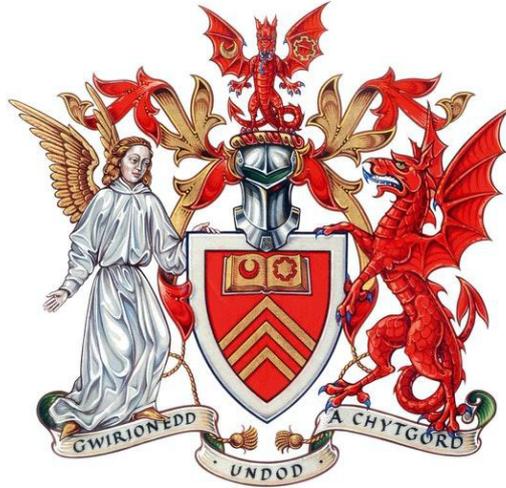


# Can Music Make You Run Faster?



Rhys Mathew Douglas

BSc Computer Science (With a Year in Industry)

CM3203 – Final Report

Supervisor: Dr Martin Chorley

Moderator: Surya Thottam Valappil

School of Computer Science and Informatics

Cardiff University

Module number: CM3203

Module title: One Semester Individual Project

Credits due for this module: 40

28 May 2021

## Abstract

No matter who you are in the world, or what sport you partake in, you are always looking to improve your performance to become the best version of yourself. In running, this often comes down to personal determination to persevere and run faster and further than you had before.

A lot of runners believe music is the key to continued success – and with a rise in smartphone apps to track activities and listen to music, it is now easier than ever to measure your performance over a large period and watch your abilities flourish. This project aimed to supplement that growth, by producing a web user interface that allows users to connect their running activities from some of the most popular apps, including Strava and Fitbit, and sync those activities with their music history from services such as Spotify and Last.FM. Inferences can then be made from said apps, and ultimately the project is able to produce a list of songs that truly make the runner exercise at their best.

This report follows the development process of this project, from understanding basic concepts of BDD-TDD to the finer details of determining what songs make a user run faster. Backed by rigorous testing, this report is then rounded off with an overall evaluation of how effective the project is against its original aims, ultimately answering the question of “Can Music Make You Run Faster?”.

## Acknowledgements

Firstly, the biggest thank you goes out to my family and friends; it has been a tough year for us all, and your support has helped me get this project completed successfully.

Many thanks to Dr M J Chorley for supervising this project, providing excellent support and feedback throughout the course of this project, and suggesting the idea for it. His enthusiasm and reinforcement in belief helped keep this project on schedule and enabled me to get the project fully completed.

## Table of Contents

---

1	Introduction .....	5
1.1	Motivation.....	5
1.2	Assumptions.....	5
1.3	Beneficiaries.....	5
1.4	Project Aims and Objectives .....	6
1.5	Scope of project .....	6
1.6	Approach.....	7
1.7	Summary of outcomes.....	7
2	Background .....	8
2.1	Wider problem context.....	8
2.2	Identified problem and stakeholders.....	9
2.3	Associated theory .....	9
2.4	Existing solutions.....	10
2.4.1	Closest existing solutions .....	10
2.4.2	Automatic Tempo-matchers .....	10
2.4.3	Manual tempo-matchers .....	11
2.4.4	Other running apps.....	11
2.4.5	Music providers.....	11
2.4.6	Conclusions .....	12
2.5	Methods and tools used .....	12
2.6	Constraints on solution.....	13
2.7	Research questions .....	14
3	Specification and Design .....	15
3.1	Specification and approach to problem.....	15
3.1.1	Core requirements and constraints .....	15
3.2	Business Requirements.....	15
3.3	Solution Architecture .....	17
3.3.1	High level architecture and flow of data.....	17
3.3.2	High level testing architecture .....	18
3.3.3	UML Activity Diagram .....	19
3.3.4	Example data flow using Strava. ....	20
3.3.5	Module Partitioning .....	20

## Can Music Make You Run Faster?

3.4	Web front-end user interface .....	21
3.5	What songs make me run faster? .....	22
4	Implementation .....	23
4.1	Agile development process .....	23
4.2	BDD and TDD .....	25
4.2.1	Fakes and mocks .....	25
4.3	User authentication .....	26
4.4	Unforeseen problems .....	26
4.4.1	Mobile frontend .....	26
4.4.2	Automated code style checking .....	27
4.4.3	Spotify limitations .....	28
4.4.4	Ethical approval .....	28
4.4.5	Fitbit limitations .....	29
5	Results and evaluation .....	30
5.1	UAT results .....	31
5.2	BDD and TDD automated testing .....	36
5.3	Evaluation from UAT, BDD and TDD .....	37
5.3.1	UAT .....	37
5.3.2	Limitations on UAT .....	39
5.3.3	BDD and TDD .....	39
5.3.4	General project evaluation .....	40
5.4	Future tests required .....	41
5.5	Summary and critical appraisal of approach .....	41
6	Future work .....	43
7	Conclusions .....	45
8	Reflection on learning .....	47
9	Table of Abbreviations .....	49
10	Appendices .....	50
11	References .....	58

# 1 INTRODUCTION

---

## 1.1 MOTIVATION

After running for a few times, runners may begin to question whether the music they listen to (or choose not to listen to) has any impact on their running. This “impact” could be in the form of anything; average cadence, heart rate and average speed to name a few potential impacts. This project therefore aimed to answer the question: Can Music Make You Run Faster?

This project is particularly interesting because it is a topic that has had little investigation; most people *assume* that music does have an impact on their running ability, but it is unclear to what extent the impact listening to music whilst running has. This opens an avenue to more questions that could be asked; do different genres of music affect runners in different ways? What is the effect of listening to music with a high or low BPM?

## 1.2 ASSUMPTIONS

This project is therefore based on the assumption that music does have some impact on a runner’s ability to perform during exercise. In 2014, it was shown from a study that when a group of college students listened to music whilst running “their running performance improved collectively” (Bonette et al. 2014). Other scientists have also stated that music has a positive impact on pace, where Jasmin Hutchinson, PhD stated that “Matching your stride to a particular beat can help you better regulate your pace” (Runner’s world, 2018). Without this assumption in place, the project would not have any grounding.

This project also assumes that runners are likened to utilise mobile apps whilst they run, primarily activity-tracking apps like Strava, and music streaming apps like Spotify. Different avenues exist to track activities and listen to music, but in a world more and more orientated around smartphones, this project assumes that runners exclusively log their runs and listen to music using their mobile phone.

## 1.3 BENEFICIARIES

As this project is orientated around the impacts of music on a runner’s performance, this project is intended to be used primarily by runners who wish to investigate the impact of music on their running performance. It is hoped that these runners can then benefit from the insights generated from this project to better tailor their music choices whilst they run to maximise their performance.

## 1.4 PROJECT AIMS AND OBJECTIVES

The aims of this project were to create a mobile application that allows a user to determine whether music made them run faster. As such, the following aims were set:

1. To create a mobile application for Android, integrating at least 1 running application and 1 music application.
2. To create the necessary logic required to enable a comparison between the user's running activities and their listening history, comparing metrics such as pace and average cadence to the music they were listening to.
3. To create a user-centred, clean user interface that is fit for purpose and encourages ease of use.

To supplement the aims above, secondary aims were also set for this project:

1. To create a secure login system, complimented with back-end database to store user information.
2. To extend the logic comparing music and running activity data, enabling the user to view various impacts and music on their activity.
3. To extend the functionality to suggest other songs that may be used to increase a user's running average pace.

To achieve all of the project aims, objectives were also created to ensure that the project was delivered on time:

1. Investigate the extent of data available from music and running sources' respective public APIs.
2. Create a mobile application using the Xamarin framework, that will enable the creation of both iOS and Android mobile applications.
3. Employ Test Driven Development and Behaviour Driven Development to ensure the creation of a well-tested, robust mobile application.
4. Create an intuitive and functional user interface.
  - a. Log in screens to enable connectivity between the Xamarin application and the user's running or music application.
  - b. Activity screens that enable a user to view their running history over a period, view what music they were listening to and the effect it had on their activity and health metrics.
  - c. 'Insights' screens that enable the user to view what music made them run faster.

## 1.5 SCOPE OF PROJECT

The scope of this project therefore was to create a mobile application, capable of producing "insights" that allowed a runner to determine the impacts of listening to music with their running history. The project began with the development of an Android application, but in later iterations it was also made to support iOS devices, should the progress of the project be ahead of schedule.

## 1.6 APPROACH

As the author has experience as a full stack software engineer throughout his year in industry placement, this project mimicked a SCRUM project; using an agile methodology, features were incrementally added to the solution, produced as close to end-to-end as possible. This gave the project versatility to adjust its scope where necessary, and to create sprints which could be adjusted perfectly to meet the author's capabilities as the software developer in this project.

As such, the project began as a versatile backend adaptable to support multiple user interfaces, including mobile applications and web applications. "Stories" were first refined into Gherkin (Gherkin,2019), before being implemented using BDD, with TDD to incrementally further the story to completion. To support BDD and TDD, fakes are prominently used, including the "faking" of external services, such as Spotify and Strava to ensure testing can take place as much as possible.

## 1.7 SUMMARY OF OUTCOMES

The primary outcome of this project is to create a mobile-usable application, which will enable runners to connect their running and music listening services and be able to view insights based on the services they connect with. In turn, this would hopefully be used by said runners to better tailor the music they listen to, to further their performance whilst running.

## 2 BACKGROUND

---

This chapter aims to provide a detailed explanation of background topics that are necessary to understand this report. This will begin by explaining the wider problem context, before identifying the problem and its stakeholders. From this, similar and existing solutions are explored, before discussing what elements of those solutions will be incorporated. Finally, the remainder of this chapter will explore what methods and tools are used in the solution, as well as the constraints that are in place.

### 2.1 WIDER PROBLEM CONTEXT

Running and activity tracking apps are widely used across all sports, with huge use from professionals and amateurs alike. These tracking apps often support the measuring of multiple metrics, including average pace, cadence, heart rate, aerobic and anaerobic effects, stride length and calories burned to name a few. The source of these metrics does not come from the activity tracking apps themselves – runners often utilise smart watches, heart rate bands and other wearable technology to improve the accuracy of the metrics gathered during their activities.

If the runner does not have access to this degree of wearable tech, mobile phones are increasingly improving their ability to measure health metrics, but this is often limited to steps taken during an activity, average pace and calories burned, of which can be sometimes inaccurate, and mostly relies upon other, unrelated current activity metrics such as height, weight, gender, and age (Apple, 2020). Despite this, mobile phones and smart watches are increasingly becoming more accurate, with some studies showing only minor disparities between calculated calories burned and actual calories burned.

These metrics proved useful for this application, as the activity tracking apps that hold this personal information can be accessed with public-facing web APIs. This means that, providing users grant authorization rights for the application to get their data, these health metrics can be used to help analyse the impacts of music on the user's running activities.

Music and music streaming apps are also widely used for multiple purposes, one of which is for listening to music whilst running. The days of Sony Walkman and iPods has ended, and now most of the music streaming now takes place on smartphones, with 32% of the music streaming market being occupied by Spotify (Mulligan, 2020). Similar to the previously mentioned activity tracking services, these music streaming providers also provide a publicly facing web API, allowing developers to view an individual user's listening history, including song metadata and the time the song was played.

Combined, these two different web APIs can be used by an application to allow mapping between a user's recorded exercises on Strava and Fitbit, as well as their listening history from providers such as Spotify and Last.FM. Consequently, inferences can be made by using the metrics of the recorded exercises (such as average pace) which can lead to further inferences, such as determining what songs made the user run faster.

## 2.2 IDENTIFIED PROBLEM AND STAKEHOLDERS

Based on the wider problem context, the identified problem is to create an application that allows a user to connect to their running and music phone services (Spotify, Strava etc.), and identify what music will make them run faster.

As this problem encompasses a multitude of external providers and technologies, there are several key stakeholders that have been identified:

- Runners / users, who will use the application to infer what music made them run faster.
- Public API providers, who provide the music and running data recorded by the runners using the application.

Runners are a key stakeholder, as they are the core users of this application. Not only will they benefit from viewing the insights generated by the application, but it will also be their own personal health data and private music listening history being used by the application to determine what songs made them run faster. As such, runners / users have a nested interest to ensure that their data is used responsibly; it is only used for its purpose, and it is not stored for longer than it is required. In accordance with this, the solution to this problem must comply with the user's right to privacy and will only retain personal data for as long as its required.

Public API providers or "data sources" also have a stake in this problem as the data to solve said problem will come from these public API providers. As data controllers, they have a responsibility to ensure that the data they collect is stored and used correctly – extending to how their public APIs are used by applications. This means that the solution to this problem must comply with the data sources' individual requirements for using their public APIs, where some requirements could involve using a specific design of login button or limiting the lifetime of transferred data when it has been requested from said public API.

## 2.3 ASSOCIATED THEORY

In order to establish what songs made a user run faster, the logic for determining said songs should be understood. At this point in the flow of the program, songs have already been mapped to each activity, regardless of what service it originated from (e.g., it is possible to have a Strava activity with Last.FM music and Spotify tracks linked to it).

The application can then infer what songs made the user run faster by comparing activities to one another and the metrics within said activities, such as pace and cadence. Using these metrics, the songs that can be determined to make a user run faster can be established by getting the activity with the fastest pace, or an activity with a cadence that best matches the ideal value of 170 or above (Lobby, 2013). The activity with the "best metrics" can then be used as the activity most ideal for improving, and as such the songs / tracks that are mapped to said activity can be returned as the songs that made the user run faster.

This could then be extended further, depending on what services are used. For example, Strava offers the ability to view "segments" (Strava, 2020). Segments could prove useful for

this application, as in-detail inferences on what songs made a user run faster could be broken down to specific parts of a user's run. In turn, this provides a more useful application for users as the accuracy of determining what songs made a user run faster can be improved.

## 2.4 EXISTING SOLUTIONS

From research conducted, there is no current existing solution that perfectly matches the problem space, of providing analysis of what songs made a user run faster after they have finished their running activity.

### 2.4.1 Closest existing solutions

Currently, there is only one existing solution that almost solves the problem stated. PUMATRAC is a standalone running application that provides detailed insights based on a user's running activity, including the "grading" of runs to describe its quality. PUMATRAC also generates personalized insights based on a user's running behaviour, noting the specifically the genre that makes the user run faster (Corpuz, 2021).

Other insights are also produced, such as the effects of weather and the time of day (Johnson,2013), proving that this app provides excellent functionality and almost solves the problem identified. This app could partially solve the problem if the user listened to a diverse range of music, because the user could identify easily what songs belong to said genre they were listening to. However, it cannot be assumed that every user will listen to a multitude of genres, and consequently would make it difficult for a user to determine what songs made them run faster.

### 2.4.2 Automatic Tempo-matchers

The closest examples of existing solutions are tempo-matchers, that offer automatic changing of music based on your cadence and pace as you run. Examples of this type of solution has been provided below.

For a time, Spotify offered a service known as Spotify Running. This feature was designed as part of the Spotify application, and using your phone's accelerometer and gyroscope, Spotify automatically adjusted the songs you were listening to, in order to match your tempo (Zamorski, 2018). The feature also allowed users to change the tempo manually using the main screen. Support for this feature ended on 09/02/2018 (Spotify,2018), but many standalone versions of the same concept also exist, including DrawRun, Tempo Run Nike+ Run (Zamorski, 2018), and RockMyRun.

Runkeeper also previously allowed a user to sync their music to the app, and then assess the songs for music and tempo. Ergo, you could then select the genre and tempo of the music you want to listen to whilst running (Independence Blue Cross, 2017). Runkeeper and Spotify also produced functionality for the Runkeeper app (Runkeeper,2016), whereby the Spotify Running features and Runkeeper features were able to work seamlessly together, but this was discontinued with removal of Spotify Running.

Other competitors in this space, such as the previously mentioned Nike+ Run App, offers a function called Power Song, allowing users to give themselves an extra "boost" by playing a

specific set of songs when the user wants to, in order to boost their performance (Alger, 2021).

These tempo-matchers partly solve the problem identified, because they will always push the user to run at their best by playing music to match their current tempo. The user could then arguably remember what songs were playing when they were running at their peak and use that as their list of songs that make them run faster. However, these tempo-matchers lack the scope to display what songs made them run faster, or in the case of Nike+, it assumes that the user already knows that songs will make them run faster before they exercise.

### 2.4.3 Manual tempo-matchers

To supplement automatic tempo-matchers, there are also different versions of the aforementioned applications, such as TrailMixPro. Instead of playing a pre-set platform generated playlist, TrailMixPro allows the user to choose some music from iTunes, and then adjusts the songs playing using the user's tempo and the playlist they added previously. The application also supports a "magic shuffle" feature that uses music in the user's library to match the user's current tempo (Pickett, 2016).

Manual tempo-matchers do not solve the problem specified, but they benefit from the choice of using the user's own music to match tempo. The user could then infer that the music playing when they were running fastest is music that makes them run faster, ergo the user can see what songs made them run faster.

### 2.4.4 Other running apps

There are also some running apps that provide audio coaching, or pre-set playlists. Weight Loss Running (Corpuz, 2021) provides audio coaching and pre-set playlists for users to listen to as they run. There is no correlation between the user's tempo, or what the user prefers, but as pre-set mixes they should have some basis to prove that they have a positive effect on a user's running activity.

There are many other apps that also provide the same functionality, such as Couch-To-5K and RunCoach to name a few, but they do not differ much if at all to the functionality provided by Weight Loss Running.

### 2.4.5 Music providers

Whilst these music providers may not directly solve the problem, music providers are key for the functionality of the planned application. One such music provider is Spotify, who in 2020 had a market share of 32% of the global streaming market (Mulligan, 2020). Spotify provides music streaming on desktop and mobile, including iOS, Android, Mac, and Windows. As previously mentioned in the "Automatic tempo-matchers" section, Spotify did also offer tempo-matching for a while, but that service has now been retired, prompting other market forces to employ the functionality. Spotify also features a public API allowing applications to query user's recently played tracks, but this only offers the 50 most recently played songs.

