“Music makes you run faster…?”

Author Student No: C1827663

Author Name: Tyler Cole Jenkins

Module Code: CM2303

Module Title: One Semester Individual Project

Module Leader: Dr. Frank Langbein

Supervisor: Martin Chorley

Moderator: Carl Jones

Word Count: 23538

Word Count excluding Title Page, ToC, ToF, Appendices and References: 21032

Abstract

Running is one of the most accessible sports in the world. With more and more people becoming interested in the sport, there are also more people looking for ways to improve their performance. Many runners like to listen to music whilst they run, and many believe that certain songs can help them to push that little bit further and motivate them during their activities.

With a recent rise in smartphone applications that allow users to track their runs, it is now easier than ever to track your running performance over a period of time and visualise areas where you could improve. In addition to this, the number of people using a music streaming service which allow them to stream music anywhere and anytime is at an all-time high. These two services together provide the basis for this project. This project aimed to investigate ways in which we could help runners understand if music does have an impact on their runs. By creating a web-application which allows users to connect their accounts from a popular activity tracking service known as Strava, and a popular music streaming service known as Last.FM, this project will act as a tool for allowing users to visualise the effects that music has on their runs and thereby ultimately help them to decide if "Music makes you run faster.".

Acknowledgments

I would like to thank my supervisor, Martin Chorley, for meeting with me consistently throughout this project to monitor the project's progress and for providing valuable suggestions throughout development.

Table of Contents

Table of Figures

1. Introduction

1.1 Introduction

This project is based on the idea that music has an impact on people's performance when they are exercising. It is an interesting concept to think that different kinds of music can have both positive and negative effects on somebody's performance, but how exactly does a song affect someone whilst they are exercising? Do certain songs increase their average heart rate or their average running speed? Or how about different genres of music? Is a certain genre of music the best for listening to during exercise? In order to help users better understand what music they perform best to; this project aims to create a web-application (web-app) that will allow a user to compare their tracked exercise history with their music listening history. This web-application has been named "RaceTracks" and will be referred to by this name throughout this report. RaceTracks aims to help users determine whether or not "Music makes you run faster".

1.2 Initial Assumptions

There are a number of assumptions which were made at the start of the project. Firstly, it was assumed that most runners are likely to utilise a mobile application (app) to track their runs, for example, Strava, Fitbit or Apple Fitness. It was also assumed that whilst doing so, these users also employ the use of a music streaming service to listen to music, such as Last.FM, Spotify or Apple Music. The final assumption made was that most of the end-users for this solution will make use of some form of smart watch or fitness tracker during their exercises which will measure their heart rate during the course of their activities.

1.3 Project Aims and Objectives

During the initial planning phase for this project, several aims were outlined to determine the goals of the project, and these were split into primary and secondary aims. The primary aims of the project were outlined to be the aims which were most important to achieve in order to create a minimum viable product (MVP and were expected to be achievable within the time frame of the project. The secondary aims were outlined as further functionality which would further improve the solution, although the feasibility of being able to deliver the features and achieve these aims within the time frame of the project was unclear. The list of aims were as follows:

Primary Aims:

1. To create a web-application which will display a user's running activity history as well as their music listening history for each activity.
    a. Recent activities should be displayed to the user with an option to search for activities by date.

2. To implement functionality which allows comparisons between a user's single running activity and the music they were listening to during the time-period of that activity.
    a. This should include information such as which song they were listening to when their heart rate / pace was at its highest.

3. To implement functionality which compares all of a users' running activities and all of the music listening history for those activities, across a given time frame by the user.

    a. This should include also include information such as what genre of music the user typically performs best at when listening to.

4. To design and implement a suitable and user-friendly user interface.
    a. It should be intuitive to use with a mobile-first approach taken for development.

Secondary Aims:

1. To update the application logic to allow for different kinds of activities to be included as opposed to just running. For example, cycling, weightlifting, boxing etc.

2. To extend the level of detail which is offered by the insights of the music and activity comparison, for example calculating the optimum beats per minute (bpm) of the music the user should listen to when they exercise.

3. To add functionality for song suggestions based on how the user performs whilst listening to certain genres of music.

4. To implement a log-in system which would mean that users would not have to connect their Strava and Last.FM accounts each time they want to use the application.

After beginning development and progressing through the early stages of the project, it was realised that there were several other features which hadn't been outlined in the original set of aims but would provide a great opportunity to improve the implementation. One of the new aims which was set during the development process was:

- To deploy the web-application on a live-hosting service, to make it accessible online to any user.

This aim is a strong measurement of the success of the project. By being able to host a fully functional application on a live server, this would be a strong demonstration that the development of the solution was successful. The final aim which was set during development for this project was:

- To investigate the incorporation of mapping tools, to display to a user where they have run during their recorded activities.

This aim was set following the design phase for the initial user interfaces. It was noticed that the page designs felt as though they were missing something and could possibly feel quite uninteresting to a user. Adding a map to the designs aimed to engage the user and provide them with more real-world context as to what section of a run the song data is referring to. It also allowed me to set myself a technical challenge, as I have minimal experience in working with web-development mapping tools, and I was interested to see what I could do with my implementation.

Alongside these aims, there were also some personal objectives which outlined outcomes that I wanted to achieve and things that I wanted to learn as I went through the process of developing my solution. These were as follows:

Objectives

- To investigate, understand and work with external developer API's available from several companies and see the extent of what data can be obtained from each one.
- To investigate, understand, and implement authentication workflows using authentication procedures used by external API's.
- To work with available resources to better understand and learn some best practices for front-end JavaScript development.

1.4 Intended Audience

The intended audience for this project are runners who actively track their activities already through a mobile application and listen to music through a music streaming service whilst doing so. The audience of the solution however is not limited to those who are already tracking their activities, it is open to anybody who would like to gain a better understanding of how music can affect their performance, and once they have begun collecting data on their workouts, they will also be able to use the solution. The existing user base for activity tracking and music streaming services is huge. One of the largest activity tracking services in the world is Strava, which as of 2021 reportedly had over 76 million users (1). Similarly, looking at one of the world's leading music streaming services, Spotify, who as of when this report was written, currently have over 406 million users, 180 of which are subscribed to their premium paid service (2). Anybody who regularly uses both of these services at the same time will be included in the intended audience. Whether a user would like to gain more of an insight into how music affects their performance or whether they would just like to get a recap of what songs they listened to on a specific run, the RaceTracks application would suit their needs. With music being an important aspect of many people's runs, I think that the opportunity to have an application which can correlate data between the two services could be beneficial to all runners looking to improve their performance, whether they be at a casual or at a high-level.

1.5 Project Approach

The approach to this project was set out in the initial plan. A rough timeline was created in the form of a Gantt chart which outlined the tasks that would need completing, the order in which they should be completed and roughly how long they should take. This way of working initially sounds very much like the waterfall methodology way of working, where every stage of work must be done in order and the next phase of the project is not started until the current one has been completed. However, the approach on this project was much more in tune with the agile methodology, where I developed work in 1-week sprints that allowed me to prioritise tasks each week and work on developing the features which were brought into each sprint. The priority of certain tasks rose or declined as things changed during development (3). Depending on the success of a sprint in a week, if all of the tasks had been completed early, I was able to bring in new tasks which would have been set for the following week. This allowed for a high degree of freedom and flexibility during the development process and granted me the ability to adjust the projects scope where necessary. Because of this way of working, I was also able to implement some of the extra features which I had hoped to, whilst scaling down on other features I initially thought might have been quite important to the solution but ultimately were not.

1.6 Project Scope

The initial scope of this project was to create a web-based application which would allow a user to connect to their account with an activity tracking service and a music streaming service to receive an analysis of how the music they listen to affects their running performance. The scope that was defined in the initial plan was certainly feasible within the time frame of the project, and I was able to implement almost all of the features that I had aimed to, as well as add a few new ones which were not specified in the original plan. However, one of the secondary aims was to investigate the implementation of a log in system for my application, and although it was possible to delegate some time to looking at implementing this, the feature unfortunately had to be de-scoped from the project. The technical requirements and time investment for implementing this functionality were just not feasible within the time left after the first phases of development. The addition of this feature would also have meant that I would have had to through gain ethical approval for the project since I would have been storing users' data, and again unfortunately, I did not have the time remaining to do so. It is unsurprising however that this feature had to be de-scoped, as it was clear from the beginning that this would be a big undertaking for the project. The exemption of the login system did not have any impact on the functionality of the system, and it presented itself as more of a "nice-to-have". Disregarding this feature, the scope of the project was realised as being suitable and manageable for my skillset with the time given to develop it.

1.7 Summary of Outcomes

The main outcome from this project, given the aims that have been listed above and the scope of the project, is to create a live hosted web-application which is accessible online, that provides runners with a tool to connect to both their activity tracking and music streaming services in order to gain a better understanding of how music affects their performance. In regard to my own personal outcomes, I aimed to gain more experience in JavaScript web-development, external API's (Application Programming Interfaces), mapping frameworks, version control and live web-hosting applications.

2. Background

This section of the report aims to provide an explanation of some necessary background topics which are needed to fully understand the later content of this report.

2.1 Wider Project Context

As previously mentioned, the use of activity tracking applications is becoming more and more common. They are used across a wide variety of sports such as running and cycling, to help the athlete better gauge their performance during their activities, and visual their progress over time. These apps support the measuring of a user's pace and activity time through smart phone applications. Smartphones are becoming increasingly more accurate on measuring this data, and use metrics such as a person's height, stride, the accelerometer, and GPS location data to calculate an accurate as possible reading (4). Although smart phones are becoming more and more reliable for measuring this data, the activity tracking service is best used in combination with some form of smart watch or fitness band which supports the tracking of the user's heart rate, calories burned and other biometric data. If a user is not using one of these devices whilst tracking their workouts, then the data which can be obtained is somewhat limited and only includes metrics such as their pace, step count and activity time.

The activity tracking service which has been chosen to implement into the RaceTracks solution is Strava. Strava is one of the largest activity tracking services in the world and provides users with detailed breakdowns of their activities. One of the main reasons for choosing to implement a solution with Strava, is that all of the data collected from a user can be accessed through a well-documented public-facing API, as long as the user has authorized an application to access their data. In addition to this, it was not feasible to implement a hexagonal architecture (5) into the solution which allow a user to connect to several different activity tracking services given the time frame of this project, and so therefore it was an important decision on which service was going to be chosen. Ultimately, Strava was chosen for a very important reason, Strava allows users to connect their accounts from several different third-party activity tracking services and import all their activities to Strava seamlessly, this includes services such as Fitbit, Apple Fitness, Nike+, Garmin, Samsung Health and many more (6). This helps to mitigate the issue of RaceTracks not being able to connect a number of different services. Any user who tracks their activities on a service which is not Strava, would simply need to create themselves a Strava account and connect their account for the service which they use regularly. Following this, all of their activity data will be automatically synced with Strava and would then be made accessible to RaceTracks. By using Strava as almost a hub for the activity tracking services, we also eliminate the issue of completely isolating an entire user base from other applications and allow RaceTracks to become a much more inclusive application.

Music streaming services are also widely used by people who exercise, with most people being subscribed to a paid service such as Spotify, Apple Music or Tidal. Before beginning development, I did some research around these services in order to see how accessible a user's data would be. After reading through their API documentations, it appeared that none of these services would be suitable for the approach I was taking. A key feature of the RaceTracks application is that it needed to allow a user to view any of their activities recorded on any given date and the music they listened to during that activity. Because of this, we need a complete record of a user's entire music listening history. This posed a severe issue with the previously mentioned services into RaceTracks.

For example, although Spotify offers a public-facing API, Spotify does not keep a permanent record of a user's listening history which is directly available through their API, we can only get a list of a user's most recently listened to tracks, this was also true for Apple Music. This meant that by using one of these services, it would not be possible to implement the solution as intended. This is where a website called Last.FM becomes very useful. Last.FM is also a music streaming service, but unlike the other services previously mentioned, it keeps a complete record of a user's listening history. In addition to this, users can connect their accounts from other services including Spotify, Apple Music and Tidal, and Last.FM will keep a record of any songs listened to on those services too (7). Last.FM also provides a well-documented public-facing API which will allows access to a user's listening history.

By using Last.FM there is also the benefit we saw with using Strava as a hub for external applications, in that it is helping to mitigate the issue of isolating an entire group of users. For example, if Spotify had been chosen, then any users who listen to music through apple music or another service would be unable to use the application.

2.2 Identified Problem and Stakeholders

The problem which has been identified is investigating the possibility of building a web-based application that will allow users to connect an activity tracking service (Strava) and a music streaming service (Last.FM) to gain an understanding of what music positively and negatively affects their performance whilst exercise.

The relevant stakeholders which have been identified are:

- People who exercise regularly whilst tracking their workouts and listening to music through a supported platform.
- The chosen activity tracking platform: Strava.
- The chosen music streaming service: Last.FM.
- An external data source which was used for retrieving extra music data: TheAudioDb.
- Myself as the project developer.
- My project supervisor.

The first mentioned stakeholder and the projects main target audience are people who exercise regularly, tracking their activities whilst streaming music. The end users are one of the most vital stakeholders as they are the people who will ultimately use the solution (8). Their interest in the project is a deciding factor between the project's success or failure. The project should meet their expectations and provide them with the features and the functionality that they expect.

Another stakeholder of the project is the activity tracking service which was chosen to base the implementation around; Strava. Strava has a stake in the project as without the data provided by the, the solution fails to function. The solution works on the basis that the data received from Strava contains specific fields and is structured in a very specific way, there is a responsibility which lies with Strava to ensure their data is stored and provided correctly. If the structure of their data or the way in which their API functions were to change, then it is highly likely that the solution will fail to work. In addition to this, Strava have a stake in the project as they may be interested in novel uses of their data as well as the potential outcome of the project. For example, if the project were to be majorly successful, could it become something that Strava as a business would like to explore themselves.

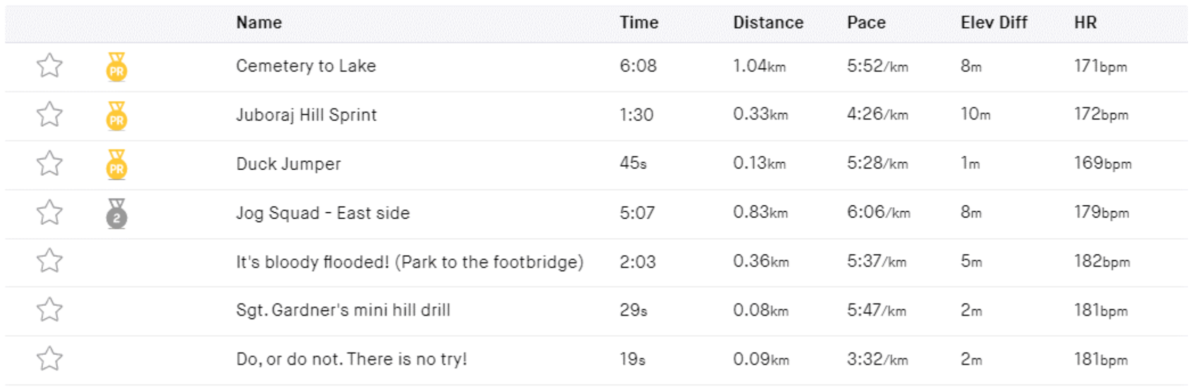
Furthermore, the music streaming service which has been chosen to implement the solution around; Last.FM, are another key stakeholder to the project. They provide an API similar to Strava, with data which is crucial for the RaceTracks application to function. If the structure or contents of the data provided by Last.FM's API, or the way in which their API handles requests were to change, then the application would fail to function. They also may have a vested interest in the project's outcome and seeing different possible uses of their data.

A further stakeholder is one more source which is used to obtain music data, a web data base called TheAudioDb. Their service requires a paid monthly membership to be able to use their API and there are several different tiers at different price points which allow you to make a certain number of API calls each day. As an organization, they would have an interest in the overall success of the project; as if the scope of the RaceTracks application were to increase and there were to be multiple daily users, the tier of the current paid membership would need to be increased, ultimately increasing their revenue from this project.

The final two stakeholders that have been identified for this project are both myself as the developer, and the project supervisor. Both of us have a vested interest in the project and would like to see it succeed to its full potential.

2.3 Associated Theory

It is important to understand how we can infer from the activity data provided by Strava and the music data provided by Last. FM, how music is having an impact on a user's performance. There can be different types of activities recorded on Strava, however the focus for this implementation was on running activities. For every recorded run, Strava provides several pieces of useful data which we can use to measure a user's performance across the duration of their run. This includes data such as their max heartrate, average heartrate, distance covered and time elapsed. In addition to this, Strava breaks down each run into several separate sections which it calls "Segments".



		Name	Time	Distance	Pace	Elev Diff	HR
☆	PR	Cemetery to Lake	6:08	1.04km	5:52/km	8m	171bpm
☆	PR	Juboraj Hill Sprint	1:30	0.33km	4:26/km	10m	172bpm
☆	PR	Duck Jumper	45s	0.13km	5:28/km	1m	169bpm
☆	2	Jog Squad - East side	5:07	0.83km	6:06/km	8m	179bpm
☆		It's bloody flooded! (Park to the footbridge)	2:03	0.36km	5:37/km	5m	182bpm
☆		Sgt. Gardner's mini hill drill	29s	0.08km	5:47/km	2m	181bpm
☆		Do, or do not. There is no try!	19s	0.09km	3:32/km	2m	181bpm

Figure 1: Example of a run activity on Strava separated into segments

These segments are different sections of a run, it could be that the road from your house to the park is one segment, and then half a lap of the park is the next segment. For each of these segments Strava provides a start time, an end time, the distance covered, the max heartrate achieved during the segment and the average heart rate across the entirety of the segment. Using the distance and time taken to complete each segment, we can then calculate the user's speed for each segment for a run as “speed = distance / time”. With this we can then identify the segment where a user was running at their fastest.

Using the start and end time for the segment, we are then able to get the songs the user listened to during that segment and provide them with this list, as well as suggest similar songs they could try listening to. Using this method of analysing segments, we are also able to provide the user with the songs they listened to when they performed at their lowest on their run.

Expanding on this, one of the project aims was to allow a user to analyse their performance across several activities. By selecting a date range and looking at all of the runs within that time-period, we can get the segments for each run, calculate the pace for each of them and then find which segment was the users best and worst then perform the same action of getting the songs for each segment and displaying them to the user. This was the most suitable method I found for evaluating a user’s performance during a run.

2.4 Existing Solutions

The closest existing product which aims to provide a similar solution to the problem proposed by this project is a smart phone application called Perform (9). Perform allows users to connect their Strava account alongside their Spotify or Apple Music account to provide their service. Perform aims to help users improve their workouts by using artificial intelligence to create a playlist of songs in real-time as you run. The application looks to be a well built and very professional service, which has a whole range of features which the RaceTracks solution would be unable to offer under the current aims and scope. Although this service can be labelled as similar to RaceTracks, there are several features which have been implemented into RaceTracks which Perform do not offer as part of their service. Firstly, Perform only provides support for running activities. RaceTracks introduces support for visualizing a user’s listening history over different kinds of exercises such as High Intensity Interval Training and weightlifting. Although the support for these types of activities may be minimal currently, the option is still there for users who do not exercise only by running and can be built upon in the future. Secondly, RaceTracks offers users the ability to compare a selected number of activities across a specific date range and view which songs the user performed best to during those activities. This is something which Perform, to my knowledge, does not currently offer. Lastly, Perform limits its users to those who subscribe to only Spotify or Apple Music, and only keeps a record of their listening history from the moment they create a Perform account and link their music streaming service. RaceTracks allows users to connect a Last.FM account which allows for the integration of a number of different music streaming services. This makes RaceTracks much more open to different user groups, and those who already have a Last.FM account will already have all of their music listening data stored.

One other solution which is similar to RaceTracks is a service called RockMyRun (10). Like RaceTracks and Perform, RockMyRun’s goal is to aid users in better understanding what kind of music can make them run faster. RockMyRun acts as its own activity tracking and music streaming service and creates playlists of optimized running music for users to listen to whilst they run. It provides heart rate tempo matching and allows the user to manually set the BPM of the songs being played to match their stride pattern. As RockMyRun is a stand-alone service, this negates the need for users needing to have a specific external activity tracking or music streaming service. Although this may be a benefit for some users, this also has its deficiencies. For example, users will not be able to retrospectively analyse any of their runs which they have already recorded on other platforms such as Strava. This may also be off-putting for some users, as they will need to exclusively switch over to

RockMyRun and will not be able to compare any of their new runs with their old runs on other services. RaceTracks aims to improve upon this deficiency and integrates directly with Strava and so there is no need for a user to switch services and start tracking their runs from fresh, they can simply connect their pre-existing account.

In conclusion, to my knowledge, there is no pre-existing web-application which allows a user to connect directly with Strava and Last.FM to provide an analysis of how a user’s performance during exercise is impacted by the music they are listening to in the way that my solution does so. There are similar solutions which have a similar goal which have been discussed above (Perform and RockMyRun), however they act as their own activity tracking and or music streaming service, and do not support the option for users to connect their pre-existing accounts from other services.

2.5 Methods and Tools Used

Regarding the development tools used and those that will be referred to in the upcoming sections of this report, the entirety of the RaceTracks web-application was written in JavaScript. More specifically, JavaScript alongside React, version 17.0.2 (11). React is a free and open-source, front-end JavaScript library which is incredibly useful for building user interfaces and single page applications (12). I used a tool called “Create React App” (13) which creates a basic front-end pipeline build and also sets up certain other JavaScript packages such as Babel (14) and Webpack (15) which are needed for developing web-applications with React. Babel is a JavaScript compiler and Webpack is used for bundling assets and the compiled JavaScript files together to create a production-ready build of the web-application, which can then be deployed on a webserver. “Create-react-app” also sets up Node.js (16), which is a JavaScript package manager, that lists the dependencies of the packages which are used in the web-app and ensures they are all installed to their correct versions.

One of the aims for this project was to develop a web-application which would be accessible on both desktop and mobile platforms. To do so, a mobile-first development approach was taken, and several tools were utilised to ensure the web-app had a responsive design that would dynamically fit the screen size of the device it was being used on. In order to facilitate this, I used the Bootstrap CSS styling package to promote quicker development and help me structure the elements on the site (17). Alongside this I utilised a more specific package called “React-Bootstrap” (18), which as the name suggests, is provided by bootstrap to be used specifically in React applications. It provides several pre-created, complex UI components which can be embedded onto a react application and then customised. Some of the components that were utilised were the loading spinner which provided a spinning animation in place of data which hasn’t loaded yet and the carousel component to display various data to the user on my web-apps home screen. Shown below in Figure 2 is an example of the default carousel component included in React-Bootstrap. Figure 3 shows a customized version to display a user’s recently listened to songs on the RaceTracks home page.

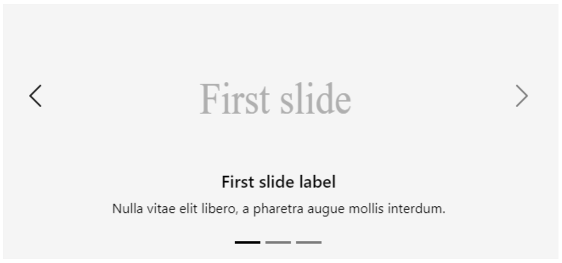


Figure 2: Default React-Bootstrap Carousel Component

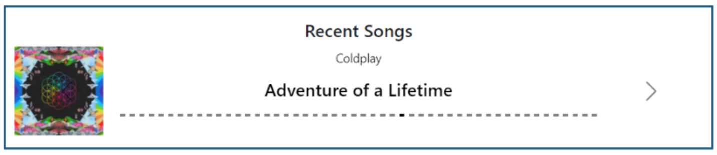


Figure 3: Customised React-Bootstrap Carousel Component used on RaceTracks

Other JavaScript libraries which were used during development were Leaflet (19) and React-Leaflet (20). Leaflet is the leading open-source JavaScript library for developing mobile-friendly and interactive maps. React-Leaflet is an extension which adds a few components to make these mapping tools much easier and suitable to work with in React. I also utilised the Luxon.JS (21) package which is a library for dealing with dates and times in JavaScript. This allowed for the conversion of dates from a UTC format as provided by Strava and Last.FM, to a more human readable format. A final JavaScript package that was used was called “React-Minimal-Pie-Chart” (22), a simple React component which comes bundled in a Node.js package, that allows the creation of easily customizable pie charts.

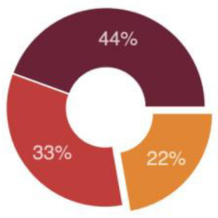


Figure 4: Default React-Minimal-Pie-Chart Component

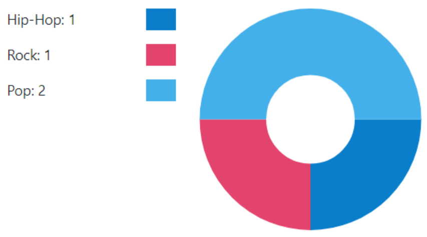


Figure 5: Customized React-Minimal-Pie-Chart Component

One of the most important tools during development was Microsoft Azure DevOps (23). Azure DevOps provides unlimited free Git hosted repositories which allowed me to employ correct version control and therefore I was able to set up the project with a main branch and a development branch. This allowed me to work on different branches of code and experiment with different solutions, before committing any changes to the main code repository. I used Microsoft Visual Studio Code as my Integrated Development Environment (IDE) which connects directly to my Azure development account to pull, push, and commit code changes. I also used Microsoft Azure for hosting a live version of the completed RaceTracks web-application. This was done using their build and release pipelines.

2.6 Further and Implicit Assumptions

As the project developed, there were some further assumptions that were made. It was assumed that most users would want to retrospectively view their activities from any previous date and not just recent ones, which was an important factor when choosing between Last.FM and Spotify, as Last.FM provided a complete historical data set. The assumption was also made that users would be looking for a strong level of detail on their performances and not just a brief overview or reflection of an activity, which was made during the design process.

2.7 Concluding Research Questions

2.7.1 Aim:

The aim of this project is to develop a responsive web-application that allows users to connect their accounts from an activity tracking service and a music streaming service, in order for them to compare their exercise data against their music listening data to see how music impacts their performance during exercise.

2.7.2 Research Question(s):

"In order to demonstrate the achievement of the stated aim, this project will provide support for connecting a single activity tracking service and a single music streaming service, use an algorithm to process the data provided from both services such as the users' pace and heart rate, and therefore help the user conclude which songs made them run faster."

3. Specification and Design

This section of the report aims to provide an explanation of the business model supported by the solution, the design of the user interfaces and the structure of the system architecture.

3.1 Business Requirements

As part of the specification and design process, the business requirements of the solution needed to be outlined. Business requirements can be defined as being used to describe the characteristics of a proposed system from the viewpoint of the end-user (24). With this in mind, and using the project aims as a basis, these are the following requirements which have been outlined in order for the system to meet the user's needs:

- A user can view their recently tracked activities.
- A user can search for activities within a given date range.
- A user can view the songs listened to during those activities.
- A user receives information about those songs, (BPM, genre).
- A user can compare their activity and listening history for a single activity.
- A user can compare their activity and listening history for a group of activities.
- A user can see an analysis of their activity, and view segments where they performed at their highest and lowest.
- A user can easily navigate the interface.

Each requirement has been refined from the high-level project aims and attempts to provide a direct reference point as to what the end-system should be able to do.

3.2 User Interface

One of the primary aims for this project was to design and implement a user-friendly interface which was suitable for both desktop and mobile views. Before beginning development on this project, I did not have much experience in designing user interfaces and so I decided to begin by conducting some research into best practices of designing suitable interfaces and how to strive towards a good user experience. After some research, I found some ideals around developing web-apps with mobile interfaces in mind, these included points such as keeping content and interface elements to a minimum to ensure only relevant data is displayed to the user and minimizing user input so that the user can easily navigate the application (25).

Before creating a design for the user interface, I began development work on retrieving data from the Strava and Last.FM APIs to see what data was available to work with. After early stages of development and setting up the data flows for the API's, the solution had this simple user interface (UI) as shown in Figure 6.

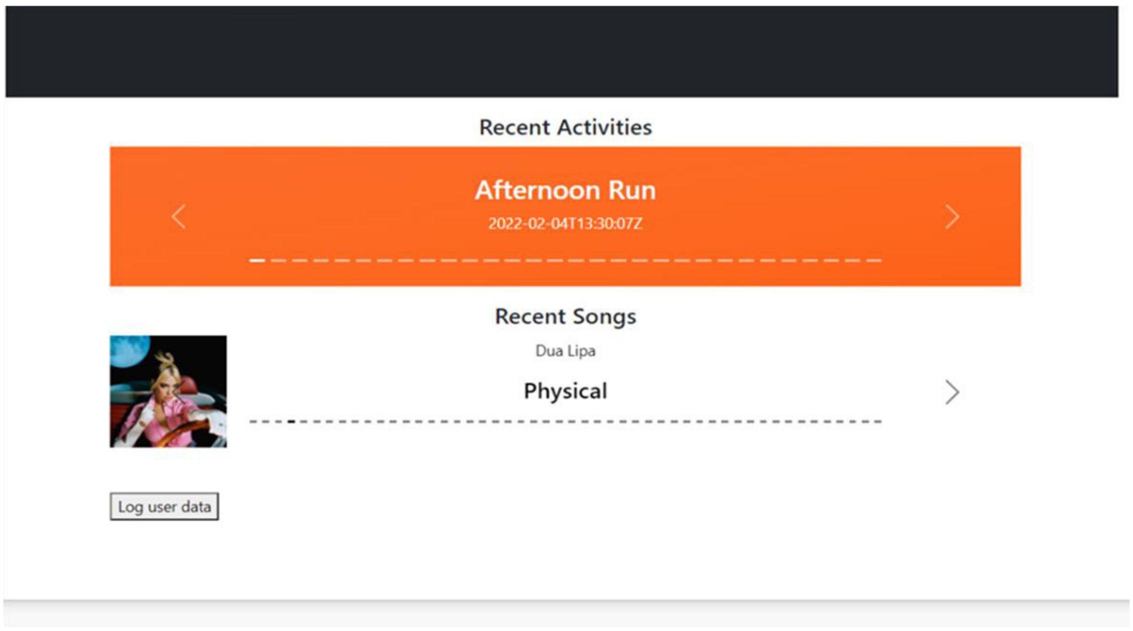


Figure 6: Home page view during early stages of development

This initial page became the inspiration for how the final solution would look and some of the functionality that would be present.

3.2.1 Connect Strava and Last.FM Accounts Screens

Once the data from Strava and Last.FM was available in the application I began designing the various pages which would make up the RaceTracks site. The first designs made were for the first pages the user would visit when they access the application. These were the screens for connecting their Strava and their Last.FM accounts. I wanted to keep these simple, as there didn't need to be anything displayed to the user besides some information telling them to connect their accounts and some buttons. My initial designs can be seen below in Figure 7 and Figure 8:

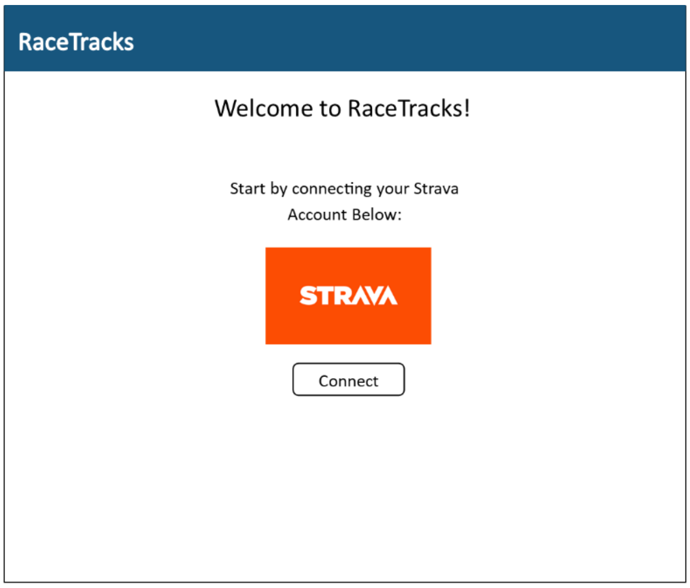


Figure 8: Wireframe showing Connect Strava Account screen

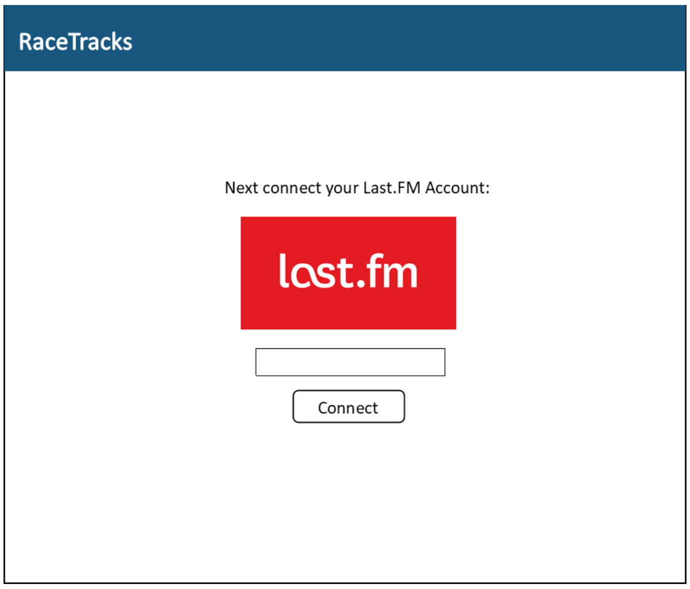


Figure 7: Wireframe showing Connect Last.FM account screen

These designs were kept simple for the user. With a button to redirect the user from the Strava page, and a text box on the Last.FM page for the user to enter in their username to connect. This is also where the colour scheme for RaceTracks was decided, I chose to use a blue colour after reading about how users can perceive websites with different colours. A blue colour is often associated with optimism and reliability, making users feel more confidence in a system, whilst also making it appear more professional (26).The final designs for these screens did not change much in the final implementation. The only difference being that there were a few icon images added to give the page some more substance and allow the user to infer what the application would offer more from imagery. The final designs can be seen below in Figure 9 and Figure 10.

