# 音 Oto

The Music Platform

Design Document

Neo Sahadeo

## Table of Contents

## Table of Figures

## Executive summary

This design document outlines the development to deployment of **音 Oto**, a fully free music streaming platform, intended as a portfolio piece for Front End Web Development at DKIT. The project aims to demonstrate the my skills in web development by providing a user-friendly interface for streaming music directly from GitLab without a backend server.

## Introduction

The main objective of the project is a music stream site. A shallow clone of Spotify that was entirely free to use and very little to no maintenance for the site.

### Frameworks

- Gitlab v4 API (REST api)
- TailwindCSS
- Svelte / Sveltekit 5

### Pages:

1. Home
2. Sign Up
3. Music Player
4. About
5. Profile

### Features of the site

The features here are the intended list of full features the site should have.

- Queue system
- Notification system
- Navigation system
- Storage control system
- Playback and session management

## Rationale for Tool Selection

An explanation of why I chose the tools that I did.

### Why Svelte / Sveltekit 5

Svelte and Sveltekit has been my choice for front-end framework due to its simple, clear, expandable and rapid development of small to medium sites.

Svelte 5 introduces a new update of managing application state using 'runes' which are wrappers and re-writes of the store system.

### Why TailwindCSS

Tailwind is the only way to rapidly develop reliable CSS for modern web platform. It will save me hours of writing out classes by hand.

### Gitlab REST api

Gitlab offers a 10GB storage quota and 300 raw endpoint calls per minute. As this site is only meant for my portfolio, classmates and friends scalability is not a priority.

## Storyboard

The following are renditions from my PenPot. It is the intended designs for the site and may be subject to change. There will be five pages [Project Link]

The five pages:

1. Home
2. Sign Up
3. Music Player
4. About
5. Profile

### *Home page - Signed In (index.html)*

This is the home page assuming the user has a local account created.

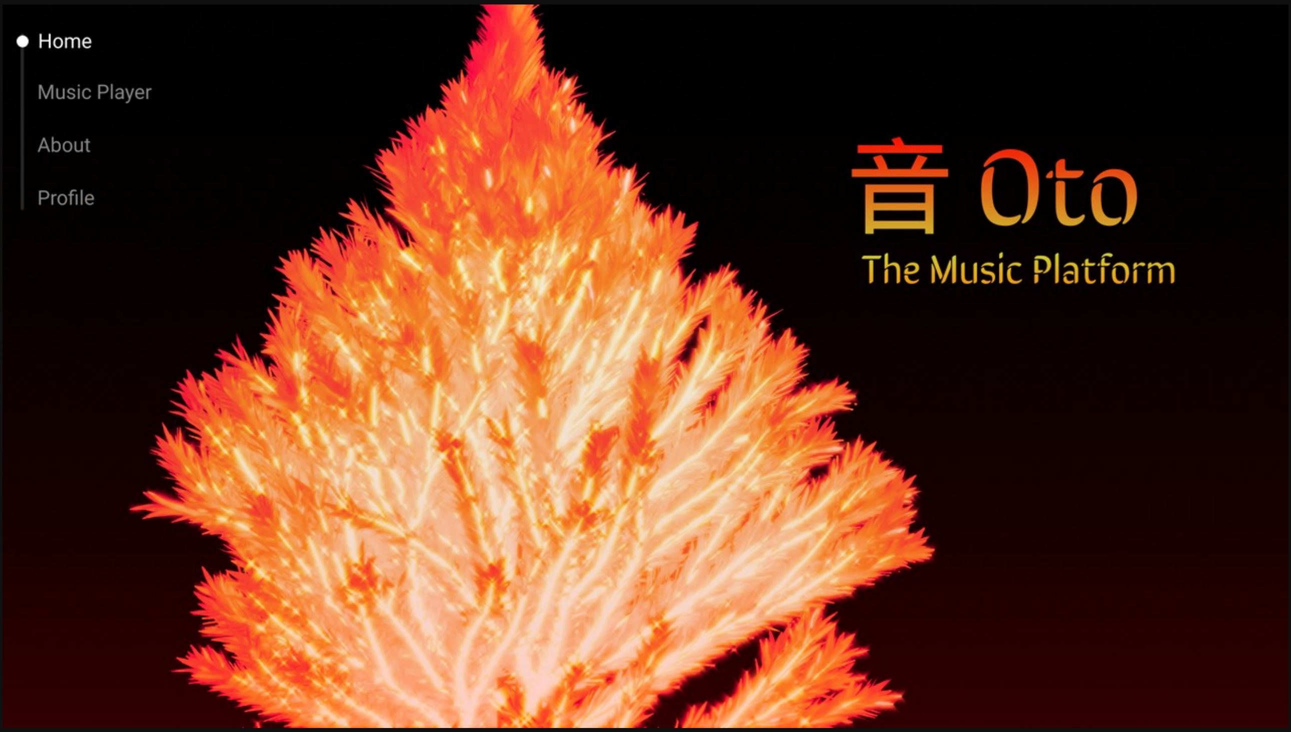


*Figure 1: home page - signed in*

### *Home page - Not Signed In (index.html)*

This is the home page assuming the user is not signed in. They have the option to set up an account.