Another music provider is Last.FM, providing a service like Spotify, but acting instead as a middleman; Last.FM streams music from YouTube rather than streaming music from itself.

Accordingly, Last.FM offers insights based on the user's listening history, and actively builds suggested music for the user to listen to, based on their already recorded listening history (Last.FM,2021). As a middleman, Last.FM also offers the ability to connect other data sources, including Spotify and subsequently can store a permanent record of the user's listening history, compared to Spotify's limited 50 songs maximum. Last.FM also offers a public API, allowing applications to access this record of information.

### 2.4.6 Conclusions

Based on the findings of the above existing solutions, this project aimed to solve the primary deficiency of the existing solutions by showing the user what songs made them run faster retrospectively, instead of relying on the user to remember what songs were playing when they were running at their fastest. This project also aimed to solve the primary deficiency of PUMATRAC with this problem, by providing information on what songs made a user run faster instead of genres.

As the findings from the existing solutions have shown that the vast majority record the running activity themselves, this project differed by not supporting this functionality. This is because the reduction in scope in this area allowed the author to focus on supporting multiple data sources connecting to the application, meaning that the user does not need to re-record their running activity in the application if they wanted to find out what music made them run faster.

This decision has therefore shaped the application to focus on the inferences that could be made by connecting some of the existing solutions to the application, rather than collecting the information required and then analysing what music made the user run faster. This decision has also meant that the potential uses for tempo-matching music whilst running cannot be employed. However, the idea of creating a playlist that would use the average BPM of the user's fastest song and populating with similar songs has been considered and has such been moved to the Future Work section.

## 2.5 METHODS AND TOOLS USED

In order to provide an application that could work on Android, iOS and a web frontend, a versatile and resilient backend was required. To ensure that this backend also was compatible with BDD, TDD and had a long service life, the backend was developed using .NET Core 3.1, released in 2020. By using .NET Core 3.1, a robust backend could be created that supports all features necessary for this project.

.NET Core 3.1 uses C# as a programming language, and in conjunction with the need for a web frontend and a fake backend for BDD and TDD, ASP.NET Core is also used. ASP.NET core allows for the creation of simple servers (using Kestrel) that can hosted on any device with the .NET Core 3.1 SDK installed and allows for the use of several third-party clients to connect to the running service and music service providers.

The third party "clients" used are as follows:

- Fitbit.NET (Coleman et al. 2021)

- SpotifyAPI.Web (Dellinger, et al. 2021)
- Inflation.Lastfm (Inflation Friends, et al. 2020)

These third-party clients use an MIT license. They provide simple methods to get data from their relevant APIs, once authenticated using OAuth 2. The author could have used a third-party client for Strava as well, but due to necessity it was avoided, and a simple API call was used instead using built-in C# methods. The same could have been repeated for all the third-party clients, but for interest of time, these clients were used for their ease of use.

The SpotifyAPI.Web developers also have developed a package known as SpotifyAPI.Web.Auth, which contains a simple authentication web server to handle call backs for OAuth 2 authentication. This was adopted for all external services used, as the server was fit for purpose to meet the author's needs.

Kestrel is a cross-platform web server for ASP.NET Core (Dykstra et al. 2020). As aforementioned, Kestrel provides useful functionality for hosting servers, both for hosting a web frontend, or hosting an API to be used for testing. In combination with NUnit, a unit-testing framework for all .Net languages (NUnit, 2019), Kestrel can be used to host a localhost API containing fake data – meaning that unit, integration, and behaviour testing can take place regardless of the state of the external APIs, and independent of an internet connection. This also means testing can take place much faster than if compared to using the “real” backend and ensures that the application remains within the usage limits of the external APIs being called.

To enable BDD, SpecFlow has been adopted which enables written-as-English business requirements to be tested against (SpecFlow,2020). The syntax used by SpecFlow is Gherkin, providing a simple, testable method of writing business requirements.

The now deprecated frontend for mobile devices was produced using the Xamarin framework, a C# middleware that connects iOS and Android SDKs to it, enabling C# applications to run on these devices.

Instead of a mobile frontend, a web frontend was chosen to be implemented (as further explained in Section 4 Implementation). This was implemented using React Typescript, which compliments the object-orientated nature of the backend and supports TDD through Jest and Enzyme.

## 2.6 CONSTRAINTS ON SOLUTION

As there are multiple pathways for this project to take, constraints were required to be put in place to ensure that a high quality and feasible proof of concept were produced. Consequently, the primary constraint on the solution is on the depth of analysis performed by the app. The app will only compare activities from different data sources against one another and will not support analysis on part of activities (e.g., Strava Segments).

Additionally, as this project serves as a proof of concept and the fact that each data source provides different levels of information to analyse, analysis on the activities will also be constrained to average speed and pace. This choice ensures that the project's aims are met,

but also ensures that the developer of the project is not overextended, which would reduce the quality of the solution.

## 2.7 RESEARCH QUESTIONS

The following research question has been deduced:

*“In order to demonstrate the achievement of the stated aim of the project, this solution will allow a user to connect their recorded running history and music history, use an algorithm to determine their fastest activity and therefore deduce what songs made the user run faster in a versatile and user-friendly manner.”*

## 3 SPECIFICATION AND DESIGN

---

This chapter aims to explore the specification and design behind this project, as well providing justification for any major design choices that have been made. This begins with outlining major objectives and constraints, before exploring the details of the curated specification through the remainder of the chapter. This is done by first breaking the problem down into business requirements, before exploring the design of the system from a top-level overview.

### 3.1 SPECIFICATION AND APPROACH TO PROBLEM

#### 3.1.1 Core requirements and constraints

Based on the aims, objectives, and wider context from the background section, the system was required to:

- Allow a user to connect at least 1 running and at least 1 music data source.
- Allow a comparison between the connected data sources to take place.
- Allow a user to see what songs made them run faster, retrospectively to when they completed the activity.

This specification closely encapsulates the aims of the project as defined in the Introduction section, but also is simple enough to be interpreted in multiple ways, giving the project flexibility as it progressed. These requirements are then further explored in the Business requirements section as to how they are translatable into software.

Reflecting on the findings in the background section, the following constraints have been put in place:

- The project will only initially compare simple metrics to determine what songs made a user run faster, such as average or average speed.
- The depth of analysis on user's exercise will also be constrained to an activity-level, meaning features like Strava Segments will not be utilised.

Constraints such as these allowed the author to focus on the core aspects of the system, whilst maintaining the correct level of complexity and scalability required for a project such as this, that could encompass a variety of features in future development.

### 3.2 BUSINESS REQUIREMENTS

As a system that is reliant on the user's interaction and personal data, it is necessary that business requirements were written to match the user's needs. As such, Gherkin (Gherkin,2019) was adopted to write business requirements, as writing requirements in Gherkin follows the process a user follows to achieve an aim by using "Given", "When" and "Then" steps. By using Gherkin, BDD could also be adopted, meaning that testing against these business requirements could be tested against automatically, and as often as development required. A typical Gherkin business requirement looked like this:

## Can Music Make You Run Faster?

```
1 Feature: Spotify API Calls
2
3 Background:
4 Given a list of users
5 | user      |
6 | User1     |
7
8 Given a list of Spotify listening history
9 | user | Song name                | Time of listening |
10 | User1 | The Chain - 2004 Remaster | 15/02/2021 15:45:30 |
11 | User1 | I Want To Break Free - Single Remix | 15/02/2021 15:40:01 |
12 | User1 | Good Vibrations - Remastered | 15/02/2021 15:30:59 |
13 | User1 | Dreams - 2004 Remaster | 15/02/2021 00:05:00 |
14 | User1 | Stayin Alive | 14/02/2021 23:59:59 |
15
16
17 @MVP-0-Add-Spotify-API-Call
18 Scenario Outline: Get User's Spotify listening history
19     Given a user <user>
20     And their Spotify listening history
21     When the user's Spotify recently played history is requested
22     Then the user's Spotify recently played history is produced
```

Appendix A – Typical Gherkin business requirement.

By using Gherkin, the following business requirements were produced to ensure that the system truly reflected what the user required:

1. Get a user's Spotify listening history.
2. Get a user's Strava running history.
3. Get a user's Last.FM listening history.
4. Get a user's Fitbit running history.
5. Compare Spotify and Strava history on a singular date.
6. Compare Spotify and Strava History with a date range.
7. Compare Listening and Running history on a singular date using Last.FM and Fitbit.
8. Compare Listening and Running history on a singular date using all data sources.
9. Compare listening and running history using all data sources with a date range.

The details of these business requirements are viewable in Appendices A-E.

Each requirement was refined based on the typical process the user would follow when using this application, as well as speaking to likely users on what they would define their requirements to be. These were then sometimes checked by the supervisor to ensure that they truly kept within the scope of the system. By following this process, the top-level requirements always reflected what the user wanted, and what the system should do, as such always meant that the development process of completing each user requirement kept within the scope of the system.

### 3.3 SOLUTION ARCHITECTURE

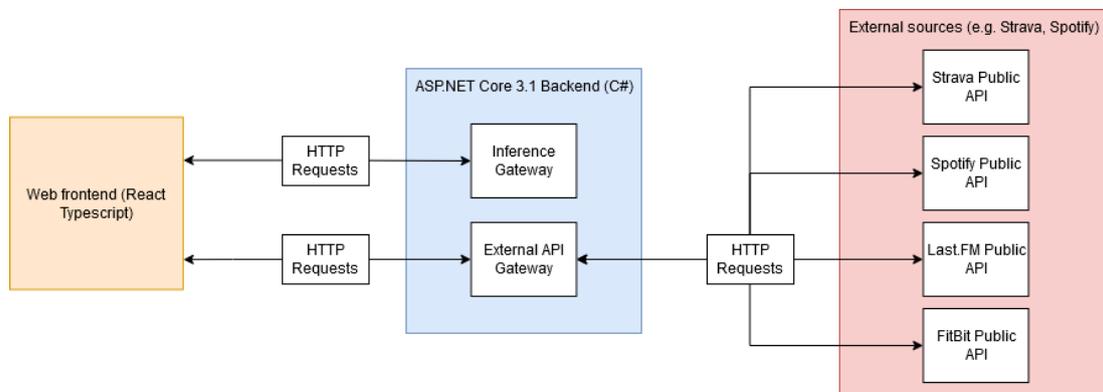
#### 3.3.1 High level architecture and flow of data

The highest level of the system architecture can be defined as shown below in Appendix A. The user interacts with the system using the web frontend, built using React Typescript. After interacting with the React TS frontend, HTTP GET / POSTS requests are sent to an ASP.NET Core Web API with two public API gateways: Inference Gateway and External API Gateway.

The Inference Gateway simply performs “inferences”, including the logic to determine what music made a user run faster. The External API Gateway is used as a middleman between the web application and the external sources, whose sole purpose is to authenticate the user and get data from said external sources. This communication takes place also using HTTP GET requests.

This design has not changed much throughout development, with the “Inference Gateway” being the only major modification. This is because the design of the system demanded that the backend had versatility, meaning that the frontend should possess no logic to provide scalability to other frontends (such as a mobile or a Smart Watch frontend).

Data then flows from the external sources to the External API Gateway, when of which data is processed to ensure it meets the required criteria requested from the Web frontend, and then returned as expected. When the user queries data from the Inference Gateway, requests are received, processed and then the correct response is returned, like the external API Gateway. The web frontend then handles this data and displays it in a user-friendly manner.



Appendix F – Highest level system overview

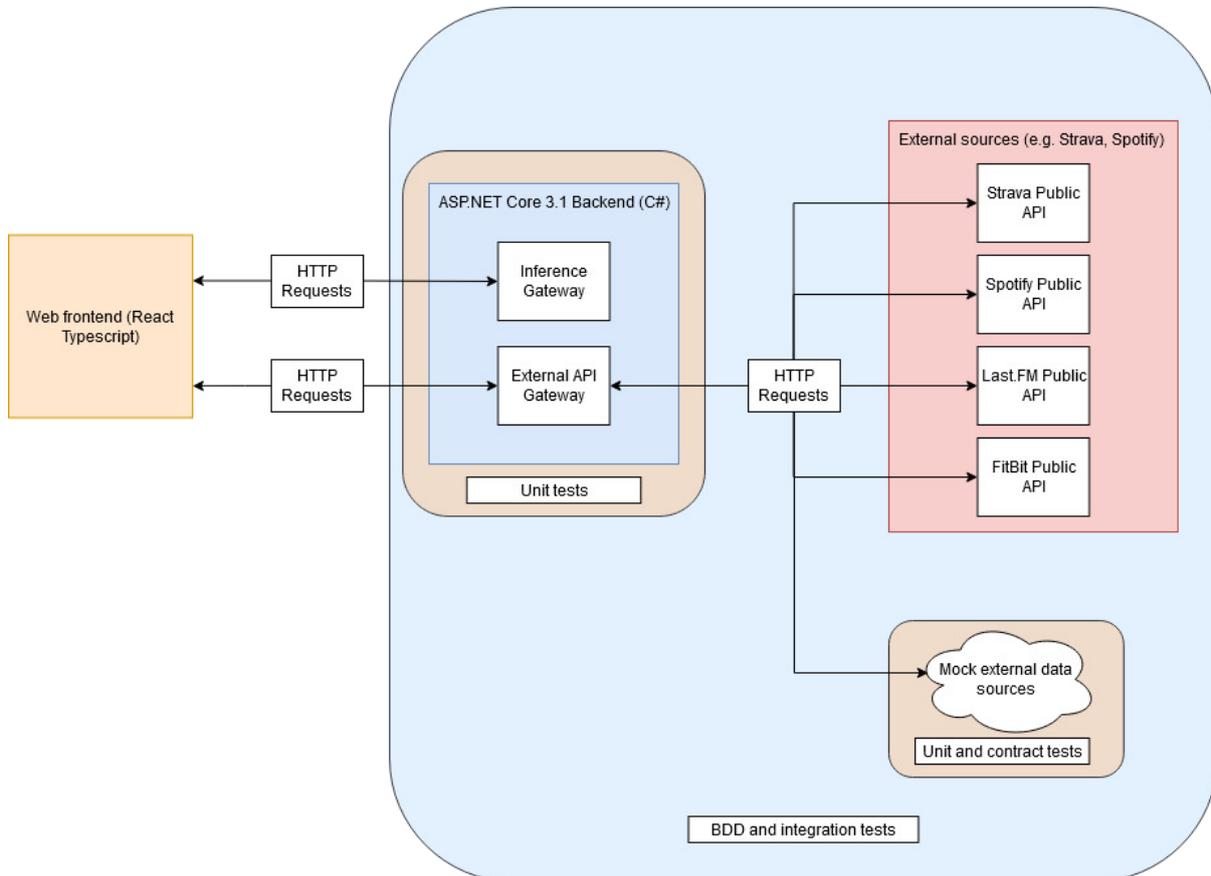
Within the ASP.NET Core 3.1 Backend, data is received from the external sources predominantly in a JSON or XML format. This ‘raw’ data is then serialized into strongly typed C# objects that correspond to each parameter within each JSON object. As such, these objects are referred to as “DTO” or Data Transfer Objects. They are then again converted in “entity” objects before any comparison takes place to ensure that the JSON/ XML is serialized correctly, and error checks are put in place in case parsed JSON/ XML data is fragmented or missing data.

This design ensures validation takes place across all levels of data handling, and the benefits of an object orientated approach can be adopted, such as polymorphism and encapsulation.

These design choices also ensure that all the core objectives have and can be achieved, with scalability offered to a potential future increase in business requirements or core objectives.

### 3.3.2 High level testing architecture

To prove confidence in the system, BDD, TDD and integration tests have been included within the system to prove robust and versatile functionality. This is illustrated in the following diagram:



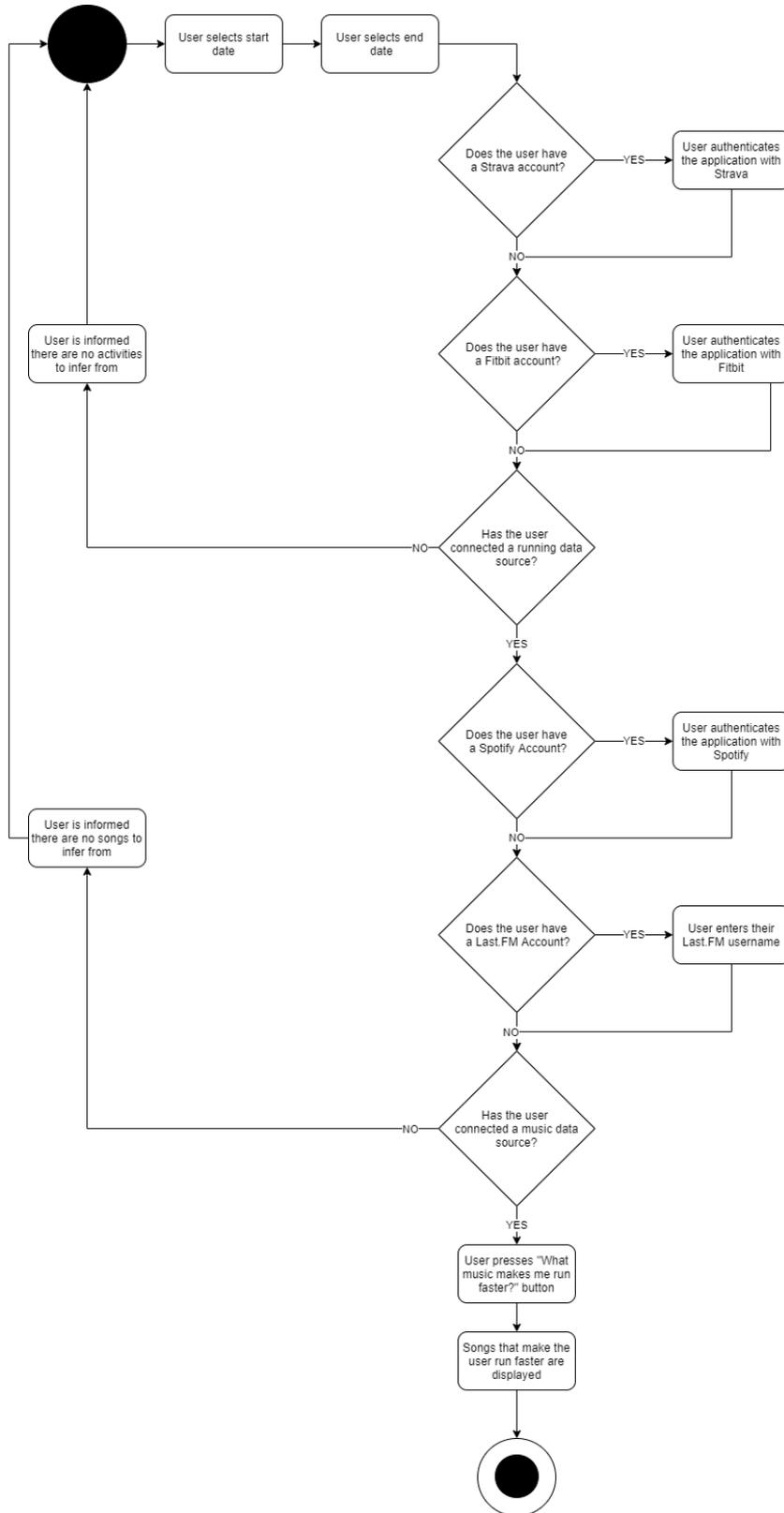
Appendix G – Highest level testing architecture.

