
}

export function decrSongIndex() {
    songIndex--
    //Expection of if you are trying to go over the max number
    if (songIndex < 0) {
        songIndex = songs.length - 1;
    }
}
```

I've included the variables with the appropriate exceptions, so now when I import the functions and include the loadSong(songs[songIndex]) and playSong(). This is now fixed 21..

```
import { nextButton, prevButton, songIndex, songs, decrSongIndex, incrSongIndex } from "../variables.js";
import { loadSong, playSong } from "./playPause.js";

//Previous Song Function
function prevSong() {
    decrSongIndex(); {
        loadSong(songs[songIndex]);
        playSong()
    }
}

//Forward Song Function
function nextSong() {
    incrSongIndex(); {
        loadSong(songs[songIndex]);
        playSong()
    }
}
prevButton.addEventListener('click', prevSong);
nextButton.addEventListener('click', nextSong);
```

Progress Bar

ProgressBar was what I wanted to do next, two main things about the progress before anything else is I need to make it so the progress bar updates and allows you to click on the progress bar.

Of course first I have to include the file within the HTML

```
<script type="module" src="JS/Functions/progressBar.js"></script>
```

This is done with the functions “updateProgress” and “setProgress” , within updateProgress it takes in an event object and takes in the currentTIme and the duration, then you need to take it as a percentage so you take the currentTime divide it by duration and multiple by 100 to get the percentage, this is for the progress bar to fill up as the song plays.

setProgress, we get the width of the bar, we also get where we click on it on the x axis and then we get the duration with const duration and the element audio.duration. From here we set it from where we click from divided by the width multiplied by the duration of the song.

public > JS > Functions > JS progressBar.js > setProgress

```
import { progress, audio, duraTime, currTime, progressContainer } from "../variables.js";
import { nextSong } from "./NextPrev.js";

function updateProgress(e) {
    const { duration, currentTime } = e.srcElement;
    const progressPercent = (currentTime / duration) * 100;
    progress.style.width = `${progressPercent}%`;
};

//Set progress bar
function setProgress(e) {
    const width = this.clientWidth;
    const clickX = e.offsetX;
    const duration = audio.duration;
    audio.currentTime = (clickX / width * duration);
}
```

Also these are the eventlisteners that are used in order to update the time progress and where you click on the progress container.

```
//Time/Song Update
audio.addEventListener('timeupdate', updateProgress);
//Click on Progress Bar
progressContainer.addEventListener('click', setProgress);
```

Next is for me to get the duration and current time of the song, to show the user how long left and the full duration time of the song.

Firstly I have to define the minutes by taking the currentTime getting the largest integer and then dividing it by 60 for the seconds this defines the minutes,

In order to define the seconds the “get_second” function will only run if the time is less than 60, then it will add up and then it takes into account the minutes from before and calculates what is left.
Then it is changed within the html to display as minutes and seconds.

```
//get duration & currentTime for time of of song
function durationTime(e) {
    const { duration, currentTime } = e.srcElement;
    var sec;
    var sec_dura;

    // define minutes currentTime
    let min = (currentTime == null) ? 0 :
        Math.floor(currentTime / 60)
    min = min < 10 ? '0' + min : min;

    //define seconds currentTime
    function get_second(x) {
        if (Math.floor(x) >= 60) {
            for (var i = 1; i <= 60; i++) {
                if (Math.floor(x) >= (60 * i) && Math.floor(x) < (60 * (i + 1))) {
                    sec = Math.floor(x) - (60 * i);
                    sec = sec < 10 ? '0' + sec : sec;
                }
            }
        } else {
            sec = Math.floor(x);
            sec = sec < 10 ? '0' + sec : sec;
        }
    }
    //define seconds duration
    get_second(currentTime, sec);

    //change currTime for HTML
    currTime.innerHTML = min + ':' + sec;
```

Now is time to define the duration left within the song,firstly we calculate the minutes for the duration and then calculate the seconds within from what is left out of the seconds from the minutes. Once done we update the duraTime within the HTML with the minutes and seconds.

```
    //define minutes duration
    let min_dura = (isNaN(duration) === true) ? '0' :
        Math.floor(duration / 60);
    min_dura = min_dura < 10 ? '0' + min_dura : min_dura;

    function get_sec_d(x) {
        if (Math.floor(x) >= 60) {
            for (var i = 1; i <= 60; i++) {
                if (Math.floor(x) >= (60 * i) && Math.floor(x) < (60 * (i + 1))) {
                    sec_dura = Math.floor(x) - (60 * i);
                    sec_dura = sec_dura < 10 ? '0' + sec_dura : sec_dura;
                }
            }
        } else {
            sec_dura = (isNaN(duration) === true) ? '0' :
                Math.floor(x);
            sec_dura = sec_dura < 10 ? '0' + sec_dura : sec_dura;
        }
    }
    //define seconds for duration

    get_sec_d(duration);

    //change duraTime for HTML
    duraTime.innerHTML = min_dura + ':' + sec_dura;
}
```

We then add the eventlistener that will update the time of song for a new song. I also realised as well that the user will probably want the song to change once it has ended. This is done here with the eventlistener which uses the nextSong function which means I also had to export the nextSong function. This can be seen here 23.

```
//Forward Song Function
export function nextSong() {
    incrSongIndex(); {
        loadSong(songs[songIndex]);
        playSong()
    }
}
```

```
//Song ends play next song
audio.addEventListener('ended', nextSong);
//Time of song
audio.addEventListener('timeupdate', durationTime);
```

Volume Controls

Next I'll move onto volume controls. As always I will include the file into my HTML.

```
<script type="module" src="JS/Functions/volumeControl.js"></script>
```

Volume controls are fairly simple, I've set a default volume level which will not blast music out on the first load on the page. Then created a volume variable which uses the value from the range slider within volume-control id and since volume is based off of decimals I need to set it and divide the value by 100 and this is the outcome [24.]

public > JS > Functions > JS volumeControl.js > volume.addEventListener("change") callback

```
1  //Set a default volume which is considerate on first load
2  audio.volume = 0.5
3
4  //select the volume-control ID and then allow it to change based on its value out of 100 /100
5  let volume = document.querySelector("#volume-control")
6  volume.addEventListener("change", function(e) {
7      audio.volume = e.currentTarget.value / 100;
8  })
```

Dark Mode

Like always I'll link the JS to the HTML.