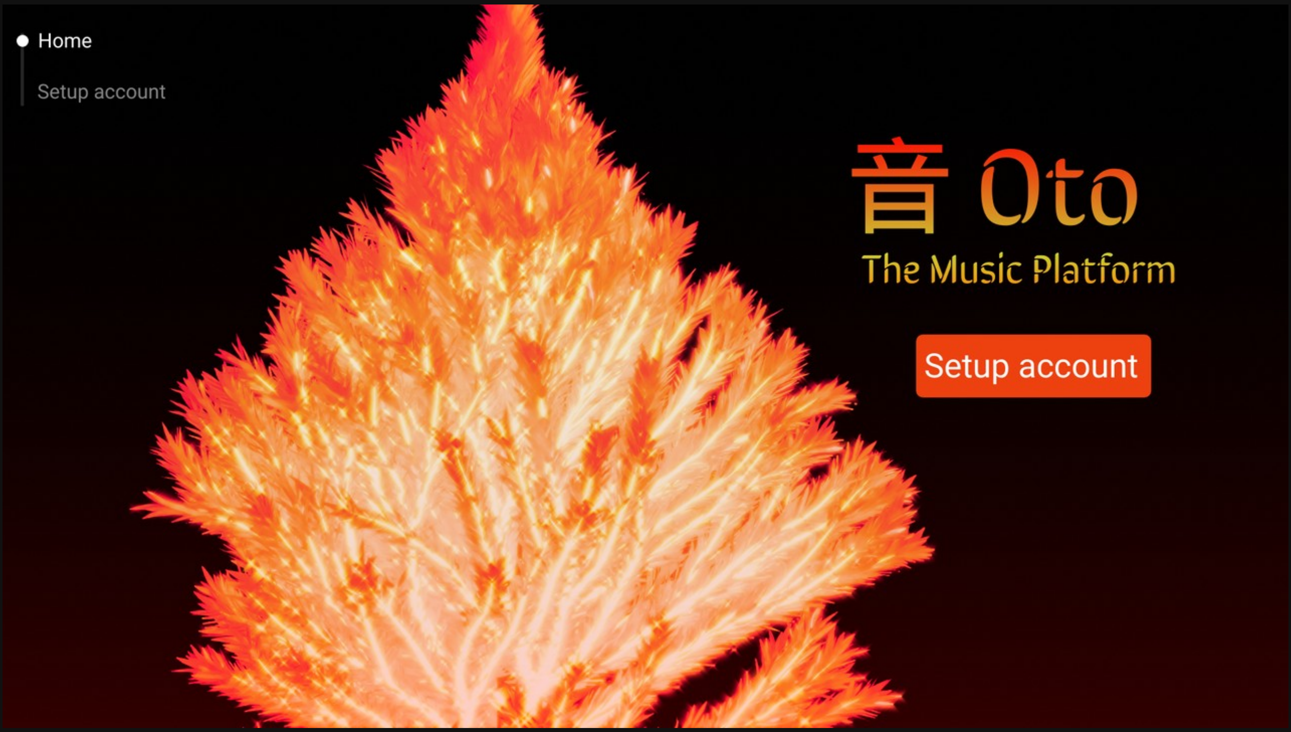