By adopting this testing architecture, full confidence could be given in the system as the project grew, meaning that each iteration of development always contributed to the overall growth of the project. In other words, every addition to the project had some business value.

As alluded to in the background section, quality unit, integration and behaviour tests allow the user to gain confidence in the system when connecting their exercise activity as these tests prevent errors from occurring, and prove the solution is sturdy enough to handle their personal data. Therefore, by adopting this testing architecture confidence can be given in the system by both the author and the user base as the project grew by each iteration.

### 3.3.3 UML Activity Diagram

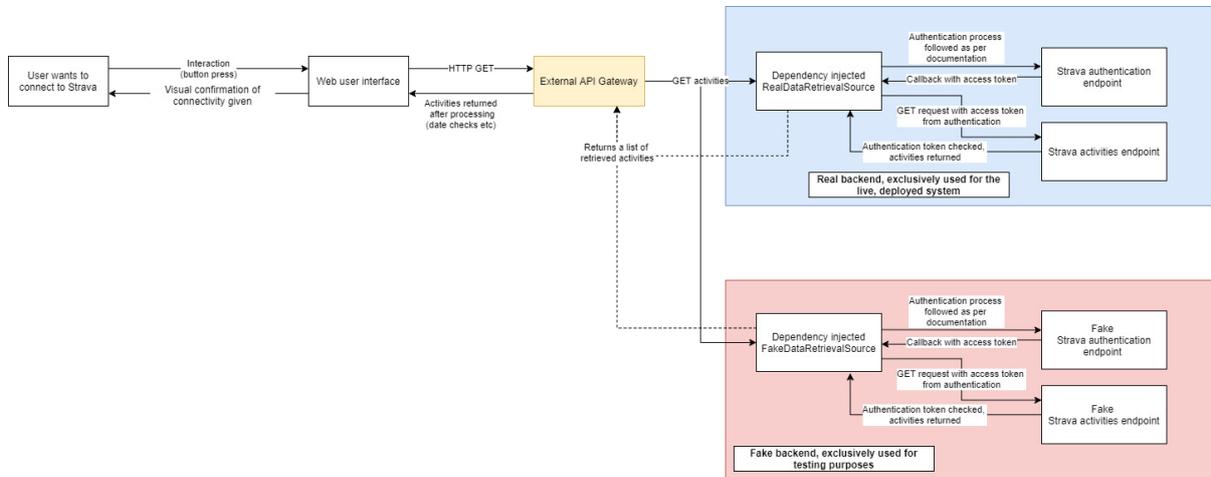
Below is an example diagram of how the user may choose to interact with the system. This diagram breaks down the process a user would follow when they wish to determine what songs make them run faster.



Appendix L – UML Activity Diagram

### 3.3.4 Example data flow using Strava.

The diagram below illustrates how data flows from one end of the system to the other, without explaining technical details too importantly. In this diagram, the process of getting Strava activities has been explored:



Appendix J – Example Strava data flow.

This diagram also mentions an authentication process. More information about this can be found with the reference of (Strava,2021). OAuth2 authentication is supported by all external data sources chosen, with Strava and Fitbit using Implicit Grant Flow, and Spotify using PKCE. In an ideal world, PKCE would have been used for all external sources as this negates the need to keep client credentials in the source code. PKCE has been adopted by the only data source that supports it, Spotify. Last.FM in its current scope does not require authentication, and as such no OAuth2 authentication process has been implemented.

This mixture of implementations of OAuth2 Authentication also proves flexibility and scalability in the system, meaning that it will be much easier to add more data sources in future if required.

### 3.3.5 Module Partitioning

As this project was developed as a .NET Core 3.1 application using Visual Studio, module partitioning had to take place to ensure that a scalable, easy to understand and maintain system was created. As such, the following partitions (projects) have been created:

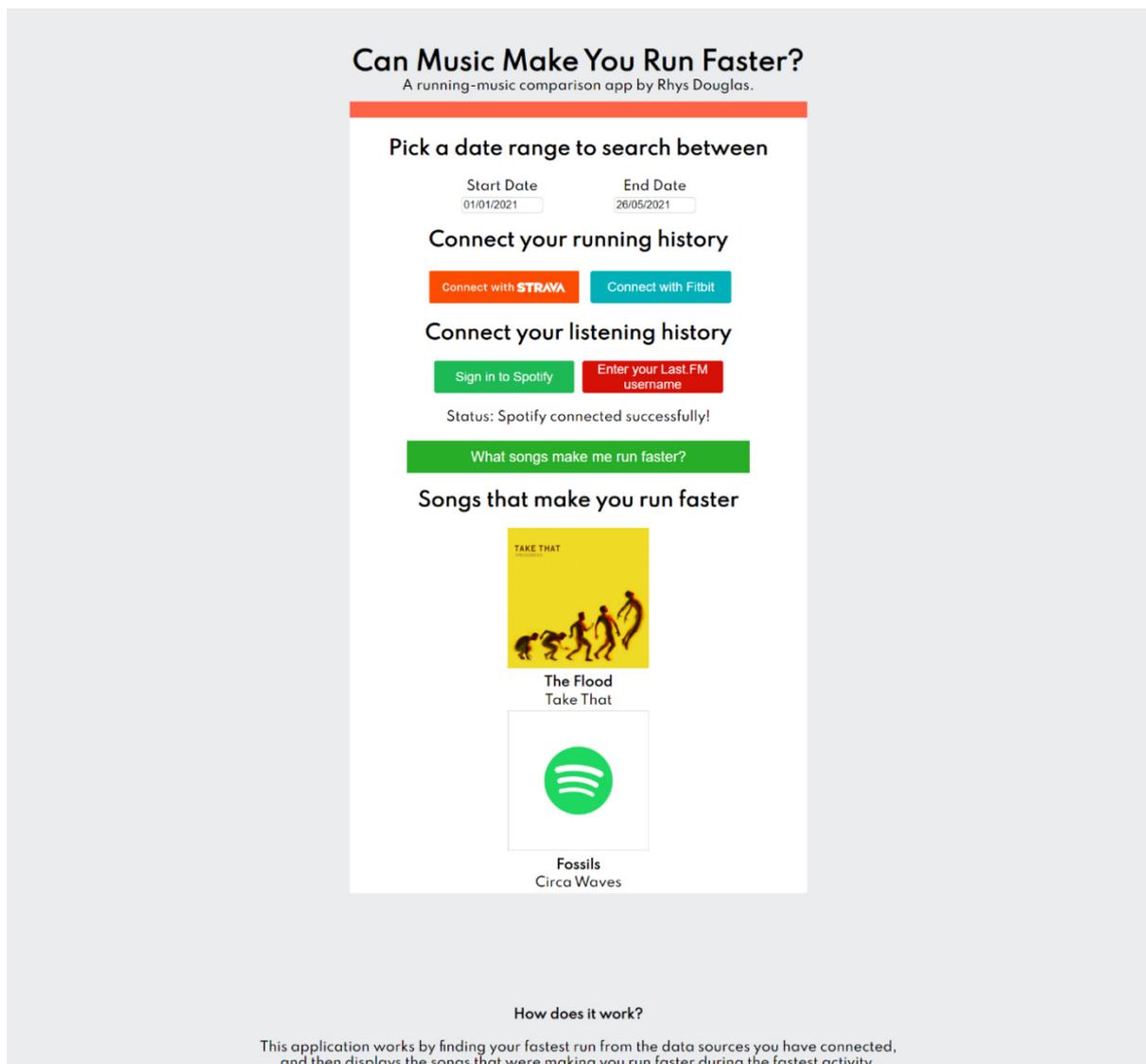
- **RD.CanMusicMakeYouRunFaster.CommonTestUtils**
  - Contains testing utilities, such as an InMemoryFactory used for integration and behaviour driven tests.
- **RD.CanMusicMakeYouRunFaster.ComparisonLogic**
  - Contains logic to compare activities and make activity inferences.
- **RD.CanMusicMakeYouRunFaster.ComparisonLogic.UnitTests**
  - TDD (Unit tests) for the ComparisonLogic project.
- **RD.CanMusicMakeYouRunFaster.FakeResponseServer**
  - Contains “fakes” used for integration and BDD testing.
- **RD.CanMusicMakeYouRunFaster.FakeResponseServer.UnitTests**
  - Contains TDD (Unit and Contract) tests to ensure the FakeResponseServer acts properly.
- **RD.CanMusicMakeYouRunFaster.Rest**

## Can Music Make You Run Faster?

- Main entry point of the application. Contains a RESTful API allowing frontends to communicate with the remainder of the solution.
- RD.CanMusicMakeYouRunFaster.Rest.IntegrationTests
  - Integration tests for the REST API, using the FakeResponseServer.
- RD.CanMusicMakeYouRunFaster.Rest.UnitTests
  - TDD (Unit) Tests for the REST API.
- RD.CanMusicMakeYouRunFaster.Specs
  - BDD tests for the entire solution.
- RD.CMMYRF.WebPortal
  - Contains the web frontend, created using React Typescript.

### 3.4 WEB FRONT-END USER INTERFACE

To achieve one of the project’s primary aims of a creating a “user-centred, clean user interface that is fit for purpose and encourages ease of use”. The following design has been adopted to reflect this:



Appendix H – Example web user Interface

The above user interface has been designed in such a way that it adheres to Nielsen's 10 usability Heuristics (Nielsen, 1994). These usability heuristics ensure that the third project primary aim is achieved to the highest degree possible.

### 3.5 WHAT SONGS MAKE ME RUN FASTER?

Building on the foundations of the background's associated theory section, the process of deciding what songs make a user run faster must be understood.

At first, the system began by retrieving all the exercises required within the date range specified before mapping songs to each activity. These activities were then compared using a "king of the hill" style algorithm, where the activity with the fastest average speed was selected, and then the songs paired with said activity were returned. However, this design brought limitations when development began, namely in those frontends were unable to support the method of analysis as mentioned above.

Instead, a new design had been adopted. Firstly, activities are requested from the relevant data source before they are "posted" (using HTTP POST) to the InferenceAPIGateway which is used to determine the fastest activity. This is determined using the exact same logic as previously stated, of which the fastest activity is then returned. This is then repeated for every data source connected, until there is a fastest activity for each data source.

When the user is ready, the property from each activity is then sent back to the InferenceAPIGateway, where the fastest overall activity is determined. After such activity is determined, the date and time of that activity is returned. This is then sent to the ExternalAPIGateway, which returns all the songs that were playing whilst the activity took place. These songs are then displayed to the user, proving that those songs were the songs that have the most effect on them.

This design may sound complicated at first, but it affords a lot of scalabilities for other metrics to be added in future, meaning only a small number of additional endpoints need to be created in the InferenceAPIGateway to support deeper analysis. This design also keeps the concept of having all the business logic in the backend, with the frontend being used only as a method of communicating with the user. This again provides versatility, adaptability, and scalability for multiple frontends to be added in future, perfect for a project that is designed to be used by as many users as possible.

## 4 IMPLEMENTATION

---

This chapter aims to cover the development cycle as the project grew, including any major alterations to the project that occurred as well as exploring key aspects of the system that made it what it is today. This chapter begins by exploring the process followed during implementation, before exploring key concepts and then covering any major unforeseen projects that altered the project.

### 4.1 AGILE DEVELOPMENT PROCESS

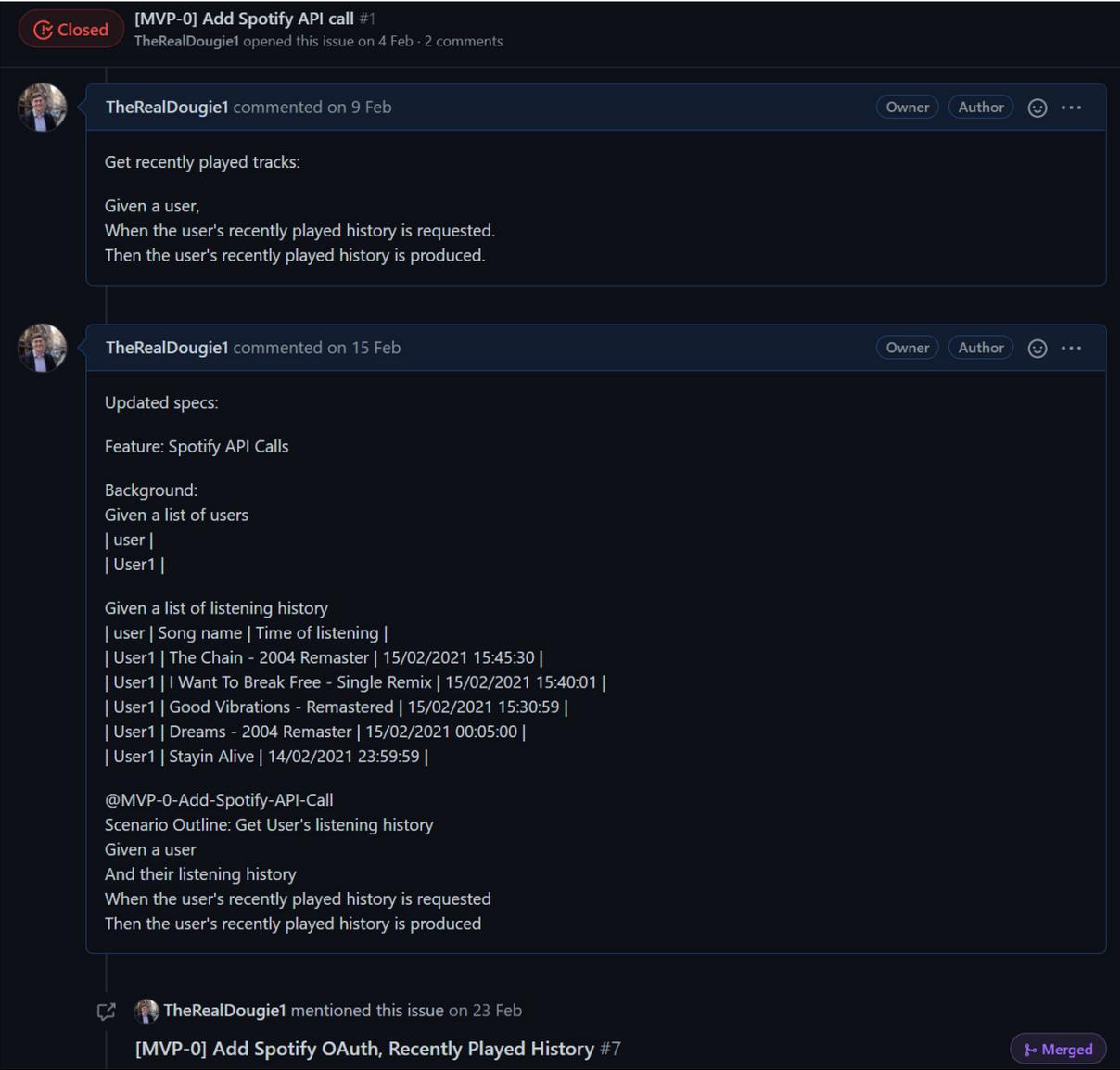
This project was developed using an AGILE approach, with BDD and TDD at the core of it. Due to time constraints of the project, 1-week sprints were adopted instead of traditional 2-week sprints. At the beginning of each sprint, typically a pre-refined story was focussed on and at the end of each sprint the work for each week was reflected upon. At that stage, problems faced that week were actively identified and plans were made to avoid the same issues arising again.

To reflect the AGILE process, a Kanban board was created on Github that allowed the author and other interested parties (such as the supervisor) to track the progress of the project as it grew. This can be seen in Appendix I, with a link to the board found at (Douglas, 2021). This contributed to development as a central knowledge repository as well as a code repository, meaning that design decisions and issue tracking was maintained through this system.

Each “story” contained at least 1 business requirement, of which was refined as aforementioned at the beginning of each sprint, with 1 story being refined in advance as a minimum. In other words, each business requirement as shown in appendices A-E was required at the beginning of a sprint throughout the development process, and not refined at the beginning of the entire project (as would be expected in a waterfall project).

