

B.Sc. (Hons) in Software Development



Ollscoil  
Teicneolaíochta  
an Atlantaigh

Atlantic  
Technological  
University

# Rubato: A web-based social networking service for music

By  
**Shane McDonagh**

for  
**Daniel Cregg**  
*Lecturer in Department of Computer Science and Applied Physics*

APRIL 23, 2023

## Minor Dissertation

Department of Computer Science & Applied Physics,  
School of Science & Computing,  
Atlantic Technological University (ATU), Galway.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Project Description . . . . .	2
1.1.1	Project Purpose . . . . .	3
1.1.2	Scope . . . . .	3
1.1.3	Project Objectives . . . . .	4
1.2	Chapter Breakdown . . . . .	4
1.2.1	Methodology . . . . .	4
1.2.2	Technology Review . . . . .	5
1.2.3	System Design . . . . .	5
1.2.4	System Evaluation . . . . .	5
1.2.5	Conclusion . . . . .	6
1.2.6	References and Appendices . . . . .	6
1.3	Project Workspace . . . . .	6
<b>2</b>	<b>Methodology</b>	<b>7</b>
2.1	Research Methodology . . . . .	7
2.2	Software Methodology . . . . .	8
2.3	Models . . . . .	9
2.3.1	Waterfall . . . . .	9
2.3.2	Iterative . . . . .	10
2.3.3	Agile . . . . .	11
2.3.4	Scrum . . . . .	12
2.3.5	Meetings . . . . .	13
2.4	Planning . . . . .	16
2.5	Testing . . . . .	19
2.5.1	Handling of problems . . . . .	21
<b>3</b>	<b>Technology Review</b>	<b>22</b>
3.1	Languages . . . . .	23
3.1.1	HTML . . . . .	23
3.1.2	CSS . . . . .	25
3.1.3	JavaScript . . . . .	28
3.2	MERN . . . . .	29
3.2.1	MongoDB . . . . .	30
3.2.2	Express . . . . .	34
3.2.3	React . . . . .	39
3.2.4	Node.js . . . . .	41
3.3	Heroku . . . . .	43

<b>4</b>	<b>System Design</b>	<b>44</b>
4.1	System Architecture . . . . .	44
4.2	Front-End . . . . .	46
4.2.1	User Interface . . . . .	49
4.3	Back-end . . . . .	51
<b>5</b>	<b>System Evaluation</b>	<b>53</b>
5.1	Testing . . . . .	53
5.1.1	Postman . . . . .	53
5.1.2	Selenium . . . . .	58
5.2	Weaknesses . . . . .	66
<b>6</b>	<b>Conclusion</b>	<b>67</b>
<b>7</b>	<b>Appendix A: User Manual</b>	<b>70</b>
7.1	Introduction . . . . .	70
7.2	Prerequisites . . . . .	70
7.3	Installation . . . . .	70
7.4	Troubleshooting . . . . .	71
7.5	Conclusion . . . . .	71

# Chapter 1

## Introduction

Within this dissertation I will be discussing and noting the work undertaken during my time studying this module. The goal of this module and dissertation is to design and deploy an application which would be similar to one that could be encountered within the software industry, in terms of quality and functionality.

This will be done by collaborating and working alongside my academic supervisor in order to achieve the following:

- Design and implement a project which is of suitable technical challenge and depth
- Apply critical thinking skills and demonstrate the ability to research challenging computer-based problems
- Demonstrate ability to apply appropriate research and software methodologies
- Evaluate and use suitable technologies with an awareness of the current climate within the industry
- Critically evaluate the work done and the potential of future improvements, contributions, and alternatives

### 1.1 Project Description

The project in question is a *web-based social networking service*, which allows for users to review albums and keep track of what they are listening to. This will be an experience which can be shared between other users, where they can view each-others activities. This website can be used as a diary to note and share your opinion on an album, or can be simply used to keep track of albums you

have already listened to. Users will have the option to personalise their profiles with their favourite albums, create custom lists to organise their music and keep a "listen list" for those albums they haven't gotten around to yet.

The platform will allow for users to follow one another, so they can see what their friends are listening to. Reviews can be shared on various social medias based on user desire and interest. This in turn allows for a new and unique experience, among so many websites.

### 1.1.1 Project Purpose

The general reasoning for this application to exist is that currently, there is not a web application which provides this experience in a more *social* aspect. The gap which is present in this field is needed to be filled, which I felt I could contribute to. Since I myself wanted an application of this description available for me to use, I felt that I could develop one myself. Due to this, I feel an incredible amount of passion and enthusiasm for the project.

There are equivalent applications for film and television (e.g. *Letterboxd*), however it is not the same case for music. Since music is such a vast ocean of creativity and brings people together, I felt that a social platform revolving music would be a positive and fun experience. Both developing and using such an application would feel quite satisfying, as I can consider this to be a passion project of sorts.

### 1.1.2 Scope

The scope of this application is currently a solid depth for a solo developing project, especially after discussion with my supervisor. Due to the demands and deliverables of this project, including this dissertation, the scope is at a reasonable size. The features in which I will be implementing will provide a good challenge as a developer.

Deploying a live web application to a URL which can be accessed anywhere at anytime is something I currently have little experience in. Since the full-stack application will also be working alongside an API to fetch relevant metadata and handle user credentials, this provides a difficult enough task.

Due to the focus on reviewing music and tracking user information, there will be a lot of data to keep track of. This is where I feel the most difficulty will lie for me, especially in handling user log-ins.

### 1.1.3 Project Objectives

The objectives of this project, in which will be used to determine if the application is a success are as follows:

- Provide a platform in which users can register and sign-up to use the application
- Produce a system which allows for the ability to search for artists and albums
- Let users review and track albums in which they have listened to
- Allow for users to create and add to lists to keep up-to-date on what they have yet to listen to
- Give the users the ability to follow other users to see what they are listening to and enjoying

To ensure these objectives are met, these explicit requirements will be tested throughout the development life-cycle. Tests will be carried out on each level (unit, integration, regression etc.) to ensure that the metrics for determining a successful application are fulfilled. Implicit requirements will also be tested through performance tests to ensure the application performs as intended. Most tests will be done manually, however certain aspects will be tested with the assistance of specific tools.

## 1.2 Chapter Breakdown

### 1.2.1 Methodology

Within this chapter you will find a breakdown in which I will discuss the methodologies that I applied to my project as a whole. These will include both the research approach that I undertook in relation to technologies, and the software development approach I took. I will display in great detail the approach I took during the development life cycle, the storyboards and backlogs I created during the process, as well as the frequency in which work was done. This chapter will consist of diagrams to give a visual representation of the work undertaken before the coding process began as well. The tools used in accordance to testing will also be mentioned within this chapter, giving them context before I discuss how they were applied in helping within the development of the project.

## 1.2.2 Technology Review

The focus and objective of this chapter is to give a descriptive and thorough review of the technologies integrated into the project. Each technology used will be discussed at a conceptual level and in-depth, to explain the exact reasoning of its implementation (the why of the technologies).

This section will be research-oriented and aims to provide reputable sources when a particular technology is being explored. This helps to understand exactly why a particular technology is being used for the project, and what its functionality will be in the overall structure.

## 1.2.3 System Design

The purpose of this section is to give an overall explanation of the system architecture and the components within. This will showcase the application of the research which was gained during the process of the technology review. Essentially, it describes how these technologies were applied and how they worked together.

To demonstrate the system design, this section will have a lot of visual diagrams to provide a conclusive understanding of the inner workings of the application at hand. Standards will be discussed here. The main focus will be on how the front-end and back-end are implemented and how they work in tandem to allow for the application to function.

## 1.2.4 System Evaluation

To follow the given design of the system, it is also important to evaluate and assess its application. For this reasoning, this section will be used to showcase the testing process which was taken to review our system for robustness and efficiency. Testing will be done for both the routes of the back-end, and the functionality present within the front-end.

Tests and their results will be showcased, alongside any relevant information. This information will be then compared to the objectives in which were discussed in our initial scope.

Lastly, any limitations of the system itself will be touched upon. Alternatives may be proposed which could provide an easier or more efficient experience.

### 1.2.5 Conclusion

It is important to summarize and conclude the information and processes which were explored within this dissertation. This section will briefly re-establish the initial objectives and rationale behind the project. These will be reflected upon again and will reference the insights garnered from evaluating the system from the previous section.

Any and all opportunities which can be investigated further to improve or enhance the finished project will be noted as well. This will provide a good resolution to the project as a whole.

### 1.2.6 References and Appendices

All materials reviewed during the work of this project will be referenced here, alongside their active links. These will be those in which I used for providing context to the main points being discussed, alongside providing evidence of research that was undertaken during this process. This includes any articles, books, or documentation that is relevant to the overall review of the technologies that were selected.

The appendices will contain a relative user-manual, to help with installing and running the project locally on any machine. It is important to be able to determine that the project works as described within the dissertation.

## 1.3 Project Workspace

Within the link featured below, you will find the repository for this project. At this location, you will find the following:

- The full source code of the application, alongside all resources needed
- A detailed README file, which will introduce the application and steps involved to install locally if necessary
- A file containing a link to the screen cast for the application, demonstrating the main features and functionality
- This dissertation, alongside the relevant files
- An extensive commit history, detailing each change made and their purpose

Github Repository: <https://github.com/shanemcdonagh/rubato>



# Chapter 2

## Methodology

The general plan in which is followed for a project is extremely important. This process or methodology that is used, whether within the research phase or the development phase, is as vital as the finished product. These processes determine the quality of the result of a development life-cycle, therefore it is essential to establish a methodology to be followed in both research and development. Within this section, you will find a detailed explanation of a few different methodologies, and the eventual ones that were selected.

### 2.1 Research Methodology

A research methodology is not often associated with software in its traditional usage. In many scenarios, research methodologies are focused on the "how and why" of a particular research paper. It is most often associated with scientific fields. However, research methodologies can be highly useful within software development, and can be useful when adapted to be used to improve the products in which we engineer. [1]

For my project, I knew the route in which I wanted to take, which was a music related web application. However, it was important to be able to establish a way to research exactly how I would take this approach. The research that needed to be conducted was mainly applied to the tools in which I would use for my project and how relevant/useful they were in relation to what I wanted to do. This meant that the development environment had to meet certain criteria in order to satisfy certain elements.

This is where researching came into play. The approach in which was taken was to observe and analyse **secondary data** [2], which helps us avoid sinking time and other resources into researching the technologies and collecting the information we need first-hand. This helps us to focus the time that we have into the development

process of the application itself.

The outcome that we want from our research is that the technologies in which we are observing and collecting information on, meet the specified criteria we have outlined before we begin. The criteria for the technologies we want for our web-based application are as follows [3]:

- **Features:** The extent the current features meet the projects requirements
- **Flexibility:** How adaptable is the technology as the application changes and evolves
- **Interoperability:** How well does it work with other tools
- **Ecosystem:** How healthy is the community around the tool
- **Security:** Can the technology be trusted to keep data safe?

These specific criteria will be applied to the platforms, tools, and the language in which we want to be using for our application. The research will be undertaken and discussed in the *Technology Review* section of this dissertation.

## 2.2 Software Methodology

The approach in which we develop software is extremely important. For us to build high-quality software in a time-efficient process, it is essential to establish a set of rules and processes to follow in developing our application. A software methodology is the ways in which we implement the SDLC ("a cost-effective and time-efficient process that development teams use to design and build high-quality software") [4] into our application, for managing purposes.

Due to the application changing scope, requirements and other challenging aspects, implementing the SDLC (Software Development Life Cycle) helps us maintain control over the application as a whole.

The benefits of the SDLC are as follows: [4]

- Easier to visualize the development process (for the developer and for the supervisor)
- Helps to estimate and schedule specific deliverables
- Reduces risk of problems and loss of time

Due to the fact that this is a solo development project, having a methodology to keep track on all aspects of the application can be extremely helpful to ensure deadlines are being met where they need to be.

## 2.3 Models

There are different approaches and models in which can be taken to implementing the SDLC. Many companies arrange and change the various phases of the SDLC for optimization purposes, to fit what best suits themselves. This is a approach that I tend to use myself as well, by using existing models and changing them to suit my specific needs. It is important to explore and discuss the most common models to help establish the best fit for this application and how the methodology used was selected and changed.

### 2.3.1 Waterfall

A Waterfall model is one of the most well-known, and oldest models within the industry. It takes a sequential and rigid approach, where each stage must be completed before the next step can be started.[5] This keeps a general focused approach, where everything is taken step-by-step.

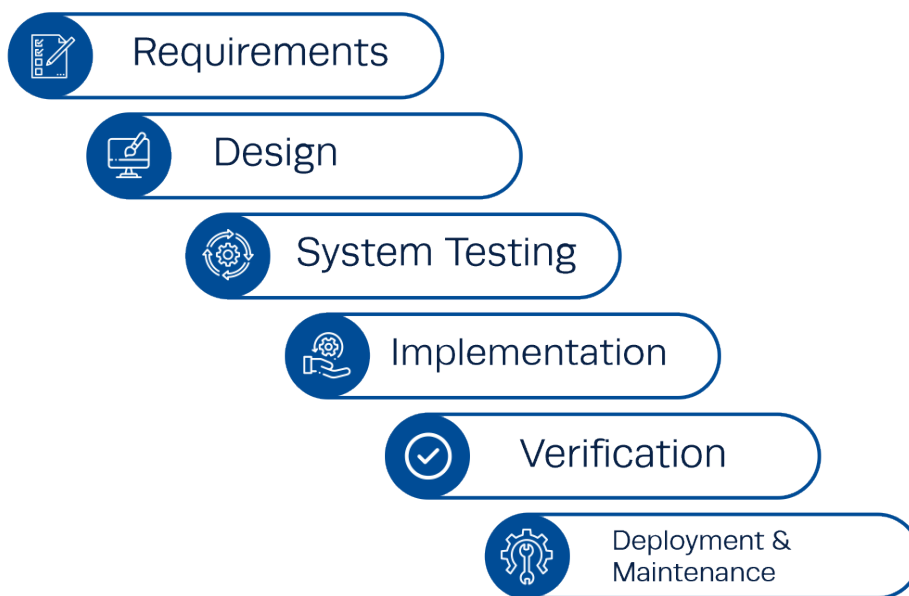


Figure 2.1: Waterfall Model

Although this approach is good for smaller projects which has well-defined requirements, it is not great in adaptability. The waterfall model has quite a heavy focus on documentation as well, which simply would be extremely time-consuming in

relation to this project, not making it a good fit in this instance. It is also known that seeing any relevant application iterations are only possible in later stages of this cycle. This in turn would not fit well with my project, since there would be little to show to my supervisor as we discuss the progress that was being made each given week.

### 2.3.2 Iterative

A iterative approach to implementing the SDLC doesn't follow the same sequential approach as the Waterfall model. Instead of a full specification of requirements, the initial planning and documentation specified only general parts of the software. Once done with the initial planning, we can then focus on the software. This in turn allows us to develop the software continuously and *iteratively* improve it as it goes [6].

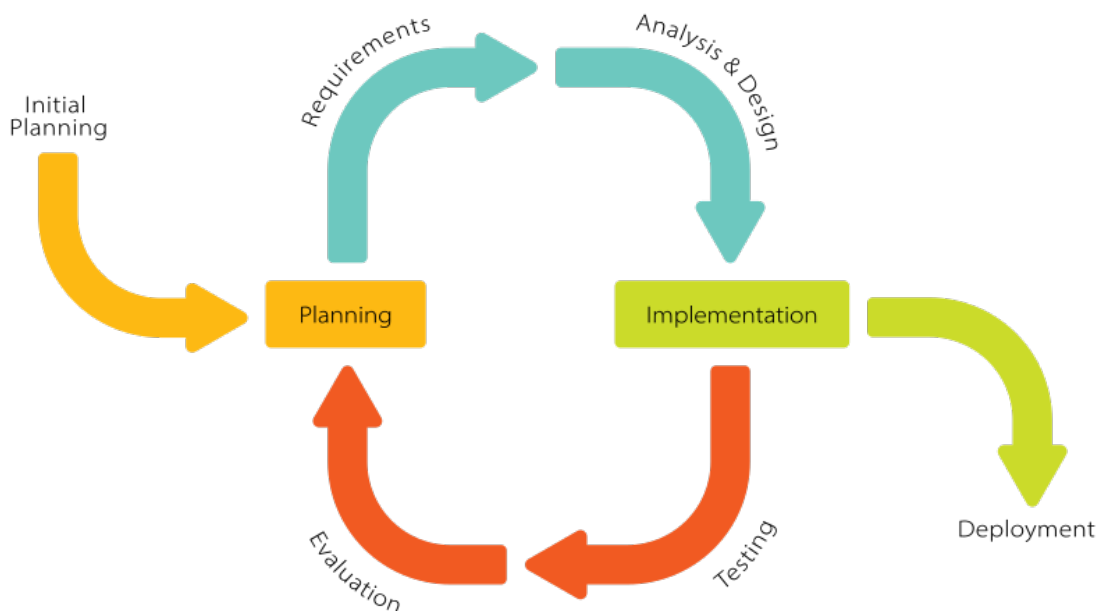


Figure 2.2: Iterative Model

Although this is much more useful for the project at hand, it still is too rigid in its approach. Once we are in our iterative loop, each phase has little to no overlap. It is incredibly useful for being able to reflect and improve on the application (even with defects) but there are other models which help give us much more flexibility in the way we need.