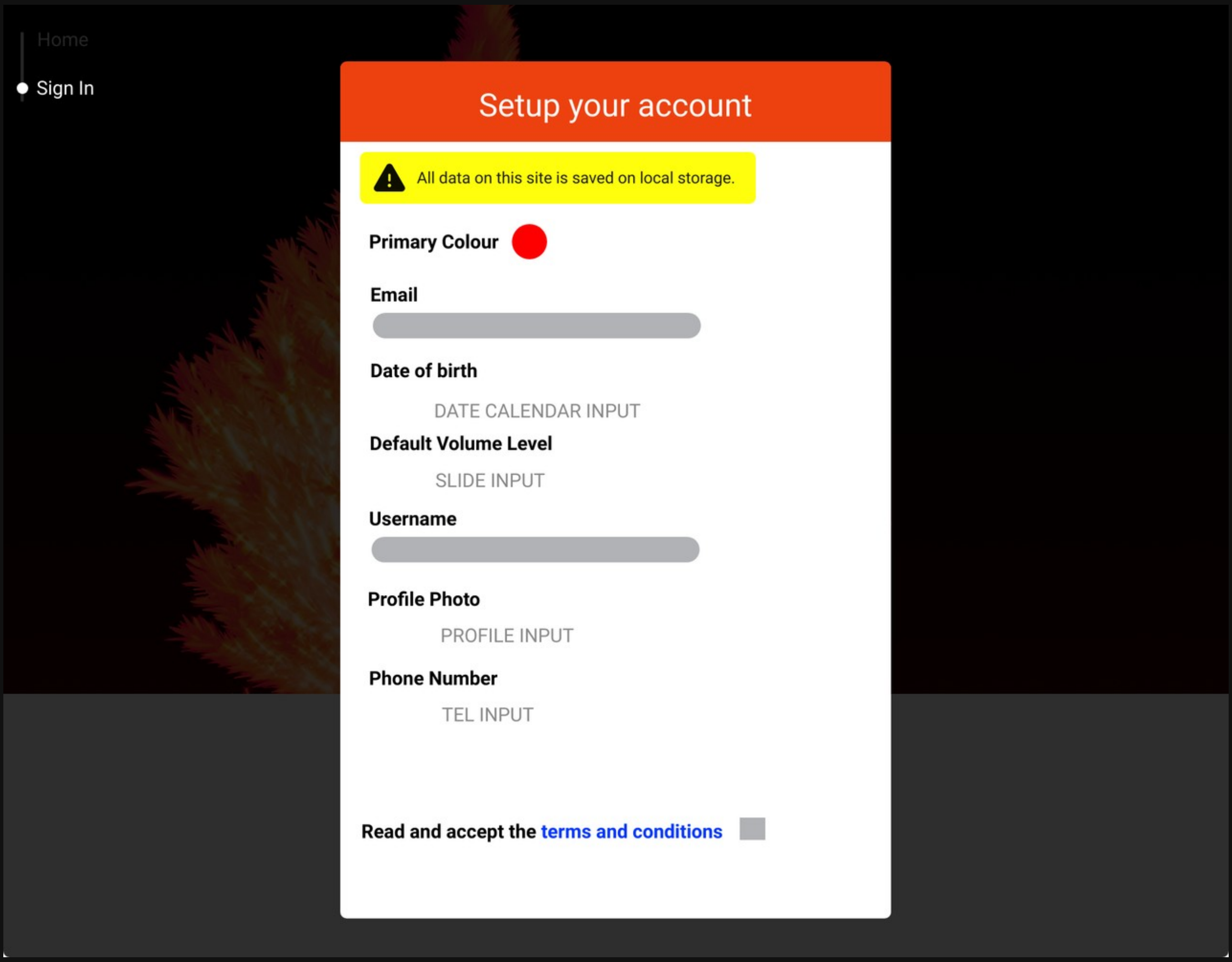