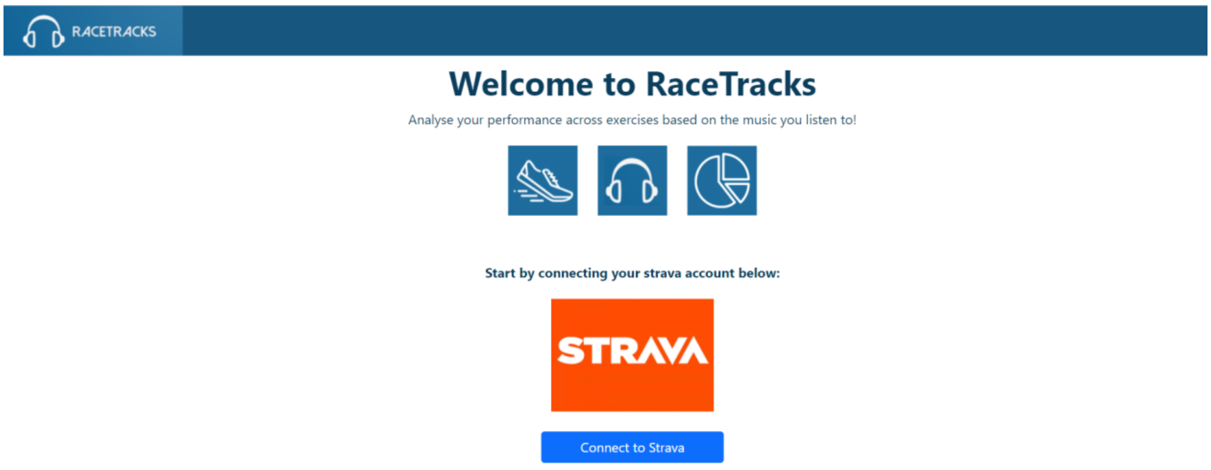

RACETRACKS
Welcome to RaceTracks
Analyse your performance across exercises based on the music you listen to!
Start by connecting your strava account below:
STRAVA
Connect to Strava

Figure 9: Screenshot showing final Connect Strava Account Screen

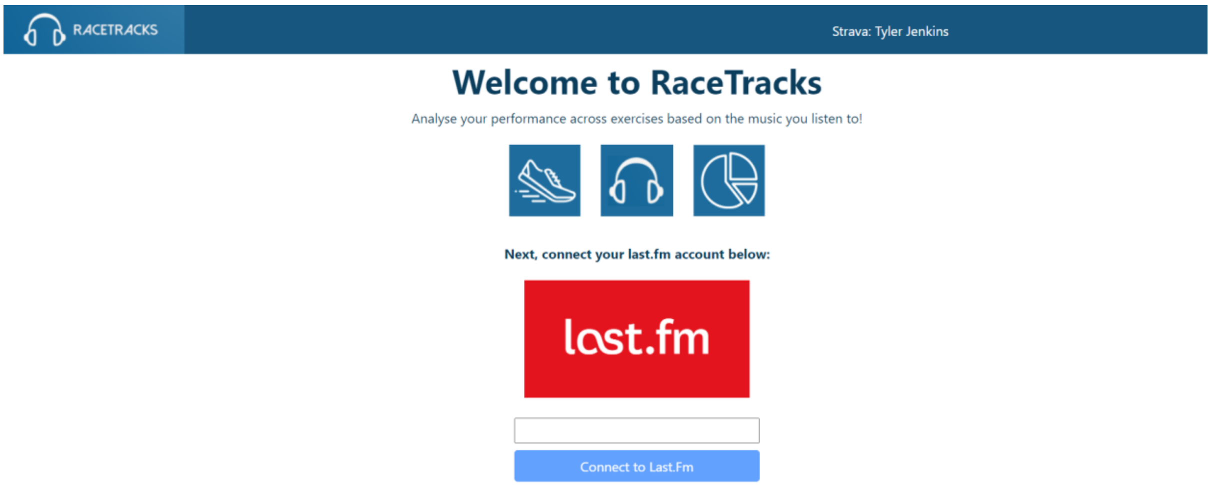

RACETRACKS
Strava: Tyler Jenkins
Welcome to RaceTracks
Analyse your performance across exercises based on the music you listen to!
Next, connect your last.fm account below:
last.fm
Connect to Last.Fm

Figure 10: Screenshot showing final Connect Last.FM Account Screen

3.2.2 Home Page

The next page that wireframes were created for was the home page. These designs were inspired heavily from the initial prototyping solution shown in Figure 6. There were wireframes created for both the desktop and mobile view of the page, both of which can be seen below in Figure 12 and Figure 11.

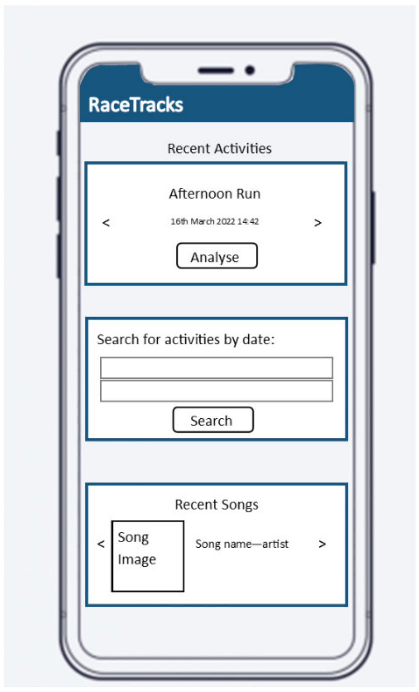


Figure 12: Wireframe - Mobile view of home page

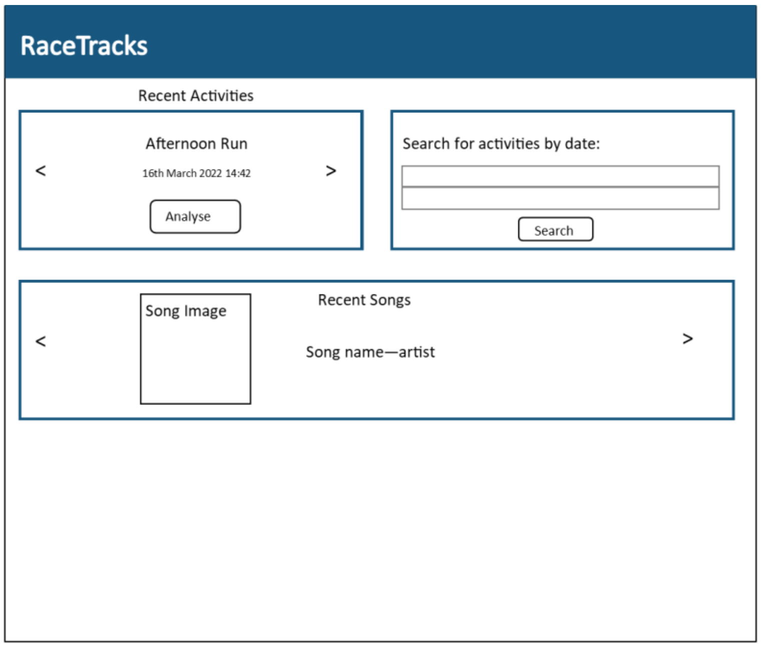


Figure 11: Wireframe - Desktop view of home page

The aim with these designs was to keep the view simplistic and provide the user with relevant data, without overwhelming them with too much unnecessary information. The three sections on the page provide the user with three different options, they can either scroll their recent activities and choose one to analyse, search for an activity on a specific date to analyse or they can view their recently listened to songs.

The outcome of the final solution can be seen below in Figure 13 and Figure 14. The design of the home page stayed very true to the original wireframes, with new additions such as displaying the users connected account usernames along the navigation bar with log out buttons, and I also chose to introduce a logo alongside the website name. The different views for mobile and desktop were made possible through the use of CSS media breakpoints which allow for declaring what styles an element should have on screens with a certain number of pixels.

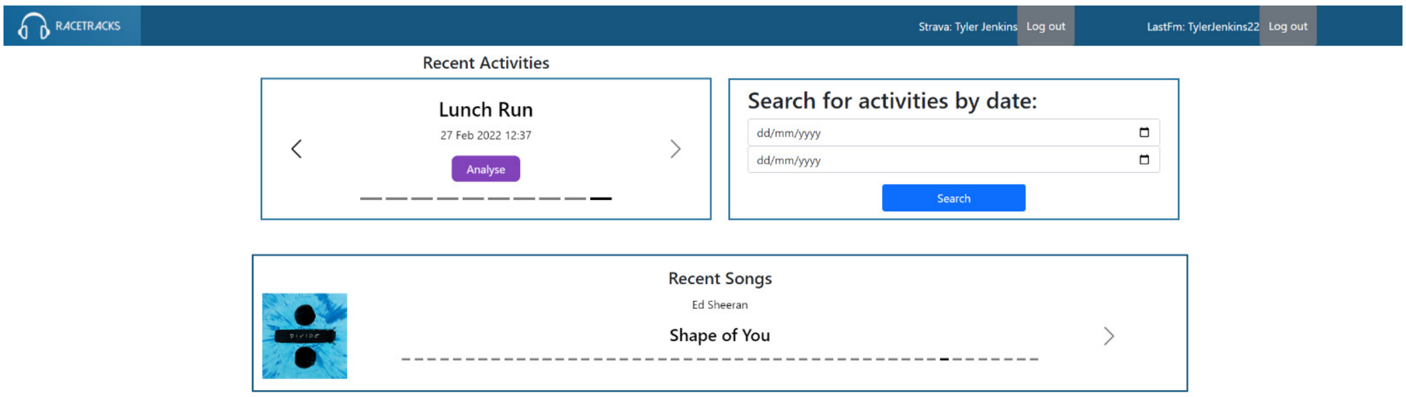


Figure 13: Final Solution - Desktop Home Page

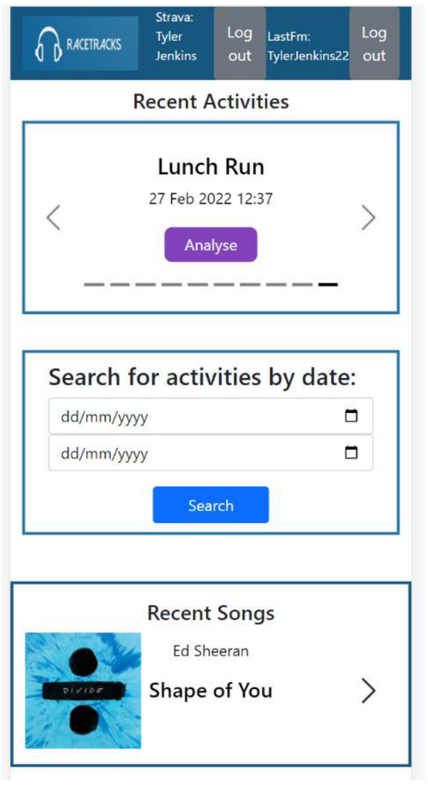


Figure 14: Final Solution - Mobile Home Page

3.2.3 Analyse Activity Page

Another very important design which was created for the solution was the “Analyse Activity” page. This is an important page which displays the breakdown analysis of a user’s performance to them. Before designing these wireframes, I had the idea of using Strava’s segments to analyse a user’s performance and therefore the UI for the “Analyse Activity” page was designed surrounding this idea. Figure 15 and Figure 16 below show the wireframes for both the desktop and mobile version of this page.

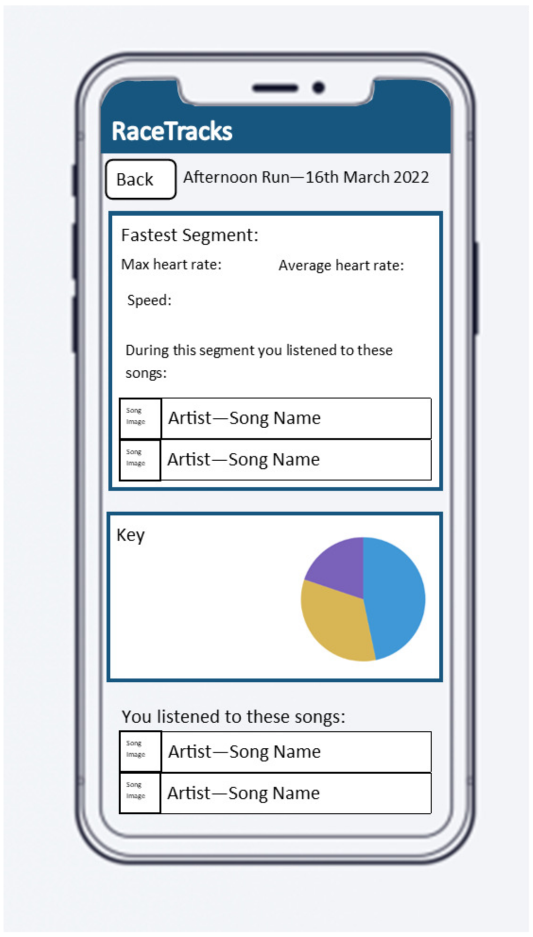


Figure 15: Wireframe - Mobile view of Analyse Activity Page

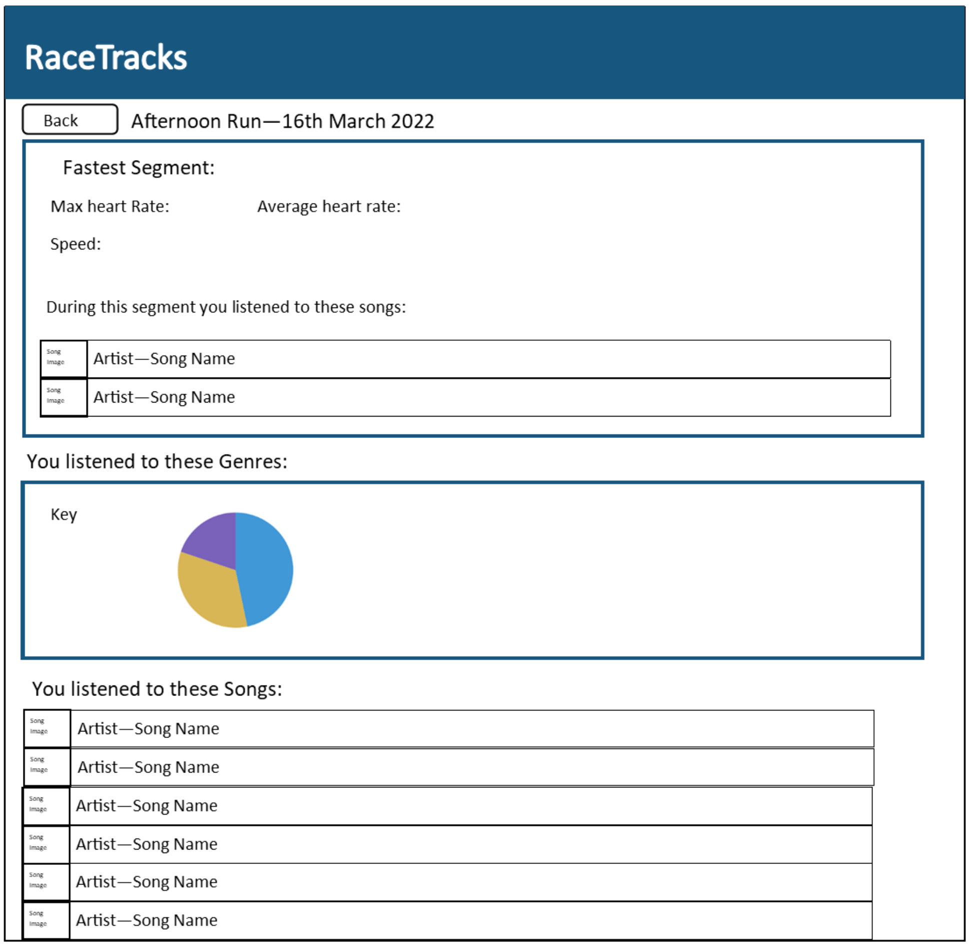
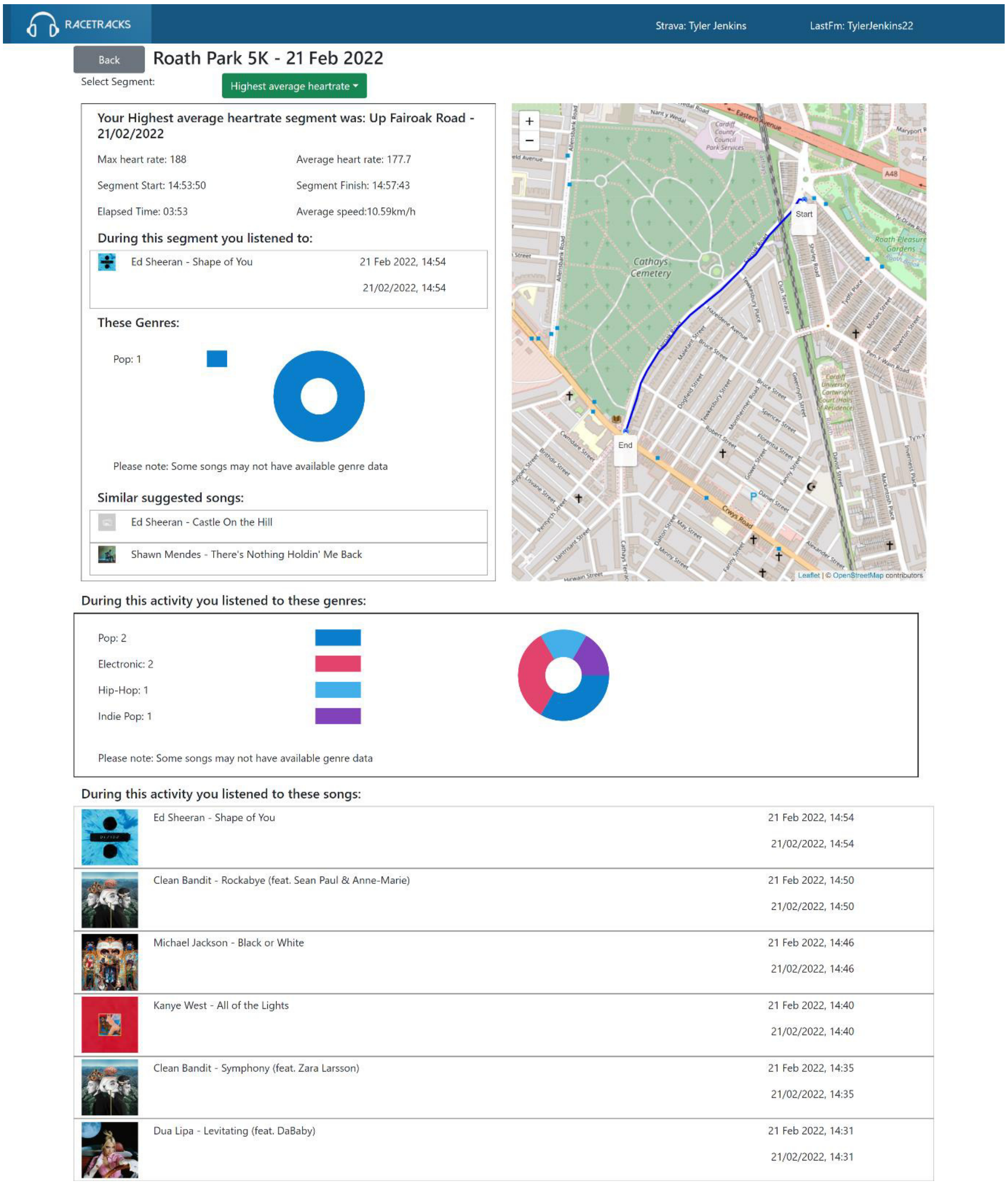


Figure 16: Wireframe - Desktop view of Analyse Activity Page

The aim with these designs was to provide the user with information on their activity and the music they have listened to. I tried to keep the theme of the home page by keeping the page split into 3 different sections, starting from the top of the page with the chosen segment data, moving to the pie chart displaying what genres the user listened to during that activity and then lastly the list of the songs the user listened to. Shown below in Figure 17, Figure 18, Figure 19 and Figure 20 are the completed solutions for the desktop and mobile view for the Analyse Activity page.



Roath Park 5K - 21 Feb 2022

Select Segment:

Your Highest average heartrate segment was: Up Fairoak Road - 21/02/2022

Max heart rate: 188	Average heart rate: 177.7
Segment Start: 14:53:50	Segment Finish: 14:57:43
Elapsed Time: 03:53	Average speed:10.59km/h

During this segment you listened to:

Ed Sheeran - Shape of You	21 Feb 2022, 14:54 21/02/2022, 14:54

These Genres:

Pop: 1 🟦

Please note: Some songs may not have available genre data

Similar suggested songs:

Ed Sheeran - Castle On the Hill

Shawn Mendes - There's Nothing Holdin' Me Back

During this activity you listened to these genres:

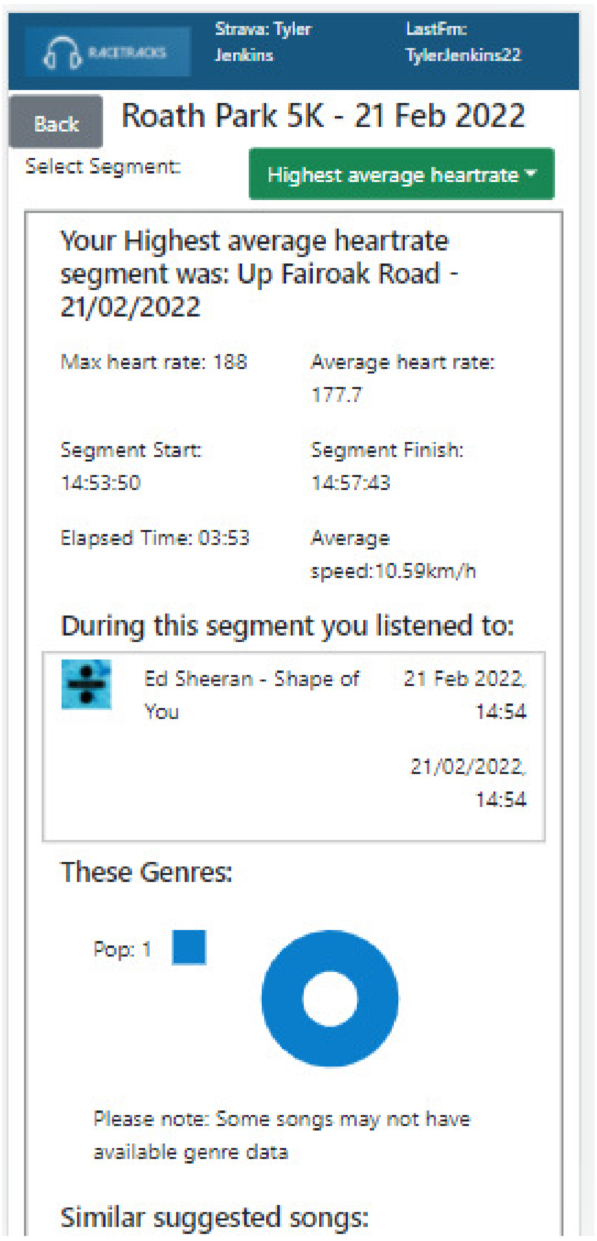
Pop: 2	🟦
Electronic: 2	🟥
Hip-Hop: 1	🟦
Indie Pop: 1	🟪

Please note: Some songs may not have available genre data

During this activity you listened to these songs:

Ed Sheeran - Shape of You	21 Feb 2022, 14:54 21/02/2022, 14:54
Clean Bandit - Rockabye (feat. Sean Paul & Anne-Marie)	21 Feb 2022, 14:50 21/02/2022, 14:50
Michael Jackson - Black or White	21 Feb 2022, 14:46 21/02/2022, 14:46
Kanye West - All of the Lights	21 Feb 2022, 14:40 21/02/2022, 14:40
Clean Bandit - Symphony (feat. Zara Larsson)	21 Feb 2022, 14:35 21/02/2022, 14:35
Dua Lipa - Levitating (feat. DaBaby)	21 Feb 2022, 14:31 21/02/2022, 14:31

Figure 17: Screenshot showing desktop view of Analyse Activity page



RACETRACKS Strava: Tyler Jenkins LastFm: TylerJenkins22

Back Roath Park 5K - 21 Feb 2022

Select Segment: Highest average heartrate ▾

Your Highest average heartrate segment was: Up Fairoak Road - 21/02/2022

Max heart rate: 188 Average heart rate: 177.7

Segment Start: 14:53:50 Segment Finish: 14:57:43

Elapsed Time: 03:53 Average speed:10.59km/h

During this segment you listened to:

Ed Sheeran - Shape of You 21 Feb 2022, 14:54 21/02/2022, 14:54

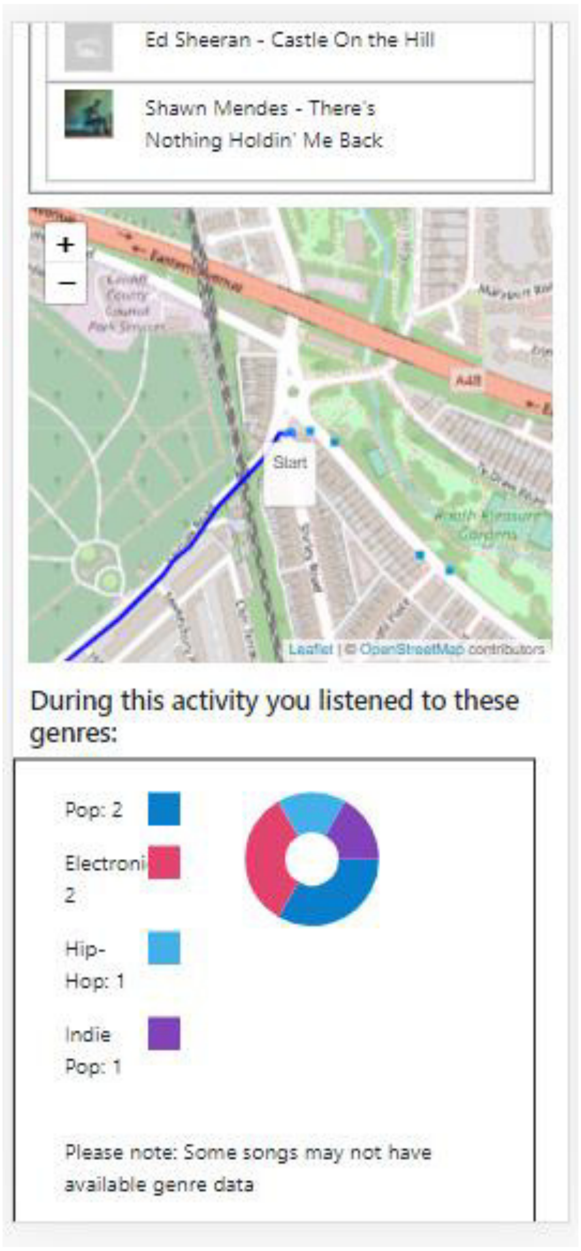
These Genres:

Pop: 1 🟦

Please note: Some songs may not have available genre data

Similar suggested songs:

Figure 19: Screenshot showing first section of mobile view of Analyse Activity Page



Ed Sheeran - Castle On the Hill

Shawn Mendes - There's Nothing Holdin' Me Back

During this activity you listened to these genres:

Pop: 2 🟦

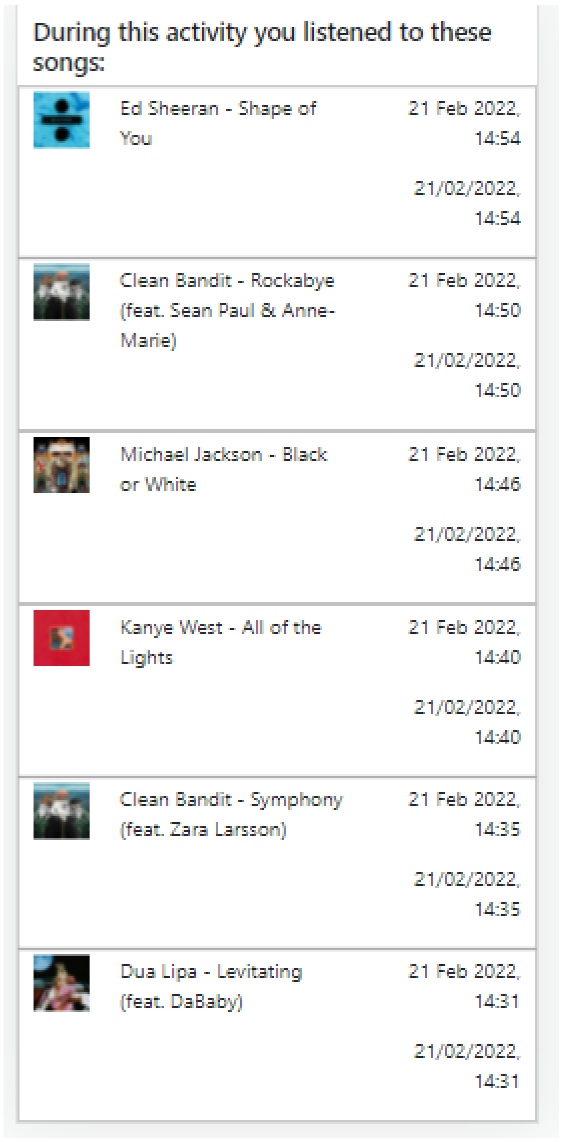
Electronic: 2 🟥

Hip-Hop: 1 🟦

Indie Pop: 1 🟪

Please note: Some songs may not have available genre data

Figure 18: Screenshot showing second section of mobile view of Analyse Activity Page



During this activity you listened to these songs:

	Ed Sheeran - Shape of You	21 Feb 2022, 14:54 21/02/2022, 14:54
	Clean Bandit - Rockabye (feat. Sean Paul & Anne-Marie)	21 Feb 2022, 14:50 21/02/2022, 14:50
	Michael Jackson - Black or White	21 Feb 2022, 14:46 21/02/2022, 14:46
	Kanye West - All of the Lights	21 Feb 2022, 14:40 21/02/2022, 14:40
	Clean Bandit - Symphony (feat. Zara Larsson)	21 Feb 2022, 14:35 21/02/2022, 14:35
	Dua Lipa - Levitating (feat. DaBaby)	21 Feb 2022, 14:31 21/02/2022, 14:31

Figure 20: Screenshot showing final section of mobile view of Analyse Activity Page

As can be seen in the figures above, the final designs stayed mostly true to the original wireframes. The standout difference of course being that there is now a map showing the chosen segment on the page. Upon implementing the other features on the page, the page felt as though something was missing. Being an application, which dealt heavily with location data, a map view seemed like a necessary and welcomed addition to the project. The other addition being a pie chart upon the segment showing the genre of songs listened to during that chosen segment.

There are more wireframes which were created for the final solution, all of which can be seen in the appendix of this report.

3.2 Dynamic Behaviour and Data Flows

There are several important data flows which take place in the system. The first data flow which takes place is the Strava user authorization flow. For a user to connect their Strava account to RaceTracks, they must be redirected to the Strava website, log in and then authorize RaceTracks to access their data. Strava uses OAuth2 authentication for their API (27). OAuth2 allows external applications to request authorization to a user's data to allow users to grant access to their data on a per-application basis. Shown below in Figure 21 is the exact data flow which takes place to obtain access to a user's Strava data in RaceTracks.

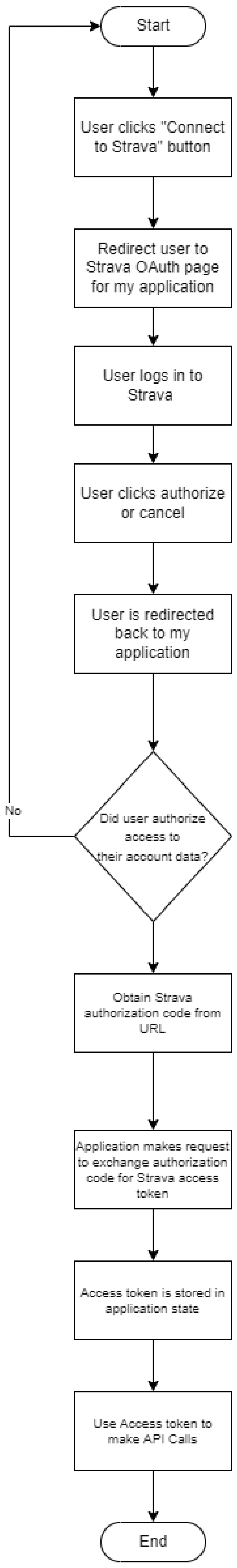

Start
User clicks "Connect to Strava" button
Redirect user to Strava OAuth page for my application
User logs in to Strava
User clicks authorize or cancel
User is redirected back to my application
Did user authorize access to their account data?
No
Obtain Strava authorization code from URL
Application makes request to exchange authorization code for Strava access token
Access token is stored in application state
Use Access token to make API Calls
End

Figure 21: UML Diagram showing Strava OAuth Data Flow

After this OAuth data flow has successfully completed, RaceTracks has access to a user's Strava data and is then able to make API calls to their account data using their unique access token. With the user's data that is returned from the Strava API, an object called "stravaUserData" is created which stores all the users account information. Figure 22 below shows the created stravaUserData object. There are a number of fields included by Strava which RaceTracks does not make use of currently, but they are also added to the object by default and could potentially be used in any future work.

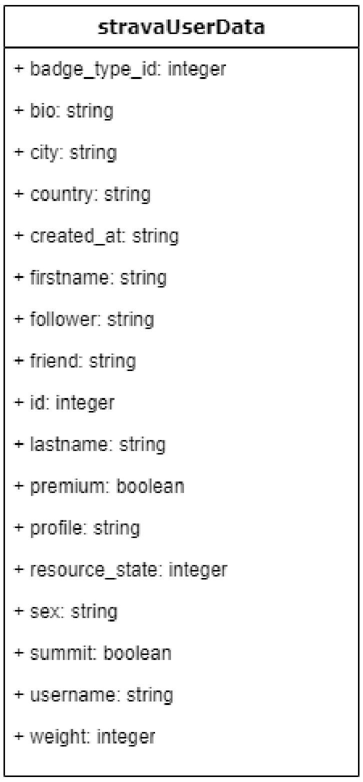


Figure 22: UML Diagram showing stravaUserData Object

In addition to the Strava authentication flow, there is a data flow which is needed to get the users Last.FM account data. This process is only allowed to occur once a user has successfully connected a Strava account. It is important that the dataflow for obtaining a users'Last.FM account runs after the Strava authentication has been successful. This is because authorizing a users' Strava account requires a redirect to another website, whereas obtaining their Last.FM data does not. If we were to obtain the users' Last.FM data before their Strava data, upon the Strava redirect, their account data would be lost by RaceTracks. This is because the account data is currently stored as a variable and is not cached or stored in a cookie.

Shown below in Figure 23 is a UML diagram of the Last.FM data flow for connecting a user's account to the system.

