

CSP400 - SOFTWARE DELIVERY PROJECT

BUILDING A FULL-SCALE MUSIC STREAMING WEB APPLICATION - INTERIM REPORT

April 11, 2018

James Watson
Student ID: 789963
Swansea University
College Of Science

Contents

1	Introduction	3
1.1	Project Overview	3
2	Progress	4
2.1	Overview	4
2.2	Designs	4
2.2.1	Design Screenshots	4
2.3	Running Build	6
2.3.1	Search Results	6
2.3.2	Video Playback	8
2.3.3	Redux	9
2.4	Current System	10
2.4.1	Methodology	10
2.4.2	Deployment	11
2.5	Authentication	13
2.6	Testing	14
2.6.1	Angular Component Unit Testing	14
3	Progress Assessment	16
3.1	Requirements And Specification Progress	16
3.2	Time Plan Progress	19
3.3	Personal Progress Thoughts	19
3.4	Final Progress Assessment	20
4	Risk Analysis	21
4.1	Overview	21
4.2	Encountered Risks	21
4.3	Risks Encountered Outside Of Risk Analysis	22
4.4	Other Risks	22
4.5	Updated Risks Table	23
4.6	Technical Risks	24
4.7	Non-Technical Risks	26
4.8	Risks in Descending Order	27
5	Updated Time Plan	28
5.1	Issues With The Original Time Plan	28
5.2	Updated Gantt Chart	29

Chapter 1

Introduction

The aim of this document is to report on the progress that has been made on the project since the submission of the *Methodology, Requirements And Specification* document submitted in December 2017.

To analyse the progress made so far, several examples of how progress has been made is presented. This is followed by a detailed progress assessment. Finally, an updated risk assessment and time plan have been produced to include newly identified risks and more accurately plan out the remainder of the project.

1.1 Project Overview

The aim of this project is to produce 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.

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.

Chapter 2

Progress

2.1 Overview

In order to demonstrate the progress this project has made since the initial document, this chapter includes many examples of work produced so far, challenges faced during development, and design screenshots from the current build.

2.2 Designs

Early on in development, designs for the overall shared page layout were created. The goal of these designs was to give a consistent look and feel to the website, and also provide a common shared component for new pages to be quickly designed from. This design has been created for three major screen sizes: mobile, tablet, and desktop. One of the major considerations for the design was that a typical music streaming site shares many common design choices with a typical administrator control panel, commonly found in most content management systems. After careful research, I made the decision to use a popular CSS template known as CoreUI [10], built on top of *Bootstrap* [4], the project's chosen CSS Framework. This template was perfect as it has both strong *Angular* support, (the project's chosen front-end framework), and fits the original specification of using Bootstrap to give a modern look and feel. It was also possible to easily remove the majority of features that were not relevant to this project and simply use it for a solid starting template.

2.2.1 Design Screenshots

Below are the original website designs for the three mentioned screen sizes, using the first designs for the search feature as an example page. Some of the most prominent features include a collapsible side menu, a scaling content area, and a *breadcrumb trail* navigation bar. Some of the information on this page has still been left with placeholder information for the time being, such as the website name and logo.

Figure 2.1: The website's design for mobile devices.

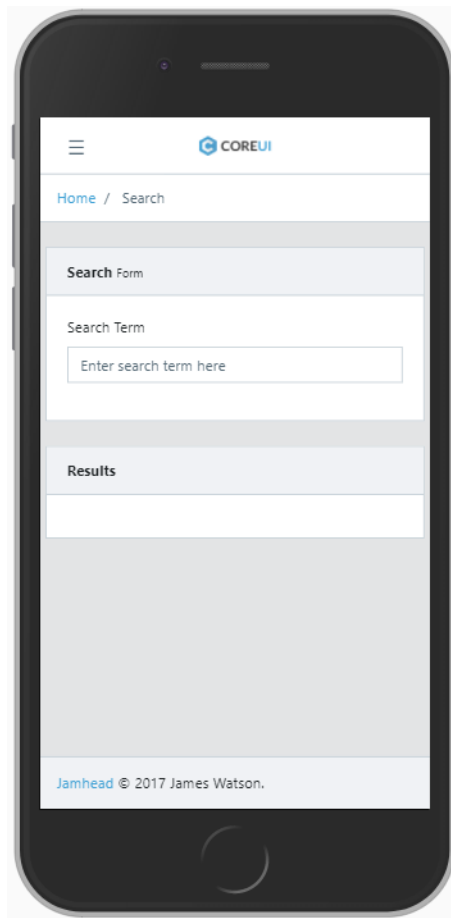


Figure 2.2: The website's design for tablet devices.

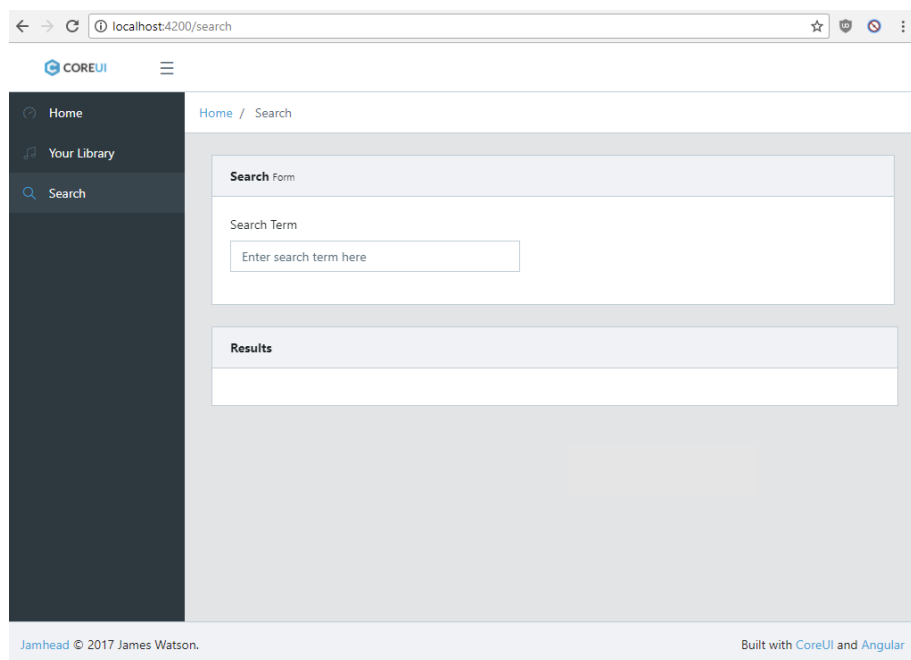
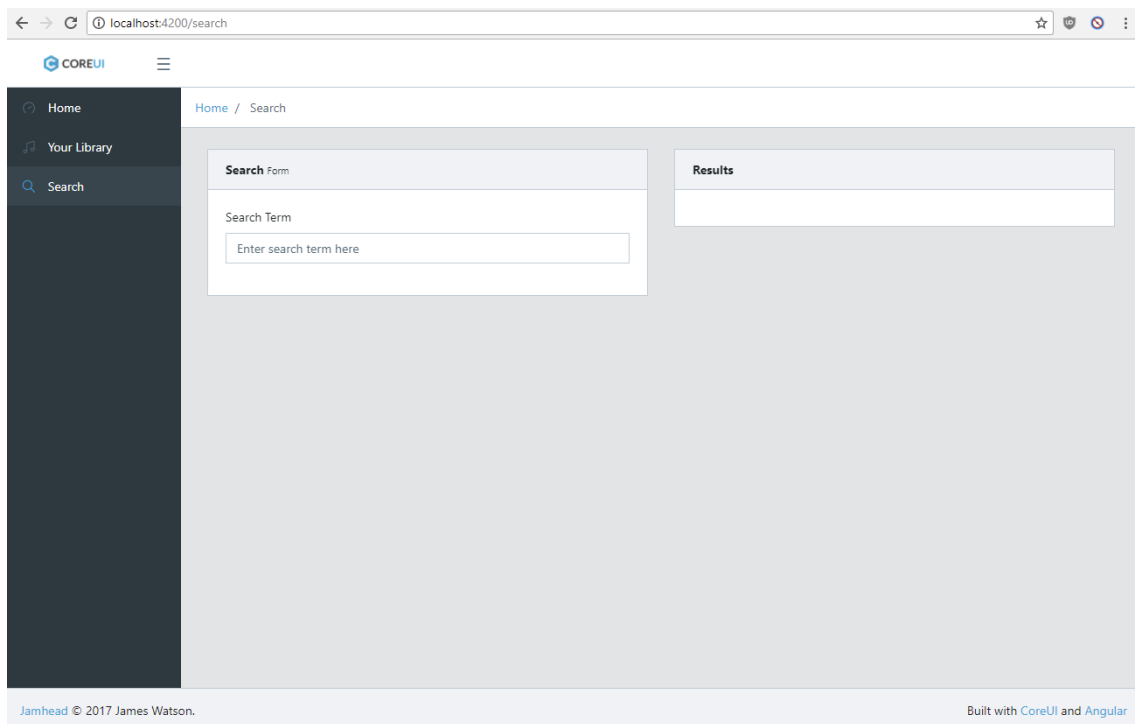