```
<script type="module" src="JS/Functions/darkMode.js"></script>
```

I started off by creating a function for the darkModeTimeout that will take into consideration all the window actions, with the mouse and keyboard, on mouse movement, mouse press, touchscreen presses as well for potential touchscreen integration and key press.

```
import {progressContainer,musicContainer} from "../variables.js";

function darkModeTimeout(){
    var timeout;
    window.onload = resetTimer;
    window.onmousemove = resetTimer; // catches mouse movement
    window.onmousedown = resetTimer;  // catches touchscreen presses as well
    window.ontouchstart = resetTimer; // catches touchscreen swipes as well
    window.onclick = resetTimer;      // catches touchpad clicks as well
    window.onkeydown = resetTimer;   //catch key press
    window.addEventListener("scroll", resetTimer, true); // catches scrolling
```

Within the darkModeTimeout I have the darkMode function this basically grabs all the html elements with the class ".control-btn and ,.fa-volume-up", I also iinclude the document body , progressContainer and musicContainer defined before and provide them with a darkMode class (body has darkBody). I use a FOR loop in order to go through all the darkModeButtons and assign them the darkmode (I also made a mistake here).

```
    //darkMode function
    function darkMode() {
        var darkModeButtons = document.querySelectorAll(".control-btn,.fa-volume-up")
        document.body.classList.toggle("darkBody");
        progressContainer.classList.toggle("darkMode");
        musicContainer.classList.toggle("darkMode");
        for (let i = 0; i < darkModeButtons.length; i++) {
            darkModeButtons[i].classList.toggle("darkmode");
        };
    }
```

Here is the function outside darkMode where the timer will reset and clear the timeout when one of the windows actions is completed the timeout timer will reset and the timeout is set to 30 seconds (it's in milliseconds so I just multiply 30 by 1000)

```
function resetTimer() {
    clearTimeout(timeout)
    timeout = setTimeout(darkMode, 30 * 1000); //time is set in milliseconds
}
```

And then I run the function here.

```
darkModeTimeout();
```

Back to CSS I started by trying to just apply and inverted filter to the page and see how it turns out.

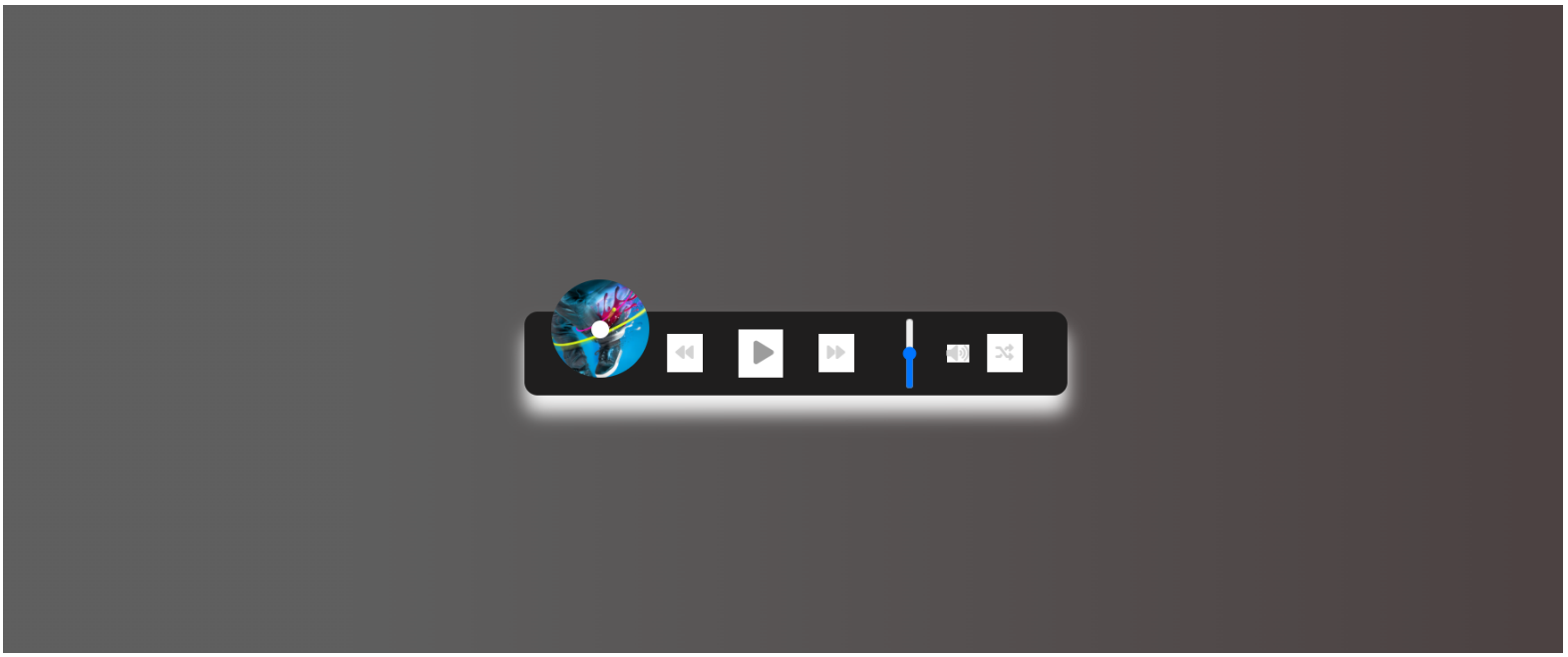
```css
.darkMode {
    -webkit-filter: invert(100%);
    filter: invert(100%);
}

body.darkBody {
    -webkit-filter: invert(100%);
    filter: invert(100%);
}

.music-container.darkMode {
    -webkit-filter: invert(100%);
    filter: invert(100%);
}

.progress-container.darkMode {
    -webkit-filter: invert(100%);
    filter: invert(100%);
}
```

And it slightly worked however, it inverted the vinyl image, the slider and the body didn't change(This is due to the body not being able to receive filters), so instead of doing it this method I just rewrite some of the CSS.

This is where I've rewritten the CSS and overwritten some of the colours of the elements, by applying a new background gradient to the body and having darkmode just provide a dark background and to help the other containers pop a bit I've added white box shadow.