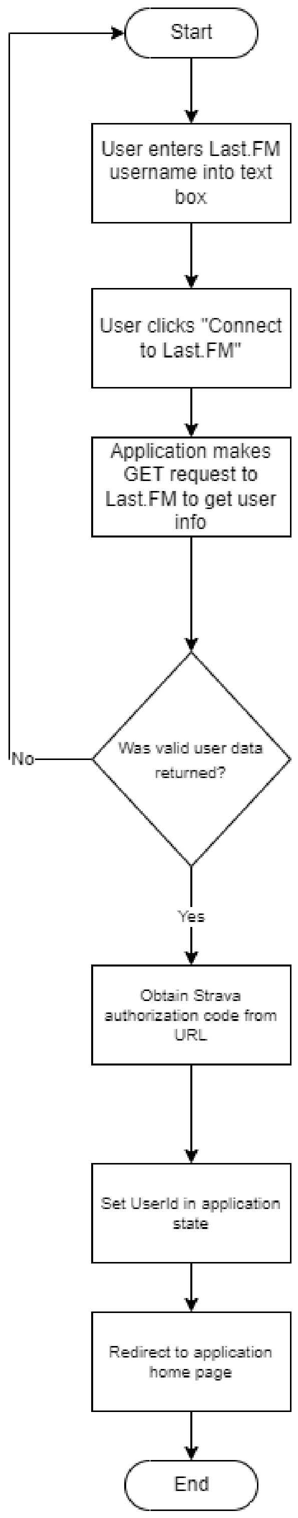

Start
User enters Last.FM username into text box
User clicks "Connect to Last.FM"
Application makes GET request to Last.FM to get user info
Was valid user data returned?
No
Yes
Obtain Strava authorization code from URL
Set UserId in application state
Redirect to application home page
End

Figure 23: UML Diagram showing data flow for connecting a Last.FM account

Following the response from this data flow, a “lastFmUserData” object is created with the data returned from this process. Figure 24 shows a UML diagram of this object.

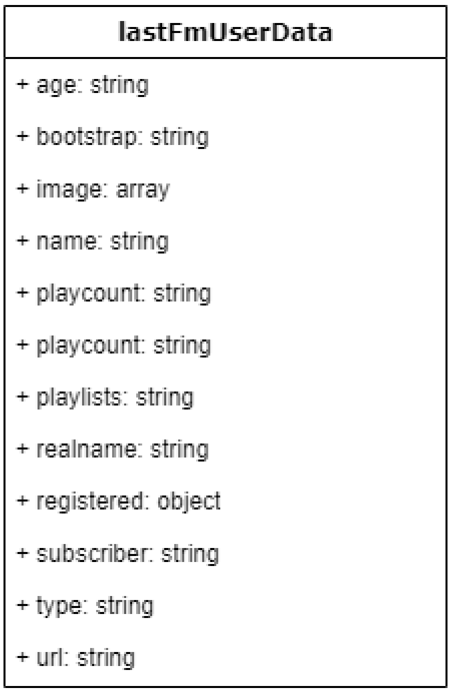


Figure 24: UML Diagram showing created lastFmUserData object

The final important behaviour and flow of data within the system is how the user can interact with the system. Figure 25 below shows a very high-level overview of the interactions a user can make with the system, and how the data flows through to each section.

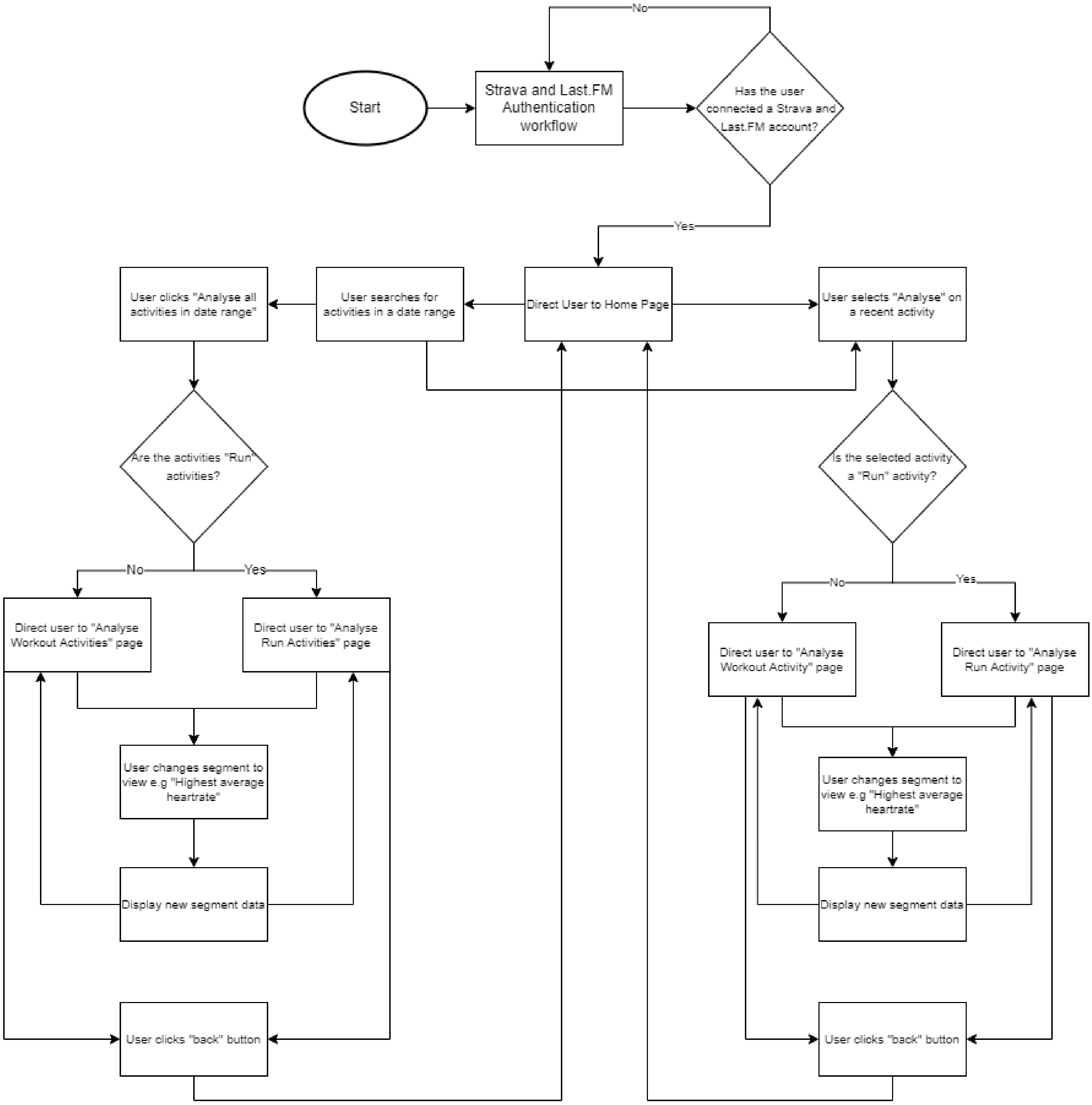

Start
Strava and Last.FM Authentication workflow
Has the user connected a Strava and Last.FM account?
No
Yes
Direct User to Home Page
User searches for activities in a date range
User clicks "Analyse all activities in date range"
User selects "Analyse" on a recent activity
Are the activities "Run" activities?
No
Yes
Direct user to "Analyse Workout Activities" page
Direct user to "Analyse Run Activities" page
User changes segment to view e.g "Highest average heartrate"
Display new segment data
User clicks "back" button
Is the selected activity a "Run" activity?
No
Yes
Direct user to "Analyse Workout Activity" page
Direct user to "Analyse Run Activity" page
User changes segment to view e.g "Highest average heartrate"
Display new segment data
User clicks "back" button

Figure 25: UML Activity Diagram showing high level overview of user interaction with the system

3.3 System Architecture

As the project was developed using solely JavaScript and React, it was very important to outline a suitable system architecture. The structure of the code base and the partitioning of the modules are set up in a way to encourage easily understandable, maintainable, and scalable code.

The top level of the project repository contains several files and folders, these include:

- **(Folder): Public**
  - Contains web configuration files and website favicon.
- **(Folder): Src**
  - Source code folder; contains image assets, JavaScript files and CSS files.
- **(File): .env**
  - A top-level file used for declaring environment wide variables such as API keys, URLs, and client ID's.
- **(File): package.json**
  - A top-level file used for declaring all the package dependencies required for building and running the solution.

Of the files and folders outlined above, the "Src" folder is the one which contains the written code which produced the solution. This folder is organised as follows:

- **(Folder): Assets**
  - Contains image assets used.
- **(Folder): Components**
  - Important folder containing all the JavaScript files and their corresponding CSS files.
- **(Folder): Utils:**
  - Contains two files:
    - StravaFunctions.js
    - LastFmFunctions.js
  - These files contain reusable functions for calling the different endpoints for each of the used API's.
- **(File): App.js**
  - The main component file containing routing logic which decides which components to show to the user.
- **(File) App.css**
  - The CSS file for the App.js file.
- **(File): Index.js**
  - The top-level file of the web-application which is rendered by default by the website.

Of the files and folders outlined above, the "Components" folder is of the most interest. As previously stated, this is the main folder containing all of the JavaScript source code for the RaceTracks site. This folder is broken down into separate folders for each unique component. Each unique folder contains a JavaScript file for the component and a relating CSS file which styles it. This helps keep code separated and easily manageable. The high-level structure of the "Components" folder looks as follows:

Components:

- **(Folder): Activities**
  - Contains folders for displaying recent activity components and search activity components.
- **(Folder): AnalyseActivities**
  - Contains several folders for different components used for analysing activities.
- **(Folder): ConnectAccounts**
  - Contains the logic for setting up and rendering the connection components for users Strava and Last.FM accounts.
- **(Folder): ConnectLastFm**
  - Contains the JavaScript component file used to allow a user to connect a Last.FM account and its CSS file.
- **(Folder): ConnectStrava**
  - Contains the JavaScript component file used to allow a user to connect a Strava account and its CSS file.
- **(Folder): Home**
  - Contains the JavaScript and CSS files for rendering the Home page component.
- **(Folder): NavBar**
  - Contains the JavaScript and CSS files for rendering the NavBar component.
- **(Folder): SegmentEfforts**
  - Contains JavaScript and CSS files for rendering segment effort components.
- **(Folder): Songs**
  - Contains several folders for different components used in displaying song data. This includes “SongList”, “GenrePieChart” and “RecentSongs”.
- **(Folder): StravaMap**
  - Includes JavaScript and CSS files for rendering the map displaying segment data.

This file structure allows the files to be easily found and easily managed.

4. Implementation

This section of the report aims to give a more detailed insight into the implementation of the solution. We will look at some of the important functions within the code which are especially critical to the operation of the system, as well as discuss some of the difficulties faced during the development process due to various factors such as complexity or lack of documentation.

4.1 External API Limitations

The external APIs from Strava and Last.FM both employ rate limits; this means that there is a limit to the number of requests that any application can make to the services each day. For Strava, this rate limit is set at 100 requests every 15 minutes, with an allowance up to 1000 requests every day (28). For Last.FM there is no particular set limit, however under their terms of service which must be agreed to in order to use the API, it states that applications should not put the API under excessive strain, and they employ a “Daily Reasonable Usage Cap” of 100MB, and any applications abusing their API will have their API access terminated (29). The Last.FM API also only permits 1 request per second. For the scope and scale of this project, there is no worry that the limits of the APIs will become an issue. However, if the application were to have a large user base in the future, then licensing to upgrade the applications Strava API call limit would need to be obtained.

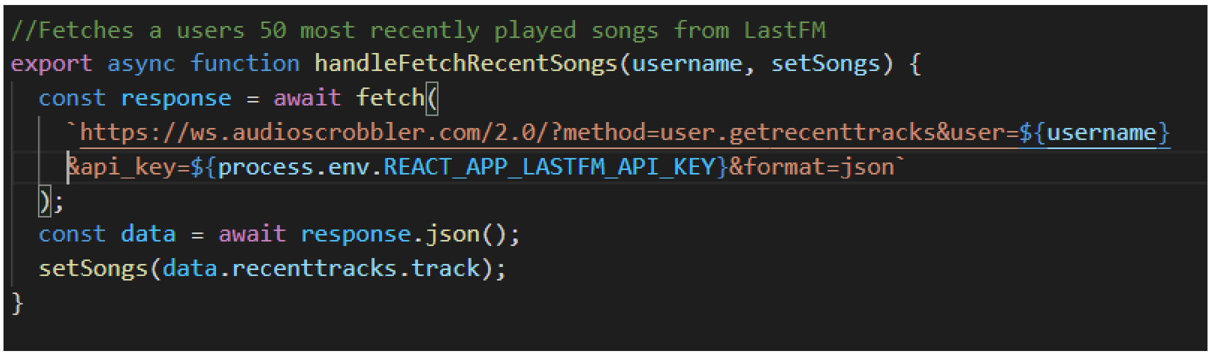
Although these limits are not an issue for the live production build of the application, I did encounter some issues with them when developing. Whilst implementing new features and testing how different solutions would work, I was routinely making API calls by refreshing the page an excessive number of times. There were several occasions where I unintentionally made 100 requests to the Strava API within a 15-minute window, which led to my application being blocked from making requests until the next 15-minute window. This was not too detrimental to development as I would only have to wait a few minutes before I could continue. The bigger issue came when I unintentionally reached the daily limit of 1000 API calls. This happened on a few occasions when I was trying to implement new features and would accidentally create an endless loop which caused API calls to be made repeatedly. This would cause me to reach the 1000 call limit within seconds and led to the RaceTracks application being locked out of the Strava API for the rest of the day. This did cause some setbacks as I was then unable to continue with development on those days, since the application relied on the data from Strava to function. In hindsight, if I were to work on a similar project again, I would use a tool such as JSON server (30) to overcome situations like this. JSON server allows you to set up an offline, “Fake” API, with endpoints and seeded data which an application running in development mode on localhost can make requests to. Using this tool, I could’ve set up a dummy version of the Strava API with my own data to make requests to, allowing me to bypass the issue of call limits during development.

There were also limitations to the Last.FM API in terms of the amount of data they provide for songs. Song data returned from Last.FM is lacking things such as genre data and beats per minute (BPM) for a given song. To try and overcome this issue, RaceTracks makes use of, and external API called TheAudioDb which acts as a community run database for song data. The data returned from TheAudioDb includes genre data for most songs, however it does have limitations itself, as some songs which may have only just released or are not well-known, may not exist within the database.

4.2 Asynchronous Functions and Promises

One of the biggest challenges faced during development was dealing with making API requests to various external API's. For the system to function and load correctly, it must have complete sets of the user's data from both Strava and Last.FM. During the runtime of the solution, when the user is interacting in different ways with the system such as choosing to analyse a specific activity or a range of activities, then new data must be called from 3 separate API's: Strava, Last.FM and TheAudioDb. These API calls need to happen in a very specific order otherwise the system will error. Upon analysing an activity, firstly the Strava API needs to be called to get the data for that activity. Once the data has been returned for this activity, then the call to Last.FM can be made with the date and time-period of the activity to get the songs listened to during that time stamp. Once that call has completed and the data from Last.FM has been returned, we can make the call to TheAudioDb to get the genre data for the songs listened to. It is crucial to the operation of the system that these functions happen in that order, otherwise the system will fail and try to make calls with data it does not have or display data that does not yet exist.

At its base, JavaScript is a synchronous and single-threaded language (31), this means that functions are run line by line one after another, which can cause some issues when we are trying to retrieve data from an API which is not returned instantly. The "Fetch" API method in JavaScript is used to access resources across a network using HTTP requests, which acts as an asynchronous method. This means that when this line of code is run, the rest of the code that follows it will continue to be executed whilst this process completes in the background (32). This can cause some problems when trying to run two asynchronous methods one after another when the second one relies on the data that would be achieved from the first, because it will be missing. To overcome this issue, the use of JavaScript promises is required. A promise is a temporary representation of a completion of a request. They can be in one of three states; fulfilled, pending, or rejected. Using promises, we can make asynchronous code work in a synchronous way. Figure 26 below shows an example code snippet of an asynchronous function making use of these promises.

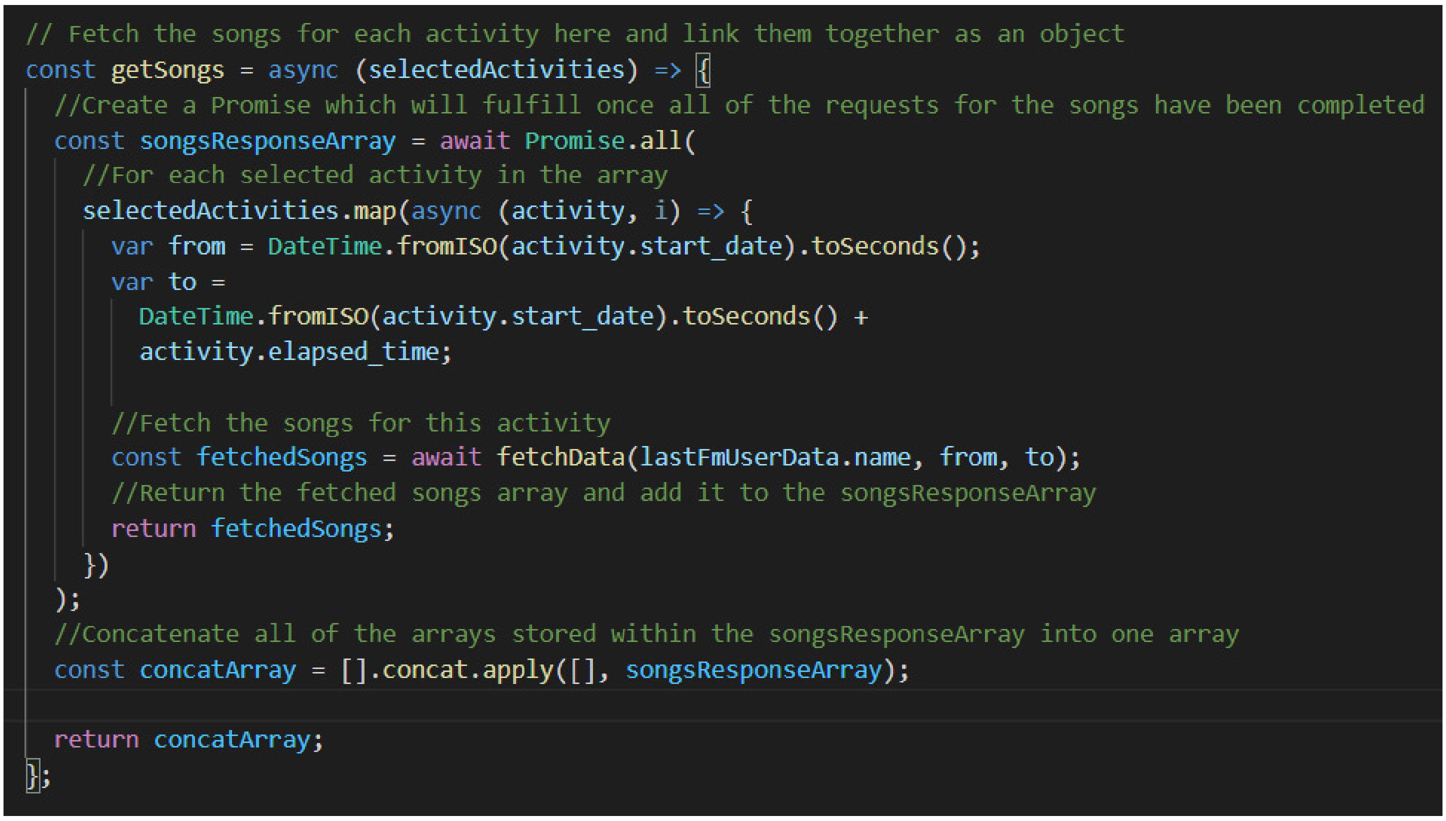


```
//Fetches a users 50 most recently played songs from LastFM
export async function handleFetchRecentSongs(username, setSongs) {
  const response = await fetch(
    `https://ws.audioscrobbler.com/2.0/?method=user.getrecenttracks&user=${username}
    &api_key=${process.env.REACT_APP_LASTFM_API_KEY}&format=json`
  );
  const data = await response.json();
  setSongs(data.recenttracks.track);
}
```

Figure 26: Code Snippet showing asynchronous function that fetches a user's recent songs in LastFmFunctions.js file

The "await" statements tell the code to wait for a fulfilled promise response before continuing. So, in this example, which is getting a user's recent songs, we "await" the fetch method's response from Last.FM, then once the response has been fulfilled, we extract this as JSON to the "data" variable, and then we can successfully set the songs. This method of using async functions and await statements are commonly used throughout this solution and will be spoken about again in further sections.

In addition to these basic asynchronous functions, I also had to utilise asynchronous mapping functions. These functions can quite quickly become a lot more complex. The underlying idea of this is that we need to make lots of asynchronous calls to an API, but we want the code to wait until **all** of these requests have been completed successfully before continuing, rather than just waiting for one call to finish like we see above in Figure 26. Shown below in Figure 27 is an example of one of the methods within the system which makes use of these asynchronous mapping promises.



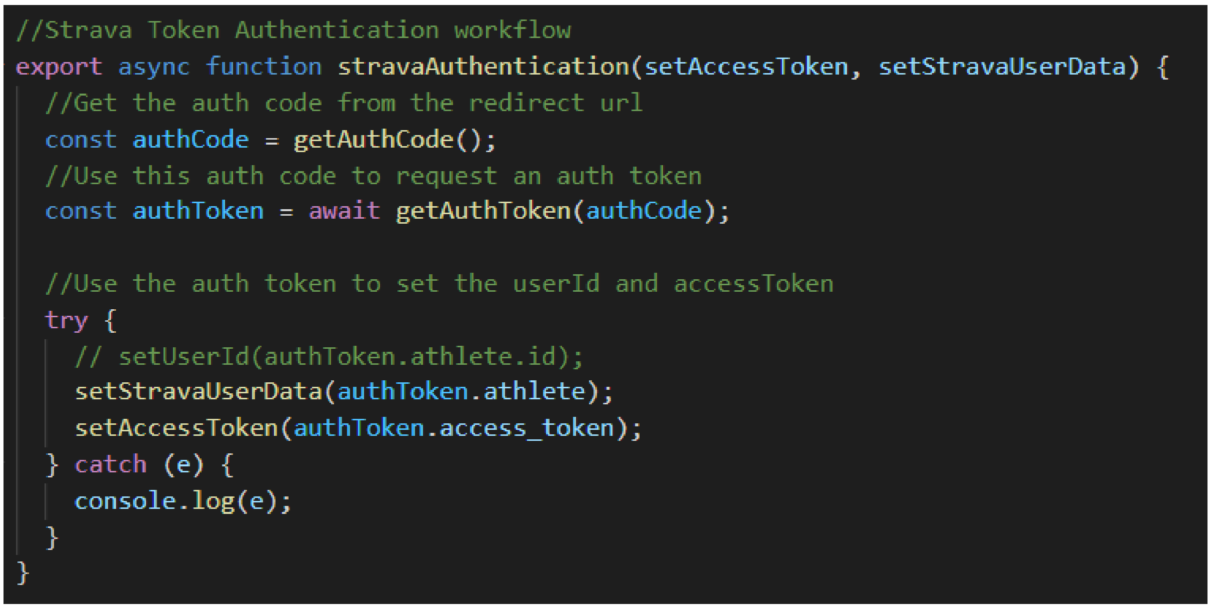
```
// Fetch the songs for each activity here and link them together as an object
const getSongs = async (selectedActivities) => {
  //Create a Promise which will fulfill once all of the requests for the songs have been completed
  const songsResponseArray = await Promise.all(
    //For each selected activity in the array
    selectedActivities.map(async (activity, i) => {
      var from = DateTime.fromISO(activity.start_date).toSeconds();
      var to =
        DateTime.fromISO(activity.start_date).toSeconds() +
        activity.elapsed_time;

      //Fetch the songs for this activity
      const fetchedSongs = await fetchData(lastFmUserData.name, from, to);
      //Return the fetched songs array and add it to the songsResponseArray
      return fetchedSongs;
    })
  );
  //Concatenate all of the arrays stored within the songsResponseArray into one array
  const concatArray = [].concat.apply([], songsResponseArray);

  return concatArray;
};
```

Figure 27: Code Snippet showing asynchronous mapping function for getting songs for multiple activities

This method is used when analysing a group of activities over a date range. It takes a list of activities as a parameter, and for each of these activities we need to find the list of songs that were listened to. As finding the songs for each activity needs to be done in separate API requests, we employ the asynchronous mapping method as shown above. The “await Promise.all” declaration says for the code to wait for **all** of the asynchronous calls within the promise to be fulfilled before continuing (33). This ensures that all of the calls will be completed, and the code will not try to continue without the data it needs.

4.3 Strava Authentication

The high-level data flow of how the Strava Authentication method works was previously discussed in section 3.2 and shown in Figure 21. Here we will discuss in finer detail the code which handles this process. Following the user being redirected to Strava and them logging in and providing RaceTracks with access to their profile, they are redirected back to the RaceTracks home page where the following functions shown in the figures below, run on page load:

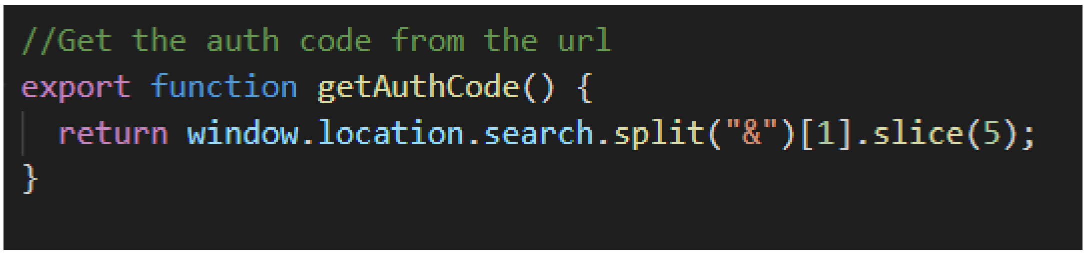


```
//Strava Token Authentication workflow
export async function stravaAuthentication(setAccessToken, setStravaUserData) {
  //Get the auth code from the redirect url
  const authCode = getAuthCode();
  //Use this auth code to request an auth token
  const authToken = await getAuthToken(authCode);

  //Use the auth token to set the userId and accessToken
  try {
    // setUserId(authToken.athlete.id);
    setStravaUserData(authToken.athlete);
    setAccessToken(authToken.access_token);
  } catch (e) {
    console.log(e);
  }
}
```

Figure 28: Code Snippet showing Strava Authentication Function in StravaFunctions.js file

Figure 28 shows a very important function in the solution. It is the function that handles the data for the user's Strava account authentication. This function needs to be asynchronous as it contains fetch requests to another resource over HTTPS and so we make use of the "await" statements for the code to run in a seemingly synchronous way. The "stravaAuthentication" function calls two other functions within itself, one being the "getAuthCode" function which can be seen below in Figure 29 and the other being the "getAuthToken" function which can be seen below in Figure 30.

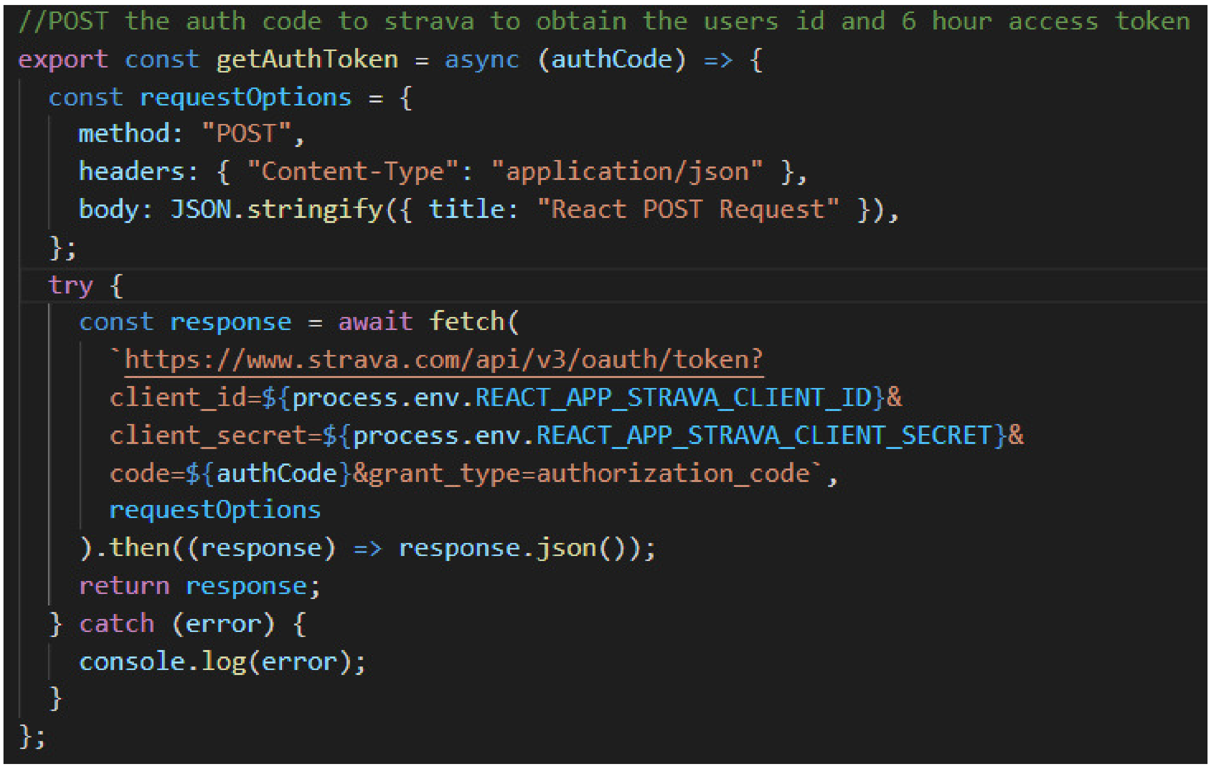
Once the user is redirected back to the application from Strava, the auth code we need is placed into the URL. Shown below in Figure 29 is a simple function which takes the current URL and extracts the auth code from it so that we can make use of it.



```
//Get the auth code from the url
export function getAuthCode() {
  return window.location.search.split("&")[1].slice(5);
}
```

Figure 29: Code Snippet showing function to get the returned Strava Auth code from the URL in StravaFunctions.js file

The last function called within the "stravaAuthentication" function is the "getAuthToken" function which can be seen below in Figure 30. This function is also asynchronous as it makes a POST request to the Strava API with the users auth code to obtain a valid authorization token which can be used to then make further API calls. The endpoint is called with the RaceTracks Strava client id, client secret and the auth code which we obtained previously from the redirect URL.



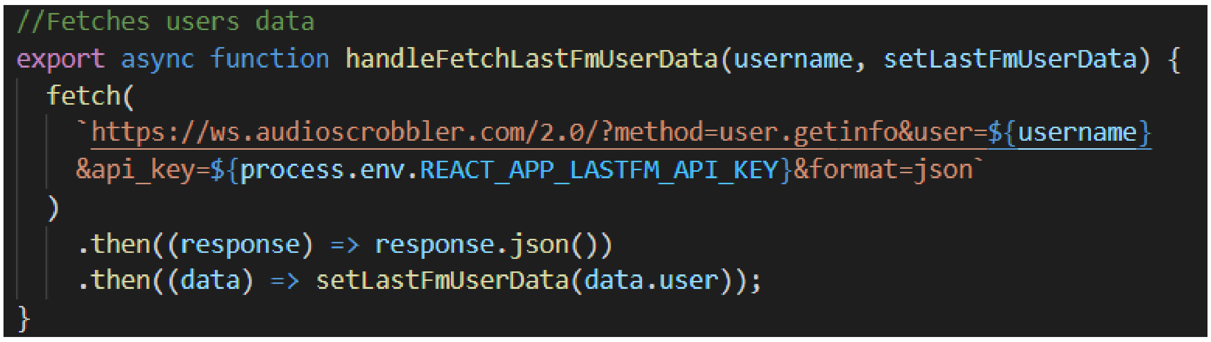
```
//POST the auth code to strava to obtain the users id and 6 hour access token
export const getAuthToken = async (authCode) => {
  const requestOptions = {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ title: "React POST Request" }),
  };
  try {
    const response = await fetch(
      `https://www.strava.com/api/v3/oauth/token?
      client_id=${process.env.REACT_APP_STRAVA_CLIENT_ID}&
      client_secret=${process.env.REACT_APP_STRAVA_CLIENT_SECRET}&
      code=${authCode}&grant_type=authorization_code`,
      requestOptions
    ).then((response) => response.json());
    return response;
  } catch (error) {
    console.log(error);
  }
};
```

Figure 30: Code Snippet showing get Strava Auth Token Function in StravaFunctions.js file

Once this method has successfully run, then the users userId and access token is available to RaceTracks, and it can then make further API requests to retrieve user data using the userId and the access token.

4.4 Last.FM User Data Retrieval

In contrast to acquiring the users Strava data, acquiring their Last.FM data is much more straight forward. Last.FM's user data is public facing and does not require user authentication in order to access it. Because of this there are no required redirects from the application, and all that we need from the user is their Last.FM username. Once the user inputs their username into the given text box and clicks the "Connect to Last.FM" button, the following function shown below in Figure 31 is run.



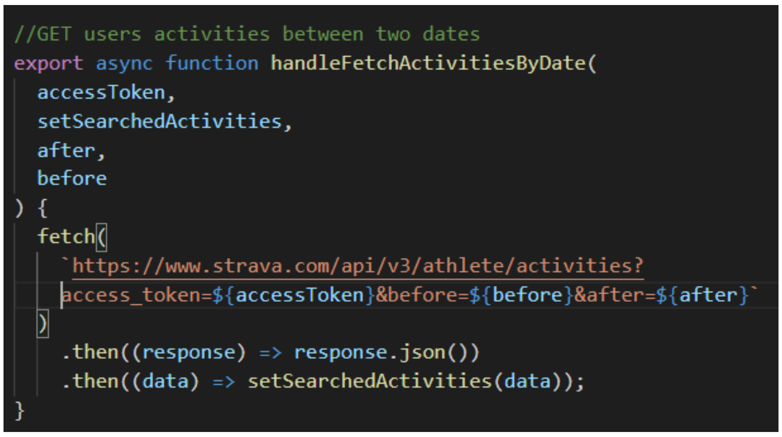
```
//Fetches users data
export async function handleFetchLastFmUserData(username, setLastFmUserData) {
  fetch(
    `https://ws.audioscrobbler.com/2.0/?method=user.getinfo&user=${username}
    &api_key=${process.env.REACT_APP_LASTFM_API_KEY}&format=json`
  )
    .then((response) => response.json())
    .then((data) => setLastFmUserData(data.user));
}
```

Figure 31: Code Snippet showing function for getting a user's Last.FM account data

This is a simple asynchronous function which makes an API GET request to Last.FM for the users account information. We can also then use this username to make all of the other API requests to Last.FM.

