CSP400 - Software Delivery Project

METHODOLOGY, REQUIREMENTS & SPECIFICATION DOCUMENT

April 7, 2018

James Watson
Student ID: 789963
Swansea University
College Of Science

Contents

Chapter 1

Introduction

The projects aim is a fully featured online music streaming service using the YouTube API for content. YouTube has a large community of amateur and professional musicians, but its focus is being a video sharing site, not a music streaming platform. The end goal of this will bring a sophisticated login system to allow users to save and seamlessly listen to different artists on the site. Right now this is only possible using the site's basic playlist feature.

Another significant portion of this project is to introduce a new approach to how displaying and shuffling artists, genres, albums and playlists is handled. A typical site will have concrete sections for all of these, but no way to mix these categories. This project's approach is to treat all of these as one single object, referred to as a *tag* throughout the project. These tags can be embedded within each other and give far more freedom and control to the user.

The site will also give users significantly more control over how shuffling is handled. A typical music site will only offer a single method of shuffling, which is either completely random or weighted to songs which are frequently listened to by the user. Using the tag system described above, users will have the option to shuffle both songs and any tag objects, as well as embed tags within tags. When tags are embedded like this, the highest level tag is chosen, and a random song from that list is chosen afterwards. This means advanced shuffling techniques such as by artist is entirely possible.

1.1 Main Project Aims

The project has three major aims in which to accomplish:

- Build a fully featured music streaming web application using the YouTube API for results.
- Introduce a new system for library and playlist management which gives increased options and control to the end user.
- Give users significantly more control and choice over song shuffling algorithms.

Chapter 2

Background

2.1 Current Applications

In order to justify the need for more robust playlist and shuffle features, research has been done on three current popular music streaming services.

2.1.1 Spotify [1]

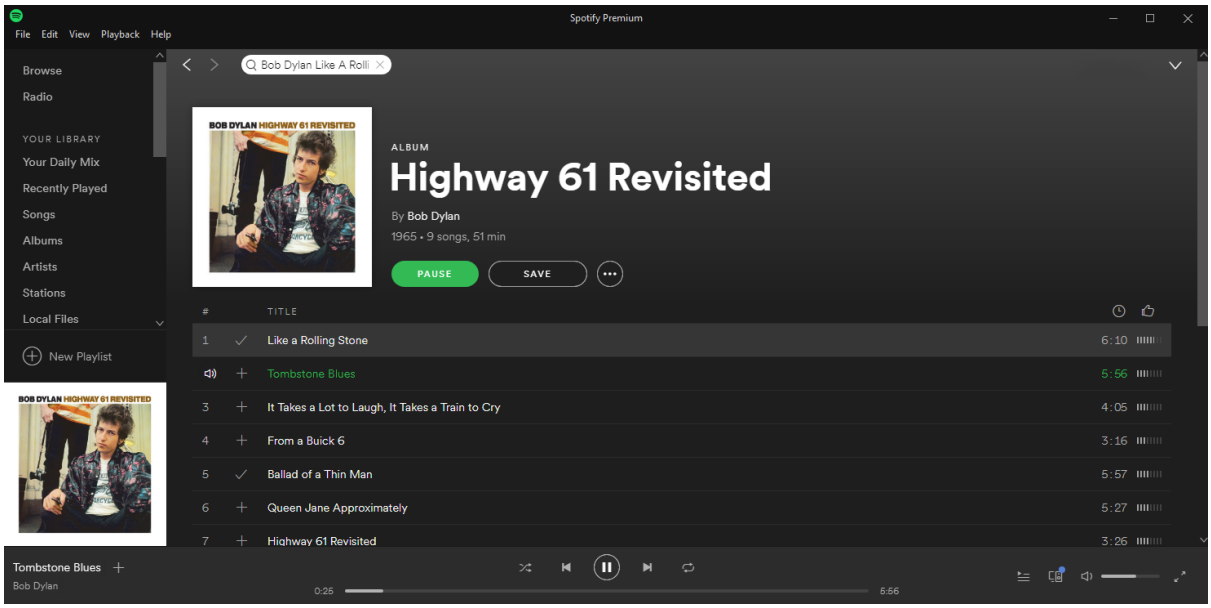

Figure 2.1: Screenshot of the Spotify desktop application as of December 2017.

Spotify offers browsing and searching for music via several parameters including artist, album, genre and playlist. This is conveniently presented on various pages, such as an artist's page. This is then presented as several sub-playlists which represent content such as the artist's albums and most popular content. It also offers the ability to make a traditional playlist from the user's song choices. While Spotify offers several pre-made playlists which are conveniently located throughout the site, it does not offer any means to create playlists of anything other than song objects. You cannot add an entire artist or album to a playlist without manually including every song into that list.