```css
.darkMode {
    background-color: rgb(31, 30, 30);
    color: white;
}

body.darkBody {
    background-image: linear-gradient( 0deg, rgba(247, 247, 247, 0.384) 23.8%, rgba(252, 221, 221, 0.301) 92%);
    background-image: -webkit-linear-gradient( 0deg, rgba(247, 247, 247, 0.384) 23.8%, rgba(252, 221, 221, 0.301) 92%);
    background-color: black;
}

.music-container.darkMode {
    box-shadow: 0 20px 20px 0 whitesmoke;
}

.progress-container.darkMode {
    box-shadow: 0 20px 20px 0 whitesmoke;
}
```

This is the outcome, however the control.btn css still havent changed.



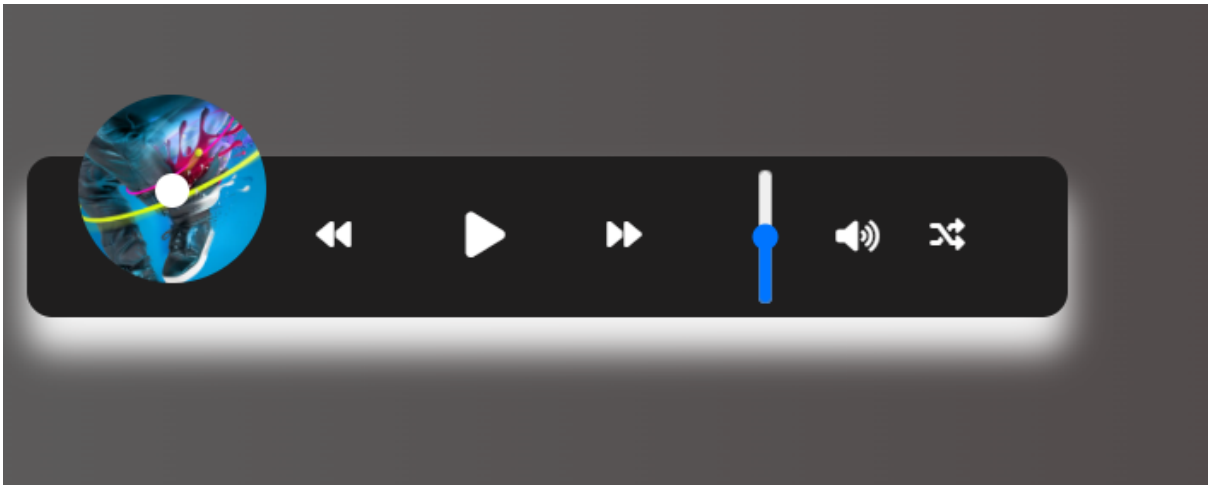
Doing some debugging, in chrome webdev tools I can see the that control-btn class is receiving darkMode. So it has nothing to do with the CSS and HTML, meaning I have to check the JS

```
▶ <button id="prev" class="control-btn darkMode">…</button>
  <!-- Play button will change with JS -->
```

And within the JS I end up discovering that the classList.toggle was “darkmode” and not “darkMode” this is why nothing was happening to the CSS and the class it was being applied to was “darkMode” yet “darkmode”.

```
for (let i = 0; i < darkModeButtons.length; i++) {
    darkModeButtons[i].classList.toggle("darkMode");
};
```

When changed this is now fixed. And here is the completed timeout (I can't record for 30 seconds however I will just show how it switches to darkmode) 25.



Shuffle Button

With any of the JS functions I'll start by adding in the function file to the HTML.

```
<script type="module" src="JS/Functions/shuffle.js"></script>
```

However there are a few more setups that needed to happen before" songs" was defined as a constant. This needs to be defined with "let" as we're going to change the order of the songs.

```
let songs = ["dubstep", "betterdays", "sunny"];
```

I also didn't include in my original variables getting the id for the "random" which is the id given to the shuffle button.

```
const shuffleButton = document.getElementById("random");
```

Also included it into the export

```
export {
    musicContainer,
    playButton,
    prevButton,
    nextButton,
    audio,
    progress,
    progressContainer,
    title,
    vinyl,
    currTime,
    duraTime,
    songs,
    songIndex,
    shuffleButton
};
```

Here is shuffle.js this is where the shuffle button functions come in, I have it so that the function shuffles the array, by getting the array length and a random index number, it checks all the elements within the array and sees what it hasn't shuffled already and picks a random element and swaps it with the current item in the array (so the item 0 is now item 3 and item 3 is now item 0)

```
import { shuffleButton, songs, songIndex } from "../variables.js";
import { loadSong, playSong } from "./playPause.js";

function shuffle(array) {
    let shuffleIndex = array.length,
        randomIndex;
    //While there are elements left to shuffle
    while (shuffleIndex != 0) {

        //Picks a random element
        randomIndex = Math.floor(Math.random() * shuffleIndex);
        shuffleIndex--;
        // And swaps it with the current element.
        [array[shuffleIndex], array[randomIndex]] = [array[randomIndex], array[shuffleIndex]]
    }
    return array;
}
```

Now just like with Nextprev.js we pass this through a function with an event listener on the button for it to shuffle. This now shuffles all the songs, within the “songs” array whenever you click the shuffle button. This is the video displaying this I’ve added in a console.log as well so you can see the whole array randomize everything.[26.]

