TheMusicExchangeProject

A web application for connecting musicians.

Luke Hengstenberg

878876

May 2019

Music is a fundamental part of our society and culture. Playing an instrument is one of the richest human emotional experiences providing opportunities for sharing and connecting with others and enhancing well-being. Creating the right social setting is integral to generating an enjoyable and satisfying musical experience. Many of the existing products fail to facilitate a setting that includes musicians that are beginners, lacking confidence, or want to play on a casual basis. TheMusicExchangeProject implements a modern, safe, and easy to use web application, establishing a new way for musicians of all abilities to connect. Providing a range of customisable search preferences, a relationship system, and a real-time messaging service, this project acts as a catalyst for pairing likeminded individuals. Powered by a large array of tools and frameworks including ASP.NET Core MVC, this project has produced a platform with an aesthetically pleasing client side and complex server side functionality. With a sophisticated database, the project has the ability to handle expansion as popularity increases and has been designed to be built upon in the future.

Final Dissertation Document submitted to Swansea University
in Partial Fulfilment for the Degree of Bachelor of Science.

Department of Computer Science Swansea University

Declaration

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

10th May, 2019

Luke Hengstenberg Signed: [illegible]

Statement 1

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

10th May, 2019

Luke Hengstenberg Signed: [illegible]

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 page(s) using the bibliography / references section. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

10th May, 2019

Luke Hengstenberg Signed: [illegible]

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 organizations.

10th May, 2019

Luke Hengstenberg Signed: [illegible]

Table of Contents

Table of Figures

1: Introduction

The experience of listening to and playing music is probably the richest human emotional, sensorimotor, and cognitive experience [1]. It involves listening, watching, feeling, moving, remembering, and expecting. Involvement in music provides opportunities for sharing and connecting with others, enhancing well-being and providing therapeutic benefits. Music is a fundamental part of our society and culture, naturally leading to a large percentage of the population playing or learning to play a musical instrument.

The Making Music research project [2], interviewed 1,255 adults about their musical ability. Amongst the questions asked was whether the adults currently play a musical instrument. *Figure 1* below shows the statistics split by age group.

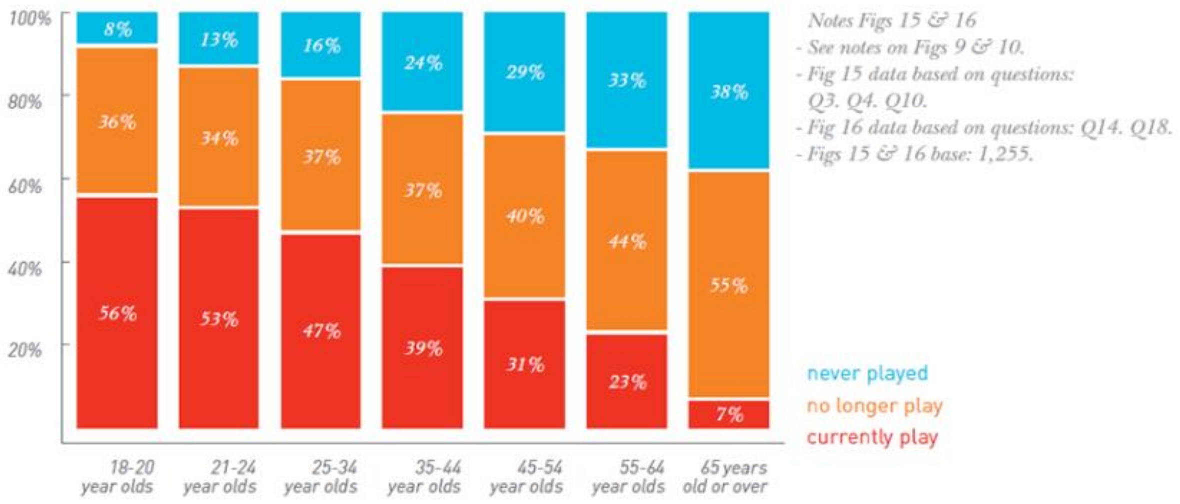


Figure 1: Percentage of adults that play musical instruments, The Making Music Report [2].

F*igure 1* illustrates the percentage of adults that currently play musical instruments. The percentage gradually rises as the age groups get younger, with over half of 18-24 year olds currently playing an instrument. This supports the view that there is a definite and probable growth in interest for playing and learning to play music within our society.

Many resources exist for improving musical ability, the most popular being paid lessons with a tutor. However, these resources can prove costly, inaccessible, or inappropriate for a lot of the consumers using them. When 326 adults were asked their reasons for never taking musical instrument lessons 31% of adults said they were/are too expensive, 29% stated they were not interested in lessons, and 27% said they only wanted to play music for fun [2]. With hourly lessons for an instrument such as the guitar ranging in cost from £25 to £30 (National Average on Bidvine) [3], it is no surprise that money is a big barrier for learning.

When conducting background research, tools designed for meeting people with the purpose of playing music together were found, however the majority of these are not appropriate for beginner or casual use. Popular websites such as joinmyband or bandmix take the form of classified ads, focused on recruiting band members with a high level of experience. For an individual with low confidence in their musical abilities, little experience, or someone simply looking for casual interactions, it is clear that an alternative resource should be made available.

This project outlines the development of a web application which aims to cater for anyone that plays or is willing to learn a musical instrument. TheMusicExchangeProject is a platform that openly welcomes all musicians from those just starting out to those that already play at an advanced level. Although the interactions between two musicians cannot be directly influenced this project aims to act as a catalyst to the process of finding the right person to learn with, learn from, or play with on a casual basis. The project will provide a free to use, modern application that can be tailored to the individual's every need. By providing a high level of interactivity and a rich set of features, the project aims to make the experience of meeting musicians online effortless and enjoyable.

TheMusicExchangeProject offers a range of search preferences which can be used to find a person that plays a specific instrument, at a certain skill level and at a convenient distance radius. With descriptive profile cards and a matching system that requires all connections to be consensual, TheMusicExchangeProject will provide a safe and personalised user experience. The project implements a real-time messaging system that allows communication to be simple, direct, and practical. A sophisticated database safely stores all data and a unique frontend interface allows the user to view and perform CRUD (Create, Read, Update, Delete) functions.

The reader of this dissertation should gain an in-depth understanding of the web application development process, the techniques used by the developer to design and integrate the features, and the technologies involved in its implementation. The dissertation will detail the challenges the developer faced throughout the process and the decisions made to ensure a high quality, fully functioning product was successfully produced.

The overall objective for this project was to provide a sophisticated web application that can be used as a tool by musicians of all abilities to find and connect with likeminded people. To reach this objective, the following aims were established:

- Provide a platform that encourages and facilitates social interaction between musicians of all abilities.
- Offer a range of customisable search preferences.
- Implement the functions necessary to ensure full user control responsible for account details, skills and skill levels.
- Create a relational database for storing all user, skill, relationship, and message related data.
- Ensure all connections are consensual before communication can occur.
- Create an interface that is elegant, practical and has high learnability.

To describe the work conducted for this project the dissertation has been split into the following sections:

- Section 2: Provides an overview of the background research conducted at the start of the process. This includes a comparison between the similar applications that currently exist and the produced application.

- Section 3: Outlines the specification of the required and optional features for this project. Following this, a detailed analysis of the technology choices used for the implementation is conducted.
- Section 4: Documents the timeline over which project work took place and the software development methods used. Moreover, it explains the planning and design of the frontend and backend of the application.
- Section 5: Describes the full implementation of the product, covering which specified features were added and how each area of the application functions. It then goes on to explain the testing that took place and reflects on some of the technical issues that were encountered.
- Section 6: Evaluates what was achieved in this project, the challenges experienced, how they were mitigated, and if development will continue in the future.

2: Background

This section discusses the research that was conducted in preparation for this project. Background research was important in gaining an understanding of some of the core concepts of web development. Following this, a comparison between the characteristics of similar applications and this project was made.

2.1 Relevant Research

Before work on this project could properly begin, it was important to build a strong initial understanding of the approaches and tools available for web development. The research documented in this subsection was essential for making the technology choices for this project.

The first topic of research looked at how software architectural patterns and styles are used to structure software systems. The paper "Architectural Patterns Revisited – A Pattern Language" [4], explains that architectural patterns offer well-established, reusable solutions to architectural problems and help to document the architectural design decisions. Herberto Graca [5] explains that architectural styles identify how to organise the code by specifying layers, high-level modules of the application, and how the modules and layers interact with each other. Patterns and styles are complimentary, with architectural patterns solving problems related to architectural styles. Certain architectural patterns and styles are used by web development frameworks, so should be fully understood before the framework can be used. The architectural pattern chosen for research is Model-View-Controller. This was chosen because of its use in the frameworks ASP .NET and Laravel. The architectural style chosen for research is REST. This was chosen to understand how it works and evaluate its suitability for this project.

The Model-View-Controller (MVC) pattern is used for organizing a projects code into sections, with each section fulfilling a specific purpose. Code Academy [6] introduces the concept of a Model-View-Controller in their article "MVC: Model, View, Controller". The Model code is used for holding raw data or defining the essential components of the application. In this application models are used to define many components such as the users and skills. The View code defines all the functions that directly interact with the user and is used to decide how the application looks. In this application each page is represented by a different View. The

Controller code liaises between the model and the view, receiving user input and deciding what to do with it. Different controllers are used to create the logic behind each part including Searching Users and Managing Skills. The Models, Views and Controllers used in this project will be demonstrated in section 5.

REST or Representational State Transfer is an architectural style for providing standards between computer systems on the web. The article "What is REST?" [7], explains that REST-compliant or RESTful systems separate the concerns of the client and server, meaning the implementation of the client and the server can be done independently. In the REST, architecture clients send requests to retrieve or modify resources and servers send responses to these requests. These requests consist of a HTTP verb, a header, a path to a resource and an optional message body containing data. A RESTful system uses four HTTP methods (verbs), GET, PUT, POST, and DELETE [8]. Dr. Roy Fieldings' introduced the terms that define the REST principles in his thesis, "Architectural Styles and the Design of Network-Based Software Infrastructure" [9]. To summarise, the guiding constraints of REST are:

- Separation of the Client and the Server, as explained above.
- Statelessness: Client-Server communication must be stateless such that each request from the client to the server contains all the information necessary to understand the request.
- Cache: Server responses must be labelled as either cacheable or not.
- Uniform Interface: Single interface used within the API.
- Layered System: Multiple, hierarchical layers used to grow and expand interface. Layers can be added without impacting original functions.
- Code-On-Demand: Client can download features after deployment.

When development on this project initially began, it was planned to build the application as completely RESTful API using the REST architecture. However, technology choices were altered and the final application only partially uses the REST architecture in structuring parts of the chat system. This will be explained further in section 5.7.

Researching REST naturally led to learning about the CRUD functions and how they could be used in this project. CRUD or Create, Read, Update, Delete, are the four basic functions of persistent storage [10]. As explained by Stackify [10], each letter in the acronym can also refer to all functions executed in relational database applications and are mapped to a standard HTTP method, SQL statement or DDS operation. The term "CRUD" was likely to have been popularized by James Martin in his book "Managing the Data base Environment" published in 1983 [11]. Similarities can be identified between the protocols of a RESTful architecture and a CRUD cycle. CRUD can be mapped to REST by design [12]. Create maps to POST, Read maps to GET, Update maps to PUT, and Delete maps to DELETE. However, making a comparison between CRUD and REST is inconsequential. As covered above, REST governs much more than permanence within its principles of architecture and represents an entire architectural system rather than a cycle [12]. The CRUD functions have had an integral role in this project, in particular for managing skills and skill levels. CRUD's usage will be demonstrated in section 5.5.

The final topic researched was AJAX technology and how it could be utilised to make the application fast and more interactive. AJAX meaning Asynchronous JavaScript and XML, is not a technology or programming language in itself, rather a combination of several technologies [13]. AJAX provides the ability of communicating with the server asynchronously. This means when interactions occur on the frontend interface, JavaScript and DHTML can be used to refresh the interface, make a request to the server, and perform operations on the database simultaneously [13]. This is useful as the user will not be able to tell that the browser is communicating with the server as the response is immediate and the page will only be partially updated rather than refreshed. Understanding AJAX was important before starting development as areas including the chat system needed to be responsive and required frequent communication with the server.

2.2 Similar Applications

In preparation for the specification and design of this project it was necessary to identify the similar applications that currently exist. Initially this was done to gain an understanding of the current options available for musicians to meet and how the project will stand out once it has been developed. Once the product had been created, an examination of the similarities and differences between them is undertaken from a practical standpoint instead of a theoretical one.

The first application researched was a website called Flint which can be accessed using the URL: https://www.flint.cool. Flint is free to use and is available as a website as well as on the app store. *Figure 2* shows a screen shot of the 'Discover' page [14] used for searching and matching with users on the site.

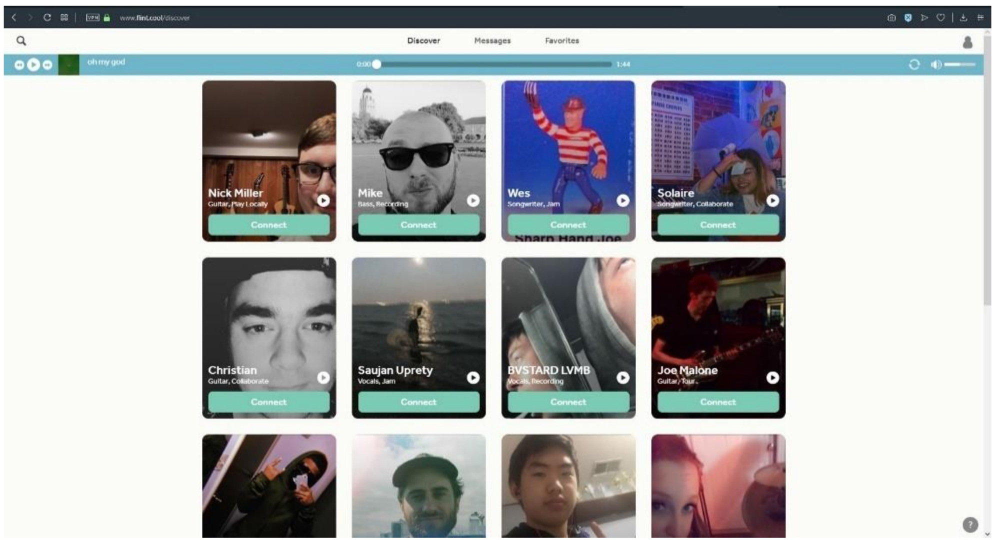


Figure 2: Screen shot taken of the website Flint showing the 'Discover' page.