### 2.3.3 Agile

The agile model takes the approach of splitting each phase into several development cycles, being seen as both **iterative and incremental**, being very efficient [7]. The Agile model puts the software first, focusing on the product itself. This is defined by the Agile Manifesto, written by a group of software developers: [8]

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Agile is seen as an "umbrella term", which contains many differing frameworks that follow the basic principle of Agile in different ways.

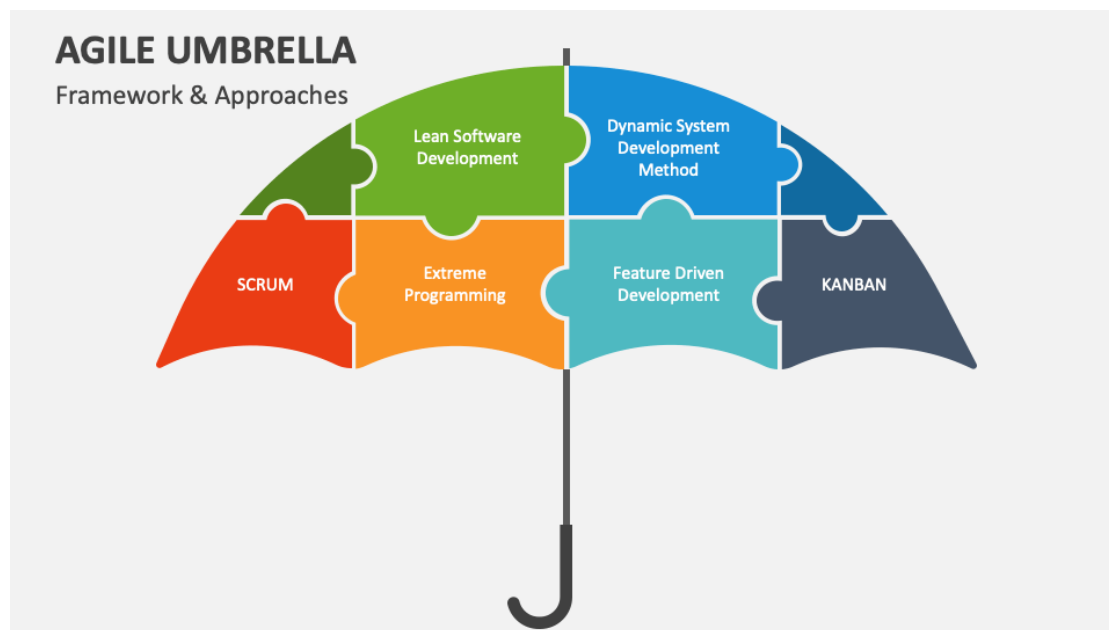


Figure 2.3: The Agile Umbrella

One of the most popular frameworks is Scrum, which is the framework that was eventually used for this project, since it was overall the best fit.

### 2.3.4 Scrum

Scrum can be described as an "empirical approach" to software development, where all decisions are based on observation and experimentation [9]. This is done by carrying out a select amount of work in a set amount of time, described as a *sprint*. Sprints allow us to focus on specific features of the application at a given time, giving the project a much more focused approach

In this particular instance, each sprint was a week duration.

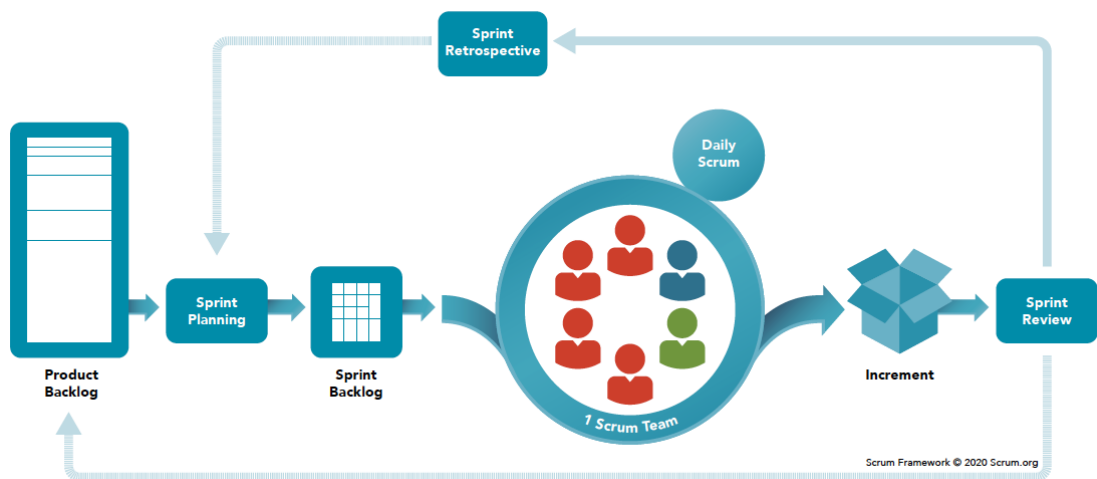


Figure 2.4: Scrum Framework

Since Scrum is a team-based framework, it was adapted to suit the development of a project carried out by a single developer. Highlighted are the various features of Scrum, and how they were adapted for this particular instance.

- **Product Backlog:** Ordered list of what is needed to be implemented (done by developers so no need for changes)
- **Sprint Backlog:** Ordered list of what is needed to be implemented in the current sprint (done by developers as well)
- **Sprint Planning:** Discussing what will be focused upon for the current sprint (This was done alongside the supervisor in our weekly meetings)
- **Sprint Retrospective:** Reflects on the work done and what needs to be added in future sprints (A team effort, adapted to be discussed between my appointed supervisor and myself)

- **Sprint Review:** Reflects on where to improve for an increase of effectiveness (A team effort, adapted for myself to self-analyze my own efficiency)

By adapting the framework to what I needed in relation to my project, I can focus on the other tasks at hand.

### 2.3.5 Meetings

Meetings (which worked as the Sprint Review and Retrospective of a typical Sprint framework in this case), were done in weekly occurrences over the course of development, alongside my supervisor.

All work done within the sprint was discussed and thoroughly demonstrated to my supervisor during this time-frame.

Feedback was verbally communicated and noted after these specific meetings. The sprint backlog (what was being worked on for the next week) was communicated in two ways: By writing it for log purposes in the meeting chat, and by using a *Kan-ban* board, a popular addition to Scrum. Kan-ban boards are commonly used in relation to Scrum, to easily visualize what is currently being worked on in each given sprint.

This helped to keep the general scope at a high enough level where I could implement newer features if thought to be in the realm of possibility or to pull back on certain aspects if necessary. This type of flexibility was extremely vital to keep the project on course.

This allowed me to focus and successfully analyze where work needed to be done and to stay on top of the scope of the project as a whole. It gave an overall structure to the project, to help with management loads.

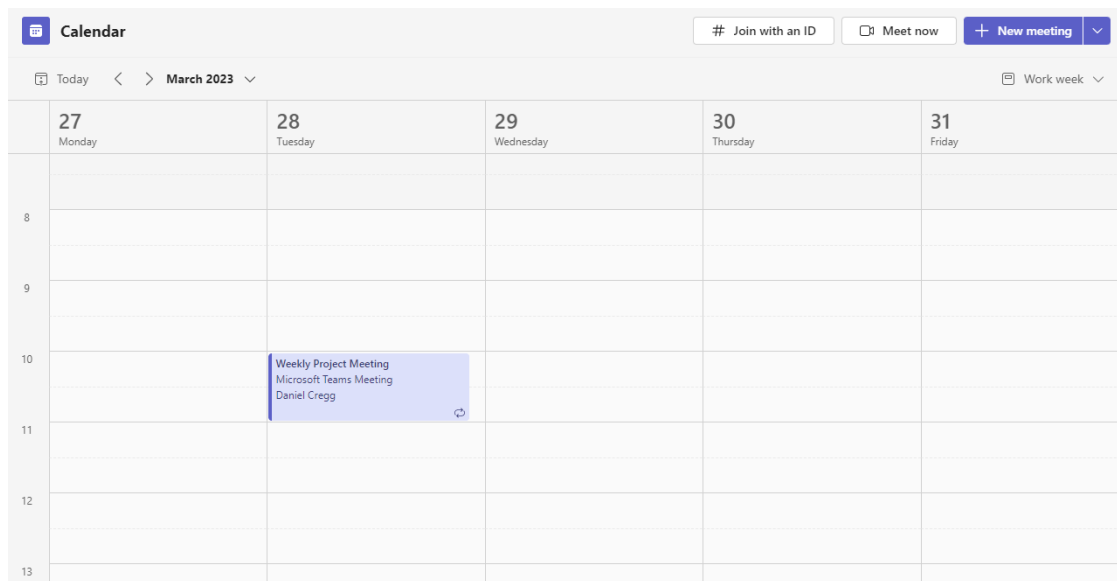


Figure 2.5: Weekly Series of Meetings

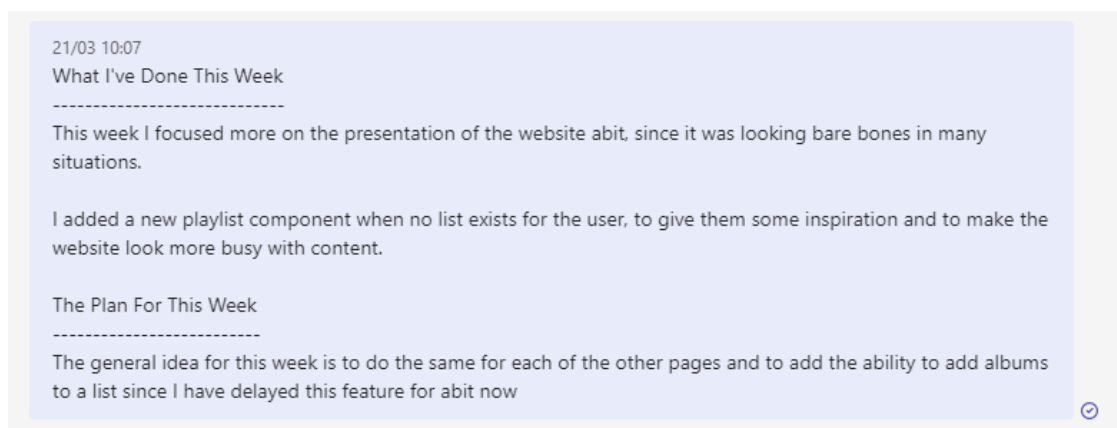


Figure 2.6: Logging of Sprint Planning and Retrospective



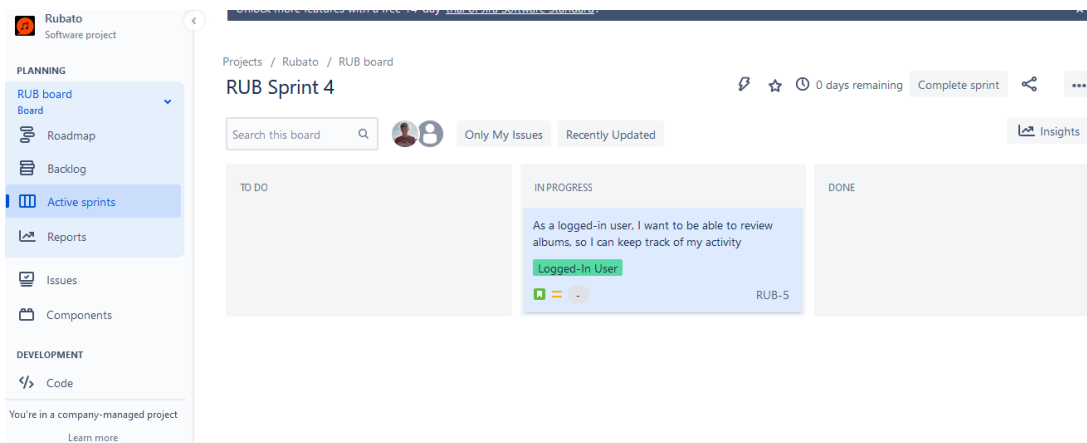


Figure 2.7: Jira Software Kanban Board

## 2.4 Planning

The planning of the project began with initial story-boarding of the web application to get an idea and feel of the user experience as a whole. This process ultimately comes down to creative choices, since design of a website can be relatively subjective. However, it is important to achieve a design which is both easy to use and is aesthetically sound.

Since the application does get inspiration from such websites like Letterboxd, a social media film equivalent, it was important to be able to capture the same type of experience for music fans.

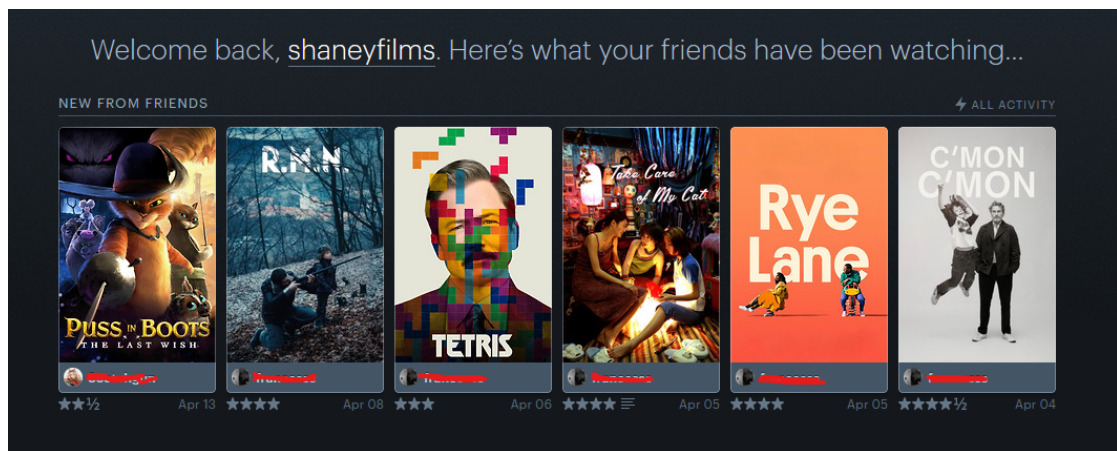


Figure 2.8: Example of my own Letterboxd feed

For this to work as intended, usability and visuals were vastly essential to deliver a satisfying user interaction with the most important features to be easily accessed at any given moment. All pages below (besides the *Home* page) are examples of when the user is logged in, since these features cannot be used otherwise.

For example, the below image corresponds for the first drafts of what I envisioned for the *Home* page and *List* pages.

It is helpful for the user to be able to fully understand what the web application is wanting to achieve in its goal. This is the purpose of the *Home* page. This is communicated by the descriptions and leading text boxes that the user will initially see when they visit the home page of the application, which explains all the features that are available to be used extensively.

The *List* page helps the user track their own listening habits and ratings based on what they've added to specific lists. This adheres to the objective of letting the user track their listening habits as they go.