Figure 2.3: The website’s design for desktop devices.



2.3 Running Build

A running build has been created and regularly extended since very early in the project. This was originally ran using the Angular CLI, and eventually extended to be using the ASP.NET Core back-end framework once the back-end design had been implemented.

2.3.1 Search Results

The search result component has made significant progress. The design has been split into two sub-components. The first is a simple HTML form which is asynchronously linked to the YouTube search API. This is handled in Angular using an *observable*, (essentially a library for simplifying asynchronous API callbacks).

The second sub-component on the right displays the results from an array of search result objects. All the basic information like the thumbnail image URL, title, description, and YouTube link are stored here.

The search page has been made fully responsive, with the search form and results displayed in 2 rows on desktop, and condensed to 1 on smaller screen widths. The “Play” and “Add To Library” buttons are also changed to icons to give a cleaner and more condensed look on mobile.

Figure 2.4: Search results on desktop, as seen on the current build.

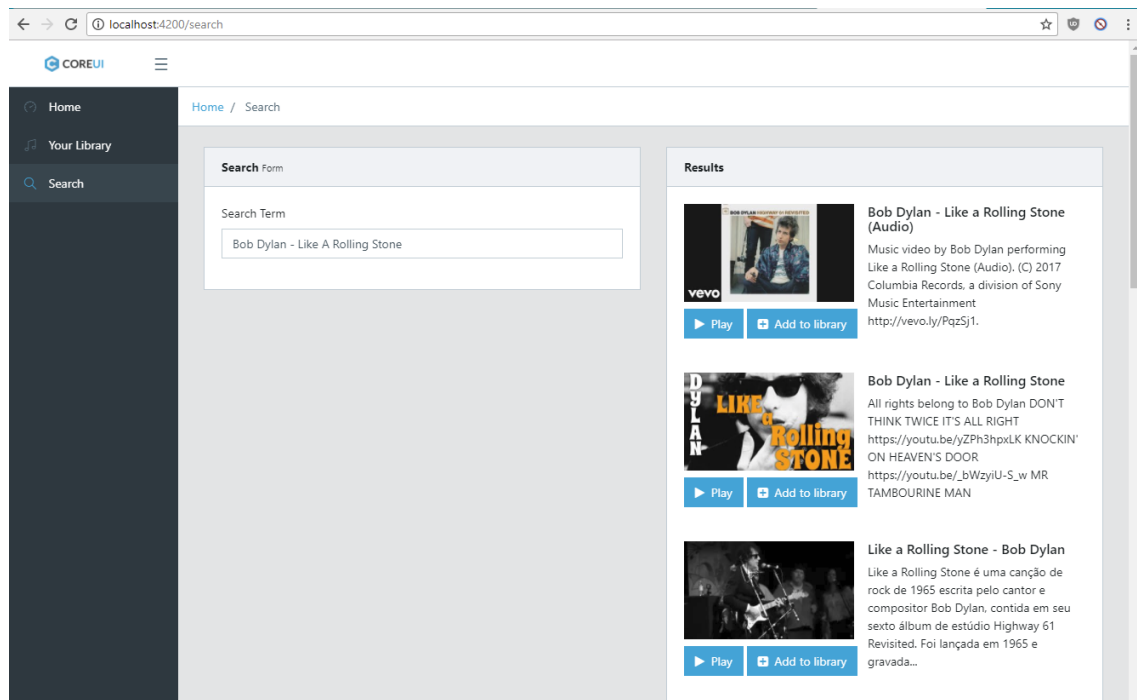
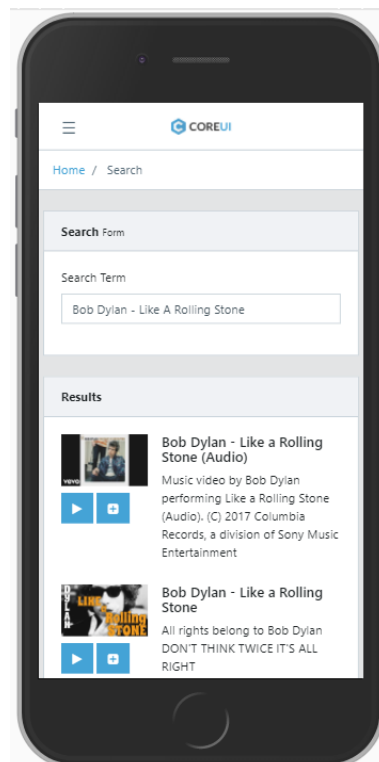


Figure 2.5: Search results on mobile, as seen on the current build.



Search Results Challenges

The biggest challenges faced with the search result component was the complex differences between desktop and mobile. A large amount of information needed to be condensed down to a small screen, including a thumbnail, description and buttons for playing and saving the result.

To solve this, I took advantage of the grid system present in Bootstrap. As an example, the following code could be used to use 6 columns of space on small devices, and 4 columns on medium width devices or higher:

```
1 <div class="row">
2   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
3   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
4   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
5 </div>
```

This was a good first step, but still did not solve the problem of not being able to condense all the information present to such a small screen. I ultimately took a number of steps to arrive at the design in figure 2.5:

- Store a copy of a lower resolution thumbnail image from the returned JSON provided by the YouTube API.
- Remove the text from buttons and simply rely on the icons to communicate the action.
- Reduced the maximum vertical length of the search result.