This gradual development of business requirements granted a lot of flexibility throughout the development cycle and meant that the author was not constrained to business requirements that may have become outdated as the project grew. An example of this can be shown for the “Add Spotify API Call” issue, where the business requirements had to be changed as development of the project continued. This is shown in a screenshot below:

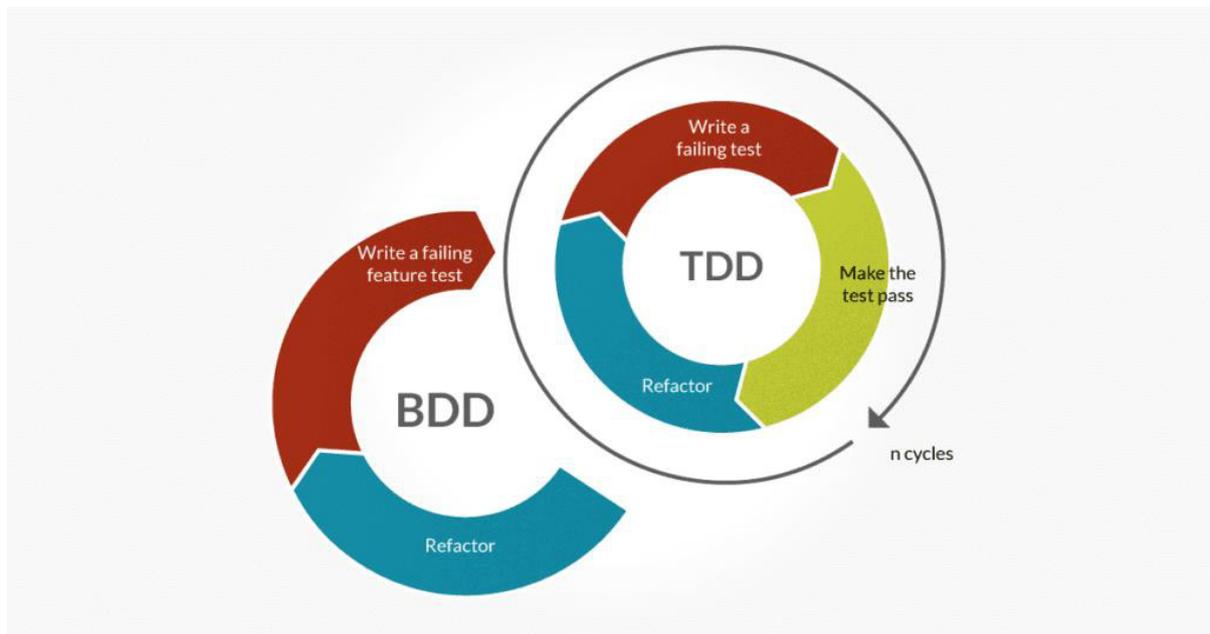
# Can Music Make You Run Faster?



Screenshot reflecting how business requirements could change throughout the development process.

## 4.2 BDD AND TDD

At the heart of this project, BDD and TDD were the two core principles that drove the project forward. By adopting the process of writing a test, writing code to pass the test and refactoring code after the passing test on all levels, confidence in the system grows as each test is added. When using BDD and TDD together, a “cycle” is adopted, as shown in the figure below:



An infographic regarding the process of BDD-TDD. (Nair, 2018).

BDD-TDD has been adopted by beginning with the business requirements as defined in Gherkin. These tests are then run in different “modes” depending on the system, starting with an ‘API’ mode. The ‘API’ mode purely tests the flow of data end-to-end, without any frontend involved. However, other modes can be added including a “Web” mode that involves the use of the web frontend to test the system end-to-end. This would have been included in the project, but the author ran out of time to implement the ‘Web’ mode.

After the framework for each mode has been created, TDD tests are created to incrementally pass the BDD test, until which a time all tests pass, and the code is refactored to work better, and the cycle starts again. This ensures complete code coverage as the project progresses so that project aims are always progressing as each individual test, no matter the level, contributes to the overall confidence in the system.

### 4.2.1 Fakes and mocks

To supplement BDD and TDD, mirror-image components have been created to allow integration and top-level BDD tests to take place, and the range of complexity these fake components have vary on the required scenario. For example, a mock HTTP client could be used to integration test the controllers within the External API Gateway to prove functionality, whereas a BDD test would use an entire fake-backend. This fake backend exists as the FakeResponseServer, of which is used to mimic the behaviour of a specific public API, such as Spotify. As the author has created this fake backend, this FakeResponseServer can also be

used to serve whatever response is needed for the test – allowing for a range of tests using normal, abnormal, and extreme data; stress testing the real system that has been designed.

Coupled with BDD and TDD, this creates a development environment that pioneers the system to be as robust as possible – you cannot progress onto the next task until the current test is passing. As such, each test proves that the system works as it should, putting more confidence into the solution, and less stress on quality assurance.

It is also worth mentioning that the FakeResponseServer is also unit and contract tested – meaning that it will always work as expected and will not differ from the expected behaviour from its counterpart.

### 4.3 USER AUTHENTICATION

As mentioned in the specification and design section, multiple methods of user authentication have been integrated into the system, namely Implicit Grant Flow and PKCE. To support this, an authentication server taken from the SpotifyAPI.Web.Auth package (see background section for more information) is used to handle call backs from each data source. This provided a great time-saver for the author, as alone he would have had to create a server himself to handle call backs and serve HTML content. Unfortunately, this authentication server was only designed to be used for Spotify, and as such returns a “connected to Spotify!” page no matter what data source is used; creating the potential for confusion.

However, the provided solution has provided a strong basis for authenticating users. Consequently, a strong basis offers ease of scalability when integrating more data sources, and as a result the project overall is better off for it.

### 4.4 UNFORESEEN PROBLEMS

In this subsection, major unforeseen problems are detailed. This includes an explanation of what happened, how the problem was overcome and why the problem had such an impact on the progress of the project.

#### 4.4.1 Mobile frontend

Perhaps the largest challenge this project faced was the implementation of a Mobile Frontend. Originally, plans were put in place to develop an Android and iOS app using the Xamarin C# Framework, soon to be replaced by .NET MAUI. Xamarin was a perfect choice for this project, as it seemingly offered itself as a middleman between Android and iOS SDK’s and a C# backend. However, problems began to arise when BDD and TDD was first attempted to be integrated with Xamarin development. Due to incompatibility issues, the BDD and TDD projects were unable to “reference” the Xamarin project and as such the author decided it would be fine if there was no BDD or TDD tests covering the project.

However, more problems began to arise as development started. The author had gotten as far as producing several functional UI screens, and even as far as getting Spotify songs by allowing a user to authenticate using the incomplete application. However, when these songs

were attempted to be sent to the already created .NET Core 3.1 backend the same incompatibility issue arose.

After some research, the author discovered that all the development work prior to the Xamarin implementation was useless as it had been developed using .NET Core 3.1. To give context, .NET Core 3.1 is the latest release from Microsoft, and is meant to support all the features from sister frameworks such as .NET Standard and .NET framework. However, this was not the case, and any .NET Core 3.1 content was unable to be used with the Xamarin project.

The author was then left with a choice; attempt to refactor the previous development cycles to be compatible with the Xamarin frontend at the cost of all BDD and TDD tests or create a web frontend that was compatible on Smartphones and lose all the Xamarin progress. Because of the huge benefits BDD and TDD provide, and that the author had more experience with web development, a decision was made to abandon Xamarin. This ultimately meant that no Smartphone app was developed for this project, but a web frontend was used instead that could be compatible.

As such, the idea for an app has not been removed completely. As detailed in the Future work section, a simple app could be created that contains an internal web browser and would serve content from the web frontend within that. This situation could have perhaps been avoided had the author chosen to prototype early on, but the same circumstances would have been faced. As a result of this, the author still believes that the same decision would have been made, but a lot earlier on in the project.

The original UI designs of the mobile app have been included in Appendix K. About 2-3 weeks was spent on attempting to implement the Xamarin frontend, to no avail.

#### 4.4.2 Automated code style checking

One of the earliest ideas in the project was to implement the highest degree of code quality throughout the entire development process, including the adoption of Style checkers. StyleCop is one of the largest code checkers used by .Net developers and ensures code quality is kept consistent over the entire codebase (Stylecop,2021).

Several attempts were made to try and integrate StyleCop into the project in its infancy, using multiple versions and other packages that style check. The main issue was that due to a lack of correct documentation, rules could not be customized, and sometimes code checking was buggy. For example, StyleCop would state that a method name should begin with a capital letter, but another rule may conflict and say that it should begin with a lower-case letter. This meant that it was impossible to write code without warnings or errors, and as such it had to be avoided.

As a solution, the author introduced self-disciplined code styling based on their experience previously. This included specific test and method naming conventions, method and class documentation and spacing rules. It is only human to miss these rules, especially when self-disciplined. As an extra check, the author decided to use the GitHub feature of Pull Requests

to enable a retrospective self-review of the code being merged, ensuring that style and quality had been adopted across the entire codebase.

#### 4.4.3 Spotify limitations

After the production of a Minimum Viable Product, a major system limitation was discovered. Based on the Spotify documentation, no limitations on what songs could be retrieved were discovered, but it was only until the system was tested using real activity data it was discovered that the Spotify Get User Recently Played Tracks endpoint was limited to the 50 most recent songs. As a result, integrating Last.FM became a priority as Last.FM keeps a log of all songs listened to, with no data being lost.

This issue with Spotify still has not been resolved but could be resolved with the creation of a data repository that actively logs Spotify tracks as they are listened to. However, Last.FM already does this, so the priority for this solution is not high.

In simple terms, songs for a fastest activity can only be retrieved if the user's last 50 songs were played during said activity. This is ideal if Spotify was only used for running, but with so many other purposes, it is unlikely that all songs that were playing during the user's fastest activity will be returned. The only way to avoid this is to encourage users to use the app immediately after they finish exercising and hope that their most recent exercise was their fastest.

This technical limitation has meant that the best way to use this system is to have a Last.FM account, as otherwise there is a strong possibility no Spotify songs will be displayed. However, Last.FM does not have a large market share, and as such this system should be marketed towards those with a Last.FM account should it be made publicly available.

#### 4.4.4 Ethical approval

One of the more nuanced problems this project faced over the course of its development was achieving ethical approval. As the project's nature is to perform analysis and infer based on personal health data, it became a core requirement to ensure that ethical approval was given to continue with the project. With User Acceptance Testing (UAT) as the primary method of user testing, ethical approval had to be given so that testing could take place.

Initially, it was unclear whether UAT should involve tester's own personal health data; testers would get a better understanding and feel more connected to the test if it were their own data but would come at the cost of ensuring personal data was handled well and that the testers had their own Fitbit, Strava, Spotify and Last.FM accounts.

Due to time constraints and a requirement from the ethics committee for clearer justification on how personal data will be handled, it was decided to use "fake" test accounts with no personal health data. Participants were then required to interact with the system using these fake accounts, and then give their opinion based on their interaction with the system using a questionnaire.

#### 4.4.5 Fitbit limitations

Part of the requirements of this system was that the user should be able to use activities over a large period of their choice to be able to determine what songs made the user run faster. As this was one of the last features to be added to the project, a lot of research took place into the public Fitbit API to find the correct endpoint that would return an authenticated user's exercises.

The "Get user Activity Logs List" endpoint was identified as the endpoint that would satisfy this behaviour. This endpoint seemed to act as intended, as required parameters for use include a "before" or "after" parameter, as well the Oauth2 Authentication token. However, after implementing this endpoint to be integrated with the system a problem was discovered. For some reason, even with multiple checks that the before/after time was correct, only activities from that "week" were being returned. As a result, the system is only able to get activities that took place within the last week of querying.

Checks are still put in place to ensure that any activities returned by the endpoint are within the correct time frame, in case that the origin of this bug is on Fitbit's end. There are several discussions on Fitbit's community regarding the functionality of this endpoint (Fitbit, 2020), but at the time of writing it is unclear whether this is a fault on the authors end or Fitbit. As such, this is an unresolved issue that will require further investigation.

## 5 RESULTS AND EVALUATION

---

In this chapter, the aims and the objectives of the report are directly tested against and evaluated upon, determining the success of the project. This is done using a mixture of UAT and BDD.

As this project has been centred around producing a user-centric solution, user acceptance testing (UAT) has been adopted to ensure that the project's aims have been met to their full capacity. This testing involved users following open-ended instructions whilst interacting with the web frontend, before providing their opinion on the usability, design, and ease of use of the system through the means of a questionnaire. Open-ended instructions have purposefully been used to ensure that users can use the application un-aided, and that the accuracy of responses given in the questionnaires are as unbiased as possible.

BDD, TDD and integration tests were also adopted throughout the development of this project. These tests are explored later within this section, but they are employed to ensure the continued confidence of the system throughout the development lifecycle. These tests also 'stretch' the system to prove robustness and agility of the system under test or "SUT", and give real, actionable, and provable confidence that developed code works as intended.

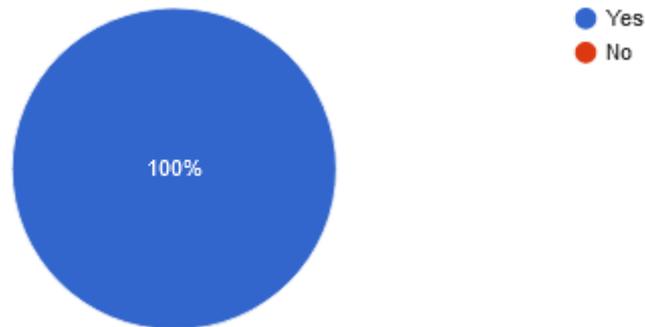
As this project was developed using AGILE, several rounds of UAT were to take place. However, due to a slow ethical approval and a change in project scope, only one round of UAT took place. The aims of this round of UAT were to prove:

1. Primary aim: To create a user-centred, clean user interface that is fit for purpose and encourages ease of use.
2. Implicitly prove the other primary aims, including the creation of an application that has at least 1 music and 1 running data source, as well as the creation of necessary logic to determine what songs made a user run faster.
3. Objective: Employ Test Driven Development and Behaviour Driven Development to ensure the creation of a well-tested, robust mobile application.
4. Objective: Create an intuitive and functional user interface.

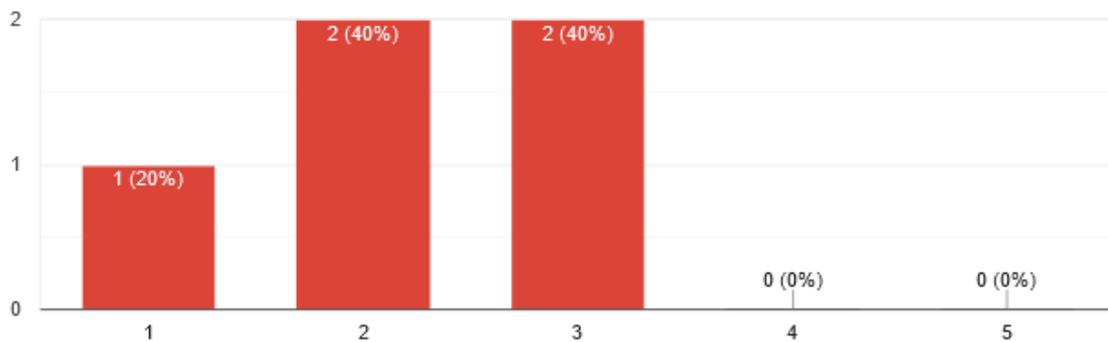
### 5.1 UAT RESULTS

The following questions were presented to individuals, following a short period of interaction with the web app following a test script as aforementioned.

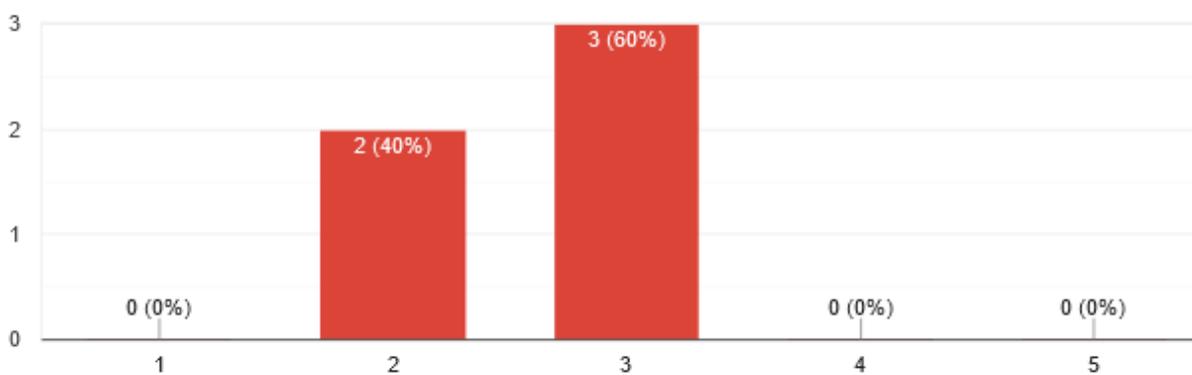
1. After using the provided credentials to log into Strava, Spotify, Fitbit and Last.FM, were you able to view insights based on the user's running and listening history.



2. Rate the experience of connecting the application to Strava (1 being very easy, 5 being very hard).

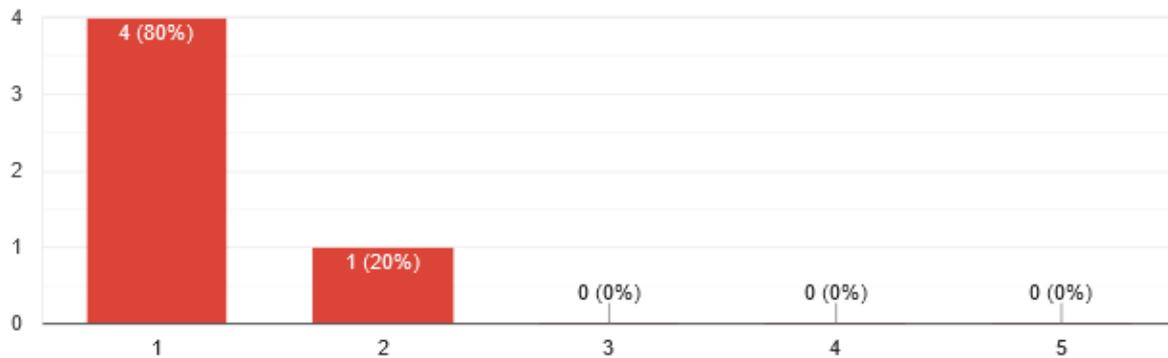


3. Rate the experience of connecting the application to Fitbit (1 being very easy, 5 being very hard).

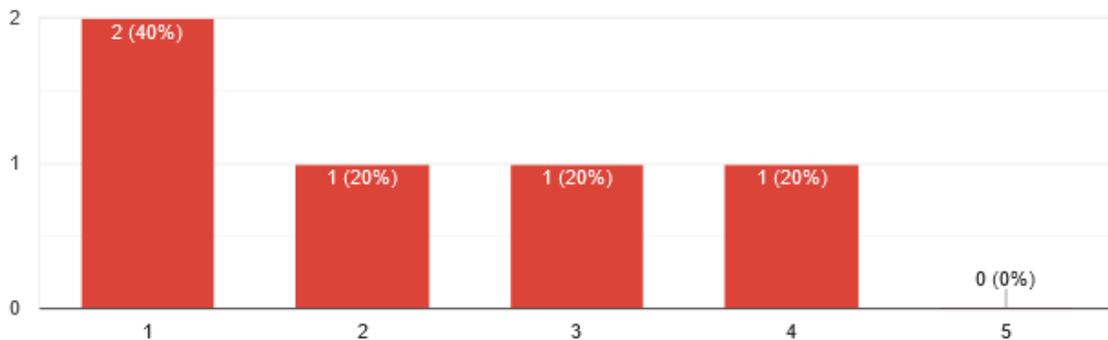


## Can Music Make You Run Faster?

4. Rate the experience of connecting the application to Spotify (1 being very easy, 5 being very hard).



5. Rate the experience of connecting the application to Last.FM (1 being very easy, 5 being very hard).



6. Do you have any comments about your experience with connecting the external services to the application?

The accessibility of connecting my music and fitness apps was really easy.