Flints design shares many similarities and differences with the design of this project. It's dating app like structure requires two accounts to mutually choose to connect with each other. This same design feature has been implemented in this project to ensure all connections have been consensual. However, a request feature has also been added to this project to allow for connections to be formed at a faster rate. Flint offers a number of search tools and preferences, allowing the user to filter by a search term, distance, or a number of predefined instruments and interests. Although these are similar to the search preferences offered by this project, there are some limitations when compared. The distance preference offered by Flint is vague and does not seem accurate, with the locations used being cities or states inputted by the user. In this project distance has been calculated in miles using exact latitude and longitude coordinates.

Flint's use of 9 predefined instruments limits the customisability of the search options for the user and does not allow all users to correctly represent the instrument they play. In this project, any instrument as well as a skill level can be linked to a user's account and therefore be used as a search preference. Finally, the majority of Flint's users are based in America. After personally testing the search feature only 2 Welsh users and 20 English users were found. This supports TheMusicExchangeProject in its intended purpose, to be used as a tool for musicians who are UK based.

The next application researched is a website called Hendrix which can be accessed using the URL: https://www.gohendrix.com. Hendrix is free to use and is available on all current web browsers. *Figure 3* shows a screenshot of the 'Profiles' page [15] used for finding musicians on the site.

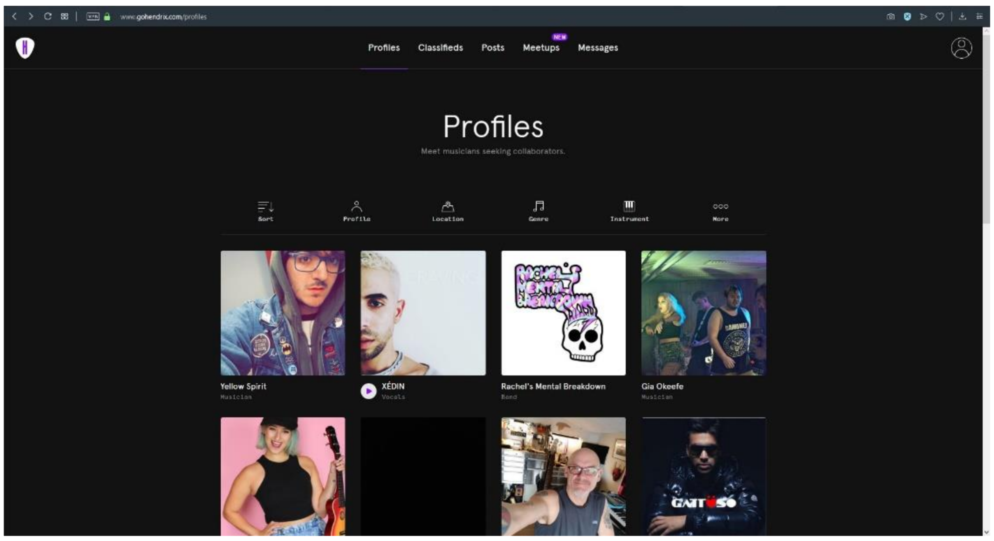


Figure 3: Screen shot taken of the website Hendrix showing the 'Profiles' page.

Hendrix shares certain similarities to this project; however, the purpose of the application is different in essence. As well as connecting musicians to other musicians, Hendrix connects musicians to businesses and venues. It is designed to be professional, dedicated to promoting high quality musicians. This differs to this project as the motivation behind it has been to create an application for musicians of all abilities. Hendrix provides a number of the same search preferences including searching by location and instrument, however it does not use the same connection based design as implemented in this project, allowing messages to be sent to any registered user. Hendrix is also solely designed for American musicians which further supports the projects intended use as a tool for UK based users.

The next application researched is a mobile app called Jamseek which can be found through the URL: https://www.jamseekapp.co.uk/en/index. Jamseek is currently only available in app form and can be downloaded for free on the Googleplay and Apple App store. *Figure 4* shows a screenshot of the website [16] used to provide information about the app and direct traffic to the app store.

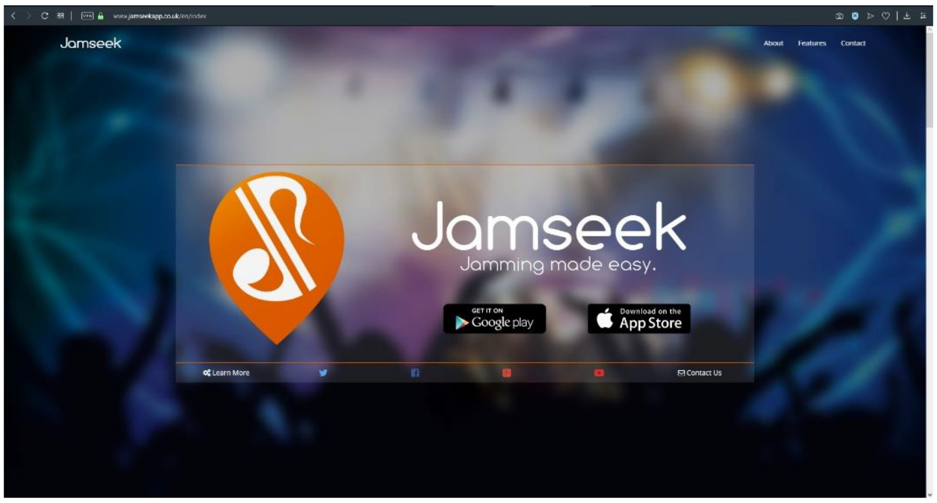


Figure 4: Screen shot taken of the website Jamseek directing traffic to the app store.

The Jamseek app is designed for organising meet ups between musicians to play music in a real life social setting. The motivation behind the app is explained in the 'About' section [17] of the website and immediately highlights similarities between the app and this project. Jamseek is aimed at musicians of all abilities, with searching preferences based on instruments, favourite genres, and skills. It seems to be the most similar in essence due to its audience and motivations.

However, Jamseek's design differs as it offers no connection based system, allowing users to message anyone on the app. This has the potential to cause problems for the user as there is no initial choice on receiving messages. This security issue has been address in this project by implementing a system where users mutually agree to a connection before any communication can begin. Since the researchers last visit to the Jamseek website in October 2018, there seems to have been no changes made to the website or app. The last sign of any internet presence was a post to their Twitter [18] on the 20th November 2018. This leads the researcher to believe that Jamseek would not be a suitable tool for a current musician in the UK to find other musicians, and gives this projects usefulness further credibility.

3: Project Specification

This section of the document outlines the required and optional features for the application. These have been selected to meet the project aims set out in section 1. Following this, the section breaks down the technology choices made to develop these features. Discussing the languages, frameworks and development tools used to create the web application.

3.1 Feature Specification

During the creative process of this project, correct planning and specification of the features was of upmost importance. To deliver a technical and user-friendly final product the features must be carefully selected before technology choices can be made and development can begin. Once the features have been specified, the appropriate technology was chosen to implement them.

The required features were chosen to provide a basic working product for the user. They were created by imagining how the user would use the product, what the user wants to get out of the product and by looking at how similar applications have structured their products. The additional features were chosen to provide a better experience for the user and to increase the quality of the product.

The required features are as follows:

- Basic management of user accounts:
  1. Interface to register a new account with appropriate input validation.
  2. Interface to login with appropriate account authentication.
  3. Interface to view account, choose information to display and modify information.
  4. Account privacy setting customisation.
- Database for storing information:
  5. Storage of data associated with user account.
  6. Storage of data associated with skills.
  7. Storage of data associated with relationships.
- Appropriate searching system:
  8. Instrument(s) played can be linked to the users account.
  9. Page to display the current accounts in the system.
  10. Search tool for filtering the displayed accounts using user input as a keyword linking to information related to the accounts.
  11. Interactive tool for filtering the displayed accounts based on instrument(s) played.
- System for establishing relationships between users:
  12. Button on accounts that stops a target user from being visible or able to contact the current user.
  13. Button on accounts that prompts the system to automatically send a match request to the specified account from the current user.
  14. Match requests made visible to the user with the options accept and decline.
  15. Automatic linking of two accounts when match request is accepted or both users have mutually chosen to connect.
  16. Communication via email made possible between the matched accounts once linked.

After the implementation of the above features, the main focus turned to increasing the pleasure of the experience and the convenience of using the product. Additional features for the user to interact with on the frontend of the application helped to achieve this.

The additional features are as follows:

1. Expansion of interactive tool within the searching system to allow filtering by more preferences, such as location/proximity of other users and skill levels.
2. Further Account customisability – Profile pictures, bio, age.
3. Scrollable list of accounts the user is connected to.
4. Scrollable list of accounts the user has blocked.
5. Options to modify existing relationships.
6. Embedded instant messaging tool which can be used for communication between matched users.
7. Live notifications listing message and match requests.
8. Match suggestions that are automatically generated by the system using account information such as the instrument a user knows and the instrument a user wants to learn.
9. Embedded Video chat tool.
10. Expansion of the interests linked to an account, such as genres of music, purpose of account.

A review of the implemented features is documented in section 5.1.

3.2 Technology Choices

After laying out the features to be implemented the next task was choosing the set of tools. Making the correct technology choices required careful planning, research and evaluation. Some of the technology choices have changed significantly since the beginning of the project, others have remained the same throughout. This subsection discusses the tools used in the completion of the project and the reasons behind these changes.

3.2.1 Software Languages

Each language has a specific purpose and is used to power the features and functionality of the web application. There have been a variety of factors that have impacted the developer's decisions, such as:

- The frameworks available.
- The documentation and support behind the language.
- Personal knowledge and experience with the language.
- The suitability of the language in regards to its purpose within the project.

Each choice for this project, how they have changed and their usage is discussed below.

PHP: Hypertext Preprocessor [19] is a widely used open source, general-purpose scripting language. It was originally developed by Rasmus Lerdorf to assist users with web page tasks, before rapidly growing to become the full-featured language it is now. As detailed by Valade, J[19] PHP has many advantages which have influenced its popularity. PHP is fast (Embedded in HTML code), versatile (Variety of OS), secure (Hidden from user), and customizable (Open source). In the early stages of this project PHP was chosen as the language to power the backend (server-side) of the application. This decision was based on its advantages and the desire to use the PHP framework Laravel for development. As project development continued, progress began to fall behind schedule due to my lack of experience with PHP and Laravel. After evaluating the initial technology choices and consulting the project timetable the developer felt the project was at risk of incompletion. To mitigate this risk and get the project back on schedule the decision was made to no longer use PHP and switch to a language and framework the developer had more experience using. This led to the use of the language C# for the completion of this project.

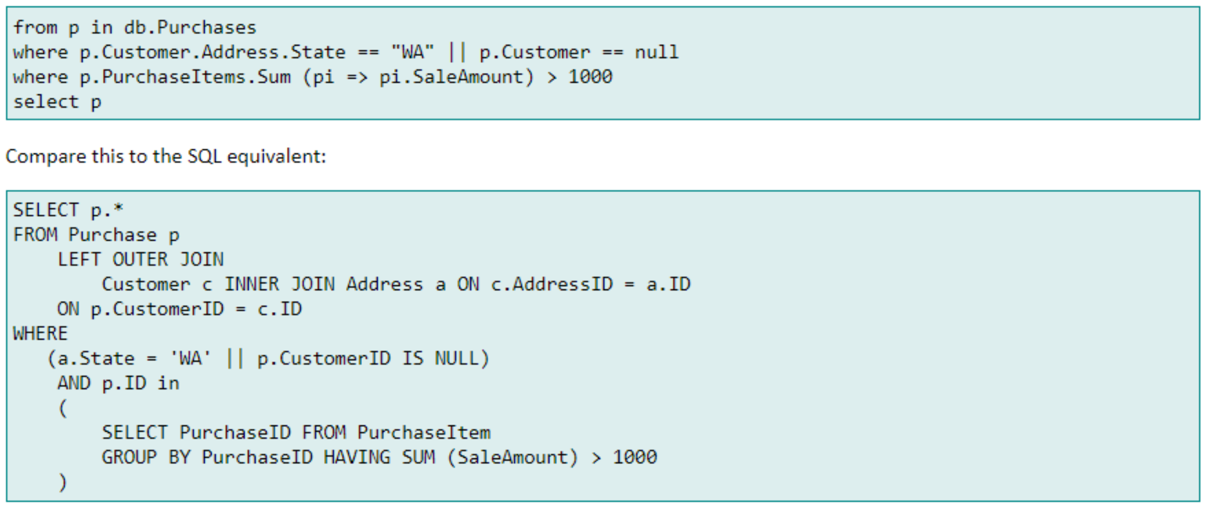
C# (C-Sharp) [20] is a general-purpose, multi-paradigm, object-orientated language. As explained by Joel Martinez [20], it was developed by Microsoft in 2000 and aimed to take the best aspects of the programming languages that came before, namely the strength and power of C++, the simplicity of JavaScript, and the ease of hosting of VBScript/ASP. C# is a powerful programming language which contains some attractive features. These include:

- Common Language Runtime (CLR): This allows interoperability meaning the project can be written in a number of different languages and still run on the same runtime. This helps to maximize code reuse and improves the efficiency of development.
- Memory management: CLR provides a garbage collector, which means memory will be collected automatically when an object is no longer referenced by other objects.
- Base Class Library (BCL): C# comes with a rich and vast library which provides many inbuilt functions to speed up development.
- Language Integrated Query (LINQ): A set of diverse features which provide first class support for querying data from a multitude of sources. Provides strong abstraction around the concept of querying, allowing fast reference to the projects database.