These changes now gave enough room for the content to display, but in practice was very challenging to implement. I ultimately did not find a 100% clean solution for this. I instead opted to render both the desktop and mobile thumbnails and icons and simply use further CSS to hide irrelevant content for the given resolution.

```
1 <div class="mb-3 d-block d-sm-none">
2   <button class="btn d-sm-none" type="button">
3 </div>
```

In the code above, “d-block” will show the button on extra-small screen sizes, and then “d-sm-none” will override this and hide it on all other screen sizes. This approach was used multiple times to hide different parts of the search-results for mobile and desktop. In testing this showed no sign of sluggish performance or being error prone on any tested browsers, and so I considered this an adequate solution to a very complex layout challenge.

2.3.2 Video Playback

The video playback component has also made significant progress. In order to prominently display embedded results, the HTML iframe is “stickied” to the bottom of the screen, and centered. This scales excellently with different screen sizes, see figure 2.7.

As a minimum viable product, this component is mostly complete. One small caveat of this is the inability to hide the iframe. This can be a small issue when search results are hidden behind the iframe and aren’t clickable.

Figure 2.6: Video display on desktop, as seen on the running prototype. An official upload of Bob Dylan’s “Like a Rolling Stone” is used as an example [1].

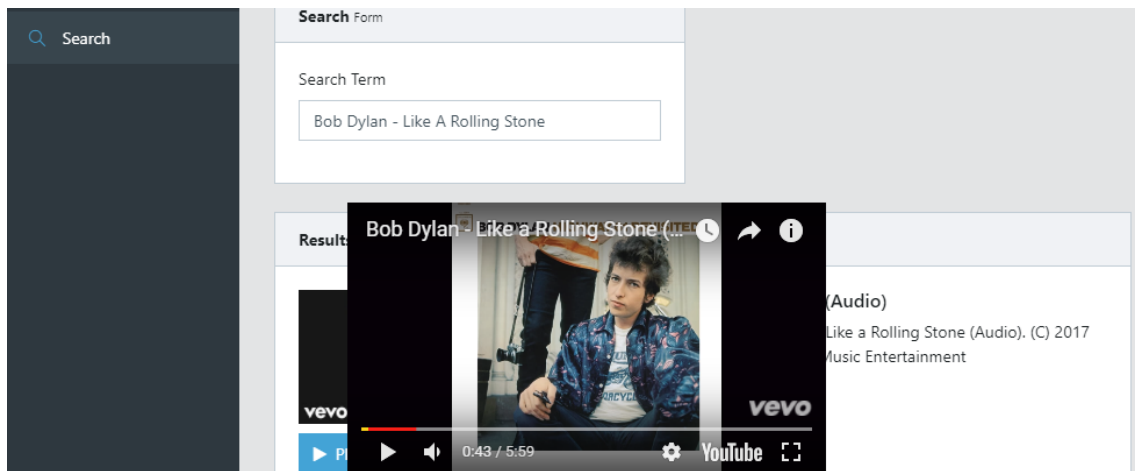
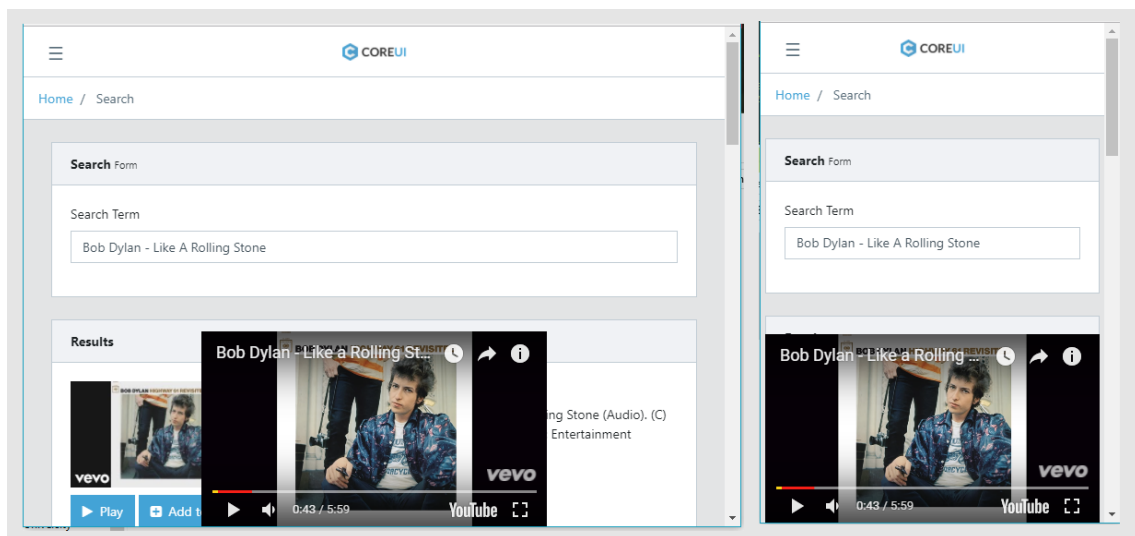


Figure 2.7: Video display on smaller devices, demonstrating the responsive design.



2.3.3 Redux

One major challenge that was faced while building the video playback component was how to correctly communicate between the search, playlist, and video display component. All of these *should* be independent parts of the application without any common parent to communicate through. This became an increasingly noticeable issue as more features were added during development. In the interest of time, the current solution to this problem was to simply have the video component be a child of the search component. This doesn’t make sense as a best approach, and so research has been done into finding a better solution moving forward.

One of the most frequently mentioned solutions to this problem was a JavaScript library known as Redux [5]. Redux is designed for managing the entire application state of a project. Libraries like Redux attempt to solve the state management and communication problem in the view layer by removing both asynchronous and direct DOM manipulation. This is instead replaced by a single plain JavaScript object which holds the entire state of the application.

Redux also works on a read-only basis, where the only way to update the state of the application is through actions. The other large part of Redux is that all functions used must be *pure functions*. This simply means that given a certain input, the same output should always be expected with no side-effects.

This would be a serious benefit to the project from a testing perspective. One of the major challenges faced in testing was dealing with complex component relationships. It would also help address issues with testing components that rely on API calls. Separating out the application state from the API calls would have a great positive impact on the project.

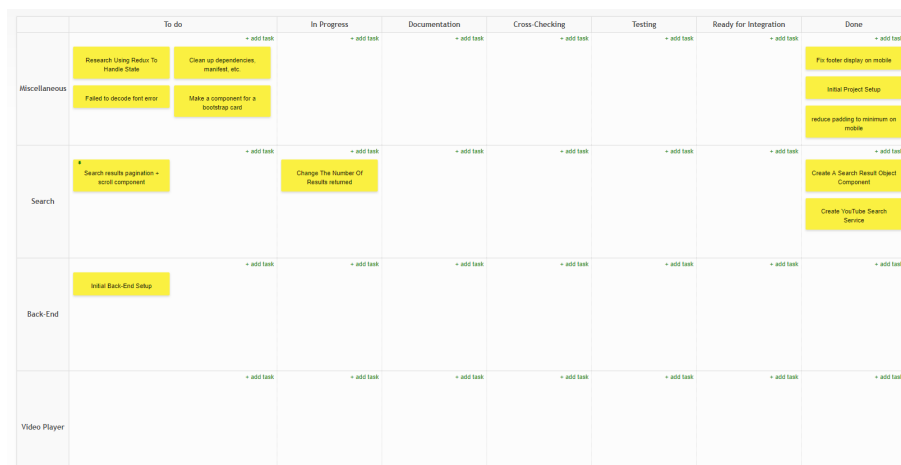
While the final solution to this problem is still under consideration, further research and experimenting will be done to determine if Redux would be a good solution to the challenging topic of state management within the program.

