Combining Music Streaming Services

James Watson

May 2016

Project Dissertation submitted to the University of Wales, Swansea in Partial Fulfilment for the Degree of Bachelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
University of Wales Swansea

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

March 28, 2018

Signed:

Statement 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of a BSc in Computer Science.

March 28, 2018

Signed:

Statement 2

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

March 28, 2018

Signed:

Statement 3

I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

March 28, 2018

Signed:

Acknowledgements

A big thank you to Neal Harman for being an excellent supervisor over the course of this project and an excellent lecturer over the last three years.

Contents

Chapter 1

Introduction

With the increased use of music streaming services (and the increased competition between these services [14]), music sites have become more specific on the type of music they provide. For example, YouTube.com offers an excellent platform for undiscovered talent, but often can be a poor choice for popular studio quality albums. A tool is therefore desirable to combine searching and playlists from a range of music streaming services. This eliminates the need to manually search each site for them.

The aim of this project is to create this tool in the form of an asynchronous web application. The tool will allow you to search a range of different sites simultaneously, asynchronously playing results to the site. With this, dynamic playlists can be created where music from different sites can be combined into one convenient list.

This tool opens up the following benefits:

- Faster results - no manual searching necessary.
- More unified content - content from different sites can now be enjoyed under one roof.
- Dynamic playlist creation - provides the potential to create a playlist consisting of content from multiple sites.
- Because no music is hosted by the tool itself, no bandwidth or royalty fees are required.

1.1 Minimum Features Set

In order to evaluate whether the project was a success, a minimum features set was created in order to judge whether the system had the minimal features necessary in order to be useful. The list was as follows:

- Accept a textual description of a song as input from the user.
- Dynamically search potential sites based on this input - i.e. find the song that most closely matches the description without a fixed URL given.
- Locate songs automatically without visiting potential sites.
- Support results from a minimum of 2 different music streaming services.
- Ability to create a basic dynamic playlist consisting of only a textual description of each song beforehand, with no fixed URL to a song used.

Although not strictly essential, the ability to create simple playlists from the application ensures the application is more than a glorified search engine, but a viable method to combine music streaming services conveniently. This basic feature works excellently as a measurement of the unification of sites supported.

The text entered by the user will simply be a single string from a single text box, without any separation of song name and artist. For the minimal set of features parsing text is not covered as it can be handled by the sites themselves through API calls.

Chapter 2

Background Research

2.1 Similar Applications

The concept of combining multiple streaming services into one is something that has not been explored to a large extend. Upon research, only two applications exist which have made a strong attempt at this concept; however, the former does not offer even basic features such as playlist creation, and the latter only works for a single music site.

2.1.1 TuneCrawl [19]

TuneCrawl.com is the most similar service to the proposed tool, offering users the ability to search and play music from three major sites - YouTube, Spotify and Soundcloud. The site also functions as a *Single-Page Application*, meaning all content is condensed to one page and results are returned asynchronously [18]. To this end, results are embedded to the site and can be listened to from the site itself. The site's intended use is a quick and easy search engine for finding a particular song compared to having to search each site individually.

In its current form, the site resembles a state of abandonware, demonstrated by figure 2.1. Of the three sites, only results from SoundCloud could be played, with YouTube infinitely loading and Spotify not displaying at all. The site also offers no playlist capabilities. In its current state the site has potential as a convenient search tool, but falls short as a way to combine results.

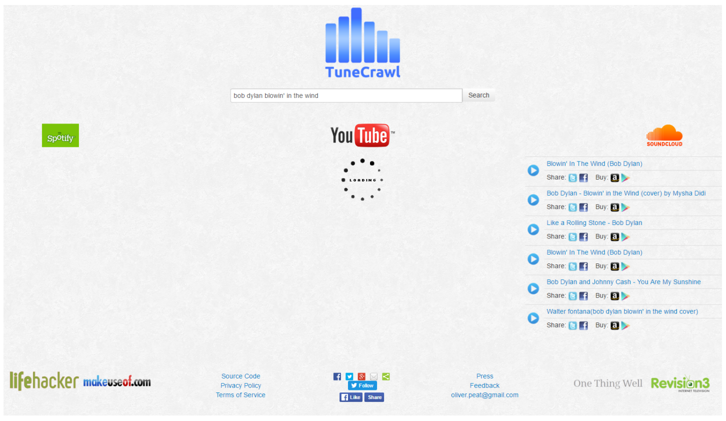
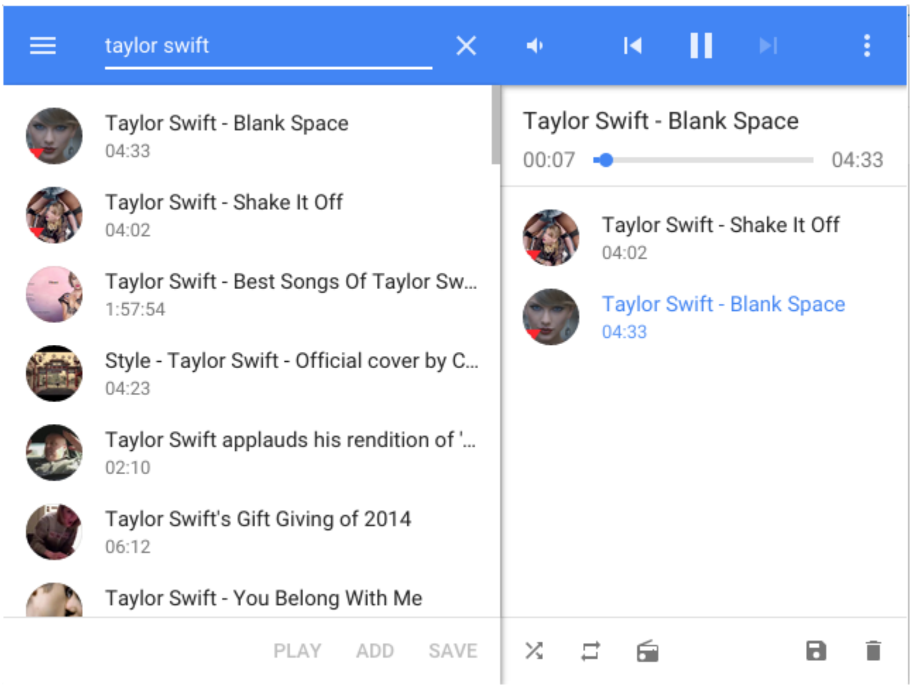

TuneCrawl
bob dylan blowin' in the wind
Search
Spotify
YouTube
LOADING
SOUNDCLOUD
Blowin' In The Wind (Bob Dylan)
Share: Buy:
Bob Dylan - Blowin' in the Wind (cover) by Mysha Didi
Share: Buy:
Like a Rolling Stone - Bob Dylan
Share: Buy:
Blowin' In The Wind (Bob Dylan)
Share: Buy:
Bob Dylan and Johnny Cash - You Are My Sunshine
Share: Buy:
Walter fontana(bob dylan blowin' in the wind cover)
Share: Buy:
lifehacker makeuseof.com
Source Code
Privacy Policy
Terms of Service
Follow
Like Share
Press
Feedback
oliver.peat@gmail.com
One Thing Well
Revision3

Figure 2.1: Screenshot of the TuneCrawl website as of February 2016.

2.1.2 Streamus [2]

Streamus is an extension for Google Chrome which was built to turn YouTube into a music player. Playlists can be created and combined from within the extensions window. It also offers a built in search tool, asynchronously displaying and playing results from YouTube's API. Although Streamus adds a lot of functionality to turn YouTube into a music player, it does not offer results from other sites. An ideal solution would also preferably be a web application and not a chrome extension.

As of October 2015, the extension was shutdown and has shown no signs of returning during this project's development [33].



Taylor Swift - Blank Space
04:33
Taylor Swift - Shake It Off
04:02
Taylor Swift - Best Songs Of Taylor Sw...
1:57:54
Style - Taylor Swift - Official cover by C...
04:23
Taylor Swift applauds his rendition of '...
02:10
Taylor Swift's Gift Giving of 2014
06:12
Taylor Swift - You Belong With Me

Taylor Swift - Blank Space
00:07 04:33
Taylor Swift - Shake It Off
04:02
Taylor Swift - Blank Space
04:33

Figure 2.2: Screenshot of the Streamus Chrome extension in 2014. Screenshot by Owen Williams for thenextweb.com [32].

2.2 Accessing Music

In order to learn how to retrieve and embed songs from different sites, research was undertaken on how communication with various music APIs could work programmatically. A common trend observed was that all researched sites used some form of *RESTful web service*, almost always sending JSON objects to receive. A RESTful web service in simple terms being a form of web service where communication is done via HTTP protocols. The basic structure of a request would always take this form:

- Set up a developer API Key.
- Make an HTTP GET request using a search query.
- Receive JSON object representing a song.
- Extract only the useful information from this request. This includes the song's unique ID.

- Manipulate relevant song data as desired.

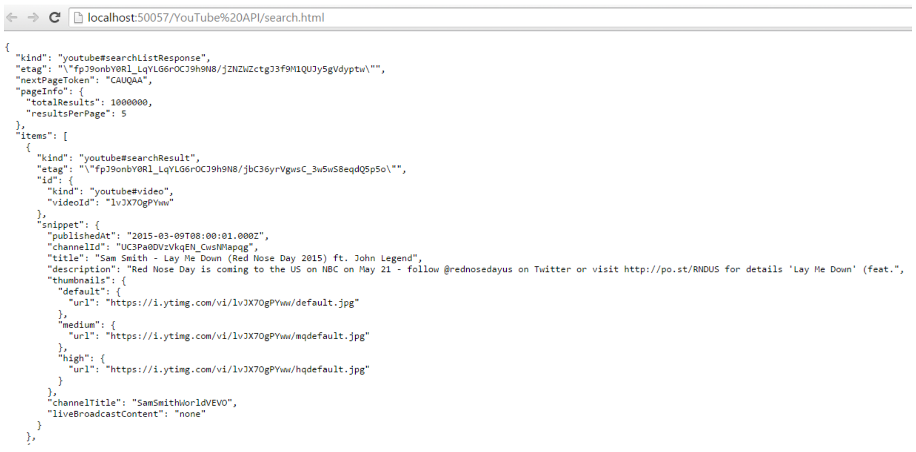
The main difference observed between each site was the initial setup to make API calls. Each site usually featured its own SDK which could be easily integrated into JavaScript or other languages. The structure of returned JSON objects was another key difference. Isolating only the relevant details from these JSON objects, such as title or description, had to be done on a site-by-site basis and simplified later on.

In order to further research API calls, two sites were researched to a greater depth - YouTube and SoundCloud.

2.2.1 YouTube API [12]

The YouTube API features a large set of tools for embedding YouTube functionality into a website. For simplicity, a combination of HTML and JavaScript was used in researching the YouTube API. An official JavaScript SDK is provided for use with the YouTube API making this a straightforward choice of technology.