In addition to offering a multitude of useful features, C# is also syntactically familiar to the developer and has a large amount of support and documentation. These factors, as well as previous experiences of working in C# influenced the decision to change from using PHP.

SQL (Structured Query Language) [21] is a domain-specific programming language for accessing and manipulating the data held in a relational database. SQL can be used to query a database to retrieve, insert, update or delete the data it contains. Although, SQL forms the base of the code behind the projects database it has not been necessary to make SQL queries directly due to C#'s integrated feature LINQ.

As mentioned above, LINQ or Language Integrated Query is a set of technologies based on the integration of query capabilities directly into the C# language [22]. LINQ turns a query into a first-class language construct such as classes, methods and events. Query expressions are written in a declarative query syntax, thus performing filtering, ordering, and grouping operations on the data from the database with a minimum amount of code. The article "Why LINQ beats SQL" [23] demonstrates the efficiency and tidiness of a LINQ query in comparison to an SQL query, an extract can be seen in *figure 5* below.



```
from p in db.Purchases
where p.Customer.Address.State == "WA" || p.Customer == null
where p.PurchaseItems.Sum (pi => pi.SaleAmount) > 1000
select p
```

Compare this to the SQL equivalent:

```
SELECT p.*
FROM Purchase p
    LEFT OUTER JOIN
        Customer c INNER JOIN Address a ON c.AddressID = a.ID
    ON p.CustomerID = c.ID
WHERE
   (a.State = 'WA' || p.CustomerID IS NULL)
    AND p.ID in
    (
        SELECT PurchaseID FROM PurchaseItem
        GROUP BY PurchaseID HAVING SUM (SaleAmount) > 1000
    )
```

Figure 5: Comparison of LINQ Query to SQL Query - "Why LINQ beats SQL" [23]

LINQ's functionality is explained further in Microsoft's article "LINQ to SQL" [24]. To summarise, the data model of a relational database is mapped to an object model which is expressed in C#. When the application runs, the language-integrated queries in the object model are translated into SQL and are sent to the database for execution. Once the results are returned, they are translated back into objects that can be used in the application. LINQ has been utilised throughout this project to perform various operations on the database and plays an extremely important role. LINQ's use will be demonstrated in section 5.

HTML (Hypertext Markup Language) [25] is considered the standard for building web pages and is the most commonly used markup language. Hypertext is simply the text displayed on a computer device and contains links to navigate to other hypertext documents. A hypertext document may describe objects such as lists, tables, images and forms. A markup language is used to annotate documents in a way that is distinguishable from the text, creating a structure. The latest version HTML5 is used throughout the application to define the elements and structure of each page. As explained by Budi Kurniawan [25], HTML5 is well understood by virtually all browsers in use today, therefore making it the obvious option for this project.

CSS (Cascading Style Sheets) [26] is a style sheet language that describes how a document written in HTML is displayed on screen to the user. CSS is designed to enable a separation of presentation and content to give a HTML page visual dynamics such as fonts, colours and layouts. This allows a page to be presented in many different ways to improve content accessibility, make adjustments based on screen size and improve the overall aesthetics of the website. The latest version CSS3 [27] is used throughout this project to style each page of the application.

JavaScript (JS) [28] is a lightweight, interpreted, object-orientated language with first-class functions, best known as the scripting language for web pages. As explained by MDN web docs [28], is multi-paradigm and supports object-oriented, imperative, and functional programming styles. It runs on the client side of the web and can be used to program how web pages behave when an event occurs. JavaScript has been used to make the application more dynamic and responsive. It powers many of the features on the frontend (client-side) and was chosen due to its vast documentation and its compatibility with the other software languages.

3.2.2 Language Frameworks and APIs

A Software framework is a platform for developing software applications [29] that is often a library or collection of programs written in a specific programming language to accomplish a specific task. A programming or software language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform the specific tasks [30]. A framework provides a foundation for software development and helps to streamline the development process by providing components such as, libraries of code, support programs, compilers, tool sets, and specific APIs that facilitate the flow of data [31].

Using frameworks for the development of this project instead of the raw software languages has massively increased the speed and ease of development. Frameworks were chosen based on the same deciding factors behind the selected languages. This subsection discusses all the frameworks used in this project, the reasons they were chosen and how choices have changed from the beginning to the end of the development process.

Laravel is an open-source PHP framework that follows the Model-View-Controller design pattern [32]. It was developed by Taylor Otwell in 2011 and incorporates the basic features of other PHP frameworks such as CodeIgniter and Yii. Saunier, R [33] provides a useful overview of Laravel's main features in his book 'Getting Started with Laravel 4'. Laravel offers a rich set of features for web development, to summarise several from the book:

- Modularity: Laravel was built on top of over 20 different libraries which are tightly integrated with Composer Dependency Manager and can be updated with ease.
- Testability: Laravel ships with several helpers that add the ability to visit routes from tests, crawl the resulting HTML, ensure that methods are called on certain classes, and even impersonate authenticated users.
- Routing: Laravel offers a lot of flexibility when defining the routes of an application. For example, it is possible to manually bind a simple anonymous function to a route with the HTTP verbs like GET, POST, PUT, or DELETE.
- Query builder and ORM: Laravel ships with a fluent query builder which allows the developer to issue database queries with a PHP syntax using chain methods. It provides an Object relational mapper (ORM) and ActiveRecord implementation called Eloquent.

After researching Laravel in the early stages of this project and discovering its powerful features, it was initially chosen for developing the back-end of this application. This decision required a steep learning curve due to lack of experience with the syntax of PHP and the first time use of Laravel. Due to these personal limitations, progress became slow during the implementation of this project and started to fall behind schedule. The risk of incompletion started to become apparent and it was clear that the technology choices needed to be evaluated. The decision was made to change to a framework that was as rich in features as Laravel, but more familiar to the developer and therefore easier to use. This led to substituting Laravel for the framework ASP.NET Core MVC.

ASP.NET Core MVC is a lightweight, cross-platform, open source, highly testable presentation framework optimized for use with ASP.NET Core [34]. The web application development framework created by Microsoft combines the effectiveness and tidiness of MVC architecture, ideas and techniques from agile development, and the best parts of the .NET platform [34]. Adam Freeman provides a useful overview of the frameworks benefits in his book 'Pro ASP.NET Core MVC' [35]. To summarise, several benefits of using ASP.NET Core for web development are:

- Extensibility: The framework is built as a series of independent components that have well defined characteristics, satisfy a .NET interface or are built on an abstract base class. These key components can be easily replaced with your own implementation.
- Tight Control over HTML and HTTP: The framework produces clean, standards-compliant markup. The framework works in tune with HTTP, so the developer has control over the requests passing between the browser and server, allowing user experience to be fine-tuned.
- Testability: The framework helps make the application maintainable and testable because you naturally separate different applications concerns into independent pieces. Each piece can be isolated and replaced for unit testing.
- Modern API: ASP.NET Core MVC is built for .NET Core, so its API can take full advantage of its language and runtime innovations, including the await keyword, extension methods, lambda expressions, anonymous and dynamic types, and Language Integrated Query (LINQ).

The decision to switch to using ASP.NET Core MVC and C# instead of Laravel and PHP was an important one that was crucial to the successful completion of this project. Research into the benefits and features offered by the ASP.NET Core MVC framework supported its suitability as a replacement to Laravel. The developer's previous experience using the ASP.NET Core MVC framework and familiarity with the C# language further supported and solidified this decision. ASP.NET Core MVC was therefore chosen as the framework to implement the backend (server-side) of this project.

Bootstrap is the world's most popular front-end component library for developing responsive, mobile-first projects with HTML, CSS, and JS [36]. Bootstrap is free, open source and easy to use with a basic understanding of HTML and CSS. It includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, and many others [37]. Bootstrap provides the functionality to create responsive designs. In this project, Bootstrap 4 (the latest stable release), has been used on the frontend (or client-side) to enable the web pages to automatically adjust themselves to look good on all devices. This was important to allow users of the application access from a mobile or other device. This project also uses a number of the other features the framework provides such as modals, forms and navigation.

Vue.js is a progressive JavaScript framework for building user interfaces and single-page applications [38]. The core library is focused on the view layer only, and can be easily integrated with other libraries or existing projects. Vue.js provides many useful features for front-end development, such as using a virtual DOM, Data Binding, Event Handling, Animation/Transition, Routing, and many more [39]. Vue.js was initially chosen for developing the frontend, in combination with Laravel for the backend. The compatibility of Vue.js and Laravel, and their retrospective features formed the basis of this decision. However, as previously discussed, the decision was made to change from using Laravel to ASP.NET Core MVC, structuring the project as a Multi-Page Application instead of a Single-Page Application. The use of Vue.js no longer became necessary in this application and the combination of Vue.js and ASP.NET Core MVC is currently not well documented. The ASP.NET Core MVC project templates for web applications come with jQuery integrated and it was felt that the jQuery library provided the sufficient tools for scripting the frontend of the application.

jQuery is a fast, small, and feature-rich JavaScript library that works across a multitude of browsers[40]. It aims to make it easier to use JavaScript on the website, with a "write less, do more" mentality. jQuery takes a lot of common tasks that take many lines of JavaScript code to accomplish, and wraps them into methods that can be called with a single line of code[41]. jQuery contains many useful features such as HTML/DOM manipulation, CSS manipulation, HTML event methods, Animations and effects, AJAX, and Utilities. The decision to use jQuery in this project was largely based on the methods it provides for AJAX functionality. In this project jQuery and Ajax technology have been used in combination with the hosted service Pusher to build the real-time messaging system. jQuery has also been used to script certain aesthetics by manipulating the CSS and HTML content.

Pusher is a hosted API for adding real time bi-directional functionality to web and mobile apps using WebSockets [42]. Pusher sits as a real-time layer between the server and the client. It tries to maintain persistent connections to the clients over WebSockets, and failing that, uses HTTP-based connectivity. Using Pusher the server can push new data to the clients instantly. Pusher Channels provide real-time communication between servers, apps and devices [43]. It uses a publish/subscribe model where an application can subscribe to a specific channel and then publish data on that channel, which will be seen by all the subscribers of that channel. Pusher hosts public channels for holding data that can be accessed by anyone and private channels which require permission before a user can subscribe. Pusher has been used in this project because of its suitability for providing real time functionality to the application. Using Pusher Channels and jQuery with HTTP requests executed via AJAX, a real-time messaging system for the users was built. How Pusher, AJAX and jQuery were implemented is discussed in section 5.7.

Postcodes.io is a free, open source, postcode and geolocation API for the UK [44]. It offers a range of API endpoints and methods for incorporating bulk reverse geocoding, auto completion and validation. The API is regularly updated with the latest data from the Ordnance Survey and Office for National Statistics. The main reason for choosing to use Postcodes.io as opposed to the GoogleMaps API is the pricing and ease of use. The API is used to lookup the details of a user's postcode after it is entered on account creation or updated on account modification. The Latitude and Longitude values are then parsed from the returned JSON and stored within the database. These coordinates are then used to calculate distances between users. How the formula is used for calculating distance is demonstrated in section 5.6.

Google Fonts [45] is a library of 915 free, open source, designer web fonts with hosted APIs for importing the fonts via CSS. Using the code generated by Google Fonts, their servers automatically send the smallest possible file to every user based on the technologies that their browser supports [45]. This allows designer fonts to be used in the styling of the website without the risk of massively slowing down the application. By importing the fonts from the hosted API the styles will be displayed to users accessing the application on all major browsers. Google Fonts was chosen to give the styling of the text a unique, crafted look, instead of the basic, boring look provided by the default font-families.

3.2.3 Development Tools

The last section of the technology choices covers the software tools used for the development of the application. Selecting the correct tools for development required careful consideration of the compatibility with the chosen Software Languages and Frameworks. The change from Laravel to ASP.NET Core MVC has meant the set of tools initially used has been completely changed when compared to the final set of tools. For brevity, this section will only be covering the final development tools chosen for this project.

Visual Studio is an integrated development environment that can be used to edit, debug, and build code, and then publish an app [46]. Visual Studio provides more than the standard editor and debugger most IDEs provide. It includes compilers, code completion tools, graphical designers, and many more features to ease the software development process [46]. Visual Studio can be used for development in C#, F#, Visual Basic, C++, Python, JavaScript, and many more languages [47]. It is by far the most popular and well documented IDE for developing in the framework ASP.NET Core MVC, and was therefore chosen as the IDE for this project. It’s all encompassing nature and useful features have aided productivity in the development of this application. Some of the useful features [47] Visual Studio provides are:

- Squiggles and Quick Actions: Wavy underlines immediately point to errors or potential problems in the code whilst typing. Quick Actions suggest and provide ways to find a solution to these problems.
- IntelliSense: A set of features that display information about the code directly in the editor. Provides basic documentation inline in the editor which saves having to look up type information.
- Call Hierarchy: A window that shows the methods that call a selected method. This helps with changing or removing methods and tracking down bugs.
- CodeLens: A tool for finding references to the code, changes to the code, linked bugs, work items, code reviews, and unit tests, all inside the editor.

The Visual Studio IDE comes preinstalled with a sophisticated package manager NuGet, and Team Foundation Server for control over Git. Visual Studio also directly integrates MySQL with tools for SQL server management. These tools have been heavily used throughout this project so will be discussed in detail below.