2.4 Current System

2.4.1 Methodology

The choice to go with the Kanban methodology has proven to be an excellent fit for this project. The Kanban board has been split into 7 swimlanes: “Miscellaneous”, “Search”, “Back-End”, “Video Player”, and “Playlist”.

Figure 2.8: The project’s Kanban board early on in development.



Where applicable, the testing and documentation steps have been an excellent natural location to ensure the application has kept up to date with its requirements to test and document the project.

A private GitHub repository has been set up where the *Git Flow* strategy has been followed. By ensuring that code is clean and follows a set of consistent conventions, the final pushed code to the repository has consistently been clean. The cross-checking column has meant all code must be double checked and cleaned up before it is committed to source control. This has shown to be a perfect step in the Kanban board.

The only notable challenge that this Kanban board approach has faced is that Kanban somewhat

assumes an already established project. Steps such as documentation are not always applicable to tedious tasks such as installing boilerplate code or the initial setup of the application. This is especially true when researching deployment and authentication strategies. It would often be unproductive to thoroughly test and document various experimental builds which will likely not be in the final product. For this reason the likelihood of risks relating to documentation and testing have been increased in the updated risk assessment.

2.4.2 Deployment

DigitalOcean Ubuntu Server

Deployment of the application has proven to be a much more complex task for the application than was originally thought. Early on in the project, the intention was to take advantage of ASP.NET Core becoming open sourced. This meant the application could be hosted away from the usual Windows-based hosting, and instead a more continuous integration strategy could be done on a Linux server.

Originally, a running prototype had been successfully configured to work on an Ubuntu server hosted on DigitalOcean [6]. The application could be easily hosted on what DigitalOcean call *droplets*, essentially fixed cost virtual machines [7]. By taking advantage of the Student Developer Pack [8] offered by GitHub [9], \$50 of free DigitalOcean credit was easily obtained to use. Given that this credit would easily cover the cheapest hosting plan of \$5 per month, this meant the application could be hosted for free for the duration of development.

Deploying the application to this Ubuntu server involved connecting through SSH using a root account. ASP.NET Core, Angular, and package managers could then easily all be installed, followed by the application source code being pulled from a Git repository. The final application could then easily be run with a few simple commands:

```
1 $ webpack
2 $ dotnet restore
3 $ dotnet run
```

This was an excellent solution from a continuous integration standpoint. Changes in the code would simply need to be pushed to a git repository, and the host server would simply need to pull and execute the above commands afterwards. While this was an excellent solution early on in the project, many challenges and limitations were discovered with this approach.

The biggest challenge was that many parts of a typical ASP.NET Core stack were still not compatible with a Linux server, the major one being *IIS* [12], ASP.NET Core's extensible web server. This meant an alternative had to be found to expose the application to the world wide web. This meant connecting to a database in a clean, continuous integration manner was also difficult to produce.

A working web server was added to replace IIS called Nginx, which allowed exposing the application through a reverse proxy. This solution worked well for a basic static implementation, but many crucial features such as ASP.NET Identity, Entity Framework, and connecting to an SQL Server database proved lacking in either cross-platform support, or any documentation online around this less common stack.

Microsoft Azure

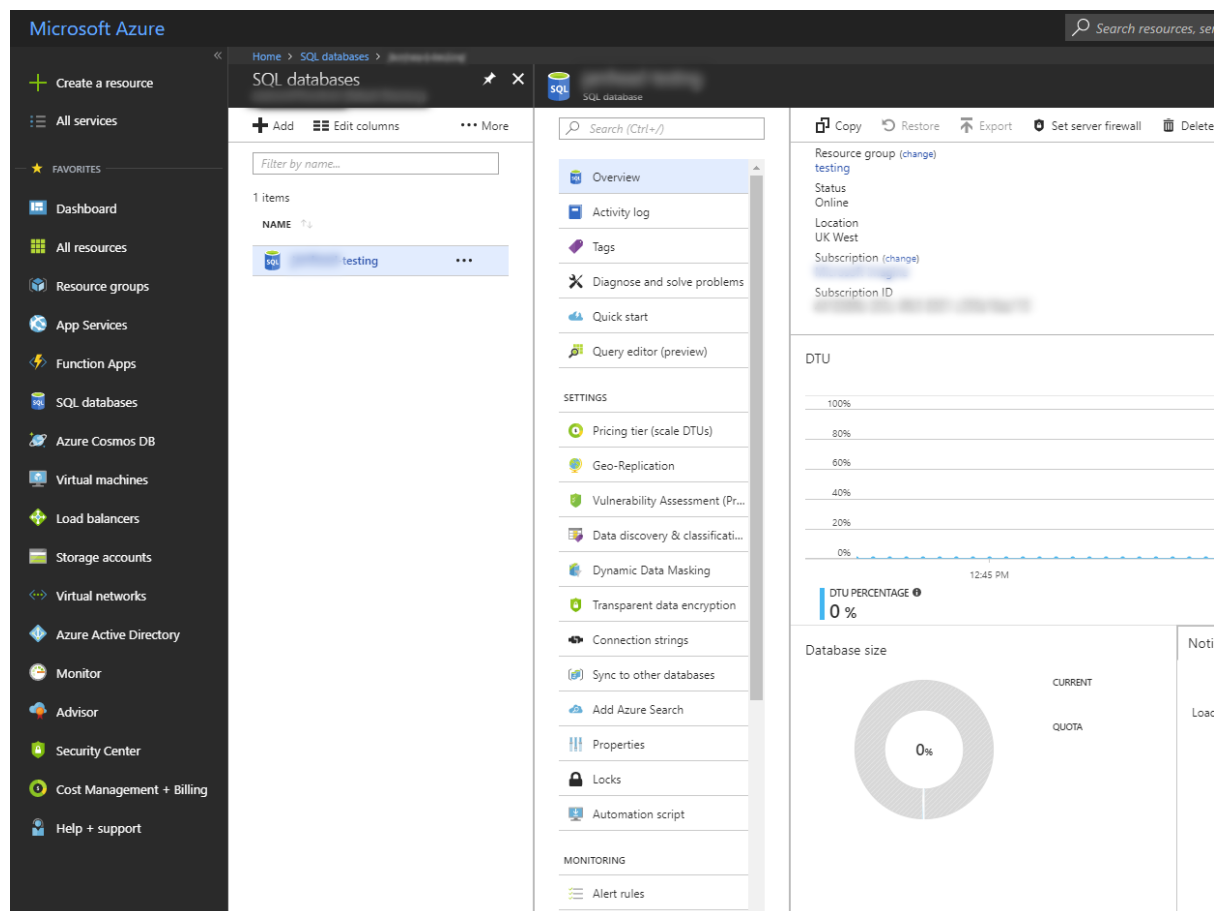
With the above solution causing issues, it was decided to explore a more standard deployment strategy. After careful research, *Microsoft Azure* was the decided upon deployment and hosting strategy [13]. Azure is another cloud computing service, similar to DigitalOcean, but also offers a full suite of services for managing applications, services, and deployment.

This fixed many problems with the original solution. As it is also a Microsoft product, and built to work closely with ASP.NET, this meant there was an abundance of resources and tools to easily deploy.