4.5 API Request Methods

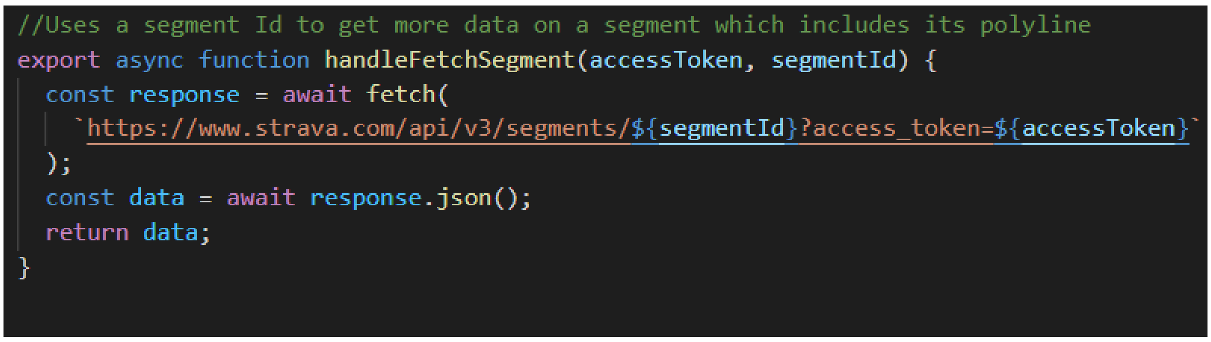
Within the system architecture there are two files stored within the “utils” folder. These two folders are the “LastfmFunctions,js” file and the “StravaFunctions.js” file. Both of these folders contain the methods for making the requests to both of the respective services APIs. Storing the methods here in this file separately allows them to be imported into other JavaScript files within the code base and reused in different ways. All of the fetch requests used, receive a JSON response which is then serialized before being manipulated and set as a variable within the system. Within the “StravaFunctions.js” file, some of the methods include “handleFetchActivities” which fetches a specified number of the users most recent activities. There is also a function for getting a users activities between a specified date range called “handleFetchActivitiesByDate”, which can be seen below in Figure 32.



```
//GET users activities between two dates
export async function handleFetchActivitiesByDate(
  accessToken,
  setSearchedActivities,
  after,
  before
) {
  fetch(
    `https://www.strava.com/api/v3/athlete/activities?
    access_token=${accessToken}&before=${before}&after=${after}`
  )
    .then((response) => response.json())
    .then((data) => setSearchedActivities(data));
}
```

Figure 32: Code Snippet showing handleFetchActivitiesByDate method

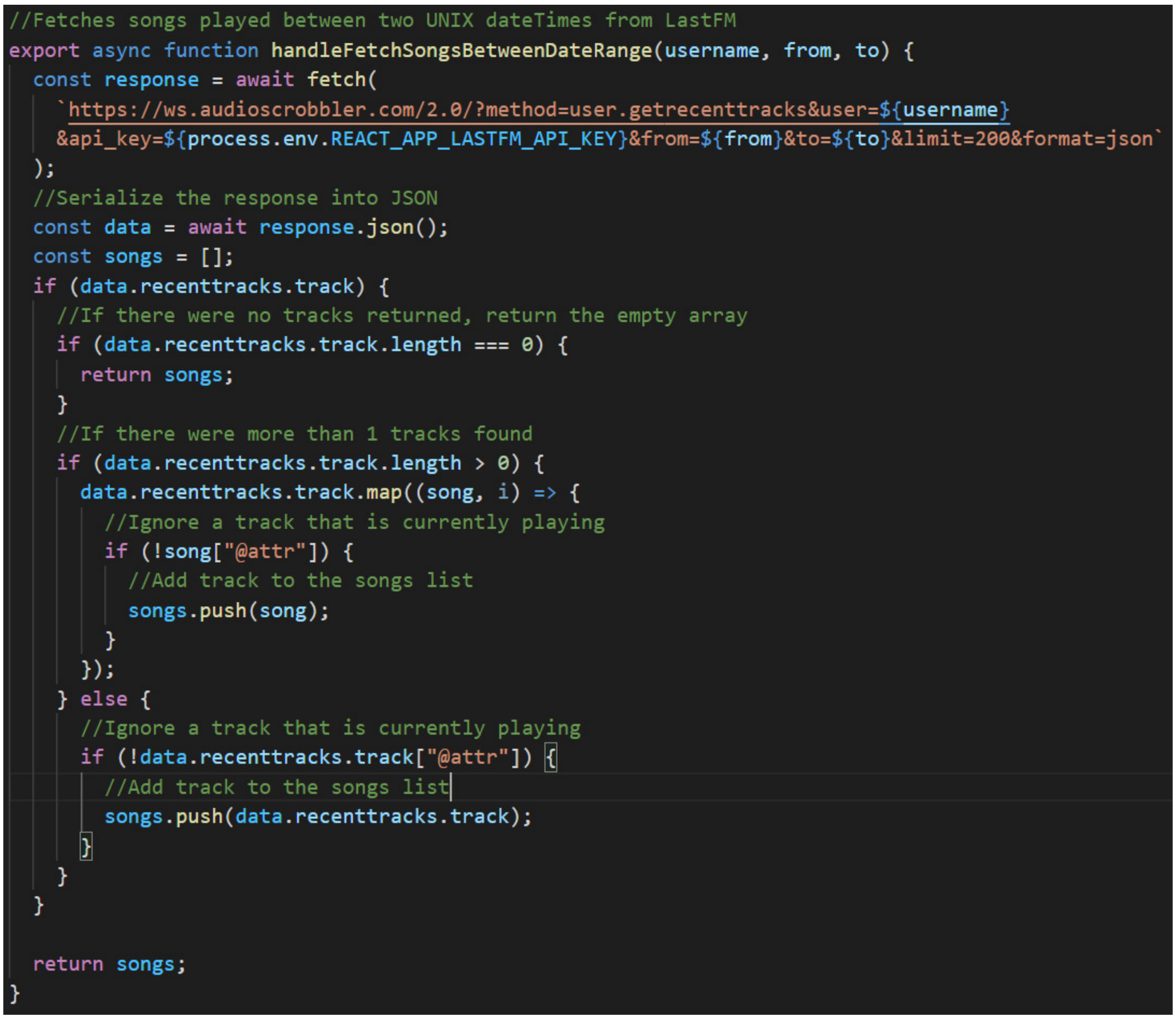
Another important method in the “StravaFunctions.js” file is the “handleFetchSegment” method which can be seen below in Figure 33. This is used for requesting more data on a specific segment of a run. This is necessary to retrieve the polyline data to be displayed on a map on the analyse activity page.



```
//Uses a segment Id to get more data on a segment which includes its polyline
export async function handleFetchSegment(accessToken, segmentId) {
  const response = await fetch(
    `https://www.strava.com/api/v3/segments/${segmentId}?access_token=${accessToken}`
  );
  const data = await response.json();
  return data;
}
```

Figure 33: Code Snippet showing handleFetchSegment method

As well as the methods for requesting data from the Strava API, there are similar functions which request data from the Last.FM API. The most important method here is the "handleFetchSongsBetweenDateRange" method, which as the name suggests, fetches all of the songs that the user listened to during a specified date range and can be seen below in Figure 34.



```javascript
//Fetches songs played between two UNIX dateTimes from LastFM
export async function handleFetchSongsBetweenDateRange(username, from, to) {
  const response = await fetch(
    `https://ws.audioscrobbler.com/2.0/?method=user.getrecenttracks&user=${username}
    &api_key=${process.env.REACT_APP_LASTFM_API_KEY}&from=${from}&to=${to}&limit=200&format=json`
  );
  //Serialize the response into JSON
  const data = await response.json();
  const songs = [];
  if (data.recenttracks.track) {
    //If there were no tracks returned, return the empty array
    if (data.recenttracks.track.length === 0) {
      return songs;
    }
    //If there were more than 1 tracks found
    if (data.recenttracks.track.length > 0) {
      data.recenttracks.track.map((song, i) => {
        //Ignore a track that is currently playing
        if (!song["@attr"]) {
          //Add track to the songs list
          songs.push(song);
        }
      });
    } else {
      //Ignore a track that is currently playing
      if (!data.recenttracks.track["@attr"]) {
        //Add track to the songs list
        songs.push(data.recenttracks.track);
      }
    }
  }

  return songs;
}
```

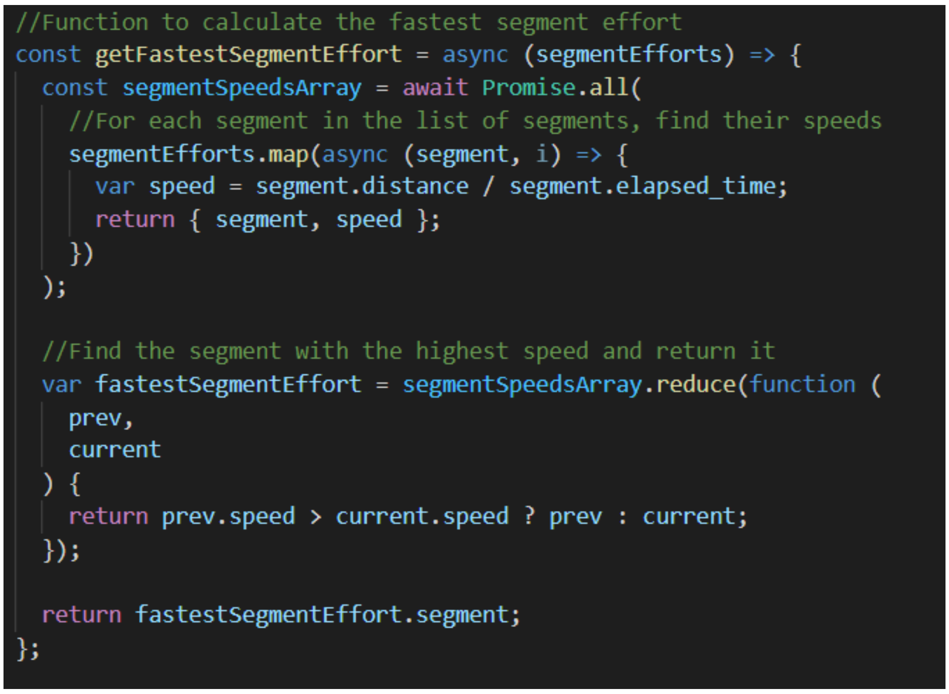
Figure 34: Code Snippet showing handleFetchSongsBetweenDateRange method

This is one of the most complex methods for fetching data within the system. This is because Last.FM has a quirk where if a user is currently listening to a song when the request is being made, it will always return that song regardless of what date range was selected. For example, if we were to request all of the songs listened to between the 1st of March 2022 and the 5th of March 2022 on the 30th of March, and we currently have a song playing, then we will get the list of songs between the 1st and 5th, as well as the current song playing. We need to account for this, and so extra logic is needed to find this song in the list and remove it before returning the list of songs.

There are a number more methods for requesting data from both Strava and Last.FM. All of which can be found as previously stated, within the codebase in the "StravaFunctions.js" and "LastFmFunctions.js" files in the utils folder.

4.6 Analysing Activities and Segment Efforts

One of the most important functions of the system is the Strava segment analysis functionality. This is where the users runs are analysed in segments to find the parts of their run where their heart rate was the highest or lowest, and where their pace is the highest or lowest. There are 4 functions in total, one for each of these options to calculate the specified segment from a list of given segments. Each of the functions takes in a list of segments as a parameter and runs the specific algorithm to find the segment that matches the chosen criteria. The algorithms for making the necessary calculations went through many iterations throughout the projects lifecycle before finally settling on the final methods. Figure 35 below shows a code snippet of the method used for calculating the fastest segment effort from the given list.

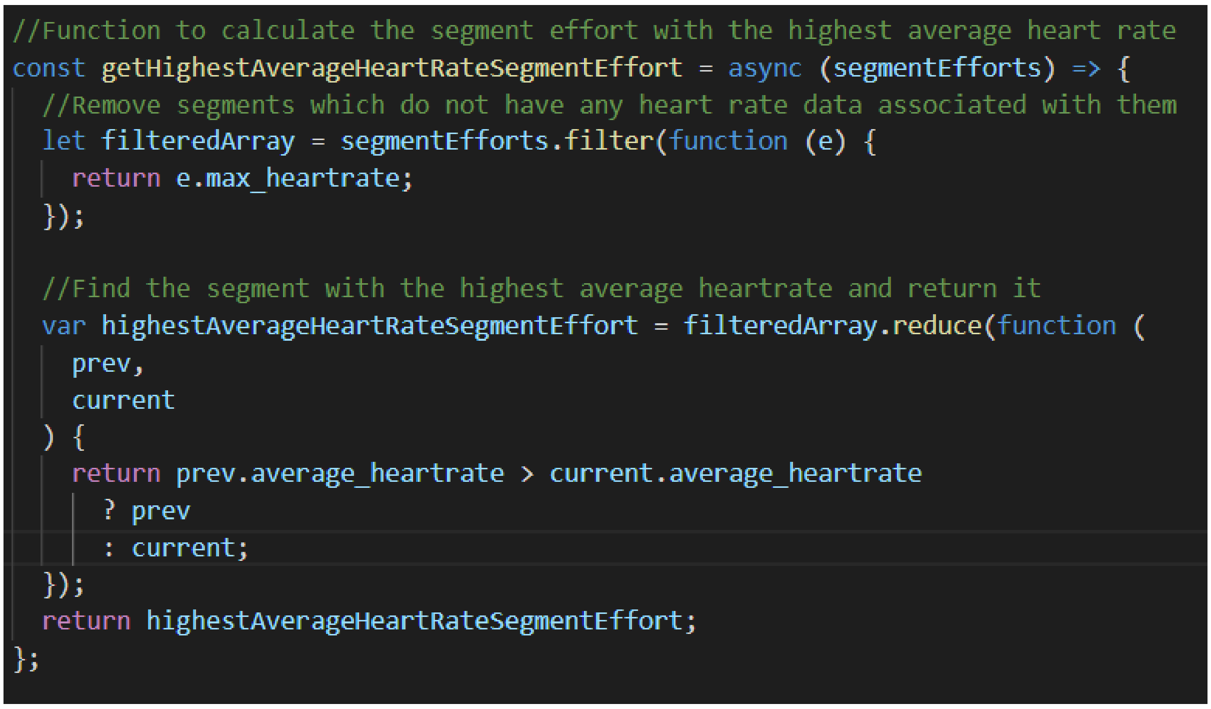


```
//Function to calculate the fastest segment effort
const getFastestSegmentEffort = async (segmentEfforts) => {
  const segmentSpeedsArray = await Promise.all(
    //For each segment in the list of segments, find their speeds
    segmentEfforts.map(async (segment, i) => {
      var speed = segment.distance / segment.elapsed_time;
      return { segment, speed };
    })
  );

  //Find the segment with the highest speed and return it
  var fastestSegmentEffort = segmentSpeedsArray.reduce(function (
    prev,
    current
  ) {
    return prev.speed > current.speed ? prev : current;
  });

  return fastestSegmentEffort.segment;
};
```

Figure 35: Code Snippet showing function for getting a User’s Fastest Segment Effort

The above function outlines the basis of the 4 different functions. Each of the functions is asynchronous and makes use of the mapping promises as discussed previously in section 4.2. With this function, we find the users running speed for each of the segments, and then return the segment with the highest speed value. There is a similar function for finding the users slowest segment, the only difference being that instead of returning the segment with the highest speed value, we return the segment with the lowest speed value.

There are also two functions which find the users highest and lowest average heartrate. Figure 36 below shows the code snippet for getting the segment with the highest average heartrate.



```
//Function to calculate the segment effort with the highest average heart rate
const getHighestAverageHeartRateSegmentEffort = async (segmentEfforts) => {
  //Remove segments which do not have any heart rate data associated with them
  let filteredArray = segmentEfforts.filter(function (e) {
    return e.max_heartrate;
  });

  //Find the segment with the highest average heartrate and return it
  var highestAverageHeartRateSegmentEffort = filteredArray.reduce(function (
    prev,
    current
  ) {
    return prev.average_heartrate > current.average_heartrate
      ? prev
      : current;
  });
  return highestAverageHeartRateSegmentEffort;
};
```

Figure 36: Code Snippet showing function for getting a User's highest average heart rate segment

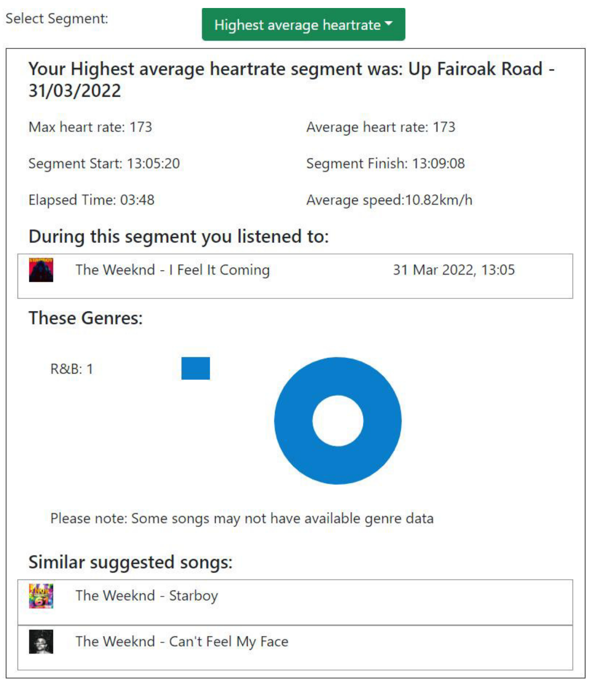
This function is incredibly similar to the one for finding the users fastest pace. The difference being that we do not need to calculate the users heart rate, we only need to remove any segments from the list which do not have any heartrate data associated with them, this is to catch any errors where the user might not have worn a smart watch or activity tracker on their run. Following this we can find the segment with the highest average heart rate and return it. There is a similar function for finding the lowest average heart rate segment, the difference being that we return the lowest average heartrate segment instead. These functions work whether the user is analysing one single activity, or several activities collectively across a date range.

As mentioned in the project aims, there were plans for integrating support for analysing other types of workouts such as HIIT workouts, weightlifting, boxing etc. Upon investigating the addition of this functionality however, I discovered that there were limitations in the data provided by Strava for these kinds of activities. Whereas running activities are broken down into segments with start and end times and relevant heartrate data, other activities such as boxing or HIIT workouts are not. Strava does not split these workouts into segments, and so they cannot be analysed in a similar way to the running activities. The only data we are provided with is the start and end times and the users average heart rate across the entire activity. After some investigation it was concluded that unfortunately there was no suitable way to analyse these activities individually in a way that would provide any useful information to the user. The most that the application can provide on a single one of these activities is just the list of songs that the user listened to during that workout, and the genres of the songs.

Although there was this limitation with the data, I was still able to implement some form of analysis for other types of workouts. As a compromise, I implemented the functionality for analysing a number of different workouts of the same type across a specified date range collectively. This would instead allow the user to find the specific workout across a number of workouts where their average heartrate was the highest or lowest, see what songs they listened to during that workout, and then receive recommendations for similar songs based on that workout.

4.7 Mapping Strava Data

The decision to introduce a map into the solution came midway during development. When analysing "Run" activities, the user is now presented with a map which shows them the map for the segment which they currently have selected, for example, if the user has selected to view their segment with their lowest average heart rate, then the map will show them the route of that segment. An example of this can be seen below in **Error! Reference source not found.**.



Select Segment: Highest average heartrate

Your Highest average heartrate segment was: Up Fairoak Road - 31/03/2022

Max heart rate: 173 Average heart rate: 173

Segment Start: 13:05:20 Segment Finish: 13:09:08

Elapsed Time: 03:48 Average speed:10.82km/h

During this segment you listened to:

The Weeknd - I Feel It Coming 31 Mar 2022, 13:05

These Genres:

R&B: 1

Please note: Some songs may not have available genre data

Similar suggested songs:

The Weeknd - Starboy

The Weeknd - Can't Feel My Face

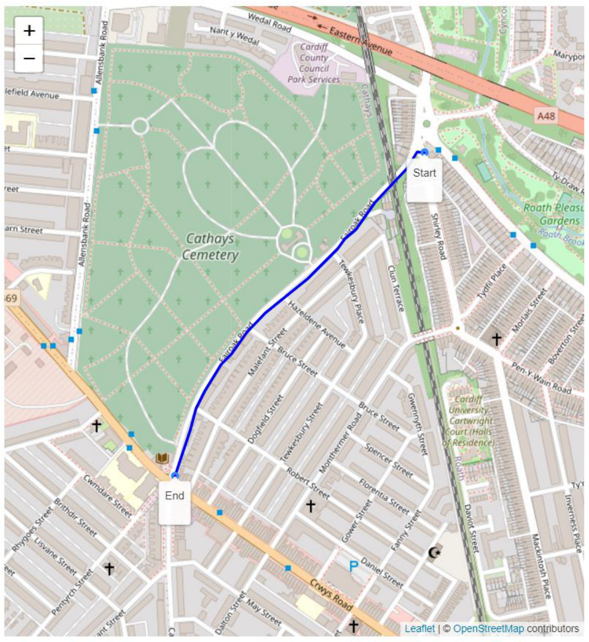
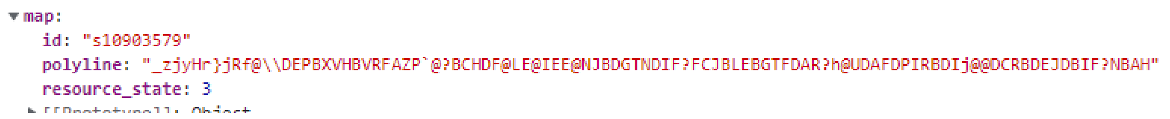


Figure 37: Screenshot showing example map for a highest average heartrate segment

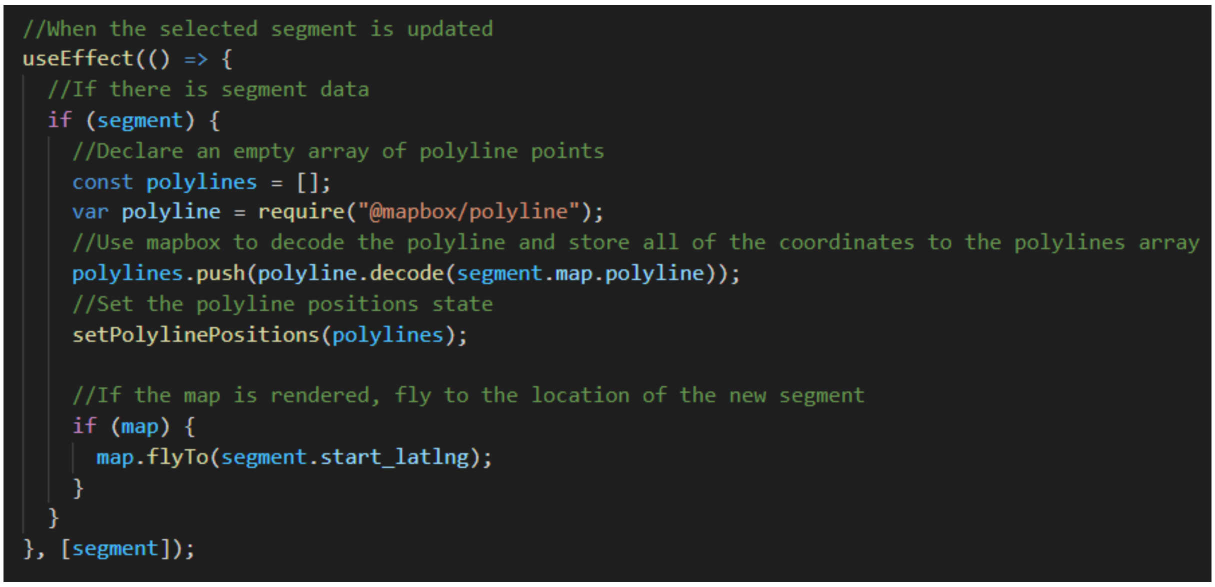
Implementing this feature into the application became quite a challenge, and the solution to the problem is particularly interesting. As previously mentioned, the mapping library which was used was Leaflet.js, alongside the React-Leaflet package. To be able to display location data on the map, it is necessary to obtain the encoded polyline data from Strava for the segment that needs to be shown, decode this polyline data and then set the result as a line on the map. An example of the encoded polyline which Strava returns can be seen below in Figure 38.



```
▼ map:
    id: "s10903579"
    polyline: "_zjyHr}jRf@\\DEPBXVHBVRFAZP`@?BCHDF@LE@IEE@NJBDGTNDIF?FCJBLEBGTFDAR?h@UDAFDPIRBDIj@@DCRBDEJDBIF?NBAH"
    resource_state: 3
```

Figure 38: Example encoded segment polyline returned by Strava

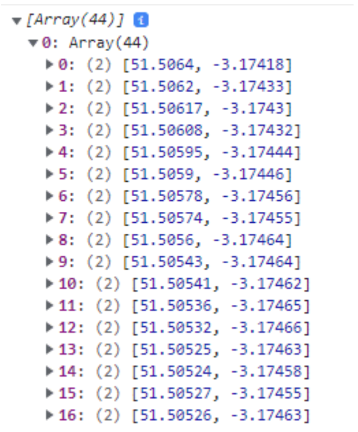
Unfortunately, we cannot use the encoded polyline directly to display data on the map, it first needs to be decoded. In order to do this I researched into various different possible solutions, and finally settled on using an available JavaScript package called "Mapbox" (34), which provides a number of tools for manipulating map data. The method that we need in particular is the “decode” method which can be used on the encoded polyline data to convert it into usable data.



```
//When the selected segment is updated
useEffect(() => {
  //If there is segment data
  if (segment) {
    //Declare an empty array of polyline points
    const polylines = [];
    var polyline = require("@mapbox/polyline");
    //Use mapbox to decode the polyline and store all of the coordinates to the polylines array
    polylines.push(polyline.decode(segment.map.polyline));
    //Set the polyline positions state
    setPolylinePositions(polylines);

    //If the map is rendered, fly to the location of the new segment
    if (map) {
      map.flyTo(segment.start_latlng);
    }
  }
}, [segment]);
```

Figure 39: Code Snippet showing method for decoding Strava Polyline data into an array of coordinates in StravaMap.js

Figure 39 above shows the method for decoding the polyline data using “Mapbox” in the StravaMap.js file. Upon a new segment being selected, the encoded polyline is run through the “polyline.decode” method, which returns an array of coordinates.



```
▼ [Array(44)]
  ▼ 0: Array(44)
    ▶ 0: (2) [51.5064, -3.17418]
    ▶ 1: (2) [51.5062, -3.17433]
    ▶ 2: (2) [51.50617, -3.1743]
    ▶ 3: (2) [51.50608, -3.17432]
    ▶ 4: (2) [51.50595, -3.17444]
    ▶ 5: (2) [51.5059, -3.17446]
    ▶ 6: (2) [51.50578, -3.17456]
    ▶ 7: (2) [51.50574, -3.17455]
    ▶ 8: (2) [51.5056, -3.17464]
    ▶ 9: (2) [51.50543, -3.17464]
    ▶ 10: (2) [51.50541, -3.17462]
    ▶ 11: (2) [51.50536, -3.17465]
    ▶ 12: (2) [51.50532, -3.17466]
    ▶ 13: (2) [51.50525, -3.17463]
    ▶ 14: (2) [51.50524, -3.17458]
    ▶ 15: (2) [51.50527, -3.17455]
    ▶ 16: (2) [51.50526, -3.17463]
```

Figure 40: Example screenshot of decoded polyline coordinates

This array of coordinates is what is then plotted on the map component to display the segment line. To make this feature more usable for the user I also implemented a “map.flyTo()” function, which moves the map to the location of the new segment once it has been loaded onto the map so that the user can see it straight away and does not need to manually search for it.

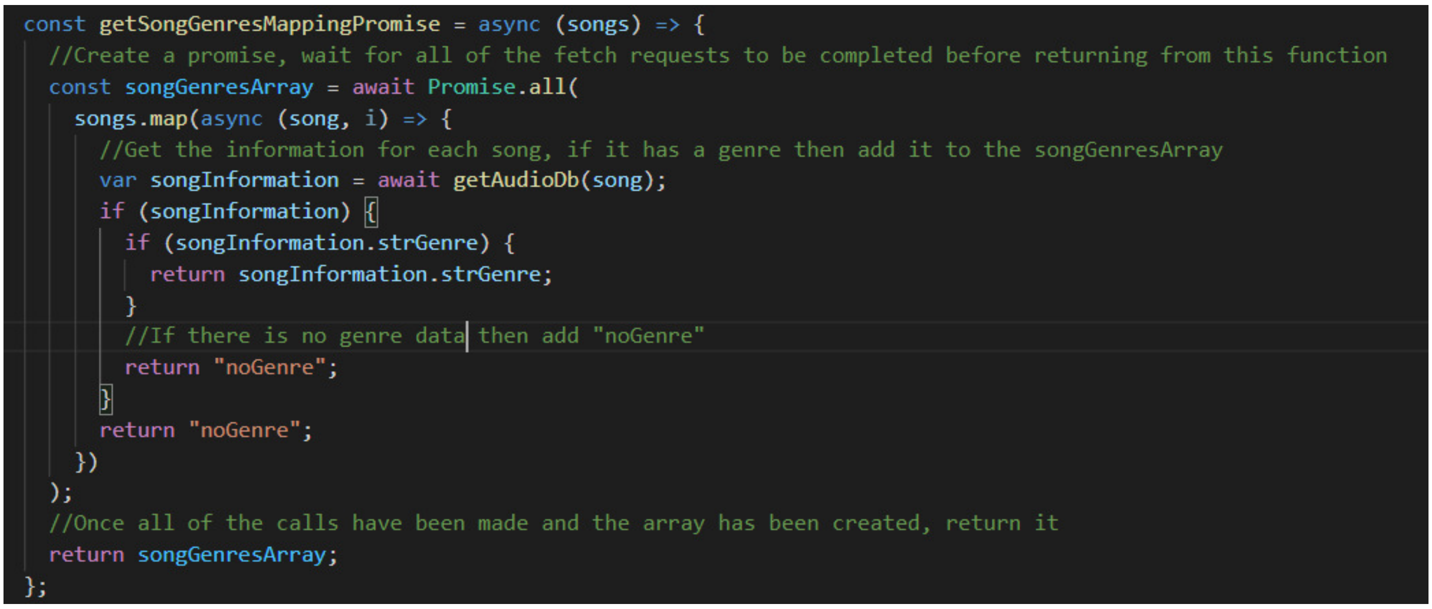
4.8 Song Genre Data

A particularly interesting and unexpected issue which was encountered during the development process was the implementation of song genres. One of the primary aims for this project was to display information to the users about the songs that they have listened to, including their genres. This was one of the later features implemented into the project, and due to an oversight on the data available from the Last.FM API, caused a momentary roadblock in development. The encountered issue was that data for any song returned by the Last.FM API does not include a genre field. This meant that as of that moment, the application had no way of defining songs into certain genres and therefore no way of displaying this data to the user.

To overcome this issue, I researched into different ways in which I could obtain song data and at first, I turned my attention to the Spotify API. Although I had already decided not to use Spotify as part of the implementation, as an API provider, Spotify is one of the world's leading data providers with an extensive amount of data on each song, including the needed genre data. The idea was to continue to have a user connect a Last.FM account and use this for getting their listening history, and then take that list of Last.FM songs and make a call to the Spotify API to get extra data on each song. Unfortunately, I encountered another issue here where the Spotify API requires an authenticated user to make any API calls. This meant that in order to obtain genre data for their songs, I would have had to require the user to also connect a valid Spotify account. This would have defeated my purpose for choosing Last.FM as the service used, as only Spotify account holders would then have access to this feature, and it would exclude users of other services as well as adding unnecessary level of complexity for the user by requiring them to log in and use 3 different external services.

Continuing research into other solutions, I found a website called TheAudioDb. TheAudioDb is a community run database of audio artwork and metadata with a JSON API (35). The API is well documented and provides the exact functionality that was needed for the application. The only caveat to this service is that it is no longer free and currently costs £3 a month to receive access to an API key. Ultimately it was decided that TheAudioDb would work as the better method in this instance, and so after obtaining an API key, I began implementing the solution.

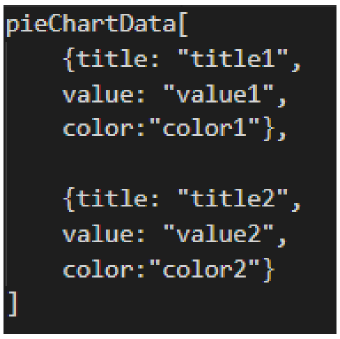
Working with TheAudioDb API proved to be tricky, as I needed to obtain genre data for each song, and as there is no endpoint for taking a list of songs and just returning this data, I had to make an individual API call for each song in the list. This led to the implementation of a somewhat more complex and interesting method for solving this problem. As we have already discussed, making a fetch request is an asynchronous function. However, because we are going to be making a lot of these requests, it is vital that we ensure these calls are handled properly using an asynchronous mapping method. Figure 41 below shows a code snippet of the function created for handling this.



```
const getSongGenresMappingPromise = async (songs) => {
  //Create a promise, wait for all of the fetch requests to be completed before returning from this function
  const songGenresArray = await Promise.all(
    songs.map(async (song, i) => {
      //Get the information for each song, if it has a genre then add it to the songGenresArray
      var songInformation = await getAudioDb(song);
      if (songInformation) {
        if (songInformation.strGenre) {
          return songInformation.strGenre;
        }
        //If there is no genre data then add "noGenre"
        return "noGenre";
      }
      return "noGenre";
    })
  );
  //Once all of the calls have been made and the array has been created, return it
  return songGenresArray;
};
```

Figure 41: Code Snippet showing function for making a call to TheAudioDb to get genre data for each song in a list of songs

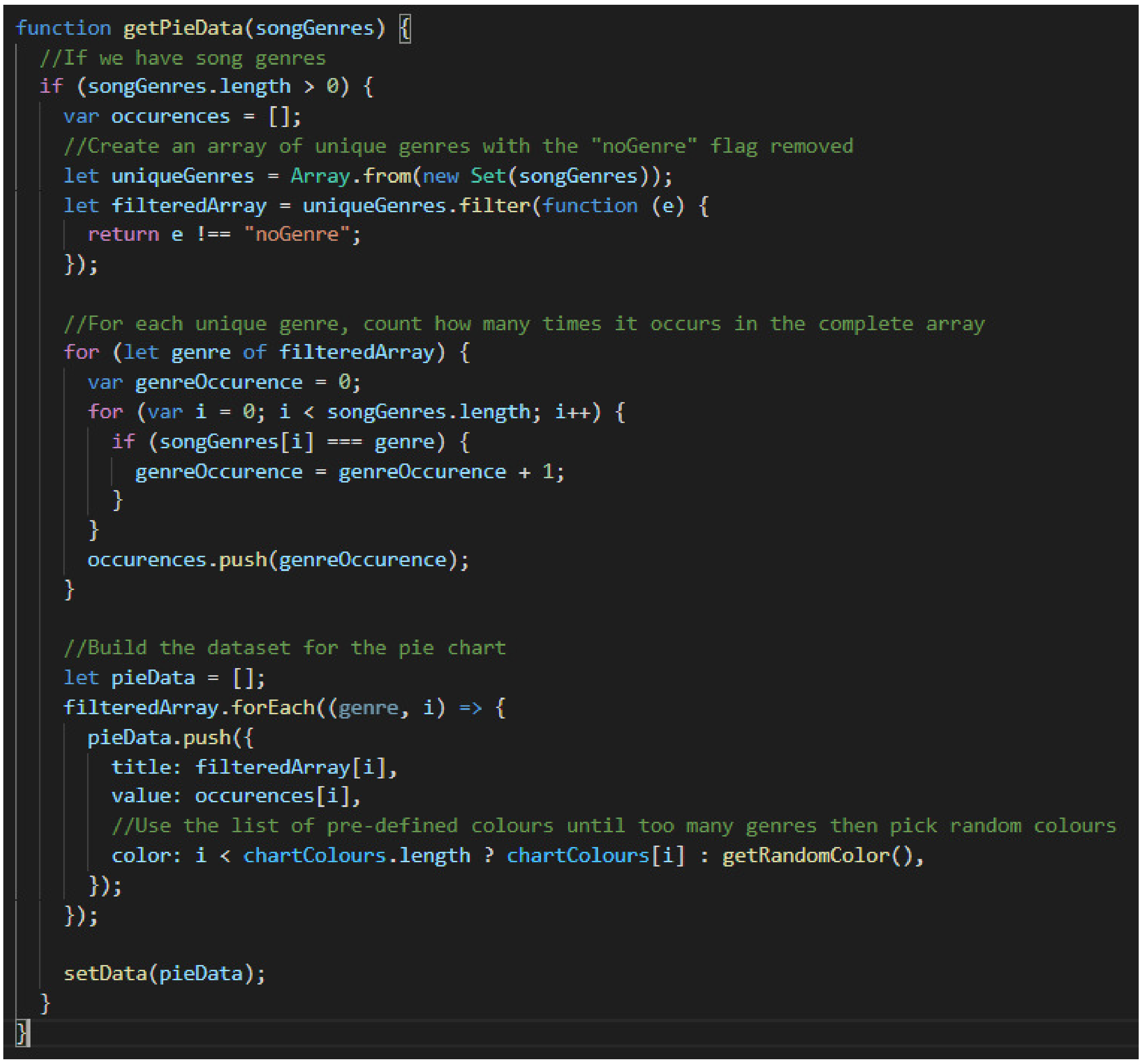
Once the above function has run successfully, we should have a set of data for each of the songs which includes the genre data. We then need to process this data order to provide the information to the user in a useful way. The way that we display the genre data to the user is through pie charts. The pie charts provide a visual representation of what genres they listened to and make the data easily readable for the user. To create the pie chart component I used a JavaScript package called "React-minimal-pie-chart" (22), as previously mentioned, which enables quick creation of pie charts on a web page. The pie chart component needs to be provided with an array of objects, each with a "title"; the name of the genre, a "value"; the number of songs with this genre, and a "colour"; the colour of the pie chart segment for the specified genre. In order to use the genre data, I had obtained from TheAudioDb I first had to process the data into a specific format so that I could supply it to the pie chart. Figure 42 below shows an example of how the data given as a parameter to the pie chart should be structured.



```
pieChartData[
    {title: "title1",
    value: "value1",
    color:"color1"},

    {title: "title2",
    value: "value2",
    color:"color2"}
]
```

Figure 42: Example data structure which should be supplied to the React-Minimal-Pie-Chart Component

To format the data in this way, a function was created called "getPieData" that takes a list of all of the song genres and returns the data structured in this way. This function can be seen below in Figure 43.