NuGet is a free, open source package manager designed for .NET [48]. NuGet defines how packages for .NET are created, hosted, and consumed, and provides the tools for each of these roles [48]. NuGet packages contain reusable code that can be integrated and used within a project. NuGet assess the compatibility of a package to the projects framework, and if they are compatible, will manage the dependencies [49]. NuGet maintains a reference list to the packages a project depends on, removes packages from the list if they are uninstalled, and can restore all referenced packages upon request. NuGet Packages have been essential for powering significant areas of this project. The MVC component of the ASP.NET Core MVC framework has been integrated by NuGet. The NuGet package Identity [50] has also been used to add a membership system with login functionality to the application.

Team Foundation Server (TFS) is the application lifecycle management hub for Visual Studio [51]. It can be used to manage the source code held in Git repos hosted by any provider. TFS provides all the features necessary to support teams of developers working on the same project. As this has been a solo project changes have simply been committed to a local repository and then pushed straight to the master branch of the code. Bitbucket has been used to host the Git repository for this project and has been have accessed through TFS.

Bitbucket [52] is a Git repository management solution that provides a central place to manage git repositories and collaborations on the source code of a project. Unlike other Git solutions such as Github, Bitbucket provides access control to restrict access to the source code [52]. Bitbucket has been used to provide the version control for this project. Version control is a system that records changes to a file or set of files, such that specific versions can be recalled later [53]. This means that any major mistakes made during the development process have been easily resolved by rolling back the application to a previous version.

MySQL is the most popular open source SQL database management system [54]. MySQL is used to add, access, and process data stored in a database. MySQL databases are relational, meaning data is stored in separate tables with rules governing the relationships between different data fields. In this project, the SQL Server Object Explorer tool has been used throughout the coding process to access MySQL objects and the data contained within them.

The SQL Server Object Explorer is a tool in Visual Studio that provides a view of the objects inside the database [55]. SQL Server Object Explorer enables the developer to carry out light-duty database administration and design work. The tool allows editing of table data, comparison of schemas, query execution, and the performance of many more functions on the database. Since Visual Studio integrates MySQL directly into the Server Explorer, database management has been simple and has not required constant switching between programs.

4: Project Planning and Design

This section of the document breaks down the planning that went into this project and looks at the timeline over which development took place. Furthermore, the section covers the software development methodologies used and the planning and design of the database and the frontend of the application.

4.1 Progress Timeline

During the course of this project, the desired timeline for progress was first developed at the beginning of October 2018 and then assessed and altered in mid-January 2019. The chosen method for planning the timeline was two Gantt charts, an initial one created in October and a revised one created in January. As explained by Gantt.com [56], A Gantt chart is one of the most popular ways of displaying activities, tasks or events against a timeframe. The left side of the chart shows a list of the activities and the top shows the chosen time scale. Each activity is represented by a bar, with its position and length reflecting the start date, duration and end date.

The first Gantt chart created with the tool GanttPRO [57] during the early planning stages of this project. This Gantt chart was made before any development had begun and did not account for a complete change in technology choices. For better visibility and improved analysis, the full chart has been divided into two sections (*figure 6*).

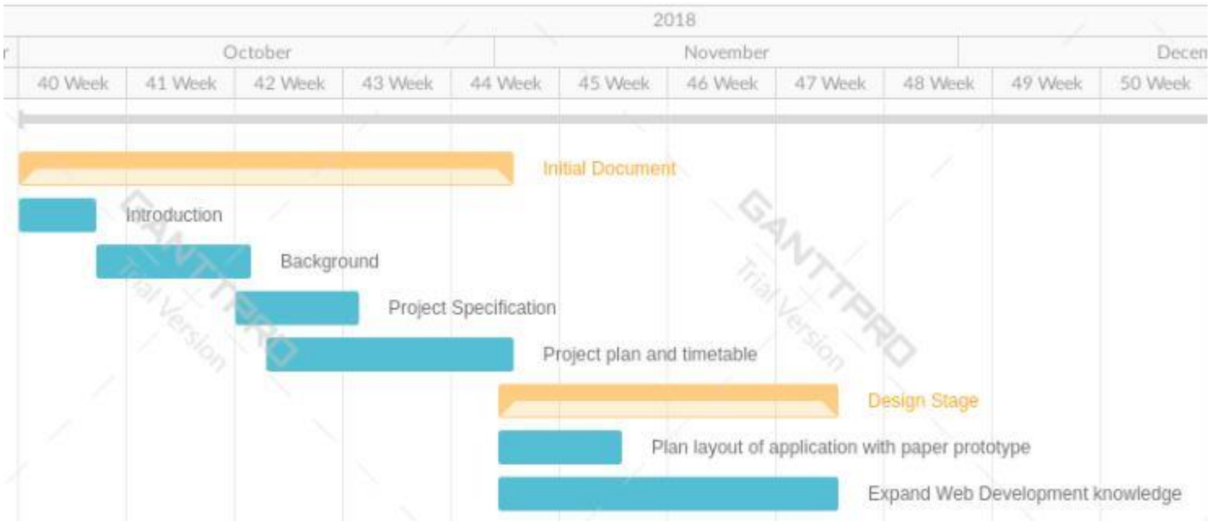


Figure 6: Gantt chart planning activities for the start of October 2018 to the third week of November 2018.

The first section of this chart details the planned activities for the start of October to the third week of November as seen in *figure 6*. These activities took place more or less as planned and progress followed the chart well during these months. However, the timeline set out in the second section of the Gantt chart shows almost no accuracy when compared to the actual timeline of this project (*figure 7*).

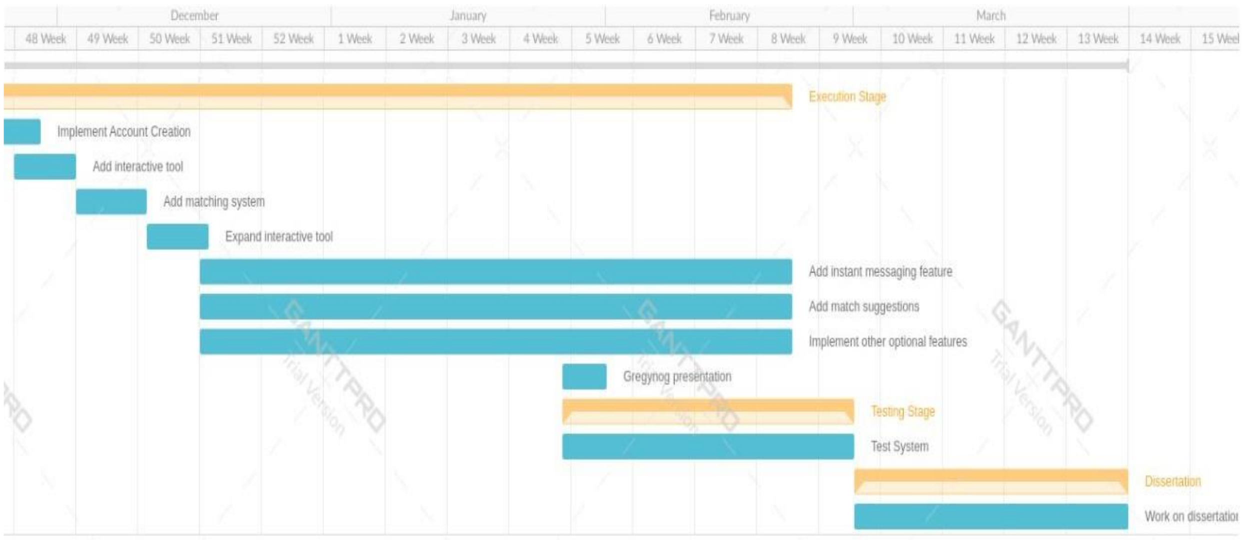
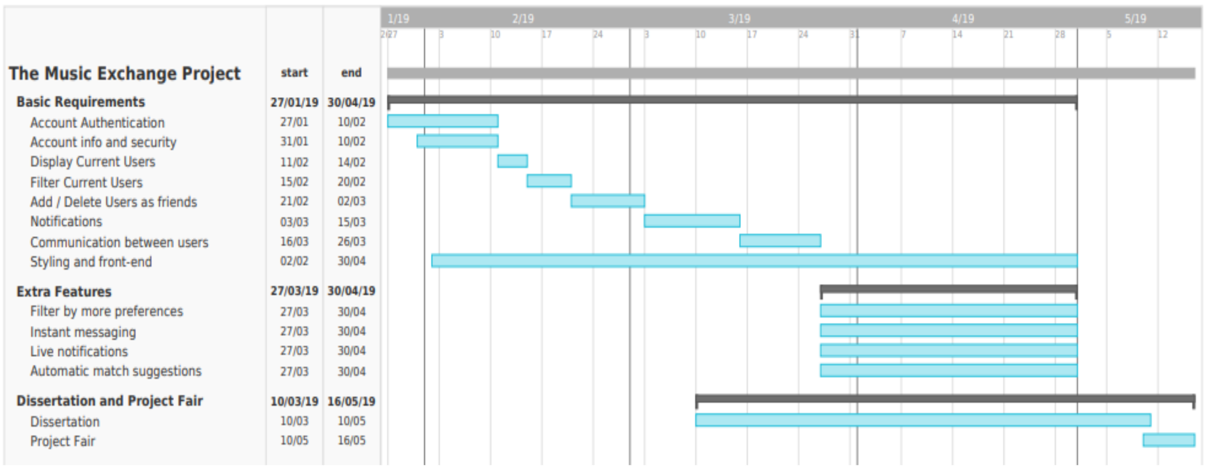


Figure 7: Gantt chart planning activities for the end of November 2018 to the start of April 2019.

In the second section of the Gantt chart (*figure 7*), it was planned that the implementation of the project would take place from the third week of November to the third week of February. This timeline was inaccurate by over a month and failed to account for problems occurring during implementation. Limitations in knowledge and a lack of progress meant activities allocated to a short range of time took much longer than expected.

As January began it was clear that the originally planned timeline no longer held any relevance or accuracy to the progress of the project. During this month the decision was made to assess the current technology choices, plan a new timeline, and rethink the design of the application. A second Gantt chart was created for the remaining timeline of the project and implementation was restarted with new technology choices and a different design (*figure 8*).



The Music Exchange Project	start	end
Basic Requirements	**27/01/19**	**30/04/19**
Account Authentication	27/01	10/02
Account info and security	31/01	10/02
Display Current Users	11/02	14/02
Filter Current Users	15/02	20/02
Add / Delete Users as friends	21/02	02/03
Notifications	03/03	15/03
Communication between users	16/03	26/03
Styling and front-end	02/02	30/04
Extra Features	**27/03/19**	**30/04/19**
Filter by more preferences	27/03	30/04
Instant messaging	27/03	30/04
Live notifications	27/03	30/04
Automatic match suggestions	27/03	30/04
Dissertation and Project Fair	**10/03/19**	**16/05/19**
Dissertation	10/03	10/05
Project Fair	10/05	16/05

Figure 8: Revised Gantt chart planning activities for the end of January 2019 to project completion (May 2019).

The Gantt chart shown in *figure 8* was created with TeamGantt's online Gantt chart software [58], provided an up to date timetable for the progress of this project. Progression followed the timeline with a good amount of accuracy as a whole, with the cumulative timings of feature implementation close to the timings in reality. Certain features took longer or shorter amounts of time, some of the extra features were not implemented and features not detailed by the chart were added. However, certain inaccuracies are expected when planning exact dates, months in advance and the chart fulfilled its purpose and helped plan the overall timings of the project.

4.2 Software Development Methodologies

A methodology can be defined as a system of ways of doing, teaching, or studying something [59]. When choosing a software development methodology to follow for the development of this project, it was important to study the corresponding stages of their Software Development Life Cycles (SDLCs). As explained by Gerald D. Everett and Raymond McLeod [60], A SDLC is a series of stages within a methodology, which are followed in the process of developing and revising an information system. Each stage is a segment that consists of certain types of activities. The classic SDLC consisted of four stages: Planning, Analysis, Design, and Implementation.

The classic SDLC is used by what are considered to be traditional methodologies for software development, the most popular being the Waterfall model. The Waterfall model is a sequential software development process in which progress is regarded as flowing increasingly downwards through a list of stages. Its SDLC comprises five stages: analysis, design, implementation, testing, and maintenance [61]. Due to its design, the Waterfall model is easy to explain to users, has well defined stages and activities, offers management simplicity, and can be used to classify and prioritise tasks [62]. However, the model offers little flexibility and its rigidity makes it difficult to return to any previous stage. Careful planning is therefore required to successfully use the waterfall model.

Next, the modern and popular model Agile was considered. Agile is based on the idea of incremental and iterative development, in which the phases within the life cycle are revisited over and over again [63]. Agile methods put emphasis on teams, customer collaboration and satisfaction, and continuous delivery of small and useful software at a rapid rate. It is extremely useful when a product requirements are subject to change and the development process requires customer interaction throughout.

There are positives and negatives to using either model, with each being chosen based on its suitability within the organization developing the software. If the project requirements change frequently and the project is expected to deliver a product in a short amount of time then the Agile model is the appropriate choice. If the project requirements are clear and well planned then the Waterfall model is the appropriate choice [64].

In the initial planning of this project the developer first considered following the methodology presented by the Waterfall model. This choice was initially made because it was felt that the specification of the project was clear and that the approach would provide a well-structured SDLC to follow for the projects development. However, after carrying out a more in-depth risk assessment it was felt that the Waterfall model was too rigid and an incremental model should be used. An incremental model combines elements of the waterfall model in an iterative fashion [65]. This is perfect for projects where there is a high risk in having to develop the whole system at once and the developer has limited experienced. The model constructs a partial implementation of a total system, and then slowly adds increased functionality. This was appropriate as the basic requirements were not subject to change but other elements like the design and implementation were.

4.3 Backend Planning

The backend or server-side of this application enables the server, application, and database to communicate with each other. The backend focuses on database administration, software architecture structuring the project, logic behind application functionality, security of the application, and many more important aspects. This section explains the design of the database and the relationships between the tables.