```
function shuffleSongs() {
    shuffle(songs); {
        loadSong(songs[songIndex]);
        playSong()
    }
}

//Event Listeners Defined
shuffleButton.addEventListener('click', shuffleSongs);
```

3.3 Database

Currently I’ve gone through all the requirements, however I’m missing playlist creation, search option for audio files within a database and list display options by song track or album.

Some exceptions will need to be made with how I’ve coded everything so far, When it comes to the “list display options by song track or album” this may be an issue as the files shown are all .mp3 files and are read off by the file name (excluding the .mp3 parts) I have done this to the best of my ability by displaying the file name within the music player and having it play music.

Another exception that will need to be made is that the search function will search within an array not a database, as the songs that are stored are put within an array and not a database meaning it will be pointless to sort a database when songs are already done within an array.

After some consideration on how to deal with the playlist database I’ve chosen to use pgAdmin [27.] and postgresql [28]. This is because it’s open source and a good alternative to mysql.

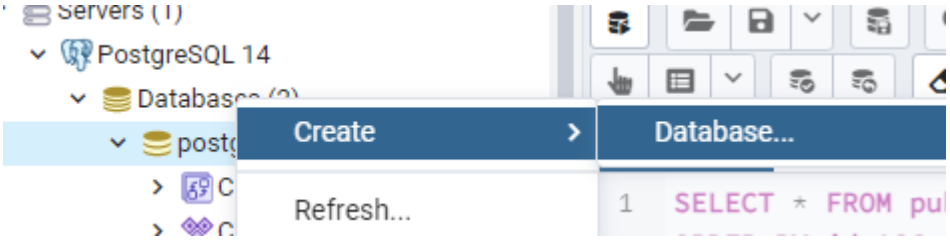
Database Playlist Creation

I have to install the packages for postgres.

```
PS C:\Users\WentaoShum\Desktop\Music Player Project> npm install pg

added 17 packages, and audited 454 packages in 5s
```

.

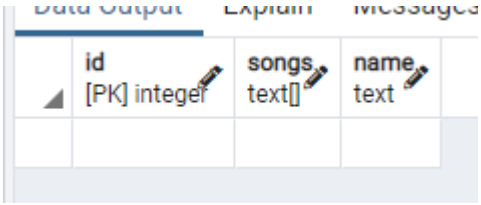
Below is how to connect the pg database client to the index.js file this is done by having a new client which is the user connection to the database, bodyParser is required as well as we are going to pass information through the body of the html document. To add in my database credentials into database connection I use a const client and add in my credentials in order to connect, I also need to pre create a database within pgadmin called “project_music_player” any and all tables that are added will be within this database.



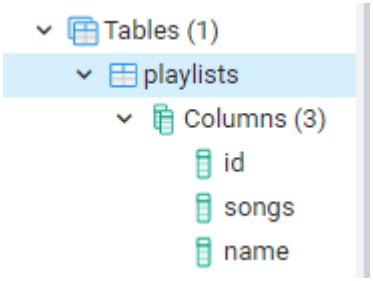
I then start off by creating a database query and create a table in the called “playlists” when the page loads it creates the table with the columns “id”, “songs” and “name” with “id” as the primary key.

```
const { Client } = require('pg');
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

//Create database connect file for it's self
const client = new Client({
    user: 'postgres',
    host: 'localhost',
    database: 'project_music_player',
    password: 'password',
    port: 5432,
})

client.connect()
    //creates the database if it doesnt exist with id and songs
client.query(`
    CREATE TABLE IF NOT EXISTS "playlists" (
        "id" SERIAL,
        "songs" text[],
        "name" text,
        PRIMARY KEY ("id")
    );`, (err, res) => {
        console.log(err, res)
        client.end()
    })
    //Doing this songs list may lead to the memory may run out if there are too many songs
```

Here is the database being viewed within pgadmin (currently has no data in as we just made the table)



Here is the table within another view within pgadmin showing the columns and tables.



Now within index.HTML I need to create a way to create a way to input a playlist name and playlist song.

```
<div>
    <form id="playlist_form" action="/create/playlist" method="post">
        <label for="playlist_name">Name:</label>
        <input type="text" name="playlist_name"><br><br>
        <label for="playlist_name">Song:</label>
        <input type="text" name="playlist_song>"><br><br>
        <button type="submit" class="btn-link">Go</button>
    </form>
</div>
```

Unstyled playlist form.

This is my first attempt at making the client query to input data from the HTML form into the psql database. I start off with an app.post to define that I am posting data.
I state what I want the name and song constants to be, so the req.body allows access to data from HTML, it uses the data inputted from the form identified by id. I will set up the client user connection to the database. And now to the create query, I look to insert the name and songs constant defined before into the playlists table under the “name” and “song” heading however this first attempt threw me an error.

```
app.post('/create/playlist', function(req, res) {
    const name = req.body.playlist_name
    const songs = req.body.playlist_song
    const client = new Client({
        user: 'postgres',
        host: 'localhost',
        database: 'project_music_player',
        password: 'password',
        port: 5432,
    })
    client.connect()
        // update playlists
    client.query(`
        INSERT INTO playlists(name,songs) VALUES('${name}' ,'${songs}');
       `, (err, res) => {
        console.log(err, res)
        client.end()
    })
    res.send('POST request to the homepage')
})
```

This is the error it threw within the terminal in VScode.

```
error: unterminated quoted string at or near "'playlistname ,undefined);
       "
```

To start investigation of this issue by console.logging the “name” and “songs” and also the req.body

```
app.post('/create/playlist', function(req, res) {
    const name = req.body.playlist_name
    const songs = req.body.playlist_song
    const client = new Client({
        user: 'postgres',
        host: 'localhost',
        database: 'project_music_player',
        password: 'password',
        port: 5432,
    })
    console.log(name, songs)
    console.log(req.body)
    client.connect()
        // update playlists
    client.query(`
        INSERT INTO playlists(name,songs) VALUES('${name}' ,'${songs}');
      `, (err, res) => {
        console.log(err, res)
        client.end()
    })
    res.send('POST request to the homepage')
})
```

This is what console.logs returned, and as soon as I saw this I realised what the error was, it was that the playlist_songs had an extra “>” tag at the end which made it undefined.