*Figure 2: home page - not signed in*

### ***Account set up page (signup.html)***

This is the sign up page where the user can set some default variables for the site. Some of the values will be cached for later use.



Setup your account

All data on this site is saved on local storage.

Primary Colour

Email

Date of birth

DATE CALENDAR INPUT

Default Volume Level

SLIDE INPUT

Username

Profile Photo

PROFILE INPUT

Phone Number

TEL INPUT

Read and accept the terms and conditions

*Figure 3: sign up page*

### ***Music Player Page***

The music player page will display a rotating image of the album cover, playback position, volume controls, queue controls, shuffle button, playback controls and the current artist that is playing.

There will also be an input bar to search for songs and a cog wheel to let the user change there settings on this page.

#### **Default settings**

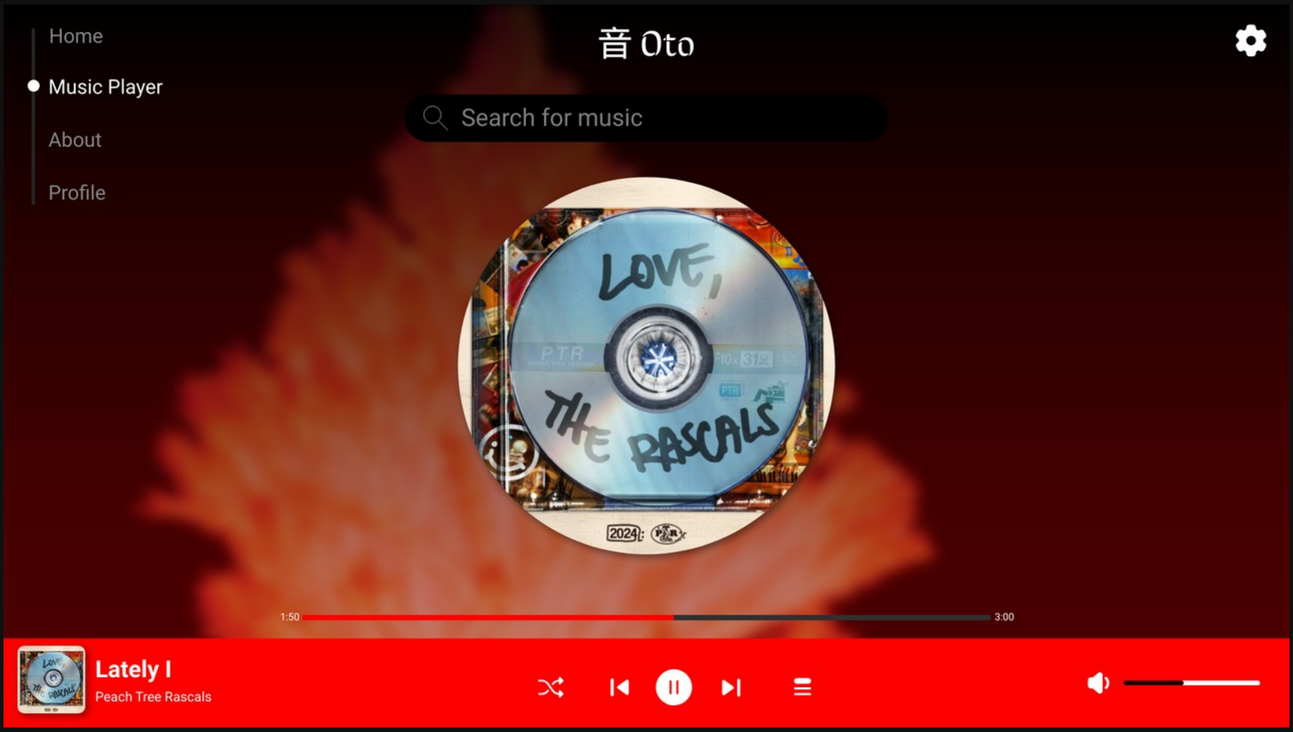


*Figure 4: Music Player - Default Settings*

#### **User defined settings**

The user will also be able to change the colour of their player.