Spotify uses a weighted shuffling algorithm which aims to evenly spread out artists among the full list. This is to give the illusion of a well shuffled, random playlist [6][7]. There are no options given for choosing an alternative shuffling algorithm, or to shuffle by a specific instance, such as artist shuffle.

2.1.2 Google Play Music [4]

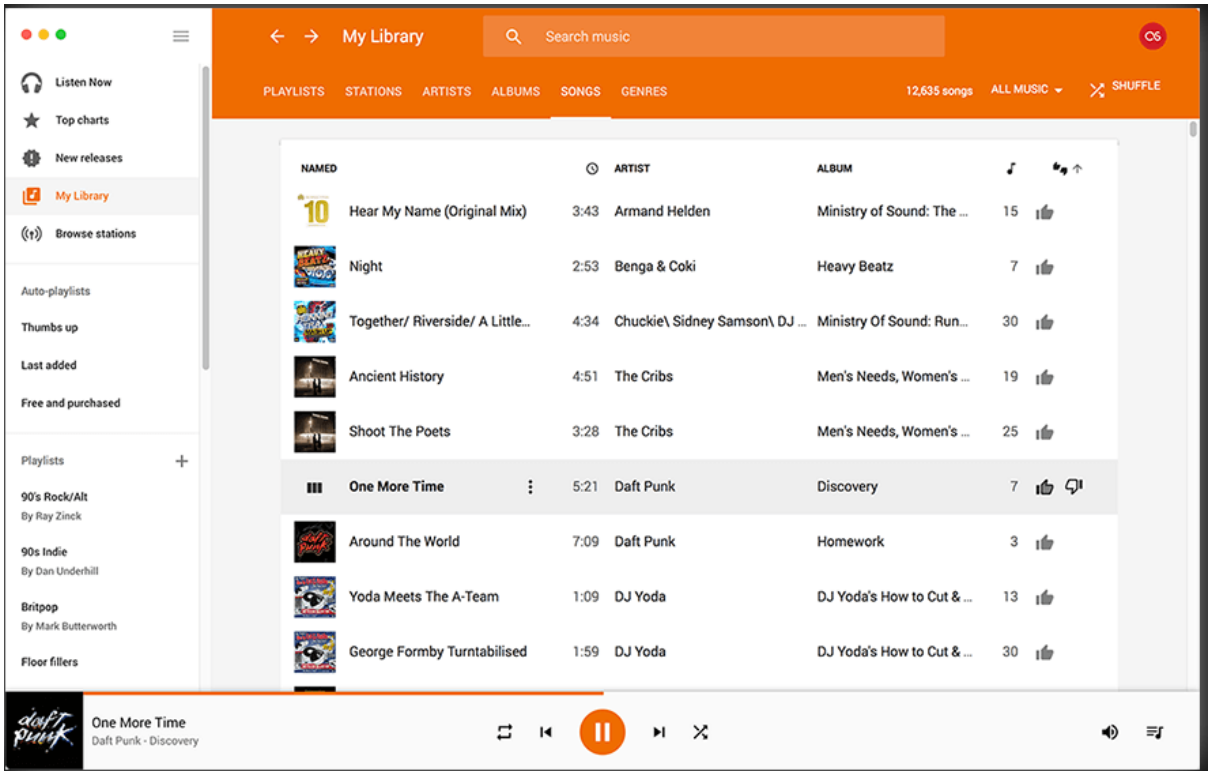

Figure 2.2: Screenshot of the Google Play Music Desktop Application [3]

Google Play Music offers a similar experience to Spotify, with a higher focus on its mobile application than desktop. Similarly to Spotify, it offers a large collection of songs organised into essentially playlists titled as artist, genres, and albums. It also does not offer the ability to create anything more than a standard playlist, and all songs from artists and albums need to be included into a playlist.

Google Play music also offers no control over shuffling algorithms. Although the exact algorithm is not public knowledge, it is likely a mostly random shuffle similar to Spotify's implementation.