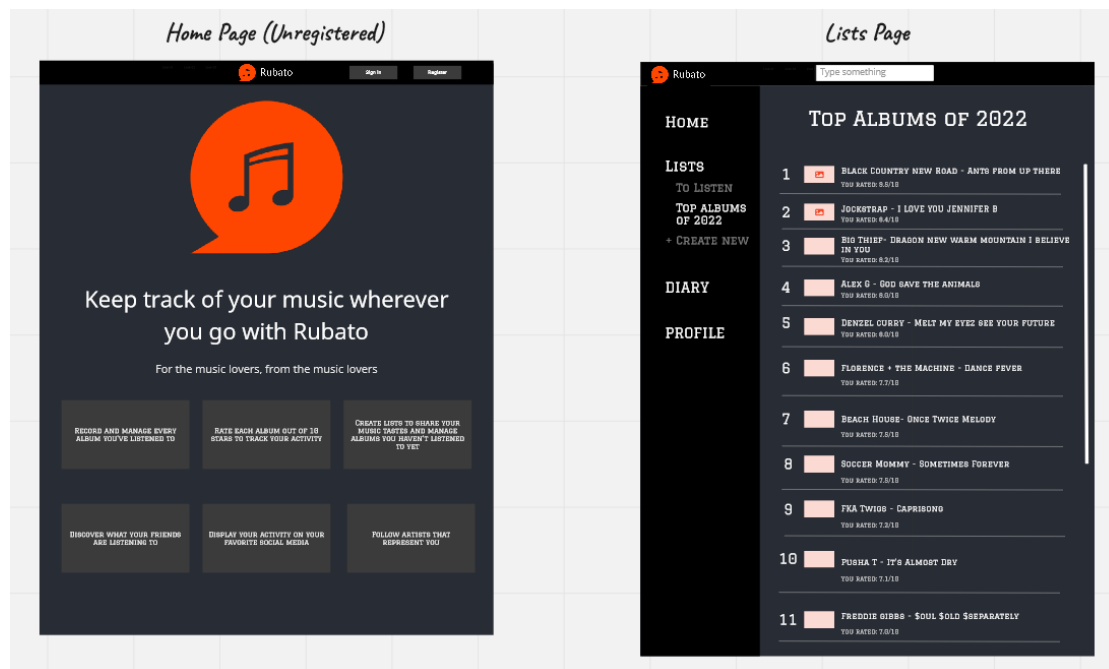


Figure 2.9: Home and Lists pages

The next set of storyboard designs correlate to the the *Diary* and *Profile* pages.

The purpose of the *Diary* page is to allow the user to see their past activity, whether that's an album they recently reviewed, or an album in which they added to a list. This organizes and provides the user with a time-stamp, giving a complete history to their listening habits.

The *Profile* page is then used as place where the user can see their specific details and all albums in which they reviewed thus far on their page. This can also be used for if other users want to see your activity.

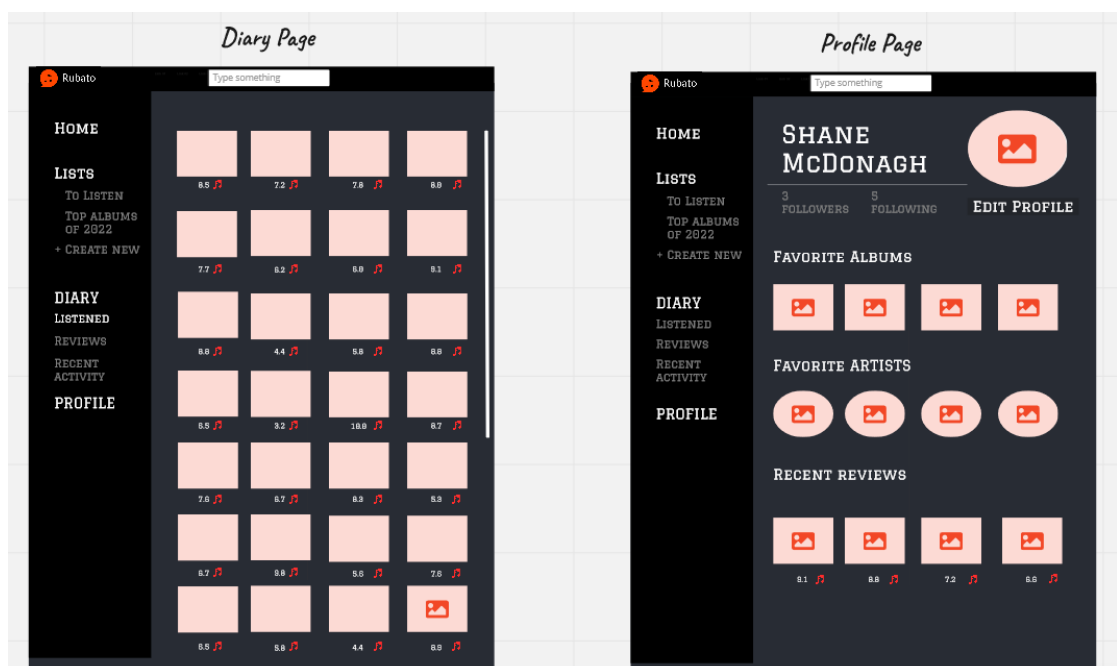


Figure 2.10: Diary and Profile pages

Although the end-result is different from the initial concepts that we have discussed here, getting the wire-frames designed was helpful in providing a foundation for us to start off with.

This also played a role in the approach I took in implementing features in an incremental and iterative way, by following the Agile principles that are a part of the Scrum framework.

## 2.5 Testing

Since the project in question is a web application, and some web testing was done in relation to my current studies, the choice of using *Selenium* felt like a perfect choice for automated web testing.

Selenium is described as a tool for "automating web applications for testing purposes" [10]. By simply recording the interactions and playing them back through the Selenium IDE, it was straightforward to create test suites for the application to fully test each individual page of the application. This helped me continue to develop and improve as more steps were taken in getting the application to its goal.

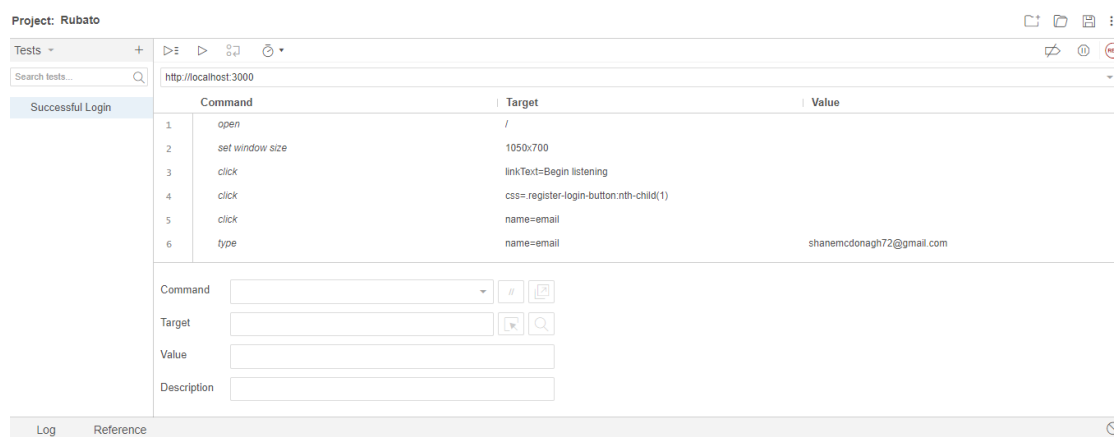


Figure 2.11: Example of a Selenium Test

Another useful tool to ensure that specific endpoints of our application were returning the necessary information, was *Postman*.

Postman works as an API platform, which allows for us to test APIs [11]. This becomes extremely helpful to determine quickly what data is being passed to the client from the server, and if the correct information is being retrieved at any given time.

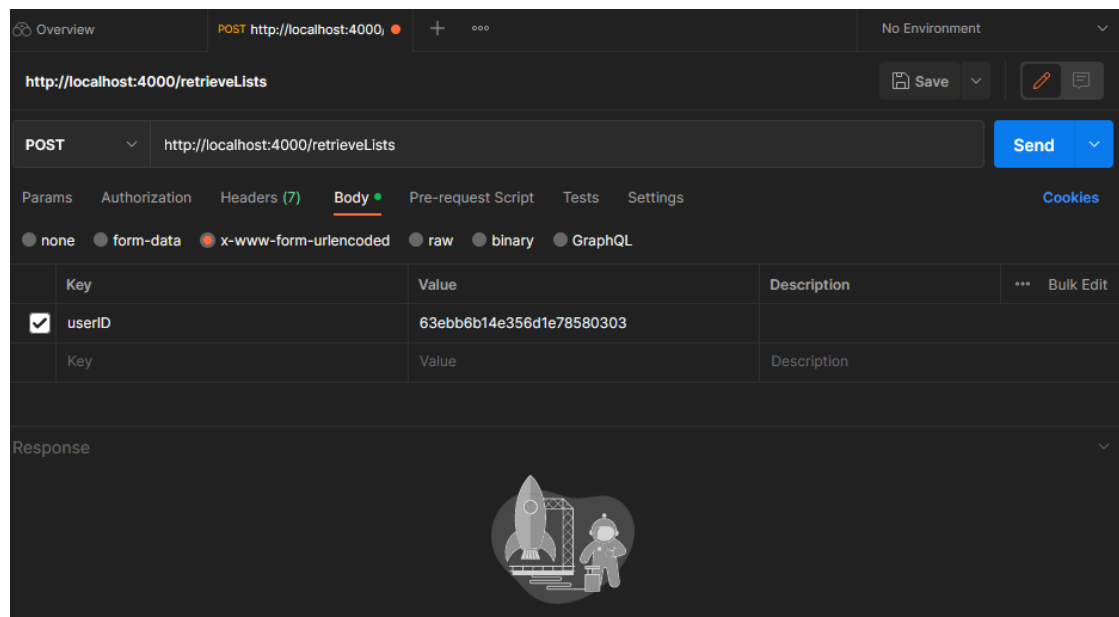


Figure 2.12: Example of a Postman PUT request

The essential aspect of this tool in relation to testing is the ability to test both the database and the API in which we will be using for this application. The application of these tools will be seen in a later chapter.

### 2.5.1 Handling of problems

The handling of problems in relation to any particular faults discovered from the automated testing or from working on the project, were noted using Jira. Jira acts as the software in which was used for the Kan-ban board, which allowed for tracking and noting features in which I needed to pay particular attention to.

These were also noted to my supervisor while discussing the current progress of the project within the weekly meetings, helping to understand where priorities with issues should be.

Projects / Rubato

Issues

Share Export issues Go to advanced search LIST VIEW DETAIL VIEW

Search issues Project: Rubato Type Status Assignee More + Save filter BASIC JQL

Type	Key	Summary	Assignee	Reporter	P	
🔖	RUB-30	Add separate album page with song details etc.	👤 Unassigned	👤 Shane McDonagh	=	🔖
🔖	RUB-29	Add rating system to albums	👤 Unassigned	👤 Shane McDonagh	=	🔖
🔖	RUB-28	Retrieve data from API, display within application	👤 Unassigned	👤 Shane McDonagh	=	🔖
🔖	RUB-27	Connect to Spotify API	👤 Unassigned	👤 Shane McDonagh	=	🔖
🔖	RUB-26	As a user, I want to be able to see various album on the home page	👤 Unassigned	👤 Shane McDonagh	=	🔖

Figure 2.13: Issue board of the Jira project

# Chapter 3

## Technology Review

Within this chapter, I discuss in great detail, the technologies that were selected for allowing us to build the web application as needed. This is where the necessary research methodology that was discussed earlier was applied. This is to help understand the advantages and limitations each tool had in the overall scheme of the end result.

The following are the technologies that were chosen:

- HTML, CSS, and JavaScript (*Languages*)
- MongoDB (*Database*)
- Express (*Web Application Framework*)
- React (*Front-end JavaScript Library*)
- Node.js (*Server Environment*)
- Heroku (*Cloud-Hosting Service*)

Other technologies will be discussed and reviewed as well, which give assistance to the major technologies to reach their full potential.



## 3.1 Languages

When discussing any software project, the usual question that follows is what languages were used to achieve the ending result. Since this project is a web-based application, there were a few options which could be used, however there are specific languages which are considered staples of web development. These are the languages that will be discussed and explored here.

The following are the languages in which initially helped build a good basis for the application.

### 3.1.1 HTML

HTML (*Hyper Text Markup Language*) is considered to be the basic foundation of all things Web-related. HTML describes the general structure of a given web page. Other technologies (which we will discuss further) allow us to style and add functionality to HTML pages.

Hypertext allows us to connect web pages through links, which is considered a fundamental aspect of web development [12].

Through the use of markup, we can annotate text and other content to display in any given browser.

The way in which we can manipulate the HTML is due to a model which can represent a HTML document and allow us to interact with it, which is the *DOM model*.

## DOM

DOM (*Document Object Model*), is a neutral and language independent interface, which allows programs and scripts to dynamically access and update the content, style, and structure of documents [13].

DOM breaks the document down into a tree-like structure, allowing us complete control on every single aspect of the document. This gives freedom in styling and updating the document in the exact way in which the developer wishes.

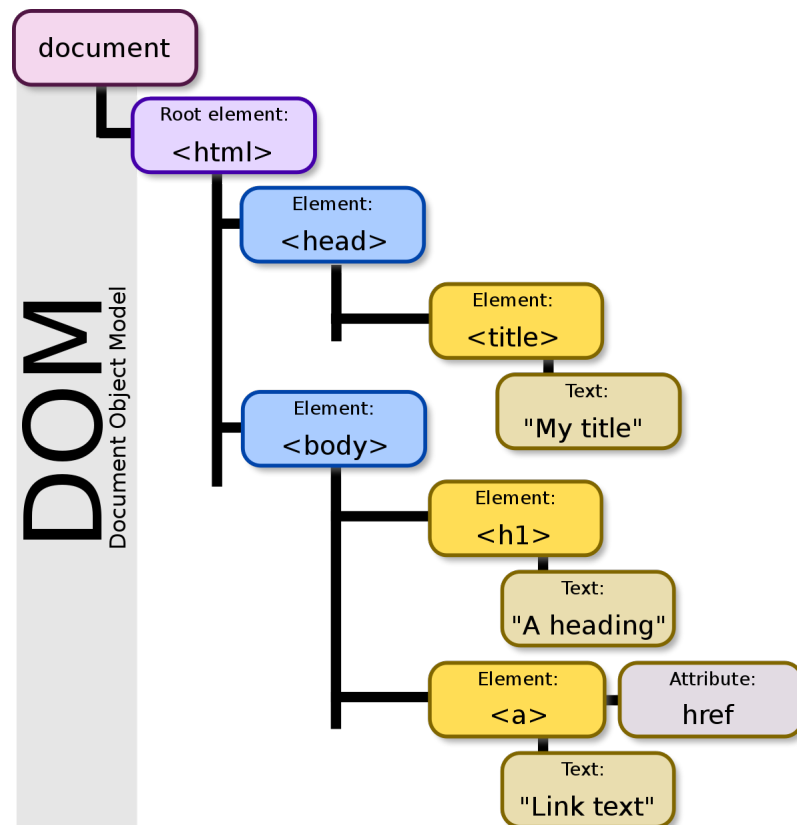


Figure 3.1: DOM Tree Structure

Whenever a given web browser parses a HTML document, it creates a DOM tree and displays the document in this fashion [14]. This is where the other technologies come into fruition, to help us manipulate this DOM tree as we see fit.

### 3.1.2 CSS

To allow us to style the DOM tree in the way in which we would like, we use the *CSS* technology.

CSS (*Cascading Style Sheets*) is a style-sheet language, which allows us to describe the presentation of a HTML document.

By using CSS, a lackluster web-page can become much more visually appealing. This allows us to create an exciting experience for the user in the way in which we want to [15].

For example, the below figure allows us to style every single paragraph within a document with the color red. This significantly reduces time spent on styling and allows for more control.