```
playlistname undefined
```

```
{ playlist_name: 'playlistname', 'playlist_song>': 'playlistsong' }
```

```
<div>
    <form id="playlist_form" action="/create/playlist" method="post">
        <label for="playlist_name">Name:</label>
        <input type="text" name="playlist_name"><br><br>
        <label for="playlist_name">Song:</label>
        <input type="text" name="playlist_song>"><br><br>
        <button type="submit" class="btn-link">Go</button>
    </form>
</div>
```

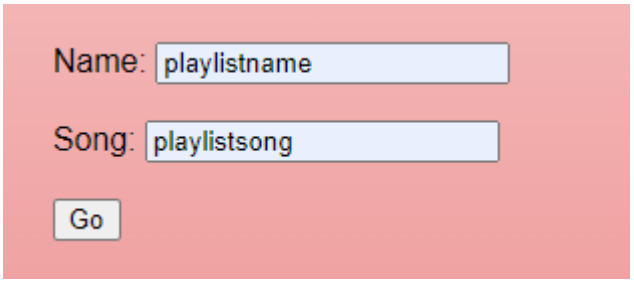
However from here I ran the function with the correction to the name “playlist_song” it stated that there is a malformed array. Which is because I forgot that the songs and names aren’t formatted the same and songs are formed in an array. Meaning that in order to correct this I need to change the song's value within the query.

```
error: malformed array literal: "undefined"
```

This is the final change and the completed version of the app.post in order to create a playlist entry within the database. (With removed console.log and the correct “songs” value)

```
app.post('/create/playlist', function(req, res) {
    const name = req.body.playlist_name
    const songs = req.body.playlist_song
    const client = new Client({
        user: 'postgres',
        host: 'localhost',
        database: 'project_music_player',
        password: 'password',
        port: 5432,
    })
    client.connect()
        // update playlists
    client.query(`
        INSERT INTO playlists(name,songs) VALUES('${name}' ,'{${songs}}');
       `, (err, res) => {
        console.log(err, res)
        client.end()
    })
    res.send('POST request to the homepage')
})
```

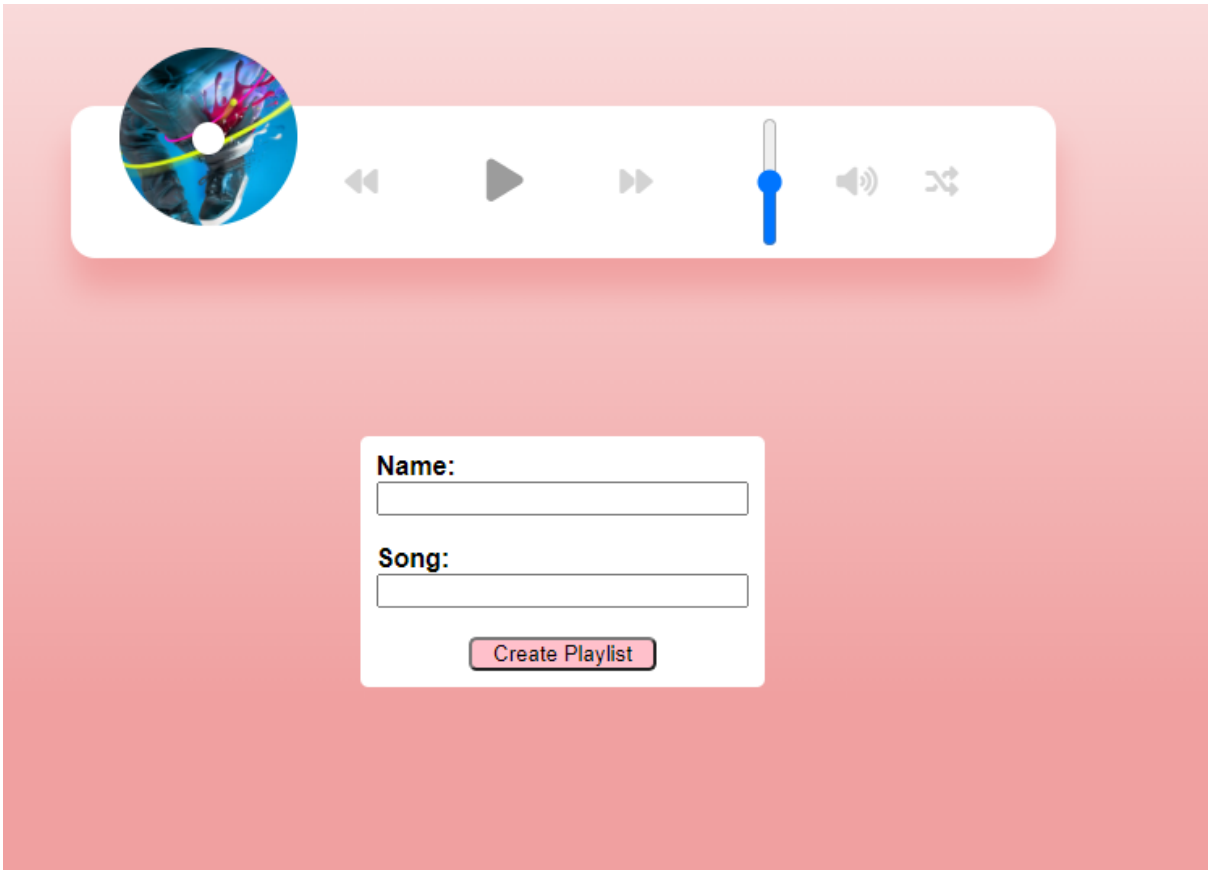
So when I input into the form (currently unstyled) playlistname will go under name in the playlist table and playlistsong will go under songs within the database and id auto increments.



	id [PK] integer	songs text[]	name text
1	1	{playlistsong}	playlistname

Below is the finished styled playlist form. I wanted to keep it consistent so it is based off of the player above. This is the HTML updated to include the css style classes.

```
<div class="playlist-creation">
    <form id="playlist_form" action="/create/playlist" method="post">
        <label class="label-text" for="playlist_name">Name:</label>
        <input class="label-input" type="text" name="playlist_name"><br><br>
        <label class="label-text" for="playlist_name">Song:</label>
        <input class="label-input" type="text" name="playlist_song"><br><br>
        <button class="btn-link" type="submit">Create Playlist</button>
    </form>
</div>
```



This is the CSS styling for the playlist form. I made the label-text the same text formatting as used in the music player for the song name.