The database architecture has also been greatly simplified by switching to Azure. An SQL server has been set up within the Azure account, creating a full separation of concern between how the database is hosted versus the host machine. This also simplifies security, by easily restricting IP addresses which may connect to the database.

Another major consideration was the cost of hosting. Microsoft Azure can quickly become costly on a large scale. It also uses an hourly payment rate, which was not ideal given the project's lack of budget. Upon research, it was possible to get a free hosting plan through Microsoft's Imagine program [11]. This gives the project access to free hosting for the remainder of development, without any risk of an unexpected bill from excessive bandwidth usage.

Figure 2.9: Screenshot of the Microsoft Azure console for the project.



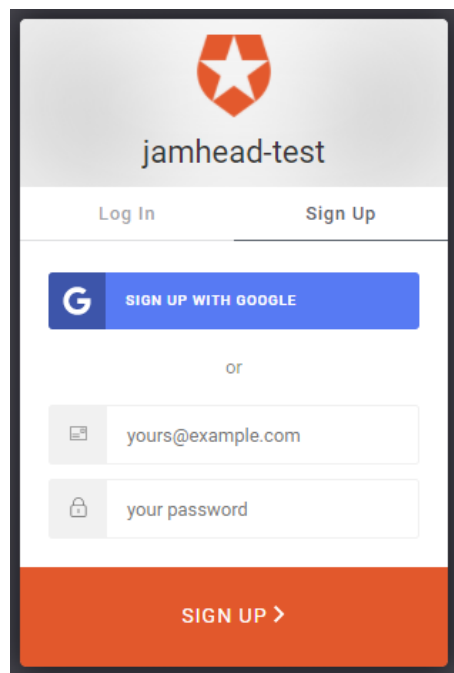
2.5 Authentication

Although it was decided to use ASP.NET Core for securely storing users, authenticating a user's session proved to be a separate issue. Early on in the project, before the back-end had been fully set up, options for a temporary front-end solution was explored. This would mean significant progress could be made on the bulk of front-end work, without being put on hold for an extended period of time while deciding on the back-end architecture.

The chosen method for this was using a service called Auth0 [3] for a very quick way to have a secure and trustworthy user authentication system. This has exceeded expectations as a temporary means to store user authentication. For the purposes of this project, the pricing has a generous free tier and very easily integrated with both Angular and ASP.NET Core.

The other added bonus to this was the possibility of adding social media authentication, such as authentication with Facebook or Google. Although this is beyond the minimum viable product, it adds strong scalability in future. Below is an example of this authentication page on the live prototype:

Figure 2.10: Auth0 sign up page.



Although this was only intended to be a temporary solution, I am planning to experiment and research further into the possibility of using Auth0 for the authentication, and ASP.NET Core for the actual API and database storage. This would be a slightly unusual solution, but would provide the benefits of a secure authentication channel from Auth0, and the benefits of a strong database connection with ASP.NET Core. If this proves problematic or impossible, the original solution of using just ASP.NET Core as described in the specification will be used.

2.6 Testing

Testing has made strong progress, with all tasks being tested as a clear step in the Kanban board mentioned earlier. This has helped catch numerous bugs before being merged and has kept the program running like clockwork.

The Kanban methodology has also made it easy to create new tasks for found bugs not related to the task in which it was found. For example, a display issue was found with the site's footer, which had it displaying incorrectly on mobile devices, pushing it to multiple rows. A new task could easily be created without disrupting the progress on the current task.

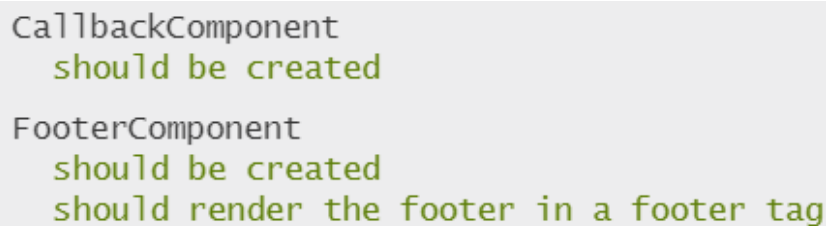
2.6.1 Angular Component Unit Testing

All components created so far have been given unit tests before being merged. This has made it very simple to quickly identify that any changes made have not broken previous functionality. An example of this is the footer bug mentioned above. After this issue, new unit tests were added using *Karma*, a JavaScript testing library included within Angular projects. An example of two of these tests is below:

```
1  it('should be created', () => {
2    expect(component).toBeTruthy();
3  });
4
5  it('should render the footer in a footer tag', async(() => {
6    const fixture = TestBed.createComponent(FooterComponent);
7    fixture.detectChanges();
8    const compiled = fixture.debugElement.nativeElement;
9    expect(compiled.querySelector('footer')).toBeTruthy();
10  }));
```

This gives the following output when tested in the Angular CLI:

Figure 2.11: Passing unit tests using Karma and the Angular CLI.



```
CallbackComponent
  should be created

FooterComponent
  should be created
  should render the footer in a footer tag
```

One major challenge that has been faced with testing is how to handle testing components which rely on asynchronous API calls. It is difficult to achieve a set of unit tests with impure functions. To solve this, a *service* class was created in order to isolate calls to the YouTube API. By isolating API calls in this way, it becomes much simpler to “mock” the API calls and focus on testing the actual program logic. Below is a snippet of the YouTube service used to isolate calls:

```

1  @Injectable()
2  export class YoutubeService {
3
4  constructor(private _http: Http) {
5  }
6
7  public search(query: string, maxResults = 10) {
8      maxResults = this.getMaxResults(maxResults);
9
10     return this._http.get(`${SEARCH_URL}?q=${query}&part=snippet,id&key=${
11         API_TOKEN}&maxResults=${maxResults}&type=video" )
12     .map(response => response.json());
13 }

```

In the above code snippet, the service uses the Angular *Injectable* annotation to allow the class to be injected into an Angular *module*. Modules in Angular provide all contained components an implementation of this class. This has made it very straightforward to mock the class by swapping out the provider for a different implementation. This is known as *Dependency Injection* and is conveniently built into the Angular framework [2].

Chapter 3

Progress Assessment

In order to assess the project's progress so far, two major factors will be considered, along with a personal reflection of how the project is progressing as a whole. These factors are the percentage of requirements complete so far, followed by how closely the project has managed to stick to the project plan outlined in the initial document.

3.1 Requirements And Specification Progress

In order to identify how much progress has been made in completing the original requirements, tables 3.1, 3.2, and 3.3 below shows the status of each requirement, along with comments related to that requirement. Currently the *miscellaneous* and *non-functional* requirements have made excellent progress. The *playlist* and *shuffling requirements* have not made progress as they are scheduled to be the focus for the remainder of this project.

Table 3.1: Miscellaneous Requirements Progress

Risk Analysis Table			
Requirement ID	Specification IDs	Status	Comment
MISC-REQ1	DATA-SPEC1, DATA-SPEC2, DATA-SPEC3, DATA-SPEC4	Complete with caveats ✓	Users are able to log in to the system, but the method put in place using Auth0 is not necessarily the final log in system. Discussed further in section 1.7.
MISC-REQ2	WEB-SPEC4	Completed ✓	Users are able to search through a large catalogue using the search results from the YouTube API, displayed on the search page.
MISC-REQ3	WEB-SPEC4, WEB-SPEC5	Completed ✓	Users are able to stream search results from within the website on all screen sizes.
MISC-REQ4	WEB-SPEC6, PL-SPEC1	Not Yet Complete	Playlist management is set to be the focus of time spent for the remainder of this project.