2.1.3 SoundCloud [2]

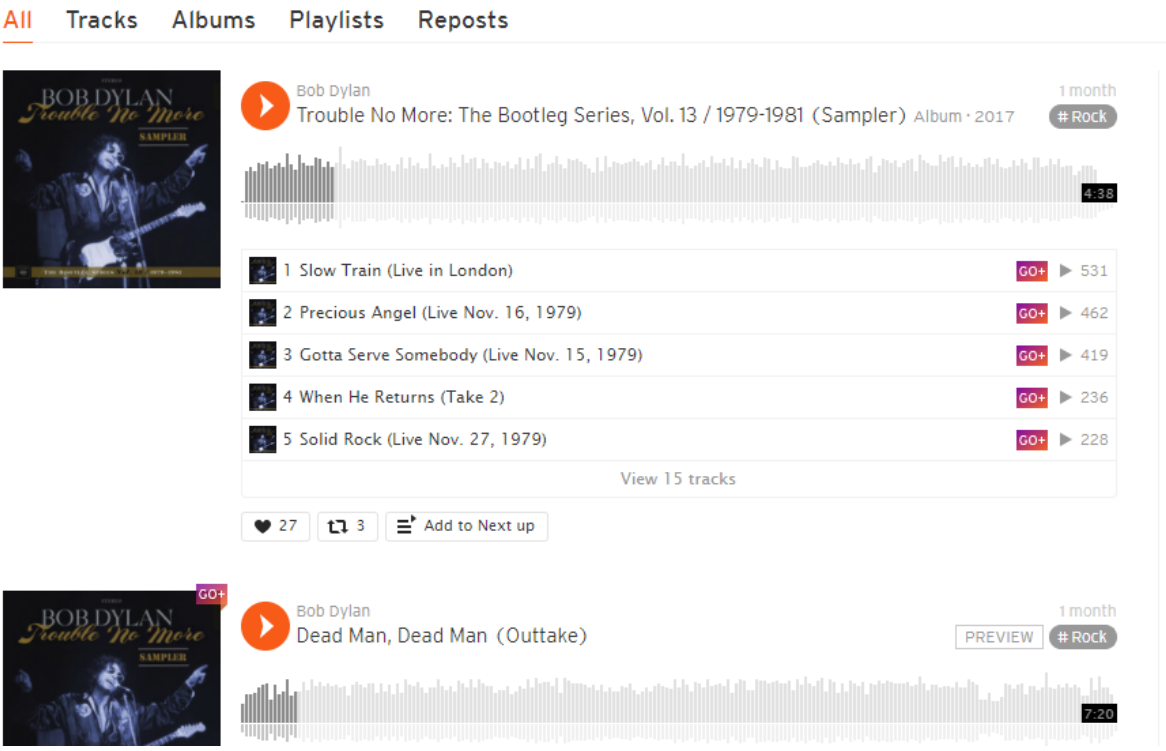

Figure 2.3: Screenshot of the SoundCloud desktop application as of December 2017.

SoundCloud has a higher focus on allowing individuals to upload original recordings than popular artists, but also offers a vast library of music from large record labels. Music can either be listened to individually, or is presented in a series of tabs on an artists page (see figure 2.3).

While SoundCloud doesn't offer any alternative shuffling algorithms also, its layout is designed such that this is often not the typical listening method. The layout encourages either discovering new tracks through related songs, or by linearly working through an artist's content.

2.2 Findings

In all three cases the user is not given control into how shuffling is handled. All three applications did not offer a means to shuffle or create playlists by anything other than individual songs. This shows a clear motivation for a music streaming application that allows for greater control.

Chapter 3

Methodology

While choosing a software methodology for this project, a number of key factors about the project were taken into consideration:

- A number of fixed deadlines are known ahead of time that cannot be changed.
- The majority of the intended features are independent and can be completed in any order.
- There is no outside client for this project, and so priorities and requirements will not change.
- The lack of client also means prototypes are not needed outside fixed deadlines, such as the interim report.

With these considerations in mind, an agile development model is the obvious choice. This allows for a highly flexible working environment where work can be done as time permits. Unexpected circumstances, such as deadlines in other modules, can also be more easily taken into consideration when fixed builds are not required on a weekly basis. Two agile methodologies were considered as being the most appropriate, along with alternatives to agile all together, discussed below.

3.1 Considered Methodologies

3.1.1 Scrum

The scrum methodology is where software gets developed in a series of *cycles*. These cycles last for a fixed period of time, typically around two weeks. These cycles are known as *sprints*. Before a sprint is begun, a sprint planning meeting is held to determine what features are to be added during the next sprint cycle. During this meeting the backlog of work for the project is examined and tasks deemed most suitable for the next build are assigned to developers. This is often chosen based off the urgency of the task, the complexity of the task, and also an estimate by the developer on how much time is needed to complete it. At the end of the sprint cycle, a review meeting is held to evaluate the success of the work done. If not all work was completed during the sprint, this is then added to the backlog to be completed in a future sprint.

Scrum offers some strong advantages to the project, namely that the project could be easily broken down into weekly or bi-weekly sprints which match to requirements. This also gives strong flexibility into workload, as each sprint can be can be increased or reduced in work size depending on outside circumstances, such as deadlines in other modules. This would give a

clear, defined schedule for the project and also provide regular builds to indicate progress. This also offers a clear time in which to merge and integration test new tasks.

This would, however, need quite drastic changes to accompany the lack of team. Scrum meetings would be replaced by a personal planning phase. Daily stand-up meetings would be replaced by a personal reflection period also.