A Google developer account was set up for this research, using specifically a YouTube Data API key. Once this was done, GET requests could easily be made within JavaScript. A collection of results was successfully returned in the form of JSON objects converted to a string. This is displayed in the browser through HTML as such:



```
{
  "kind": "youtube#searchListResponse",
  "etag": "\"fpJ9onbY0Rl_LqYLG6rOCJ9h9N8/jZNZWZctgJ3f9M1QUJy5gVdyptw\"",
  "nextPageToken": "CAUQAA",
  "pageInfo": {
    "totalResults": 1000000,
    "resultsPerPage": 5
  },
  "items": [
    {
      "kind": "youtube#searchResult",
      "etag": "\"fpJ9onbY0Rl_LqYLG6rOCJ9h9N8/jbC36yrVgwsC_3w5wS8eqdQ5p5o\"",
      "id": {
        "kind": "youtube#video",
        "videoId": "lvJX7OgPYww"
      },
      "snippet": {
        "publishedAt": "2015-03-09T08:00:01.000Z",
        "channelId": "UC3Pa0DVzVkqEN_CwsNMapqg",
        "title": "Sam Smith - Lay Me Down (Red Nose Day 2015) ft. John Legend",
        "description": "Red Nose Day is coming to the US on NBC on May 21 - follow @rednosedayus on Twitter or visit http://po.st/RNDUS for details 'Lay Me Down' (feat.",
        "thumbnails": {
          "default": {
            "url": "https://i.ytimg.com/vi/lvJX7OgPYww/default.jpg"
          },
          "medium": {
            "url": "https://i.ytimg.com/vi/lvJX7OgPYww/mqdefault.jpg"
          },
          "high": {
            "url": "https://i.ytimg.com/vi/lvJX7OgPYww/hqdefault.jpg"
          }
        },
        "channelTitle": "SamSmithWorldVEVO",
        "liveBroadcastContent": "none"
      }
    },
```

Figure 2.3: Resulting JSON object for a simple search using the YouTube API.

By Extracting useful fields from this, such as the title or description, this data could then be used for more useful tasks. An example of using this looked as follows:

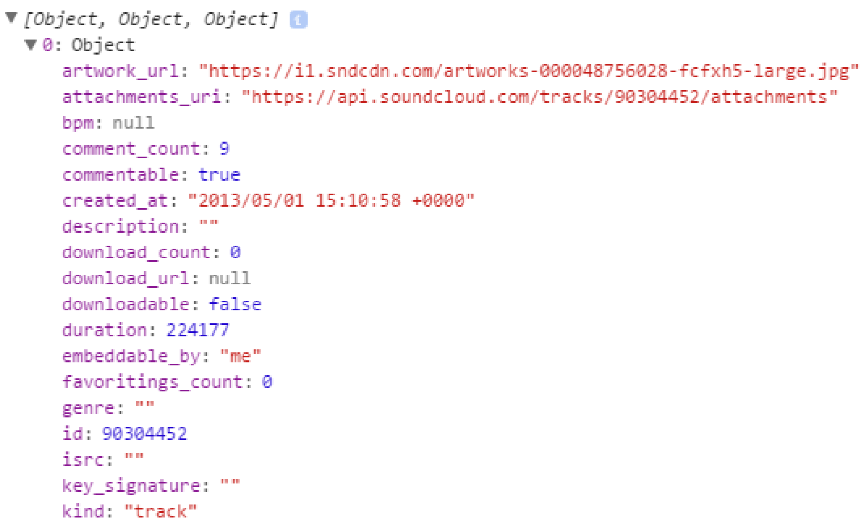
```
console.log(items.snippet.title +", "+items.snippet.description);
```

2.2.2 SoundCloud API [26]

Working very similarly to the YouTube API was the SoundCloud API. A similar JavaScript SDK was provided for use which allowed for very simple RESTful GET requests to songs.

A SoundCloud developer account was set up much like with the YouTube API, with a unique API key also provided. Once this was setup, GET requests were easily made within a few JavaScript methods. A similar collection of JSON objects could be found, but with a simpler layout than the YouTube API.

When viewed from the console within a browser, the resulting JSON looks as such:



```
▼ [Object, Object, Object]
  ▼ 0: Object
      artwork_url: "https://i1.sndcdn.com/artworks-000048756028-fcfxh5-large.jpg"
      attachments_uri: "https://api.soundcloud.com/tracks/90304452/attachments"
      bpm: null
      comment_count: 9
      commentable: true
      created_at: "2013/05/01 15:10:58 +0000"
      description: ""
      download_count: 0
      download_url: null
      downloadable: false
      duration: 224177
      embeddable_by: "me"
      favoritings_count: 0
      genre: ""
      id: 90304452
      isrc: ""
      key_signature: ""
      kind: "track"
```

Figure 2.4: Resulting JSON object for a simple search using the SoundCloud API.

Useful information could then again be extracted. Replicating the task from the previous API might look as such:

```
console.log(result.title +", "+result.description);
```

Chapter 3

Development Tools

3.1 Front-End Framework and Design

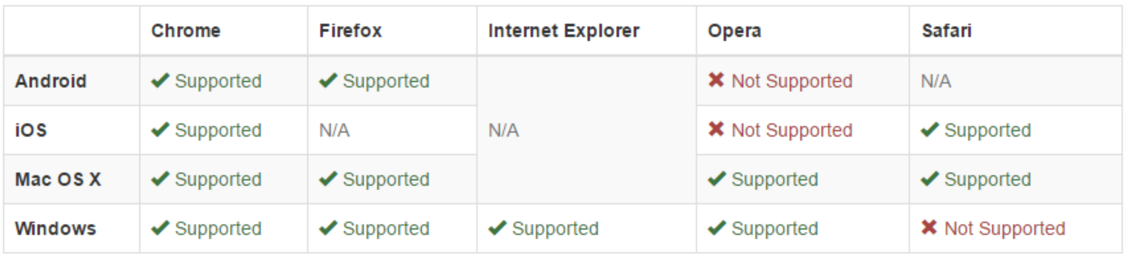
Because the design portion of this project was not a major focus, a *CSS framework* was used in order to simplify the design process and allow time to be spent more efficiently on the important aims of this project, and not on the design portion. A CSS framework in this case refers to a pre-written set of rules for styling elements with CSS.

3.1.1 Bootstrap [24]

In order to choose a front-end framework for this project, a set of requirements was set:

- Support media queries which allow a document to be styled differently depending on the device size.
- Include a grid layout which will adapt to a variety of device sizes.
- Include sufficient documentation and browser support.

With the above considerations in mind, the chosen front-end framework was Bootstrap. It is recognised as the world's most popular mobile-first and responsive front-end framework [29]. Bootstrap is sufficiently documented and is supported by all major browsers, as demonstrated in figure 3.1.



	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✓ Supported	✓ Supported		✗ Not Supported	N/A
iOS	✓ Supported	N/A	N/A	✗ Not Supported	✓ Supported
Mac OS X	✓ Supported	✓ Supported		✓ Supported	✓ Supported
Windows	✓ Supported	✓ Supported	✓ Supported	✓ Supported	✗ Not Supported

Figure 3.1: Supported browsers for Bootstrap [22].

Bootstrap uses a responsive 12 column grid system which allows developers to dedicate a certain amount of width to each section of a page. This grid can be customised further to adapt to four different screen widths, with different amounts of space dedicated depending on which is detected:

- xs - "Extra-Small" screens such as smart phones.
- sm - "Small" screens such as larger smart phones.

- md - “Medium” sized screens such as tablets.
- lg - “Large” screens such as full monitors.

A further large benefit to choosing Bootstrap was its large community support. As the most widely used mobile first front-end framework today [29], countless templates and stylesheets exist allowing quick and easy layout for the project. Given that the projects core framework was built with AngularJS (discussed below), the two are very commonly made to work in harmony with each other.

Although competitor frameworks exist, such as Foundation [35] or W3.CSS [28], (which early builds of the project used - see interim document [31]), Bootstrap was eventually picked. Both competitors offered all the essential features needed with good support, but lacked the countless pre-defined styles and templates offered by Bootstrap’s large community. Given that the focus of the project was not on the design aspect, this was a large selling point.

3.2 Main Web Application Framework

During the projects development, it became increasingly apparent that the entire site would function on one single page as a *Single-Page Application.* The content of the site was also clearly defined into separate modules, namely searching, playlists and video display. These requirements led to the decision to use AngularJS as the core framework of choice. AngularJS is a JavaScript framework which extends HTML vocabulary [11].

AngularJS is designed primarily for building *Single-Page Applications.* A SPA in this case is a web application delivered to the browser that doesn’t reload the page during use [18]. For this project, the website never attempts to reload an entirely new page, but instead partially loads the one page asynchronously to update content. AngularJS’s MVC style control also lends itself perfectly to the incremental methodology used for this project (discussed in the Design and Implementation section).

Although AngularJS is a relatively new technology, more common web technologies such as ASP.NET did not prove adequate for this style of website. In earlier versions of the project, ASP.NET and Visual Studio were used as the main framework and IDE. Because all back-end functionality was generally handled by API calls within JavaScript, the benefits of ASP.NET proved unhelpful. Having to rely on vanilla JavaScript to juggle multiple API calls asynchronously proved increasingly messy, which led to the swap to AngularJS.

3.3 APIs Used

Because the site requires searching and retrieving data from popular music streaming sites, calls to the APIs of each site was used. A typical communication of this process involved some form of RESTful web service calls. This had a strong impact on the choice of programming language, as each site usually provided an official SDK which could be integrated into certain programming languages. The one language which all sites’ SDKs supported was JavaScript.

Chapter 4

Older Prototypes

4.1 Prototype 1

Before the Gregynog presentations (which took place in November 2015), a prototype was built with the intention of showing a basic implementation of the search module of the project. The prototype demonstrated the application's potential to return results from a range of sites in a matter of seconds, but was quite poor at selecting the optimal choice.

Experiments were done in order to try and improve returned results by filtering certain results and parsing input. Ultimately this didn't prove effective. Each user has a slightly different style of searching; some users will search a select few keywords while others might concisely search their desired song. What ultimately proved to work much better after this prototype was providing a short list of the top results in order to give a much larger margin of error. Embracing the asynchronous changes made in newer prototypes also improved this.

4.1.1 Technology Used

This particular prototype was built using ASP.NET, but did not end up taking advantage of its features. Because of all the APIs involved, JavaScript showed to be an essential choice of language for most communication. This is because the major sites implemented all provided convenient JavaScript SDKs (discussed in the development tools section). The end result was a single page which functioned almost completely client-side, with almost no server-side code.

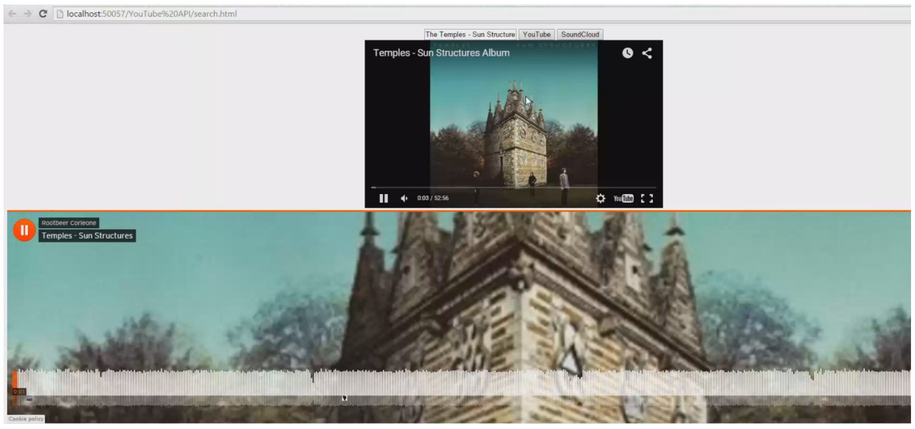

localhost:50057/YouTube%20API/search.html
The Temples - Sun Structure
YouTube
SoundCloud
Temples - Sun Structures Album
0:03 / 52:56
Rootbeer Corleone
Temples - Sun Structures
Cookie policy

Figure 4.1: Screenshot of the first prototype built. The song Sun Structures by the Temples was used as the example [27].