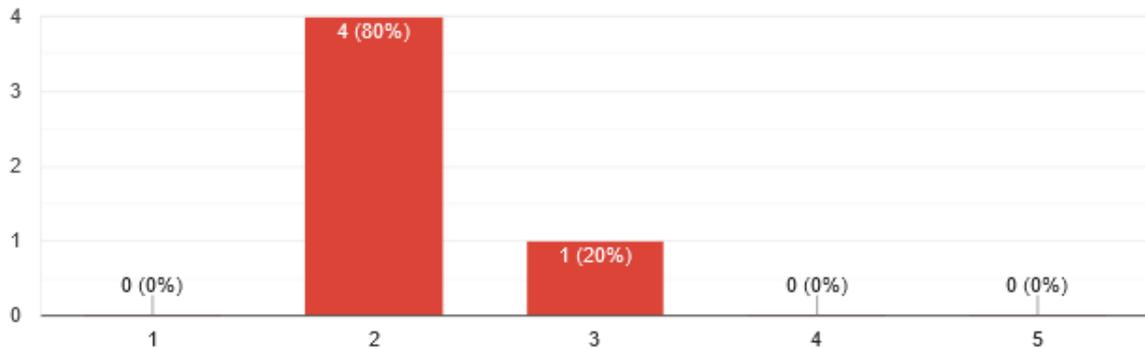
No

Strava and Fitbit both took a long time to connect and said spotify has been connected. Would be nice if it took you back to the initial screen after they had been completed.

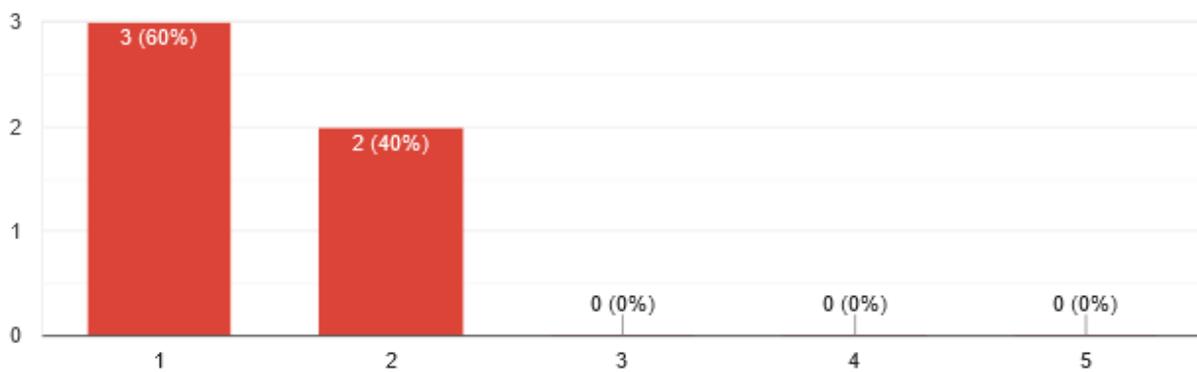
it worked

## Can Music Make You Run Faster?

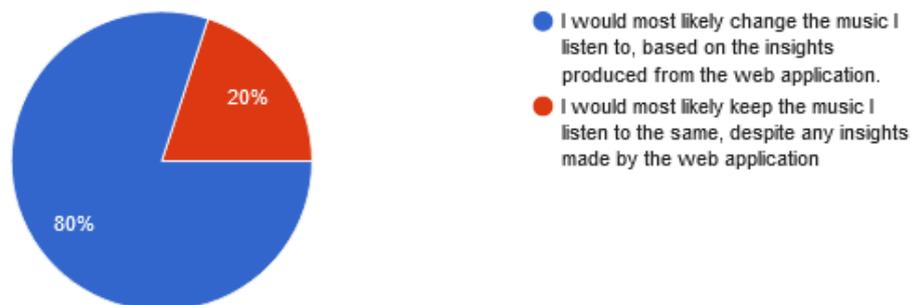
7. Rate the quality of the insights the web app made (1 being very good, 5 being very bad).



8. Rate the usefulness of the insights the web app made (1 being very good, 5 being very bad).

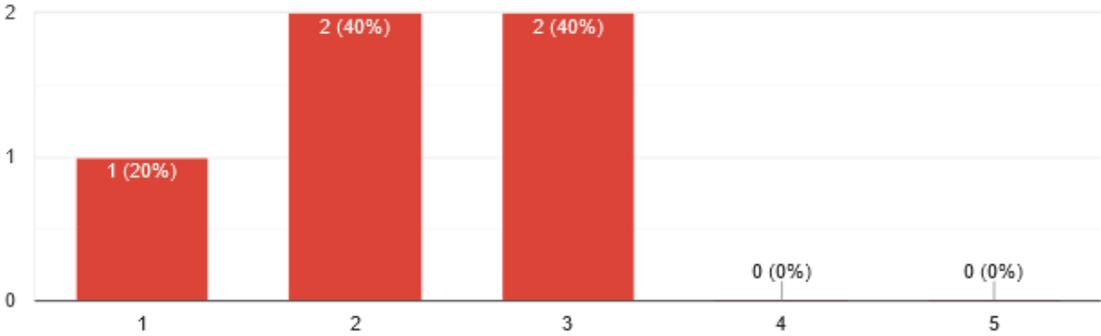


9. If you were to use this application using your own running and listening history, what effect do you think it would have on you, based on your experience with the test you have conducted today?

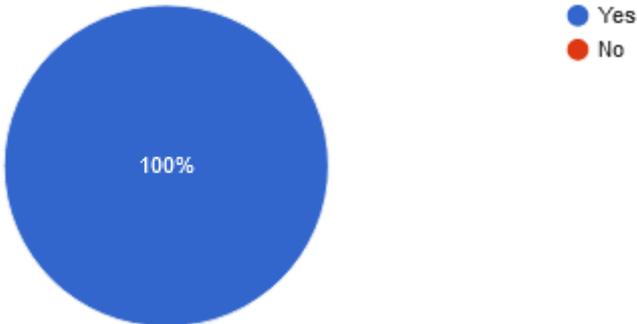


Can Music Make You Run Faster?

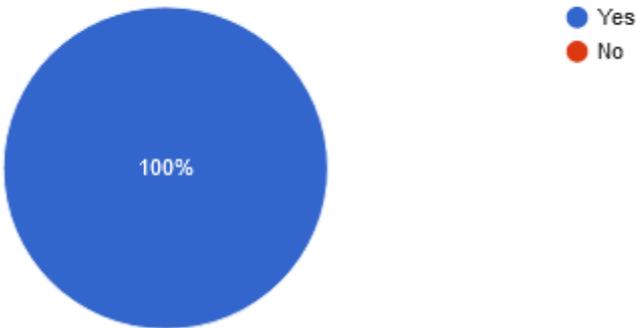
10. How would you rate the layout, aesthetic appeal, and usability of the web app? (1 being very good, 5 being very bad).



11. Do you think the user interface is "fit for purpose"?

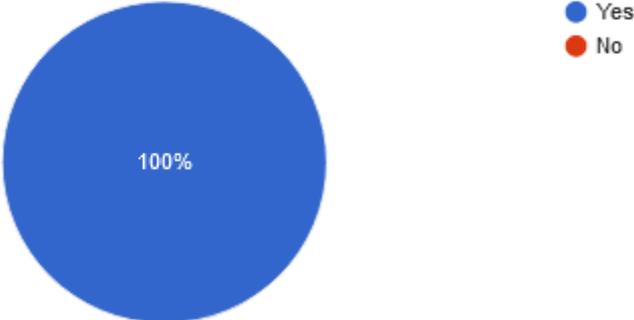


12. Do you think the web application is well built and designed?

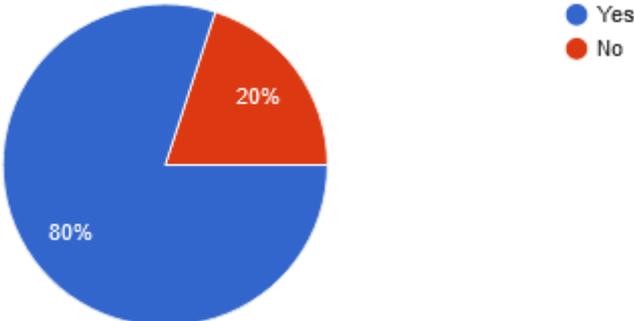


Can Music Make You Run Faster?

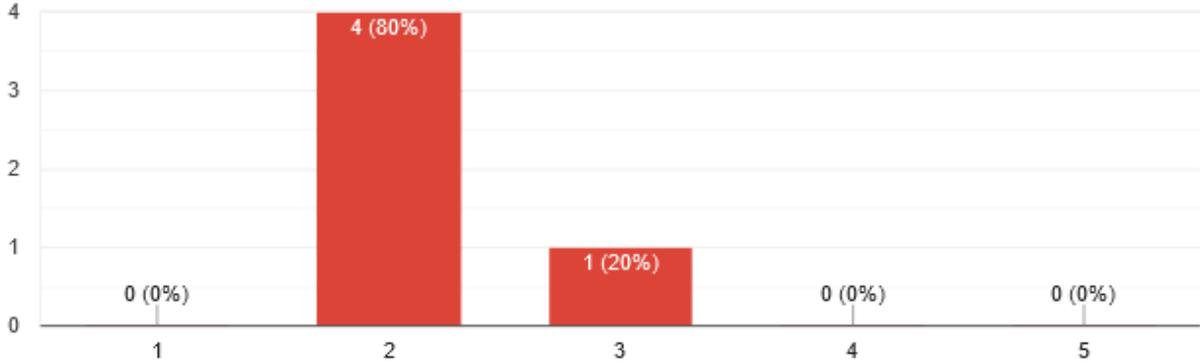
13. Do you think the web application is useful?



14. Do you think you would use this web application, were it to be available publicly?



15. Rate your overall experience using the web app (1 being very good, 5 being very bad).



16. Do you have any other comments about the web application?

Improve website aesthetic
N/A
No
It would be cool to have the songs in some sort of order, either alphabetical, or how much faster they make you run by.

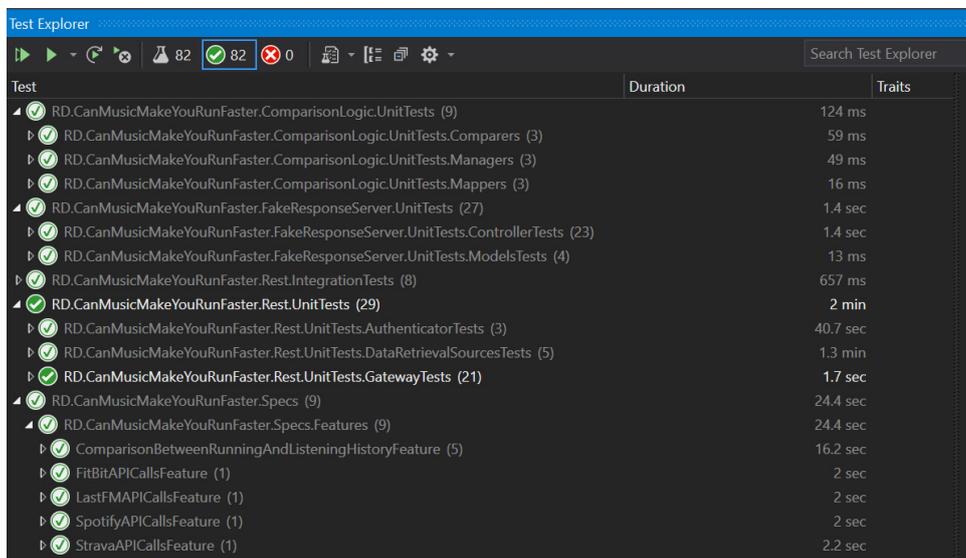
17. Are there any other features you would like to see included within the web app?

No.
Not that I know of
organise songs by fastest to slowest

## 5.2 BDD AND TDD AUTOMATED TESTING

As part of the development of this project, BDD and TDD were employed to ensure that a robust solution was employed, and that the aims of the project could be met to their full potential. As such, **full testing coverage** has been achieved up until the inclusion of the web application, which was manually tested throughout development and then covered by UAT. In other words, the backend has full testing coverage, but when coupled with the web app, there is no end-to-end test, nor TDD tests to accompany the web app.

In total, there are 82-unit, integration and behaviour tests that have been created, employing the use of “fakes” and “mock-ups” to ensure functional components worked as intended. A screenshot of all the required tests passing has been included below:



### 5.3 EVALUATION FROM UAT, BDD AND TDD

Based on the tests conducted throughout development, as well as manual testing and UAT, the author can confidently state that the system works as intended, but there is room for more tests to be conducted.

#### 5.3.1 UAT

User acceptance testing saw all the primary aims of the project being tested against, as well as most of the secondary aims and objectives that were originally described in the initial report. To evaluate the success of the system, it is important to use the results from this round of UAT to define how well the system matched against its original aims and objectives.

##### *5.3.1.1 Primary aim of UAT – Usability*

The primary aim that was tested against during this round of UAT was the third original aim of the initial plan: “To create a **user-centred, clean user interface** that is **fit for purpose** and **encourages ease of use**”. This aim was explicitly and implicitly answered by the questionnaire participants used, including question 11; “Do you think the user interface is ‘fit for purpose’” of which 100% of participants agreed with the statement.

Other questions asked to participants, including questions 10 and 12 also reinforced the idea that this aim had been achieved. Question 10 saw at least 60% of the testers identifying the UI as “good” in response to being asked “how would you rate the layout, aesthetic appeal and usability of the web app?”, with the remaining 40% answering with “neutral”.

Question 12 asked participants on their opinion if they thought the web application was well built and designed, with 100% of participants stating “yes”. Using the responses to questions 10, 11 and 12, clear confidence has been given that the system has been designed to support a clean user interface, that is fit for purpose and encourages ease of use.

Participants were also asked to rate their experiences of connecting the application to the four available external data sources (Strava, Spotify, Last.FM, Fitbit) in questions 2-5, of which the average response was “good” across all questions, as well as 80% of users finding the application very easy to connect to Spotify. Unfortunately, Last.FM’s connectivity did not fare as easily compared to the other sources, with 20% of participants finding the process “hard”, showing more work is required to make the application as user friendly as possible.

To support this, a response to question 16 “Do you have any other comments about the web application” stated that they would suggest to “Improve website aesthetic”, as well as another response to question 6 stating how it was confusing when they connected Strava, and a Spotify authorization feedback screen appeared.

This clearly identifies that more work needs to be conducted to achieve the aim of creating of user-centred, clean user interface that is fit for purpose and encourages ease of use. Nevertheless, the overall user feedback given in questions 10-12 prove that the majority of the userbase was satisfied with aesthetic appeal and the design of the user interface, showing that this primary aim of the project has been achieved, but could use some minor adjustments to become perfect.

Aesthetic appeal and usability are also subjective topics, and no two testers will feel the exact same way about a system. As such, it should not be believed that this aim will be ever fully satisfied because a binary choice on a subjective topic will never cover how the tester truly feels about a system. As such, the author is satisfied that this aim has been achieved, but there are some small adjustments that could be made.

#### *5.3.1.2 Other primary aims covered.*

The other primary aims covered by this round of UAT were:

1. To create a mobile application for Android, integrating at least 1 running application and 1 music application.
2. To create the necessary logic required to enable a comparison between the user's running activities and their listening history, comparing metrics such as pace and average cadence to the music they were listening to.

Whilst these questions are not typically answered by UAT, there were questions put to participants that covered these aims. For example, Question 1 asks "after using the provided credentials to log into Strava, Spotify, Fitbit and Last.FM, were you able to view insights based on the running and listening history?" of which 100% of participants agreed.

Question 6 also reinforces this idea, where one user stated, "the accessibility of connecting ... music and fitness aps was really easy", and another simply stated "it worked". However, one tester identified that "Strava and Fitbit took a long time to connect and said Spotify has been connected. Would be nice if it took you back to the initial screen after they had been completed". This response implies that the system achieved the primary aims but behaved abnormally to what they would have expected. This could illude to real-world users becoming confused with the state of the system and lead to dissatisfaction, showing that users may not get as much out of the solution as they want to.

Another caveat is that the first primary aim has required a "mobile application for Android", and whilst react does typically support android functionality, the project has not been directly tested on an Android device, and as such there is no way of knowing if the application does support Android.

Based on the evidence above, the author believes that these primary aims have been achieved as expected, but more work could be done to better reflect the state of the system to the user so they can gain the confidence that the system matches their mental model. Multi-platform tests will also need to be introduced to give the system further confidence that it is compatible with Android devices.