3.1.2 Kanban

Kanban is a popular methodology used to implement a real-time overview of all tasks in a project. All tasks are represented visually on a kanban style board. This allows team members to see the exact state of every piece of work at any time. Tasks are represented as notes on a board which travel horizontally through *swim lanes*. Each swim lane divides tasks into key areas of the project vertically. Tasks then travel through a series of states. Some common states include "To Do", "In Progress", "Testing", and "Done". If a task fails any particular state, such as testing, it can be easily moved back to a previous state. This board means developers can immediately pick up tasks as soon as the previous one is finished, and allows for strong coordination between other roles in the team such as managers, testers and designers.

This methodology gives the heaviest flexibility in terms of when tasks are started and finished. Any unforeseen circumstances, such as minor illness, can be easily worked around. Tasks can also be easily published as soon as they are complete using a continuous integration workflow. There is some potential for heavy feature creep with this methodology; however, the lack of client means this is not an issue.

This methodology would also need adjusting to account for the lack of team. A typical Kanban project would generally have multiple stages such as testing, cross-checking and user-acceptance testing for each task. These tasks will all need to be done by the sole developer where applicable. Testing can easily be done as a step within the Kanban flow, cross-checking can be replaced by a thorough check of code before committing to source control. As there is no customer in this case, user-acceptance is not applicable.

3.1.3 Waterfall

The waterfall methodology is a model which closely follows a typical life-cycle pattern. This comes in many different variations, but most commonly has around 6 major phases:

- Requirements and specification.
- Design.
- Implementation.
- Testing.
- Deployment.
- Maintenance.

Software made with this approach is developed sequentially through each of these phases. For the project to reach the next phase, the previous phase must be fully complete. The logic behind this is that phases are never revisited. This forces requirements to stay concrete and

not change throughout development. The term waterfall comes from progress flowing in one direction downwards like a waterfall through each phase of development.

While this methodology fits well with the projects fixed deadlines and knowledge that requirements won't change, it adds some unnecessary restrictions on the work schedule. Being forced into developing features sequentially removes the projects strongest advantage, in which features can be developed independently. It also reduces the flexibility of reducing or increasing workload depending on outside circumstances. This methodology also poses a difficult challenge in that progress is difficult to demonstrate for the interim report.

3.1.4 Spiral

The spiral methodology is an iterative approach which puts high emphasis on risk assessment and management. Development is done through a series of prototypes, each prototype being based on the outcome of the previous iteration. Each phase is made up of four stages, which loosely follow the same flow as the waterfall methodology. In most cases this model will be iterated over multiple times as is necessary to reach the initial design. Each iteration adds a new spiral to the process and produces a prototype which is closer to the design than the previous. Each prototype can be easily given to a client for feedback.

Although this approach improves upon some of the issues with the waterfall methodology, its biggest advantage, a high emphasis on risk analysis, is not useful to a non-critical system such as a music streaming service. The concept of regular deployable builds is however a useful concept that does translate well to the project.

3.2 Chosen Methodology

With all these methodologies considered, kanban has been chosen as the methodology of choice. This methodology gives by far the most flexible schedule of the discussed methodologies, and allows features to be easily integrated over the duration of the project. Given that the project is a web based project, this also allows the possibility of additional features to be added in future. I also have previous experience working with this methodology as part of a team, and so is a strong personal preference. As discussed earlier, some aspects of this methodology will need to be altered to account for the lack of team. Each stage, such as testing and code checking, will need to be done by the sole developer.

Seven stages in the Kanban workflow will be used, as demonstrated in figure 3.1. With this structure, tasks can be freely created and worked on without restrictions. Swim lanes can also be used to visually group different task types. Each task will follow a strict flow that ensures time is devoted to documentation, testing, and integration. This means that all code is appropriately written in preparation for the design and testing documents in milestone 3.

3.2.1 Version Control

To support this methodology, the git version control system will be used to handle merging and separating code. The *git flow* workflow can be utilised here in order to allow tasks to be tested and built in isolation from other tasks. A master branch will be present which holds the current live version of the code. A develop branch is then made off of this which is used to allow multiple merged tasks to be integration and end-to-end tested together before being committed