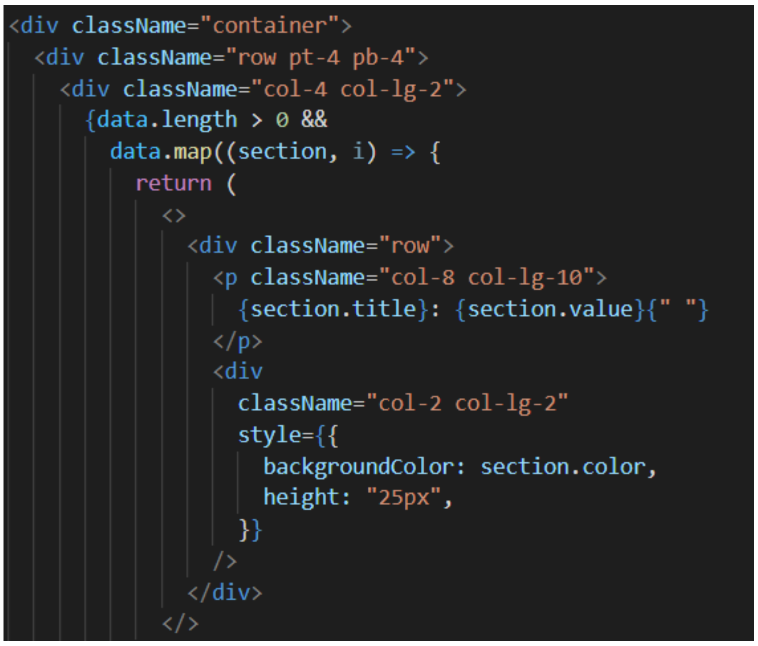


```
function getPieData(songGenres) {
  //If we have song genres
  if (songGenres.length > 0) {
    var occurences = [];
    //Create an array of unique genres with the "noGenre" flag removed
    let uniqueGenres = Array.from(new Set(songGenres));
    let filteredArray = uniqueGenres.filter(function (e) {
      return e !== "noGenre";
    });

    //For each unique genre, count how many times it occurs in the complete array
    for (let genre of filteredArray) {
      var genreOccurence = 0;
      for (var i = 0; i < songGenres.length; i++) {
        if (songGenres[i] === genre) {
          genreOccurence = genreOccurence + 1;
        }
      }
      occurences.push(genreOccurence);
    }

    //Build the dataset for the pie chart
    let pieData = [];
    filteredArray.forEach((genre, i) => {
      pieData.push({
        title: filteredArray[i],
        value: occurences[i],
        //Use the list of pre-defined colours until too many genres then pick random colours
        color: i < chartColours.length ? chartColours[i] : getRandomColor(),
      });
    });

    setData(pieData);
  }
}
```

Figure 43: Code Snippet showing function taking a list of Song Genres and structuring it for the pie chart

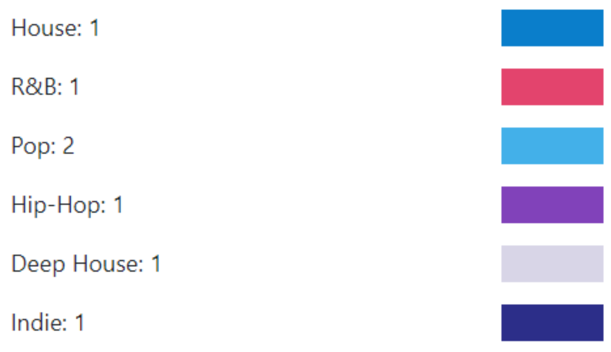
There are 3 main sections to this function. As the provided "songGenres" list can contain multiple of the same genre, the first step is to create a new list of genres which only contains one occurrence of each. In this step we also remove the "noGenre" flags which were added to the array for songs where genre data did not exist. The next step in the function is to count how many times each genre appears, meaning how many songs the user listened to in each genre. These values are then stored in the "occurrences" array. The final step is to build the dataset for the pie chart, looping through each unique genre, setting the title as the name of the genre, the value as its number of occurrences and then its colour. This complete array is then set as the data variable used by the pie chart.

Styling the pie chart component in a suitable way for the page was more time consuming than I had first anticipated. The documentation available for how the pie chart works and how to style the different segments was very comprehensive, however the details on how it should be sized on the page were lacking. In addition to this, I did not want to display the values of the chart segments on the chart itself, but instead I wanted a separate key alongside the pie chart, and I was unable to find any documentation on how to do this easily. My solution to this was to use the pieData array of objects I had created, and then loop through it to render a label and a div block with the background colour set to the matching colour for each of the items in the array. The code for generating this key and an example of the resulting key can be seen below in Figure 44 and Figure 45.



```
<div className="container">
  <div className="row pt-4 pb-4">
    <div className="col-4 col-lg-2">
      {data.length > 0 &&
        data.map((section, i) => {
          return (
            <>
              <div className="row">
                <p className="col-8 col-lg-10">
                  {section.title}: {section.value}{" "}
                </p>
                <div
                  className="col-2 col-lg-2"
                  style={{
                    backgroundColor: section.color,
                    height: "25px",
                  }}
                />
              </div>
            </>
```

Figure 45: Code Snippet showing rendering of pie chart key



House: 1	🟦
R&B: 1	🟥
Pop: 2	🟦
Hip-Hop: 1	🟪
Deep House: 1	⬜
Indie: 1	🟦

Figure 44: Screenshot showing example of a rendered pie chart key

4.9 Suggesting Similar Songs

As outlined in the project aims, one of the features that was discussed was some functionality to suggest similar songs to the user based on the ones that they performed best to during their workouts. This functionality had many implications which needed to be addressed before it could be implemented. It was decided that the suggested songs would be based upon the music listened to during the users' best segments for the analysed run, for example their segment with the highest heart rate or fastest pace. My original idea was to take the songs the user had listened to in these segments, find information about them such as their genre or tempo, and find songs similar with these properties. This proved to be much more difficult than I had first anticipated due to the issue I had where there was no genre or tempo data for songs provided by Last.FM. Although I worked around the genre data issue with TheAudioDb, TheAudioDb does not provide tempo data either. This meant that I would have had to find another external data source which could provide this data to the application, however this could have put a serious strain on the loading with another layer of API calls which would need to be made and also would increase the risk of the application failing should one of the data source providers being unavailable.

After some further reading, I discovered that the Last.FM API actually has a method available called "track.getSimilar". This allows us to make a request to the API with a song name and its artist and receive similar songs as a response. It is not clear in the documentation exactly how the songs returned are classed as "similar", however after some testing. In most cases it appears to choose similar songs from the same artist, but this is not always the case. I opted to utilise this method as a solution to the problem in order to implement the functionality in some way albeit I was unable to match the songs by specific beats per minute.

Although Last.FM offers this method, as of the date this project was developed, there is an issue with their "track.getSimilar" method. For all of the song images which are displayed on RaceTracks, the images are being loaded directly from Last.FM. The image URLS are included as part of the song objects returned by the API, and these are then used to display the images on site. Unfortunately the "track.getSimilar" method does not correctly return the URLs for song images, and instead returns invalid URLs. This means that using this method alone would result in all of the songs being displayed with null images. In order to work around this bug, RaceTracks gets the list of similar songs from Last.FM and then makes another set of API requests to get information on each of those songs, which includes the correct image URL.s. This became one of the more complex features within the system and required careful planning on how to properly handle the API requests using async methods and promises. The "getSimilarSongsForTracks" method which I created can be seen below in Figure 46.

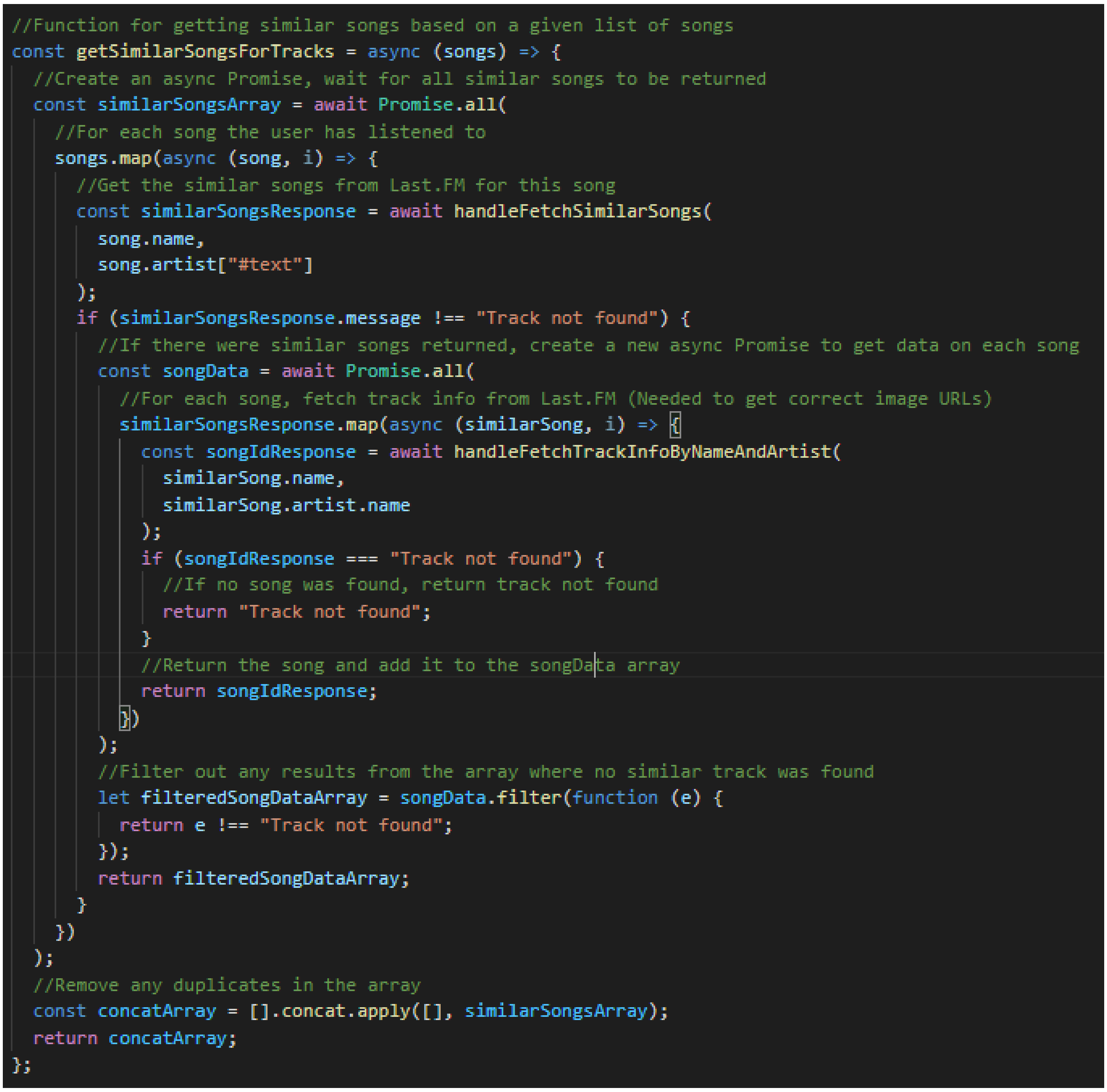
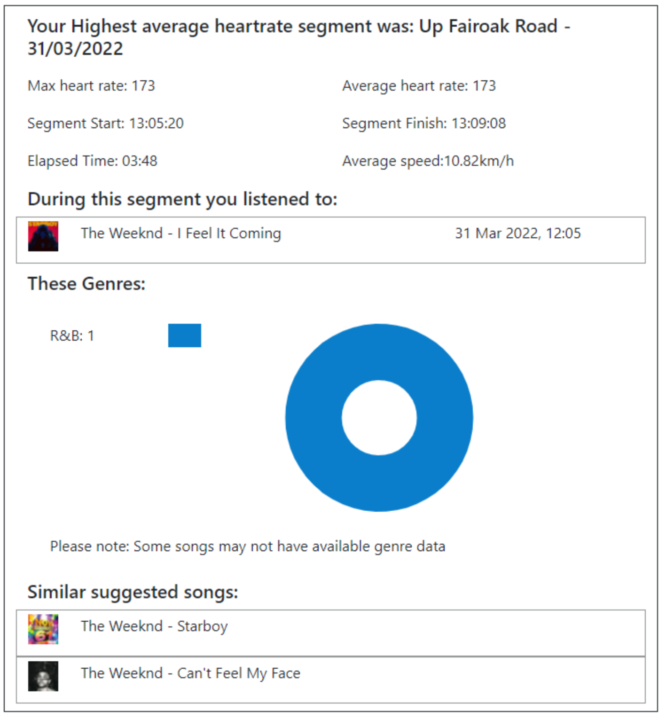

```
//Function for getting similar songs based on a given list of songs
const getSimilarSongsForTracks = async (songs) => {
  //Create an async Promise, wait for all similar songs to be returned
  const similarSongsArray = await Promise.all(
    //For each song the user has listened to
    songs.map(async (song, i) => {
      //Get the similar songs from Last.FM for this song
      const similarSongsResponse = await handleFetchSimilarSongs(
        song.name,
        song.artist["#text"]
      );
      if (similarSongsResponse.message !== "Track not found") {
        //If there were similar songs returned, create a new async Promise to get data on each song
        const songData = await Promise.all(
          //For each song, fetch track info from Last.FM (Needed to get correct image URLs)
          similarSongsResponse.map(async (similarSong, i) => {
            const songIdResponse = await handleFetchTrackInfoByNameAndArtist(
              similarSong.name,
              similarSong.artist.name
            );
            if (songIdResponse === "Track not found") {
              //If no song was found, return track not found
              return "Track not found";
            }
            //Return the song and add it to the songData array
            return songIdResponse;
          })
        );
        //Filter out any results from the array where no similar track was found
        let filteredSongDataArray = songData.filter(function (e) {
          return e !== "Track not found";
        });
        return filteredSongDataArray;
      }
    })
  );
  //Remove any duplicates in the array
  const concatArray = [].concat.apply([], similarSongsArray);
  return concatArray;
};
```

Figure 46: Code Snippet showing function for getting similar songs from Last.FM

Although it is not exactly how I would have liked to implement the feature, due to time constraints and limitations with the Last.FM API, it was a suitable work around and meant that the feature would not be entirely missing from the completed solution. The similar suggested songs feature can be seen below in Figure 47.



Your Highest average heartrate segment was: Up Fairoak Road - 31/03/2022

Max heart rate: 173 | Average heart rate: 173

Segment Start: 13:05:20 | Segment Finish: 13:09:08

Elapsed Time: 03:48 | Average speed:10.82km/h

During this segment you listened to:

The Weeknd - I Feel It Coming | 31 Mar 2022, 12:05

These Genres:

R&B: 1

Please note: Some songs may not have available genre data

Similar suggested songs:

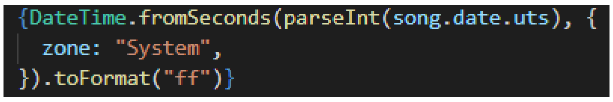
The Weeknd - Starboy

The Weeknd - Can't Feel My Face

Figure 47: Highest average heart rate segment showing similar suggested songs

4.10 Dealing with DateTime Objects

Appropriate and correct handling of dates and times are a critical part of the application. Both activities and songs have DateTime data associated with them, which we use to display the correct songs listened to during specific workouts. As previously discussed, a JavaScript package called Luxon (21), is used within the system that allows the manipulation of dates and times. Both Strava and Last.FM provide their dates in a UTC / Epoch date format in seconds, which should allow for consistency across the application and easier date matching between activities and songs. There were little to no issues in dealing with DateTimes throughout development, until towards the end of the project where I noticed that the times that songs were listened to on some of the activities did not match up with the time the activity actually took place and that they were 1 hour off. This was due to the daylight-saving change on the 27th of March 2022, where the United Kingdom time zone moved from 1:00am to 2:00am. Strava accounts for this change on an account specific basis, and as my account which I was using for testing purposes was based in the UK, the UTC times provided by the API were correct following this time change. Last.FM however, does not account for daylight savings. Therefore, all of the song data which I was receiving and using to display when the songs were listened to, was an hour behind the correct time. To counter this issue, I used a feature within Luxon to take the UTS time provided and convert it to the time zone of the user's system. Assuming the user's system is correctly set to their time zone, this should mitigate this issue. A future improvement would be to find a way to get the time zone that Strava is using for the user's data and use this to convert the Last.FM datetimes. A code example of a Last.FM DateTime being converted can be seen below in Figure 48. We take the date provided in UTC and convert it to the systems time zone and then format it appropriately. In Luxon, "ff" formats a date time in the format of: "31 Mar 2022, 13:10" (36).



```
{DateTime.fromSeconds(parseInt(song.date.uts), {
  zone: "System",
}).toFormat("ff")}
```

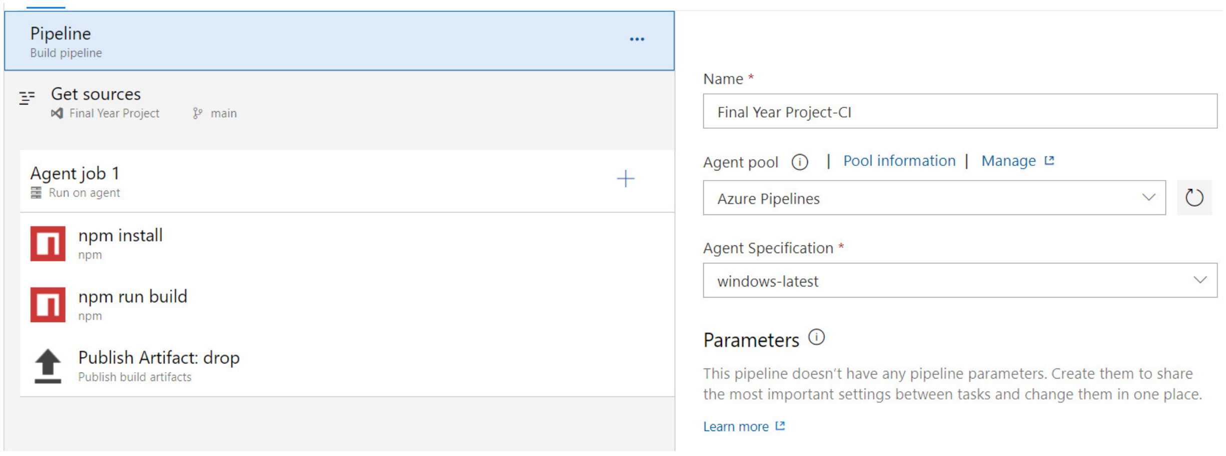
Figure 48: Code Snippet showing conversion of Last.FM date to the local system time zone

4.11 Live Webhosting

As previously discussed, something which become an aim of the project during the development process was the have the application hosted on the web so that a live version of it is available to access. Several different solutions for free and simple ways of hosting React applications were investigated. Initially I looked at using GitHub Pages as a solution (37), which allows the hosting of live sites using production code directly from a GitHub repository. I encountered issues with this service however since it could not handle the URL routing on single page applications in the way I needed. Due to my application redirecting to Strava for authorization and then being redirected back with a new URL, GitHub pages would error since it did not recognise the new redirected URL as a valid one.

Following this issue and as I was hosting my code repository in Microsoft Azure DevOps, I decided to investigate the available options for hosting the RaceTracks with Azure. Azure provided the support for handling the single page routing which my application needed, although to make this possible, I needed to add a ".web-config" file, which can be found in the "public" folder of the project source code. Azure uses the rules outlined in this config file to ignore the URL routing on the application and always redirect to the correct page.

Using Azure's tools, I created a new "App Service", which is the HTTP-based service used for hosting web applications. I then began creating a production pipeline which when triggered, runs pre-set commands on the code base in order to create a production ready build of the source code which can be hosted on the created app service. Figure 49 below shows the azure GUI with the commands run by the pipeline. The pipeline runs "npm install" which is a Node.js command to install all of the required JavaScript packages, and then "npm run build" which is also a Node.js command, to create a production ready build of the codebase.


Pipeline
Build pipeline

Get sources
Final Year Project main

Agent job 1
Run on agent

npm install
npm

npm run build
npm

Publish Artifact: drop
Publish build artifacts

Name *
Final Year Project-CI

Agent pool | Pool information | Manage

Azure Pipelines

Agent Specification *
windows-latest

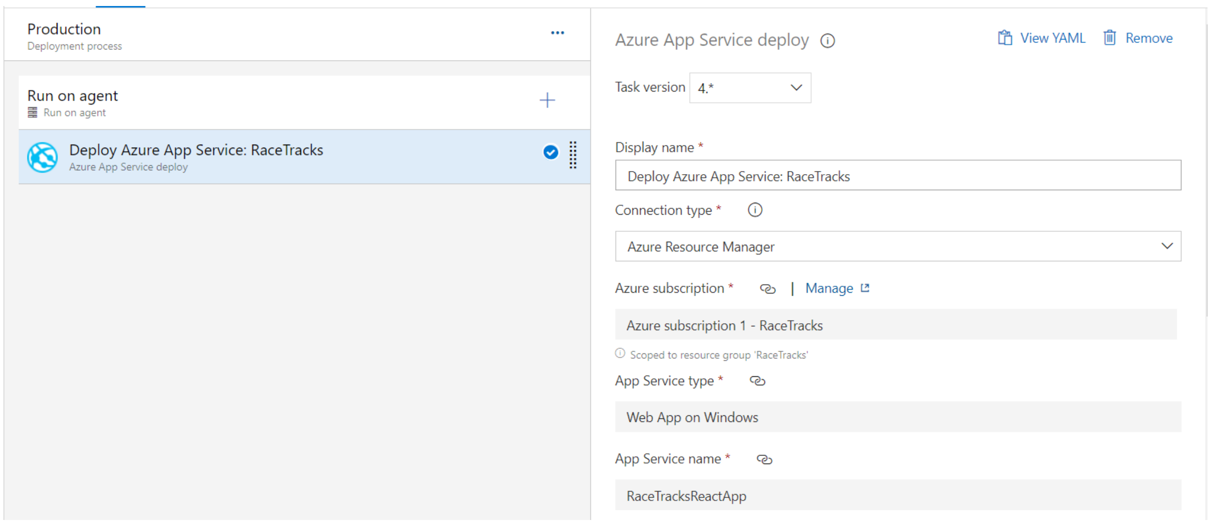
Parameters

This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.

Learn more

Figure 49: Screenshot showing the Azure Pipeline commands for creating a production build of the application

Once the pipeline has run, then a new "Release" is created. This release deploys the production-ready code onto the previously mentioned App-service where it is then available to access at: https://racetracksreactapp.azurewebsites.net/. The release pipeline can be seen below in Figure 50.


Production
Deployment process

Run on agent
Run on agent

Deploy Azure App Service: RaceTracks
Azure App Service deploy

Azure App Service deploy

View YAML Remove

Task version 4.*

Display name *
Deploy Azure App Service: RaceTracks

Connection type *
Azure Resource Manager

Azure subscription * | Manage
Azure subscription 1 - RaceTracks
Scoped to resource group 'RaceTracks'

App Service type *
Web App on Windows

App Service name *
RaceTracksReactApp

Figure 50: Screenshot showing the Azure Release Pipeline commands

4.12 Security

An important thing to consider when building and deploying a new web application is the Security measures put in place. There are various aspects of the system that promote a secure implementation. Firstly, the system implements the secure HTTPS protocol. HTTPS is an extension of the HTTP protocol which is used for secure communication over a computer network and is widely used on the internet. The HTTPS protocol is encrypted using Transport Layer Security (38). Because RaceTracks is using solely HTTPS to make requests to the external API's, the traffic between the two servers is encrypted to prevent eavesdropping and the exposure of any private data.

As previously discussed, the application also implements OAuth2 authorization when making requests to the Strava API. As this has been implemented correctly and uses HTTPS for making the requests, the user's data is protected from eavesdropping and man-in-the-middle attacks. Furthermore, all the data obtained from users Last.FM accounts is all public-facing data, and anyone could obtain this data should they wish to, so there is no risk of exposing anything personal or private here.

Lastly, as the application is being hosted on Microsoft's Azure servers, there are several security features which are provided as part of the service. The Azure servers provide an Internal Load Balancer (ILB), which provides the web application with a firewall to offer enterprise-level protection from security threats such as DDoS attacks, URI filtering and SQL injection (39).

5. Results and Evaluation

5.1 First round of testing – 25/02/2022

After completing development work on the features that met that primary aims for this project and satisfied the criteria laid out for a minimum viable product, the first round of testing on the system was carried out. As a test-driven development approach was taken, many of the bugs and issues which could have been picked up in the round of testing, may have already been fixed or resolved. With this testing session the aim was to check and verify that all the key features and functionality were working as intended.

The data for this round of testing was collected by myself using an Apple Watch series 6, running the latest version of the Strava app as of this date, whilst listening to music through my Spotify account that was connected to my Last.FM account through my iPhone X. I carried out the testing of the web-app on my most recent codebase running as a live website hosted on Microsoft Azure. This allowed me to also test for any issues which may not be present during local development. The results for this round of testing can be seen in the table below:

Test Number	Action	Expected Result	Achieved Result	Test Result	Comments	Evidence
1	Click connect to Strava button	Redirect to Strava Login page	As expected	Pass		Appendix: Figure 58
2	Log in to Strava	Redirects to Authorize app page	As expected	Pass		Appendix: Figure 59
3	Click Authorize on Strava account	Redirects to my web-app with connect to last.fm screen showing	As expected	Pass	Connect Strava screen shows momentarily after redirecting back to my web app	Appendix: Figure 60
4	Enter LastFm username and click connect to last.fm	Home page should load with loaded recent strava activities and recent LastFm songs	As expected	Pass	Username was "TylerJenkins22"	Appendix: Figure 61
5	Enter a non-existing LastFm username and click connect to last.fm	App should not direct to home page and should stay on connect to last.fm screen	As expected	Pass	App tries to make the GET request but receives a 404 message, app does not crash	Appendix: Figure 62
6	Click analyse on a recent activity	Analyse activity page loads with correct data in songs list, segments, and genre pie chart	As Expected	Pass		Appendix: Figure 63
7	Change dropdown for segment filter to each option	Loads selected segment	As expected	Pass		Appendix: Figure 64
8	Click back button on analyse activity page	Returns to home page	As expected	Pass		N/A
9	Enter invalid date ranges and click search for activities by date	No results should be returned	As expected	Pass	A message to say please select valid dates or disable search button for invalid search dates could be in place. Also, should add "Start date" and "End date" labels	Appendix: Figure 65
10	Enter valid date ranges and click	Results between date ranges	As expected	Pass	If date range is too big, only most	Appendix: Figure 66

	search for activities by date	should be displayed			recent 30 activities will be returned	
11	Click analyse all activities between these dates button	Analyse activities page loads with data from all activities in that selected range	As expected	Pass		Appendix: Figure 67
12	Change dropdown for segment filter to each option	Loads selected segment and should be correct across all selected activities	As expected	Pass		Appendix: Figure 68
13	Click back button on analyse activities page	Returns to home page	As expected	Pass		N/A
14	After successful activity date range search, change activity filter to “Workout”	Should show “Workout” activities	As expected	Pass	These activities currently cannot be analysed like “Run” activities	Appendix: Figure 69
15	Clicking analyse on an activity which is not “Run”	Should load analyse page with any music data but inform the user that this activity cannot be analysed	As expected	Pass	Support for other activities should be added as a future feature	Appendix: Figure 70
16	Entering another users username on “Connect to Last.FM” screen	App will load with authorized strava account activities but another users songs	As expected	Pass	Username used:”John” This is not a bug, as Last.FM profile data is public. However, there is no log out option currently so if someone enters their username wrong they must refresh the app.	Appendix: Figure 71

Figure 51: Testing table showing first round of testing on 25/02/2022

Following the results of the first round of testing, I noted that there were a few small things which could benefit from being changed, such as displaying certain messages to the user or implementing a log out feature so that the user wasn't stuck if they had connected an incorrect Last.FM account. I was confident enough with my system that I would be able to move forward in the development process and begin implementing the new proposed features that I had outlined in my secondary aims. This included features such as song suggestions, more detailed insights in data comparison and the possibility of investigating a log in system for the application.

5.2 Second round of testing – 14/04/2022

Upon reaching the end of development, I began a final round of testing on the system I had developed. This followed the previous round of testing, and I had also been testing throughout development. With this testing session I aimed to check and verify that all the key features and functionality were working as intended.

The data for this round of testing was collected similarly to the first round. It was collected by myself using an Apple Watch series 6, running the latest version of the Strava app as of this date, whilst listening to music through my Spotify account that was connected to my Last.FM account through my iPhone X. The testing was done against the latest codebase running as a live website hosted on Microsoft Azure.

Test Number	Action	Expected Result	Achieved Result	Test Result	Comments	Evidence
1	Click connect to Strava button	Redirect to Strava Login page	As expected	Pass		Appendix: Figure 72
2	Log in to Strava	Redirects to Authorize app page	As expected	Pass		Appendix: Figure 73
3	Click Authorize on Strava account	Redirects to my web-app with connect to last.fm screen showing	As expected	Pass	Connect Strava screen shows momentarily after redirecting back to my web app	Appendix: Figure 74
4	Enter LastFm username and click connect to last.fm	Home page should load with loaded recent strava activities and recent LastFm songs	As expected	Pass	Username was "TylerJenkins22"	Appendix: Figure 75
5	Enter a non-existing LastFm username and click connect to last.fm	App should not direct to home page and should stay on connect to last.fm screen	As expected	Pass	App tries to make the GET request but receives a 404 message, app does not crash	Appendix: Figure 76
6	Click analyse on a recent activity	Analyse activity page loads with	As Expected	Pass		Appendix: Figure 77

		correct data in songs list, segments, map and genre pie chart.				
7	Change dropdown for segment filter to each option	Loads selected segment and updates map with new segment location	As expected	Pass		Appendix: Figure 78
8	Click back button on analyse activity page	Returns to home page	As expected	Pass		N/A
9	Enter invalid date ranges and click search for activities by date	No results should be returned	As expected	Pass	Still no message to notify the user of an incorrect search. Could also have disabled search button when incorrect dates selected.	Appendix: Figure 79
10	Enter valid date ranges and click search for activities by date	Results between date ranges should be displayed	As expected	Pass	If date range is too big, only most recent 30 activities will be returned	Appendix: Figure 80
11	Click analyse all activities between these dates button	Analyse activities page loads with data from all activities in that selected range	As expected	Pass		Appendix: Figure 77
12	Change dropdown for segment filter to each option	Loads selected segment and should be correct across all selected activities	As expected	Pass		Appendix: Figure 78
13	Click back button on analyse activities page	Returns to home page	As expected	Pass		N/A
14	After successful activity date range search, change activity filter to “Workout”	Should show “Workout” activities	As expected	Pass	These activities currently cannot be analysed like “Run” activities	Appendix: Figure 81
15	Clicking analyse on an activity	Should load analyse activity page	As expected	Pass	It was not possible to analyse a single workout activity as	Appendix: Figure 82

	which is not "Run"				discussed, these activities can only be analysed as a group	
16	Clicking analyse all activities in date range for "Workout" activities	Should load analyse activities page	As expected			Appendix: Figure 83
17	Changing filter on analyse workout page should	Should display workout activity matching the selected filter	Application crashed and displayed a blank page	Fail	Seems to be an issue with displaying the similar songs.	Appendix: Figure 84
18	Entering another users username on "Connect to Last.FM" screen	App will load with authorized Strava account activities but another users songs	As expected	Pass	Username used:"John" This is not a bug, as Last.FM profile data is public. There is now a "Log Out" button, which was picked up in the first round of testing	Appendix: Figure 85
19	Click "Log Out" of Strava account button	App should return to connect Strava account screen and retain users Strava account info	Application redirects to the connect Strava account and retains Last.FM account	Pass with concern	Although technically the Last.FM account is retained, upon connecting a new Strava account, the Last.FM account is lost during the redirect to Strava. This is a minor issue and could possibly be resolved using browser caching and cookies.	N/A
20	Click "Log Out" of Last.FM account button	App should return to connect Last.FM account screen and retain users Strava account info	As expected	Pass		N/A

Figure 52: Testing table showing first round of testing on 15/04/2022

The second round of self-testing aimed to build upon the first round. I re-tested all of the features which had been previously tested, as well as any new ones which had been implemented. Some of the suggestions from the first round of testing were implemented to improve the user experience such as the log out functionality. The majority of tests passed with results as expected, however there were a few outlying cases. In test case 17 of the second round of testing, it was expected that changing the filter of which best activity to view, that the page would update to show the matching activity. However, changing the filter resulted in the page crashing with errors being thrown in the console. This test highlighted a core issue with the system. After some investigation it appeared that the bug had been introduced when the functionality for suggesting similar songs was implemented, following this testing session, I was able to re-visit the code base and implement a fix for this issue so that this page works as expected and now passes the test case.