```css
/* Playlist Creations form CSS */

.playlist-creation {
    background: white;
    border-radius: 5px;
    margin: 10px;
    padding: 10px;
}

.label-text {
    margin-block-start: 1.33em;
    margin-block-end: 1.33em;
    margin-inline-start: 0px;
    margin-inline-end: 0px;
    font-weight: bold;
}

input[type=text]:focus {
    background-color: lightblue;
}
```

I've also added a pink button to match the background and aligned it within the center of the form.

```css
.btn-link {
    text-align: center;
    align-content: center;
    margin: 0;
    background-color: pink;
    border-radius: 5px;
    width: 50%;
    margin-left: 25%;
    margin-right: 25%;
}

.label-input {
    width: -webkit-fill-available;
}
```

3.4 Search Function and unfortunate events

When I am developing the search function my original plan is to use fs within node this allows me to read all the file names within a directory.

```
//require the node.js file system module
var fs = require('fs');
var files = fs.readdirSync('public/music/');
```

From this I would import the variable “files” and make songs = files to make it load songs onto the page dynamically.

```
import { files } from "../../index.js";
```

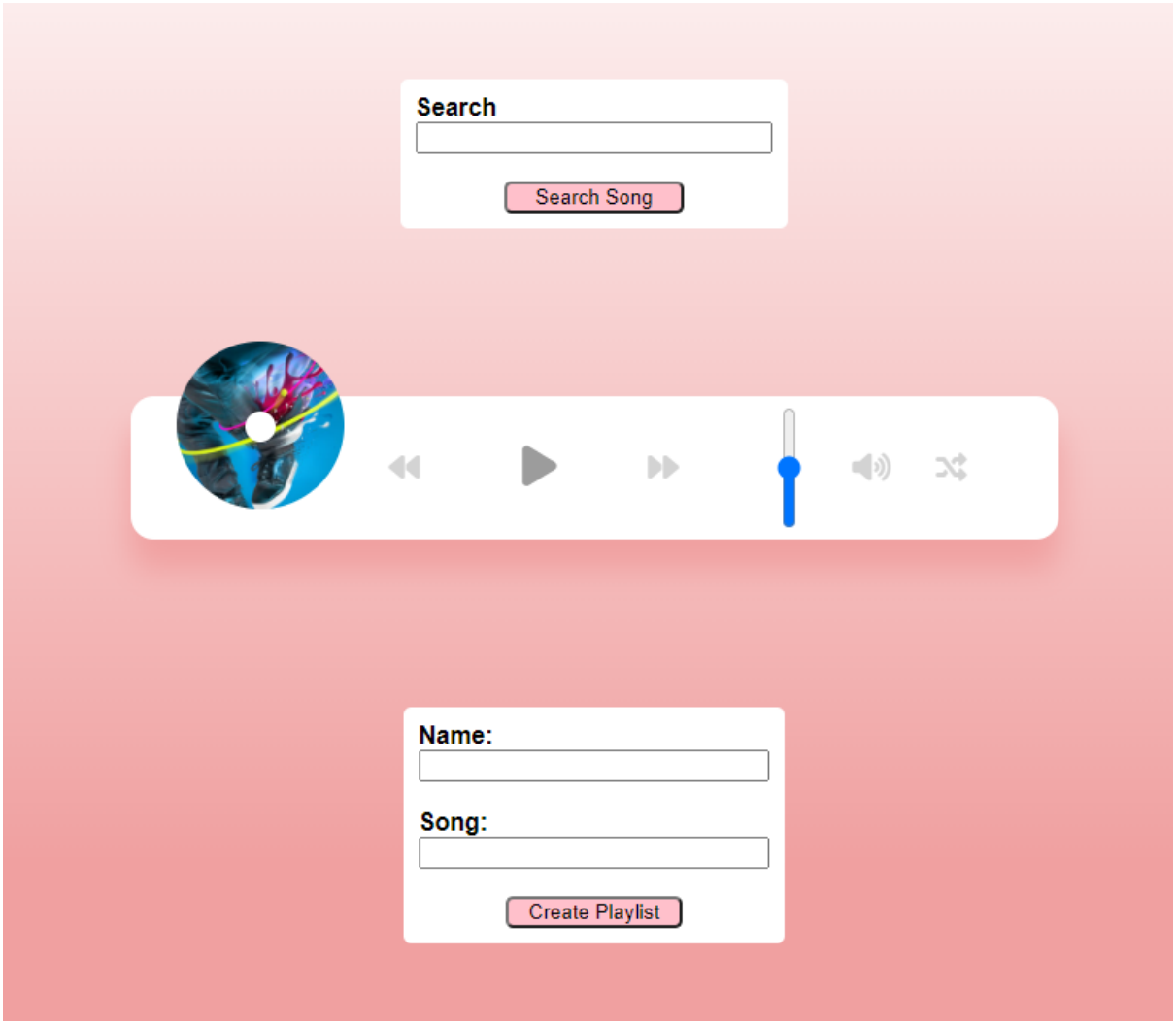
```
let songs = files
```

Then from being able to import files I would write a HTML form with the input having the “.song-search” class.

This is the HTML Form that I would create.