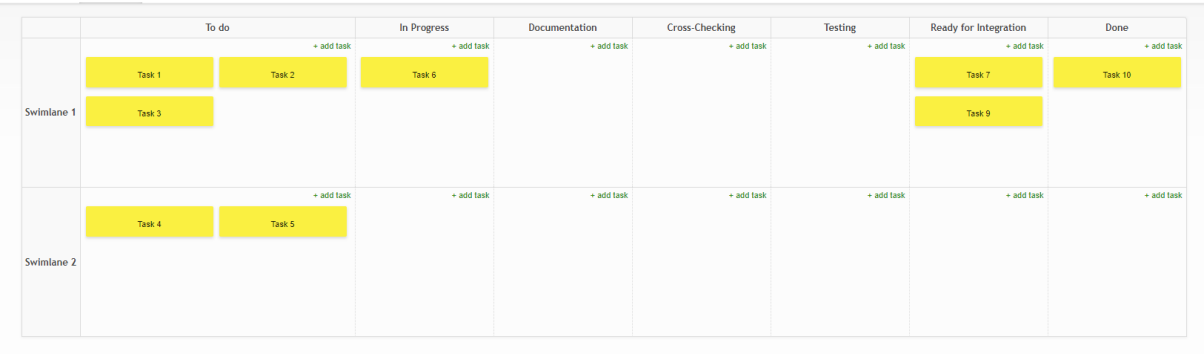

Figure 3.1: Mock-up of the suggested Kanban board structure for the project.

for publishing.

Each task will be given a dedicated feature branch which is branched off of this develop branch. This allows each new task to be tested and worked on in isolation, independent of the state of any other task. *Hotfix* branches can also be created for any bugs which are found after merging.

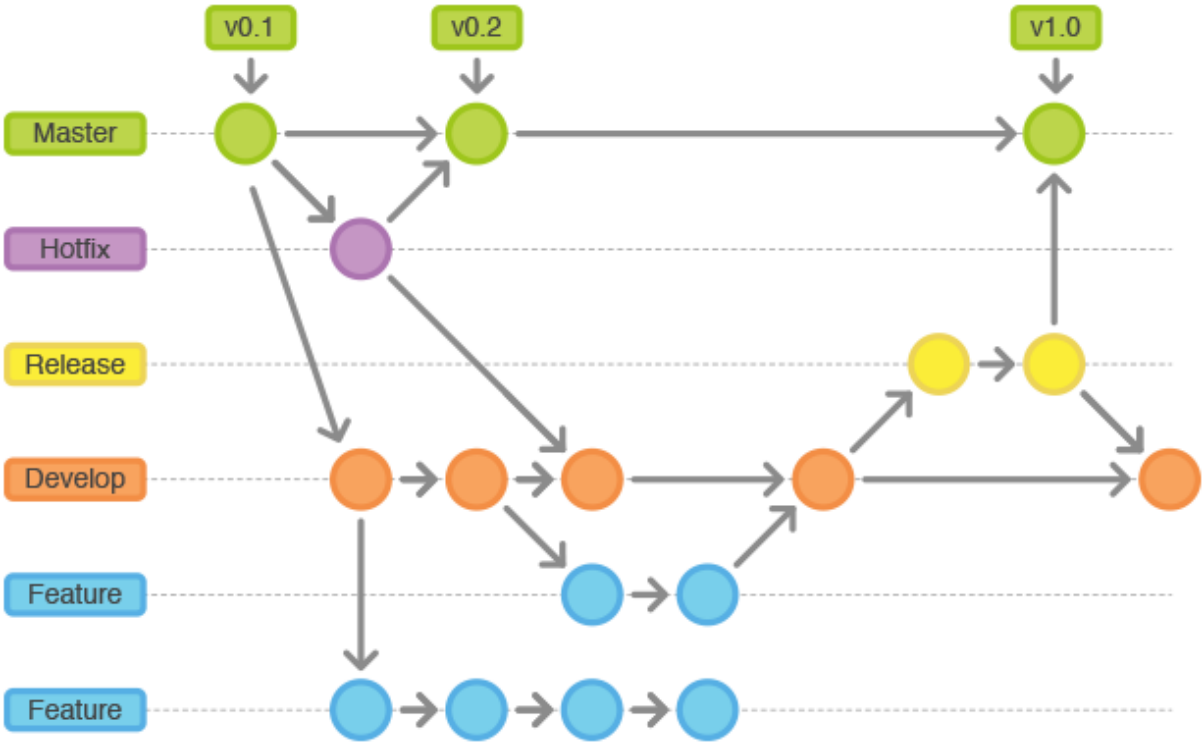

Figure 3.2: A visual representation of the git flow workflow by leanpub.com [5]

Chapter 4

Testing

As mentioned in the previous chapter, the chosen methodology is kanban. This gives testing within the project a clear step in each task's lifecycle.

4.1 Unit Testing

Before being made ready for integration, all relevant methods will firstly be unit tested. Testing in this step will be performed using white-box testing. This is where the system under test is observable to the tester, and so the expected output for a given input is known and can be tested for.

Unit testing is performed on individual units that make up a system. For this project, a unit refers to an *Angular component* within the system. An Angular component is an isolated feature of an Angular project which features CSS styles, HTML, and scripts which are locally scoped to that feature. If implemented properly, this can be easily reused throughout the project. For the back-end implementation, the public facing methods will also be unit tested for correct input.