4.2 Prototype 2

The project underwent 2 major prototype changes following the Gregynog presentations. In both cases considerably different technologies were used. In the first prototype a combination of vanilla JavaScript and jQuery [25] was used to search and play results from two major streaming sites. In the second prototype a more sophisticated system was developed using the AngularJS framework to create a *Single-Page Application* (SPA). SPA in this case being a web application delivered to the browser that doesn't reload the page during use [18].

From the first prototype, it was clear that the project would be best suited to being completely asynchronous, especially when displaying results from multiple sites and playing back content. Because of this, a prototype was built using JavaScript and jQuery to handle communicating between various APIs.

It was at this point that design considerations were taken into consideration. To simplify the front-end design process, a *CSS framework* was selected to more efficiently spend development time. The choice of framework for this prototype was W3.CSS [28]. Although in later builds this CSS framework proved a bit limiting, W3.CSS proved very easy to use and pick up without major design knowledge. Within the context of the second prototype, this was satisfactory during this period.

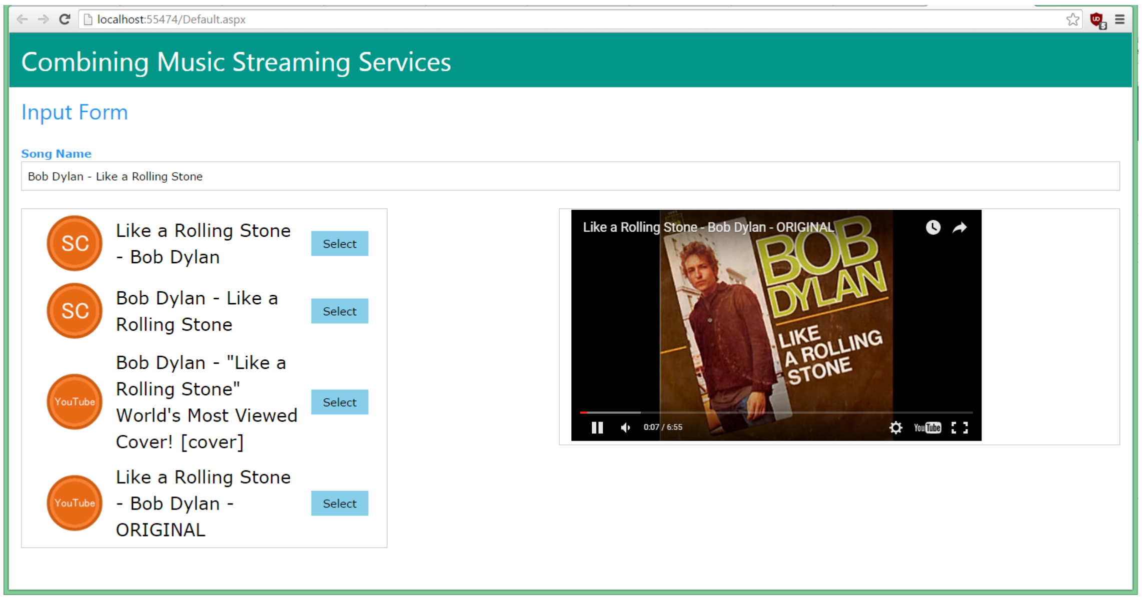
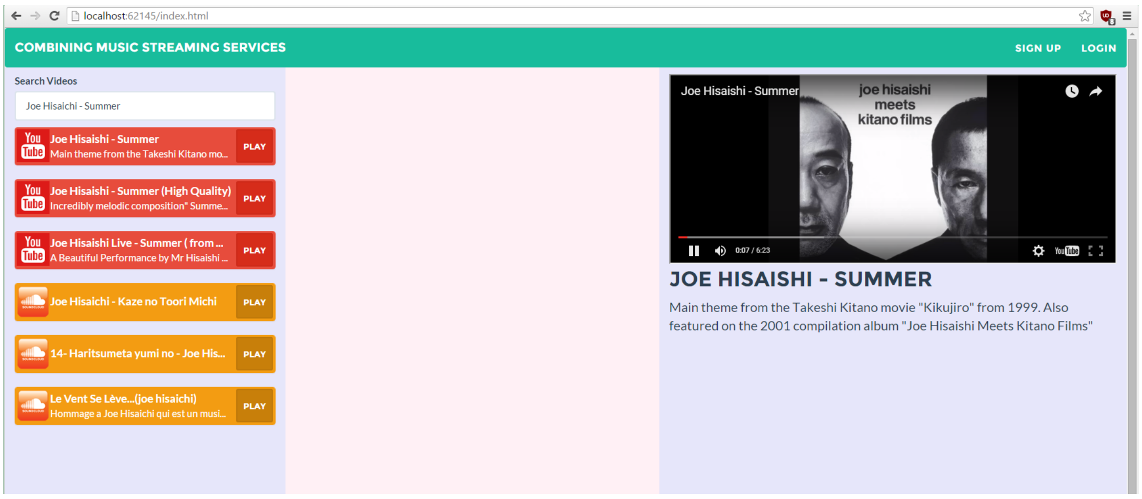

localhost:55474/Default.aspx
Combining Music Streaming Services
Input Form
Song Name
Bob Dylan - Like a Rolling Stone
SC
Like a Rolling Stone - Bob Dylan
Select
SC
Bob Dylan - Like a Rolling Stone
Select
YouTube
Bob Dylan - "Like a Rolling Stone" World's Most Viewed Cover! [cover]
Select
YouTube
Like a Rolling Stone - Bob Dylan - ORIGINAL
Select
Like a Rolling Stone - Bob Dylan - ORIGINAL
0:07 / 6:55

Figure 4.2: Prototype 2 using JavaScript and JQuery.

4.3 Prototype 3

Upon review of the first prototype discussed above, concerns with scalability began to emerge. By relying so heavily on only basic JavaScript to handle all functionality a ridiculous class explosion began to emerge. This meant code was not properly modular and classes meant to handle one aspect of the site were overlapping with others. To solve this, the third prototype was built.

The third prototype was built from the ground up using *AngularJS*, a JavaScript framework which extends HTML vocabulary [11]. This prototype became the final design and implementation of the project. It became increasingly apparent that the entire site would function on one single page at this moment in time. Because of this, a framework centered around building Single-Page Applications, such as AngularJS, was perfect.


localhost:62145/index.html
COMBINING MUSIC STREAMING SERVICES
SIGN UP
LOGIN
Search Videos
PLAY
joe hisaishi meets kitano films
0:07 / 6:23

Joe Hisaichi - Summer

Joe Hisaishi - Summer
Main theme from the Takeshi Kitano mo...

Joe Hisaishi - Summer (High Quality)
Incredibly melodic composition" Summe...

Joe Hisaishi Live - Summer (from ...
A Beautiful Performance by Mr Hisaishi ...

Joe Hisaichi - Kaze no Toori Michi

14- Haritsumeta yumi no - Joe His...

Le Vent Se Lève...(joe hisaichi)
Hommage a Joe Hisaichi qui est un musi...

Joe Hisaishi - Summer

JOE HISAISHI - SUMMER

Main theme from the Takeshi Kitano movie "Kikujiro" from 1999. Also featured on the 2001 compilation album "Joe Hisaishi Meets Kitano Films"

Figure 4.3: Prototype 3 using the AngularJS framework. Summer by Joe Hisaichi is used as the search query [13].

AngularJS worked well here because of its support for MVC style web pages. In figure 4.3 above, functionality was separated into three very clear modules: searching, playlists and the video player. HTML was also separated out into different fragments for each section. This made debugging simpler, as both design and logic for one fragment can be modified without risk of breaking another.

Taking advantage of *controllers* and *services* within AngularJS also allowed for straightforward communication between each fragment. An AngularJS controller is a regular JavaScript object which controls the data of AngularJS applications. Services are javascript functions and are responsible to do a specific task only. Results from each API are handled by a central controller, which is then sent to a single service to allow all results to be treated as one universal object. This object can be easily extended to include fields such as the title, description, and view count - as demonstrated by the right section of 4.3.

Similar to the first prototype, a small consideration has been given to the design. Because of its easy integration with AngularJS and large community support, Bootstrap was used in place of W3.CSS. The change proved a great success with support for all screen sizes, including mobile [24].

For a more in-depth discussion on each previous prototype, refer to the projects interim document [31].

Chapter 5

Design and Implementation

5.1 Methodology

The chosen approach for building the web application followed the incremental methodology. In this approach, the requirements for the web application are divided into various builds. Multiple development cycles take place in a "multi-waterfall" like style. These cycles can be broken down into smaller sub-tasks, such as playlists or searching. Each of these cycles were then broken down into a design, implementation and testing phase [7].

Given the multiple deliverables expected of the project, such as the initial document and interim report [30] [31], an incremental model which produces working builds after each cycle was ideal. A working build is produced right from the first finished increment, with each cycle adding new features. This continues until a complete system is produced. The incremental approach is also flexible by incorporating user feedback during each finished cycle, allowing changes to easily be made along the way. Shorter development cycles also reduces risk by allowing each new feature to be isolated and tested separately.

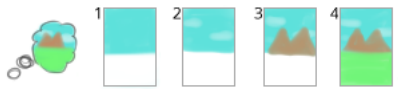


Figure 5.1: Incremental Development Illustrated

5.1.1 Other methodologies

Numerous other methodologies were considered, but were not as appropriate for this project. One such model was the classic waterfall model. In this model we use one linear cycle where each phase must be completed fully before the next phase can begin. This type of model is suited best to small projects with no uncertain requirements. Because this is a fairly large project with some uncertain elements, the waterfall model did not make sense to use.

Iterative models such as the spiral model, which puts a higher emphasis on risk analysis, would not have been suitable either. With an iterative model, we focus more on partially implementing the finished product over time, as opposed to regularly producing working prototypes. In the spiral model specifically, a thorough design is completed before any implementation is produced. Given the multiple deliverables expected throughout the project, this was not realistic.

5.2 Increments

In the project specification, 5 features were recognised that the application must include in order to be useful. This was used as a basis for the increments used in the methodology. Major deliverables with concrete deadlines were also known prior to development which were considered when designing increments. These deliverables included:

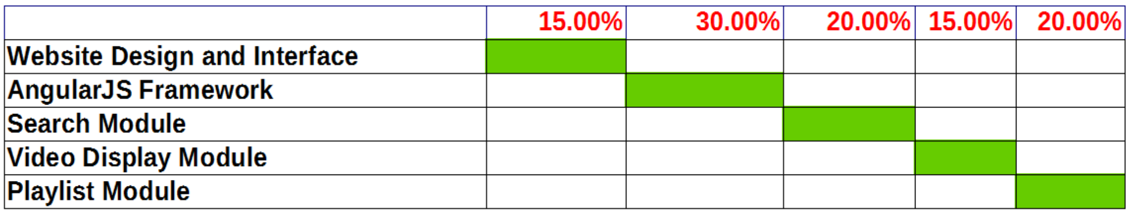
- **Gregynog Presentation** - 19th October 2015.
- **Interim Report** - 15th February 2016.
- **Final Dissertation** - 9th May 2016.