### 5.3.1.3 Objectives

This round of UAT also saw two of the objectives of the project being tested against, including:

- Employ Test Driven Development and Behaviour Driven Development to ensure the creation of a well-tested, robust mobile application.
- Create an intuitive and functional user interface.

These objectives were directly tested against throughout the round of UAT, but the question that addresses both objectives the most is number 15: “Rate your overall experience using the web app”. 80% of users rated their experience as “good”, with the remaining 20% rating their experience as “neutral”. This implicitly proves an intuitive and functional user interface was created, as if the opposite were true the user’s overall experience of using the application would be significantly lower. This also implicitly proves a well-tested, robust application has been created, as otherwise any issues the tester may have encountered would severely reduce the overall rating of the web application.

Additionally, question 12 “Do you think the web application is well built and designed?” directly assesses the creation of a well-tested and robust mobile application. In question 12, 100% of participants agreed that the application was well built and designed, reinforcing the author’s opinion that this objective has been achieved. However, this objective is orientated around a “robust mobile application”, meaning that this test suffers from the same problems detailed in the “Other Primary Aims” section where these tests need to be repeated on an Android device to prove mobile portability.

The remaining objective to create an intuitive and functional user interface has been previously answered in the “Primary aim of UAT – usability” section. Based on this, and the evidence listed above, the author believes that both objectives have been fully achieved but would require multi-platform testing to give complete confidence that the application works as intended.

### 5.3.2 Limitations on UAT

Favourable opinions have been given by the author on the state of the system based on the evidence presented from UAT, but there are issues over the validity of UAT. Preceding this section, concerns have been raised over the lack of mobile platform testing and should be addressed in future tests if this project were to continue.

Another issue with this round of UAT was that the sample size was small, with only 5 participants taking part. To ensure that the application continues to enjoy meeting its aims, future rounds of UAT should encompass more participants to ensure that the results of said tests are the most reflective of a public userbase as possible.

### 5.3.3 BDD and TDD

Behaviour Driven Development and Test-Driven Development enabled the author to achieve test coverage of almost 100%, with some tests missing on the React Typescript web frontend and end-to-end tests to compliment the system. Despite this, the tests that are currently in place cover the two out of three primary aims of the project, namely:

1. To create a mobile application for Android, integrating at least 1 running application and 1 music application.
2. To create the necessary logic required to enable a comparison between the user's running activities and their listening history, comparing metrics such as pace and average cadence to the music they were listening to.

Integration and behaviour driven development tests ensure that both the first and second aim has been achieved, as integration and behaviour driven tests directly ensure that each running and music application has been tested thoroughly, even in atypical situations where access tokens have expired etc.

Integration and BDD tests directly ensure that both the first and second aims have been achieved. To begin with, the first aim has been achieved fully due to the rigidity of the tests that have been employed; integration and BDD tests assure that each running and music application has been tested thoroughly, including each stage of interaction; from OAuth2 token acquisition to querying each data source with variable parameters. Each test added gives more and more confidence that the first aim has been achieved, and that the application has versatility and portability through the interchangeable use of testing with a mock-API and the "real" API.

The second aim has been fully achieved predominantly by the BDD tests, where the logic required to enable a comparison between user's running and music is directly queried. BDD tests as previously mentioned in "Specification and Design" use Gherkin syntax as instructions to run each test, and the screenshots of each business requirement can be found in appendices A-E. Appendix E details a set of business requirements that directly require the use of multiple data sources and expected results for each, thus proving that the necessary logic required to enable a comparison between the user's running activities has been made.

The latter part of the second requirement includes the "[comparison] of metrics such as pace and average cadence to the music they were listening to." This project has only been successful in using the average pace metric and has been tested again using BDD tests. Each BDD test involves a "given" step, in which variable data can be inserted and objects can be created from this. For example, it is possible to have two different activities where one has a faster pace than the other, and accordingly this concept has been adopted in the BDD tests to ensure that the second requirement is proven to its full capacity.

### 5.3.4 General project evaluation

The success of the project can also be assessed in terms of overall user acceptance, and not just the aims of the project. UAT results saw users generally satisfied with the solution produced, with 80% of testers rating their experience as "good" in UAT question 15. The concept of the application has also been tested against, with UAT question 9 asking if users would change the music they listen to based on the insights produced by the app, with 80% of users saying they would. In a similar question, UAT question 13, 100% of respondents found the web application useful, and 80% of users would use the application were it to be available publicly (as shown in UAT question 14).

The responses to the aforesaid questions also undoubtedly reinforce the idea that this project is a success, and there is an appetite from testers to see more functionality. As a result, the author's opinion of this project being a success is strengthened by this hypothesis. However, in UAT question 16 and 17 said they would like to see some organisation of songs being displayed, whether it be alphabetical or sorted by impact. This shows that there is still room for improvement with the solution, and the provided software has created a strong base for future development.

#### 5.4 FUTURE TESTS REQUIRED

As previously mentioned from the "Evaluation from UAT, BDD and TDD" section, the next round of UAT must take place on a multi-platform level to prove smartphone capability. This set of UAT would provide full confidence in the system, as only at this stage would the aims of the project be fully tested against.

The other aspect of testing that is required is to expand the level of BDD and TDD tests that are currently employed within the system. Current BDD tests do not cover an end-to-end level of testing when using the web application, nor TDD tests on the web application to prove a robust application has been produced. These tests are essential, as without these tests no grounded confidence can be given to the system when required to perform as expected.

#### 5.5 SUMMARY AND CRITICAL APPRAISAL OF APPROACH

Based on the collective evidence shown in this section and the evidence presented within the "implementation" section, the author has a strong belief that all major aims and objectives have been completed, but further testing is required to give full confidence of a fully robust and functional solution. User sentiment from UAT supports this as previous shown in this section, with some small improvements suggested by users.

In future, further testing should include multi-platform UAT, as well as a small change in BDD tests to increase overall test coverage. This increase in "technical" testing should also later include deployment testing as to ensure the system maintains its crucial ability to be versatile and operate on as many mobile platforms as possible as per the original aims of this project.

Overall, coupled with some small adjustments stated above, a great basis for future development on this project has been created with all primary and some secondary aims achieved. If a percentage of completion is required to be stated, a confident figure of 80% completion could be given. The remaining 20% is unknown, as there are no mobile platform tests to prove the system works. This 80% figure could be higher as the system could work as intended on smartphones, but as aforementioned there is no real way of knowing unless tested.

By achieving this 80% figure, support for the author's choice of adopting BDD, TDD and an AGILE methodology is given. If the author were to choose a waterfall methodology to begin with, the project would have most likely suffered as an agile methodology allowed the author to gain flexibility with what frontend was used. This flexibility was paramount as the development of the project would have to have been restarted due to the compatibility issue

between the Xamarin project and the .Net Core backend as highlighted in the implementation section. An AGILE methodology allowed enough freedom to avoid this, and plan well enough in advance to allocate enough time to create a replacement web frontend.

In terms of technology used, it was the correct choice to use a .NET backend. Due to the original secondary aim of supporting iOS and Android, a decision was made to have a centralised backend and only use the mobile apps for iOS and Android as a user interface, with no business logic stored in either app. This gave the author a relatively painless transition to switch to a web frontend when the time came.

If the business requirements were different, and the project was focussed on creating an android only application, the technology used would have been different. Instead, Android Studio would've most likely been used instead, with all business logic stored within the android app that would have been developed. This approach would have avoided the compatibility issue with a .NET core backend but would also be exclusively locked into Android support only. As such, the author believes this versatility provided by avoiding using technology such as Android studio has meant that the project was able to continue despite adversity, and if this alternate approach faced the same problems as the original approach faced, this project's completion would not be what it is today.

Using C# did have its limitations, however. As an object orientated language, simple operations can have a large amount of complexity, and a lot of time had to be spent on ensuring a good design was adopted throughout development of the system. Coupled with the technical overhead of using BDD and fakes, the process had to be repeated for both "ends" of the system, meaning that scrapped secondary features such as the suggested music functionality could have been included if time were not spent on ensuring a robust OOP-SOLID design.

Building on the limitations of BDD, adopting the process of testing-develop-refactor meant that a lot of overhead code had to be developed to ensure confidence was given in the system. The benefit of this confidence is immeasurable, but time spent working on BDD tests could have otherwise been spent on developing other secondary features. This would have come at the cost of robustness of the system, but otherwise would have advanced the system to achieve more aims.

Overall, the author believes that a great foundation for future development has been established, with the correct choice of methodology, techniques and programming languages used. These choices may have come at the cost of some features that were originally included within scope, but instead confidence has been put in place that ensures functionality really does work as expected.

## 6 FUTURE WORK

---

When this project first began, some features have been removed from scope. For example, one of the secondary aims was to create a “a secure login system, complimented with back-end database to store user information”. The current state of the system does not require this and would only require this feature if the application had the need to store user data over a long period of time.

As this project began and end as a prototype, there is a large potential for future work to be performed on this project. In the immediate term, the very first thing that needs to be done is to update the BDD tests. This would involve refactoring of logic in the “ApiClientDriver” to better reflect the current state of the system, and the creation of a “WebClientDriver”, which would involve the use of the web frontend in testing. This would also involve the addition of TDD tests for the React application. Had I not run out of time, this would and should have been completed.

To support automated testing and a DevOps CI and CD approach, GitHub actions could then be employed to automatically test any updates to the application and ensure the portability of the solution on multiple platforms.

After this, the project would have good grounding for more additional features. As this project only has the capability to compare activities based on the user’s average speed, it would be apt to increase the number of “insights” given to the user i.e., the user should also be able to determine what songs are best for their pace. These insights could then be extended to what genre is best for their exercises, or even finding the artist that has the greatest effect on the user’s running activity. The addition of this functionality would most likely take place in the medium term, because it would be necessary for a small amount of refactoring of backend logic to support different metrics being measured other than average speed.

As the project was built in a scalable manner, there is more potential for other “data sources” to be added to the project, including more activity tracking applications and music applications. In the medium term, this would be ideal for users as this gives them more flexibility of choice when it comes to what services they would like to connect to the application. In the longer term, more data sources offer the opportunity to compare other activities that differ from running such as cycling, walking, and hiking. Again, this would create more flexibility for the user, and force the application to be more robust as more logic is added.

Current data sources in the system also need adjustment, namely Spotify and Fitbit. As aforementioned, Spotify is currently limited to returning 50 songs, and Fitbit can only get activities from the current week. As such, a storage system could be corrected that adds activities and music as an authenticated user logs them. However, this would increase the complexity of the project and require several security and data protection environments, potentially exceeding the scope of what is required for the system.

The user knowing the state of the system is also very important, especially for a project such as this. It has been considered to include the potential use of highlighting dates on a calendar

where activities have taken place once the user has connected their running history. This however would require significant modification to the system to be a complete feature, as the solution currently only gets activities within the specified date range and not the user's complete activity history.

As this project has always been user-need centred, in the longer term more features can be added that will complement their needs. As a React Typescript frontend has been used, the opportunity for mobile usage has been retained. A mobile app could be created containing a simple integrated web browser that is directed at the website, and as a result would give mobile users access to the application. This however would require hosting of the web frontend and the API backend, as well as the potential requirement of a "secondary" website that would serve mobile-sized content instead of using the frontend that has been produced so far.

This application also has the potential to offer more advanced features, including automatic creation of a playlist based on what songs made them run faster, as well as more in-depth analysis of running activities. For example, this would include analysis of Strava Segments and the metrics produced from each attempt. However, the complexity of a feature such as this would increase dramatically as more data sources are added. This is because of a need of uniformity across each activity data source, and as a result each data source would be required to offer the same functionality (if possible).

## 7 CONCLUSIONS

---

When this project started, its simple aim was to really determine if music could make you run faster. This was meant to be achieved by producing a mobile application that allowed users to connect their running and music applications, and then displaying a list of songs to the user that makes them run faster. Each song was meant to be “proven” to make the user run faster, as each song would be taken from their fastest running activity. This “fastest” activity was meant to originally meant to measure multiple metrics, including average speed and pace, but ultimately receded to just average speed.

This project was also coupled with several primary and secondary aims, including:

1. To create a mobile application for Android, integrating at least 1 running application and 1 music application.
2. To create the necessary logic required to enable a comparison between the user’s running activities and their listening history, comparing metrics such as pace and average cadence to the music they were listening to.
3. To create a user-centred, clean user interface that is fit for purpose and encourages ease of use.

To supplement the aims above, secondary aims were also set for this project:

1. To create a secure login system, complimented with back-end database to store user information.
2. To extend the logic comparing music and running activity data, enabling the user to view various impacts and music on their activity.
3. To extend the functionality to suggest other songs that may be used to increase a user’s running average pace.

Based on the evidence presented throughout this report, it is the author’s strong belief that all primary aims have been achieved, with strong foundations in place to implement the remaining secondary aims. Some of said secondary aims have been removed from scope, including the creation of a secure login system. Plans for this project to support Android and iOS were at the centre point of this project as it began, but due to a lack of prototyping and research it had to be scrapped in favour of a web frontend. A web frontend now exists in their place that should be compatible on all smartphones, regardless of what operating system they run.

This project also adopted several techniques that are pioneering the software development world, namely TDD and BDD. These techniques prioritised testing above all else, and ultimately gives this system enough confidence to know that all 3 primary objectives have been achieved, and without it, there would be no way of knowing for sure.

User Acceptance Testing was also at the forefront of this project and was only implemented towards the end of the project, instead of its original intention of taking place throughout development. The evidence gathered from the singular round of testing overwhelmingly supported the author’s sentiment that the system achieved all primary goals.

## Can Music Make You Run Faster?

There is some more room for improvement however, with only small changes in the codebase and a few additional layers of tests required to make the system “perfect”. Other limitations, such as the Spotify recently played limit have hindered the potential applications of the system but have been constituted by features provided by other sources such as Last.FM.

In conclusion, a substantial foundation for future development has been established with all primary aims being achieved. Despite facing adversity within numerous technical problems, this project was able to deliver a mobile-compatible interface that can display a list of songs to the user that make them run faster. Whether said songs will make the user run faster is another question and will surely require more testing with the opinion of users on whether they truly believe the songs produced by this app makes them run faster.

## 8 REFLECTION ON LEARNING

---

Completing this project has been paramount in advancing the authors skills as a software developer, especially in understanding how best to utilise web APIs and building a system around them. As a learning process, there have been some issues faced throughout the course of this project that have been detailed below.

The largest issue that the author faced throughout development was time management. Often delivery times of stories was arbitrary at best, and often was not “timeboxed” correctly. Each story was different, and for most of the time the author was faced with one or more unexpected issues throughout each sprint, as to be expected. To improve, this could be solved in multiple ways. The first way is to reduce the size of each story into more manageable chunks – arbitrary story completion times suggest that not every story was the same size due to a lack of substantial refinement, and as such the author’s ability to research and refine stories should be improved in future.

The second method of improving the author’s time management would be to spend more time at the end of each sprint to reflect on the challenges faced whilst working on each story and identify and problems that often resurfaced. Future work after the end of sprint would then move to avoid these areas, and if not possible, work on a solution minimize the problem as much as possible. Time management issues often resurface due to a lack of reflection, and as such this method should be adopted to ensure that the assumptions of how long a task should take is always being challenged to best reflect the author’s ability.

Another issue that the author faced throughout development was a lack of research into future actions / technologies being used. This is most prevalent with the failure to implement a mobile frontend; had the author attempted to prototype much earlier into the development cycle, or spent more time investigating the capability of using Xamarin and a .NET Core backend. Had the author had to repeat this project again, he would have chosen to spend the first few weeks creating a basic prototype to ensure that all the plans he had for the project were achievable, and that plans were in place should the possibility of a mobile frontend not be possible.

Conversely, this project has highlighted the importance of adopting good software development techniques and practises. BDD and TDD enveloped the project in a large amount of technical and time overhead, but ultimately proved paramount in determining the success of the system. Without using BDD and TDD, arguably more time could have been spent fixing code after it had been manually tested as the testing architecture created by using this technique allows for easy identification of what functional components are causing issues.

Additionally, the author believes that his ability to plan paid off in this project. Whilst the project suffered from meaningful enough research, secondary and tertiary plans were always put in place as fallbacks in case anything failed. This came to fruition especially with the failure to implement a Xamarin frontend, as without the foreplaning to use a web frontend this project would not have come as far as it did.

The only remaining change the author would like to make for future projects would be to make the project as full stack as possible. This project suffered with a lack of direction in the early stages due to the author wanting to develop as much as possible within the smallest time possible. As such, the author felt that it was possible to develop a Xamarin frontend tailoring for both iOS and Android, without focussing on solely one or the other. Because of this, BDD tests were only designed to have coverage up until the REST API stage, until the web frontend was added consequently added time being wasted attempting to get the BDD tests updated with the remainder of the system, something that the author failed to incorporate in the end.

Full stack development would avoid this scenario and would mean that BDD tests have maximum coverage as early on in development as possible. In future projects, the author will choose to adopt this technique in the hope that time wasted is minimized, and more time can be spent on providing functionality for the system.

## 9 TABLE OF ABBREVIATIONS

Abbreviation	Meaning
'API'	Application Programming Interface.
'BPM'	Beats per Minute.
'BDD'	Behaviour Driven Development.
'CI'	Continuous Integration.
'CD'	Continuous Deployment.
'PKCE'	Proof Key for Code Exchange.
'TDD'	Test Driven Development.
'React TS'	React TypeScript.
'UAT'	User Acceptance Testing.