4.2 Integration Testing

Before any feature can be integrated into the live environment, it must first be merged and integrated to the develop branch to compare that there are no conflicts with other features to be published. Integration testing will be performed on all new and current features in the development branch before publishing to the live environment.

4.3 End-To-End Testing

The final testing within the pipeline involves performing regular *end-to-end testing* on the full system. End-to-end tests explore the application as the end user would experience it. In this type of testing, one process runs the real application while a second process runs a test simulator that behaves in the same way a user would. It can then assert that the running application responds as expected.

Chapter 5

Requirements

5.1 Requirements

Functional Requirements

Table 5.1: Miscellaneous Requirements

Requirement Id	Requirement
MISC-REQ1	The application must allow users to log in to the system using their account.
MISC-REQ2	The application must have the ability to search through a large catalogue of music choices.
MISC-REQ3	The application must allow customers to stream chosen videos from within the site.
MISC-REQ4	The application must allow customers to save music choices to their personal library.

Table 5.2: Playlist and Shuffling Requirements

Requirement Id	Requirement
PS-REQ1	Customers must be able to create tags to organise chosen music choices from their library.
PS-REQ2	Tags must be able to be marked as either a playlist, genre, artist, or album.
PS-REQ3	Tags must be able to be nested within each other, such that shuffling is possible with nested tags.
PS-REQ4	Customers must be able to shuffle tags either completely randomly, based on listen frequency, or by a list of percentages defined by the customer.
PS-REQ5	Customers should be able to mark tags into different categories which can be used for weighted shuffling.
PS-REQ6	Customers should be able to remove songs and tags from their collection.
PS-REQ7	Customers should be able to modify the details of songs and tags from their collection.

Non-Functional Requirements

Table 5.3: Non-Functional Requirements

Requirement Id	**Requirement**
NF-REQ1	The application must be accessible to the world wide web.
NF-REQ2	The application must be compatible with all major *evergreen* browsers. (Browsers that automatically update themselves).
NF-REQ3	The application must be a progressive web application using a mobile-first approach.
NF-REQ4	The application must feature a simple user interface with a modern look and feel.
NF-REQ5	The application should be a Single-Page Application.
NF-REQ6	Customer details should be safely secured using a trusted security method.
NF-REQ7	The website should be protected against common security vulnerabilities such as Cross-Site Scripting, Man-In-The-Middle Attack, SQL Injection and Cross-Site Request Forgery.

Chapter 6

Specification

6.1 Specifications

Table 6.1: Web Application Specification

Specification Id	Specification
WEB-SPEC1	ASP.NET Core will be used for the back-end of the application.
WEB-SPEC2	Angular will be used as the front-end framework of the application.
WEB-SPEC3	Bootstrap will be used to simplify the styling and provide features such as a mobile first workflow.
WEB-SPEC4	The YouTube search API will be used to return streaming results.
WEB-SPEC5	Found videos will be embedded into the page using HTML5.
WEB-SPEC6	Details about chosen songs will be stored against a user's account.
WEB-SPEC7	Media queries will be used to hide less important information on smaller screen widths.

Table 6.2: Shuffle Algorithm Specification

Specification Id	Specification
SH-SPEC1	An algorithm for shuffling songs completely randomly will be produced.
SH-SPEC2	The frequency in which songs and tags are listened to should be kept track of.
SH-SPEC3	An algorithm for shuffling based on the user's listen frequency to each song will be produced.
SH-SPEC4	An algorithm for shuffling based on the user's specified frequency for each song will be produced.

Table 6.3: Playlist Specification

Specification Id	**Specification**
PL-SPEC1	A button to save results returned from searching will be present to save results to the user's library.
PL-SPEC2	Buttons should be next to songs and tags to modify or delete their content.
PL-SPEC3	Songs and tags should be able to be dragged such that they can be placed inside or outside tags.
PL-SPEC4	A button to create and modify tags should be present.
PL-SPEC5	A dropdown list should be tied to tags with options for marking as either playlist, genre, artist, or album.
PL-SPEC6	A field will be present on songs and tags which can be used to indicate preferred listen frequency when shuffling.

Table 6.4: Database, User Authentication And Security Specification

Specification Id	**Specification**
DATA-SPEC1	The following details will be captured for a user at minimum: Username, Full Name, Password (hashed).
DATA-SPEC2	ASP.NET Identity will be used to safely handle account details. This includes hashing passwords and passing it to the database.
DATA-SPEC3	Users will be able to sign up to the application using a unique username and password.
DATA-SPEC4	Users will be able to log in using their username and password.
DATA-SPEC5	An SSL/TLS certificate will be used to provide HTTPS.
DATA-SPEC6	All pages will be served over HTTPS. HTTP will not be allowed and be redirected to HTTPS.
DATA-SPEC7	Requests will be protected from Cross-Site Request Forgery using Anti-Forgery Tokens.
DATA-SPEC8	All input will be sanitized before being secured in the database. This will be done on both the front-end and back-end.