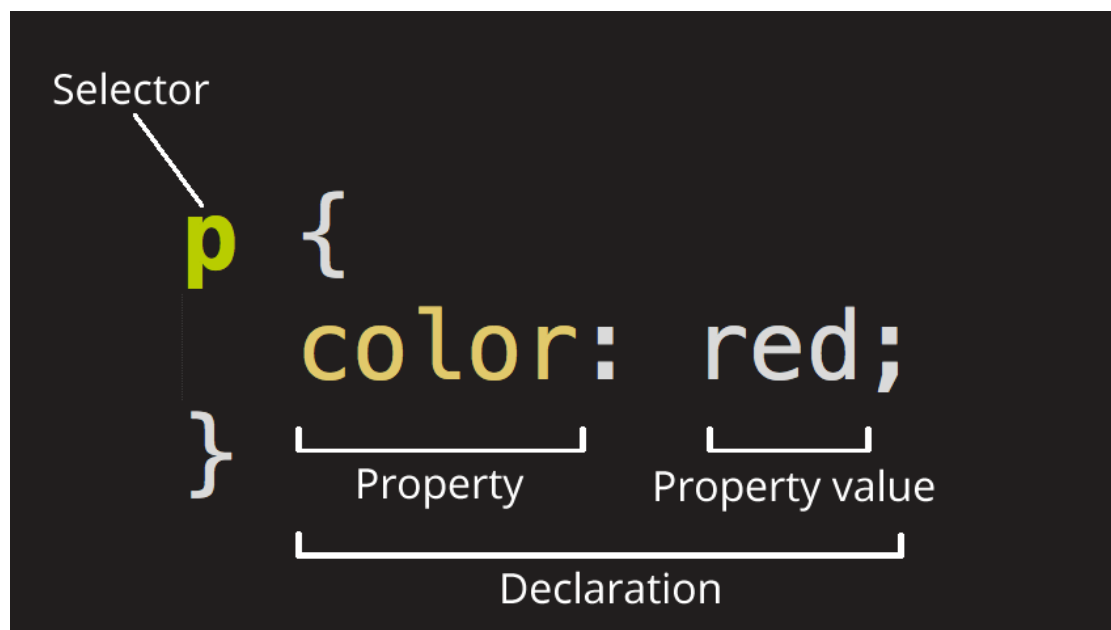


Figure 3.2: Example of CSS

Using CSS was a no-brainer in this context, due to the fact that it is used with HTML in so many instances and applications.

Although CSS is extremely useful, we can also use another framework to help in adding pre-built stylised components to our application in a faster manner.

## Bootstrap

To give users a satisfying experience in using a web application, it is vastly important to be able to offer a visually appealing experience. Although we have discussed CSS, styling every single aspect of each document would be extremely tedious and time consuming. This is where *Bootstrap* comes in.

Bootstrap is a front-end toolkit, allowing us to implement production-ready components quickly. This also gives us control in styling the components in the manner in which we need, allowing us to avoid building elements from the ground up [16]. This saves much needed time in development time.

Whether installed directly by a package manager, or using a CDN link within the document, implementing Bootstrap elements is quite straightforward to use, which made it an easy decision to make.

Here is an example of how to implement a simple Card component from Bootstraps library into a HTML document:

Listing 3.1: Bootstrap example

```
<div class="card" style="width:_18rem;">
  
  <div class="card-body">
    <p class="card-text">Some quick example text to
      build on the card title and make up the bulk of
      the card's content.</p>
  </div>
</div>
```

As you can see, implementing specific classes to HTML elements becomes quite a straight-forward way to customize the appearance of pages in an easier manner. Bootstrap offers many differing components which comes in extremely handy.

We can see the code that was displayed and specified above lead to the following visual:

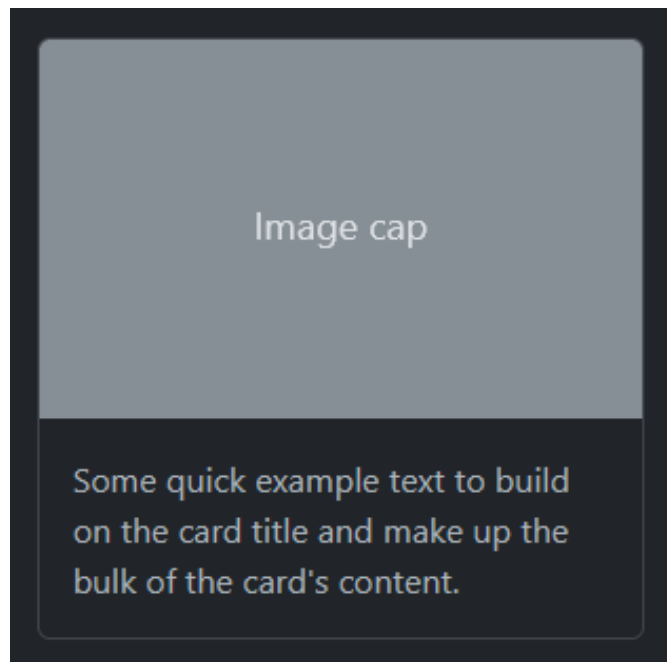


Figure 3.3: Example of Bootstrap card

Using Bootstrap and CSS to add finer details becomes a good combination in order to allow us to get the exact specifications we are looking for in order to complete the goals of the project in the nicest way possible. Especially in relation to a dynamic website, the benefits of this is extremely important to take advantage of/

### 3.1.3 JavaScript

No styling or visual components are actually useful if we cannot interact with the web-page itself in any manner. This is where *JavaScript* comes into play.

JavaScript is a programming language (also known as a scripting language) in which allows us to add functionality to a website, alongside the ability to dynamically update content as needed [17]. Without a scripting language, the website doesn't actually have any type of functionality.

Alongside HTML and CSS, JavaScript is considered a core technology of the web. JavaScript conforms to the *ECMA* standards, which follows a dynamic type system. This verifies type safety at run-time [18]. Dynamic type systems allow for much for flexibility in comparison to its counter-parts, which is really important for a website such as this.

An example of simple JavaScript would look like the following:

Listing 3.2: JavaScript example

```
<!DOCTYPE html>
<html>
<head>
  <title>
    JavaScript Example
  </title>
</head>
<body>
  <button onclick="alert ( ' Hello ! ' ) ">
    Click me
  </button>
</body>
</html>
```

To explain the following code, the **on-click** attribute calls a function with the argument *"Hello!"*. Although this seems pretty simplistic, JavaScript is quite powerful in helping us manipulate the functionality set for a web page. This makes the usage of JavaScript, alongside HTML and CSS, a must in what the project as a whole needs.

## 3.2 MERN

When researching efficient and popular ways to approach developing a web application, a specific term was consistently mentioned and seen, even playing a role in my education thus far. This term was *MERN*.

According to the MongoDB documentation [19], a MERN stack is a variation of a MEAN stack (MongoDB Express Angular Node), where the Angular front-end web framework is replaced with React.

A stack is a collection of independent components which work in tandem to help execute an application. By working with a solution stack such as MERN, it helps to create a complete platform without the need of using further tools or technologies. Other libraries can help enhance the experience and overall work, however it isn't necessary to get a fully functioning application.

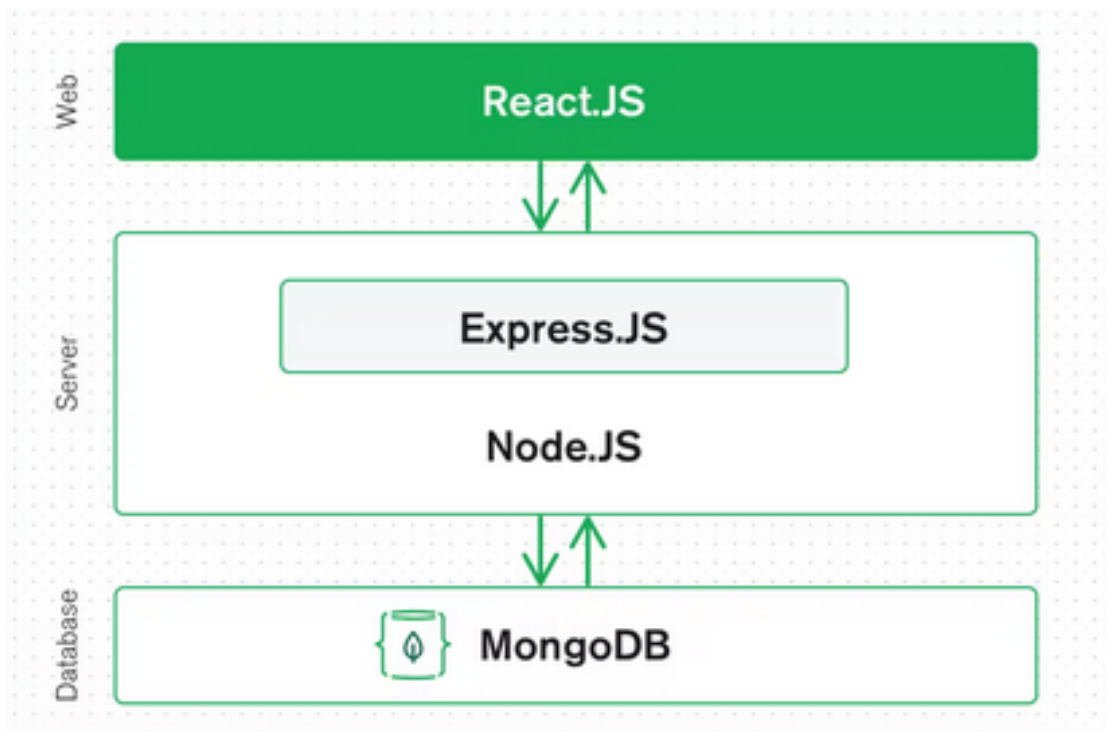


Figure 3.4: MERN application

To understand each component, let's break down each of the technologies.

### 3.2.1 MongoDB

MongoDB is a document based database, which is heavily used within the industry currently and is one of the most used. With the scalability and flexibility that MongoDB offers, querying and indexing becomes a much easier task.

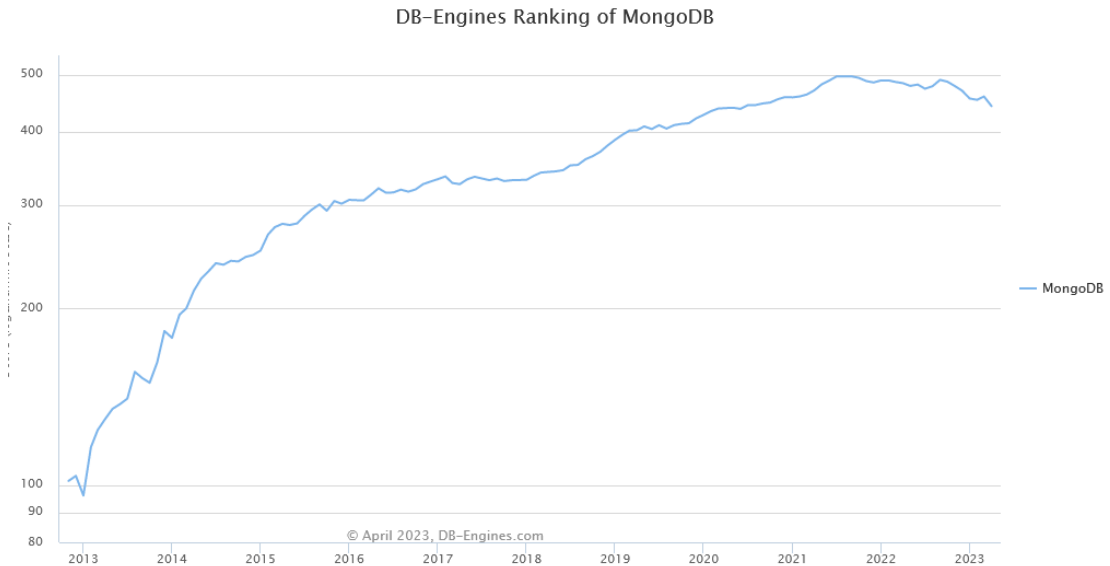


Figure 3.5: MongoDB popularity over time

MongoDB can be described as having a **NoSQL** database model (umbrella term), where data is stored in a different manner than traditional relational databases like *MySQL* etc.

NoSQL databases were introduced as an alternative to relational databases. Although relational databases (tightly coupled data) are still used, their performance says a downgrade when dealing with the increase in data, making it less scale-able, which proves to be less than ideal for web services which may have thousands to millions of users [20].

Why is it commonly used within software stacks and how does NoSQL databases handle web services? To understand the main differences, the following table showcases how they handle certain aspects [21]:

MongoDB has other advantages in this instance as well, which is vital for this particular project. Due to the structure of the data (being human-readable), it is incredibly suitable for agile development, which is the approach I am taking.



Features	Relational	NoSQL
Schema	Strict, cannot change and all rows in a specific collection must conform to the same schema	Flexible, documents in the same collection do not need to share the same exact structure with their fields and the data type for a field can differ within the same collection
Scaling	Benefits vertical scaling more (increase processing power), doesn't handle horizontal scaling as well due to breaking up related data	Handles horizontal scaling extremely well (horizontal scaling architecture) since data is loosely coupled
Load Times	Can be slow in the increase of data	Due to the data model, the load times are fast, important in a web application
Ease of use	Becomes increasingly more complex with scalability	No need to conform to specific data structures, making creating documents relatively straightforward

Table 3.1: NoSQL vs Relational Databases

Another important aspect is documents (or records in traditional database terms), are stored in BSON files, which can be retrieved in JSON. [22]

## JSON

JSON (JavaScript Object Notation) is known as a "light-weight data interchange format", which resembles JavaScript object literal format, the language in which I am using [23].

JSON, despite its resemblance to JavaScript object formats, are language independent and is widely used within the industry. The lightweight nature of JSON allows for data to be passed quickly and in low volume in relation to size, being a commonly used way to send data from a server to a client, given the ability to parse the information and structuring it to objects where necessary [24].

At a granular level, JSON consists of these data types [25]:

- String
- Number
- Boolean
- Null
- Object
- Array

Due to the flexible nature of JSON, it allows for handling data in a much more manageable way.

JSONs format can look like the following:

Listing 3.3: JSON example

```
{
  "squadName": "Super_hero_squad",
  "homeTown": "Metro_City",
  "formed": 2016,
  "secretBase": "Super_tower",
  "active": true,
  "members": [
    {
      "name": "Molecule_Man",
      "age": 29,
      "secretIdentity": "Dan_Jukes",
      "powers": ["Radiation_resistance"]
    },
    {
      "name": "Madame_Uppercut",
      "age": 39,
      "secretIdentity": "Jane_Wilson",
      "powers": [
        "Million_tonne_punch",
        "Damage_resistance",
        "Superhuman_reflexes"
      ]
    }
  ]
}
```

With the advantages of JSON for web applications and MongoDB as a whole, it makes it easy to understand data storage and handling, which makes sense as to why it is within the stack that I am currently using.

### 3.2.2 Express

Express is known and described as a Node.js web application framework, which allows us to use a myriad of features to create RESTful APIs in a more efficient manner. [26].

Express is seen as the de-facto standard server framework in relation to Node.js, which we will discuss further due to its place within the stack itself.

Express uses *routing*, which allows us to define how our server handles specific client requests. This is achieved by using the methods available within Express, relating to HTTP methods like:

- GET: Requests a specified resource, only retrieves data
- POST: Submits data to a specified resource, usually causing a state change or side effect on server
- PUT: Replaces representations of specified resources with the request data payload
- DELETE: Deletes the specified resource

Essentially this means that our back-end listens for requests made to specified routes and methods, where then the appropriate callback function is called.

Here is a basic implementation of an Express route [27] :

```
const express = require('express')
const app = express()

// Respond with "hello" when GET request is made
// to homepage
app.get('/', (req, res) => {
  res.send('hello')
})
```

## REST

REST (or Representational State Transfer), is a common web architectural style with specified principles. This allows flexibility in relation to the client-server relationship, making it platform independent [28].

The constraints placed using the REST principles allows to build scale-able, maintainable, and straightforward frameworks.

The principles are as follows [29]:

- Client-server: The client and server must be completely independent of one another, which improves portability across different platforms and improves scale-ability by simplifying the server-side
- Stateless: Each request should contain all the information necessary to fulfill the request, without using any stored information on the server side, making it so session state is entirely on the client-side
- Cache-ability: Resources on both the client and server side should be cache-able, which allows increased performance on the client-side and increased scale-ability on the server-side.
- Uniform Interface: All requests, no matter their origin, should look the same. This means requests are done in a standardized form
- Layered System: Requests and responses may pass through multiple layers (could have multiple intermediaries). Therefore, the client nor the server should be able to tell if they are communicating directly to one another or through an intermediary

By using the first two technologies within the stack currently we'll be able to connect to our MongoDB database and retrieve/create documents based on client requests.