With the above knowledge considered, the following increments were chosen:

- **Increment 1** - Web design and user interface.
- **Increment 2** - AngularJS Framework.
- **Increment 2** - Search Module.
- **Increment 3** - Video Display Module.
- **Increment 4** - Playlist Module.

5.2.1 Rough Time Schedule

To give an overview of the time which was devoted to each increment, a simplified Gantt Chart is provided below in figure 5.2. This chart shows the percentage of the projects development time devoted to each increment.



	15.00%	30.00%	20.00%	15.00%	20.00%
Website Design and Interface	🟩				
AngularJS Framework		🟩			
Search Module			🟩		
Video Display Module				🟩	
Playlist Module					🟩

Figure 5.2: Simplified Gantt Chart showing time devoted to each increment.

It should be noted that this gantt chart only reflects the work schedule following prototype 3 onwards from the previous prototypes chapter. The work schedule prior to this did not closely follow the final methodology choice, but instead focused on researching and testing deployable demo builds. A more detailed discussion of the work schedule breakdown can be found in the project's interim document [31].

5.3 Web Design and User Interface Increment

The first increment focused on establishing the structure and initial design of the web application. Although design is not a big focus of this project, establishing an effective design is still important. Given that this project was designed as a Single-Page Application, all content was designed to fit on one page. With the increasing use of smart phone browsing [21], it is important the interface scale with all screen sizes.

5.3.1 Requirements

- Create an overall structure to the website.
- Produce a sensible and effective design scheme.
- Ensure content can scale properly on a range of devices.
- Clear and consistent font choice.

5.3.2 Overall Structure

When creating the website, the structure was kept simple and summarised into three major sections: searching, video display and playlists. A graphical example of this structure is given below:

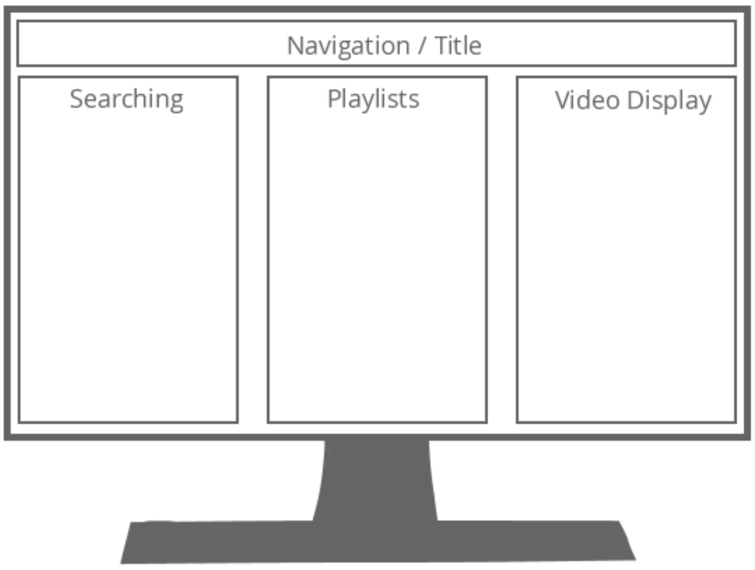


Figure 5.3: Overall website structure.

Sensible and effective design scheme

After the initial structure was established a combination of two Bootstrap templates was used to establish the overall design. Bootstwatch's Flatly theme [4] was used for its simple but effective flat design. This was used in combination with Start Bootstrap's Freelancer theme for its better fonts [3].

Using Bootstrap for this project has meant less time was needed establishing the design, which was not a major focus of this project. Supporting a range of screen sizes and treating each module separately was also made simpler using Bootstrap's grid system.

The bootstrap grid system works by separating rows by up to 12 columns across the page. You can also group columns together to make wider columns for different areas. In figure 5.5, 3 columns was given to the search module, 4 to the playlists, and 5 to the video. This gives a total of 12 to span the whole page. Different amounts of space depending on the screen size can be defined using the col properties. For example, col-md-8 would be used to devote 8 columns of space on medium screen sizes and above.

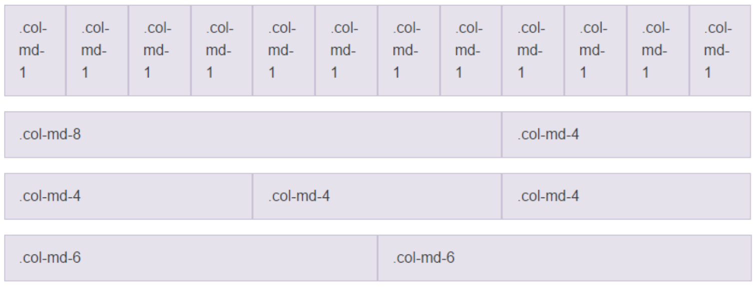


Figure 5.4: Bootstrap's Grid System, as presented by the Bootstrap website [23]

Scale properly on a range of devices

Using Bootstrap's grid system described above, content can be easily modified depending on the screen size. This is done by linking the desired number of columns and the size class together. As an example, the code below dedicates 3 columns on extra small devices, and 6 columns on small devices upwards.

```html
<div class="col-xs-3 col-sm-6"> </div>
```

Clear and consistent font choice

Using Start Bootstrap's Freelancer theme as the basis for font choice, Montserrat is used to give a clean but impactful affect. The final result of this increment is shown below in figure 5.5. With the necessary design structure in place, the remaining increments focused on implementation.

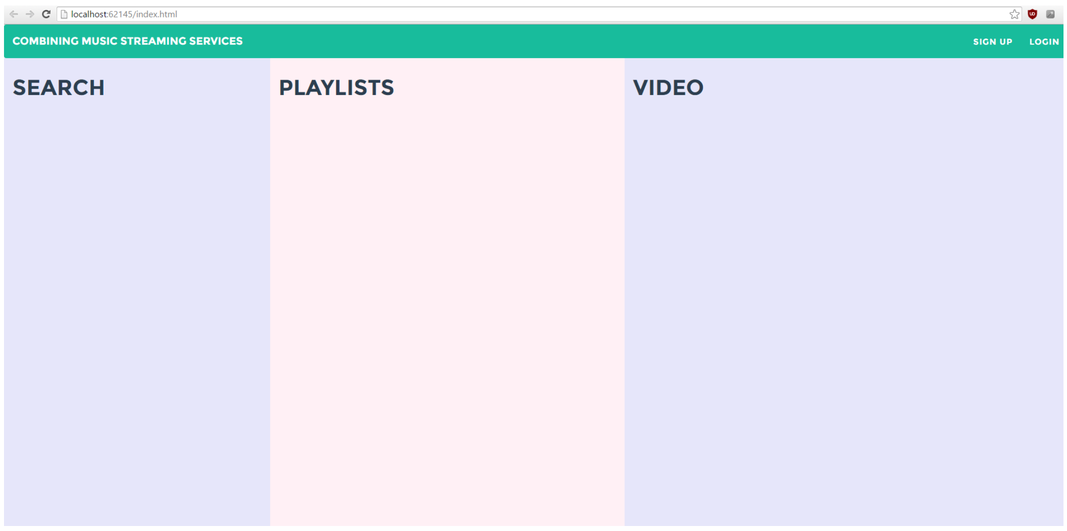

localhost:62145/index.html
COMBINING MUSIC STREAMING SERVICES
SIGN UP
LOGIN
SEARCH
PLAYLISTS
VIDEO

Figure 5.5: Overall website structure.

5.4 AngularJS Framework

During the projects development, it became increasingly apparent that the entire site would function on one single page. Because of this, a framework designed for building Single-Page Applications was used. Instead of relying on only basic JavaScript to handle all functionality, the framework of the project was built using AngularJS, a JavaScript framework which extends HTML vocabulary [11].

AngularJS works well in this project because of its support for MVC style web pages. In figure 5.5 above, the site clearly separates out into three separate modules. Doing it in this way allows for much simpler testing and debugging, as each increment can be isolated and handled separately. HTML can also be easily separated out into three separate fragments, allowing easy design changes without risk of affecting logic code.

5.4.1 Requirements

- Create folder structure for the project.
- Isolate HTML into three fragments (searching, playlists and videos).
- Create an AngularJS Controller.
- Create an AngularJS Service.

Create folder structure for the project

To better organise the projects files, a folder structure was established at the start of this increment.

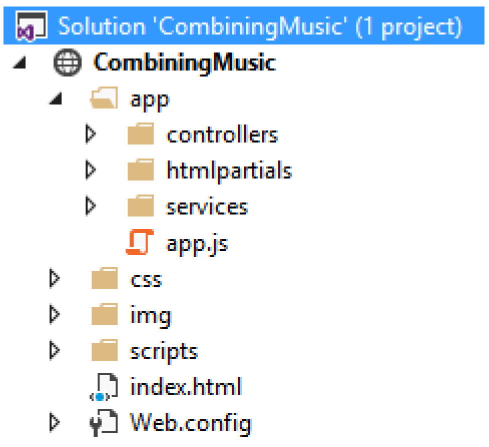


Figure 5.6: Project file structure.

Within Visual Studio a new project was set up with 4 major folders used. The *"app"* folder stores all AngularJS related folders, including its controllers, services and HTML partials. A second folder for all *CSS* related code was made to keep design code separate from logic code. To easily reference images from anywhere in the project, such as logos, an *"img"* folder was made. The final folder *"scripts"* holds any miscellaneous scripts, such as quick JavaScript functions to handle layout.

Isolating HTML into three fragments

One of the biggest benefits of AngularJS is being able to easily break up HTML into multiple files, while still displaying it as one page at runtime. At this point in the increment, four HTML partials were created. With this change the navbar, searching, playlists and videos tab can be easily modified separately.

To easily control the order and size of each fragment, a Bootstrap *container* was used to hold all fragments together. A bootstrap container is required to confine HTML into dedicated amounts of space. In appendix 1, the “container-fluid” class provides a full width container, spanning the entire width of the viewport. Using the Bootstrap grid system (discussed in the previous increment), the amount of columns allowed by each section changes whether viewing on an extra-small screen, such as a smartphone, or any larger screen.

Create an AngularJS Controller

To handle connection between HTML input and the AngularJS service, a *controller* was created. In AngularJS, a controller is a regular JavaScript object which can asynchronously provide update fields based on input. In the below example, a controller called “YouTubeController” was created which connects to the service “YouTubeService”. It holds three variables which can be updated asynchronously.

```
mainApp.controller('YouTubeController', function (\$scope, \$http, \$log,
    YouTubeService) {
    $scope.data;
    $scope.dataSoundCloud;
    $scope.playlist;
}
```

Create an AngularJS Service

To easily organize and share song objects across the program, an AngularJS *service* was made. Because each site’s API has a unique structure to its song objects, a service was created to standardise this. The service contains functions for setting and retrieving a custom song object. The object contains four fields:

- title - the title of the song.
- description - any description field the site provides.
- video id - a unique identifier string the site uses to differentiate songs.
- site - the site the song originally came from.

With the service and controller set up as it was, additional features can easily be stored and standardised within the service, such as a rating system. Additional functions were also created within the service to track things such as the currently playing selected song. A snippet of the service created for this section is provided below.

```
mainApp.service('YouTubeService', ['$http', '$log', function ($http, $log)
    {
    var results = [];
    this.getResults = function () {
        return results;
    };
}
```

5.5 Search Module

In order to easily search and retrieve songs from a range of sites, this increment focused on both retrieving and responding to user input asynchronously, as well as handling API calls to each site in order to retrieve results.