One other test which highlighted a concern within the system was test case 19. This test technically passed as expected, however there were issues which followed. Upon clicking the “Log Out” button for the users Strava account, the Last.FM account details were not lost, and the user is returned to the connect Strava account screen. However, the issue appears when the user then clicks the “connect account” button to be redirected to Strava to re-connect their account. As the application does not store Last.FM account details within the browser cache or cookies as of right now, the users Last.FM account details are then lost, and they will then have to reconnect their Last.FM account. This does not cause any errors or issues within the application itself, however it is more of a consistency issue, and inconvenient for the user to have to redo. In future work I might like to investigate the implementation of cookies and caching in order to mitigate this issue, or through the implementation of the proposed log in system for the application this could also be resolved.

5.3 Feedback from Demonstrating to an Internal Group

After obtaining permission, I was able to join a small group of other students who were all working on similar projects and demonstrate to them the final RaceTracks solution. After only a short demo, I was able to receive some feedback from them. The overall consensus was very positive, and people were impressed with the work that had been done. The main areas of the application which people found most notable were the ability to search for activities between a date range including all types of workouts and not just runs, the map feature showing the user where their best segments were, the genre pie charts and the suggestions for similar songs based on the ones that the user had listened to. I was happy with the overall feedback, and in addition to this there were also some critiques which I have taken on board. The main concern was that the analyse activity screen has the potential to be somewhat overwhelming to new users with lots of information being presented. This is a completely valid point, and something which I would to be addressed in future iterations of work.

5.4 Evaluation of Testing

After completing multiple testing sessions on the final version of the system and demonstrating the project to a small select group of people I have come to several conclusions. Due to employing a test-driven development approach I was able to mitigate a scenario where most of the test cases would result in failure during the testing sessions. Looking at the results from each testing session, I am confident in the system that I have produced and although there are some small features which could benefit from being tweaked, I am pleased with the overall performance and the way the system holds up. In order to gain further confidence in the system, I would like to have performed some real external user testing sessions, to see how the system performs with other users’ data and how the other users would interact with the system. It is quite likely that this form of testing would highlight bugs and features which could be improved in order to better the user experience, which I may not have noted during my testing. This would also help to eliminate any bias I might have had from testing my own system. In addition to user testing, the integration of automatic unit testing would have greatly improved my confidence in the system and would have highlighted any scenarios where introducing new features would have an unintended effect on another area of the system and introduced bugs. If I were to implement these two methods of testing, then the documentation for declaring the system stable and well developed would be much stronger and would strengthen my confidence in the system.

5.5 Evaluation of System Against Project Aims

As part of the system evaluation, I have used the original aims specified in section 1.3 of this document as a reference point to determine the success of the project.

Looking at the projects first primary aim; ***"To create a web-application which will display a user's running activity history as well as their music listening history for each activity.",*** I have concluded that this aim was achieved successfully. The system allows users to connect their Strava and Last.FM accounts and are able to view their recent activities as well as search for activities within a specified date range. These activities are then displayed alongside the songs which the user listened to during that activity.

The second of the primary aims; ***"To implement functionality which allows comparisons between a user's single running activity and the music they were listening to during the time-period of that activity.",*** was also achieved successfully. The application certainly does provide the user with the information that was intended, they are given a breakdown of their runs and are able to view what songs they were listening to at various segments, including where their heart rate was at its highest or lowest, or where their pace was at its highest or lowest.

The third primary aim of the project; ***"3To implement functionality which compares all of a users' running activities and all of the music listening history for those activities, across a given time frame by the user."*** was also successfully achieved. As previously mentioned, the application allows a user to search for activities within a specified date range, these activities can then be analysed as a collective and the user can view their best segments along with the songs, they listened to during them.

The final primary aim of the project; ***"To design and implement a suitable and user-friendly user interface."*** is difficult to measure in regard to its success. The user interface was developed with a mobile-first approach, and is usable on devices of all form factors, including mobile phones, tablets, and desktops. The designs were kept simple, and the aim was not to overwhelm the user with too much information. Although personally I am happy with the design and believe that this aim was successfully achieved to an extent, I would need to carry out specific user testing sessions in order to record how other people interact with the system and note their views on the suitability and usability of the user interface. From a session I was involved in with demonstrating the system, the feedback was mostly positive with people liking the overall design of the system. The only comments on things which could be changed were some of the animations on the home screen where the recent songs and activities carousels slide automatically after a few seconds, or that there could potentially become an overwhelming amount of data given to the user on the analyse activity screen.

In addition to evaluating the system against the primary aims, the system can also be evaluated against the secondary aims. The first of the secondary aims; ***"To update the application logic to allow for different kinds of activities to be included as opposed to just running. For example, cycling, weightlifting, boxing etc."*** can be classified as successful to an extent. As we discussed, the application does allow the user to view all of their activity types in addition to runs. However, it is much more difficult to analyse these types of activities as they are not broken down into segments. Because of this, the other types of activities can only be analysed in groups, and not as individual activities. Because there is some functionality for analysing these kinds of activities, I would classify this aim as achieved successfully, although I would have liked to see each of these activities being able to be analysed individually like with the runs.

The second secondary aim was ***"To extend the level of detail which is offered by the insights of the music and activity comparison, for example calculating the optimum beats per minute (bpm) of the music the user should listen to when they exercise."*** which unfortunately was not achieved during this project. Due to the limitations of the Last.FM API and it not providing any data on BPM for songs, I was unable to acquire this data. I explored other solutions, Spotify being the best option, however implementing the Spotify API introduced new hurdles such as the user needing to have a Spotify account, which as we discussed previously, isolated the user base. Because of this, I unfortunately had to forgo the implementation of this feature for the purpose of this project.

The third secondary aim was "***To add functionality for song suggestions based on how the user performs whilst listening to certain genres of music.".*** This aim was successfully achieved with the similar suggested songs feature which was discussed in section 4 of this document. The application uses Last.FM to get similar songs to songs which the user listened to during their best segments when analysing an activity.

The final secondary aim was ***"To implement a log-in system which would mean that users would not have to connect their Strava and Last.FM accounts each time they want to use the application.".*** This aim was not achieved during this project. I expected that the implementation of a log in system would be an incredibly large undertaking, and my preconceptions proved to be correct. After implementing the other features previously discussed, I began some initial research into how I could develop a log in system for the application. I quickly realised that this feature would need to be completely de-scoped, as it would be completely infeasible for me to implement this feature as I would need to rebuild a large amount of the system, which would have been too large a task to undertake within the timeframe I had left of the project. Because of this the feature had to be de-scoped. Although the feature was de-scoped, there is very minimal impact on the user because of this. The log-in system would not have provided the user with any additional functionality, and would have only prevented them from having to reconnect their Strava and Last.FM accounts each time they wanted to use it.

The last mentionable points are the two aims which I set during the project's lifecycle. These two aims were:

- ***"To deploy the web-application on a live-hosting service, to make it accessible online to any user."***
- ***"To investigate the incorporation of mapping tools, to display to a user where they have run during their recorded activities."***

Both of these aims were successfully met, as previously discussed, the application is available online on Microsoft's Azure servers and can be accessed by anybody. Mapping of the Strava segments was also implemented on the analyse activity pages using Leaflet.JS.

In conclusion, I would classify the project myself as a success. The majority of the aims that were set have been successfully met, and those that were not, were only de-scoped due to limiting factors such as time constraints or available data. All of the core functionality was successfully implemented, and I was able to include some extra features such as the map.

5.6 Evaluation of Methodology and Tools

Based on the outcome of the several rounds of testing and the evaluation of the system against the project aims, it is clear that the approach and methodology adapted over the course of this project were suitable and appropriately chosen. Adopting the agile methodology as previously discussed allowed the project to be flexible and adapt to any changes which needed to be made to designs, functionality and what tools were being used. If a waterfall methodology had been chosen, I believe that the project would have suffered as a result. By losing the flexibility to add new features during the development process, some of the most defining parts of the system would have been missing from the final product such as the map components.

In terms of the tools and technology used during development, it is my belief that all of the tools which were used were a suitable and correct choice for the project. Using JavaScript alongside the React framework proved to be a good choice as it promoted code reusability, reduced code redundancy, and allowed me to implement some features with a fair degree of complexity. There were however some limitations with using React when dealing with passing data around the system. React is commonly used with another JavaScript framework called Redux (40). Redux is a state management tool (41), that creates a centralized data store to ensure consistent data across the application. The project could have benefited from realising the potential with this tool sooner, as it would have greatly improved the way data is handled within the system and it would have made several of the main processes more efficient. In addition to this it would clean up a lot of the code throughout the system, and heavily reduced the complexity of accessing different data sets within different sections of the system.

Choosing to use Microsoft Azure DevOps for version control was also the perfect tool for the project, as it provided easy integration with Visual Studio Code, and an easy to work with and understand user interface for navigating the different branches and commits. The last tool which I used was called “Postman”. Postman is an tool which allows you to make API requests from a GUI and visualise the data that is being returned (42).

6. Future Work

Although I feel that this project was successful in many ways and achieved most of the aims outlined by the project, there are still a number of ways in which the solution could be improved and built upon in the future. I feel that RaceTracks in its current state is more of a prototype that demonstrates the feasibility of a scalable, and worthwhile system. From my own research, development, and evaluation, I believe that there is a real use for an application like this, which has the potential to appeal to a sizeable user base.

The first improvement and something which I wish I had been able to implement during development, is unit testing. Unit testing is a software development process in which small sections of the system are run through automated tests to ensure correct and properly functioning code (43). Jest is a framework which offers unit testing for JavaScript applications and works with React projects (44). This would be the perfect solution for implementing unit testing on the RaceTracks application which has been already built, and would allow the codebase to be maintainable, scalable, and also ensure that new code doesn't introduce as many new bugs. Furthermore, as part of Azure DevOps' production pipeline, these new unit tests can be set to automatically run before any new code is allowed to merge into the main code branch. This will also aid in preventing the accidental introduction of new bugs.

Once completing work on the addition of unit testing, the application would then be in a more suitable state for the addition of new features. One of the projects secondary aims was to offer a higher level of detail about the music the user listened to during their workouts. Although I was able to provide the user with the music genres, I was not able to display metrics such as beats per minute (bpm), and in turn use this data to find other songs with a matching bpm. This was mainly due to the limitations that we discussed with the APIs, sand o in future iterations of the project, I would like to investigate other available API's in more detail such as the Spotify API. I believe that a great deal of insightful data can be gained from using the Spotify API, and although it was decided that this project would not use it in order to not isolate the user base, it could be possible that in order to improve the analysis' level of detail, it will have to be utilised in some form. The implementation of a new music data source however would require an overhaul of several parts of the system. A lot of the components and functions rely on the data structure provided by Last.FM's API and it is highly unlikely that the data structure from Spotify would match this exactly. Because of this, measures for dealing with a new data structure would need to be put in place. My initial idea would be to create a generic "RaceTracksSong" object, with fields that can be populated from either Spotify's or Last.FM's respective song objects, and then overhaul the rest of the system to utilise this new "RaceTracksSong" object. This would allow for the implementation of a more hexagonal architecture (5) which aims at creating loosely coupled application components that can be easily connected to external software environments.

A further aim of the project was to investigate the addition of a log in system for RaceTracks. Whilst I would have ultimately liked to have implemented a login system, after some investigation this proved to be a very development heavy, complex, and time-consuming feature which the project scope just didn't account for. If this system were to be built upon and made for commercial use, then a log in system would be a vital addition. By allowing users to create a RaceTracks account, they will only then need to connect their external activity tracking and music listening account details to a single account one time. This will negate the need for the user to connect these accounts every single time they wish to use the application. If this feature were to be implemented, there would need to be several new additions to the system architecture to allow for it. The first being a back-end database and the second being an API. The database would need to store all of the users account details and potentially other data such as their Strava activity and Last.FM music data. When storing this data, it is crucial that appropriate security measures such as password hashing are put in place. This database would likely be implemented using Microsoft's ASP.NET framework (45), which would allow for creating a fast and scalable database alongside an API which would be needed for accessing the stored data. The API would need multiple endpoints for reading, writing, and deleting data stored within the database, and the React application will then need to be updated to be able to call these various endpoints. The overall dataflow of the system will also need overhauling to account for the data being stored within the RaceTracks database and using this as an access point rather than Last.FM or Strava. One of the main benefits to storing the data within our own database is that it will heavily reduce the number of API calls made to each of the external API's and would therefore reduce the risk of the application reaching the rate limits for one of the APIs. In addition to this, storing the data ourselves would reduce the reliance on so many other services and would mitigate the risk of the application failing if one of the external API's was unavailable for any reason. Figure 53 below shows a potential dataflow for the overhauled system which would run each time the user logs in.

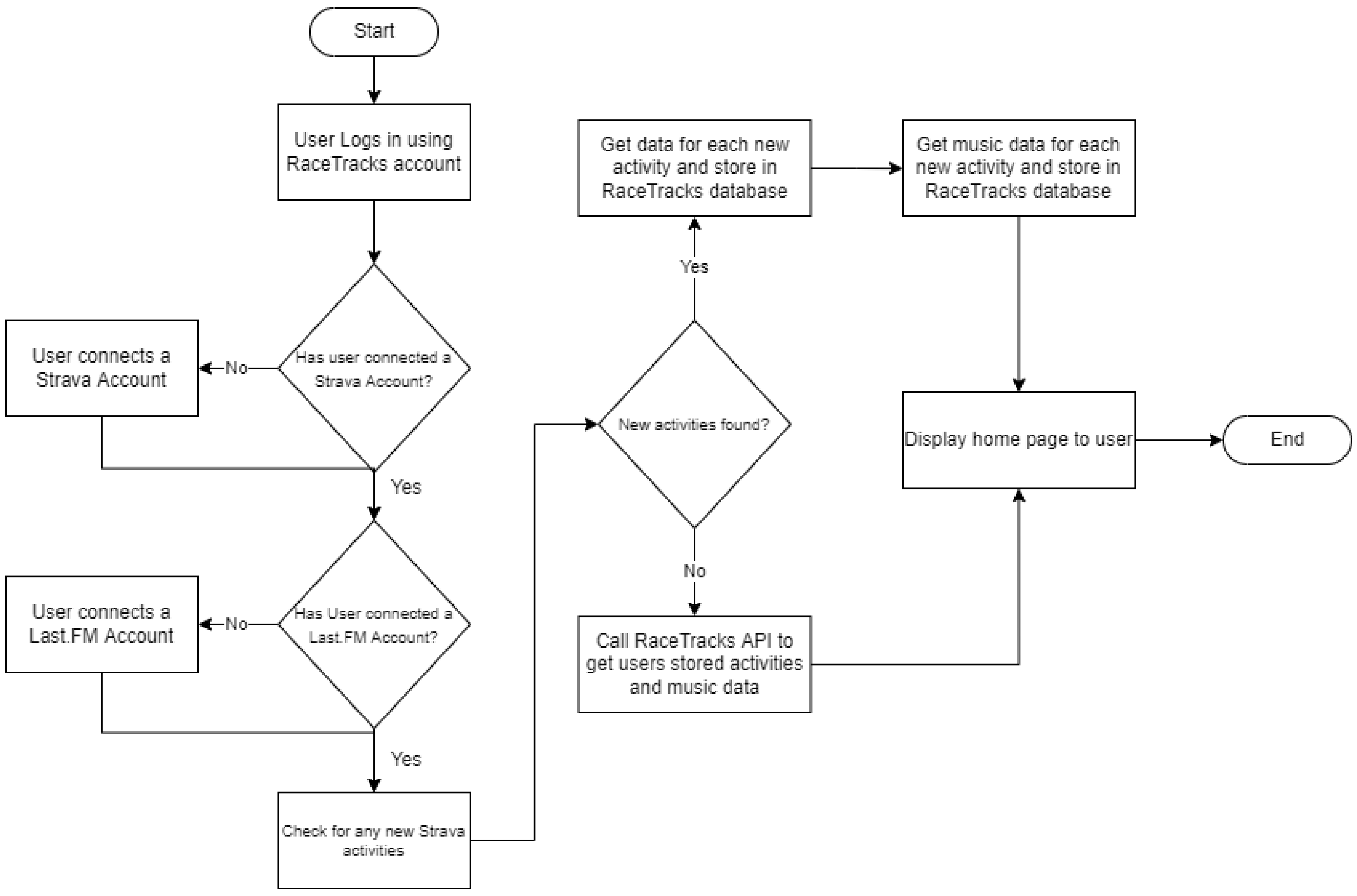

Start
User Logs in using RaceTracks account
Has user connected a Strava Account?
No
User connects a Strava Account
Yes
Has User connected a Last.FM Account?
No
User connects a Last.FM Account
Yes
Check for any new Strava activities
New activities found?
Yes
Get data for each new activity and store in RaceTracks database
Get music data for each new activity and store in RaceTracks database
No
Call RaceTracks API to get users stored activities and music data
Display home page to user
End

Figure 53: UML diagram showing proposed high-level dataflow for each log in after implementation of a new log in system

If the log in system were to be introduced as detailed above, the system would also benefit from the introduction of the Redux JavaScript package. As previously discussed, this was something that I would have liked to include had I only realised it’s potential sooner. If the system were to scale to become any larger than it is in its current state, then the addition of the Redux package for state management would be vital. Some of the final changes I would also like to add in future work are adding new ways of analysing segments and some page styling changes. Although I am happy with the styling work done on the project in its current state, I would like to further improve on this and implement a sleeker and more modern looking design. This could include an overhaul of the colour palette chosen, as well as updated structure of the web pages to reduce potentially overwhelming the user with too much information.

7. Conclusions

The initial aim of this project was to create a responsive web-application that would allow users to connect their activity tracking and music streaming service accounts together in order to gain an understanding of how the music that they listen to can affect their performance. As already evaluated, there were several primary aims which were outlined at the beginning of this project; these were:

1. To create a web-application which will display a user's running activity history as well as their music listening history for each activity.
2. To implement functionality which allows the comparisons between a single user's running activity and the music they were listening to during the time-period of that activity.
3. To implement functionality which compares running activities and music listening history across a given time frame by the user.
4. To design and implement a suitable and user-friendly user interface.

It can be deemed as conclusive that the solution provides the functionality to achieve these aims and does so to an agreeable standard. As a supplement to these primary aims, there were several secondary aims which were also outlined for this project, these were:

1. To update the application logic to allow for different kinds of activities to be included as opposed to just running. For example, cycling, weightlifting, boxing etc.
2. To extend the level of detail which is offered by the insights of the music and activity comparison, for example calculating the optimum beats per minute (bpm) of the music the user should listen to when they exercise.
3. To add functionality for song suggestions based on how the user performs whilst listening to certain genres of music.
4. To implement a log-in system which would mean users would not have to connect their Strava and Last.FM accounts each time they want to use the application.
5. To deploy the web-application on a live-hosting service, to make it accessible online to any user.
6. To investigate the incorporation of mapping tools, to display to a user where they have run during their recorded activities.

Unfortunately, not all of the secondary aims can be noted as achieved. Whilst secondary aims 1, 3, 5 and 6 can be noted as successfully achieved, sadly, secondary aims 2 and 4 were not successfully achieved. This was as discussed previously in detail, due to limitations with the external data sources used, time constraints and the levels of complexity which would come with implementing the features, primarily the log in system.

It is my belief that the project was managed well throughout its entirety. The research, specification and design phases were well documented, and therefore allowed me to make the most out of the time delegated to the implementation. I believe that some of the most notable pieces of work on this project include the location mapping functionality, the work with several external APIs in order to mitigate the issues with genre data due to the lack of its availability from Last.FM and the overall design of the system being suitable for both mobile and desktop views. As a final notable piece of work, having the solution available as a live website is something which I am particularly proud of, as initially this was not something that I had envisioned being able to achieve.

Although there was a great deal of success over the course of this project, there was certainly areas which could have been improved. Had I been able to achieve the final 2 secondary aims, they would have greatly improved the final solution. In addition to this, upon reflection I should have investigated the addition of Redux and Unit tests more with this project. Using both of these tools would have instilled me with more confidence in the final system and would have allowed for a much more dependable and scalable product.

To conclude, a successful development process has led to the creation of a system with a solid foundation for future development. The solution is a prime example of a proof of concept, and I believe there is great potential for a product like it. With the current functionality, and despite the issues posed by data limitations, the solution delivers the user with a tool that allows them to analyse their activities and see how the music they listened to during them had an impact on their performance. They can see segments of their run where they ran their fastest or the slowest, or where their heart rate was at its highest or its lowest. As intended, this project has led to the creation of a solution which allows users to analyse their activities and determine whether or not, “Music makes you run faster”.

8. Reflection on Learning

At the start of this project, I outlined some objectives which I aimed to work towards over the duration of the project. These objectives were mainly based on developing my knowledge with new technical concepts which I had not worked with before and so reflecting on them is an opportunity for me to gauge how much I have been able to learn across this project.

My first objective was ***"To investigate, understand and work with external developer API's available from several companies and see the extent of what data can be obtained from each one.".*** I was really interested prior to starting development, as to how to work with the available API's. There were three different API's which are implemented into the solution, each of which worked in slightly different ways. This proved to be an incredible learning process as I became familiar with how to navigate the available documentation for each API and learn about new concepts such as Client ID's and secret keys. I also learned to understand the importance of data security whilst working with these services and how to make requests to them securely using HTTPS calls.

My second objective was ***"To investigate, understand, and implement authentication workflows using authentication procedures used by external API's.".*** Authentication is something which I have always been aware of, however I had little to no understanding on how or why it should be implemented. Working with the Strava API presented a strong learning experience in integrating an authentication workflow into a system. As we have already discussed, the Strava API implements OAuth2 authentication, and I faced several challenges with the integration of this workflow. Although the process was well documented, I initially struggled to understand it. Although I was able to research example implementations of the Strava API, whilst I was trying to implement it into my own project, the problems I faced were ones which I felt as though I had to tackle myself as I was working as a solo developer. I was able to overcome the problems, and by doing so my understanding of the authentication process was much stronger than it would have been had I not had any problems. By repeatedly reading the documentation and finding better ways to implement a solution, my technical skills and understanding of authentication have improved greatly.

My final objective was ***"To work with available resources to better understand and learn some best practices for front-end JavaScript development.".*** This objective outlines one of my biggest personal goals with this project. Front-end JavaScript web development is something that I wanted to improve my skills in, and by developing an application entirely in JavaScript using the React framework I was able to do just that. Although there are some sections of the system which I would like to have been able to refactor, throughout the development process I was always looking for the best way I could find to implement a new feature. This led me to research ways in which certain things should be handled in JavaScript, one key thing being asynchronous calls and promises as we have discussed previously. I was also able to research several best practices for developing with React, and although I was unable to implement everything which I have learned, my understanding of how I would approach the scenario if I were to tackle this kind of project again in the future has greatly improved.

In addition to these objectives, I would also like to reflect on my general performance across the project. I am particularly proud of my time management throughout the process, and I was consistently ahead of the initial schedule which I proposed in my initial report. I worked heavily on development in the first weeks of the project and was quickly able to meet my criteria for an MVP, and although it might not be reflective in the report, an incredible amount of time went into the development process to make this project possible. I did also make sure to designate plenty of time towards the writing of this report, and so did not find myself in a situation where I felt that I had to rush sections or that they were incomplete. I am also proud of the programming skills that I have developed across the process of this project, and the work that I have accomplished. I hope to use everything that I have learnt from start to finish with this project on future projects.

Table of Abbreviations

Abbreviation	Full Terminology
API	Application Programming Interface
App	Application
BPM	Beats Per Minute
GUI	Graphical User Interface
HTTP	Hyper-Text-Transfer-Protocol
HTTPS	Hyper-Text-Transfer-Protocol-Secure
IDE	Integrated Development Environment
ILB	Internal Load Balancer
UI	User Interface
Web-App	Web Application

Appendices

Additional Wireframe designs:

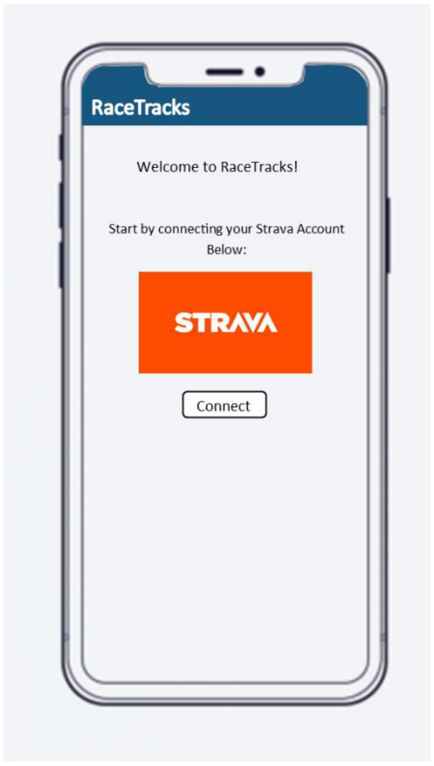


Figure 54: Wireframe showing mobile view of Connect Strava Account Screen

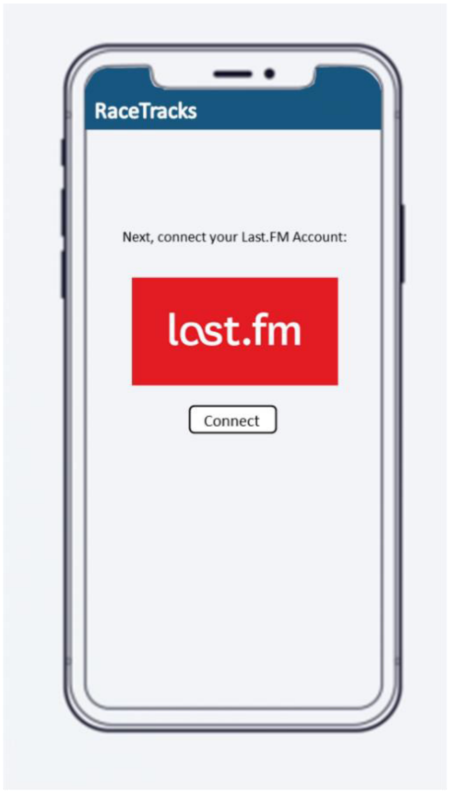


Figure 55: Wireframe showing mobile view of Connect Last.FM Account Screen

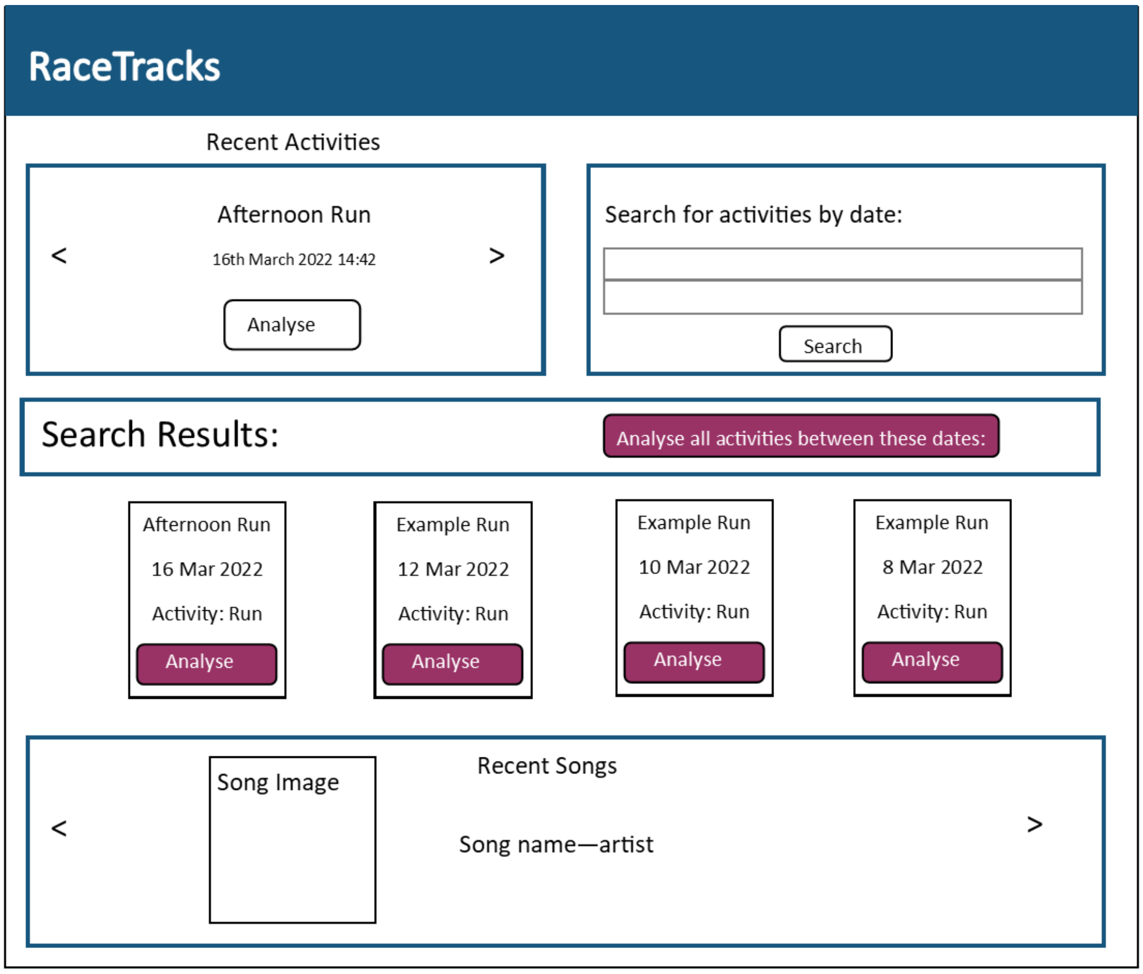


Figure 56: Wireframe showing desktop design for Activity Search Results Screen

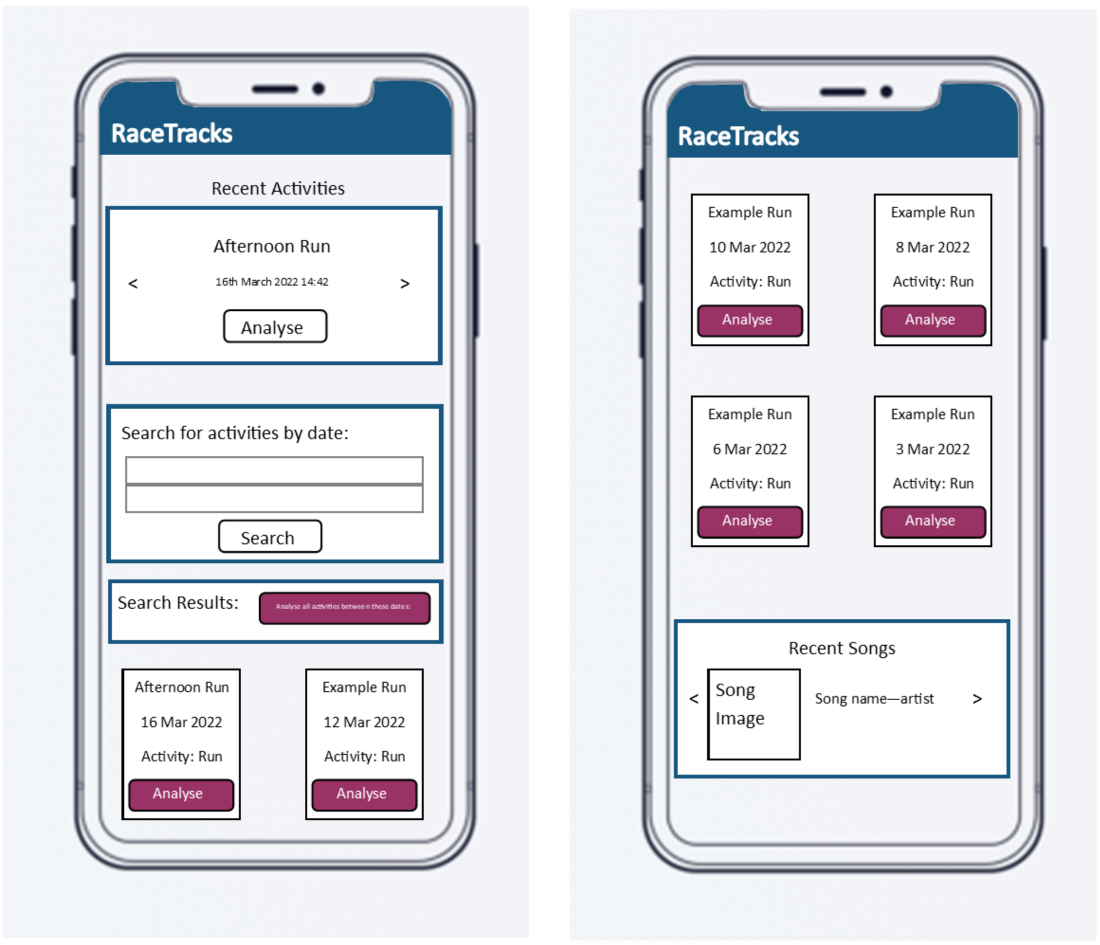


Figure 57: Wireframe showing mobile view for Activity Search Results Screen

Round 1 of Testing screenshots:

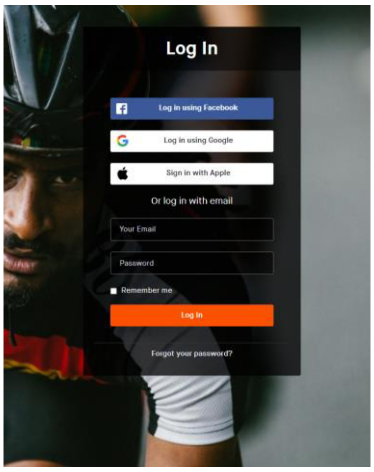
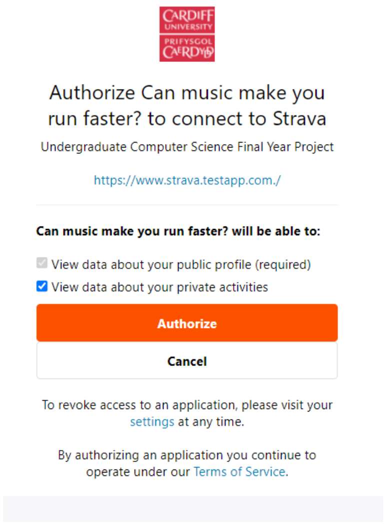


Figure 58: Redirect to Strava log in page



Authorize Can music make you run faster? to connect to Strava

Undergraduate Computer Science Final Year Project

https://www.strava.testapp.com./

Can music make you run faster? will be able to:

- [✓] View data about your public profile (required)
- [✓] View data about your private activities

Authorize

Cancel

To revoke access to an application, please visit your settings at any time.