4.3.1 Database Design

The design of the database for this project can be split into four areas: Skill related data, relationship related data, and message related data, with user related data acting as the centre point linking everything together. Each of these areas is responsible for managing the data behind the different parts of the application. Within these areas are the necessary tables, keys and relational logic for structuring how everything fits together. This subsection traverses the design of the database area by area, detailing the objects within and explaining the relationships between the tables.

The users table was the most obvious table to include in the database design as the application would not be able to function without providing the ability to make and store the details that structure an account. This table has the most columns because it holds a lot of the data needed to power different features. The primary key for this table is the Id. This holds a unique string which is generated by the ASP.NET Core Identity package when a user registers a new account. Identity also uses a password hasher to hash and salt the user's password to enforce account security. The column PasswordHash in the table holds this value. Use of the Identity package is explained in section 5.3.1.

The other attributes stored in this table are the UserName, Email, Bio, DOB, Name, ProfilePicture, Postcode, Latitude, and Longitude. The postcode, latitude and longitude coordinates are stored so the values can be returned when the distance between users is calculated. The other attributes are stored, so information about the users in the system can be provided.

The next area is designed to hold the tables for skill related data. This is comprised of two tables, Skills and SkillLevels. The table Skills holds the data for each instrument a user plays. The table SkillLevels holds the different experience levels an instrument can be played at. Although, these two tables could theoretically be combined, by splitting them the developer is able to change the skill levels easily without having to update each record of the Skills table.

The primary key for the Skills table is Id, which is simply the number of the record. The Skills table has two foreign keys, UserId and LevelId. This design choice was made so each skill is attributed to the specific id of a user and the specific id of a skill level. The other attributes stored in this table are SkillName and Description, which store the values inputted by the user when customising their skills. The primary key for the SkillLevels table is the Id, also referred to as LevelId. This table only holds the Id and a corresponding Name. The data stored in this table should always stay the same and has three records representing the experience levels, Beginner, Intermediate, and Advanced.

The third area is designed to hold the tables for relationship related data. These are the relationships that are formed between users when they interact with the matching features of the application. This area is comprised of two tables, Connections and Blocks. The table Connections holds the friendships users have formed or tried to form with each other. The table Blocks holds the sets of users that should not be visible to each other.

The primary key of the Connections table is the Id, which is simply the number of the record. The Connections table has two foreign keys, RequestFromId and RequestToId. Each of these attributes links to a specific user from the Users table, and forms the basis of a friendship request. The other attribute the table holds is a Boolean IsConfirmed. This is designed to be used by the application to check if a connection has been mutual. The primary key of the Blocks table is Id, which is also the number of the record. The structure of the Blocks table is very similar to the Connections table. It also has two foreign keys, BlockFromId and BlockToId, which link to specific users. However, as this is designed to block a user from visibility, there is no need for confirmation.

The final area of the database is designed to hold the tables for message related data. These are the messages sent between users and their corresponding groups within the chat system of the application. This area is comprised of three tables, Messages, UserGroups and MessageGroups. The Messages table holds all the data components necessary to form a message in the system. The UserGroups table stores the information for determining what users currently belong to what groups. The MessageGroups table stores the information for establishing the groups that the messages and the users can belong to. By separating the UserGroups and MessageGroups tables the database design looks cleaner and the relationship between the messages and the user groups is easier to follow and use for Pusher channels.

The primary key of the Messages table is the Id and the foreign keys are the MessageGroupId and UserId attributes. Each message is linked to a MessageGroup representing a chat between users and the identity of the user who sent the message. The other attributes in this table are the data components that construct a message. AddedBy stores a username, Message stores the text, and TimeSent stores the date and time that the message was sent.
The primary key of the UserGroups table is the Id and the foreign keys are the MessageGroupId and UserId. This table represents all the instances of users engaged in chats in the application. The only other attribute is UserName, which is stored for recall efficiency.
The primary key of the MessageGroups table is the Id, which is referenced by the UserGroups and Messages. The only other attribute in this table is the GroupName which stores the chat name set by the user.

Every table in this database either has a one to many or many to one relationship, with the user forming the centre point for the majority of the relations. Identifying how the tables fit together to form the database was crucial before any implementation can take place. Trying to mentally visualise every relationship is extremely difficult due to the complexity of the schema. This is why methods such as Entity Relationship Diagrams are used during the design stage of the software development process.

4.3.2 Entity Relationship Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity is an object representing a component of the data, in this case, that would be the data models. An ERD contains different symbols and connectors that visualize the major entities within the system scope and the inter-relationships among these entities [66].

The ERD for the design of the projects database puts the relationships covered in the previous section into a clearer, visual form. This gives us a plan to follow for the implementation and clearly shows the database schema. An online tool called VisualParadigm [67] was used for the construction of the ERD seen in *figure 9*.

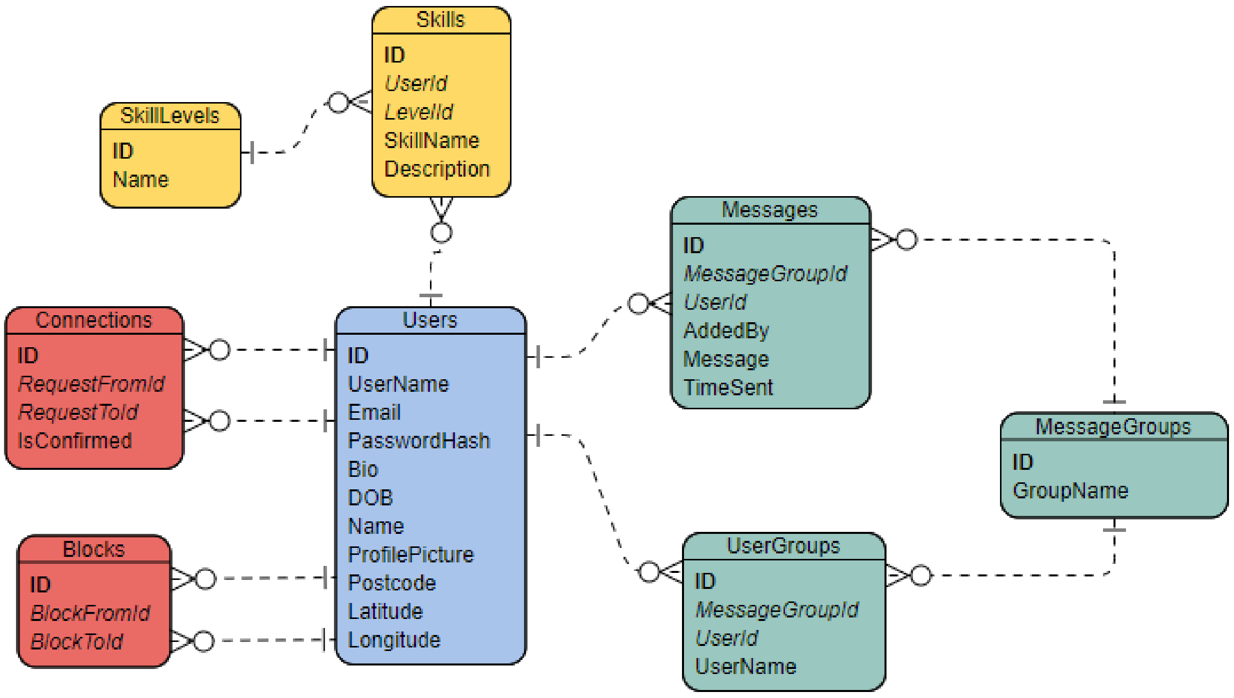


Figure 9: Entity Relationship Diagram for the application database.

This ERD is a modern version that structures the entities to resemble tables instead of using shapes for entity, action and attributes. The primary key of each table is show by the bold writing. The foreign key of each table is show by the italic writing. The diagram is split into four different colours to represent the four main parts of the application. Each line shows the relationship which is either one to many or many to one.

4.4 Frontend Planning

The frontend or client-side of this application determines the actions that take place on the client's computer within their browser. This is the area of the application that the user sees and interacts with. This means the design of the frontend needs to be easy to use, aesthetically pleasing, provide fully functioning features and offer a good user experience.

4.4.1 Colour Scheme

The way colour is used in a web application can completely change how the user perceives it. This is why choosing the right colour scheme is crucial in making this application appealing and memorable. Research into colour and its impact on emotion is well documented in the scientific community. The paper "Relationship between color and emotion: A study of college students" [68], reports the results of an experiment where ninety-eight college students were shown five principle hues, five intermediate hues, and three achromatic colours. They were asked to indicate their emotional responses to each. The study found the colour green attained the highest number of positive responses (95.9%), followed by yellow (93.9%), then blue (79.6%). It was found that green and blue indicated feelings of relaxation, calmness and happiness, and yellow indicated feelings of liveliness and excitement.

A wide range of colour schemes were tested and after conducting research into colour, it was decided that the primary colour for the frontend would be a colour named "Independence" which falls between blue and purple. This decision was made because blue is a colour linked to calmness and as such inspires security and a feeling of safety [69]. However, due to its popularity in web design it was felt that using the primary hue of blue would cause the application to appear unoriginal and a copy of a popular platform such as Facebook, Skype or Twitter. Purple has been known to represent extravagance, creativity, wisdom and peace [70]. Therefore, choosing a colour that falls between both not only gives the application a look of calm and stability, it also provides a uniqueness.

The accent colour for the frontend is a colour named "Tangerine" which is a shade of orange. This decision was made because orange, similar to yellow, adds an element of excitement and vibrancy to the application. It was felt that using a colour such as orange or yellow as the primary colour would bring too much intensity to the design and is better used in to draw the eye to specific parts. To choose the colour scheme the tool Coolors [71] was used to generate and compare different colours side by side. An image of the colour scheme can be seen in *figure 10* below.

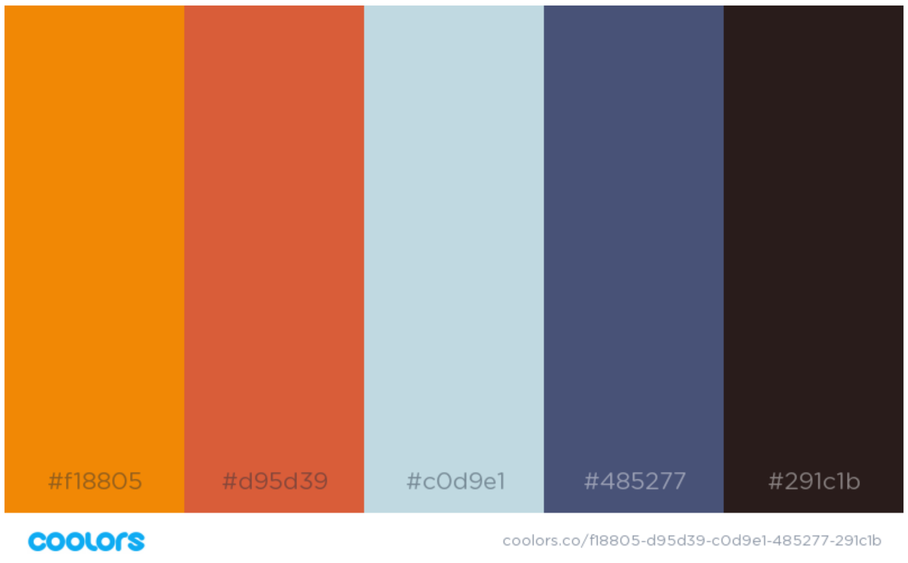


Figure 10: Application colour scheme created with the tool Coolors.

The final consideration when choosing a colour scheme is how it will be perceived by colour blind users. There is a general agreement that 8% of men and 0.5% of women have a colour vision deficiency [72]. The most common form of this is deuteranomalous which makes up approximately 5%, followed by 1% deuteranopes, 1% protanopes, and 1% protanomalous. Coolors offers a useful tool for checking how the colour scheme would be perceived by someone with different forms of colour blindness. *Figure 11* below shows its perception with deuteranomalous on the left and protanopes on the right, with the two conditions not included rendering similar images.

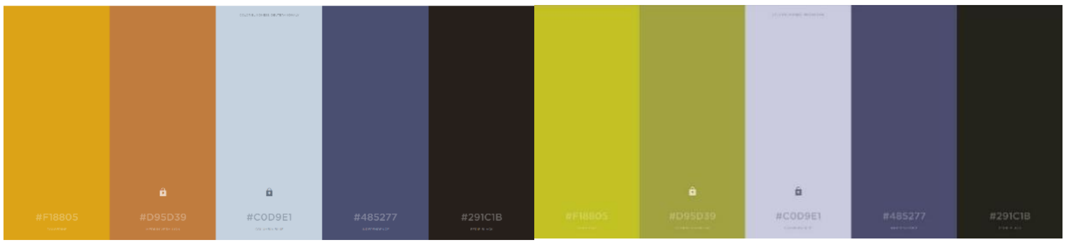


Figure 11: Application colour scheme perception with deuteranomalous (left) and protanopes (right).

Although the accent colours are different, the primary colours have remained almost identical. This supports the choice in colour scheme and assures that the design of the application would not exclude a user with a colour vision deficiency.

4.4.2 Frontend Structure

As the frontend of the application is what the client interacts with, it was necessary to plan the design with focus on creating a positive experience for the client. To do this Nielsen's 10 Usability Heuristics for User Interface Design [73] were considered to produce a user friendly interface. The relevant heuristics and how the application follows it are outlined below:

1. Visibility of system status: the frontend has been designed to be easy to navigate, providing clear feedback on what the application is, how each function works, and the purpose of each area.
2. Match between system and the real world: presentation of information is logical and the application uses language that any English speaking user could understand.
3. User control and freedom: full control is given to leave unwanted states such as the ability to block or disconnect from users, and modify or delete personal information.
4. Consistency and standards: the design is consistent throughout with a navigation bar that has the same actions every time the application is visited.
5. Error prevention: informative error messages are displayed and confirmation is required when the user performs a permanent action such as deleting a skill.
6. Recognition rather than recall: each action is clearly labelled and explained to the user. This has been followed so the interface is self-explanatory.
7. Flexibility and efficiency of use: connections and messages are saved from previous visits so they can be readily available.