5.5.1 Requirements

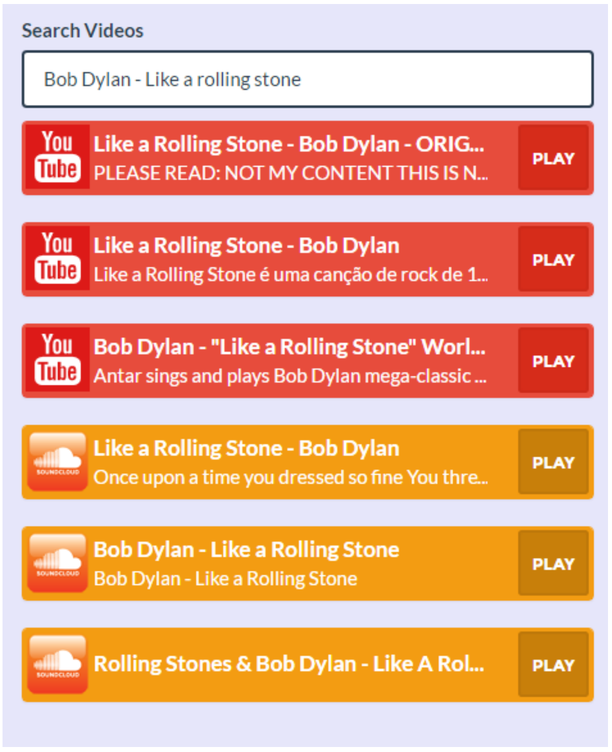
- Design the HTML for the search fragment.
- Set up API calls for two sites.
- Asynchronously retrieve top results.

Search increment HTML fragment

Because the HTML was separated out into individual fragments using Bootleg and AngularJS in increment 1, design can be kept conveniently into the search fragment. A basic HTML input was used which linked to a function to call all APIs in one sweep.

Returned results were designed so that each site could be recognised by a unique colour which closely matches the websites theme. As the two sites used for the minimum feature set, YouTube and SoundCloud were given a light red and orange respectively. These colours closely match the color of their logos.

By using the four fields stored within the AngularJS service, returned results could easily be displayed. The site name could be used to pick the correct logo and colour; the title is used to display a bold title, and the description field to give a shortened description beneath. To later play results a play button was also added in preparation.


Search Videos

Bob Dylan - Like a rolling stone

Like a Rolling Stone - Bob Dylan - ORIG...
PLEASE READ: NOT MY CONTENT THIS IS N...
PLAY

Like a Rolling Stone - Bob Dylan
Like a Rolling Stone é uma canção de rock de 1...
PLAY

Bob Dylan - "Like a Rolling Stone" Worl...
Antar sings and plays Bob Dylan mega-classic ...
PLAY

Like a Rolling Stone - Bob Dylan
Once upon a time you dressed so fine You thre...
PLAY

Bob Dylan - Like a Rolling Stone
Bob Dylan - Like a Rolling Stone
PLAY

Rolling Stones & Bob Dylan - Like A Rol...
PLAY

Figure 5.7: Final design of the search increment.

API calls for two sites

Because each site's API is slightly different, a separate function for each was required. To call each API together, a function was made that simply calls all other functions, each adding to the same array of song objects. This method has a single parameter for the search query, linked to the input field within the HTML fragment.

For the **YouTube API**, a JavaScript SDK is provided for easily returning a specified number of results. The search query is provided from the method described above. Because YouTube also allows for playlist and channel searching, results had to be restricted to videos only using the "video" type. The final returned result is a JSON object with countless unnecessary information for this project. To solve this problem, the function within the service can once again be used to convert this to a song object. The parameters used were as such:

```
addResult(items.snippet.title, items.snippet.description, items.id.videoId,
    "YouTube");
```

For the **SoundCloud API**, a similar SDK is provided. Working on RESTful principles, a simple HTTP GET request is all that was required to return a list of songs from the SoundCloud API. The results returned was limited to 3, but this could have been easily changed. Similar to YouTube's API, results were in the form of JSON objects, the only difference was that results were bundled into an array. The same function was used from the AngularJS service to convert to song objects, with slightly different input parameters:

```
addResult(tracks[i].title, tracks[i].description, tracks[i].permalink_url,
    "SoundCloud");
```

Asynchronously Retrieve Top Results

By the nature of AngularJS being a Single-Page Application, data from the controller can be automatically handled asynchronously using the double brace notation. For e.g. {{ person.forename }} will always display the up-to-date forename of the person object as soon as it changes. Using the "ng-repeat" property within AngularJS, the array of song objects retrieved from the APIs above was easily styled consistently. A small sample of this is provided below:

```
<div ng-repeat="result in data">
    <h3> {{result.snippet.title}} </h3>
</div>
```

5.6 Video Display Module

To display embedded media players within the site, an increment with the goal of translating song objects into displayed content was necessary.

5.6.1 Requirements

- Track currently chosen song with AngularJS service.
- Display chosen song's media player for both sites.
- Design simple playlist control for the playlist module.
- Add relevant title and description for chosen song.

Track currently chosen song

To track the currently chosen song with AngularJS, a new function was created which simply holds a single song object. The object can then be identified by its video id field, which uniquely identifies a particular song conveniently. From this point, the current song was asynchronously displayed by AngularJS at all times.

Display chosen song's media player

To display rich media content easily, an HTML5 iframe was set to always display a media feed. The HTML for this was as such:

```
<iframe ng-src="{{videoURL}}" style="width:100%; height:35vh; margin-bottom
    :0px;"></iframe>
```

As can be seen from the "ng-src" tag, the url in which to retrieve the media is defined as a variable. By once again using the AngularJS double brace notation, the current video can be asynchronously updated at any point that the "videoURL" variable is changed.

Using the song object created earlier in this increment, the current song's title and description was easily displayed within the video HTML fragment. By creating a song object, there was no need to construct endless switch statements to grab the description for each individual site's different API.

Playlist controls

In preparation for the playlist increment, three buttons were prepared which offer very basic playlist controls: "back", "forward" and "shuffle". The "back" and "forward" buttons can be used to change the current video to the song that appears either before or after the current song respectively. The shuffle button would offer some means of randomising the current playlist.

Difficulties hooking APIs to iFrame

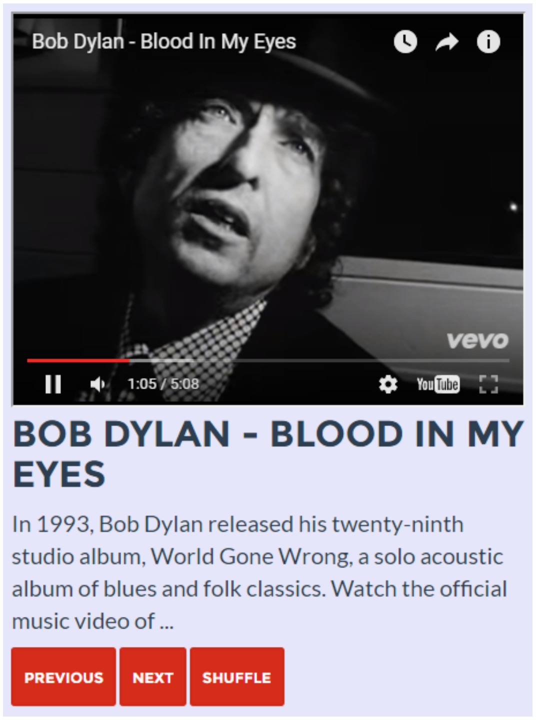
One difficult challenge which arose during the video module was attempting to receive an event from each API when a video had finished. This was important so that when a song finishes, the next one in the playlist would start. Each API had its own unique way of broadcasting this event, but in all cases the API had to be linked to an *iFrame* (short for inline frame). An iFrame in HTML is simply used to embed content from other websites onto the page, in this case a media player.

Originally, regardless of which site's widget was being displayed, the same iFrame was used. This was no longer possible as the YouTube API cannot link to an iFrame hosting SoundCloud music as an example. Trying to listen out for the "song finished" event from all sites at once caused multiple errors. The solution to this was to create a new iFrame for each site, but keep all but the appropriate site hidden.

```
<div>
    <iframe id="player" ng-if="currentVideo.site == 'YouTube'" ng-src="{{videoURL}}" style="width:100%; height:35vh; margin-bottom:0px;"></iframe>
    <iframe id="playerSoundCloud" ng-if="currentVideo.site == 'SoundCloud'" ng-src="{{videoURL}}" style="width:100%; height:35vh; margin-bottom:0px;"></iframe>
...
```

This solved the problem as each site's API could listen out for the event within its own iFrame. An unfortunate issue that arose from this solution was multiple undesirable if statements, which put a small dent to the program's structure.

The final result of this increment can be seen below in figure 5.8.


Bob Dylan - Blood In My Eyes

BOB DYLAN - BLOOD IN MY EYES

In 1993, Bob Dylan released his twenty-ninth studio album, World Gone Wrong, a solo acoustic album of blues and folk classics. Watch the official music video of ...

Figure 5.8: Final design of the video increment. An official upload of Bob Dylan's "Blood In My Eyes" was used as the example [1].

5.7 Playlist Module

The final increment of the project was the playlist module. The purpose of the playlist module was to replicate basic operations expected of a generic music streaming site. The big difference with this website's playlist function is that it mixes results from multiple sites.

5.7.1 Requirements

- Store a simple playlist array in AngularJS.
- Design and display playlist entries.
- Connect the "previous", "next" and "shuffle" buttons from the video increment.

Store a simple playlist array in AngularJS

In order to store playlists a simple array of song objects was created, with new entries appended to the end of the array. The "Add" button on search results was a very simple way to add new songs, calling a function to append the selected song.

```
var playlist = [];

this.addToPlaylist = function (title, description, videoId, site) {
        playlist.push(
            {
                title: title,
                description: description,
                videoId: videoId,
                site: site
            });
};

this.getPlaylist = function () {
    return playlist;
}
```

Design and display playlist results

The design of each playlist entry was kept consistent with the design for each search result. This can be seen in figure 5.9. The main difference that can be seen is a "Remove" button instead of an add button, and the index of each song is also displayed.

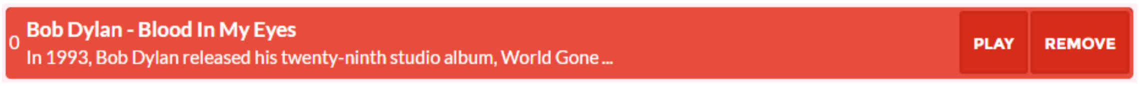

0
Bob Dylan - Blood In My Eyes
In 1993, Bob Dylan released his twenty-ninth studio album, World Gone ...
PLAY
REMOVE

Figure 5.9: A single entries design for the playlist feature.

Connecting the "previous", "next", and "shuffle" buttons

Autoplay

In order to smoothly transition from one song to the next, the "song finished" events from the video increment were used. A special function was created which triggers whenever any of these events are detected. Both the service and controller within AngularJS are notified to update and display the next entry in the playlist. If the end of the playlist is reached, the function simply stops there.

The functions used to go forward and backward in the playlist are given below. The general concept is to simply increment or decrement the index counter by 1 (assuming this is possible).