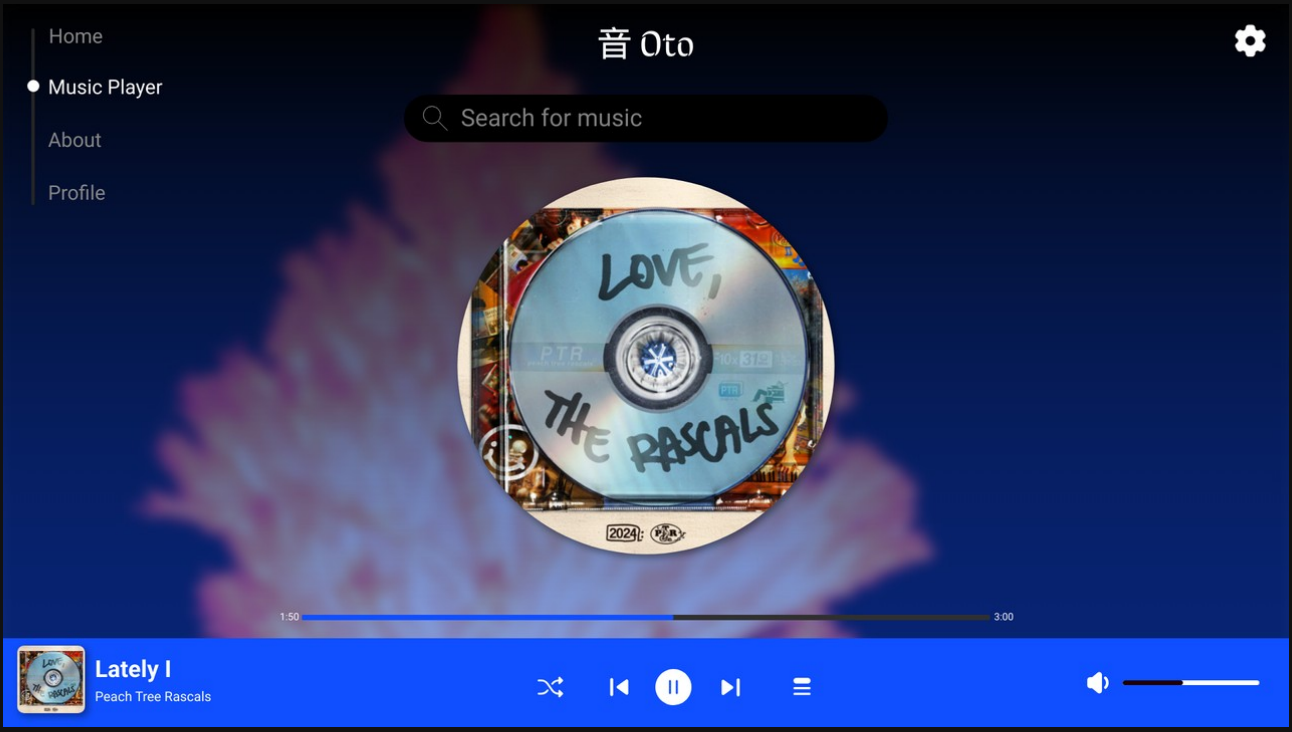


*Figure 5: Music Player - User Settings*

### Queue sketch

This is a rough sketch