Table 3.2: Playlist And Shuffling Requirements Progress

Risk Analysis Table			
Requirement ID	Specification IDs	Status	Comment
PS-REQ1	PL-SPEC4	Not Yet Complete	Playlist management is set to be the focus of time spent for the remainder of this project.
PS-REQ2	PL-SPEC5	Not Yet Complete	
PL-SPEC3	WEB-SPEC4, WEB-SPEC5	Not Yet Complete	
PS-REQ4	SH-SPEC1, SH-SPEC2, SH-SPEC3, SH-SPEC4, PL-SPEC6	Not Yet Complete	
PS-REQ5	SH-SPEC2, PL-SPEC4	Not Yet Complete	
PS-REQ6	PL-SPEC2	Not Yet Complete	
PS-REQ7	PL-SPEC2	Not Yet Complete	

Table 3.3: Non-Functional Requirements Progress

Risk Analysis Table			
Requirement ID	Specification IDs	Status	Comment
NF-REQ1	WEB-SPEC1, WEB-SPEC2	Partially Completed ✓	During the back-end setup process, an accessible build was created. The final deployment and publishing strategy is likely to differ highly at release. Many tasks such as acquiring an SSL certificate and domain name will be decided closer to release at a more sensible time.
NF-REQ2	WEB-SPEC2, WEB-SPEC3, WEB-SPEC7, TEST-SPEC1, TESTSPEC2, TEST-SPEC3, TEST-SPEC4	Completed ✓	The website has been tested on all major evergreen browsers without issue. Very few browser specific bugs have been found, and have all been fixed during the testing step in the Kanban board.

Risk Analysis Table			
Requirement ID	Specification IDs	Status	Comment
NF-REQ3	WEB-SPEC3, WEB-SPEC7, TEST-SPEC1, TEST-SPEC3	Completed ✓	The website is fully responsible, with all features implemented having specific CSS for different screen resolutions where needed.
NF-REQ4	WEB-SPEC3	Completed ✓	The choice of using Bootstrap and the Core UI administrator template has given the website an excellent, intuitive interface with a similar look and feel to a classic music streaming site or administrator control panel.
NF-REQ5	WEB-SPEC2	Completed ✓	By successfully handling all routing through the Angular front-end framework, the website has been made into a Single-Page Application.
NF-REQ6	DATA-SPEC1, DATA-SPEC2, DATA-SPEC4, DATA-SPEC5, DATASPEC6, DATA-SPEC7, DATA-SPEC8	Mostly Completed ✓	The system put in place using Auth0 has provided a heavily secure and convenient method of handling user authentication. This is likely not the final solution to user authentication, discussed further in section 1.7.
NF-REQ7	DATA-SPEC5, DATA-SPEC6, DATA-SPEC7, DATA-SPEC8	Mostly Completed ✓	Security has been implemented at each step of the project, with each framework's trusted and built-in security features used. Security is of course an ongoing requirement that will always be relevant.

From the above table, 10 out of the 18 requirements have been either completed or partially completed so far. This puts the project at a healthy 55% completion rate. As mentioned above, many of the playlist and shuffling functionality is set to be completed after this interim document. It is also worth mentioning many of these shuffling and playlist requirements should be relatively straightforward to implement once the more complex requirements are out of the way. For example, deleting a song or tag from the user's library should be a mostly trivial task if the base implementation is done well.

3.2 Time Plan Progress

In the initial document for this project, a detailed time plan, including a gantt chart, was created to outline how time was to be spent on the project. A copy of this time plan is presented below in figure 3.1.

	December			January			February			March			April			May		
Task Name	14	20	30	08	15	22	05	16	26	05	16	26	09	16	23	07	11	17
Initial Project Setup	■	■																
Search And Retrieve Results			■	■	■	■												
User Authentication			■	■	■	■												
Playlist Features			■	■	■	■												
Interim Report							■	■										
Advanced Playlist Functionality								■	■	■	■							
Advanced Shuffle Functionality								■	■	■	■							
Any Outstanding Work											■	■	■					
User Manual													■	■				
Design Document														■	■			
Testing Document															■	■		
Narrative and Reflective Account																■	■	
Final Submission																	■	■

Figure 3.1: The original gantt chart showing the time plan for the project.

From the above chart the project has kept relatively up to date with the exception of implementing basic playlist features. This proved challenging to implement leading up to this interim report as a bottleneck had appeared where the playlists could not be stored until both the authentication and database storage strategy had been finalized.

Now that the back-end, database, authentication, and deployment strategy have been given extra time and care, development of the playlist and shuffle requirements will be given a higher priority and will be faster to implement.

In hindsight, the original gantt chart should have been made more specific to break down user authentication into multiple tasks. While this was done on purpose to more closely reflect the chosen Kanban methodology, tasks for the back-end and database could have been expressed more clearly by being separated out from user authentication. The updated gantt chart in this document has been made more specific for this reason.

Overall, while the project is slightly behind schedule according to the gantt chart, it has made great progress in areas not originally shown on the gantt chart. I am confident that the project will easily be complete on time and the remaining section of the project will see speed increases as a result.

3.3 Personal Progress Thoughts

The project has faced many challenges including authentication, a complicated back-end setup, and heavy deployment issues. In each of these cases, the problems have been overcome, and the project has started to take excellent shape.

The software architecture has been kept well maintained, with best practices in both Angular and ASP.NET followed. The website's look and feel has taken excellent shape, and integrating

new features is very straightforward thanks to the component based design the project has undertaken.

I am personally very satisfied with the progress and direction the project has taken. Although progress had temporarily slowed down to ensure that the project's architecture and continuous integration had been done correctly, I am confident the project will benefit from this in the long. Bugs should be found less frequently, integrating should be much smoother moving forward thanks to the strong base architecture developed thus far.

3.4 Final Progress Assessment

By considering the above three indications of progress, a final progress assessment can be made. For each of these sections a score between 1-5 has been given, with 1 indicating no progress and 5 indicating excellent progress. Brief comments have also been provided below on why the score was chosen.

Table 3.4: The final scoring assessment for the project's progress.

Risks In Descending Order		
Assessment Method	Score	Comments
Requirements and Specification Progress	4/5	Over half of the project's requirements have now been completed, with the remaining requirements all being closely related to each other.
Time Plan Progress	3/5	The project is behind the original project plan, but the reason for this has been justified above, and the remainder of tasks should be easier to implement as a result.
Personal Progress Thoughts	4/5	The software architecture is well maintained and all challenges so far have been overcome.
Final Score	4/5	An average of the scores given above, rounded to the nearest whole number.

With the scores considered above, it is clear the project has made good progress so far. It is important that both the time plan and risk analysis table be updated in this interim report to be more specific and take into account the challenges that have slowed progress so far.

With this progress assessment result, I have high confidence that the project will be fully complete by the final deadline.

Chapter 4

Risk Analysis

4.1 Overview

In this chapter, an analysis of the risk assessment made in the initial document will be undertaken. Included is a discussion of the risks encountered, the risks that were not encountered, and risks that have been discovered outside of the original risk analysis.