This particular part of the MERN stack is incredibly important for another reason. Due to the fact that we need to display music albums for the user, it is important for us to be able to retrieve and display this information to the client when necessary. This is where another important piece of technology comes into play.

## Spotify API

Spotify API allows us to interact with Spotify's streaming service, giving us access to album metadata, recommendations, playlists etc. Since Spotify is considered a RESTful API, it also adheres to the principles in which Express does. This means we can retrieve necessary data from Spotify by simply making client requests to our Express server (specifying a route and a method), and have the server request that information from Spotify and pass back the specified resource.

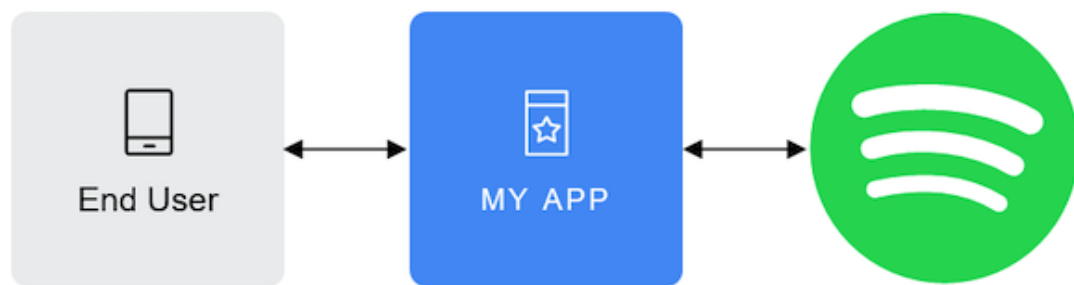


Figure 3.6: Spotify OAuth 2.0 authorization framework

For example, here is a sample of a response when a request is made to <https://api.spotify.com/v1/recommendations/available-genre-seeds> by the Express server:

```
{  "genres": ["acoustic", "afrobeat", "alt-rock",  
"alternative", "ambient", "anime", "black-metal",  
"bluegrass", "blues", "bossanova", "brazil",  
"breakbeat", "british", "cantopop", "chicago-house",  
"children", "chill", "classical", "club", "comedy",  
"country", "dance", "dancehall", "death-metal",  
"deep-house", "detroit-techno"]}
```

Since the response is in JSON, we can easily parse it and arrange the data retrieve in any way we prefer. The way this is done is by [30]:

- 1. Creating an app within the Spotify for Developers website, giving us access to the *Client ID* and *Client Secret* token for authorization process
- 2. Retrieve and request an *access token* which contains the credentials and permissions that can be used to access a given resource
- 3. Use the access token to query the Spotify API

The authorization process allows for four types of flows to request an access token:

- Authorization code
- Authorization code with PK-CE extension
- Client credentials
- Implicit grant

Flow	Access User Resources	Requires Secret Key (Server-Side)	Access Token Refresh
Authorization code	Yes	Yes	Yes
Authorization code with PK-CE extension	Yes	No	Yes
Client credentials	No	Yes	No
Implicit grant	Yes	No	No

Table 3.2: Flow Behaviour

Since the web application that I am developing does not need specific user authorization, the client credentials is perfect for our app in question.

The client credentials flow works as follows:

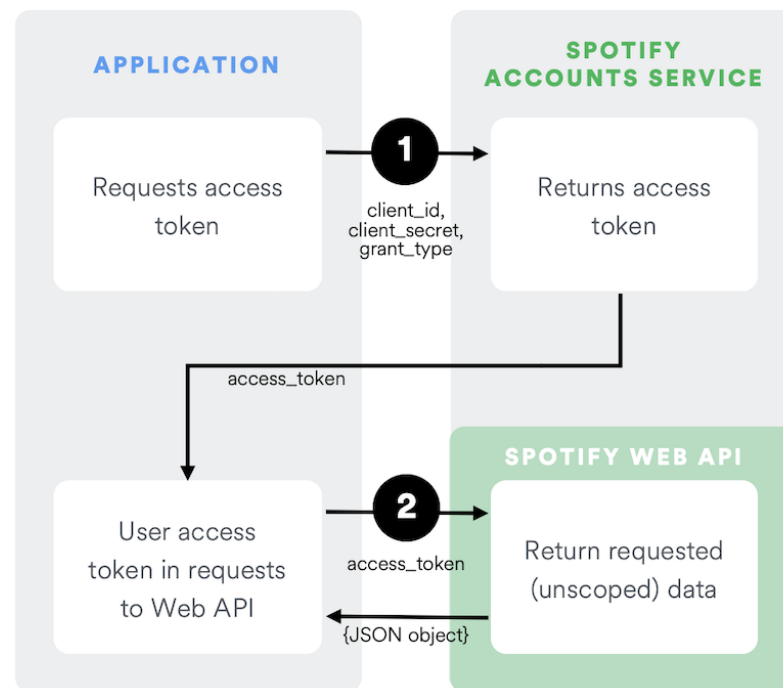


Figure 3.7: Client Credentials Flow

- 1. The server requests an *access token*, based on our *Client ID* and *Client Secret* keys
- 2. Client makes request to specific URI
- 3. Server uses access token to retrieve specified resource from Spotify Web API
- 4. API returns necessary information to the server, which is then passed to the client



### 3.2.3 React

React acts as our front-end, being a JavaScript library for building user interfaces [31]. React allows for the ability to create components (a collection of smaller units) which allows them to be re-usable and nest-able.

Components act like classes in a more traditional sense. React supports the concept of *Single Page Applications* (SPA), where a component can be individually swapped out or updated fully independently, without the need for a page refresh on each action made by a user. This allows to decrease load times and bandwidth as well [32].

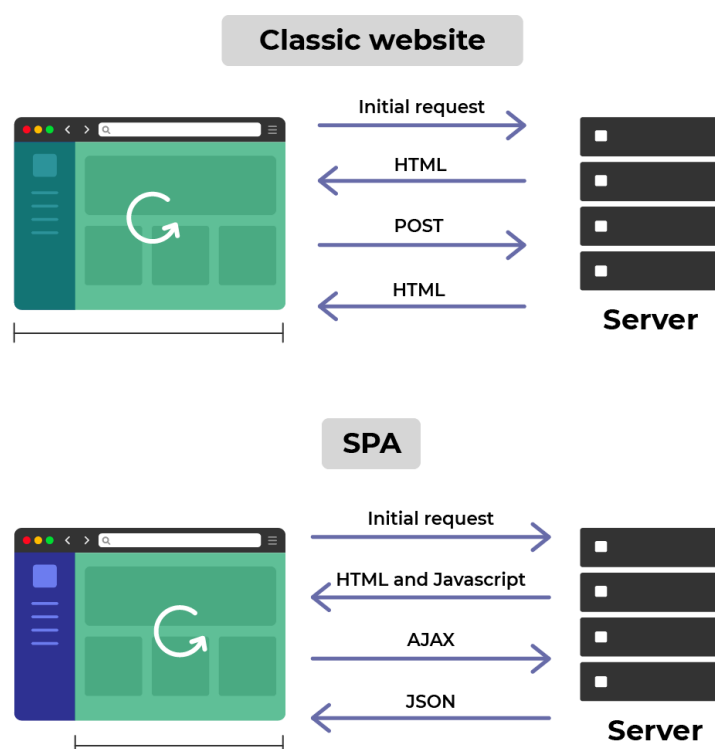


Figure 3.8: SPA vs MPA

React is becoming a well-known and popular front-end framework to use in the industry currently, and for good reason. The main reason why Single Page applications can perform in such a way is by using *AJAX*.

## AJAX

AJAX (Asynchronous JavaScript and XML) is not necessarily a technology, but is a technique in using multiple technologies in tandem (e.g. HTML, JavaScript, XMLHttpRequest etc.).

This allows for the ability to make updates to the web application without having the need to reload the entire browser page, working asynchronously [33].

Although the 'X' does stand for XML, JSON is a more commonly used format in sending information back to the client.

We can define it as follows [34]:

- Standards-based presentation using XHTML and CSS
- Dynamic display and interaction using DOM
- Data interchange and manipulation using XML (in this case, JSON)
- Asynchronous data retrieval using XMLHttpRequest
- JavaScript, to use them altogether

By using all of these technologies together, it becomes a powerful tool to allow the application to be more responsive to user action.

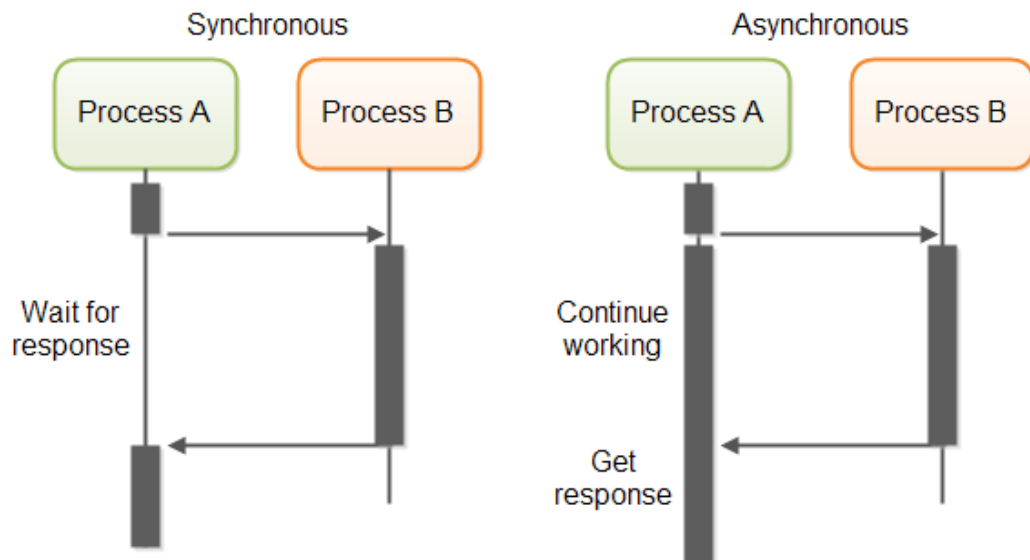


Figure 3.9: Synchronous vs. Asynchronous

We can see the main differences between a synchronous web application and an asynchronous web application from the ?? figure above.

Asynchronous web applications allow for the ability for the user to continue using the application and avail of different features, while in the background the application continues to work at specific tasks.

By using React, we can take full advantage of a more seamless experience for the user.

### 3.2.4 Node.js

The last technology within the stack, *Node*, acts as the back-end platform. While we have discussed Express, Node acts as the underlying layer which allows us to write server-side code in JavaScript [35].

Node allows us to create a way to communicate to our front-end by building a server that can handle many requests at once.

We can write a simple web server with Node.js by doing the following [36]:

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(port, hostname, () => {
  console.log('Server running at\nhttp://${hostname}:${port}/');
});
```

By simply running the server and navigating to the relevant url, we'll see "Hello world" displaying to us, showing the server is working.

For this specific project, Node.js and Express allow us to retrieve the necessary data we need in relation to albums (Spotify API) and user information (MongoDB).

On top of Node.js, another technology is needed to get the necessary drivers for our server to be able to achieve everything it needs to.

## Node Package Manager

Since we need many dependencies in order to create the application we strive for, a way is needed to install all the technologies we have discussed so far in order to work alongside the server. This is where NPM is useful.

NPM acts as our package manager, giving us access to thousands of packages listed in its registry [37].

NPM even allows us to install front-end packages (like React), as well. NPM is usually interacted with through the CLI (Command Line Interface).

The way that NPM works is as follows [38]:

- 1. Install Node.js, which includes NPM. This allows us to interact with the CLI
- 2. Create a *package.json* file within the project, which is used to contain metadata on any package dependencies of the project
- 3. Install necessary packages by running ***npm install***, alongside the package name (packages found through browsing the registry). This is then added to the *package.json* file. Any dependencies a package requires also gets installed

By using NPM alongside Node, it helps us spend less time dealing with libraries and more time on the software.

### 3.3 Heroku

The most important part of any web application is the ability to be able to actually visit the web page itself and avail of the features the website has.

All the technologies thus far allow for us to create a web application. However, none of these actually deploy the application to the web. This is where *Heroku* comes in.

Heroku is a cloud platform as a service (PaaS) [39] which supports the ability to host applications entirely in the cloud [40].

This may seem like a daunting task, to deploy a web application. However, this is actually a relatively straight forward process when well associated with the application.

By simply giving Heroku access to a GitHub repository containing the project and changing a few variables (which allows the server to build both the back-end and front-end), the application becomes deploy-able.

To begin using Heroku, a verified account must be used in order to handle billings and subscriptions, however this is a relatively simple process as well.

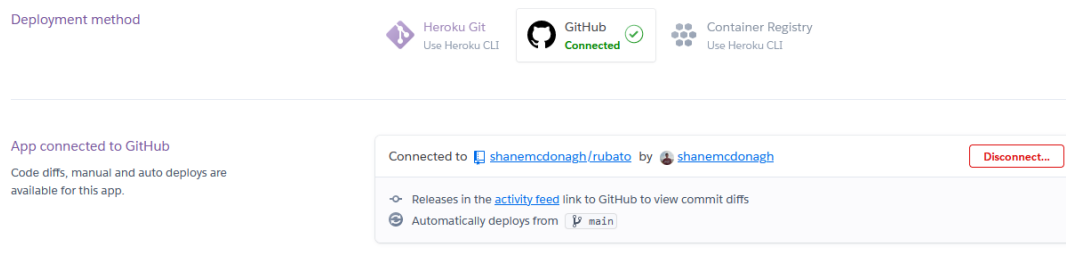


Figure 3.10: Deploying the repository as a cloud application

# Chapter 4

## System Design

In this chapter, the design of the web application will be examined in more detail. This is important to be able to fully comprehend and understand how the project works. All information within this chapter is the application of the technologies that were discussed in the previous chapter.

### 4.1 System Architecture

The system architecture can be visually represented by the following diagram:

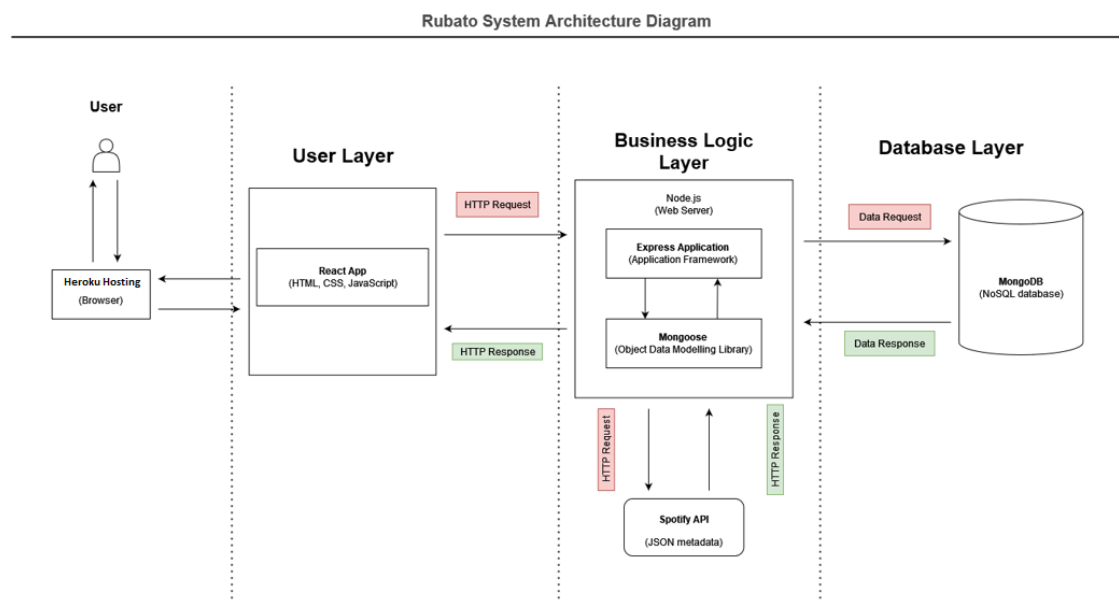


Figure 4.1: System Architecture Design

The systems steps can be broken down as so:

- 1. The user re-directs to the given web-page that the application is hosted
- 2. The hosting platform serves the React application to the user, allowing access to the features built into the website
- 3. Any requests made by the client on behalf of the user is sent to the back-end of the application
- 4. Depending on the type of request, the server will either make a request to the Spotify API, or the database
- 5. Once the server retrieves the necessary information, it then responds back to the client with the specified resource
- 6. The application then responds appropriately with the retrieved resource by updating the behaviour of the web-page in some manner, without reloading the entire page
- 7. The changes, if any, are reflected to the user

## 4.2 Front-End

As discussed before, our front-end uses the *React* framework. Each individual aspect of the application is broken down into many components, which are then used and nested as necessary in the application.

The way this is handled is by having one main component displayed (the *App.js*) continuously, while then loading in other components based on user behaviour.

Here is an example of what this might look like:

```
<div className='component-view'>
  { /* Switches between the local components */ }
  <Routes>
    <Route path='/' element={ <Home /> } />
    <Route path='/lists' element={ <Lists /> } />
    <Route path='/lists/albums' element={ <ListenListAlbums /> } />
    <Route path='/diary' element={ <Diary /> } />
    <Route path='/profile' element={ <Profile /> } />
    <Route path='/search' element={ <Search /> } />
    <Route path='/album/' element={ <AlbumDetails /> } />
    <Route path='/genre/albums/' element={ <GenreAlbums /> } />
    <Route path='/artists/albums/' element={ <ArtistAlbums /> } />
    <Route path="*" element={ <Navigate to="/" replace /> } />
  </Routes>
</div>
```

Figure 4.2: System Architecture Design

Depending on the path the user decides to visit (by using the sidebar), different components will be loaded in and out where necessary. The components seen in this figure also contains nested components, which provides even more functionality and features.



This allows for a more flexible system. The system provides the ability for us to breakdown complex features into smaller components, providing more of a fine-grained approach.

An example of this can be seen on one of our pages, namely the *Lists* page.

The Lists page acts as a collection of sub-components, all working alongside each other to ensure that the complex features are split and divided where appropriate. The displaying of each individual list acts as a separate component to the one in which displays them as a group and so on.

We can see the structure of the main Lists page as so:

```
<div>
  <ListenLists lists={lists} updateListLength={this.updateListLength} />
  <div className="listButton">
    <Button variant="danger" onClick={this.handleClick}>Create a list</Button>
  </div>
  {listLength < 10 && (
    <div className="playlists">
      <h3 className="playlist-descriptor">See these playlists for inspiration</h3>
      <Playlists />
    </div>
  )}
</div>}}
```

Figure 4.3: Structure of Components

From this figure, one can see that each component handles specific tasks which aren't inherently complex. Each component does handle more than what can be seen here (from deleting lists etc.), however each one has their specific purpose.

This is seen through the entire React application, where each individual page of the application has a myriad of components in which work in tandem to help breakup the complex aspects of them into simpler and manageable parts.

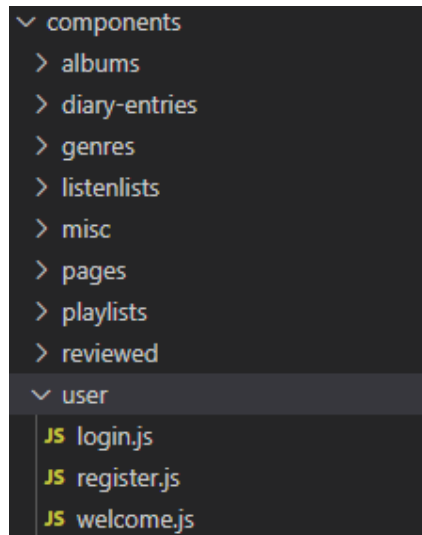


Figure 4.4: Structure of Components

### 4.2.1 User Interface

These elements of the front-end act in tandem to allow for us to give the user a fairly satisfying experience. The styling of specific components (with the help of *Bootstrap* and other libraries) accumulate to a relatively aesthetically pleasing page.

```
.component-view{
  /* Controls the display of components next to the sidebar */
  display: flex;
  justify-content: center;
  align-items: center;
}

.list-item{
  margin-bottom: 1%;
  background: #565656;
  border-radius: 20px;
}

.diary-component{
  margin-left: 20%;
}
```

Figure 4.5: Snippet of our CSS file

By using all the technologies we discussed earlier, we can make a home page like as follows:

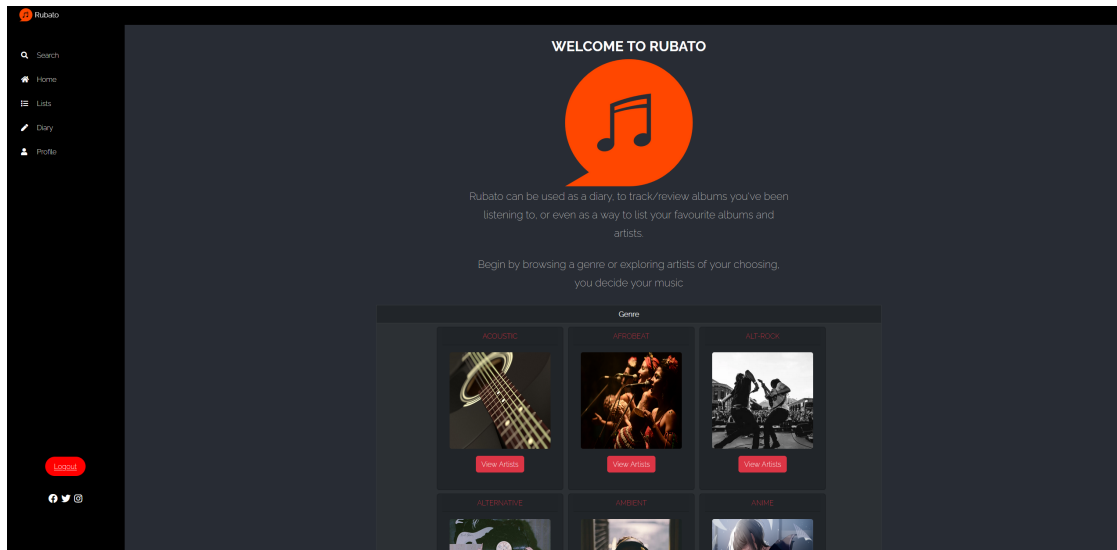


Figure 4.6: Snippet of home page

## 4.3 Back-end

The back-end of our application is laid out in a similar way. The main access point for the server logic is found in our *backend.js* file, which listens for any requests made by the client to allow it to retrieve any specific data that is needed at any given point. This is handled by *Node.js* and *Express*, as we discussed earlier.

```
// Log an error if one occurs when connecting to the database or from the Spotify API
main().catch(err => console.log(err));

// Server begins listening through port 4000, handles requests from port 3000 (our music application)
app.listen(port, (req, res) => {
  console.log(`Listening on port ${port}`);
});

// Listens for a GET request to '/accessToken'
app.get('/accessToken', async (req, res) => {
  res.status(200).json(accessToken);
})
```

Figure 4.7: Snippet of the server file

All specific aspects of the application which handles different features are broken into separate routes for the sake of being able to de-bug the application in a simpler manner and allow for more readability in fully understanding how it operates. All routes can be specifically defined in one file (which I originally had it so), however this isn't good coding convention.

```
const userRoute = require('./routes/User');
const listRoute = require('./routes/List');
const reviewRoute = require('./routes/Review');
const diaryRoute = require('./routes/Diary');
```

Figure 4.8: Snippet of declaring the routes in our main server file

All routes are defined as follows:

- **Diary:** Defines all routes which creates diary entries based on the users album ratings and lists
- **List:** Defines all routes which allow for the user to create and manage lists
- **Review:** Defines all routes which give the user the ability to see and rate albums
- **User:** Defines all routes which give a user the ability to sign-in or register to the application

```
// Listens for a POST request to '/createDiaryEntry'
router.post('/createDiaryEntry', async (req, res) => {

  try
  {
    const diaryEntry = await DiaryEntry.create
    ({
      entry: req.body.diaryEntry,
      userID: req.body.userID
    })

    res.json("Diary entry added");
  }
  catch (error)
  {
    res.json(`Diary Error: ${error}`);
  }
})
```

Figure 4.9: Snippet of the Diary route

All other general functionality exists within the back-end file, acting as the main entry-point for the front-end.

# Chapter 5

## System Evaluation

It is important to be able to reflect and evaluate the system as it has been implemented and designed by testing certain aspects of the application. We discussed the ability to test the application and its routes by using *Selenium* and *Postman*.

### 5.1 Testing

#### 5.1.1 Postman

Before we can test the functionality of the application, it is important that we know that the routes we have specified work as intended. This can be done by ensuring the appropriate data is returned when we navigate to specific routes. We will do so by using *Postman*. The server itself is running locally on port 4000, so all requests will be made to *localhost:4000*.

Here are the following aspects we will test (this is under the pre-condition that the user is logged in):

- User can retrieve a collection of lists in which they have created
- User can view a list of reviews that they have made
- User can see the diary entries, based on their activity

For this, we need to send the `userID`, to allow us to specify which user we want to retrieve the information for. We will first begin by creating these tests within the Postman application itself:

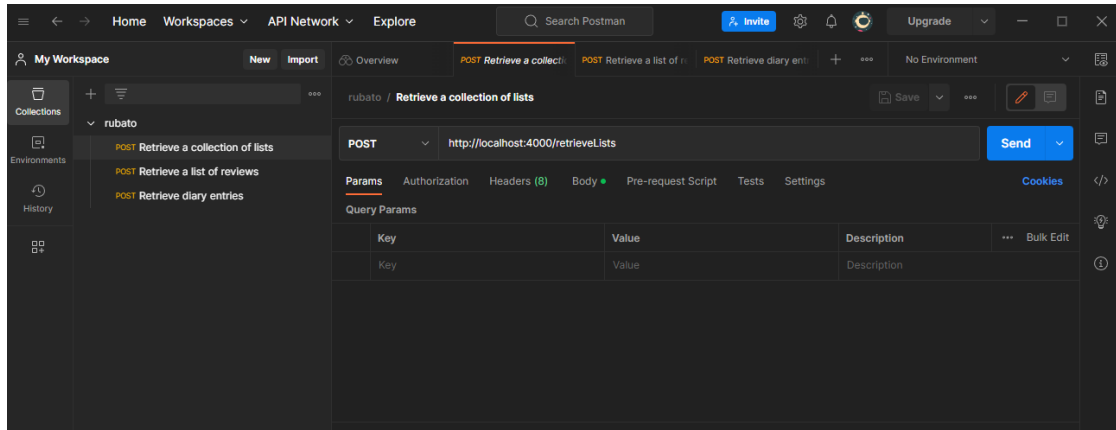


Figure 5.1: Snippet of tests

## Test 1

With our first test, we must specify we are using the correct path for our server: `http://localhost:4000/list/retrieveLists`

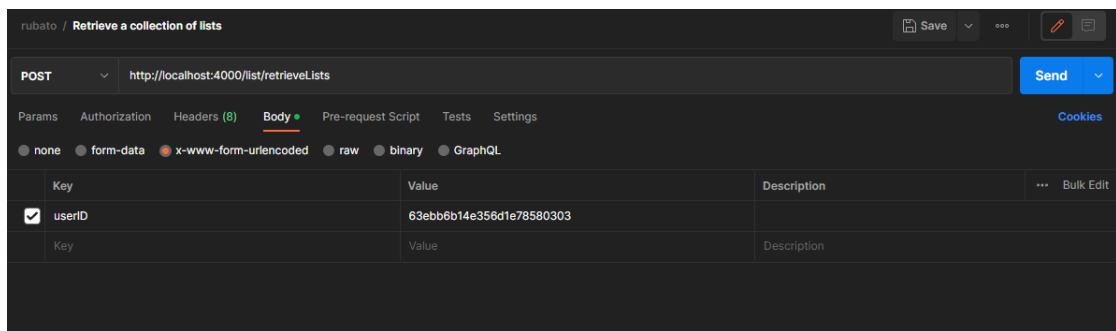


Figure 5.2: Test 1

This should display each individual list that the user has created so far. For testing purposes, lists have been already created within our database for this. The goal is to be able to retrieve all list entries from *MongoDB* and return them back to the client. The reasoning for a POST request is due to the fact that we must send a `userID` alongside these requests, which GET does not allow for.



The result is as follows:

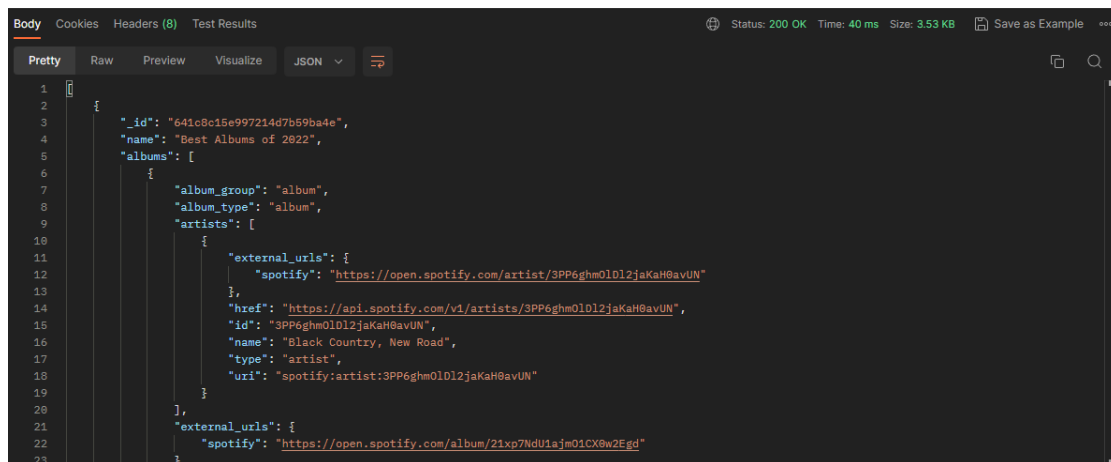


Figure 5.3: Test 1: Result

The response shows us all the lists related to the specified user, alongside all the albums in which are within those lists as well. We can determine that this test passes and the functionality is sound.

## Test 2

The second test determines if we can retrieve all the reviews that have been logged by the user from the database. Again, for testing purposes, reviews have been created to determine if the functionality is working as intended. We can see the test as follows:

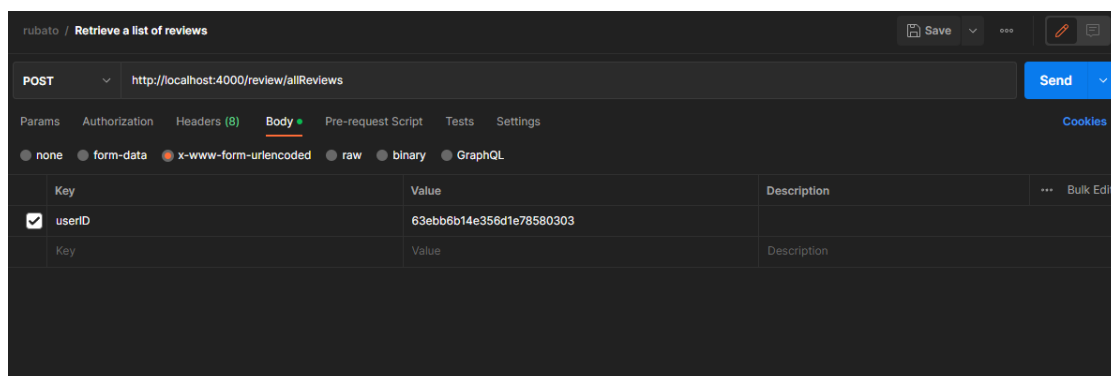
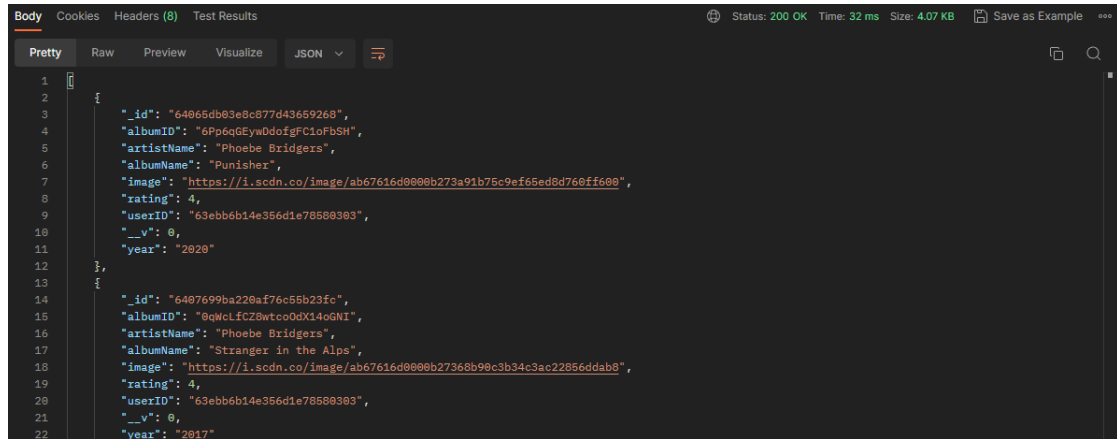


Figure 5.4: Test 2

By making a request to `http://localhost:4000/review/allReviews`, this should return an array of objects, which relates to all the albums that been reviewed thus far in relation to the currently logged-in user. The results are as follows:



```
Body Cookies Headers (8) Test Results
Status: 200 OK Time: 32 ms Size: 4.07 KB Save as Example
Pretty Raw Preview Visualize JSON
1 {
2   "_id": "64065db83e8c877d43659268",
3   "albumID": "6Pp6qGEyWdDofgFCioFbSH",
4   "artistName": "Phoebe Bridgers",
5   "albumName": "Punisher",
6   "image": "https://i.scdn.co/image/ab67616d0000b273a91b75c9ef65ed8d766ff600",
7   "rating": 4,
8   "userID": "63ebb6b14e356d1e78580303",
9   "__v": 0,
10  "year": "2020"
11 },
12 {
13   "_id": "6407699ba220af76c95b23fc",
14   "albumID": "9QWcLfcZ8wtcpOdX14oGNI",
15   "artistName": "Phoebe Bridgers",
16   "albumName": "Stranger in the Alps",
17   "image": "https://i.scdn.co/image/ab67616d0000b27368b90c3b34c3ac22856ddab8",
18   "rating": 4,
19   "userID": "63ebb6b14e356d1e78580303",
20   "__v": 0,
21   "year": "2017"
22 }
```

Figure 5.5: Test 2: Result

This passes the test successfully since we can see each individual album, and the rating that the user selected for them. We can determine that this functions as intended and is acceptable.

### Test 3

The final test we must perform is ensuring that we can retrieve the diary entries created based on the users activities straight from the database. This is essential for us to view these entries on the *Diary* page of our application. The test can be seen as follows:

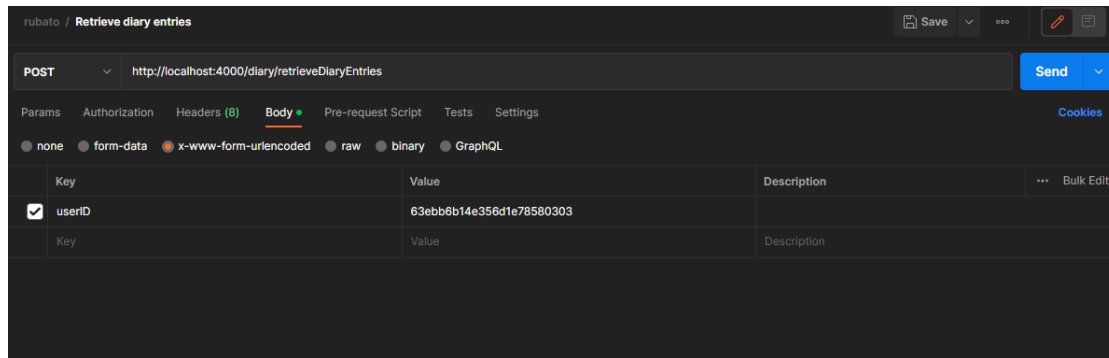


Figure 5.6: Test 3

The results of this test should return an array of objects which represent the diary entries which are created when a user reviews, or adds albums to lists. For testing purposes, a handful of diary entries were created in the database. The following is the results from executing this test:

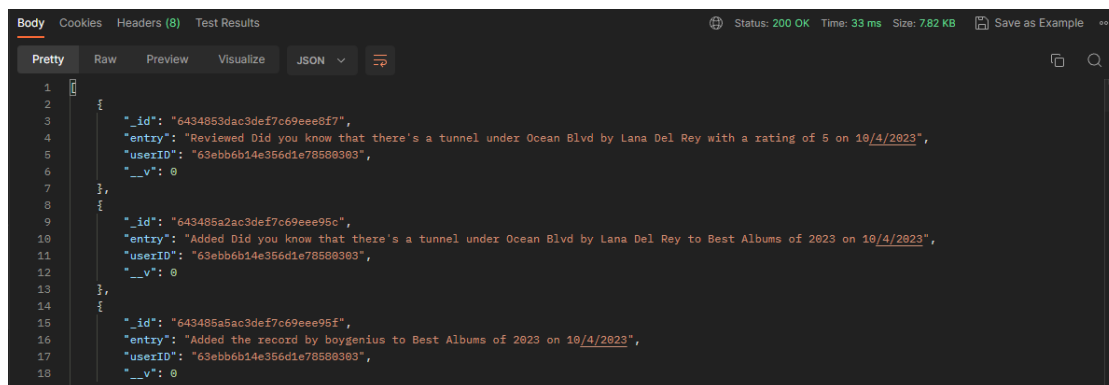


Figure 5.7: Test 3: Result

All tests pass as needed and deliver the client, in this instance, all the necessary data to allow for the application to function as expected. More routes were tested in accordance to the data related to Spotify, however the main functionality that

was important to show working were the routes in which were related to the database itself.

### 5.1.2 Selenium

As mentioned before, we can use Selenium to test the actual behaviour of our website. By testing the appropriate features, we can ensure that the website functions as we expect. To determine the robustness of the application, we can specify some criteria that the application must meet before it is considered passable or acceptable. Since this relates the front-end of the application, the tests will run using the base url *localhost:3000* (the default port which React runs on). Although many different aspects were tested, we will specify just a few acceptance tests that the application must pass to show its functionality:

- User can register or log-in to the application
- User can search for an artist and see their albums being displayed
- User can create lists
- User can listen to a song snippet

We can see these tests within the Selenium IDE like so:

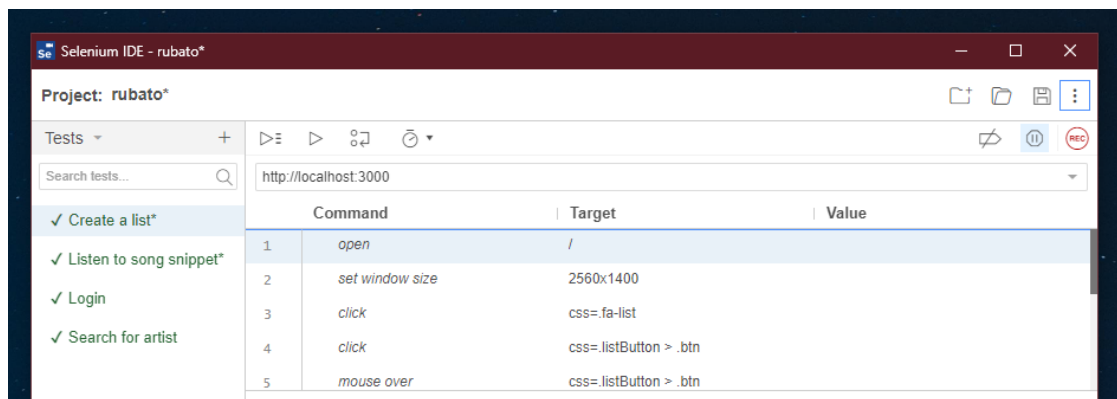
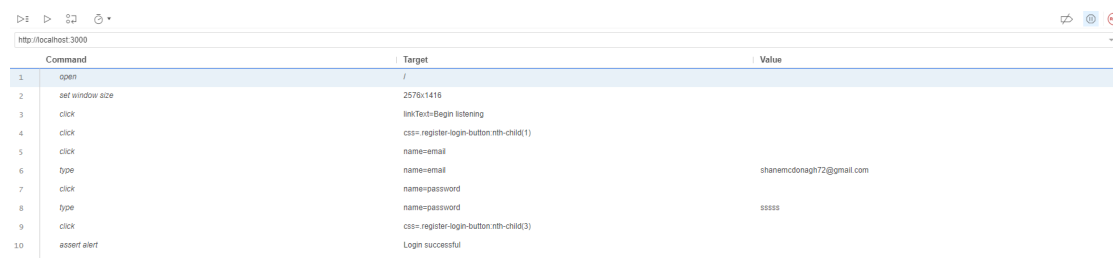


Figure 5.8: Selenium Tests

## Test 1

Arguably the most important feature in the entire application is the ability to login or register. Without being able to do either will cause the user to not be able to avail of any of the features in which the application actually provides. For this reason, we must ensure that this functions as necessary. This test can be seen as follows:



	Command	Target	Value
1	open	/	
2	set window size	2576x1416	
3	click	linkText=Begin listening	
4	click	css=register-login-button:nth-child(1)	
5	click	name=email	
6	type	name=email	shanemcdonagh72@gmail.com
7	click	name=password	
8	type	name=password	sssss
9	click	css=register-login-button:nth-child(3)	
10	assert alert	Login successful	

Figure 5.9: Selenium: Test 1

The steps can be broken down as follows:

- 1. Visit the home page of the page
- 2. Click on the *Begin Listening* button, which re-directs to the login page
- 3. Enter the email and password of a valid user
- 4. Click the *Login* button
- 5. Confirm that an alert pops up on the screen, which reads: "*Login successful!*"

Since Selenium runs these tests automatically in a simulated test environment, we can tell the results straightway by running each one. Each step of the test passed successfully, confirming to us that the step completed without any type of faults in this scenario.

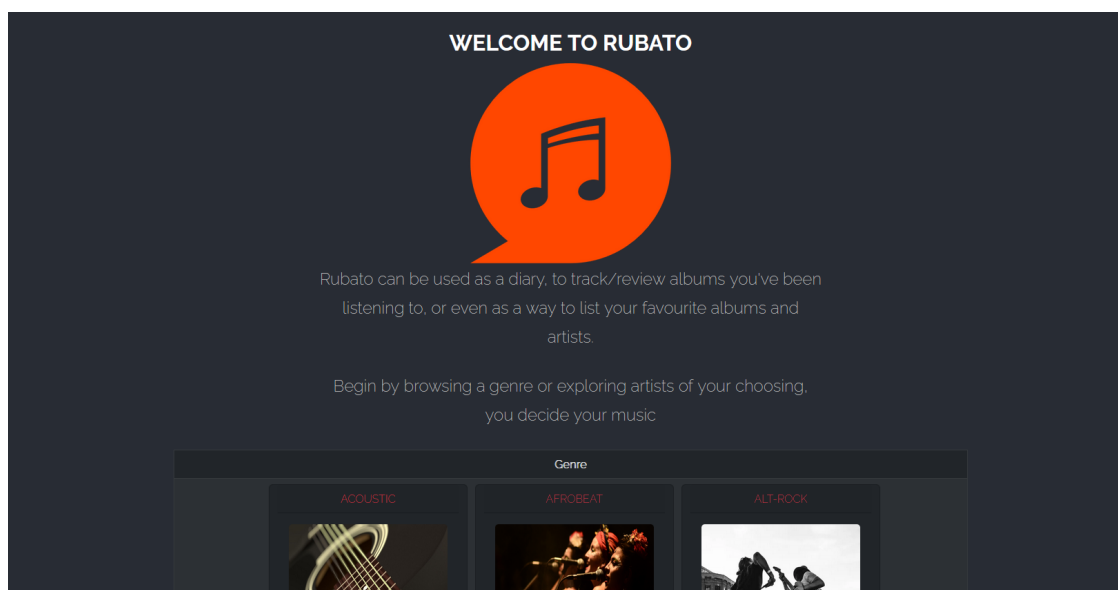
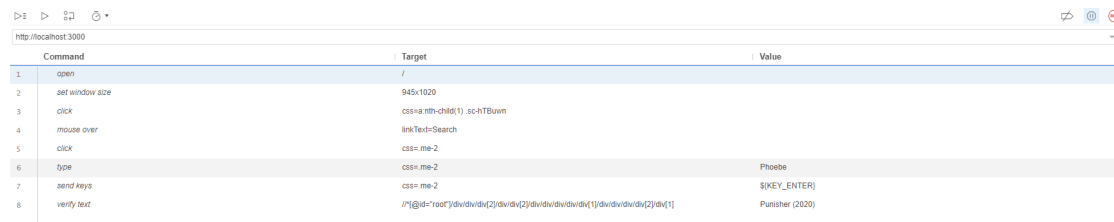


Figure 5.10: Selenium: Test 1 Result

## Test 2

The next test has the precondition that the user is already signed into the application. Once this is confirmed, the test checks to see that the user can successfully search for an artist and see each album from the specified artist.



	Command	Target	Value
1	open	/	
2	set window size	945x1020	
3	click	css=a:nth-child(1).sc-rtBuxm	
4	mouse over	linkText=Search	
5	click	css=me-2	
6	type	css=me-2	Phoebe
7	send keys	css=me-2	[KEY_ENTER]
8	verify text	//*[@id="root"]/div/div/div(2)/div/div(2)/div/div/div/div(1)/div/div/div/div(2)/div(1)	Punisher (2020)

Figure 5.11: Selenium: Test 2

The steps are as follows:

- 1. Visit the search page of the page
- 2. Click on the search bar
- 3. Enter the name of an artist (e.g. *Phoebe Bridgers*)
- 4. Confirm that the artists albums appear on the page by checking their titles (e.g. *Punisher*)

The test in question passes with no hindrance at all, confirming that the albums of the specified artist do get displayed on the page after the user searches for them.

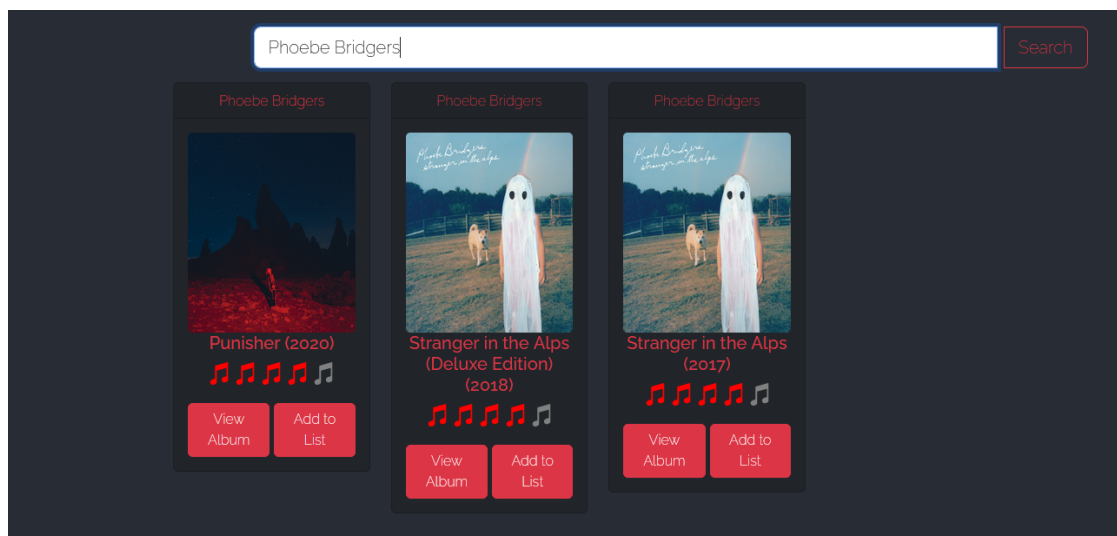
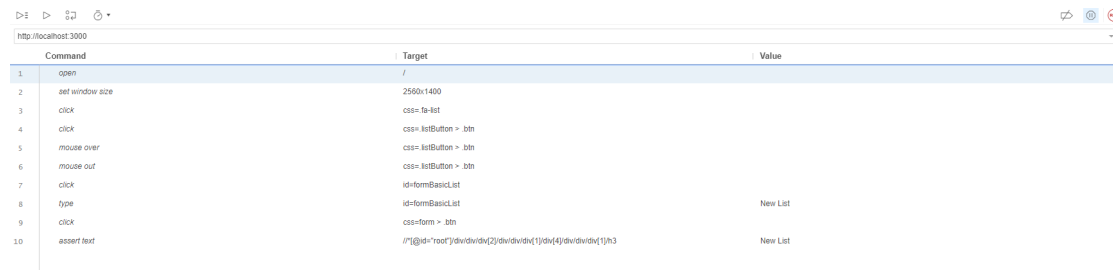


Figure 5.12: Selenium: Test 2 Result



### Test 3

This particular test ensures that when a user is logged in, they can create lists with a specified name. This is quite an important feature, due to the fact that the platform revolves around the idea of tracking albums as you go.



Command	Target	Value
1 open	/	
2 set window size	2560x1400	
3 click	css=fa-list	
4 click	css=fa-list > .btn	
5 mouse over	css=fa-list > .btn	
6 mouse out	css=fa-list > .btn	
7 click	id=formBasicList	
8 type	id=formBasicList	New List
9 click	css=form > .btn	
10 assert text	//*[id="root"]/div/div/div(2)/div/div/div(1)/div(4)/div/div/div(1)/h3	New List

Figure 5.13: Selenium: Test 3

The steps that the test are as follows:

- 1. Visit the lists page of the application
- 2. Click on the *Create a List* button
- 3. Enter the name *"New List"*
- 4. Confirm that the list is now displayed on the page

This test passes with no issues whatsoever, where the page adds the list without any refreshes required. We can the results of it as so:

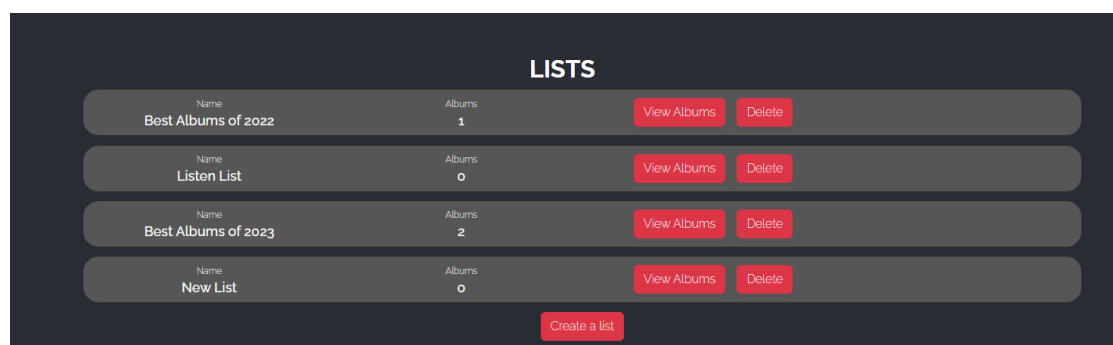
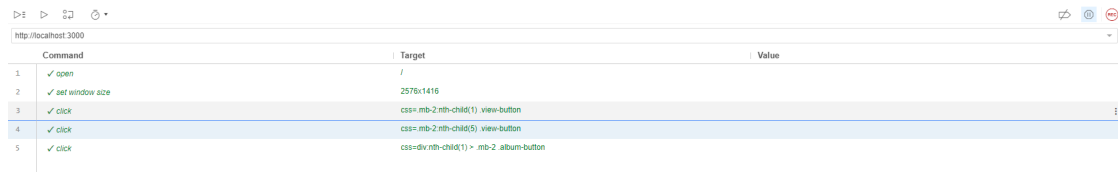


Figure 5.14: Selenium: Test 3 Result

## Test 4

The last test checks if the user can select any artist from any genre and listen to a song snippet from one of their albums. This allows to test any random genre or artist who pops up first on the page, and tests the latest album of that given artist. This ensures a certain level of randomness to the test, which allows for it to ensure that no matter what is chosen, that it passes nonetheless.



	Command	Target	Value
1	✓ open	/	
2	✓ set window size	2576x1416	
3	✓ click	css=mb-2:nth-child(1) view-button	
4	✓ click	css=mb-2:nth-child(5) view-button	
5	✓ click	css=div:nth-child(1) > mb-2_album-button	

Figure 5.15: Selenium: Test 4

We can break down the steps like so:

- 1. Navigate to the home page of the application
- 2. Select the first genre on the page (e.g. *Acoustic*)
- 3. Select the first artist on the next page (e.g. *Foo Fighters*)
- 4. Select the latest album of the artist
- 5. Play the song snippet from the first track on the album

This accounts for a visual and auditory test, where Selenium confirms that the element is present and playable. However, for the audio portion, I myself must pay attention to when the test itself is running to ensure that audio is being played. This test was run a few times for different albums and artists, where it passes each instance.

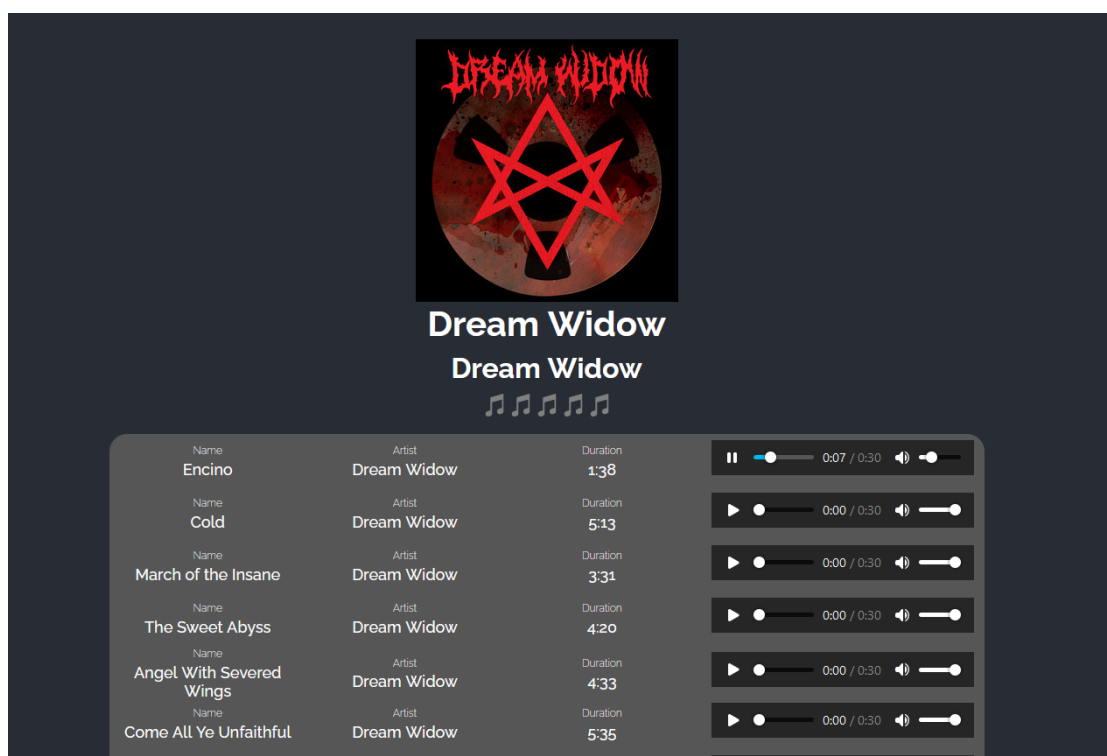


Figure 5.16: Selenium: Test 4

## 5.2 Weaknesses

With any application, there will be some weaknesses. This website is no exception to this rule. Whereas all tests pass as necessary, the weaknesses of the application and system fall into the the lack of certain features, and the visual aspects of the application.

Due to the strict time-frame, some features that I initially thought that I could implement had to be removed. The entire aspect of a *social* experience for the user was unfortunately unable to be met. The scope of the project had to be reduced during reflections on the sprints done at that stage of development (since I was following the Scrum approach). This lead to a more solo experience for the user to track their music listening journey. This feature was unfortunately the only project objective that I had laid out that I could not achieve.

On the visualization side of things, Bootstrap didn't work as well as I hoped. Although Bootstrap was useful, it's implementation for React was limited in comparison to it's original focus on static sites. For this reasoning, it was especially difficult to get components to display in the way in which I originally wanted. The difficulty of handling a dynamic website, which could have many varying things on screen at a given time, was immense. This lead to some visual quirks where certain components were not aligned the way in which I wanted, nor responding as intended.

These features in the future could definitely be improved, but for the time-being they are the lacking components of the project as a whole currently.

# Chapter 6

## Conclusion

Now that we have come to the finish of the project, it is important to be able to reflect and review the objectives we wanted to originally achieve when we set out to deliver a social media platform that solely revolved around music.