of what the queue view will look like.

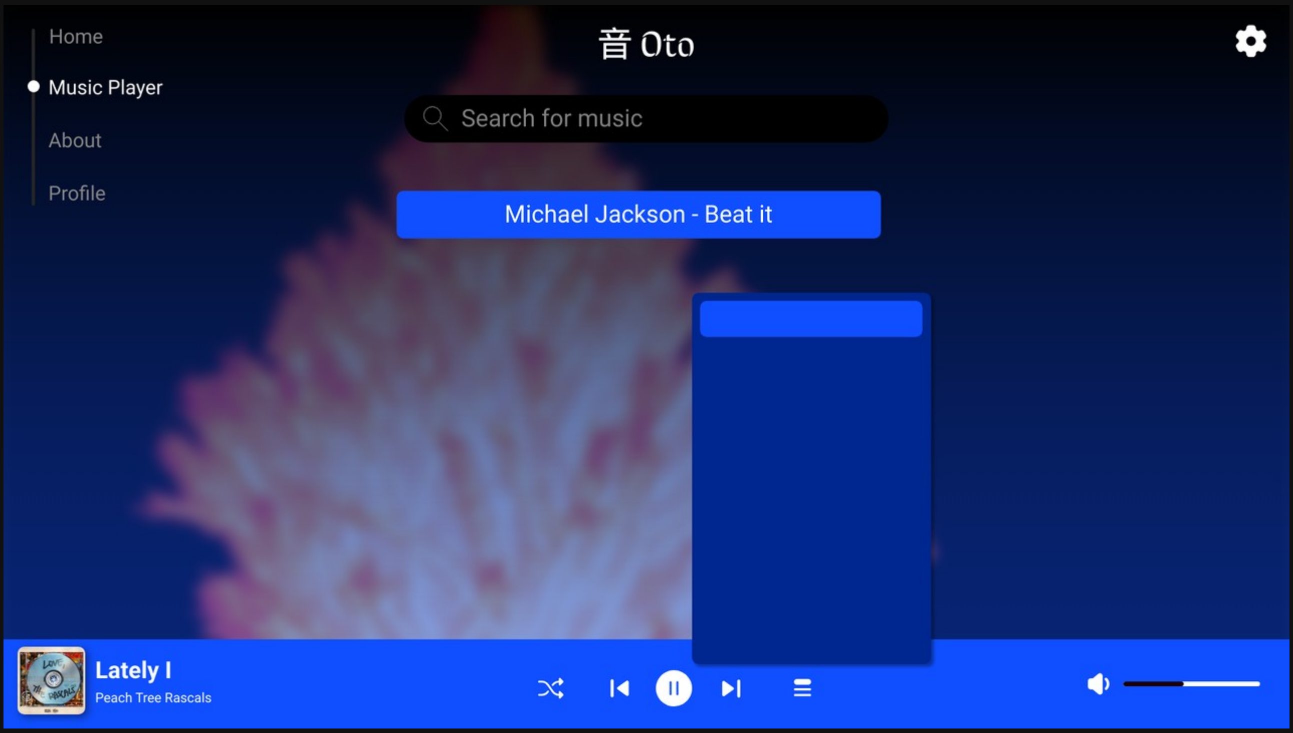
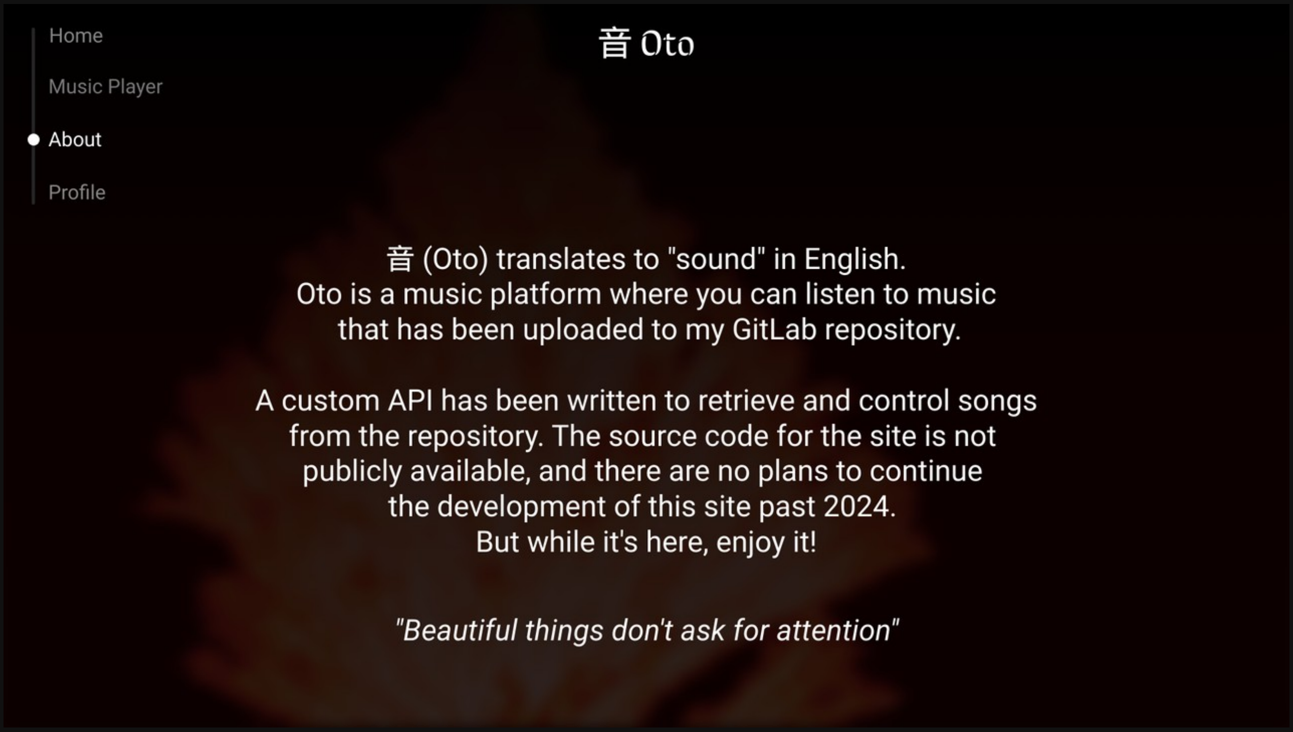