Table 6.5: Testing And Documentation Specification

Specification Id	**Specification**
TEST-SPEC1	All features will be tested using Chrome, Microsoft Edge, Firefox, Safari and mobile equivalents before merging.
TEST-SPEC2	End-to-end testing will be done on a regular basis where applicable.
TEST-SPEC3	Each page will be tested on all major screen widths.
TEST-SPEC4	Unit testing will be carried out where applicable.
TEST-SPEC5	All functions will be documented where applicable.

6.2 Requirements/Specification Cross-Referencing

To ensure that each requirement is met, each requirement is cross-referenced with one or more specifications to ensure all required functionality will be implemented.

Table 6.6: Miscellaneous Requirements Cross-Referencing

Requirement ID	**Specification IDs**
MISC-REQ1	DATA-SPEC1, DATA-SPEC2, DATA-SPEC3, DATA-SPEC4
MISC-REQ2	WEB-SPEC4
MISC-REQ3	WEB-SPEC4, WEB-SPEC5
MISC-REQ4	WEB-SPEC6, PL-SPEC1

Table 6.7: Playlist And Shuffling Requirements Cross-Referencing

Requirement ID	**Specification IDs**
PS-REQ1	PL-SPEC4
PS-REQ2	PL-SPEC5
PS-REQ3	PL-SPEC3
PS-REQ4	SH-SPEC1, SH-SPEC2, SH-SPEC3, SH-SPEC4, PL-SPEC6
PS-REQ5	SH-SPEC2, PL-SPEC4
PS-REQ6	PL-SPEC2
PS-REQ7	PL-SPEC2

Table 6.8: Non-Functional Requirements Cross-Referencing

Requirement ID	**Specification IDs**
NF-REQ1	WEB-SPEC1, WEB-SPEC2
NF-REQ2	WEB-SPEC2, WEB-SPEC3, WEB-SPEC7, TEST-SPEC1, TEST-SPEC2, TEST-SPEC3, TEST-SPEC4
NF-REQ3	WEB-SPEC3, WEB-SPEC7, TEST-SPEC1, TEST-SPEC3
NF-REQ4	WEB-SPEC3
NF-REQ5	WEB-SPEC2
NF-REQ6	DATA-SPEC1, DATA-SPEC2, DATA-SPEC4, DATA-SPEC5, DATA-SPEC6, DATA-SPEC7, DATA-SPEC8
NF-REQ7	DATA-SPEC5, DATA-SPEC6, DATA-SPEC7, DATA-SPEC8

Chapter 7

Risk Analysis

Risk management is an important aspect of a successful project. To identify and assess factors that may jeopardise the full completion of the project, a risk assessment has been undertaken. The table below identifies risks for this project. In this table a quick overview of each hazard, the potential consequences, its likelihood, and the proposed control measurements are all given.

Risk Analysis Table			
ID	**Risk**	**Likelihood**	**Danger Rating**
1	**Data Loss/Corruption** A large data loss would mean starting the project from scratch, potentially making finishing by the deadline impossible. **Contingency Plan** To prevent this from happening, all files involved will be routinely backed up in 2 separate locations. Source control will also be used to ensure a copy of the project is always available and up to date.	Low	High
2	**Short Term Illness** In the event of a short term illness, such as a cold or fever, the projects development could be held back by up to a week. **Contingency Plan** The flexibility of the chosen methodology, kanban, means that workload can be easily adjusted to be lightened or increased depending on illness or deadlines outside the project.	High	Low
3	**Feature Creep** Given the nature of the project, it is possible to easily extend the project with an abundance of features. **Contingency Plan** By adhering to a strict set of requirements and specifications outlined in this document, any risk of feature creep is avoided.	Low	Medium

Risk Analysis Table			
ID	**Risk**	**Likelihood**	**Danger Rating**
4	**Being Unfamiliar With Chosen Technologies** The project will use multiple technologies that are new to the developer. This can mean some incorrect practices may be followed at the earlier stages of production which will need correcting later. **Contingency Plan** This risk can firstly be avoided by spending an adequate amount of time researching the chosen technologies documentations. The kanban methodology will also allow flexibility in having dedicated tasks to correct this later in the project.	Medium	Medium
5	**Final Product Does Not Meet The Specification** Once the final application has been developed it will potentially not meet the requirements and specification outlined. **Contingency Plan** By strictly following the work schedule, methodology, and project plan discussed in this document, there should be no issue.	Low	Medium
6	**Poor Coding Conventions And Best Practices** Having inconsistent coding standards throughout the project would significantly slow down progress and increase the likelihood of introducing bugs. **Contingency Plan** By following best practices outlined by each technology used, and by using strict naming conventions, the risk of cluttering the project will be significantly reduced.	Medium	Low
7	**Failure to document methods and classes** It is likely during the project a portion of classes and methods will fail to have documentation written. This could either be due to forgetfulness or time constraints. **Contingency Plan** A step in the Kanban workflow will be added where all tasks should be double checked and reviewed before merging.	High	Low