8. Aesthetic and minimalist design: profiles were designed as cards to show enough detail to form an impression of the individual, but not so much that the interface becomes cluttered.
9. Error recovery: error messages are displayed in plain English and suggest a solution e.g. this field is required.
10. Help and documentation: instructions on how to use the application is provided and can easily be found on the welcome screen.

5: Implementation

This section of the dissertation provides an overview of the final product that has been produced, covering the features that have been implemented and how each area of the application works. The sections starts by reviewing the specified features, explaining the features that have and have not been implemented. It then separates the application into sections, discussing the implementation of the database, account creation and management, the welcome page, the system for managing skills, the system for searching, filtering and creating relationships, and the chat system. Finally, the section explains how the application was tested and reflects on some of the technical issues that were encountered and how they were resolved.

5.1 Overview of Implemented Features

This subsection provides a summary of the features that have and have not been implemented in this application. It will reference the required and optional features outlined in the specification section of this document (Section 3.1). The implemented features have been coloured green and are accompanied by a tick and a section number indicating the area documenting their implementation. The specified features that have not been implemented have been coloured red and are accompanied by a cross.

The required features were selected to provide an application that could be classified as a minimum working product. Without the implementation of these features the application would be inappropriate for client use.

The required features are as follows:

- Basic management of User accounts: ✓ [Section 5.3]

  1. Interface to register a new account with appropriate input validation. ✓ [Section 5.3.2]
  2. Interface to login with appropriate account authentication. ✓ [Section 5.3.3]
  3. Interface to view account, choose information to display and modify information. ✓ [Section 5.3.4]
  4. Account privacy setting customisation. ✓ [Section 5.3.4]

- Database for storing information: ✓ [Section 5.2]

  5. Storage of data associated with user account. ✓ [Section 5.2]

6. Storage of data associated with skills. ✓ [Section 5.2]
7. Storage of data associated with relationships. ✓ [Section 5.2]

- Appropriate searching system: ✓ [Section 5.6]

8. Instrument(s) played can be linked to the users account. ✓ [Section 5.5]
9. Page to display the current accounts in the system. ✓ [Section 5.6]
10. Search tool for filtering the displayed accounts using user input as a keyword linking to information related to the accounts. ✓ [Section 5.6.3]
11. Interactive tool for filtering the displayed accounts based on instrument(s) played. ✓ [Section 5.6.3]

- System for establishing relationships between users: ✓ [Section 5.6]

12. Button on accounts that stops a target user from being visible or able to contact the current user. ✓ [Section 5.6.2]
13. Button on accounts that prompts the system to automatically send a match request to the specified account from the current user. ✓ [Section 5.6.2]
14. Match requests made visible to the user with the options accept and decline. ✓ [Section 5.6.2]
15. Automatic linking of two accounts when match request is accepted or both users have mutually chosen to connect. ✓ [Section 5.6.2]
16. Communication via email made possible between the matched accounts once linked. ✓ [Section 5.6]

The implementation of these features formed the base of each area of the application. The additional features were chosen to build on this base and improve the quality and experience for the user. Many of the additional features expand on or improve a required feature. A number of the additional features were not implemented due to the time constraints of the development process. During implementation the required features were added before the additional features to ensure the application was functional and met its goals. If development was to continue after the writing of this dissertation, the features that were not implemented would be the primary focus.

The additional features are as follows:

1. Expansion of interactive tool within the searching system to allow filtering by more preferences, such as location/proximity of other users and skill levels. ✓ [Section 5.5 and 5.6]
2. Further Account customisability – Profile pictures, Bio, Age. ✓ [Section 5.3]
3. Scrollable list of accounts the user is connected to. ✓ [Section 5.6.2]
4. Scrollable list of accounts the user has blocked. ✓ [Section 5.6.2]
5. Options to modify existing relationships. ✓ [Section 5.6.2]

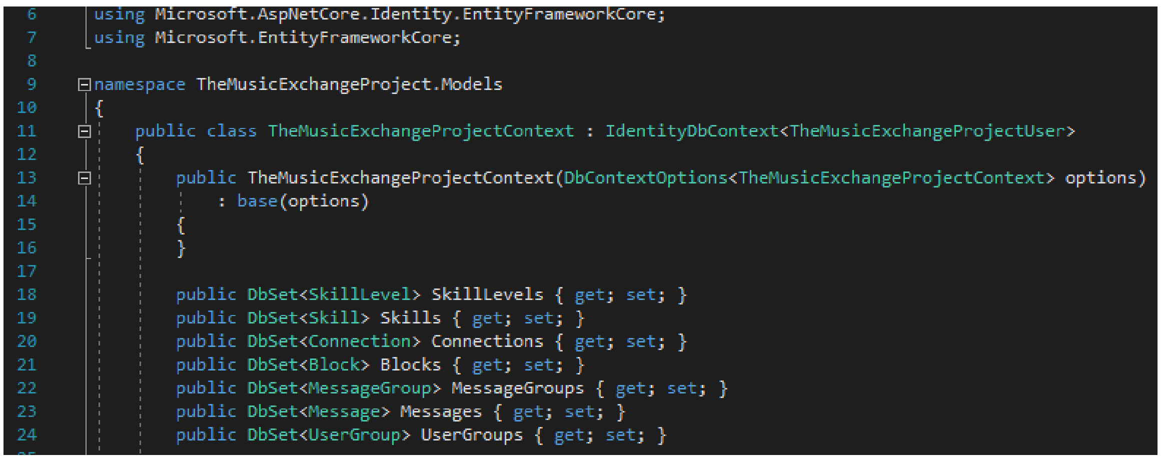
6. Embedded instant messaging tool which can be used for communication between matched users. ✓ [Section 5.7]
7. Live notifications listing message and match requests. ×
8. Match suggestions that are automatically generated by the system using account information such as the instrument a user knows and the instrument a user wants to learn. ×
9. Embedded Video chat tool. ×
10. Expansion of the interests linked to an account, such as genres of music, purpose of account. ×

5.2 Database Implementation

The database for this application was implemented using the ASP.NET Core MVC framework and is created and managed by utilising the NuGet package Entity Framework Core. Entity Framework Core is an object-relational mapper that enables developers to work with data in an object-oriented way [74]. Its implementation provided all the necessary features for database creation and management. This section explores how each feature of Entity Framework Core has been used.

DbContext represents a session with the database and provides an API for communicating with the database. The version implemented in this project is IdentityDbContext which is the base class for the Entity Framework database context specifically used for Identity (covered in section 5.3) [75]. DbContext has many responsibilities within the application, the most important being: Opening and managing connections to the database, providing the methods for performing direct operations on the data, tracking changes to the entities and generating the relevant SQL, model building, and data mapping [76].

It has been implemented with the class TheMusicExchangeProjectContext.cs which can be seen in *figure 12* below.



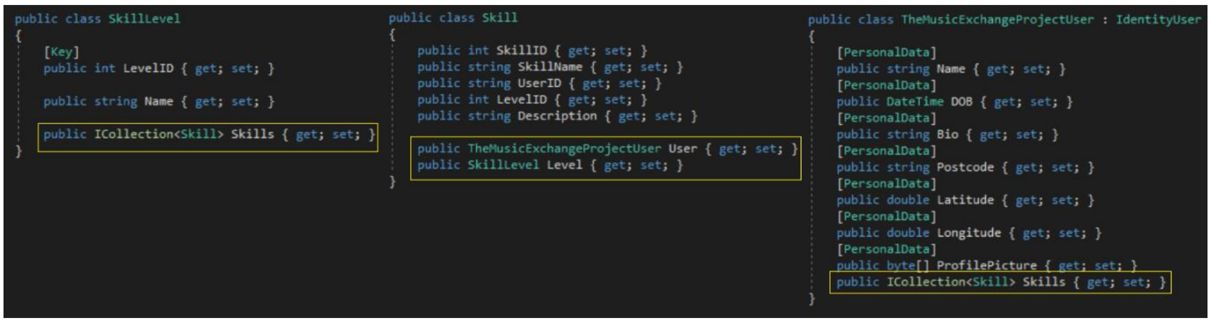
```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace TheMusicExchangeProject.Models
{
    public class TheMusicExchangeProjectContext : IdentityDbContext<TheMusicExchangeProjectUser>
    {
        public TheMusicExchangeProjectContext(DbContextOptions<TheMusicExchangeProjectContext> options)
            : base(options)
        {
        }

        public DbSet<SkillLevel> SkillLevels { get; set; }
        public DbSet<Skill> Skills { get; set; }
        public DbSet<Connection> Connections { get; set; }
        public DbSet<Block> Blocks { get; set; }
        public DbSet<MessageGroup> MessageGroups { get; set; }
        public DbSet<Message> Messages { get; set; }
        public DbSet<UserGroup> UserGroups { get; set; }
```

Figure 12: The class TheMusicExchangeProjectContext.cs, an implementation of DbContext.

This class visibly inherits the functionality defined in the Entity Framework packages base IdentityDbContext class and initialises a DbSet to represent each table in the database. The DbSet<TEntity> class represents a collection for a given entity within a model and is the gateway to performing database operations against an entity [77]. These classes enable the performance of CRUD (Create, Read, Update, Delete) and LINQ queries against a DbSet, which are translated into queries against the database tables.

Each model class defines a table of the database and the attributes within it. With Entity Framework Core, the developer can also define behaviour and relationships between tables by including certain references to the attributes of other tables within the models. This allows the structure of the models to closely match the schema of the database. A demonstration of how the relationship between the users, skills, and skill levels are established is shown below.

A user can have zero or many skills with each skill only assigning to one user. A skill level can be attributed to zero or many skills with each skill assigning to one skill level. Through Entity Framework Core table relationships have been defined using navigation properties (*figure 13*).



```
public class SkillLevel
{
    [Key]
    public int LevelID { get; set; }

    public string Name { get; set; }

    public ICollection<Skill> Skills { get; set; }
}
```

```
public class Skill
{
    public int SkillID { get; set; }
    public string SkillName { get; set; }
    public string UserID { get; set; }
    public int LevelID { get; set; }
    public string Description { get; set; }

    public TheMusicExchangeProjectUser User { get; set; }
    public SkillLevel Level { get; set; }
}
```

```
public class TheMusicExchangeProjectUser : IdentityUser
{
    [PersonalData]
    public string Name { get; set; }
    [PersonalData]
    public DateTime DOB { get; set; }
    [PersonalData]
    public string Bio { get; set; }
    [PersonalData]
    public string Postcode { get; set; }
    [PersonalData]
    public double Latitude { get; set; }
    [PersonalData]
    public double Longitude { get; set; }
    [PersonalData]
    public byte[] ProfilePicture { get; set; }
    public ICollection<Skill> Skills { get; set; }
}
```

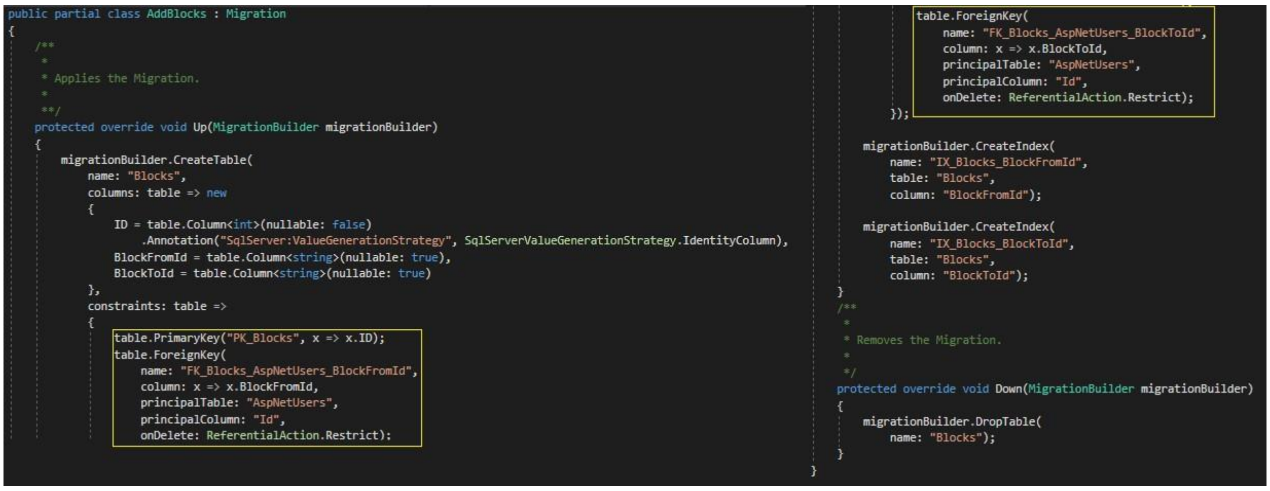
Figure 13: Defining relationships between the Users, Skills, and SkillLevels tables.

To understand navigation properties there needs to be an initial understanding of what dependent entities and principal entities are. A dependent entity is the entity that contains the foreign key property(s). A principal entity is the entity that contains the primary key property(s). A navigation property is defined on the principal and/or dependent entity that contains a reference(s) to the related entity(s) [78].

In this implementation SkillLevel (left) and User (right) are the principle entities and Skill (middle) is the dependent entity. The ICollection navigation property of the User and SkillLevel classes contains references to many skill entities (Yellow boxes in left and right of figure). This establishes the many relationship between the tables as each user and skill level now reference a collection of skills. The Skill.User and Skill.Level properties of the Skill class is a reference navigation property that holds a reference to a single user/skill level (Yellow box in middle of figure). This establishes the one relationship between the tables.