*Figure 6: Music Player - Queue Sketch*

### *About page (about.html)*



音 (Oto) translates to "sound" in English.
Oto is a music platform where you can listen to music
that has been uploaded to my GitLab repository.

A custom API has been written to retrieve and control songs
from the repository. The source code for the site is not
publicly available, and there are no plans to continue
the development of this site past 2024.
But while it's here, enjoy it!

*"Beautiful things don't ask for attention"*

*Figure 7: About page*

## Profile page (about.html)

This is rough sketch of the profile page. The page should let the user change their settings and clear the cache from the page.

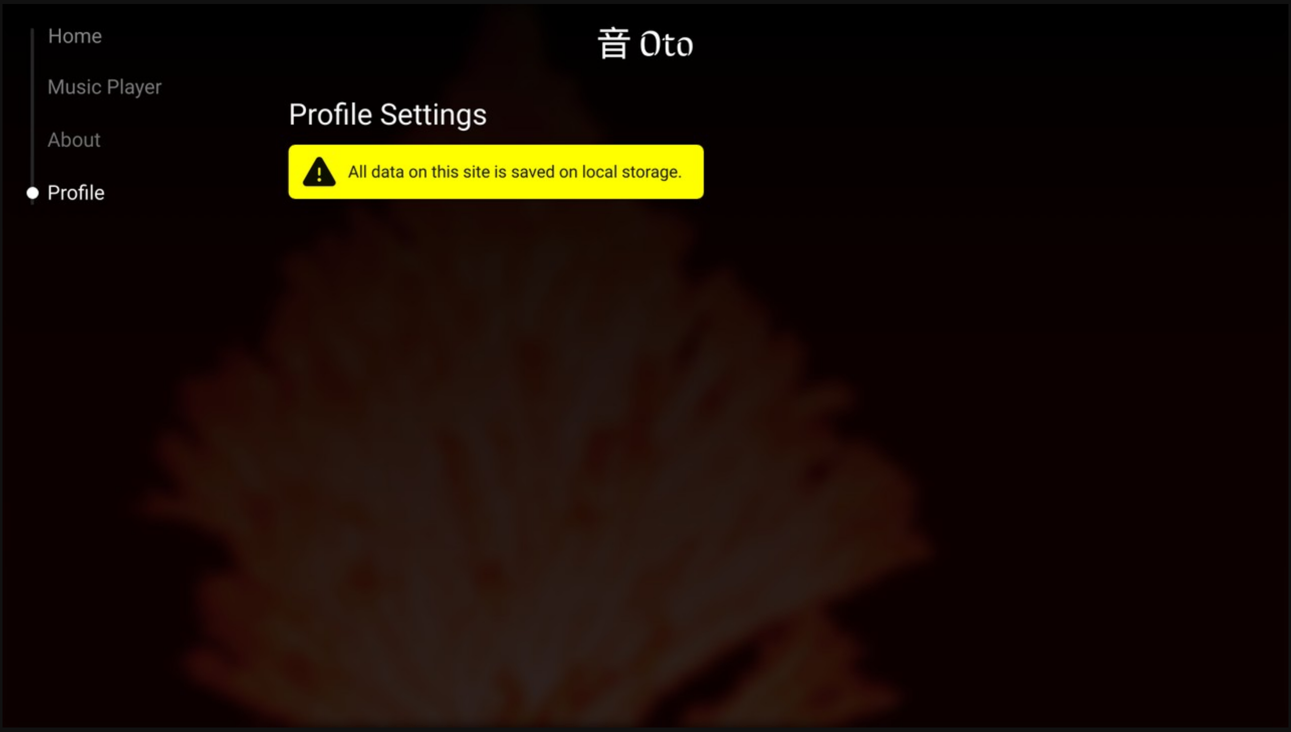


*Figure 8: Profile page*

## Mobile

The mobile aspect of the site is not as well developed. This is a rough idea of what I will design.

- Hamburger navigation menu
- The playback control should be swipe friendly, akin to Spotify

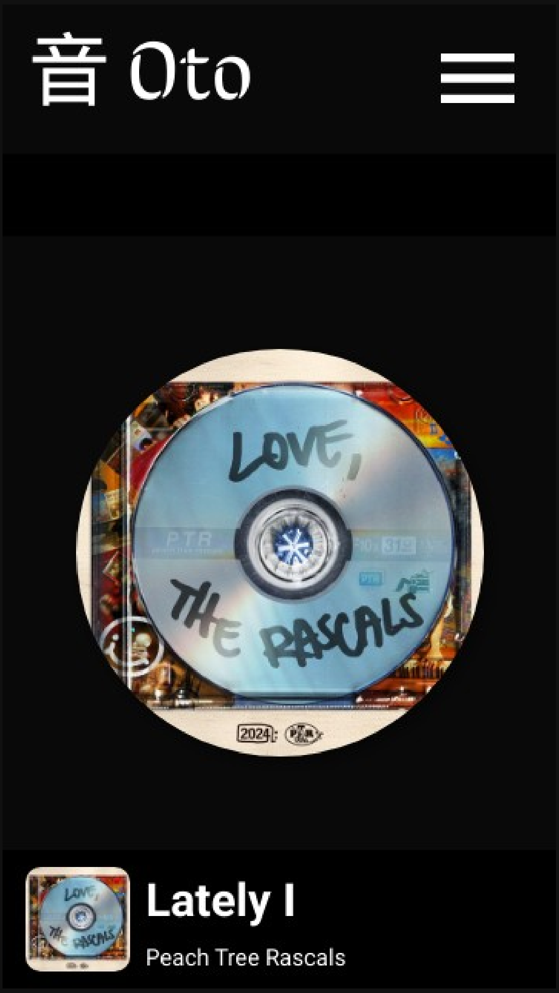


*Figure 9: Mobile Sketch*

## Development

### First stages

Before I started creating the site I read through the docs for Svelte/Svelte 5 as I was under the impression I would being using version four. But it had been recently updated.

The construction of the site started by testing out Gitlabs v4 api to see how data moves around. After this was done I move on towards developing the utility functions, state management and storage controllers for the site.

Through this stage I found that the slider for navigation was hard to use and pop when loading user data. I ended up deciding to scrap the idea for a more simple point and click solution.

localStorage was an issue due to the limited storage capacity of 5MB. Each song was roughly 3MB and would not be able to store it. I read through the indexDB api documentation on MDN and starting writing a system to cache to file.

Issues with Blob was calling the .text() method. The characters were encoded with UTF-8 resulting in data corruption of the songs. Thus, I schemed of a way to generate a arrayBuffer of size uint8 then processing these numbers into ASCII which can then be stringified with the JSON class. This fixed the corruption issues but did introduce a bloat problem where each song will be roughly 7MB. This was good enough for the time being so I left it like that.