Playing the next song

```
function playNextSong() {
    for (var i = 0; i < playlist.length; i++) {
        if (playlist[i].videoId == currentVideo.videoId) {
            if (i == playlist.length - 1) { \\do nothing }
            else {
                currentVideo = playlist[i + 1];
                break;
            }
        }
    }
}
```

Playing the previous song

```
this.previousSong = function () {
    for (var i = 0; i < playlist.length; i++) {
        if (playlist[i].videoId == currentVideo.videoId) {
            if (i == 0) { } // first song, so do nothing
            else {
                currentVideo = playlist[i - 1];
                break;
            }
        }
    }
}
```

Connecting the back and forward buttons

Because of the preparation done in the video module, connecting the buttons from the previous module was straightforward. For the forward button, the method used for the autoplay feature was called, since they both perform the same task. For the “previous” button, a method that is almost identical to the forward method, but decrements the array index instead, was used.

Shuffle Feature

For the shuffle feature, a simple implementation of the Fisher-Yates shuffle was used [9]. In simple terms, the algorithm effectively randomly picks the next element in the list one by one. After an element is picked it can no longer be randomly chosen. This continues until no elements remain.

```
function shuffle(array) {
    let counter = array.length;
    while (counter > 0) {
        let index = Math.floor(Math.random() * counter);
        counter--; // Decrease counter by 1
        let temp = array[counter]; // And swap the last element with
            counter
        array[counter] = array[index];
        array[index] = temp;
    }
    return array;
}
```

The functionality of the shuffle button uses a similar design to YouTube's shuffle feature. In this method, the first press of the button presents a randomly shuffled list, and a second press returns to the original list. This all happens asynchronously, so the currently playing song will naturally transition to the next song on the list. This correctly recognises whether the next song is shuffled or not. For example, a playlist [1,2,3] might randomly be shuffled to [3,1,2], but a second press of the button returns it to [1,2,3].

5.7.2 Final Design

The final design of this module can be seen below in figure 5.10. Each entry is clearly displayed in a consistent manner to the rest of the site. Note that the currently playing song is highlighted in a brighter red.

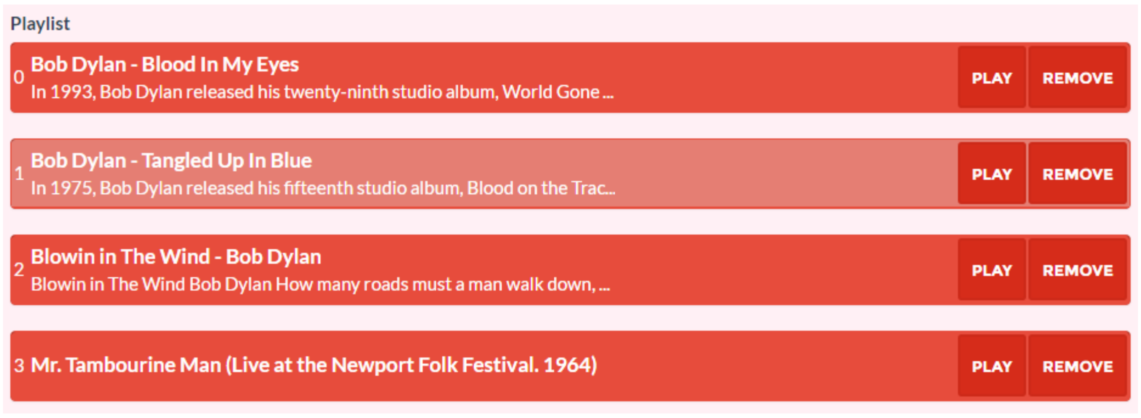

Playlist
0
Bob Dylan - Blood In My Eyes
In 1993, Bob Dylan released his twenty-ninth studio album, World Gone ...
PLAY
REMOVE
1
Bob Dylan - Tangled Up In Blue
In 1975, Bob Dylan released his fifteenth studio album, Blood on the Trac...
PLAY
REMOVE
2
Blowin in The Wind - Bob Dylan
Blowin in The Wind Bob Dylan How many roads must a man walk down, ...
PLAY
REMOVE
3
Mr. Tambourine Man (Live at the Newport Folk Festival. 1964)
PLAY
REMOVE

Figure 5.10: Final design for the playlist module.

5.8 Final adjustments and bug fixes

After all of the increments above were finished and implemented, a final working version was achieved. At this point, some final last minute improvements were made and some minor features added.

5.8.1 Saving/Loading Playlists

For demonstration purposes, a very simple way to save and load one single playlist was made. The method in which this was done was through *HTTP cookies*. An HTTP cookie is a small piece of data sent from a website and stored to the user's browser client-side. Every time the user loads the site, the browser sends the cookie back to the server to load the user's previous activity. In this case one cookie storing a playlist array is used.

Using AngularJS, managing cookies was made incredibly simple using the angular-cookies module provided by Google [10]. The module offers simple cookie management commands which could be used within a controller. Two very straightforward methods were created with this, one to save and one to load.

Saving

```
$scope.save = function () {
        $cookieStore.put('playlist', YouTubeService.getPlaylist());
}
```

Loading

```
$scope.load = function () {

    // Simple load to a new object
    var playlistSongs = $cookieStore.get('playlist');

    // Set the current playlist blank in preparation
    YouTubeService.setBlankPlaylist();

    // Each track is added one by one to avoid variable type conflictions
    for (var i = 0; i < playlistSongs.length; i++) {
        YouTubeService.addToPlaylist(playlistSongs[i].title, playlistSongs[
            i].description, playlistSongs[i].videoId, playlistSongs[i].site
            );
    }

    // Updates the controller playlist to match the service playlist
    $scope.playlist = YouTubeService.getPlaylist();

}
```

As can be seen, the saving method is effectively one line of code inside a function. The loading function was also simple, but was made slightly more complex by conflicting data types. To fix this each object was manually added one by one, and both the controller and service notified afterwards. Because of careful organisation to the project's structure, the inclusion of this method was straightforward with no alterations to other parts of the site necessary.

Smaller screen fix

A few small CSS issues were present which only appeared when displayed on smaller screen sizes. This included the buttons for the search results overlapping with the backgrounds for the playlist section. To fix this the amount of space dedicated to each section was adjusted to give more room to the search module, and less to the video module. Some small CSS mistakes were also found and corrected which resulted in this overlapping.

Slight Colour Tweaks

The colours of the background and navbar were tweaked slightly to give a more complete complimentary colour scheme. The colour scheme revolves around high saturation colours for the buttons of the site. Following some simple colour theory principles, the colours of the background have been given a lower saturation to compliment this [34]. The exception to this was the navbar, which was kept green to balance out the sites colour palette. The improved colours can be seen at the end of this chapter in figure 5.12.

Testing

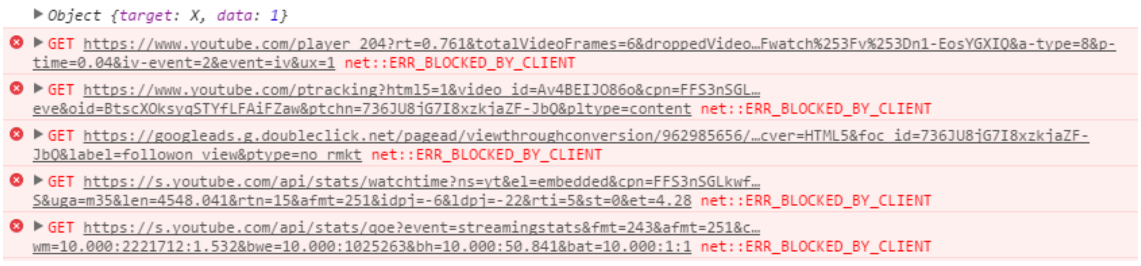
Although the incremental model for this project was followed very closely, testing was an exception. Normally testing in an incremental model is done at the end of each module. This proved ineffective for this project. The first reason for this is the high level of communication between each module. Most errors or conflictions would generally only arise when attempting to communicate between different modules. The other major issue was the many unpredictable circumstances caused by juggling multiple APIs. On a regular basis a normally simple task would result in strange behaviour caused by each API handling data in its own way.

Testing was done at the end of the project, using concepts from decision table style testing [16]. In simple terms, one test case is made for each input scenario that might occur. For this project, this meant every unusual interaction between APIs was tested. Because the site's focus was not to produce a fully functioning music site, but more a system to easily manage multiple APIs, the end result has not been noticeably affected by this loose testing strategy which strayed from the incremental methodology.

Minor Warnings

After multiple iterations of testing, no major bugs were found in the program but some warnings were noticed. Upon review of the console window within a chrome browser, warnings at regular intervals could be found. This was eventually found to all be linked to chrome extensions which were unrelated to the project. Research suggested HTTP warnings of this nature are expected when dealing with RESTful web calls. Viewing the console window on YouTube, SoundCloud and Spotify all gave similar warnings.

Extensive use of the site has suggested these warnings have no impact on the final products functions and behave as expected regardless.

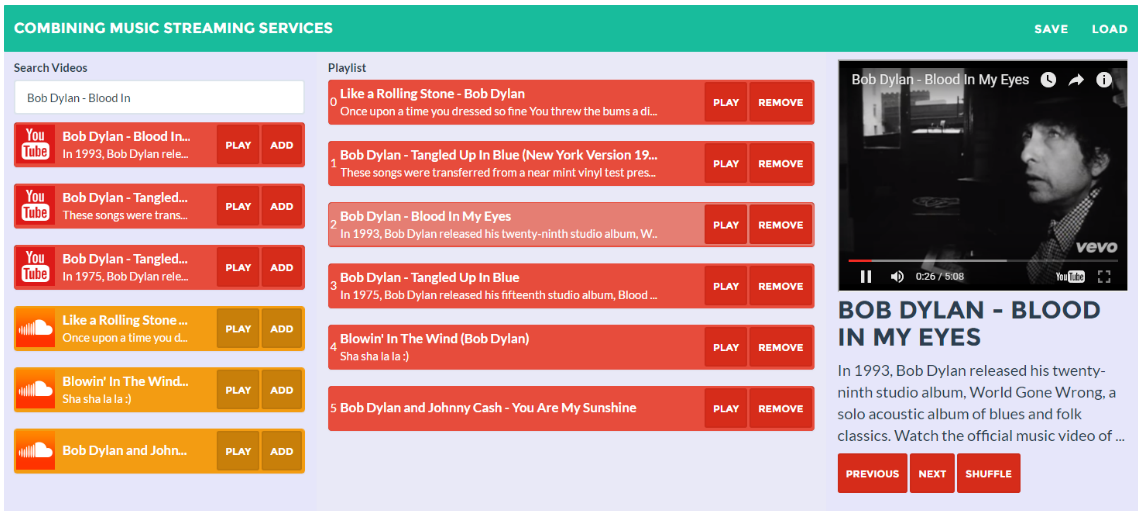


```
▶ Object {target: X, data: 1}
▶ GET https://www.youtube.com/player_204?rt=0.761&totalVideoFrames=6&droppedVideo…Fwatch%253Fv%253Dn1-EosYGXIQ&a-type=8&p-
time=0.04&iv-event=2&event=iv&ux=1 net::ERR_BLOCKED_BY_CLIENT
▶ GET https://www.youtube.com/ptracking?html5=1&video_id=Av4BEIJO86o&cpn=FFS3nSGL…
eve&oid=BtscXOksyqSTYfLFAiFZaw&ptchn=736JU8jG7I8xzkjaZF-JbQ&pltype=content net::ERR_BLOCKED_BY_CLIENT
▶ GET https://googleads.g.doubleclick.net/pagead/viewthroughconversion/962985656/…cver=HTML5&foc_id=736JU8jG7I8xzkjaZF-
JbQ&label=followon_view&ptype=no_rmkt net::ERR_BLOCKED_BY_CLIENT
▶ GET https://s.youtube.com/api/stats/watchtime?ns=yt&el=embedded&cpn=FFS3nSGLkwf…
S&uga=m35&len=4548.041&rtn=15&afmt=251&idpj=-6&ldpj=-22&rti=5&st=0&et=4.28 net::ERR_BLOCKED_BY_CLIENT
▶ GET https://s.youtube.com/api/stats/qoe?event=streamingstats&fmt=243&afmt=251&c…
wm=10.000:2221712:1.532&bwe=10.000:1025263&bh=10.000:50.841&bat=10.000:1:1 net::ERR_BLOCKED_BY_CLIENT
```

Figure 5.11: Console warnings resulting from chrome extensions.