Following this, an updated risk analysis table has been created. A more detailed scoring system has also been provided which shows a more accurate rating of how these risks apply to the project. Additionally, many of the original risks have been modified or removed to more accurately reflect the project's current state.

4.2 Encountered Risks

Being Unfamiliar With Chosen Technologies

Although I have previous experience with both the Angular front-end framework and the ASP.NET Core back-end framework, this is the first time I have ever attempted to merge the two into a single project. Although appropriate research was undertaken beforehand, I had not anticipated the complexity involved in using ASP.NET Core outside of its usual server-side rendering stack. This has proven to be a big challenge in the application leading up to this document and will be reflected in the updated risk analysis.

Final Product Does Not Meet The Specification

While the specification has largely been followed successfully, there are a few specifications which were overly specific in the original document. These specifications will still be followed, but will be more difficult to implement as a result. The risk level involved will still remain mostly unchanged, as this is not likely to have any noticeable impact on the final deliverable.

Failure to document methods and classes

With the challenges and time invested into deployment and merging the front-end and back-end together, there has been some forgetfulness in documentation. This risk will remain the same likelihood as it is likely to not be an issue once the focus is switched to the playlist and shuffling implementations.

4.3 Risks Encountered Outside Of Risk Analysis

Legal Implications Of Music Streaming

One risk that was not considered in the original risk analysis is that of the legal implications of streaming music results from YouTube. In the case of this application, by embedding results from the YouTube API, as opposed to hosting videos from the site itself, this risk has not been an issue. It is important that the YouTube API terms and conditions are followed appropriately in order to ensure this. For the most part, simply ensuring all advertisements from the embedded iframe are displayed clearly and without interference will ensure this is not an issue.

Exceeding API Access Tokens

Although the project has access to a quota of 50,000,000 per day, (roughly 500,000 search requests per day), there has been some small risk of exceeding the YouTube API quota while developing the project. The reason for this is infinite API call loops which would make a never ending stream of requests. Fortunately, the search results page is limited to showing only 10 results at a time, so an infinite loops has been avoided thus far. Ultimately, this would only result in loss of API access for a single day, so is not a major risk.

Software Deployment

One of the biggest challenges so far with the project has been that of developing a successful deployment strategy. So far this has slowed development enough to be a noticeable risk to the project, and will be added to the risk analysis with high likelihood.

4.4 Other Risks

In this section, risks that have not been encountered so far will be discussed on whether they are still applicable to the project.

Data Loss Corruption

By taking advantage of cloud storage, and by regularly pushing source code to the private Git repository, this risk has proven very unlikely to happen. It is extremely unlikely that the multiple sources this project's source code is hosted on should all become corrupt. This risk is still included as it does pose a serious danger to the project if regular uploads were not performed.

Short Term Illness

So far there has been no illness during development. This risk is still relevant to the project and has not become any less likely to occur than before. By taking advantage of the flexible Kanban methodology, and leaving a one week grace period before deadlines, this risk should continue to not have a major impact on the project if it were to occur.

Feature Creep

Care has been taken to only stick to implementing the specifications and requirements outlined in the initial document. This has meant no time has been wasted developing additional func-

tionality. This risk has become less relevant to the project now that a clear schedule and time plan has been confirmed and should not change.

Poor Coding Conventions And Best Practices

The coding conventions have been followed consistently. There has not been any major slow-downs caused by this, and so this risk has been reduced in risk in the updated risk analysis.

4.5 Updated Risks Table

An explanation of the revised scoring numbers used in these tables is presented below in table 4.1. In this risk assessment table a description of the risk is given along with a risk mitigation strategy. The likelihood and potential damage to the project is also given with a score between 1-6. This is then multiplied together to give a risk number.

Table 4.1: An explanation for the risk analysis presented in table 4.2 and 4.3

Risk Analysis Table Explanation				
Risk Chart			Damage Chart	
Level 1:	Minor Risk		Level 1:	Minor Damage
Level 2:	Low Risk		Level 2:	Low Damage
Level 3:	Moderate Risk		Level 3:	Moderate Damage
Level 4:	Serious Risk		Level 4:	Serious Damage
Level 5:	Massive Risk		Level 5:	Massive Damage
Level 6:	Guaranteed Risk		Level 6:	Complete Project Wipe
Colour Warning			Risk Number	
Green:	Minimum Risk/Damage		0-11	
Orange:	Moderate Risk/Damage		12-23	
Red:	Massive Risk/Damage		24-36	

4.6 Technical Risks

Table 4.2: Technical Risks For The Project

Risk Analysis Table				
ID	Risk	Likelihood	Damage	Risk Number
1	Data Loss/Corruption A large data loss would mean starting the project from scratch, potentially making finishing by the deadline impossible. Risk Mitigation 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.	1	6	9
2	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. Risk Mitigation 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.	3	4	12
3	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. Risk Mitigation 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.	4	3	12
4	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. Risk Mitigation A step in the Kanban workflow will be added where all tasks should be double checked and reviewed before merging.	4	3	12

Risk Analysis Table				
ID	Risk	Likelihood	Damage	Risk Number
5	<p>Exceeding API Access Tokens</p> <p>The YouTube API uses a quota to ensure that developers use the service as intended and do not make excessive requests. Although this is a very generous quota, infinite loops within the application could easily use up the daily quota and lock further requests for a short period of time.</p> <p>Risk Mitigation</p> <p>Care will be taken to ensure any methods making calls to the API are free of infinite loops.</p>	2	2	4
6	<p>Software Deployment</p> <p>Given my inexperience with deploying software, this may slow down development significantly.</p> <p>Risk Mitigation</p> <p>Extra time will given in the project plan for researching and implementing a solid deployment strategy which does not slow down development.</p>	4	4	16

4.7 Non-Technical Risks

Table 4.3: Non-Technical Risks For The Project

Risk Analysis Table				
ID	Risk	Likelihood	Damage	Risk Number
7	Short Term Illness In the event of a short term illness, such as a cold or fever, the project's development could be held back by up to a week. Risk Mitigation The flexibility of the chosen methodology, Kanban, means that the workload can be easily adjusted to be lightened or increased depending on illness or deadlines outside the project.	4	2	8
8	Feature Creep Given the nature of the project, it is possible to easily extend the project with an abundance of features. Risk Mitigation By adhering to a strict set of requirements and specifications outlined in the initial document, any risk of feature creep is avoided.	2	4	8
9	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.	2	4	8
10	Legal Implications Of Music Streaming To allow for music to be streamed online, royalties must of course be paid. Although the project does not host any music itself, the site whose music is embedded from does. In most cases advertisements are used to allow music to be streamed free of charge. Any attempt to hide or remove these advertisements would likely result in termination of support. Risk Mitigation Embedded YouTube videos will be displayed prominently and all advertisements will be displayed appropriately.	2	5	10

4.8 Risks in Descending Order

Table 4.4: Risks in descending order.

Risks In Descending Order			
ID	Risk	Risk Number	Change
6	Software Deployment	16	New
4	Failure to document methods and classes	12	Increased
3	Poor Coding Conventions And Best Practices	12	Reduced
2	Being Unfamiliar With Chosen Technologies	12	Increased
10	Legal Implications Of Music Streaming	10	New
1	Data Loss/Corruption	9	Reduced
7	Short Term Illness	8	Unchanged
8	Feature Creep	8	Reduced
9	Final Product Does Not Meet The Specification	8	Unchanged
5	Exceeding API Access Tokens	4	New

Chapter 5

Updated Time Plan

Covered in this chapter is an updated time plan for the project. Included is a discussion on the issues with the original time plan, and how this has been addressed in the updated time plan.