All these functions were put onto a page called debug.

### Second stage

After the basic utility function were created I started work on the largest page, that being player. It would have many features.

I started by templating the layout of the site with blocks then filling in each block with the item I wanted. The next steps of the process was connecting visual elements with the corresponding functions.

There were a few issues with desync and updating other as I am not used to the new way Svelte manages its state.

It was roughly at this time where I implements the Fisher-Yates shuffling algorithm which is an inplace array shuffler.

```
export const shuffler = (array: any[], type = 'fisher_yates'): any[] => {
        logger.send('Shuffling array');
        if (type === 'fisher_yates') {
                logger.send('Using Fisher Yates');
                let array_length = array.length;

                while (array_length) {
                        const random_element = Math.floor(Math.random() *
array_length);
                        const x = array[array_length - 1];

                        array[array_length - 1] = array[random_element];
                        array[random_element] = x;
```

```
                                array_length--;
                        }
                }

                return array;
        };
```

## Last stage

The final stage was to create the set up form and generate the other pages; profile about signup home. This stage was extremely quick and easy as all previous functionality had already been written.

### Proposed final stage

Normally when I finish the first iteration of a project, I then rewrite the entire thing in a more OOP / co-op focused direction. Currently the code can be easily changed, but I'm the only one who can easily do it.

## Testing methodology

To test the site I loaded it on different web browsers, display sizes and I've sent the link of my site to fellow classmates and friends.

As for automated testing with a testing library such as Jest or Playwright I decided it would be harder to implement these solution for the project. Reasons for this is that the fetch request is made through the browsers so the API endpoint will have to have CORS enabled. Jest does not support this as it runs in the node.js runtime.

Playwright seemed overkill for the project.

## Deployment

I built a globally shared state in the project called PROD:boolean that if true will switch URLs out for the different endpoint. Normally this is done to qualify a URL as relative links break on a shared domain.

As the mysql site does not serve files like a server (and I do not know if it can) I had to qualify the URLs with a '.html' string at the end. This resulted in an expression such as url_resolver() + 'player' + (() => (PROD ? '.html' : ''))() where after resolving the string, an anonymous function call (that I learnt from TamperMonkey) return a string how either '' or '.html'.

After this I set up a small build script that would statically render the site and zip the file. As for the static render, SvelteKit comes with an SSG that optimises the files and minfiys the code base which decrease load times and file size; The current folder size is 572KB. Once the script runs I manually upload the zipped file to mysql.

Songs are downloaded then committed to my private repository on Gitlab. The current size of the repository of 2.8GB which will be roughly 19GB if it were all saved on the end-user computer.

Technically, I could write a function that would allow end-users to push there own music but it would be hard to maintain the structure of data that way.

I could not download music to the Gitlab service hosted by DKIT as there is a storage capacity of 1GB.

## Conclusion

Through and through I have put in roughly 55 hours of work from research and design to implementation and troubleshooting. This was over the course of a week where it was roughly 5 hours a day after managing other obligations.

It was great fun. I learnt many new techniques and methods of implementing solutions. Some notable examples of these are:

Custom BLOB to ArrayBuffer to ASCII and back. This forced me to learn how Javascript manages its BLOB data and methods and what .text() does.

The localStorage size quota meant I needed to learn how to use indexDB. This was a relatively steep learning curve when compared with localStorage. The main concept I was stuck with was how to requests worked when trying to access the data.

> Its not a great implementation / documented API

Thus I wrote a small storage controller to perform functions such as update open states, setter and getter methods, a logger system to output errors, a proper implementation of promises so that it actually worked and few event managers to store that state of the application in memory to keep avoid race conditions.

Besides these few mishaps, none caught me more off guard by the amount of craziness than a function I wrote myself, more specifically:

```
if (!updated_player_data && can_mutate_player) {
        updated_player_data = true;
        setTimeout(() => {
                update_player_data();
                //player_data.time = current_time;
                //player_data.queue_list = queue_list;
                //player_data.current = current_song as string; // weird desync
bug fixed?
                updated_player_data = false;
        }, update_interval);
}
```

There were issues with the polling request rate not pulling cached data before updating the rotating image. There was also a weird issue of the player loading in before the song data that caused it to mutate the original dataset before the player had time to update. The setTimeout was also extremely necessary. there was probably another solution but everything else I could think of was either blocking or exponential.

Overall, the project was a great deal of fun, hard, and unique. There are so many things I would change and do differently - I would iterate to the end of the world - but as with everything, it should come to an end.