## 10 APPENDICES

```

1 Feature: Spotify API Calls
2
3 Background:
4 Given a list of users
5 | user      |
6 | User1     |
7
8 Given a list of Spotify listening history
9 | user | Song name                | Time of listening |
10 | User1 | The Chain - 2004 Remaster | 15/02/2021 15:45:30 |
11 | User1 | I Want To Break Free - Single Remix | 15/02/2021 15:40:01 |
12 | User1 | Good Vibrations - Remastered | 15/02/2021 15:30:59 |
13 | User1 | Dreams - 2004 Remaster | 15/02/2021 00:05:00 |
14 | User1 | Stayin Alive | 14/02/2021 23:59:59 |
15
16
17 @MVP-0-Add-Spotify-API-Call
18 Scenario Outline: Get User's Spotify listening history
19     Given a user <user>
20     And their Spotify listening history
21     When the user's Spotify recently played history is requested
22     Then the user's Spotify recently played history is produced
    
```

Appendix A – Typical Gherkin business requirement (Also the Spotify business requirement).

```

Feature: Strava API Calls

Background:
Given a list of users
| user      |
| User1     |

Given a list of Strava running history
| user | Activity Name                | Time of activity start | Elapsed Time of Activity (s) | Activity Id | Average Pace (m/s) |
| User1 | Cardiff Friday Morning Run | 15/03/2021 12:00:00 | 4410 | 1 | 4.5 |
| User1 | Oxford Half Marathon | 14/03/2021 08:00:00 | 9000 | 2 | 4.6 |
| User1 | Roath Lake Midnight Run | 13/03/2021 23:39:59 | 4410 | 3 | 4.2 |
| User1 | Late Night Run | 10/03/2021 00:05:00 | 1280 | 4 | 4.2 |
| User1 | Test Run | 01/01/2021 23:59:59 | 60 | 5 | 4.2 |

@mvp-4-Add-Strava-API-Call
Scenario Outline: Get User's Strava Running History
    Given a user <user>
    And their Strava running history
    When the user's recent Strava running history is requested
    Then the user's recent Strava running history is produced
    
```

Appendix B – Get a user's Strava running history.

## Can Music Make You Run Faster?

```
1 Feature: Last FM API Calls
2
3 Background:
4 Given a list of users
5 | user |
6 | User1 |
7
8 Given a list of Last.FM listening history
9 | user | Song name | Time of listening |
10 | User1 | The Chain - 2004 Remaster | 15/02/2021 15:45:30 |
11 | User1 | I Want To Break Free - Single Remix | 15/02/2021 15:40:01 |
12 | User1 | Good Vibrations - Remastered | 15/02/2021 15:30:59 |
13 | User1 | Dreams - 2004 Remaster | 15/02/2021 00:05:00 |
14 | User1 | Stayin Alive | 14/02/2021 23:59:59 |
15
16
17 @EDS-1-Add-Secondary-Data-Sources
18 Scenario Outline: Get User's Last.FM listening history
19 Given a user <user>
20 And their Last.FM listening history
21 When the user's Last.FM recently played history is requested
22 Then the user's Last.FM recently played history is produced
```

Appendix C – Get a user's Last.FM listening history.

```
1 Feature: FitBit API Calls
2
3 Background:
4 Given a list of users
5 | user |
6 | User1 |
7
8 Given a list of FitBit running history
9 | user | Activity Name | Time of activity start | Elapsed Time of Activity (s) | Activity Id | Average Speed (m/s) |
10 | User1 | Cardiff Friday Morning Run | 15/03/2021 12:00:00 | 4410 | | 1 | 4.5
11 | User1 | Oxford Half Marathon | 14/03/2021 08:00:00 | 9000 | | 2 | 4.6
12 | User1 | Roath Lake Midnight Run | 13/03/2021 23:39:59 | 4410 | | 3 | 4.2
13 | User1 | Late Night Run | 10/03/2021 00:05:00 | 1280 | | 4 |
14 | User1 | Test Run | 01/01/2021 23:59:59 | 60 | | 5 |
15
16 @EDS-1-Add-Secondary-Data-Sources
17 Scenario Outline: Get User's FitBit Running History
18 Given a user <user>
19 And their FitBit running history
20 When the user's recent FitBit running history is requested
21 Then the user's recent FitBit running history is produced
```

Appendix D – Get a user's Fitbit running history.

# Can Music Make You Run Faster?

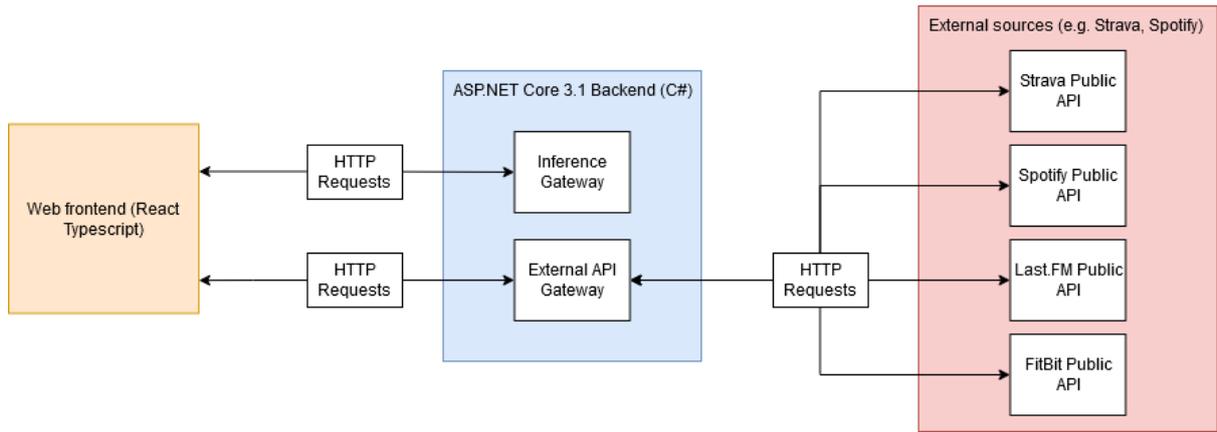
```

1 Feature: Comparison between running and listening history
2
3 Background:
4 Given a list of users
5 | user |
6 | User1 |
7
8 Given a list of Strava running history
9 | user | Activity Name | Time of activity start | Elapsed Time of Activity (s) | Activity ID | Average Pace (m/s) |
10 | User1 | Cardiff Friday Morning Run | 15/03/2021 22:00:00 | 1420 | 1 | 4.5 |
11 | User1 | Oxford Half Marathon | 15/03/2021 08:00:00 | 5000 | 2 | 4.2 |
12 | User1 | South Lake Midnight Run | 13/03/2021 23:30:59 | 4410 | 3 | 4.2 |
13 | User1 | Late Night Run | 10/03/2021 00:05:00 | 1280 | 4 | 5 |
14 | User1 | Test Run | 03/03/2021 23:59:59 | 60 | 5 |
15
16 Given a list of Fitbit running history
17 | user | Activity Name | Time of activity start | Elapsed Time of Activity (s) | Activity ID | Average Speed (m/s) |
18 | User1 | 1K Better test | 15/03/2021 15:00:00 | 4410 | 1 | 7.0 |
19 | User1 | Cardiff Half Marathon | 14/03/2021 18:00:00 | 9500 | 2 | 7.0 |
20 | User1 | Late Midnight Run | 22/03/2021 23:39:59 | 3700 | 3 | 7.0 |
21 | User1 | Late Night Run 2 | 21/03/2021 00:05:00 | 1280 | 4 | 7.0 |
22 | User1 | Oups | 15/11/2020 23:59:59 | 60 | 5 |
23
24 Given a list of Spotify listening history
25 | user | Song name | Time of listening |
26 | User1 | The Chain - 2004 Remaster | 15/03/2021 12:00:05 |
27 | User1 | I Want to Break Free - Single Remix | 15/03/2021 12:00:30 |
28 | User1 | Good Vibrations - Remastered | 15/03/2021 12:04:59 |
29 | User1 | Dreams - 2004 Remaster | 15/03/2021 12:09:20 |
30 | User1 | Stayin' Alive | 15/03/2021 12:07:20 |
31 | User1 | Junk Song | 15/03/2021 23:59:20 |
32 | User1 | Riptide | 14/03/2021 00:00:01 |
33 | User1 | Can't Hold Us | 14/03/2021 00:00:20 |
34 | User1 | Starboy | 14/03/2021 00:00:40 |
35 | User1 | Beautiful Day | 14/03/2021 00:30:00 |
36 | User1 | Starmann | 13/03/2021 00:02:00 |
37 | User1 | Kickstarts | 13/03/2021 00:05:00 |
38 | User1 | Sugar | 10/03/2021 00:05:01 |
39
40 Given a list of Last.fm listening history
41 | user | Song name | Time of listening |
42 | User1 | The Chain - 2004 Remaster | 15/03/2021 15:05:30 |
43 | User1 | I Want to Break Free - Single Remix | 15/03/2021 15:08:01 |
44 | User1 | Good Vibrations - Remastered | 15/03/2021 15:10:59 |
45 | User1 | Dreams - 2004 Remaster | 15/03/2021 00:30:00 |
46 | User1 | Last Day on Earth | 14/03/2021 18:00:00 |
47 | User1 | Superheroes | 15/03/2021 15:00:05 |
48 | User1 | Stepping Stone | 15/03/2021 15:03:30 |
49
50
51
52 @BDD-5 Single-Date-Comparison
53 Scenario: Compare Listening And Running History
54 Given a user <user>
55 And their Strava running history
56 And their Spotify listening history
57 When the user's recent Strava running history is requested
58 And the user's recently played history based on their running history is requested
59 And the comparison between running and listening history is made
60 Then the user's top tracks for running faster are produced
61 | Song name |
62 | The Chain - 2004 Remaster |
63 | I Want to Break Free - Single Remix |
64 | Good Vibrations - Remastered |
65 | Dreams - 2004 Remaster |
66 | Stayin' Alive |
67
68 @BDD-6 Notebooks: Comparison
69 Scenario: Compare Listening and Running History with date range
70 Given a user <user>
71 And their Strava running history
72 And their Spotify listening history
73 When the user's recent Strava running history is requested
74 And the user's recently played history based on their running history is requested
75 And the comparison between running and listening history is made using a specified date range
76 Then the user's top tracks for running faster are produced
77 | Song name |
78 | Riptide |
79 | Can't Hold Us |
80 | Starboy |
81 | Beautiful Day |
82
83 @BDD-7 Multiple-Date-Source-Comparison
84 Scenario: Compare Listening and Running History on singular date with alternative data sources
85 Given a user <user>
86 And their Fitbit running history
87 And their Last.fm listening history
88 When the user's recent Fitbit running history is requested
89 And the user's recently played history based on their running history is requested
90 And the comparison between running and listening history is made
91 Then the user's top tracks for running faster are produced
92 | Song name |
93 | Superheroes |
94 | Stepping Stone |
95
96
97 @BDD-8 Multiple-Date-Source-Comparison
98 Scenario: Compare Listening and Running History on singular date AND multiple data sources.
99 Given a user <user>
100 And their Fitbit running history
101 And their Strava running history
102 And their Last.fm listening history
103 And their Spotify listening history
104 When the user's recent Strava running history is requested
105 When the user's recent Fitbit running history is requested
106 And the user's recently played history based on their running history is requested using multiple data sources
107 And the comparison between running and listening history is made
108 Then the user's top tracks for running faster are produced
109 | Song name |
110 | Superheroes |
111 | Stepping Stone |
112
113
114 @BDD-9 Multiple-Date-Source-Comparison
115 Scenario: Compare Listening and Running History with date range AND multiple data sources.
116 Given a user <user>
117 And their Fitbit running history
118 And their Strava running history
119 And their Last.fm listening history
120 And their Spotify listening history
121 When the user's recent Strava running history is requested
122 When the user's recent Fitbit running history is requested
123 And the user's recently played history based on their running history is requested using multiple data sources
124 And the comparison between running and listening history is made using a specified date range
125 Then the user's top tracks for running faster are produced
126 | Song name |
127 | Last Day on Earth |

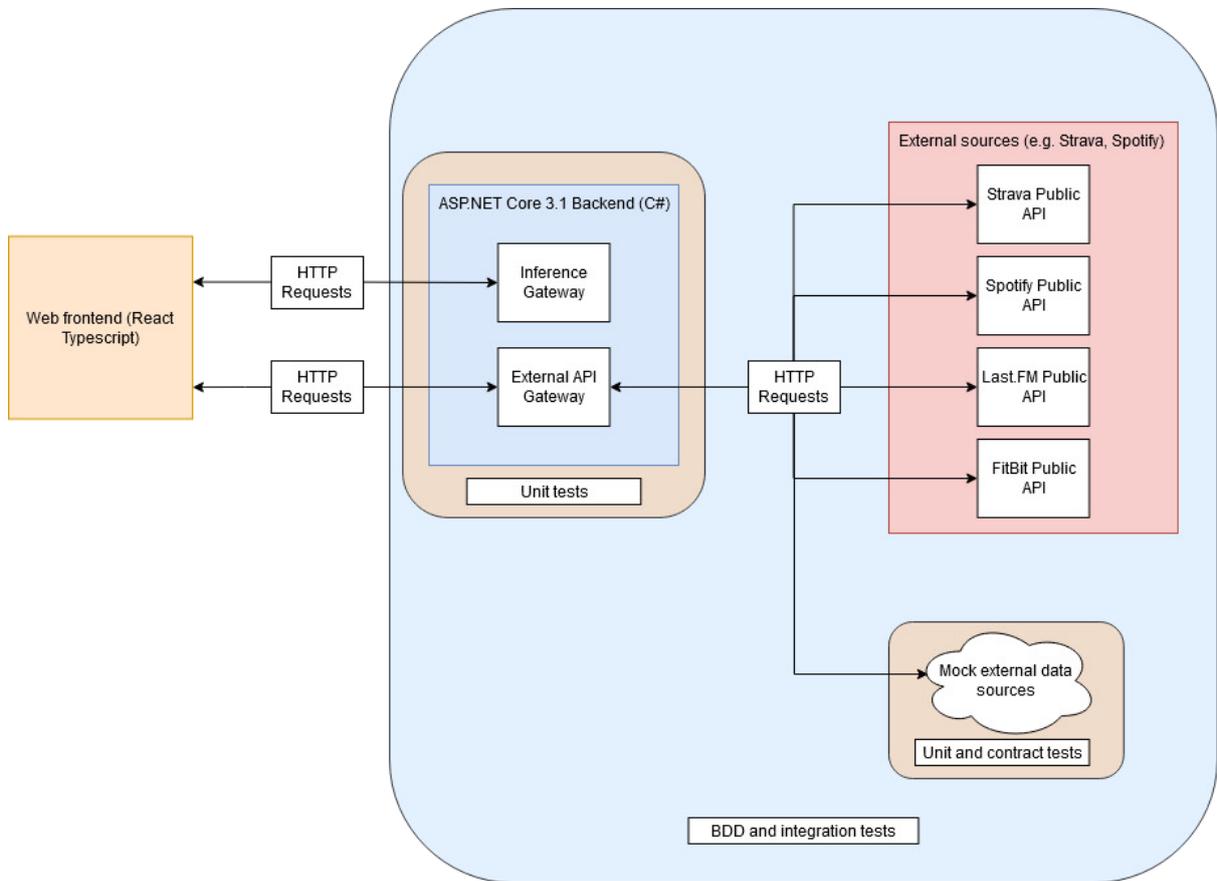
```

Appendix E – Comparison business requirements.

# Can Music Make You Run Faster?



Appendix F – Highest level system overview



Appendix G – Highest level testing architecture.

# Can Music Make You Run Faster?

## Can Music Make You Run Faster?

A running-music comparison app by Rhys Douglas.

Pick a date range to search between

Start Date: 01/01/2021      End Date: 26/05/2021

Connect your running history

Connect with STRAVA      Connect with Fitbit

Connect your listening history

Sign in to Spotify      Enter your Last.FM username

Status: Spotify connected successfully!

What songs make me run faster?

### Songs that make you run faster



The Flood  
Take That



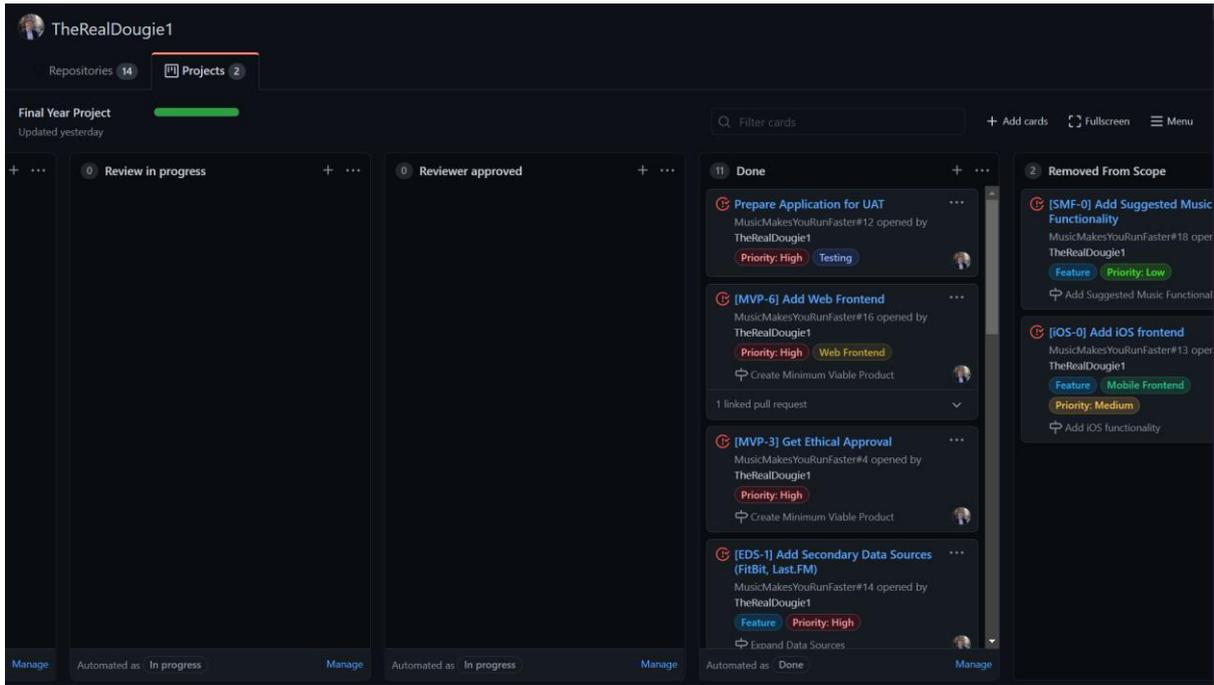
Fossils  
Circa Waves

How does it work?