Chapter 8

Project Plan

In this chapter, a clear project plan is presented which outlines how time is spent during the projects development. As discussed in chapter 3, the project will follow the kanban methodology. Because of this, the work schedule has been kept flexible in terms of which tasks are done when in order to more closely follow this methodology. While most tasks can be done in any order, three major milestones are given below, which gives a clear deadline for when certain tasks should be undertaken. This is reflected in the gantt chart presented later in this chapter.

8.1 Milestones

Milestone	**Delivarables**	**Due Date**
Milestone 1	Methodology and Requirements Document Specification Document	8th December 2017
Milestone 2	Interim Report	16th February 2018
Milestone 3	Project Demonstration Fair User Manual Design Document Testing Document Narrative and Reflective Account	11th May 2018

8.2 Gantt Chart

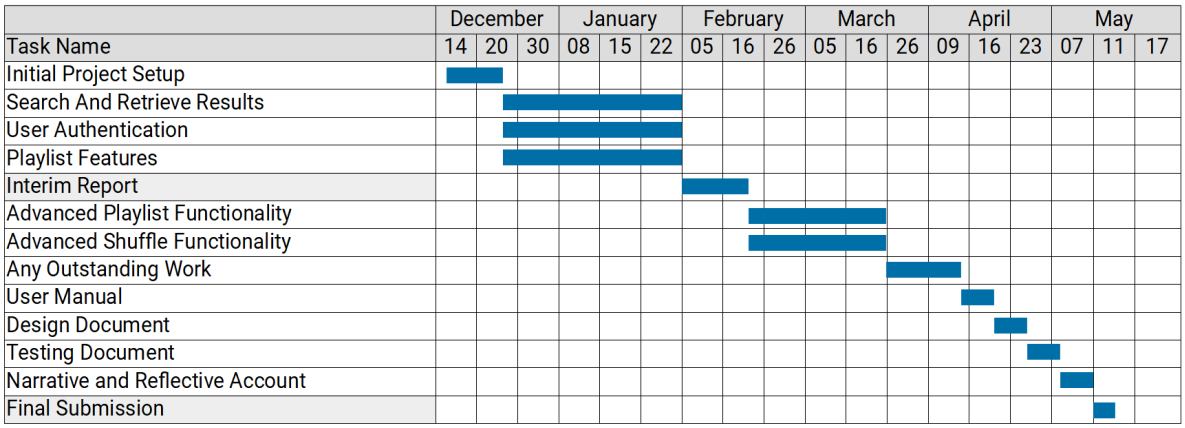

Figure 8.1: Gantt chart showing the work schedule for the project.

As previously mentioned, the majority of work to be done has been kept highly flexible in the order in which it is developed. With the exception of work related to milestones and initial setup, all other work is developed in parallel. As an example, the user authentication, playlist features, and search results are all to be developed in the same time period. This supports the flexible nature of the chosen kanban methodology.

In order to ensure time is given for reflection, a one week grace period is given before both major milestones. As shown in figure 8.1, all tasks are scheduled to be complete one week before their respective deadline. A dedicated period of 3 weeks has also been given to complete all outstanding tasks for the project. As discussed in the risk analysis report in chapter 7, a possible risk involved could be minor illness, setting development back up to a week. This one week grace period and outstanding work period doubles up as an excellent safety net for this circumstance.

Bibliography

[1] AB, Spotify. *Spotify Music Streaming Service*. 2017. URL: https://www.spotify.com.

[2] Alexander Ljung, Eric Wahlforss. *SoundCloud Music Streaming Platform*. URL: https://soundcloud.com/ (visited on 12/09/2017).

[3] Anwar, Sajid. *Screenshot of the Google Play Desktop Application for Radiant Player*. URL: http://radiant-player.github.io/radiant-player-mac/ (visited on 12/09/2017).

[4] Google. *Google Play Music*. URL: https://play.google.com/music (visited on 12/09/2017).

[5] Leanpub. *Git Flow Workflow*. URL: https://leanpub.com/git-flow/read (visited on 12/10/2017).

[6] Lee, Dave. *How random is random on your music player?* Feb. 2015. URL: http://www.bbc.co.uk/news/technology-31302312.

[7] Moskovitch, Greg. *You're Not Crazy, Spotify's Shuffle Isn't Actually Random*. Feb. 2015. URL: http://tonedeaf.com.au/youre-not-crazy-spotifys-shuffle-isnt-actually-random/.