5.9 Final site design

The finished design for the site can be seen below in figure 5.12. The end look and feel of the site is a professional but still approachable designed site. With a high level of functionality and clear 3 column layout, the finished product reflects a high quality music streaming site.



Bob Dylan - Blood In

Bob Dylan - Blood In... In 1993, Bob Dylan rele... PLAY ADD
Bob Dylan - Tangled... These songs were trans... PLAY ADD
Bob Dylan - Tangled... In 1975, Bob Dylan rele... PLAY ADD
Like a Rolling Stone ... Once upon a time you d... PLAY ADD
Blowin' In The Wind... Sha sha la la :) PLAY ADD
Bob Dylan and John... PLAY ADD

0 Like a Rolling Stone - Bob Dylan
Once upon a time you dressed so fine You threw the bums a di... PLAY REMOVE
1 Bob Dylan - Tangled Up In Blue (New York Version 19...
These songs were transferred from a near mint vinyl test pres... PLAY REMOVE
2 Bob Dylan - Blood In My Eyes
In 1993, Bob Dylan released his twenty-ninth studio album, W... PLAY REMOVE
3 Bob Dylan - Tangled Up In Blue
In 1975, Bob Dylan released his fifteenth studio album, Blood ... PLAY REMOVE
4 Blowin' In The Wind (Bob Dylan)
Sha sha la la :) PLAY REMOVE
5 Bob Dylan and Johnny Cash - You Are My Sunshine PLAY REMOVE

Bob Dylan - Blood In My Eyes
0:26 / 5:08

BOB DYLAN - BLOOD IN MY EYES

In 1993, Bob Dylan released his twenty-ninth studio album, World Gone Wrong, a solo acoustic album of blues and folk classics. Watch the official music video of ...

Figure 5.12: Final design for the full website. An official upload of Bob Dylan's Blood in My Eyes is used as the search query [1].

Chapter 6

Testing and Evaluation

6.1 Introduction

The final chapter of this project is dedicated to evaluating the finished solution. Using a systematic approach, the three critical modules of the site will be tested using a common software testing method. A discussion of ways in which the project could be extended for future-work is also included. Finally, a personal reflection of the project is given.

6.2 Procedure

The chosen evaluation strategy for this project is a user based evaluation. User based evaluation is a technique which aims to collect information regarding how much a system meets user goals, expectations and standards [20]. Because many features of a web application based project does not measure well in terms of numerical input, it is more beneficial to evaluate the system based on user interaction methods. This could include methods such as questionnaires, interviews and direct observation. This form of evaluation allows a system to be judged in a more realistic setting from the perspective of people who will potentially use and benefit from the system [20].

Participants will be asked to undertake a set of tasks relating to all three modules of the website. This includes searching, playlist management and the video feed. An important aspect of a user study is that tasks must be representative of those for which the system was intended for [15]. In this case streaming and combining playlists from a range of sites. The tasks given reflect the stages involved in a typical day on a music streaming site.

When completing the task, users will be encouraged to voice their opinions of each section and leave verbal feedback [15]. Notes of each user's thoughts will be taken to refer to in future. While 5 participants is not a large enough sample size to come to any conclusions, it does give a strong look at the system in a real setting.

6.3 Questionnaire

Users will subsequently be asked to fill out a short questionnaire after completing all tasks, rating their experience on a five point scale. A questionnaire was chosen in order to easily convey and confirm the overall average opinion of using the site. The questionnaire uses a 5 point Likert scale [17]. Using a five point scale is a common strategy in order to measure attitudes of users towards a topic. A scale between 1 and 5 still leaves adequate room to differentiate opinion, while still retaining clarity in meaning.

The questionnaire aims to gather information about the functionality, usability and overall happiness of the website's three major modules.

6.4 Results

6.4.1 Verbal Feedback from user based evaluation

Table 6.1: Verbal feedback from participant 1

Task 1	N/A.
Task 2	It's really hard to make out the red areas for me.
Task 3	I have no idea what song to choose.
Task 4	N/A.
Task 5	Works nice!
Task 6	Seems to work pretty smoothly.
Task 7	Oh, I have to hover over the video to see options.

Table 6.2: Verbal feedback from participant 2

Task 1	The site looks nice, but the green navbar is a bit out of place.
Task 2	The SoundCloud results seem a little delayed compared to YouTube.
Task 3	I like the description and title underneath the video.
Task 4	Can I not reorder the list?
Task 5	Saving and loading seems to work well.
Task 6	N/A.
Task 7	I like that you just used the YouTube player and not some awful custom one.

Table 6.3: Verbal feedback from participant 3

Task 1	The layout is nice, but the colour scheme could maybe be improved.
Task 2	Does it only work for two sites?
Task 3	I have to click a few times [on the play button] for it load.
Task 4	It's a little odd the numbers start at 0.
Task 5	I like how I don't have to bother making an account or anything.
Task 6	N/A.
Task 7	It's annoying that you can't change the volume [for SoundCloud].

Verbal feedback continued

Table 6.4: Verbal feedback from participant 4

Task 1	Well it sure is colourful.
Task 2	I can't find quite a few songs.
Task 3	Seems to play nicely from what I could find.
Task 4	Nice playlist though a bit minimal.
Task 5	At least it saves nicely.
Task 6	The playlist controls are pretty neat, reminds me of the YouTube playlists.
Task 7	It's nice that it's just the same players for the sites.

Table 6.5: Verbal feedback from participant 5

Task 1	The sites layout is really nice, but something just feels off about the colours.
Task 2	Why only three results per site?
Task 3	Plays nicely when you eventually find the song
Task 4	It's good but I wish I could reorder it.
Task 5	Umm... the play button stopped working?
Task 6	Well at least I can still use the playlist controls.
Task 7	N/A.

6.4.2 Questionnaire Results

Table 6.6: Questionnaire Results

Question	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Participant 1	4	3	4	3	3	4	5	4	5	4
Participant 2	4	5	4	3	5	4	5	4	5	3
Participant 3	4	4	3	4	5	5	4	5	5	4
Participant 4	4	5	4	2	5	5	5	5	5	4
Participant 5	5	3	3	2	4	5	1	5	5	3

6.5 Verbal Feedback and Questionnaire Evaluation

From the small sample of participants used, feedback was generally very positive. As a summary, all participations agreed the site had a strong, intuitive design and layout. Responses for the playlist and video functions were very positive with one exception for saving and loading playlists. Responses regarding the search feature were less positive.

Search evaluation

Although most users found the search feature to be reasonably intuitive, some expressed that the results gave fairly poor results. The main reasons given included the relatively small pool of results returned and the condensed descriptions of results. One observation found from comparing the users who had poor search results to those who had better results was search styles. Participants who preferred a more precise search style had reasonable results, whereas those who preferred searching with keywords had worse results. Participant 1 also noticed a small bug where hitting the play button would not work until a song was added to the playlist. Given more time, parsing text to give more accurate results would be one way to solve this.

Playlist evaluation

Participants gave very positive responses to the playlist feature. It should be noted that participants were told to evaluate the playlist as a very basic solution, and so were not expecting fully featured playlist functionality. Most participants had no trouble saving and loading playlists, however, an unexpected problem occurred for participant 5 where songs would no longer play for an unknown reason. Debugging at a later date showed this to be a very rare occurrence caused by the YouTube API.

Video evaluation

The evaluation of the video player section was similarly positive. Because the videos for each song are direct embeds of popular music players, it is not surprising results are positive. By using the widgets provided by the sites involved, participants enjoyed a large familiarity to the results.

Final evaluation of user based evaluation

The average, mean and standard deviation of the user feedback is given below. This is based off the average score over all 10 questions from the questionnaire.

	Participant 1	**Participant 2**	**Participant 3**	**Participant 4**	**Participant 5**	**Average**
Average	3.9	4.2	4.3	4.4	3.6	4.08
Median	4	4	4	5	3.5	4.1
Variance	0.54444	0.62222	0.45556	0.93333	2.04444	0.92

Table 6.7: Average, median and variance for questionnaire results.

The site received a fairly consistent "agree" over the full site. The average variance of 0.92 also shows that most participants had a reasonably consistent opinion of the sites features. Ultimately the search performed the weakest while the video display performed the strongest.

6.6 Future Work Potential

In its current state, an excellent foundation has been built for handling multiple music streaming APIs into an AngularJS project. By breaking down each sites' results into its basic elements, the basic premise for allowing different music APIs to communicate has been realised. Given more time, the site (or any new site) could be built with a vast quantity of additional features. This could include:

- Advanced playlist features such as reordering and saving multiple lists.
- Importing user created playlists and favourites from supported sites.
- Login system to let users play and share their music from anywhere.
- Advanced searching capabilities with more results and text parsing.
- Increased design and branding.

One major feature that could be added would be a custom media player in which all different sites would use. Audio and video would be streamed through this universal player, with design and media controls kept consistent throughout. This would deal with sites that have widgets that lack basic features, such as the SoundCloud player having no volume control.

To simplify the structure, only two sites were used to test the concept of the project. In a project extending this system, far more sites could easily be supported without having to completely change prior code. Because all songs are reduced to the same song object type, only a simple knowledge of a site's API is needed to hook to a few parts of the site.

With the increasing use of mobile browsing [8], a mobile app which builds upon the same foundation but optimised for a mobile app would prove extremely useful. Because the system is built upon AngularJS, transfering code into a mobile environment would be difficult but not impossible. One way this could be accomplished would be the Mobile Angular UI [6]. This allows applications to be built using AngularJS and Bootstrap, the same as this project.

6.7 Evaluation of the minimal feature set

In the initial document for this project [30], 5 features were recognised as the minimum requirements for the system to be considered useful:

- Accept a textual description of a song as input from the user.
- Dynamically search potential sites based on this input - i.e. find the song that most closely matches the description without a fixed url given.
- Locate songs automatically without visiting potential sites.
- Support results from a minimum of 2 different music streaming services.
- Ability to create a basic dynamic playlist consisting of only a textual description of each song beforehand, with no fixed url to a song used.

At the end of this project, all 5 features have been clearly implemented. Dynamically searching and inputting songs is possible. The closest match for each site is identified without visiting the sites. Results are supported from 2 sites: YouTube and SoundCloud. Basic dynamic playlists are fully implemented.

6.8 Project Evaluation

Background Research

The first few months of this project were dedicated to careful research on similar applications, as well as music streaming APIs.