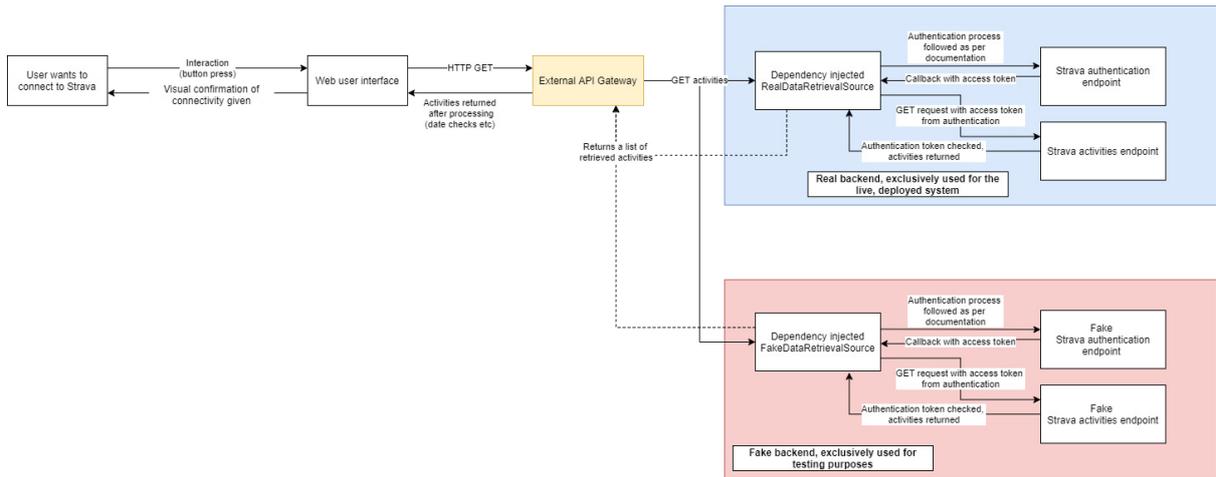
This application works by finding your fastest run from the data sources you have connected, and then displays the songs that were making you run faster during the fastest activity.

Appendix H – Web user interface

# Can Music Make You Run Faster?

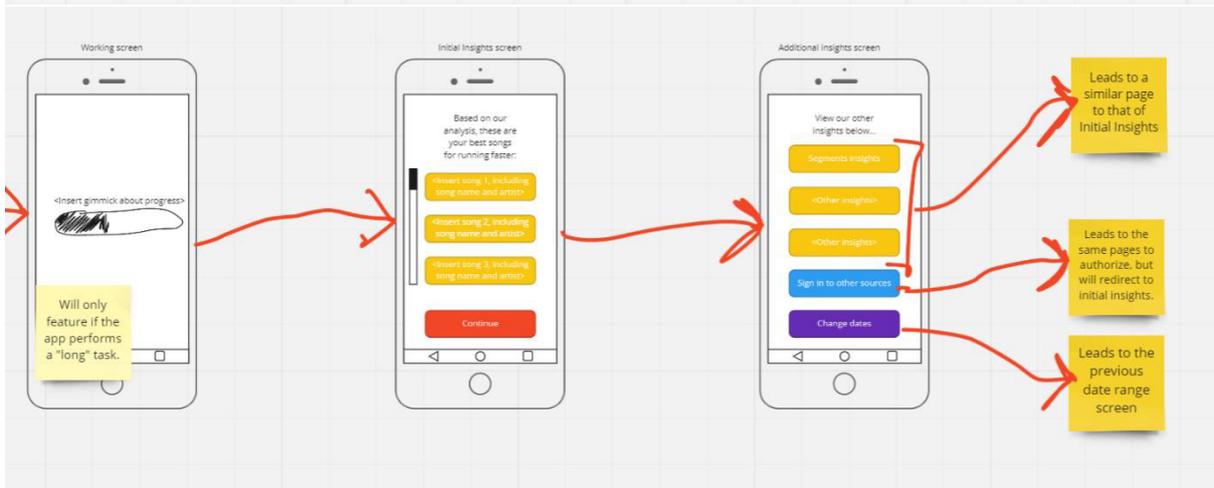
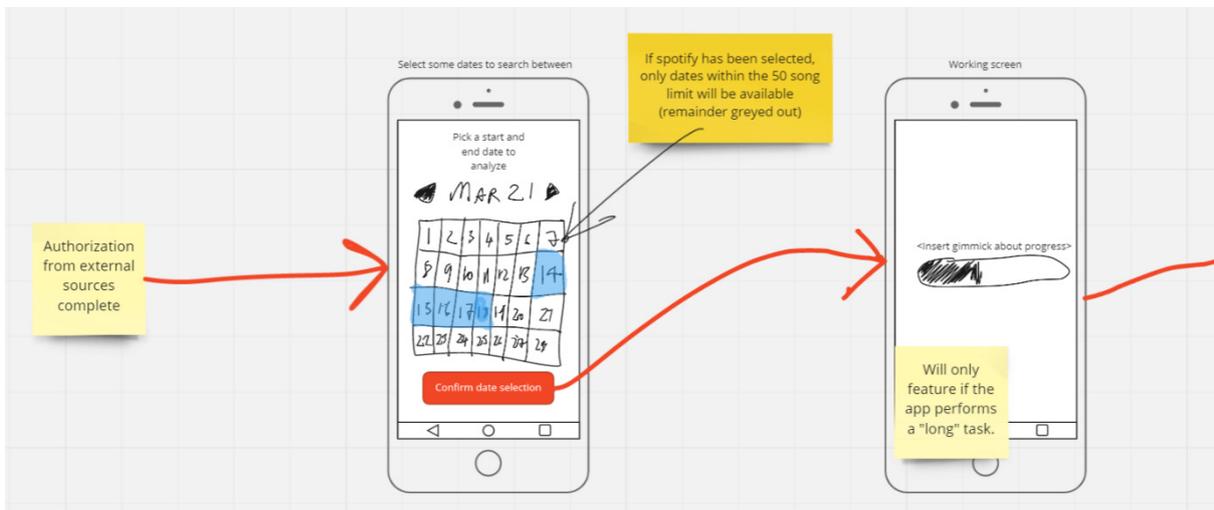
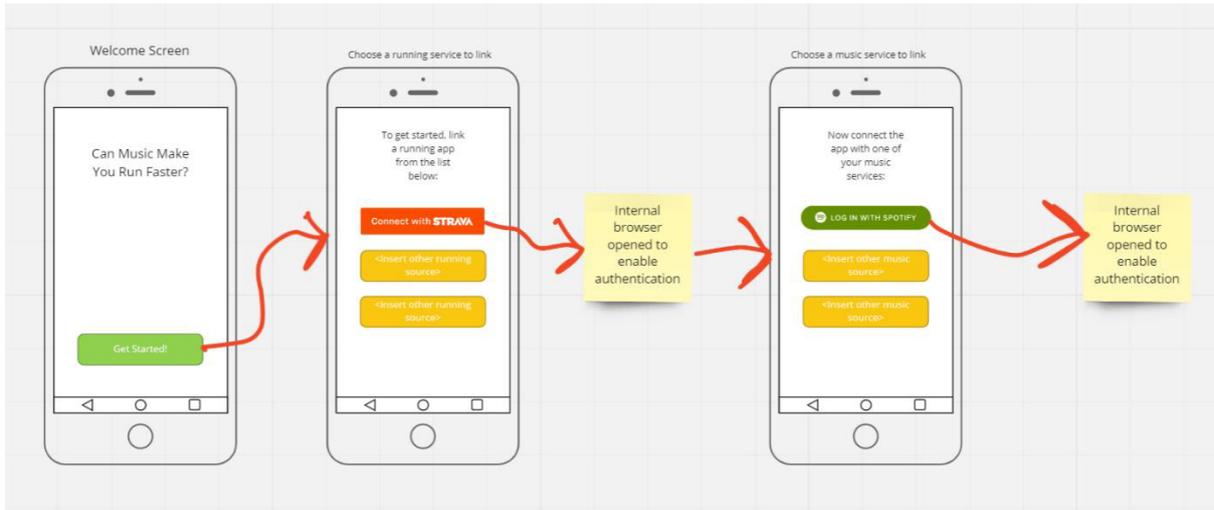


Appendix I – partial screenshot of Kanban board.



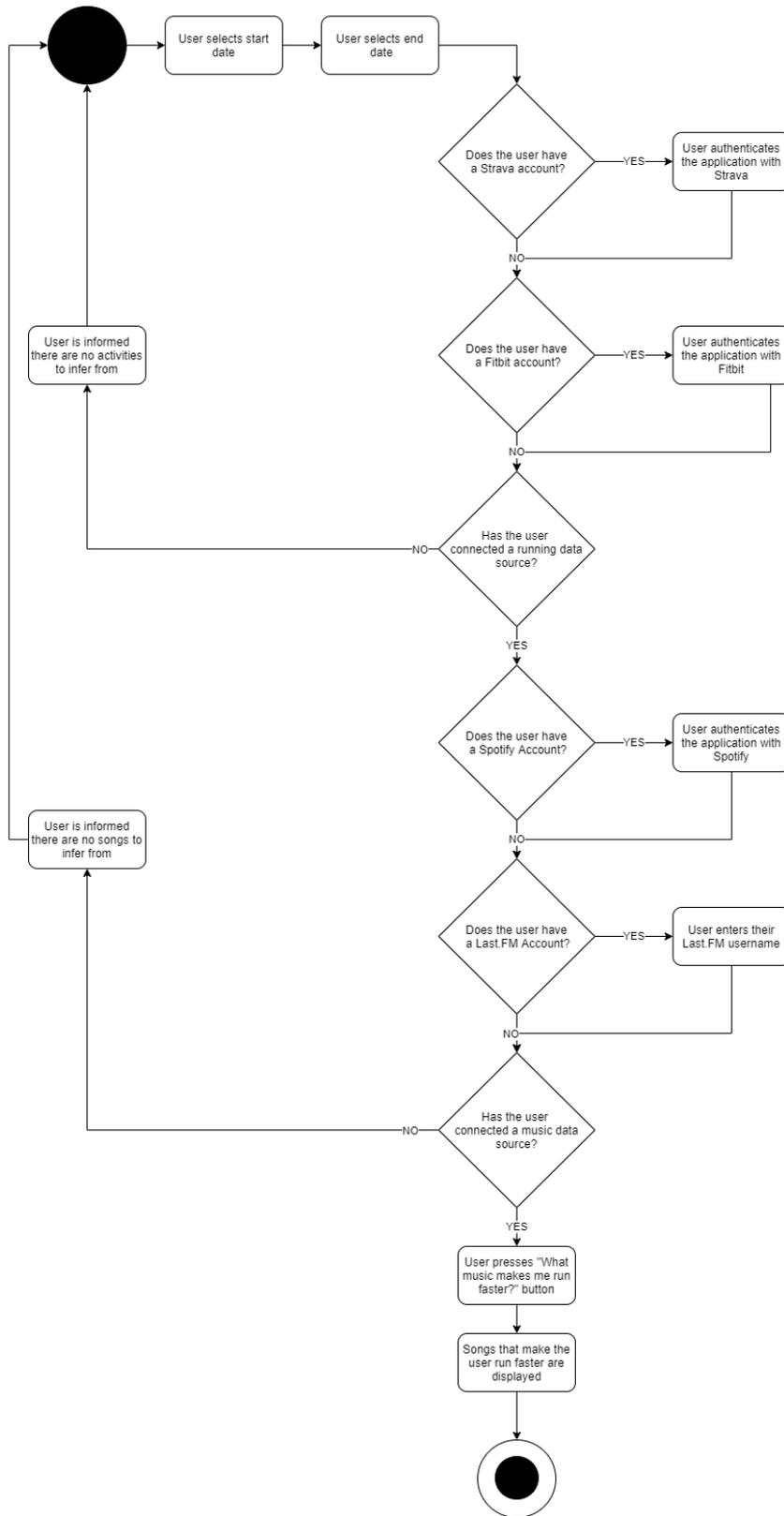
Appendix J – Example Strava data flow.

# Can Music Make You Run Faster?



Appendix K – Xamarin mock-ups.

# Can Music Make You Run Faster?



Appendix L – UML Activity diagram

## 11 REFERENCES

---

1. Bonette, R et al. 2012. The Effect of Music Listening on Running Performance and Rating of Perceived Exertion of College Students. The Sport Journal Volume 41, Issue 2. Doi: <https://thesportjournal.org/article/the-effect-of-music-listening-on-running-performance-and-rating-of-perceived-exertion-of-college-students/>
2. Runner's world. 2018. How Much Does Music Help You During a Run? Runner's world. 1 November. Available at: <https://www.runnersworld.com/training/a23471165/how-much-does-music-help-during-a-run/> [Accessed: 11 April 2021]
3. Cucumber. 2019. Tools & Techniques that elevate teams to greatness. Available at: <https://cucumber.io/> [Accessed: 22 April 2021]
4. Apple. 2020. Improve your Workout and Activity accuracy. Available at: <https://support.apple.com/en-gb/HT204516> [Accessed: 22 April 2021]
5. PractiCal. 2020. A 12-Week Study of the Accuracy of the Apple Watch's Calorie Tracking. Available at: [https://medium.com/@practical\\_app/a-12-week-study-of-the-accuracy-of-the-apple-watches-calorie-tracking-ed672cb5c333](https://medium.com/@practical_app/a-12-week-study-of-the-accuracy-of-the-apple-watches-calorie-tracking-ed672cb5c333) [Accessed: 22 April 2021]
6. Mulligan, M. 2020. Music Subscriber Market Shares Q1 2020. Available at: <https://www.midiaresearch.com/blog/music-subscriber-market-shares-q1-2020> [Accessed: 22 April 2021]
7. Zamorski, H. 2018. Spotify Running Is Gone, Check These Best Alternatives. Available at: <https://www.drmare.com/spotify-music/spotify-running-alternative.html> [Accessed: 06/05/2021]
8. Spotify. 2018. Retirement of our Running Feature. Available at: <https://community.spotify.com/t5/Content-Questions/Retirement-of-our-Running-Feature/td-p/4383603> [Accessed: 06/05/2021]
9. Independence Blue Cross. Music Apps to Help You Run Faster and Smarter. Available at: <https://www.phillymag.com/sponsor-content/music-apps-to-help-you-run-faster-and-smarter/> [Accessed: 06/05/2021]
10. Runkeeper. 2016. Introducing Runkeeper + Spotify Running! Available at: <https://runkeeper.com/cms/rkrunner-guide/use-the-app/introducing-runkeeper-spotify-running/> [Accessed: 06/05/2021]
11. Alger, K. 2021. Nike Run Club: How to use Nike's app to become a better runner. Available at: <https://www.wareable.com/running/nike-plus-run-club-guide-how-to-use-running-430> [Accessed: 06/05/2021]
12. Lobby, M. 2013. Running Technique: The importance of Cadence and Stride. Available at: <https://www.active.com/running/articles/running-technique-the-importance-of-cadence-and-stride> [Accessed: 06/05/2021]
13. Strava. 2020. Strava Segments. Available at: <https://support.strava.com/hc/en-us/articles/216918167-Strava-Segments> [Accessed: 06/05/2021]
14. Strava. 2021. Strava Authentication. Available at: <https://developers.strava.com/docs/authentication/> [Accessed: 27/05/2021]
15. Pickett, M. 2016. Running Apps for Music Lovers. Available at: <https://www.consumerreports.org/cell-phones-services/running-apps-for-music-lovers/> [Accessed: 06/05/2021]
16. Corpuz, J. 2021. Best running apps for 2021. Available at: <https://www.tomsguide.com/uk/round-up/best-running-apps> [Accessed: 06/05/2021]

17. Coleman, A et al. 2021. Fitbit.NET Api Client Library. Available at: <https://github.com/aaroncoleman/Fitbit.NET> [Accessed: 06/05/2021]
18. Dellinger, J et al. 2021. SpotifyAPI-NET. Available at: <https://github.com/JohnnyCrazy/SpotifyAPI-NET> [Accessed: 06/05/2021]
19. Inflatable Friends et al. 2020. Inflatable Last.fm .NET SDK. Available at: <https://github.com/inflatablefriends/lastfm> [Accessed: 06/05/2021]
20. Dykstra, T et al. 2020. Kestrel web server implementation in ASP.NET Core. Available at: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-5.0> [Accessed: 06/05/2021]
21. NUnit. 2019. What is NUnit? Available at: <https://nunit.org/> [Accessed: 06/05/2021]
22. SpecFlow, 2020. Behaviour Driven Development for .NET. Available at: <https://specflow.org/> [Accessed: 06/05/2021]
23. Mulligan, M. 2020. Music Subscriber Market Shares Q1 2020. Available at: <https://www.midiaresearch.com/blog/music-subscriber-market-shares-q1-2020> [Accessed: 24/05/2021]
24. Last.FM. 2020. Music counts. Track, find and rediscover music. Available at: <https://www.last.fm/about> [Accessed: 24/05/2021]
25. Nielsen, J. 1994. 10 Usability heuristics for User Interface design. Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed: 26/05/2021]
26. Douglas, R. 2021. Final Year Project Kanban Board. Available at: <https://github.com/users/TheRealDougie1/projects/4> [Accessed:26/05/2021]
27. Nair, J. 2018. TDD vs BDD – What is the difference between TDD and BDD. Available at: <https://blog.testlodge.com/tdd-vs-bdd/> [Accessed: 26/05/2021]
28. Stylecop, 2021. StyleCop Analyzers for the .NET compiler platform. Available at: <https://github.com/DotNetAnalyzers/StyleCopAnalyzers> [Accessed:27/05/2021]
29. Fitbit, 2020. Get Activity Logs List shows different data as Fitbit App. Available at: <https://community.fitbit.com/t5/Web-API-Development/GET-Activity-Log-List-shows-different-data-as-Fitbit-App/td-p/4534912> [Accessed: 27/05/2021]