```
<div>
    <form>
        <label class="label-text">Search</label>
        <input class="label-input song-search" type="text"><br><br>
        <button class="btn-link" type="submit">Search Song</button>
    </form>
</div>
```

This is what the webpage would look like with the Search Function.



Then I'd create the search.js file, query select the input by the ".song-search" class (this can also be done within variables.js exported and imported into search.js) create an event listener then whenever there was change within the input to ".song-search", it would compare the values of the input to files and check if it was a song next, if it was a song then then I would have it play the song.

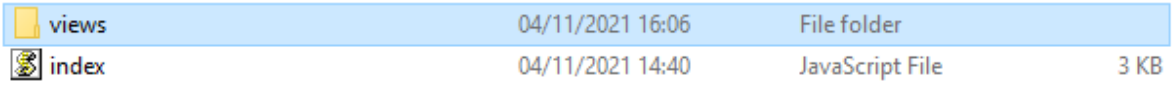
```
const { songs } = require("../variables");

const ele = document.querySelector('.song-search');

ele.addEventListener('change', (event) => {
  files.filter(file => {
    file.includes(ele.value);
    {loadSong(songs[file]);
    playSong()
    }
  })
});
```

However, having discovered and thought about it later, I can't import "files" variables from index.js to variables.js as you can only import from module files and index.js is not a module file.I will have to leave this function here as I will be able to find a way to export a server-side node.js variable to client side javascript.

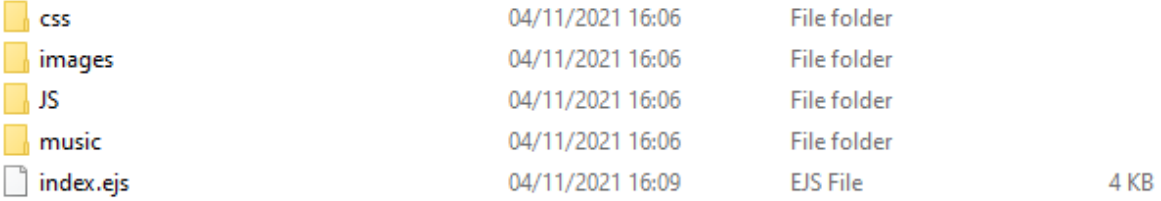
3.5 EJS and server side and client side sharing variables

I've later found a solution to this by using EJS 29.which is a templating language, compiling with express view (express.js), supporting both JS and browser support, with fast compilation and rendering and template tags, which means that I can use the "files" variable from server side to my client side.

In order to use EJS I need to set up EJS. Firstly I'll change the file structure as EJS uses views. Which index.js will be left on the outside and then index.html (soon changed to index.ejs) will also be added to views, along with the css, JS, music and image files.



This is inside the views folder. I've also renamed index.html to index.ejs.



Like with others packages I need to npm install ejs

```
C:\Users\WentaoShum\Desktop\Music Player Project> npm install ejs
```

App.set sets the view engine for ejs.

```
//sets EJS as the view engine
app.set('view engine', 'ejs');
```

And the app.getfunction changes from res.sendfile to res.render as we are now using a ejs format.

```
app.get("/", function(req, res) {
    //Sends the file to the user domain
    res.render('index');
})
```

As I'm passing files through I will need to call it hear in order to pass it through to index.ejs

```
app.use(express.static("public"));
app.get("/", function(req, res) {
    //Sends the file to the user domain
    res.render('index', { files: files });
})
```

When I added in "var songs = <%= files %>" which is how files is passed through however it would try to read the whole of the arrys like a string.

```
GET http://localhost:8081/images/betterdays.mp3,dubstep.mp3,sunny.mp3.jpg 404 (Not Found)
GET http://localhost:8081/music/betterdays.mp3,dubstep.mp3,sunny.mp3.mp3 404 (Not Found)
```

So I've used ".split(',') in order to split all the items within the array and by using map, it will go through each new element and remove the ".mp3" from the end of each item.

```
<script>
    var songs = "<%= files %>"
        //This is where the songs gets split up from the array
        //and iterates and removes the .mp3 from the end
    songs = songs.split(',').map(song => {
        return song.replace('.mp3', '')
    })
</script>
```

I then checked back in the index.ejs and corrected my form, if I'm going to use a 'change' listener then I do not want it to be in a form as the form refreshes the whole page while you press enter, and also I removed the button as they has no purpose with it not being a form anymore.

```
<div class="form-creation">
    <label class="label-text">Search</label>
    <input id="song-search" class="label-input" type="text" name="song-search"><br><br>
</div>
```

Also within the CSS I changed the .playlist-creation class to form creation in order to reuse the css I have done before and properly define it.

```
.form-creation {
    background: white;
    border-radius: 5px;
    margin: 10px;
    padding: 10px;
}
```

Like always I've added the script to my index.ejs.

```
<script type="module" src="JS/Functions/search.js"></script>
```

And I've changed the functions as well from the one I've written before, I've now added the import to get "loadSong" and "playSong".

For now I wanted to test the function so I used document.getElementById to get the input which is called at const ele..
Pass through the ele through the event listener and then filter the input of the song with the value of ele. Then if it includes the value within song it will load the song and play it.

```
import { loadSong, playSong } from "./playPause.js";

const ele = document.getElementById('song-search');
ele.addEventListener('change', (event) => {
    songs.filter(songs => {
        songs.includes(ele.value); {
            loadSong(ele.value);
            playSong();
        }
    })
});
```

Once this worked I then went back to variables.js and gave the searchBar a proper constant value with an appropriate name, "searchBar"

```
const searchBar = document.getElementById('song-search');
```

I then passed it through export.

```
export {
    musicContainer,
    playButton,
    prevButton,
    nextButton,
    audio,
    progress,
    progressContainer,
    title,
    vinyl,
    currTime,
    duraTime,
    songIndex,
    shuffleButton,
    searchBar
};
```

And then imported the new searchbar const through the search.js file and into the function. This is to keep the code clean and keep all the variables in one place. This is the final result of the search function [30.]

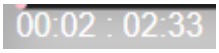
```
import { loadSong, playSong } from "./playPause.js";
import { searchBar } from "../variables.js";

searchBar.addEventListener('change', (event) => {
    songs.filter(songs => {
        songs.includes(searchBar.value); {
            loadSong(searchBar.value);
            playSong();
        }
    })
});
```

3.6 Finishing touches and clean up

A bit of slight cleaning to make the time look better at the end whilst checking, I've added a with a colon just to separate the two times a bit more.