By authorizing an application you continue to operate under our Terms of Service.

Figure 59: Redirect to Strava authorisation page

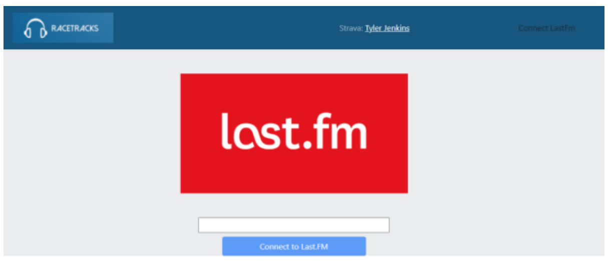


Figure 60: Redirected to connect Last.FM screen after successful Strava authorization

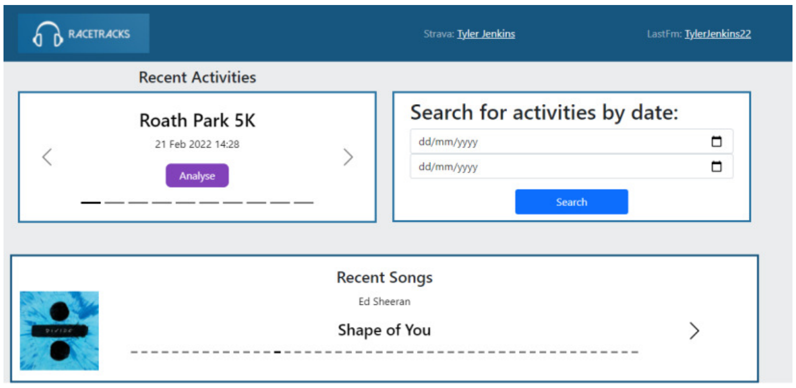
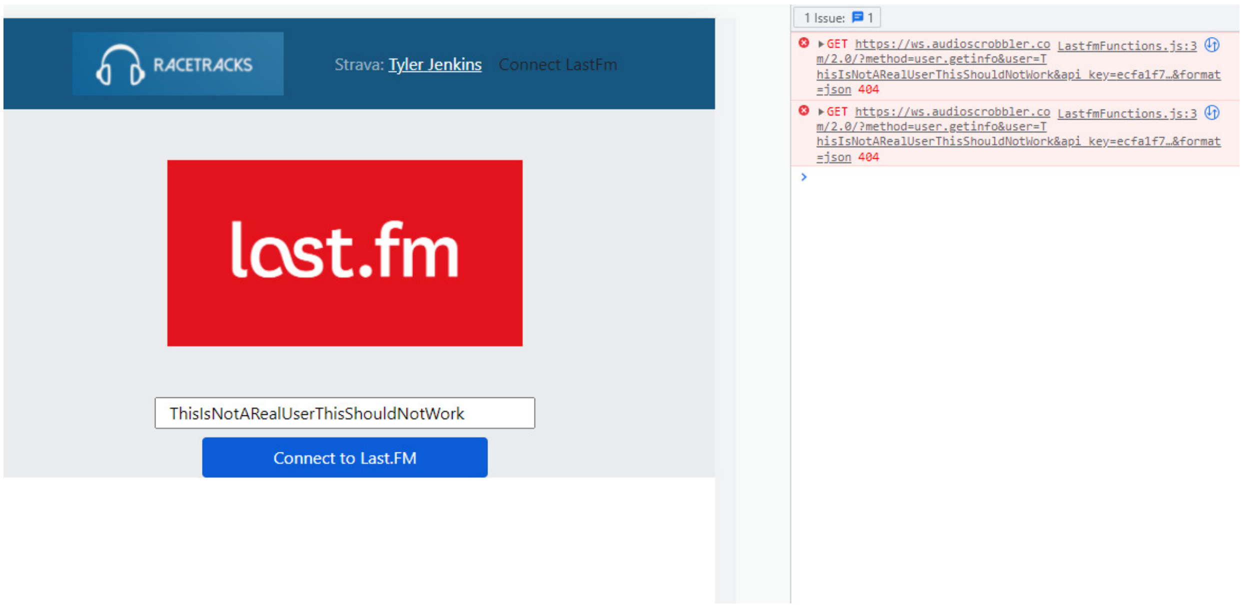
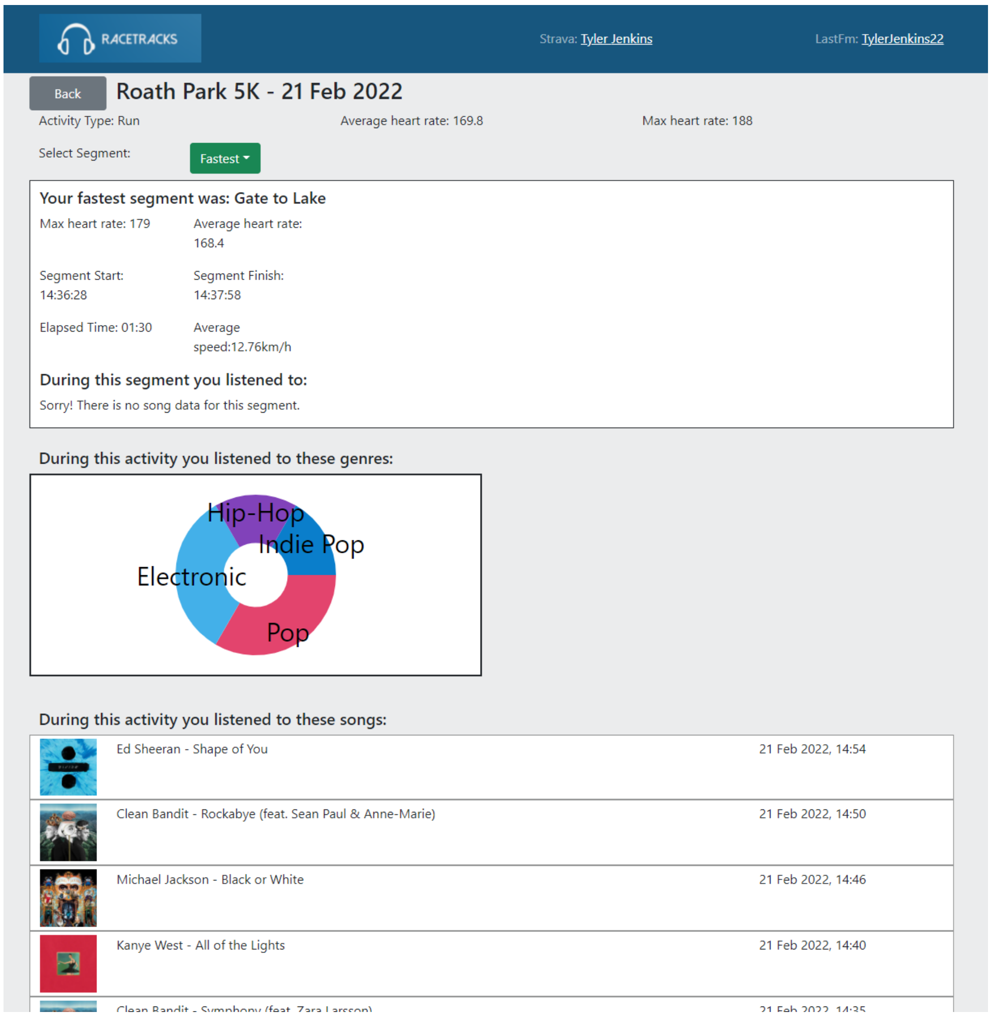


Figure 61: Home page as of first round of testing showing data from connected Strava and Last.FM accounts



```
GET https://ws.audioscrobbler.com/2.0/?method=user.getinfo&user=ThisIsNotARealUserThisShouldNotWork&api_key=ecfa1f7…&format=json 404    LastfmFunctions.js:3
GET https://ws.audioscrobbler.com/2.0/?method=user.getinfo&user=ThisIsNotARealUserThisShouldNotWork&api_key=ecfa1f7…&format=json 404    LastfmFunctions.js:3
```

Figure 62: Entering incorrect data into connect last.fm box returns a 404 message



Roath Park 5K - 21 Feb 2022

Activity Type: Run | Average heart rate: 169.8 | Max heart rate: 188

Select Segment:

Your fastest segment was: Gate to Lake

Max heart rate: 179 | Average heart rate: 168.4

Segment Start: 14:36:28 | Segment Finish: 14:37:58

Elapsed Time: 01:30 | Average speed:12.76km/h

During this segment you listened to:

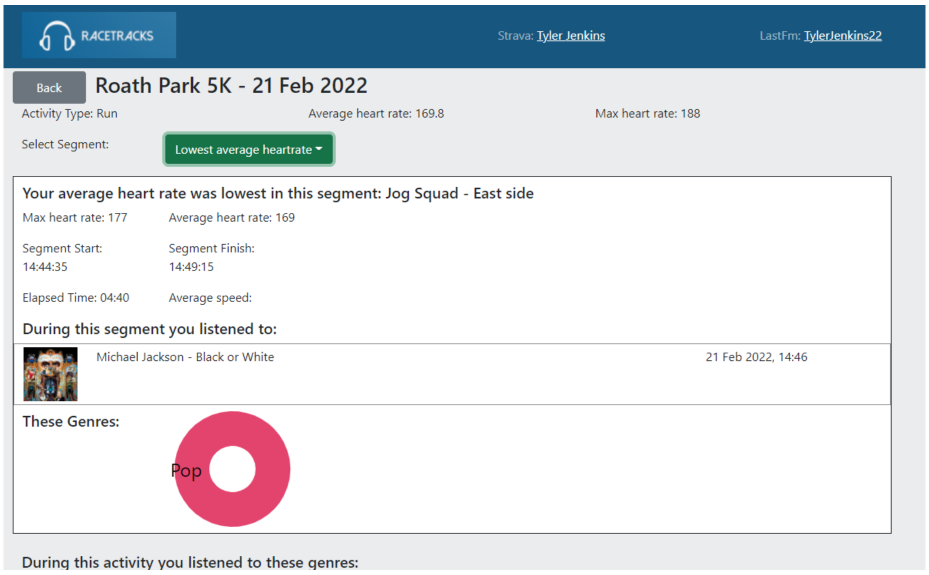
Sorry! There is no song data for this segment.

During this activity you listened to these genres:

During this activity you listened to these songs:

Ed Sheeran - Shape of You	21 Feb 2022, 14:54
Clean Bandit - Rockabye (feat. Sean Paul & Anne-Marie)	21 Feb 2022, 14:50
Michael Jackson - Black or White	21 Feb 2022, 14:46
Kanye West - All of the Lights	21 Feb 2022, 14:40
Clean Bandit - Symphony (feat. Zara Larsson)	21 Feb 2022, 14:35

Figure 63: Analyse "Run" activity screen as of first round of testing



Roath Park 5K - 21 Feb 2022

Activity Type: Run | Average heart rate: 169.8 | Max heart rate: 188

Select Segment:

Your average heart rate was lowest in this segment: Jog Squad - East side

Max heart rate: 177 | Average heart rate: 169

Segment Start: 14:44:35 | Segment Finish: 14:49:15

Elapsed Time: 04:40 | Average speed:

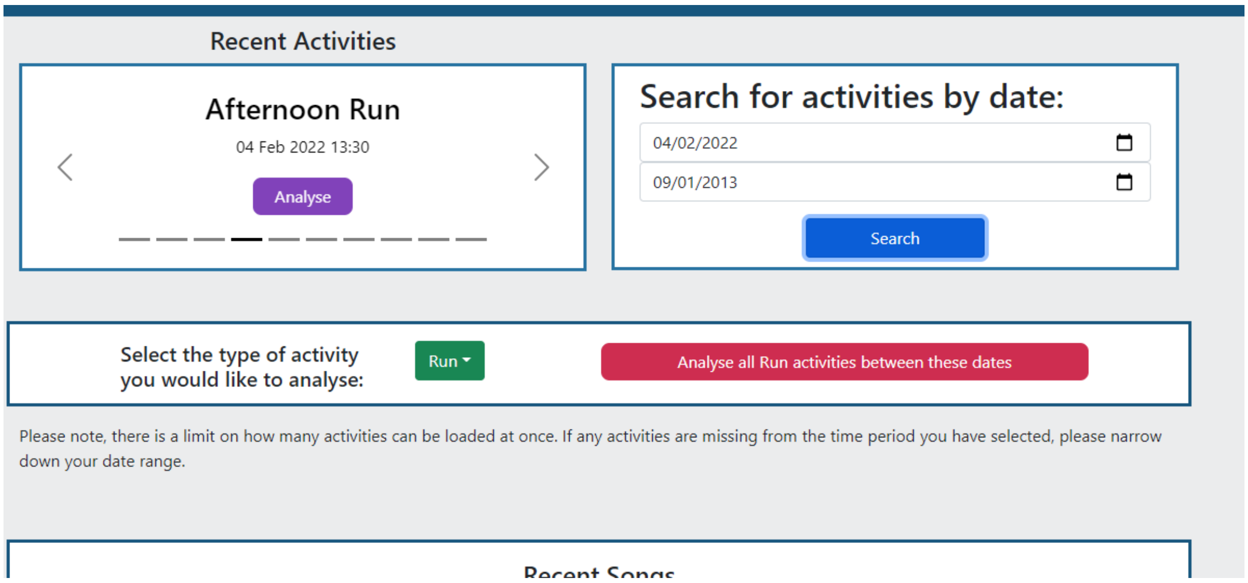
During this segment you listened to:

Michael Jackson - Black or White	21 Feb 2022, 14:46

These Genres:

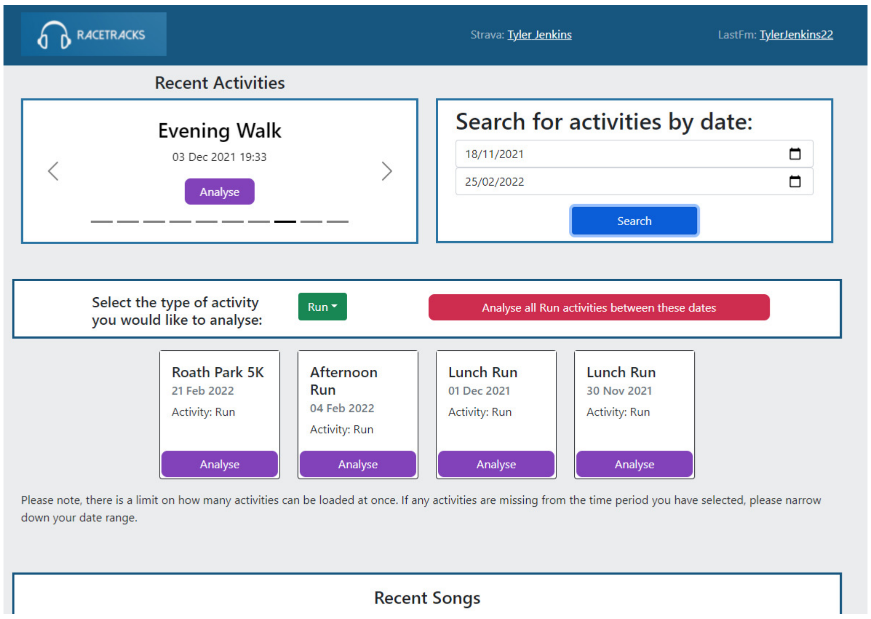
During this activity you listened to these genres:

Figure 64: Shows drop down changed to Lowest average heartrate as of first round of testing


Recent Activities
Afternoon Run
04 Feb 2022 13:30
Analyse
Search for activities by date:
04/02/2022
09/01/2013
Search
Select the type of activity you would like to analyse:
Run
Analyse all Run activities between these dates

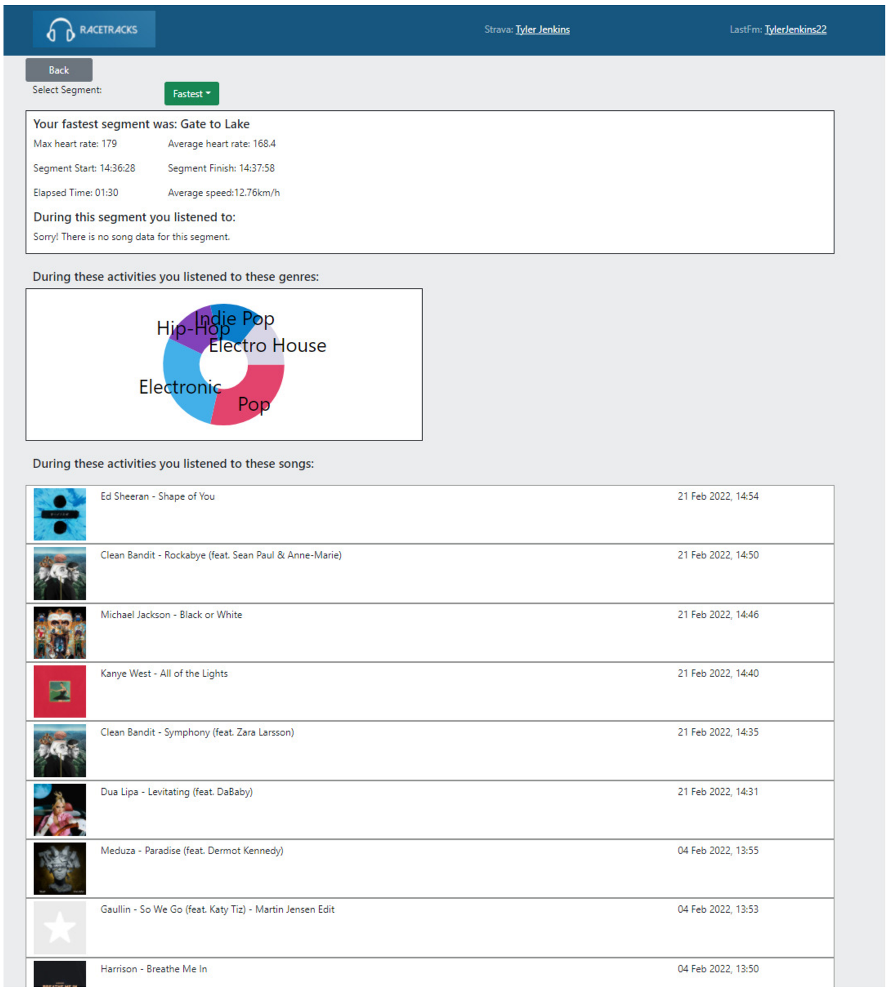
Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Figure 65: Shows invalid date search as of first round of user testing



Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Figure 66: Shows successful activity date range search as of first round of user testing



Your fastest segment was: Gate to Lake

Max heart rate: 179 Average heart rate: 168.4

Segment Start: 14:36:28 Segment Finish: 14:37:58

Elapsed Time: 01:30 Average speed:12.76km/h

During this segment you listened to:

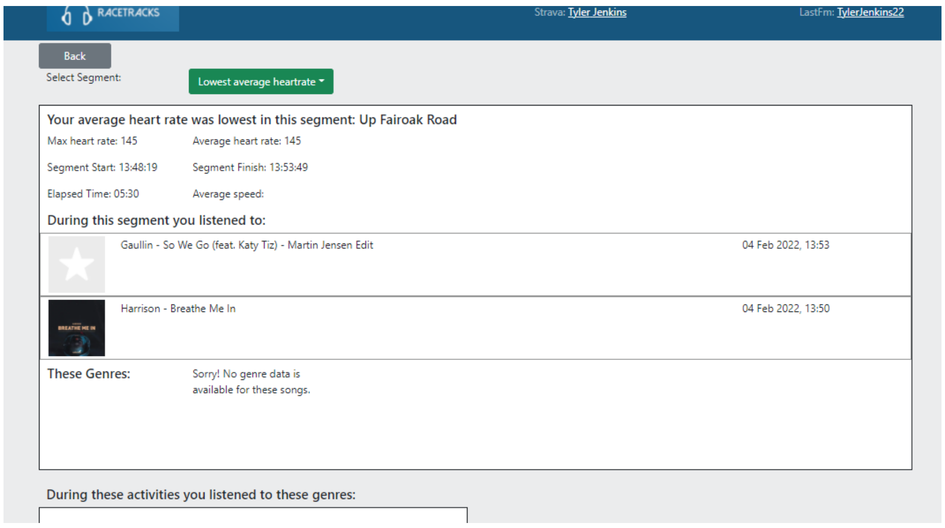
Sorry! There is no song data for this segment.

During these activities you listened to these genres:

During these activities you listened to these songs:

Ed Sheeran - Shape of You	21 Feb 2022, 14:54
Clean Bandit - Rockabye (feat. Sean Paul & Anne-Marie)	21 Feb 2022, 14:50
Michael Jackson - Black or White	21 Feb 2022, 14:46
Kanye West - All of the Lights	21 Feb 2022, 14:40
Clean Bandit - Symphony (feat. Zara Larsson)	21 Feb 2022, 14:35
Dua Lipa - Levitating (feat. DaBaby)	21 Feb 2022, 14:31
Meduza - Paradise (feat. Dermot Kennedy)	04 Feb 2022, 13:55
Gaullin - So We Go (feat. Katy Tiz) - Martin Jensen Edit	04 Feb 2022, 13:53
Harrison - Breathe Me In	04 Feb 2022, 13:50

Figure 67: Shows analyse several activities page as of first round of testing



Your average heart rate was lowest in this segment: Up Fairoak Road

Max heart rate: 145 Average heart rate: 145

Segment Start: 13:48:19 Segment Finish: 13:53:49

Elapsed Time: 05:30 Average speed:

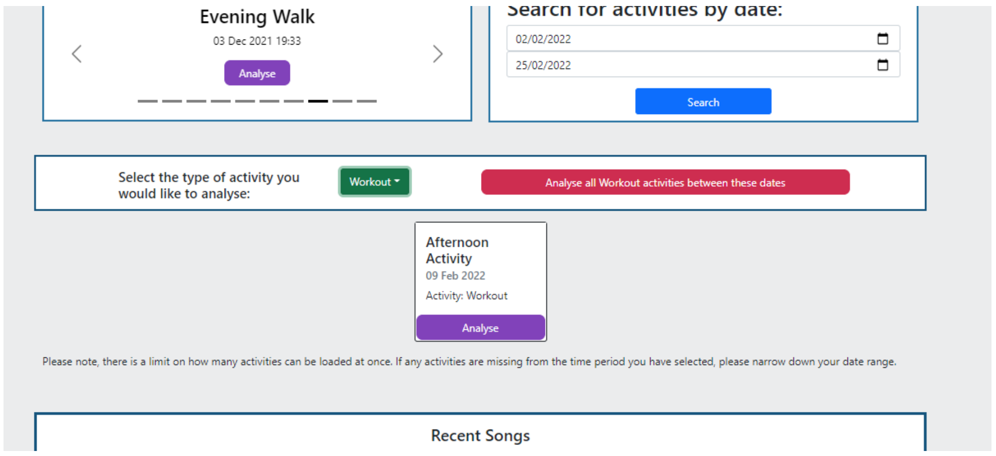
During this segment you listened to:

Gaullin - So We Go (feat. Katy Tiz) - Martin Jensen Edit	04 Feb 2022, 13:53
Harrison - Breathe Me In	04 Feb 2022, 13:50

These Genres: Sorry! No genre data is available for these songs.

During these activities you listened to these genres:

Figure 68: Shows successful segment filter change on analysing multiple activities as of first round of testing


Search for activities by date:

02/02/2022

25/02/2022

Search

Evening Walk

03 Dec 2021 19:33

Analyse

Select the type of activity you would like to analyse: Workout

Analyse all Workout activities between these dates

Afternoon Activity

09 Feb 2022

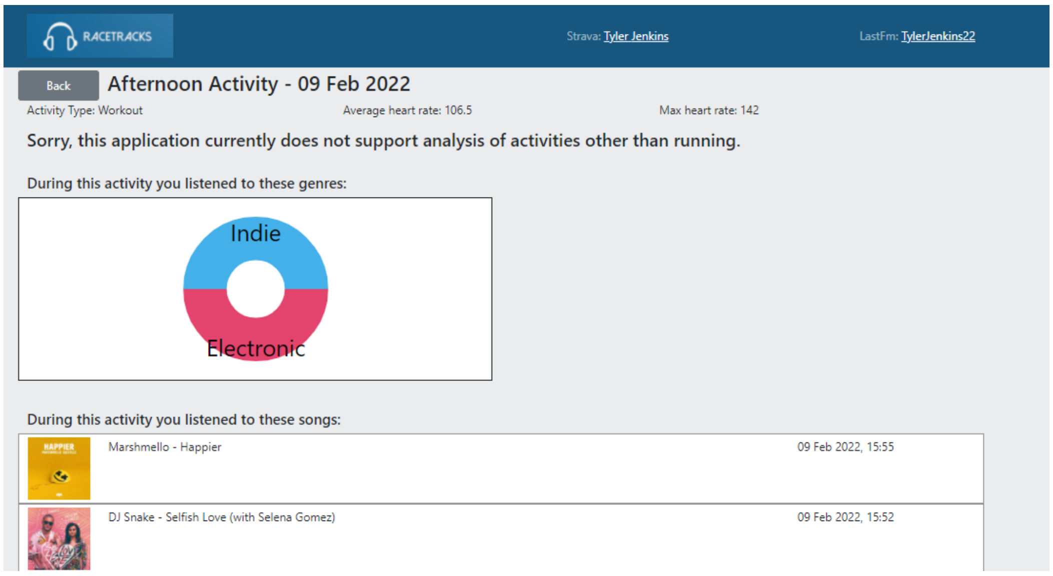
Activity: Workout

Analyse

Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Recent Songs

Figure 69: Shows activity filter changed to "Workout" as of first round of testing


RACETRACKS Strava: Tyler Jenkins LastFm: TylerJenkins22

Back Afternoon Activity - 09 Feb 2022

Activity Type: Workout Average heart rate: 106.5 Max heart rate: 142

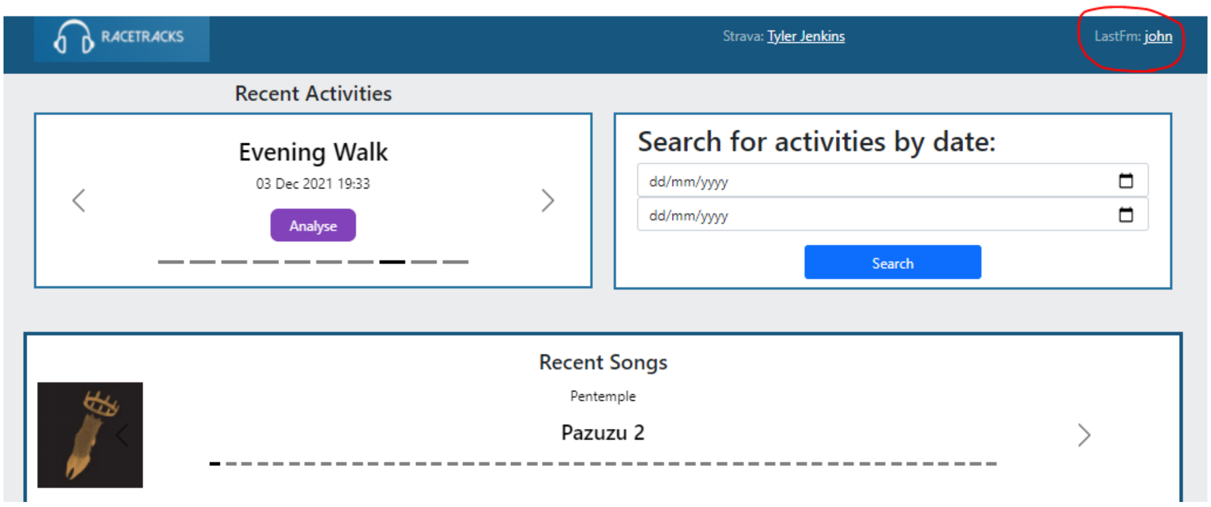
Sorry, this application currently does not support analysis of activities other than running.

During this activity you listened to these genres:

During this activity you listened to these songs:

Marshmello - Happier	09 Feb 2022, 15:55
DJ Snake - Selfish Love (with Selena Gomez)	09 Feb 2022, 15:52

Figure 70: Shows attempt to analyse a non-running activity as of first round of testing


RACETRACKS Strava: Tyler Jenkins LastFm: john

Recent Activities

Evening Walk

03 Dec 2021 19:33

Analyse

Search for activities by date:

dd/mm/yyyy

dd/mm/yyyy

Search

Recent Songs

Pentemple

Pazuzu 2

Figure 71: Shows another user's account connected as Last.FM account as of first round of testing

Round 2 of testing Screenshots:

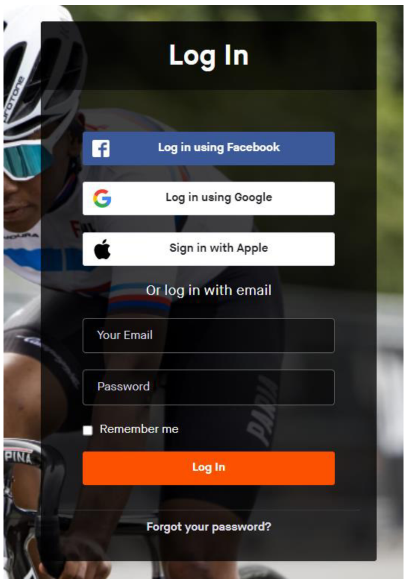
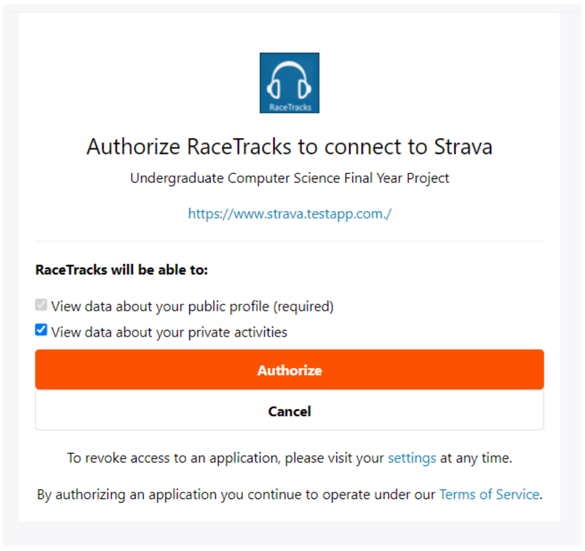


Figure 72: Redirect to Strava log in page



Authorize RaceTracks to connect to Strava

Undergraduate Computer Science Final Year Project

https://www.strava.testapp.com./

RaceTracks will be able to:

- [✓] View data about your public profile (required)
- [✓] View data about your private activities

To revoke access to an application, please visit your settings at any time.

By authorizing an application you continue to operate under our Terms of Service.