Entity Frameworks migrations feature uses the structure of the models as the database schema to create new tables or make changes to existing table structure within the database [79]. Using the models referenced in the DbContext class, when a migration is created the framework compares the current state of the models with the previous migration (if there is one) and generates a class containing the necessary code for updating the database.

The generated class contains two methods, an Up method that applies any changes made to the model to the schema of the database and a Down method that reverses those changes, restoring the database to the state of the previous migration [80]. An example migration class can be seen in *figure 14* below showing the migration that was used for creating the Blocks table.



```
public partial class AddBlocks : Migration
{
    /**
     *
     * Applies the Migration.
     *
     **/
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Blocks",
            columns: table => new
            {
                ID = table.Column<int>(nullable: false)
                    .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
                BlockFromId = table.Column<string>(nullable: true),
                BlockToId = table.Column<string>(nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Blocks", x => x.ID);
                table.ForeignKey(
                    name: "FK_Blocks_AspNetUsers_BlockFromId",
                    column: x => x.BlockFromId,
                    principalTable: "AspNetUsers",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Restrict):
                table.ForeignKey(
                    name: "FK_Blocks_AspNetUsers_BlockToId",
                    column: x => x.BlockToId,
                    principalTable: "AspNetUsers",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Restrict);
            });

        migrationBuilder.CreateIndex(
            name: "IX_Blocks_BlockFromId",
            table: "Blocks",
            column: "BlockFromId");

        migrationBuilder.CreateIndex(
            name: "IX_Blocks_BlockToId",
            table: "Blocks",
            column: "BlockToId");
    }
    /**
     *
     * Removes the Migration.
     *
     */
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Blocks");
    }
}
```

Figure 14: Migration class for creating the Blocks table.

Entity Framework Core uses the Blocks model to define the schema for the Blocks table in the database *(figure 14)*. The table is created in the Up method with the attributes ID, BlockFromId, BlockToId and their retrospective types defined in the model. The relationships defined in the model are then established in the constraints area, signalled by the yellow box. The option to drop (delete) the table is then established in the Down method. To apply the migration the command Update-Database was run which executes the code in the Up method generating and applying the necessary SQL for updating the database schema.

That concludes the summary of the database implementation. To gain a further understanding of the classes behind the implementation visit the migrations folder of the source code for the full migration history, and the models folder of the source code for each of the models defining a table.

5.3 Account Creation and Management

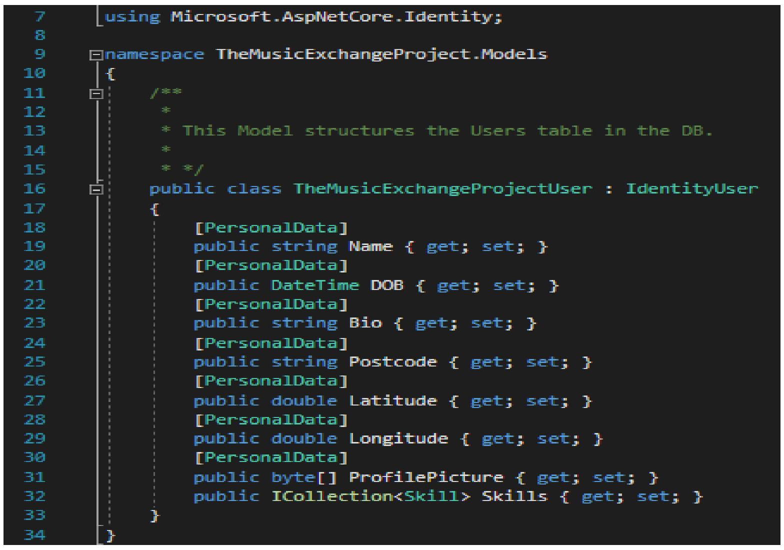
A system for user accounts, registration and login was the first feature to be implemented in this project. A benefit of using the ASP.NET Core MVC framework is the developer can utilise the NuGet package Identity which provides a secure and feature rich system for membership. This section explains what Identity is and how it has been used, following this it covers how the registration and login system works, and finishes by discussing account details and how they are modified.

5.3.1 Identity

ASP.NET Core Identity provides a framework for managing and storing user accounts in ASP.NET Core apps [81]. By default, Identity makes use of an Entity Framework Core data model and is implemented through an initial Entity Framework Core migration. This initial migration created the Users table in the database with some basic attributes for account creation, including; email, username and password. Identity also provided the basic functionality for registration, login and account management. This was a useful starting point for the application and provided a firm base to build on.

Identity contains a feature rich library and provides a sophisticated level of security and authentication to the application. Identity [81] provides useful code for security implementations including password hashing and validation, support for external login providers such as Facebook, email confirmation, password recovery, and many more features that saved development time. Its classes are fully customisable, so have been adapted to suit the requirements and objectives for this project.

After the initial migration of Identity the first customisation was implementing the data attributes that make up an account. This was done by deriving the IdentityUser class in a model for the Users table with custom properties and then applying it with Entity Framework Core migrations. This class can be seen in *figure 15* below.



```
using Microsoft.AspNetCore.Identity;

namespace TheMusicExchangeProject.Models
{
    /**
     *
     * This Model structures the Users table in the DB.
     *
     * */
    public class TheMusicExchangeProjectUser : IdentityUser
    {
        [PersonalData]
        public string Name { get; set; }
        [PersonalData]
        public DateTime DOB { get; set; }
        [PersonalData]
        public string Bio { get; set; }
        [PersonalData]
        public string Postcode { get; set; }
        [PersonalData]
        public double Latitude { get; set; }
        [PersonalData]
        public double Longitude { get; set; }
        [PersonalData]
        public byte[] ProfilePicture { get; set; }
        public ICollection<Skill> Skills { get; set; }
    }
}
```

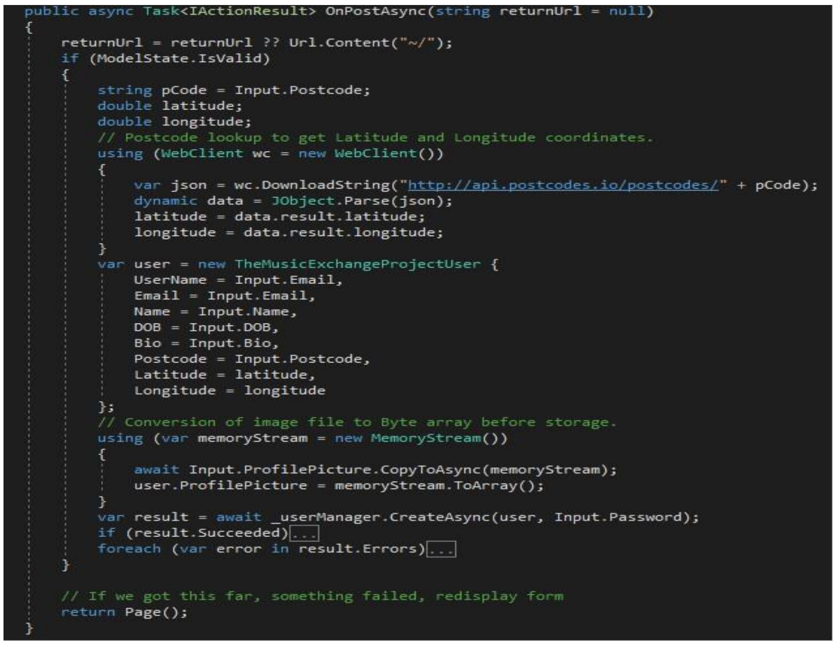
Figure 15: Model defining attributes for User accounts.

As Identity already defines id, username, email and password it was only necessary to define the custom fields for the user accounts table. Another useful feature provided by Identity is users are given full control over their sensitive data. By giving a field the tag PersonalData the user can download or delete their data using the function in the account modification page.

5.3.2 Registration

After updating the Users table with all the attributes that structure an account the next task was modifying the code for registration. The original model contained definitions for Email, Password and ConfirmPassword. This was extended to define the data fields behind the register form for the other account attributes.

After implementing the register model the next step was ensuring the data it holds was correctly used to create a new account when the Registration form is posted. This involved using the OnPostAsync method which is called on form submission. *Figure 16* below shows how this method has been structured.



```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        string pCode = Input.Postcode;
        double latitude;
        double longitude;
        // Postcode lookup to get Latitude and Longitude coordinates.
        using (WebClient wc = new WebClient())
        {
            var json = wc.DownloadString("http://api.postcodes.io/postcodes/" + pCode);
            dynamic data = JObject.Parse(json);
            latitude = data.result.latitude;
            longitude = data.result.longitude;
        }
        var user = new TheMusicExchangeProjectUser {
            UserName = Input.Email,
            Email = Input.Email,
            Name = Input.Name,
            DOB = Input.DOB,
            Bio = Input.Bio,
            Postcode = Input.Postcode,
            Latitude = latitude,
            Longitude = longitude
        };
        // Conversion of image file to Byte array before storage.
        using (var memoryStream = new MemoryStream())
        {
            await Input.ProfilePicture.CopyToAsync(memoryStream);
            user.ProfilePicture = memoryStream.ToArray();
        }
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)...
        foreach (var error in result.Errors)...
    }

    // If we got this far, something failed, redisplay form
    return Page();
}
```

Figure 16: Registration form OnPostAsync method for creating a new account.

This method provides the functionality for dealing with the post request and uses the data held in the model to create an account *(figure 16)*. The method first does a validation check on the state of the submitted model. If successful it assigns the string pCode to the postcode the user has inputted and initialises the latitude and longitude values. A WebClient is then used to make a GET request, with pCode added to the string, to the Postcodes.io API lookup postcode method which returns all information associated with a postcode. The returned string is then parsed into JSON objects which can then be used to identify and assign the latitude and longitude values. This was done to ensure the latitude and longitude coordinates were permanently stored instead of having to make an API call each time the distance between two users is calculated.

Before storing a profile picture, it was necessary to convert the uploaded image file into an array of bytes. This was done by copying the contents of the file to a MemoryStream and then writing the stream to a byte array. This array was then stored in the user object. Finally, using an instance of Identities authentication manager called UserManager, a new user is created with the user object and the password held in the RegisterModel. The UserManager will send requests to the relevant APIs and classes for password hashing, id generation and data storage. The last two methods in *figure 16* (which have been minimised for space), will either sign the user in if account creation was successful or throw an error if unsuccessful.

The registration form defined in the view uses HTML and Tag Helpers to create its structure and display validation messages. Tag Helpers enable server-side code to participate in creating and rendering HTML elements by providing server-side attributes to the elements [82]. This reduces the explicit transitions between HTML and C# in Razor Views. Tag Helpers have been used in almost every page of this application, making the code more reliable and maintainable.

On submission if the state of the input model is invalid, the Tag helpers are used to display the appropriate error message to the screen. The styling of the form is done using Bootstrap/CSS. *Figure 17 and 18* below show how the registration form looks on desktop and on mobile.

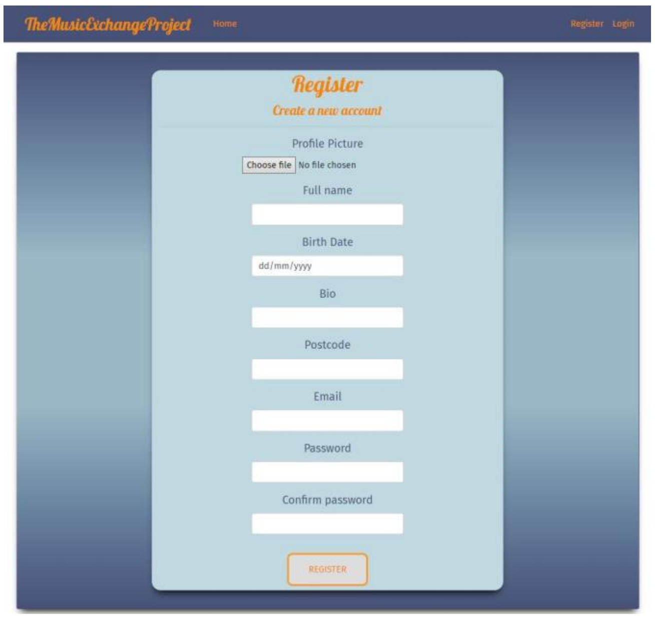


Figure 17: The Register form on desktop.

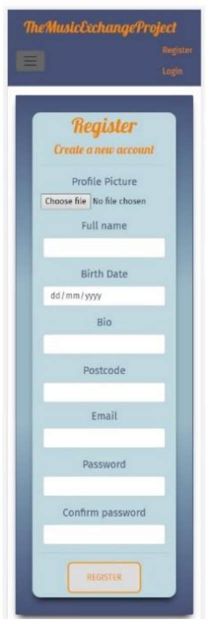


Figure 18: The Register form on mobile. (Combined scrollable screens, iPhone X dimensions).