The original rationale of the idea for this application was to create an experience that isn't fully available currently in the market. Although there were many similar ideas for other mediums (books, movies, etc.) out there, the same could not be said about one for music listeners. Now that the project is finished, we can reflect back on the original goals of the project, to see if they were met as follows.

The goals of the project originally were:

- Provide a platform in which users can register and sign-up to use the application
- Produce a system which allows for the ability to search for artists and albums
- Let users review and track albums in which they have listened to
- Allow for users to create and add to lists to keep up-to-date on what they have yet to listen to
- Give the users the ability to follow other users to see what they are listening to and enjoying

Although not all requirements were fully met to the level in which was hoped for, overall the project did achieve a solid outcome. We discussed that we weren't able to implement the more social aspect of the application (due to time constraints and challenges). This is an aspect of the project that could be explored further in future investigations of the application, which can still be implemented. Visualization issues were also discussed, also being an aspect that can be worked on for future versions.

Since the project itself was a passionate endeavour from the beginning, continuing to work on this application in the future is still something in which I want to continue with. There were challenges along the way, but everything laid out was achievable. Given the right amount of time and insight, the application can achieve so much more. This leaves a feeling of excitement for what may come later.

Certain aspects of the development process took me by surprise as well, like the practicality of following methodologies. When initially beginning work in software, following such protocols or guidelines didn't feel like it really that necessary. However, the usage of Scrum helped me stay aligned and focused on the aspects of the application that I needed to deliver upon. Without taking this approach, I feel like I wouldn't get as much work done.

The impact of testing also was surprising. The importance of testing your application always known by me, however I never realised how much time it saves in the grand scheme of things. By incrementally testing components as I went (unit testing), it allowed to ensure that each cog in the machine was working as intended.

The most difficult aspects that I originally thought would hinder my time developing the project, were not as much of a problem as I was expecting. Aspects like handling user log-ins and registration took some research but was not the behemoth that I initially anticipated. The same could be said for deploying the application to a website. With the help of a few online resources, getting the application ready for deployment was a relatively easy thing to handle. On reflection, the troubles I had were implementing the more *social* aspects and the actual design of the website.

The website, due to its dynamic nature, did not behave as expected. It wasn't a massive hindrance to the application, however it would have been nice to get it to the exact standard that I wanted.

To give a general idea of the outcomes of the project and what I learnt, we can list them as follows:

- Learnt how to develop and handle a full-stack application
- Began understanding the importance of time-management and scheduling
- Experienced working with APIs and their fundamentals
- Evaluated the importance of UX and styling websites
- Conforming to a software methodology in a good fashion

Some insights that I originally did not realise is that I quite enjoyed developing a full-stack application. Since I had little experience in this field, it did feel

daunting originally. However, the process of experiencing each stage of getting an application up-and-running felt quite satisfying and also gratifying. Being able to apply creative and thinking skills were extremely vital in getting the application to where it needed to be. By having to think in such varying ways (sometimes even considered opposites of one another), this in it of itself felt like a challenge. By having the chance to work on both the front-end and back-end, it helped me understand the importance of both. Overall, the experience I feel that I have gained by working on this project has been a fulfilling one, while also helping me grow as a developer. The freedom to develop what I wanted to felt like a good chance to show my abilities, due to the fact that once you are working on a project that is something you care for, you are more likely to put in as much of yourself into it as possible.

Overall, I enjoyed working on this project. It helped improve many of my skills, whether that be specifically my coding skills or the ability to keep on schedule to meet all the deadlines that I set for myself.

# Chapter 7

## Appendix A: User Manual

### 7.1 Introduction

This appendix contains the user manual for the software developed in this project. The software is designed to perform various functions, including data processing, analysis, and visualization. This manual provides step-by-step instructions for using the software.

Github Repository: <https://github.com/shanemcdonagh/rubato>

### 7.2 Prerequisites

To ensure that the project can work as intended, the specific packages are needed:

- *Node.js* version 16.14.0 or higher
- *Node Package Manager* version 8.19.2 or higher

Both these packages can be downloaded at: <https://nodejs.org/en/download>. All other dependencies will be installed in the next step

### 7.3 Installation

To install the software and run it locally, follow these steps:

1. Download the software from the project repository on GitHub.
2. Navigate to the route folder and run the following command in a terminal:  
**npm install.**



3. Navigate into the *rubato* folder and run the following command in a terminal:  
**npm install.**
4. Within the route folder, run the following command to start the server: **node  
.\backend.js**
5. Within the *rubato* folder, run the following command to start the front-end:  
**npm start**
6. Navigate to **http://localhost:3000**

The application is also available through the following url, without the need for installation: <https://rubato.herokuapp.com/>

## 7.4 Troubleshooting

If you encounter any issues or errors while using the software, please refer to the following troubleshooting guide:

1. If the pages are not loading any content, restart the server.
2. If the issue persists, check the log displayed by the server for additional information.
3. If the packages do not install correctly, try to re-download the project from the repository and follow the installation steps again

## 7.5 Conclusion

This user manual provides detailed instructions for installing and using the software developed in this project. It also includes a troubleshooting guide to help users resolve any issues or errors that may arise while using the software.

# Bibliography

- [1] Claes Wohlin, Martin Höst, and Kennet Henningsson. 13 empirical research methods in web and software engineering1. *Web engineering*, page 409, 2005.
- [2] University College London. University college london, 2021.
- [3] Vera Solutions. Vera solutions, 2019.
- [4] Amazon. What is sdlc (software development lifecycle)?, 2023.
- [5] Ivan Shikht. System development life cycle (sdlc): Phases, models, and benefits, 2023.
- [6] S Kumar. What is iterative model- advantages, disadvantages and when to use it?, 2017.
- [7] Wrike. What is agile methodology in project management?, 2023.
- [8] Agile Manifesto. What is agile methodology in project management?, 2001.
- [9] Scrum.org. What is scrum?, 2001.
- [10] Selenium. Selenium, 2023.
- [11] Inc. Postman. Postman api platform, 2023.
- [12] Mozilla. Html: Hypertext markup language, 2023.
- [13] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. Document object model (dom) level 1 specification. *W3C recommendation*, 1, 1998.
- [14] Mozilla. Using the document object model, 2023.
- [15] Eric A Meyer. *CSS: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2006.

- [16] Bootstrap. Get started with bootstrap, 2023.
- [17] Mozilla. What is javascript?, 2023.
- [18] Ecma International. EcmaScript 2022 language specification, 2022.
- [19] MongoDB. Mern stack, 2023.
- [20] Kosovare Sahatqija, Jaumin Ajdari, Xhemal Zenuni, Bujar Raufi, and Florije Ismaili. Comparison between relational and nosql databases. In *2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0216–0221. IEEE, 2018.
- [21] MongoDB. Nosql explained, 2023.
- [22] Why use MongoDB? Mongoddb, 2023.
- [23] JSON. Json, 2023.
- [24] Mozilla. Mozilla documentation: Json, 2023.
- [25] Oracle. What is json?, 2022.
- [26] Express. Express documentation, 2017.
- [27] Express. Express routing, 2017.
- [28] Express. What is a rest api?, 2023.
- [29] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.
- [30] Spotify for Developers. Getting started with web api, 2023.
- [31] React. Getting started - react, 2023.
- [32] Madhuri A Jadhav, Balkrishna R Sawant, and Anushree Deshmukh. Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, 6(3):2876–2879, 2015.
- [33] Mozilla. Ajax - developer guides, 2023.
- [34] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.
- [35] Node.js. About node.js, 2022.
- [36] Node.js. Node.js v19.9.0 documentation, 2023.

- [37] Node.js. An introduction to the npm package manager, 2023.
- [38] npm. npm docs, 2023.
- [39] Ling Qian, Zhiguo Luo, Yujian Du, and Leita Guo. Cloud computing: An overview. In *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*, pages 626–631. Springer, 2009.
- [40] Heroku. Cloud application platform, 2023.