Figure 73: Authorize RaceTracks access to Strava screen

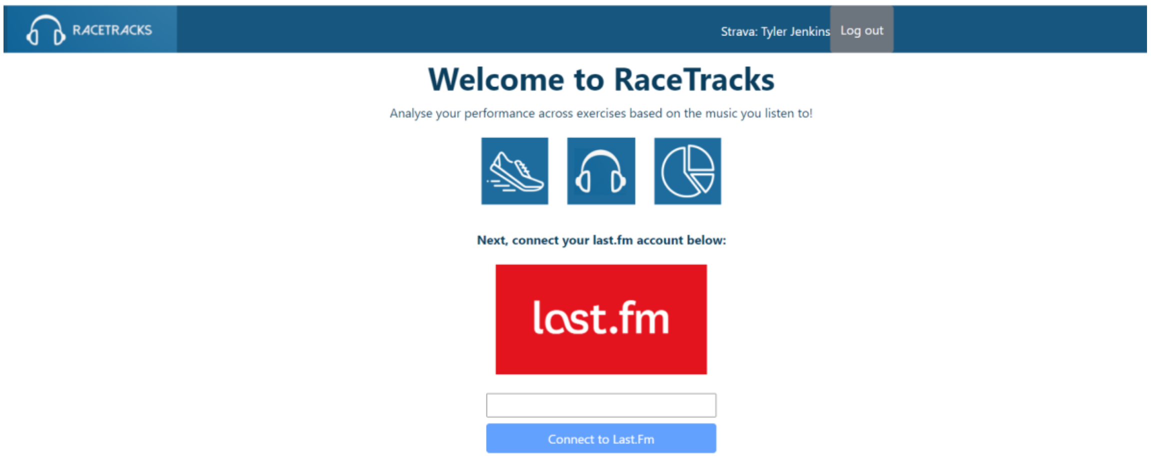

RACETRACKS
Strava: Tyler Jenkins
Log out
Welcome to RaceTracks
Analyse your performance across exercises based on the music you listen to!
Next, connect your last.fm account below:
last.fm
Connect to Last.Fm

Figure 74: Screenshot showing connect Last.FM account screen

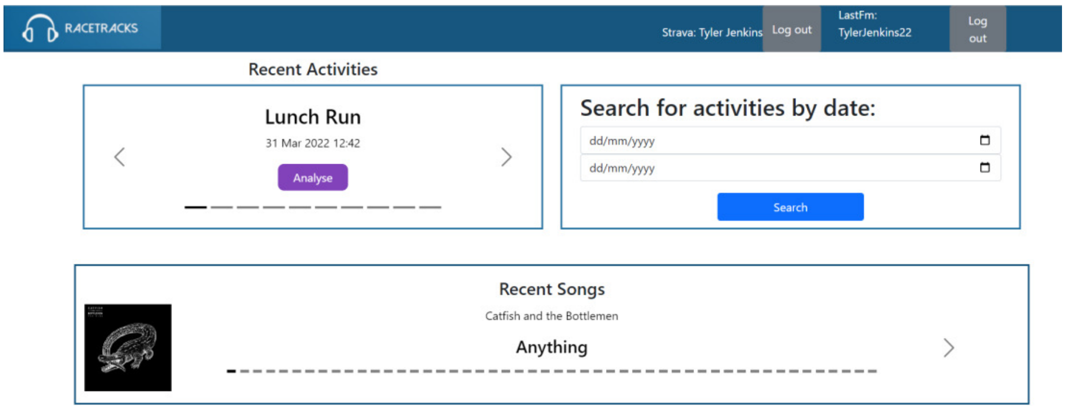


Figure 75: Screenshot showing home page as of second round of self-testing

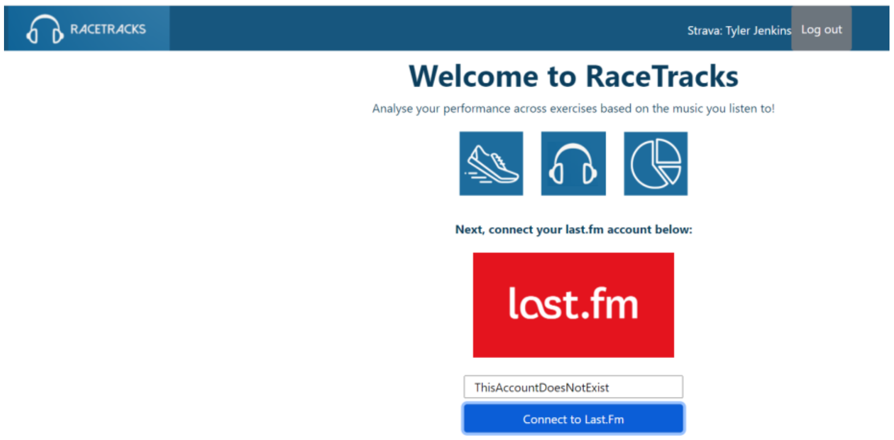
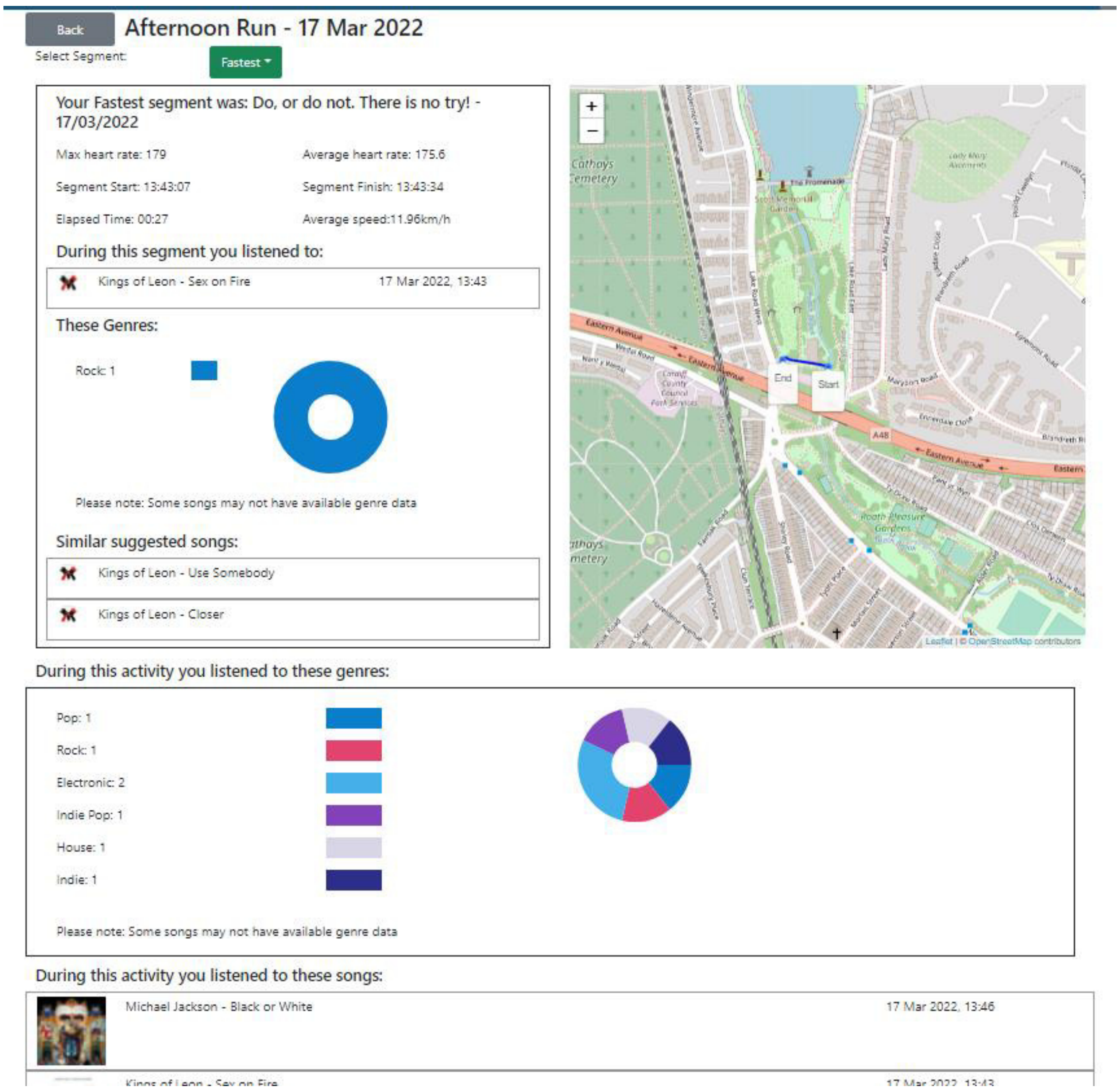

RACETRACKS
Strava: Tyler Jenkins
Log out
Welcome to RaceTracks
Analyse your performance across exercises based on the music you listen to!
Next, connect your last.fm account below:
last.fm
ThisAccountDoesNotExist
Connect to Last.Fm

Figure 76: Screenshot showing connect Last.FM account screen with invalid username entered



Back **Afternoon Run - 17 Mar 2022**

Select Segment: Fastest

Your Fastest segment was: Do, or do not. There is no try! - 17/03/2022

Max heart rate: 179	Average heart rate: 175.6
Segment Start: 13:43:07	Segment Finish: 13:43:34
Elapsed Time: 00:27	Average speed:11.96km/h

During this segment you listened to:

Kings of Leon - Sex on Fire 17 Mar 2022, 13:43

These Genres:

Rock: 1 🟦

Please note: Some songs may not have available genre data

Similar suggested songs:

Kings of Leon - Use Somebody

Kings of Leon - Closer

During this activity you listened to these genres:

Pop: 1	🟦
Rock: 1	🟥
Electronic: 2	🟦
Indie Pop: 1	🟪
House: 1	⬜
Indie: 1	🟦

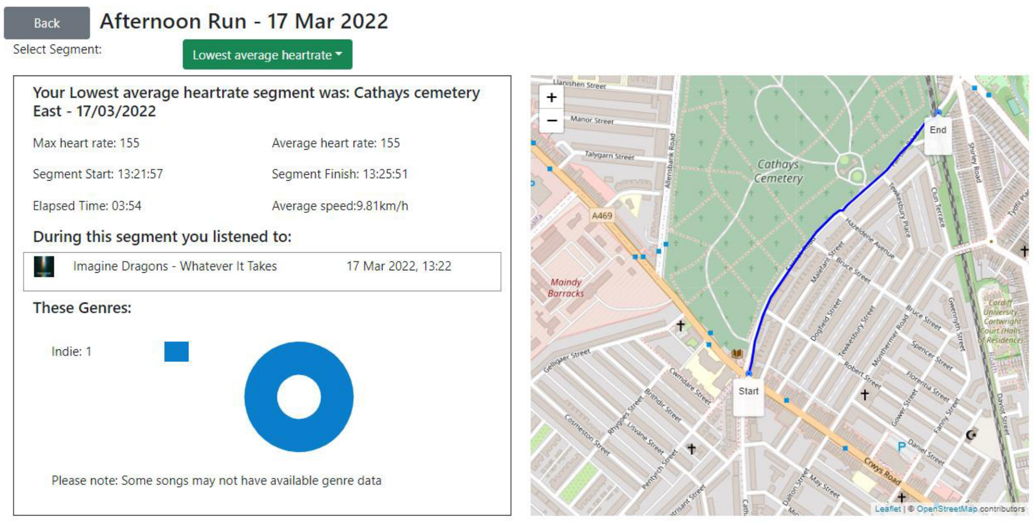
Please note: Some songs may not have available genre data

During this activity you listened to these songs:

Michael Jackson - Black or White 17 Mar 2022, 13:46

Kings of Leon - Sex on Fire 17 Mar 2022, 13:43

Figure 77: Screenshot showing Analyse Activity page after second round of self-testing



Back **Afternoon Run - 17 Mar 2022**

Select Segment: Lowest average heartrate

Your Lowest average heartrate segment was: Cathays cemetery East - 17/03/2022

Max heart rate: 155	Average heart rate: 155
Segment Start: 13:21:57	Segment Finish: 13:25:51
Elapsed Time: 03:54	Average speed:9.81km/h

During this segment you listened to:

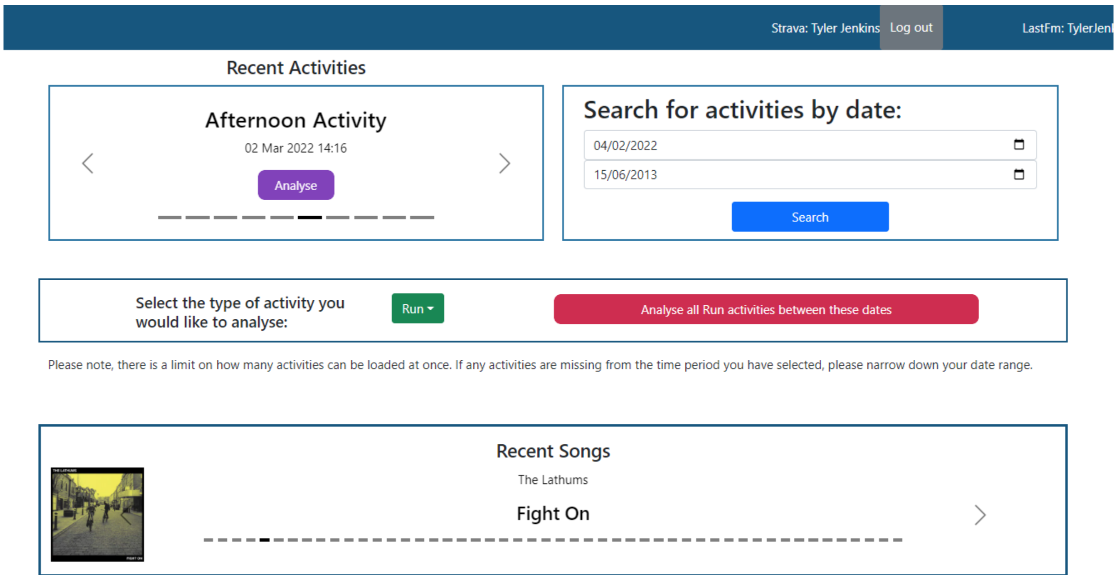
Imagine Dragons - Whatever It Takes 17 Mar 2022, 13:22

These Genres:

Indie: 1 🟦

Please note: Some songs may not have available genre data

Figure 78: Screenshot showing the map updating to show the selected segments location


Strava: Tyler Jenkins Log out LastFm: TylerJen

Recent Activities

Afternoon Activity

02 Mar 2022 14:16

Analyse

Search for activities by date:

04/02/2022

15/06/2013

Search

Select the type of activity you would like to analyse: Run

Analyse all Run activities between these dates

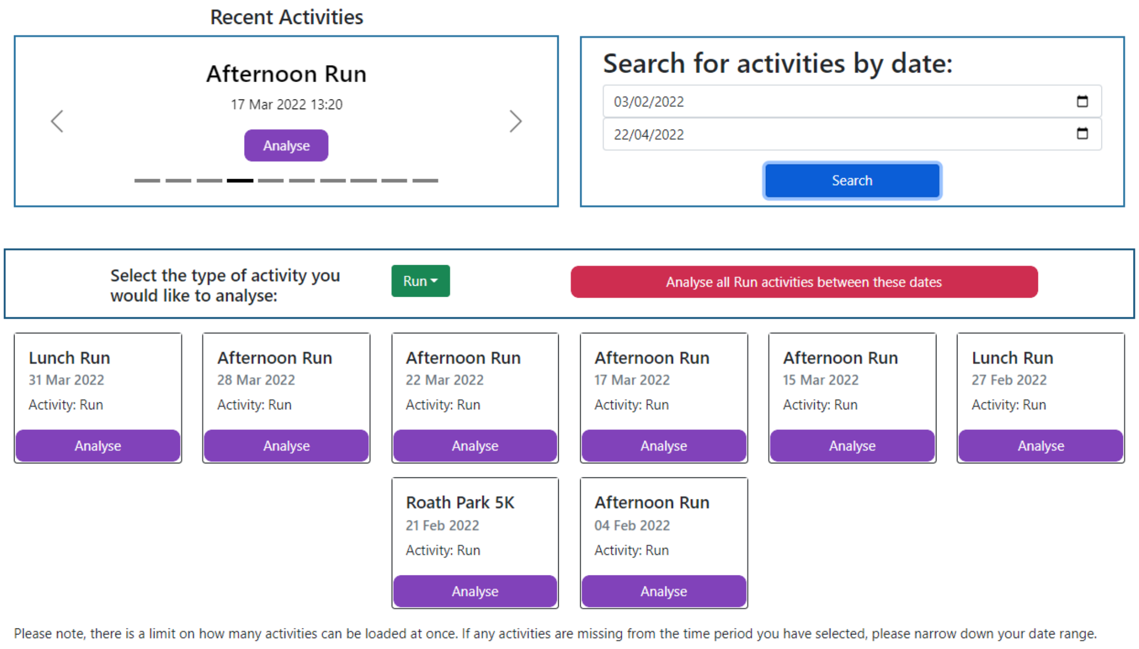
Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Recent Songs

The Lathums

Fight On

Figure 79: Screenshot showing invalid date search after second round of self testing


Recent Activities

Afternoon Run

17 Mar 2022 13:20

Analyse

Search for activities by date:

03/02/2022

22/04/2022

Search

Select the type of activity you would like to analyse: Run

Analyse all Run activities between these dates

Lunch Run
31 Mar 2022
Activity: Run
Analyse

Afternoon Run
28 Mar 2022
Activity: Run
Analyse

Afternoon Run
22 Mar 2022
Activity: Run
Analyse

Afternoon Run
17 Mar 2022
Activity: Run
Analyse

Afternoon Run
15 Mar 2022
Activity: Run
Analyse

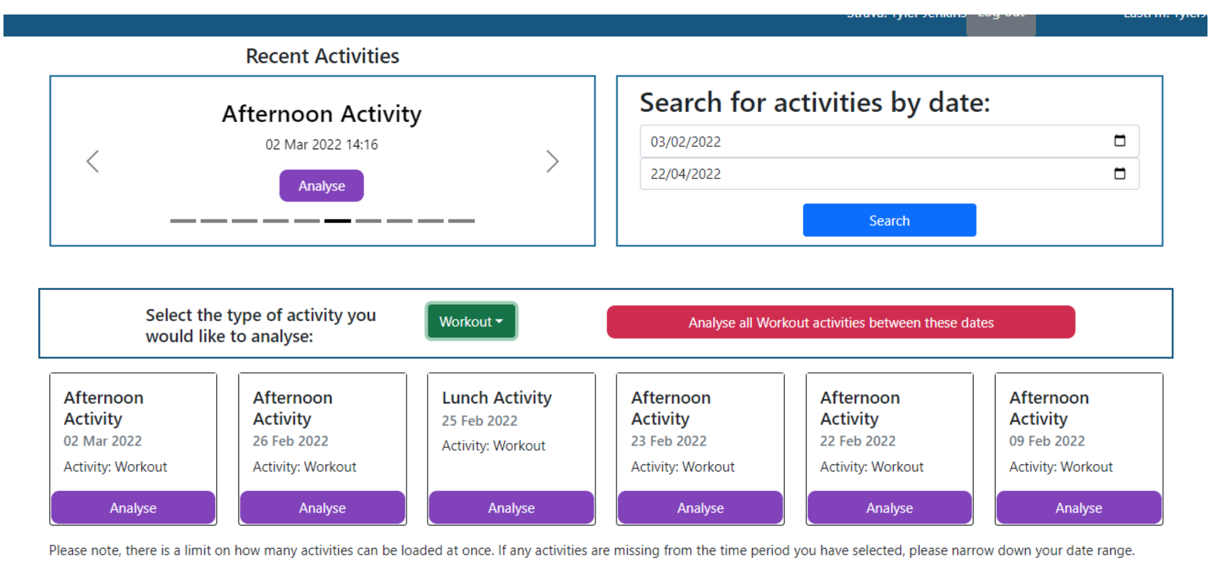
Lunch Run
27 Feb 2022
Activity: Run
Analyse

Roath Park 5K
21 Feb 2022
Activity: Run
Analyse

Afternoon Run
04 Feb 2022
Activity: Run
Analyse

Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Figure 80: Screenshot showing activity date search results after second round of self-testing



Recent Activities

Afternoon Activity

02 Mar 2022 14:16

Analyse

Search for activities by date:

03/02/2022

22/04/2022

Search

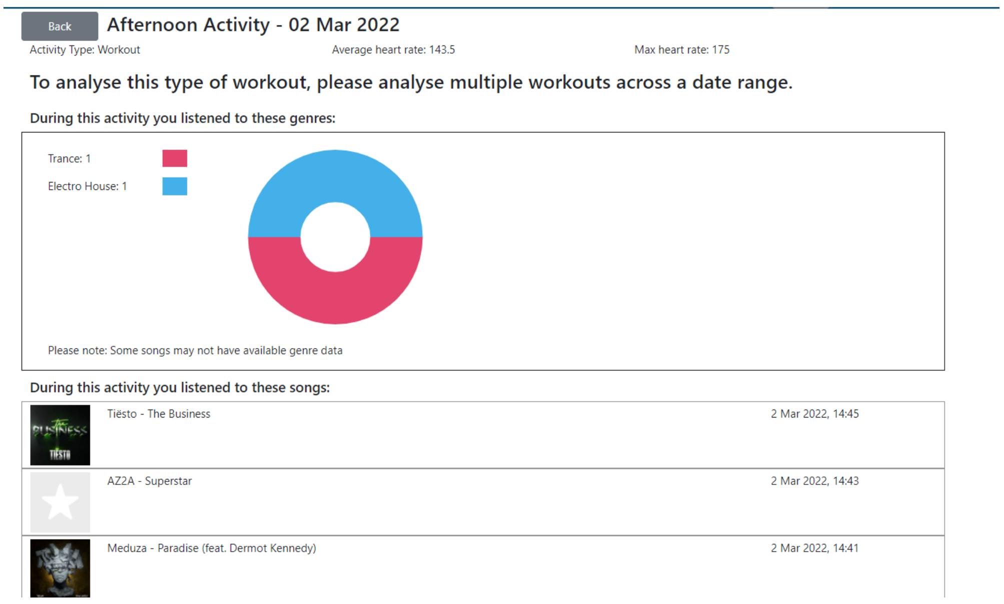
Select the type of activity you would like to analyse: Workout

Analyse all Workout activities between these dates

Afternoon Activity	Afternoon Activity	Lunch Activity	Afternoon Activity	Afternoon Activity	Afternoon Activity
02 Mar 2022	26 Feb 2022	25 Feb 2022	23 Feb 2022	22 Feb 2022	09 Feb 2022
Activity: Workout	Activity: Workout	Activity: Workout	Activity: Workout	Activity: Workout	Activity: Workout
Analyse	Analyse	Analyse	Analyse	Analyse	Analyse

Please note, there is a limit on how many activities can be loaded at once. If any activities are missing from the time period you have selected, please narrow down your date range.

Figure 81: Screenshot showing "Workout" activities as of second round of self-testing



Back

Afternoon Activity - 02 Mar 2022

Activity Type: Workout　Average heart rate: 143.5　Max heart rate: 175

To analyse this type of workout, please analyse multiple workouts across a date range.

During this activity you listened to these genres:

Trance: 1 🟥

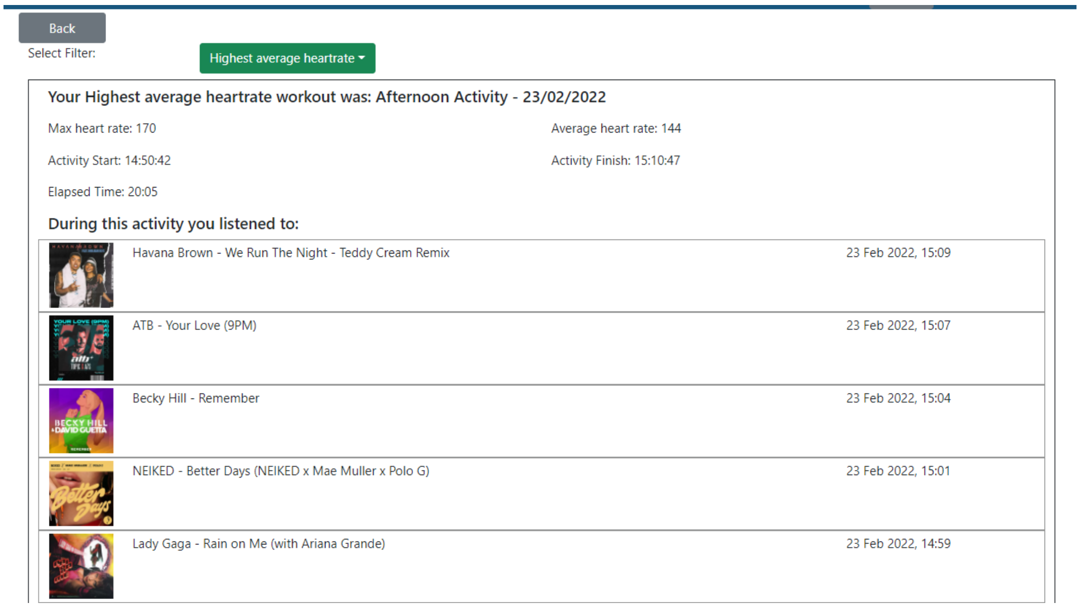
Electro House: 1 🟦

Please note: Some songs may not have available genre data

During this activity you listened to these songs:

Tiësto - The Business	2 Mar 2022, 14:45
AZ2A - Superstar	2 Mar 2022, 14:43
Meduza - Paradise (feat. Dermot Kennedy)	2 Mar 2022, 14:41

Figure 82: Screenshot showing analyse workout activity page as of second round of self-testing



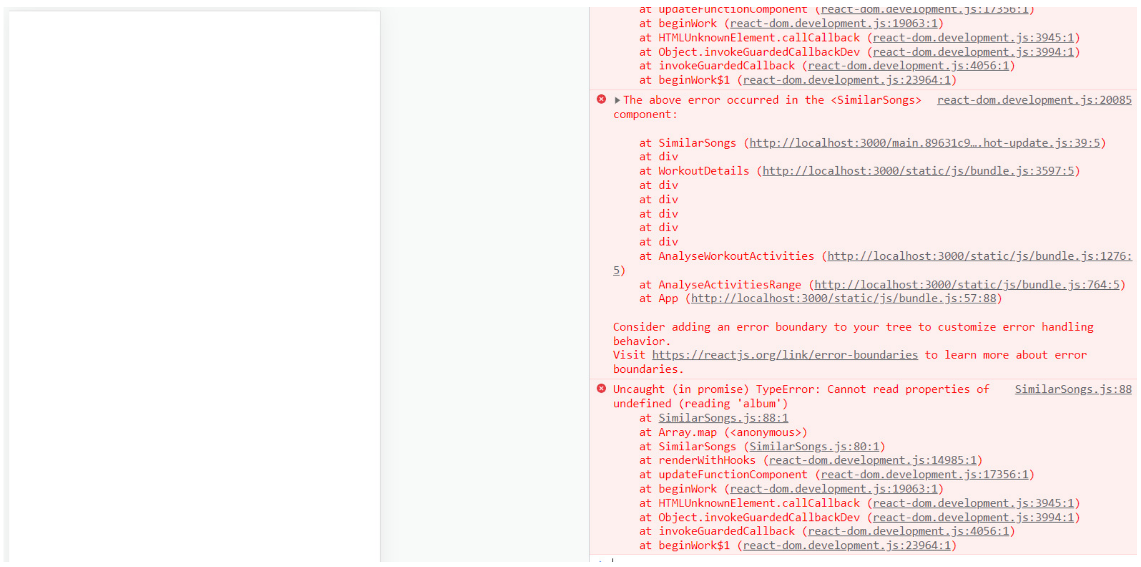
Your Highest average heartrate workout was: Afternoon Activity - 23/02/2022

Max heart rate: 170 | Average heart rate: 144
Activity Start: 14:50:42 | Activity Finish: 15:10:47
Elapsed Time: 20:05

During this activity you listened to:

Havana Brown - We Run The Night - Teddy Cream Remix	23 Feb 2022, 15:09
ATB - Your Love (9PM)	23 Feb 2022, 15:07
Becky Hill - Remember	23 Feb 2022, 15:04
NEIKED - Better Days (NEIKED x Mae Muller x Polo G)	23 Feb 2022, 15:01
Lady Gaga - Rain on Me (with Ariana Grande)	23 Feb 2022, 14:59

Figure 83: Screenshot showing analyse workout activities page as of second round of self-testing



```
    at updateFunctionComponent (react-dom.development.js:17356:1)
    at beginWork (react-dom.development.js:19063:1)
    at HTMLUnknownElement.callCallback (react-dom.development.js:3945:1)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:3994:1)
    at invokeGuardedCallback (react-dom.development.js:4056:1)
    at beginWork$1 (react-dom.development.js:23964:1)
▸The above error occurred in the <SimilarSongs>   react-dom.development.js:20085
component:

    at SimilarSongs (http://localhost:3000/main.89631c9….hot-update.js:39:5)
    at div
    at WorkoutDetails (http://localhost:3000/static/js/bundle.js:3597:5)
    at div
    at div
    at div
    at div
    at div
    at AnalyseWorkoutActivities (http://localhost:3000/static/js/bundle.js:1276:
5)
    at AnalyseActivitiesRange (http://localhost:3000/static/js/bundle.js:764:5)
    at App (http://localhost:3000/static/js/bundle.js:57:88)

Consider adding an error boundary to your tree to customize error handling
behavior.
Visit https://reactjs.org/link/error-boundaries to learn more about error
boundaries.
Uncaught (in promise) TypeError: Cannot read properties of   SimilarSongs.js:88
undefined (reading 'album')
    at SimilarSongs.js:88:1
    at Array.map (<anonymous>)
    at SimilarSongs (SimilarSongs.js:80:1)
    at renderWithHooks (react-dom.development.js:14985:1)
    at updateFunctionComponent (react-dom.development.js:17356:1)
    at beginWork (react-dom.development.js:19063:1)
    at HTMLUnknownElement.callCallback (react-dom.development.js:3945:1)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:3994:1)
    at invokeGuardedCallback (react-dom.development.js:4056:1)
    at beginWork$1 (react-dom.development.js:23964:1)
```

Figure 84: Screenshot showing error from testing on analyse workouts page during second round of self-testing

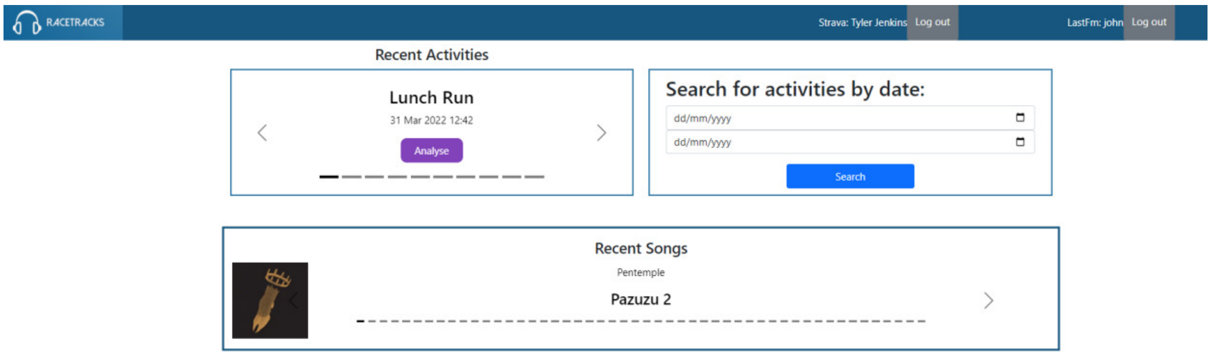


Figure 85: Screenshot showing another user's Last.FM account connected following second round of self-testing

References

1. Strava Revenue and Usage Statistics (2022) [Internet]. Business of Apps. 2020 [cited 2022 Mar 11]. Available from: https://www.businessofapps.com/data/strava-statistics/

2. Spotify — About Spotify [Internet]. [cited 2022 Mar 11]. Available from: https://newsroom.spotify.com/company-info/

3. Hoory L. Agile vs. Waterfall: Which Project Management Methodology Is Best For You? [Internet]. Forbes Advisor. 2022 [cited 2022 May 2]. Available from: https://www.forbes.com/advisor/business/agile-vs-waterfall-methodology/

4. How Does My Phone Track My Steps? [Internet]. MUO. 2021 [cited 2022 Mar 11]. Available from: https://www.makeuseof.com/how-does-my-phone-track-my-steps/

5. Hexagonal architecture (software). In: Wikipedia [Internet]. 2022 [cited 2022 Apr 6]. Available from: https://en.wikipedia.org/w/index.php?title=Hexagonal_architecture_(software)&oldid=1075385603

6. Data Importers | Strava Apps – There's one for every athlete. [Internet]. [cited 2022 Mar 11]. Available from: https://www.strava.com/apps/data-importer

7. Track My Music [Internet]. Last.fm. [cited 2022 Mar 11]. Available from: https://www.last.fm/about/trackmymusic

8. The End-User Stakeholders [Internet]. [cited 2022 Mar 15]. Available from: https://www.thesalesblog.com/blog/the-end-user-stakeholders

9. Perform | AI-Powered Music Workouts [Internet]. [cited 2022 Mar 15]. Available from: https://perform.fm

10. RockMyRun, Music that Moves You [Internet]. 2022 [cited 2022 Mar 16]. Available from: https://www.rockmyrun.com/

11. React – A JavaScript library for building user interfaces [Internet]. [cited 2022 Mar 16]. Available from: https://reactjs.org/

12. React (JavaScript library). In: Wikipedia [Internet]. 2022 [cited 2022 Mar 11]. Available from: https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=1076081030

13. Create a New React App – React [Internet]. [cited 2022 Mar 11]. Available from: https://reactjs.org/docs/create-a-new-react-app.html

14. Babel · The compiler for next generation JavaScript [Internet]. [cited 2022 Mar 16]. Available from: https://babeljs.io/

15. webpack [Internet]. webpack. [cited 2022 Mar 16]. Available from: https://webpack.js.org/

16. Node.js. Node.js [Internet]. Node.js. [cited 2022 Mar 16]. Available from: https://nodejs.org/en/

17. contributors MO Jacob Thornton, and Bootstrap. Bootstrap [Internet]. [cited 2022 Mar 11]. Available from: https://getbootstrap.com/

18. React-Bootstrap [Internet]. [cited 2022 Mar 16]. Available from: https://react-bootstrap.github.io/

19. Leaflet — an open-source JavaScript library for interactive maps [Internet]. [cited 2022 Mar 16]. Available from: https://leafletjs.com/

20. React Leaflet | React Leaflet [Internet]. [cited 2022 Mar 16]. Available from: https://react-leaflet.js.org/

21. Home [Internet]. [cited 2022 Mar 16]. Available from: https://moment.github.io/luxon/#/?id=luxon

22. react-minimal-pie-chart [Internet]. npm. [cited 2022 Mar 16]. Available from: https://www.npmjs.com/package/react-minimal-pie-chart

23. Azure DevOps Services | Microsoft Azure [Internet]. [cited 2022 Mar 16]. Available from: https://azure.microsoft.com/en-us/services/devops/

24. Business requirements. In: Wikipedia [Internet]. 2022 [cited 2022 Apr 28]. Available from: https://en.wikipedia.org/w/index.php?title=Business_requirements&oldid=1083549866

25. nishan_clrbridge. Mobile App Design: Designing for a Web App vs. Native App [Internet]. Clearbridge Mobile. 2020 [cited 2022 Mar 17]. Available from: https://clearbridgemobile.com/mobile-app-design-designing-for-a-web-app-vs-native-app/

26. Advantages of using blue in your web design – Pumpkin Web Design Manchester [Internet]. [cited 2022 May 2]. Available from: https://www.pumpkinwebdesign.com/web-design-manchester/advantages-of-using-blue-in-your-web-design/

27. Strava Developers [Internet]. [cited 2022 Mar 17]. Available from: https://developers.strava.com/docs/authentication/

28. Strava Developers [Internet]. [cited 2022 Apr 4]. Available from: https://developers.strava.com/docs/rate-limits/

29. API Terms of Service [Internet]. Last.fm. [cited 2022 Apr 4]. Available from: https://www.last.fm/api/tos

30. json-server [Internet]. npm. [cited 2022 Apr 4]. Available from: https://www.npmjs.com/package/json-server

31. jquery - When is JavaScript synchronous? [Internet]. Stack Overflow. [cited 2022 Mar 22]. Available from: https://stackoverflow.com/questions/2035645/when-is-javascript-synchronous

32. How to Use Fetch with async/await [Internet]. Dmitri Pavlutin Blog. 2020 [cited 2022 Mar 22]. Available from: https://dmitripavlutin.com/javascript-fetch-async-await/

33. Sallai T. How to use async functions with Array.map in Javascript [Internet]. [cited 2022 Apr 5]. Available from: https://advancedweb.hu/how-to-use-async-functions-with-array-map-in-javascript/

34. @mapbox/polyline [Internet]. npm. [cited 2022 Mar 23]. Available from: https://www.npmjs.com/package/@mapbox/polyline

35. Home | TheAudioDB.com [Internet]. [cited 2022 Mar 11]. Available from: https://www.theaudiodb.com/

36. Formatting [Internet]. [cited 2022 Apr 12]. Available from: https://moment.github.io/luxon/#/formatting?id=table-of-tokens

37. GitHub Pages [Internet]. GitHub Pages. [cited 2022 Apr 4]. Available from: https://pages.github.com/

38. HTTPS. In: Wikipedia [Internet]. 2022 [cited 2022 Apr 16]. Available from: https://en.wikipedia.org/w/index.php?title=HTTPS&oldid=1082479259

39. cephalin. Security - Azure App Service [Internet]. [cited 2022 Apr 16]. Available from: https://docs.microsoft.com/en-us/azure/app-service/overview-security

40. Redux - A predictable state container for JavaScript apps. | Redux [Internet]. [cited 2022 Apr 25]. Available from: https://redux.js.org/

41. State Managers [Internet]. JavaScript Stuff. [cited 2022 Apr 25]. Available from: https://www.javascriptstuff.com/state-managers/

42. Postman API Platform [Internet]. [cited 2022 May 3]. Available from: https://www.postman.com/product/what-is-postman/

43. What is Unit Testing? Definition from WhatIs.com [Internet]. SearchSoftwareQuality. [cited 2022 Apr 6]. Available from: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing

44. Jest · 🃏 Delightful JavaScript Testing [Internet]. [cited 2022 Apr 6]. Available from: https://jestjs.io/

45. ASP.NET | Open-source web framework for .NET [Internet]. Microsoft. [cited 2022 Apr 12]. Available from: https://dotnet.microsoft.com/en-us/apps/aspnet