```
<div class="title time" id="currTime"></div>
<span>:</span>
<div class="title time" id="duraTime"></div>
```

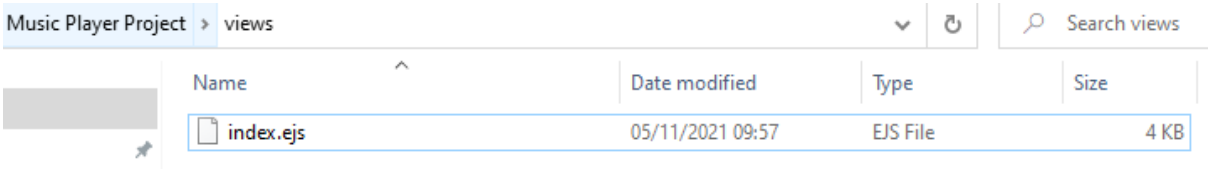
This is how the time within the player is displayed now.



Within the form creation I've added some validation in order to limit the user from inputting invalid data.

```
<div class="form-creation">
    <form id="playlist_form" action="/create/playlist" method="post">
        <label class="label-text" for="playlist_name">Name:</label>
        <input maxlength="256" minlength="1" required class="label-input" type="text" name="playlist_name"><br><br>
        <label class="label-text" for="playlist_name">Song:</label>
        <input maxlength="256" minlength="1" required class="label-input" type="text" name="playlist_song"><br><br>
        <button class="btn-link" type="submit">Create Playlist</button>
    </form>
</div>
```

Also previously files were mistakenly added to the views folder however index.ejs should be the only file in that folder this has now been changed.



After having some cleanup within the code and proper file structuring, I have reached all the requirements and am happy to finish my music player project development here.

Testing and filling in the plan

Since I'm happy with the functionality and development of my music player project, I will look to test the final product at the moment, I have been testing throughout and shown previously with any bugs and mishaps that have happened and how I have resolved this however this is the part where I test the wider scope of the functionality on multiple platforms and write my results of tests.

I have constructed an empty test plan here 30. Within this test plan I've included these headers:

Header Name	Description of the header
Test Number	Helps makes every test identifiable
Test Description	Provides more detail of what is within each test
Test platform	Test platform will be the browser that the test will be performed on

Expected Outcome	What I think will happen when I perform the test
Result of Test	What actually happened when I performed the test
Pass/Fail	Quick indicator to show whether a test has passed or failed (failed meaning could lead to a useability issue if not resolved)
Potential Solutions	A starting point for any fails, whoever works on the bug or issue can have a starting point in resolving the issue.

Here is the finished and filled out test plan 31.

Overall the test have been successful for a majority of the tests there have been no issues and only passes and expected results, however with tests in regards to “Searching for a Song and searching for an invalid song name” and “Database should create a new table entry when the playlist form is submitted, the form will not allow to leave it empty or you to have less than 1 character or over 256 characters”, have both been fails in my opinion. These tests fail in my opinion as they can harm the functionality of the music player, for example the database entry not taking into account duplicated and the 256 character limit means that memory can be overused and stress a server or wherever the database is stored. Also with being able to search invalid song names, it can lead to the music player breaking and being unable to play music. I have provided potential solutions to these issues however these were issues that I overlooked whilst I was testing during development, and since we are in the waterfall software development cycle I am not able to go back and develop on these issues.

The differential between browsers have been to a minimum however Test 5, Test 16 and Test 27 are different due to the browser, whilst Chrome once it finishes the song repeats the searched song (test 5), Firefox and Microsoft Edge both continue using the array of songs given and just moves forward (test 16,test 27). I have put test 16 and test 27 as both passes due to them not being what I expected, however they didn’t negatively impact the music player by making it unusable.

In conclusion, overall functionality is very high and the music player works for every requirement however in the long term the search function and the playlist creation form will require more development in order to reach a highly functioning level.

Maintenance and Help Document

Here I will provide a help and maintenance document to allow a user or a developer to get more information on the music player, this will include information from how to set up the music player, using the music player and any known issues. 32.

Final Thoughts

Completed Criteria

I have managed to create a music player based off of key features required within the business requirements, I have managed to include and consider every requirement, as some of there weren't so specific I have interpreted these issues in my own way and come up with a solution (power saving mode was achieved by making darkmode). I have created the music player and allowed it to playback, be able to go into an idle or power saving mode if no user interaction is recorded for over a 30 second period, have a search option for the file within the folder structure, display options by song track or album,allow for a creation of a song playlist, have user control over playback.

Looking back

If I were to go and start over the project again, something I'd do differently is probably implement ejs earlier, one of my biggest problems was finding a way to take a variable from index.js to variables.js as index.js is not a module file, this got me stuck on this issue for quite a while and if I would have discovered and implemented ejs earlier I would have saved a lot of time and energy.

Making Future Adjustments

If I able to be given more time and be able to develop on this I would firstly go back and develop on the previously mentioned bugs (being able to search invalid song name and database validation) I've had an idea of how those will be fixed and spoke about it within the test plan (potential fixes).However aside from the bugs that need to be worked on I'd like to integrate the database more so now with ejs I could provide a dropdown list with all the playlist names and when you select the dropdown it will play the songs in that playlist. Another feature I would like to add is a repeat feature, some people myself included want to just listen to one song on repeat for some time, I'd do it as a toggle and if it's toggled on it will loop the same song and if not it will read back from the songs list.

Personal thoughts

Most developers don't do this however I want to add something new to my document. I just wanted to place any personal thoughts that I have about the project here. Overall I really enjoyed it (it was stressful at times). I learnt a lot as well, with the inclusions of ejs something I can maybe incorporate into future projects. Lots to do though within 5 days the deadline is quite strict, I would have liked to do more and leave any bugs. However due to the schedule I will just have to do so, even though given me a day or two extra I could have had those sorted out. I'm very proud and feel greatly honored to have made it this far and look forward to what I can do in future.

Conclusion

In conclusion I have created a music player according to the requirements given to me, I have made it within the deadline period and provided details from the software development life cycles stages (analysis, design,implementation ,testing, maintenance) I've properly provided diagrams whenever I felt applicable and provided a sources file for all my references. I've also shown coherent structure throughout my code by leaving comments having relevant files, id, class,function names. I have created and filled in the test plan discovering potential issues that can be fixed in the future. I've even provided a help document for new users to the music player and it includes information for future development as well.