By researching the APIs of various music streaming sites and creating examples which demonstrated their functionality, a very clear path for how to build the complete system was obtained. Having very little prior knowledge of working with APIs, the time spent in the first few months of the project experimenting with each API was an excellent use of the projects time. Using this time to gain better understanding of using Javascript and HTML in more complex scenarios was also very useful when creating these experiments.

While the background research for this project was overall a success, it could have been improved by further investigating more music streaming sites and perhaps a more broad range of similar applications. One particular mistake in the background research was not thoroughly researching each APIs capabilities. Early in the project Spotify was intended to be the second site supported, but was later found to lack support in its API [5].

Given the time constraints on the project, the level of research performed was still adequate. Overall the background research conducted was sufficient in meeting the projects aims.

Technology Choices

The project began with relatively little knowledge of web development tools, and so starting the project with a fairly industry standard choice of ASP.NET was a reasonable choice. A very difficult decision was made halfway through development where it was clear that ASP.NET was not the best choice for the projects development. Although the swap to an unfamiliar new technology in AngularJS was difficult, reflecting on the finished product suggests this was certainly the right decision.

If the project was to be started from scratch today, it would be easy to say AngularJS would be used from day one. It should noted however, that the experience gained with the simpler ASP.NET was invaluable in being able to understand how websites in AngularJS are built and was certainly a factor to the projects success.

Using Bootstrap for the front-end development also worked out excellently. The website's design was able to be designed very quickly and efficiently while still scoring strong results from the user evaluation. Its strong synergy with AngularJS is further reason that both technologies were the right choice.

Development and Implementation

Given the choice of technology and size of the project, choosing the incremental methodology was a perfect choice. Dividing the website into clear subsections solved a lot of issues with scalability. Dealing with the large difference in how each API handles different tasks still proved to be a difficult task and in the later stages of the code some overlapping of increments could be found.

The specification phase of each increment was one weak area of the projects development. Not enough attention was given to clearly defining the requirements of each increment. A lot of time was wasted during development working around unforeseen requirements from each API. This would have been fixed with more time devoted to planning each increment more carefully.

Testing each increment was also a weak area of development. While the incremental methodology was closely followed for development, testing was often an afterthought. This was caused largely from how unpredictable the behaviour of juggling multiple APIs could be. Combine this with the unfamiliarity of working with AngularJS, testing became a difficult task. Because the focus of this dissertation is the core system in which other sites could be built upon, it can be considered non-critical in terms of being error free. Despite this, the final result was almost completely free of bugs and has not had any effect on the project. This is further supported by the positive results from the user evaluation results.

Overall, the implementation was a great success. The user evaluation supports the case that the site is well implemented. Given more time, each area of the incremental methodology could have been given a more even share of time.

Evaluation Strategy

The choice to go with a more direct user based approach showed to be an effective method for evaluating the site. By having a small number of users give clear feedback for the application, a simple general idea of how the site would perform publicly could be obtained. There were a few ways in which this evaluation method could have been improved. The most obvious means for improvement would be a greater number of participants. This would greatly increase the time needed to perform the evaluation, but would give much more accurate trend information.

For all 5 participants the same large screen monitor and desktop was used. With more participants it would have been useful to test the site on a range of different devices, at the users preference. This was not done as the project was only deployed locally on one machine. Deploying the site to multiple devices for each participant would have been a very time consuming process and would not outweigh the advantages. One issue which arose from the hardware given to participants was the glowing red keys of the keyboard. One participant was red colour blind and struggled to perceive the keys. Being prepared with additional choices of hardware would have been useful for such a scenario.

Another improvement would have been to have a wider range of computer literacy between users. Only participants with fairly average tech experience were used. Gathering the opinions of participants who are experts in this field would have greatly improved feedback.

6.9 Personal reflection

Over the entire course of development, the project has been both a rewarding and challenging experience. With the limited prior knowledge of web development, the chance to explore recent trends in development, particularly building Single-Page Applications, has been an excellent learning experience. Although staying motivated and disciplined could be a challenging task at times, having a keen interest in the projects subject matter made the project highly satisfying. At the end of this project a plethora of new relevant skills relating to web technology were obtained that will be helpful in the future. Overall, the projects aims were achieved and a first step into the idea of a single page hosting a large number of music sites under one roof was made.

Chapter 7

Appendix

Appendix 1 - HTML Fragments Container

```html
<div class="container-fluid" style="height:100%;">
        <div class="row" style="height:100%">
            <div class="col-xs-2 col-sm-3 search-container" style="background-color:lavender; height:100%; padding-top:10px;">
                <!-- <div ng-include="'app/htmlpartials/search.html'"></div>-->
            </div>

            <div class="col-xs-4 col-sm-4" style="background-color:lavenderblush; height:100%; padding-top:10px;">
                <!-- <div ng-include="'app/htmlpartials/playlist.html'"></div> -->
            </div>

            <div class="col-xs-6 col-sm-5" style="background-color:lavender; height:100%; padding-top:10px;">
                <!-- <div ng-include="'app/htmlpartials/video.html'"></div> -->
            </div>
        </div>
</div>
```

Appendix 2 - User Evaluation Tasks

1. Familiarise self with the overall layout and structure of the site.
2. Make a simple search for a song of the user's choice from either SoundCloud or YouTube.
3. Play the song of the user's choice from the search section.
4. Make a playlist of 5 or more songs, all of the user's choice. Choices should preferably be a mix of sites.
5. Save the playlist and load it afterwards.
6. Navigate between songs using the previous, next and shuffle buttons.
7. Pause, fast-forward and rewind a song from an embedded player of the user's choice.

Appendix 3 - Questionnaire

Please complete this questionnaire after you have completed the evaluation of the website. Please circle one answer for each of the questions.

1. The website has clear organisation of information with clear navigation

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

2. The website has a good design and colour scheme

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

3. The search feature is easy to use and intuitive

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

4. Searching gives good results

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

5. The playlist feature is easy to use and intuitive

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

6. The playlist feature has an adequate amount of features to be useful

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

7. Saving and loading playlists works as intended

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

8. video controls are easy to use and intuitive

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

9. Media plays cleanly as expected

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

10. I would use the site in future if it was publicly released

1. – Strongly disagree
2. - Disagree
3. - Neither agree nor disagree
4. - Agree
5. - Strongly agree

Bibliography

[1] Account, BobDylanVevo YouTube. *Bob Dylan - Blood In My Eyes.* 2009. URL: https://www.youtube.com/watch?v=nz542iQchN4 (visited on 05/04/2016).

[2] Anderson, Sean. *Streaming YouTube music made simple by Google Chrome - Streamus.* 2014. URL: https://streamus.com/.

[3] Bootstrap, Start. *Freelancer: A one page Bootstrap portfolio theme.* URL: http://startbootstrap.com/template-overviews/freelancer/ (visited on 05/04/2016).

[4] Bootswatch. *Bootswatch: Flatly theme.* URL: https://bootswatch.com/flatly/ (visited on 05/04/2016).

[5] Camp, Jeffrey Van. *Spotify's play button: a disappointing way to embed music into a web page.* Apr. 2012. URL: http://www.digitaltrends.com/music/spotifys-play-button-a-disappointing-way-to-embed-music-into-a-web-page/.

[6] Casimirri, Maurizio. *Mobile Angular UI.* URL: https://github.com/mcasimir/mobile-angular-ui (visited on 05/06/2016).

[7] Certification, ISTQB Exam. *what is the incremental model.* 2016. URL: http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/ (visited on 24/04/2016).

[8] Chaffey, Dave. *Mobile Marketing Statistics compilation.* 2014. URL: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ (visited on 05/06/2016).

[9] Fisher, Ronald and Yates, Frank. *Statistical tables for biological, agricultural and medical research.* Oliver & Boyd, 1948. ISBN: 0028447204.

[10] Google. *AngularJS Cookies Module.* 2015. URL: https://github.com/angular/bower-angular-cookies (visited on 05/05/2016).

[11] Google. *AngularJS - HTML enhanced for web apps!* 2016. URL: https://angularjs.org/.

[12] Google. *Google Developers YouTube API.* 2016. URL: https://developers.google.com/youtube/.

[13] Hisaichi, Joe. *Summer.* 1998.

[14] *IFPI Digital Music Report 2015.* 2015. URL: http://www.ifpi.org/downloads/Digital-Music-Report-2015.pdf.

[15] Jacko, Julie A. *Human Computer Interaction Handbook.* CRC Press, 2012. ISBN: 9781439829431.

[16] Jorgensen, Paul. *Software testing: a craftsman's approach.* Nov. 2013. ISBN: 1466560681.

[17] Likert, Rensis. *A Technique for the Measurement of Attitudes.* 1932.

[18] Michael Mikowski, Josh Powell. *Single Page Web Applications: JavaScript end-to-end.* Manning Publications Co. Greenwich, CT, USA, 2013. ISBN: 1617290750 9781617290756.

[19] Peat, Oliver. *TuneCrawl - Find and play music from the most popular music streaming providers.* 2015. URL: http://www.tunecrawl.com/.

[20] Preece, Jenny, Sharp, Helen, and Rogers, Yvonne. *Interaction Design.* John Wiley & Sons, 2015. ISBN: 9781119020752.

[21] Statistica. *Worldwide mobile app revenues.* Mar. 2016. URL: http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/ (visited on 09/03/2016).

[22] Team, Bootstrap Core. *Bootstrap browser support.* 2016. URL: http://getbootstrap.com/getting-started/#support (visited on 05/04/2016).

[23] Team, Bootstrap Core. *Bootstrap Grid Basics.* 2016. URL: http://getbootstrap.com/.

[24] Team, Bootstrap Core. *Bootstrap - The world's most popular mobile-first and responsive front end framework.* 2016. URL: http://getbootstrap.com/.

[25] Team, jQuery. *jQuery: The Write Less, Do More, JavaScript Library.* URL: https://jquery.com/.

[26] Team, SoundCloud. *SoundCloud Developers API.* 2016. URL: https://developers.soundcloud.com/docs/api/guide.

[27] Temples, The. *Sun Structures.* Heavenly Records, 2014.

[28] W3Schools. *W3.CSS - Faster and Better Responsive Web Sites.* 2016. URL: http://www.w3schools.com/w3css/.

[29] W3Techs. *Usage of JavaScript libraries for websites statistics.* 2016. URL: http://w3techs.com/technologies/overview/javascript_library/all (visited on 05/03/2016).

[30] Watson, James. "Combining Music Streaming Services - Initial Document". Oct. 2015.

[31] Watson, James. "Combining Music Streaming Services - Interim Document". Feb. 2016.

[32] Williams, Owen. *Chrome Extension Makes YouTube a Serious Music Player.* Jan. 2015. URL: http://thenextweb.com/apps/2015/01/20/chrome-extension-turns-youtube-serious-music-service/ (visited on 03/05/2016).

[33] Williams, Owen. *How YouTube killed an extension with 300,000 users.* July 2015. URL: http://thenextweb.com/insider/2015/07/21/how-youtube-killed-an-extension-with-300000-users/.

[34] Yasin, Sycra. *How to Choose Colours that Work! - YouTube Video.* Aug. 2013. URL: https://www.youtube.com/watch?v=9kQllLy_X4I (visited on 05/05/2016).

[35] Zurb. *Foundation - The most advanced responsive front-end framework in the world.* 2016. URL: http://foundation.zurb.com/ (visited on 05/03/2016).