5.1 Issues With The Original Time Plan

Below is a copy of the original gantt chart, as found in the *methodology, requirements and specification* document for this project.

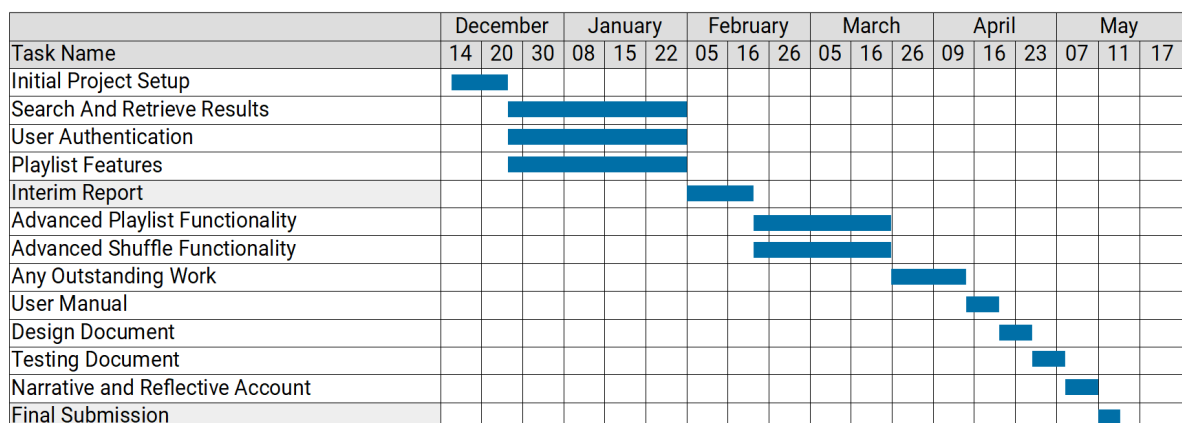


Figure 5.1: The original gantt chart for the project.

In order to accurately take advantage of the biggest strengths of the Kanban methodology, the original gantt chart for the project was kept purposely flexible and featured multiple tasks being done in parallel. While this proved effective at maintaining flexibility, the gantt chart was not detailed enough in breaking down tasks. This firstly meant tasks had no real sense of time needed. In figure 5.1 the tasks “Search And Retrieve Results” and “Playlist Features” are given equal amounts of time, but the latter was intended to be a basic implementation to be extended later in the project. This has been addressed in the updated gantt chart, with more specific times and tasks chosen.

This has also made it difficult to give an accurate assessment of the project’s progress up until this moment. As an example, there has been significant progress on the “User Authentication” task, but this covers a wide range of topics that would be better assessed on an individual basis.

5.2 Updated Gantt Chart

In order to improve upon the original time plan, and to incorporate new tasks that have been identified in the project so far, an updated gantt chart has been created. This has been made more specific to improve upon the challenges faced with the original gantt chart discussed above.

Task Name	February			March			April			May		
	05	16	26	05	16	26	09	16	23	07	11	17
Interim Report												
Perform mid-way project clean up.												
Finalise hosting strategy.												
Finalise authentication and database.												
Create a basic Angular playlist component.												
Store playlists into database.												
Basic Playlist And Shuffling												
Create a simple song object class .												
Implement basic linear and random shuffle playback.												
Create a simple version of a tag.												
Handle Back-End Storage of songs and tags.												
Specific Tags And Shuffling Tasks												
Allow tags to be marked as playlist, genre, etc.												
Allow tags to be marked for desired listen frequency.												
Allow tags to be nested within each other.												
Storage of desired and actual listen frequency.												
Implement shuffle by specified listen frequency.												
Implement shuffle by actual listen frequency.												
Final Testing And Outstanding Work												
Implement any outstanding work												
Fix all bugs and failed tests where found.												
Deploy Final Version												
Final Documents												
Write user manual.												
Write design document.												
Write testing document.												
Write narrative and reflective account document.												
Final Submission												

Figure 5.2: The updated gantt chart, showing the time plan for the project.

As can be seen from figure 5.2, the tasks have been made far more specific now that a clear list of remaining work is known. In order to keep within the chosen Kanban methodology, many tasks have still been kept flexible. Many of the playlist and shuffle related tasks have purposely been given the same time slot. These tasks are mostly independent and can be completed in any order. Tasks here have also been kept relatively consistent in size and time given.

Some time has been added to allow for a project clean up, as well as time for implementing any outstanding work near the end of the project. This gives the project plenty of time for reflection and also acts as a chance for any tasks which fall behind schedule to catch up. As the risk of minor illness has remained in the updated risk analysis, these grace periods double up as an excellent safety net for the project.

Bibliography

- [1] Account, BobDylanVEVO YouTube. *Bob Dylan - Like A Rolling Stone*. 2017. URL: <https://www.youtube.com/watch?v=Iw0fCgkyEj0> (visited on 02/16/2018).
- [2] Angular. *Angular Dependency Injection*. URL: <https://angular.io/guide/dependency-injection> (visited on 02/14/2018).
- [3] Auth0. *Single Sign On & Token Based Authentication*. URL: <https://auth0.com/> (visited on 02/12/2018).
- [4] Bootstrap. *Build responsive, mobile-first projects on the web with the world's most popular front-end component library*. URL: <https://getbootstrap.com/> (visited on 02/16/2018).
- [5] Dan Abramov, Andrew Clark. *An open-source JavaScript library designed for managing application state*. URL: <https://redux.js.org/> (visited on 02/14/2018).
- [6] DigitalOcean. *Cloud Computing, Simplicity at Scale*. URL: <https://www.digitalocean.com/> (visited on 02/11/2018).
- [7] DigitalOcean. *Droplets - scalable compute services*. URL: <https://www.digitalocean.com/products/droplets/> (visited on 02/11/2018).
- [8] GitHub. *Student Developer Pack - Learn to ship software like a pro*. URL: <https://education.github.com/pack> (visited on 02/11/2018).
- [9] GitHub. *The world's leading software development platform*. URL: <https://github.com/> (visited on 02/11/2018).
- [10] Holeczek, Lukasz. *CoreUI - Free Bootstrap Admin Template*. URL: <https://coreui.io/> (visited on 02/16/2018).
- [11] Microsoft. *Bring your ideas to life. Find out how student developers can join Microsoft Imagine, and elevate their skills with developer tools and resources*. 2018. URL: <https://imagine.microsoft.com/> (visited on 02/16/2018).
- [12] Microsoft. *Host ASP.NET Core on Windows with IIS*. URL: <https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?tabs=aspnetcore2x> (visited on 02/11/2018).
- [13] Microsoft. *Microsoft Azure Cloud Computing Platform & Services*. 2018. URL: <https://azure.microsoft.com/en-gb/> (visited on 02/16/2018).
- [14] Nginx, Inc. *High Performance Load Balancer, Web Server, & Reverse Proxy*. URL: <https://www.nginx.com/> (visited on 02/11/2018).
- [15] ONS. *Statistics On Sickness Absence in the Labour Market*. URL: <https://www.ons.gov.uk/employmentandlabourmarket/peopleinwork/employmentandemployeetypes/datasets/sicknessabsenceinthelabourmarket> (visited on 02/14/2018).