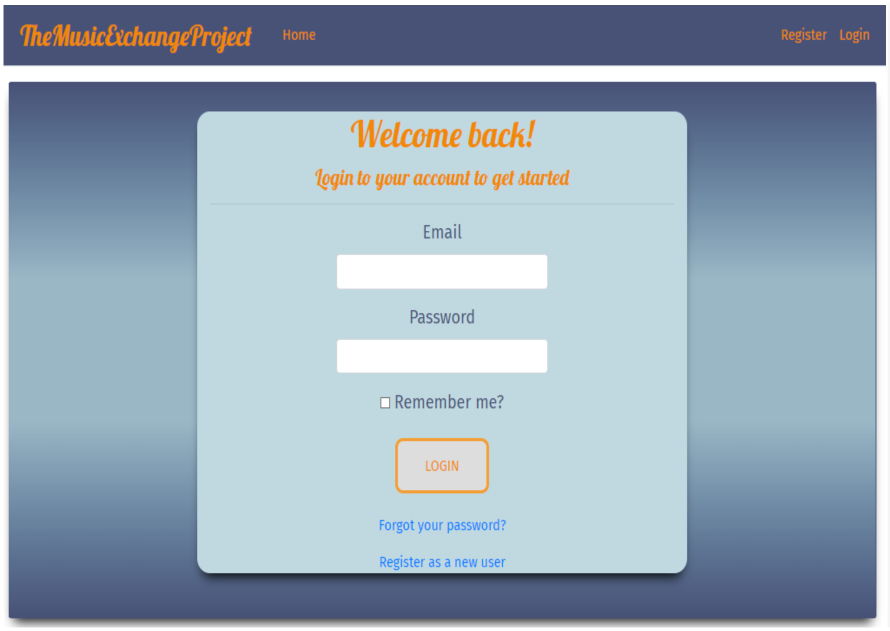
By utilising the Bootstrap framework, the implementation of the form is responsive to screen size. When screen width falls below 1000 pixels the form automatically scales to fill the screen. The design implements the colour scheme chosen in section 4.4.1. The background of the form uses a CSS gradient template generated by the tool ColorSpace [83]. The tool uses the hex codes for the primary blue and secondary blue colours in the colour scheme as input to generate the gradient.

5.3.3 Login

The login system is structured in a very similar way to registration. It is split into the classes Login.cshtml.cs, which contains the input model and the get/post methods, and Login.cshtml, which implements the view. The input model contains the definitions for Email, Password and a RememberMe Boolean. The get method, which is called when the login page is visited, makes sure another user is not signed in and clears any existing cookies to ensure a clean login process when the page is loaded. The post method, which is called when the Login form is submitted, checks that the state of the model is valid before attempting to sign the user in.

Identities SignInManager<TUser> class [84] provides the APIs for user sign in and issuing application cookies. After user input is deemed as valid the SignInManager uses the PasswordSignInAsync method to check if the email and password correspond to an account in the database. If the attempt is successful the user is logged in to the application. If the attempt is unsuccessful an error message is displayed on screen. If the account is locked the user is redirected to a lockout page displaying an account locked error message.

The Login form defined in the view has an almost identical format to the registration form, using HTML and Tag Helpers for structure and validation. The same styling preferences have been used to ensure the design is responsive and in fitting with the rest of the application. *Figure 19* and *20* below show how the login form looks on desktop and mobile.



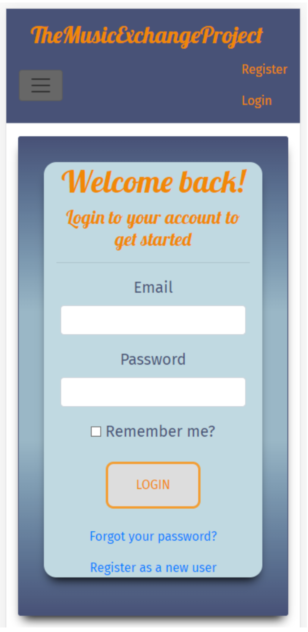


Figure 19: The Login form on desktop.

Figure 20: The Login form on mobile. (iPhone X dimensions).

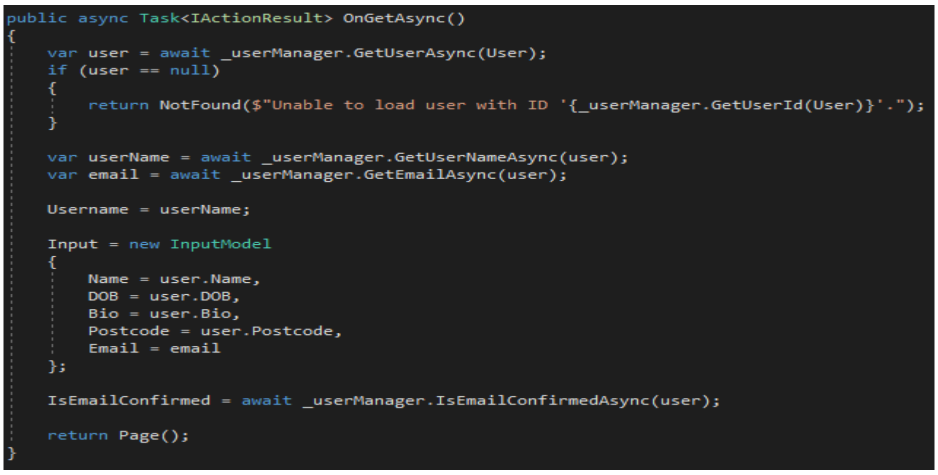
As seen in *figures 19* and *20*, by applying the same styling the design inherits the responsiveness provided by the Bootstrap framework and the characteristics defined in the CSS classes.

5.3.4 Modifying Account Details

The final area of the account creation and management system to discuss is the classes for modifying account details. The classes share the structure of the registration and login areas with Index.cshtml.cs containing the input model and the get/post methods, and Index.cshtml implementing the view.

Similarly to the server-side code of the registration, it implements an OnGetAsync method, which is called when the page is visited, and an OnPostAsync method, which is called when the form in the view is submitted.

The OnGetAsync method for this class plays a bigger role than that of the registration and login forms. Not only does the method load the page, it also maps the current details of the logged in users account to the fields of the form. The method can be seen in *figure 21* below.



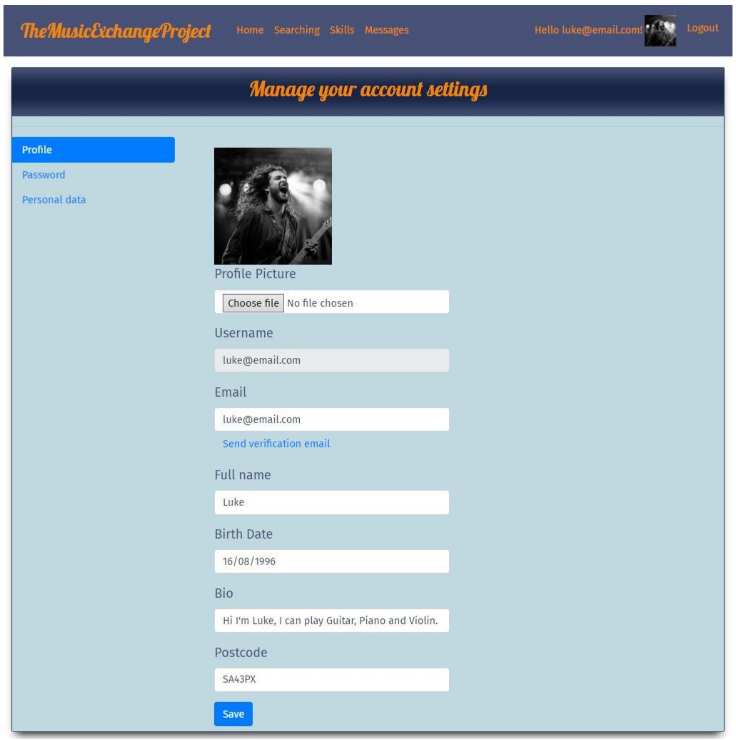
```
public async Task<IActionResult> OnGetAsync()
{
    var user = await _userManager.GetUserAsync(User);
    if (user == null)
    {
        return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
    }

    var userName = await _userManager.GetUserNameAsync(user);
    var email = await _userManager.GetEmailAsync(user);

    Username = userName;

    Input = new InputModel
    {
        Name = user.Name,
        DOB = user.DOB,
        Bio = user.Bio,
        Postcode = user.Postcode,
        Email = email
    };

    IsEmailConfirmed = await _userManager.IsEmailConfirmedAsync(user);

    return Page();
}
```

Figure 21: Implementation of the GET method for mapping current user's details.

The method starts by using an implementation of identities UserManager to get the instance of the currently signed in user from the database. This instance can now be accessed and used as an object. Using the InputModel each data field is assigned to its current value in the database. As the form in the view is mapped to the InputModel, each field will now display the value when the page is returned.

The OnPostAsync method for this class is used to make comparisons between the attributes held by the InputModel on submission, and the current attributes in the users account. Using a series of if statements, each value is individually compared and then updated if it has changed. The same Postcodes.io API request (*figure 16*), is made with the new postcode to update the latitude and longitude values. The same method for converting the profile picture to an array of bytes is used if a new image file has been uploaded.

The account modification view, as with register and login, uses HTML and Tag Helpers for structure and validation. The styling of the form uses the some of the same CSS scheme as the register and login forms, as well as having its own classes. Bootstrap has been used to ensure the page is responsive and fits looks presentable on different devices. *Figures 22* and *23* illustrate the account modification form looks on desktop and mobile.



Manage your account settings

Profile Picture

No file chosen

Username: luke@email.com

Email: luke@email.com

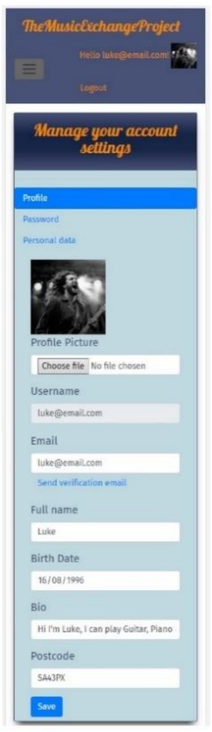
Send verification email

Full name: Luke

Birth Date: 16/08/1996

Bio: Hi I'm Luke, I can play Guitar, Piano and Violin.

Postcode: SA43PX



Manage your account settings

Profile Picture

No file chosen

Username: luke@email.com

Email: luke@email.com

Send verification email

Full name: Luke

Birth Date: 16/08/1996

Bio: Hi I'm Luke, I can play Guitar, Piano

Postcode: SA43PX

Figure 22: The Account Management page on desktop.

Figure 23: The Account Management page on mobile. (Combined scrollable screens, iPhone X dimensions).

On top of providing the form for modifying account details the page also adds navigation to areas where passwords can be reset and personal data can be downloaded or deleted (*figure 22 left*, *figure 23 top*). This gives the client full control over sensitive data and helps the application function ethically.

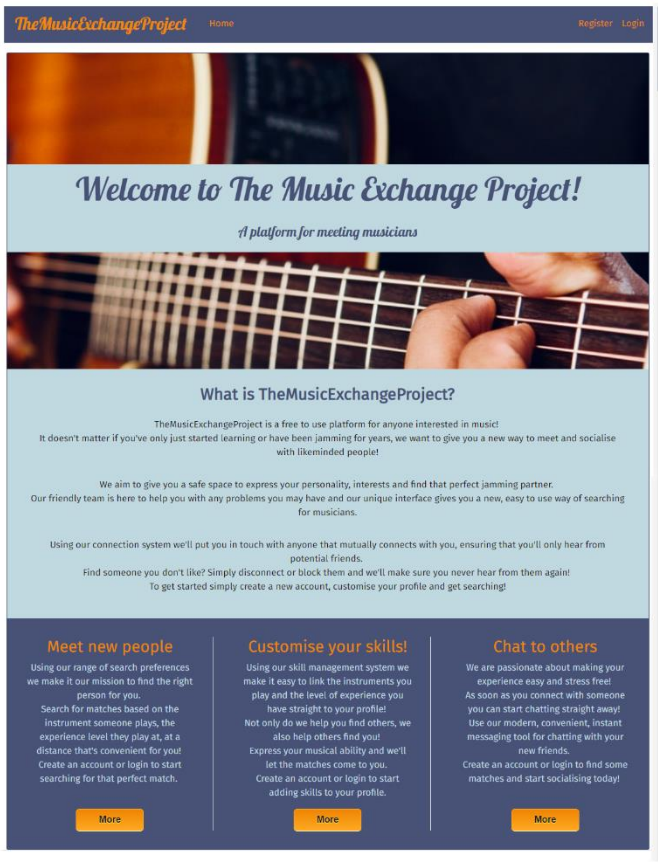
5.4 Welcome Page and Navigation

This section of the document looks at the implementation of the welcome page and explains how the application can be navigated.

5.4.1 Welcome Page

The welcome page is the home page of the project and is the first thing a new user sees. It is a static page [85], meaning it is unchanged and looks exactly the same each time it is loaded. Its frontend implementation is purely HTML, CSS and Bootstrap, requiring no client-side scripting. The server-side implementation requires minimal code, using the HomeController's index method to display the page and a small amount of server-side scripting to redirect the users that click the "More" buttons. Its purpose is simply to provide a new or existing user with information about what TheMusicExchangeProject is and some of the main features that are available.

Although simple, the welcome page still plays an important role in the application. It is the first impression a user gets when they visit the site so has to essentially sell the product. This has meant the focus when implementing the page has been almost entirely on the styling and presented information. The colour scheme chosen in section 4.4.1 has been used for the design but in a nonconventional way. *Figure 24* is a screenshot of the welcome page on a desktop screen.



Welcome to The Music Exchange Project!

A platform for meeting musicians

What is TheMusicExchangeProject?

TheMusicExchangeProject is a free to use platform for anyone interested in music!
It doesn't matter if you've only just started learning or have been jamming for years, we want to give you a new way to meet and socialise with likeminded people!

We aim to give you a safe space to express your personality, interests and find that perfect jamming partner.
Our friendly team is here to help you with any problems you may have and our unique interface gives you a new, easy to use way of searching for musicians.

Using our connection system we'll put you in touch with anyone that mutually connects with you, ensuring that you'll only hear from potential friends.
Find someone you don't like? Simply disconnect or block them and we'll make sure you never hear from them again!
To get started simply create a new account, customise your profile and get searching!

Meet new people

Using our range of search preferences we make it our mission to find the right person for you.
Search for matches based on the instrument someone plays, the experience level they play at, at a distance that's convenient for you!
Create an account or login to start searching for that perfect match.

Customise your skills!

Using our skill management system we make it easy to link the instruments you play and the level of experience you have straight to your profile!
Not only do we help you find others, we also help others find you!
Express your musical ability and we'll let the matches come to you.
Create an account or login to start adding skills to your profile.

Chat to others

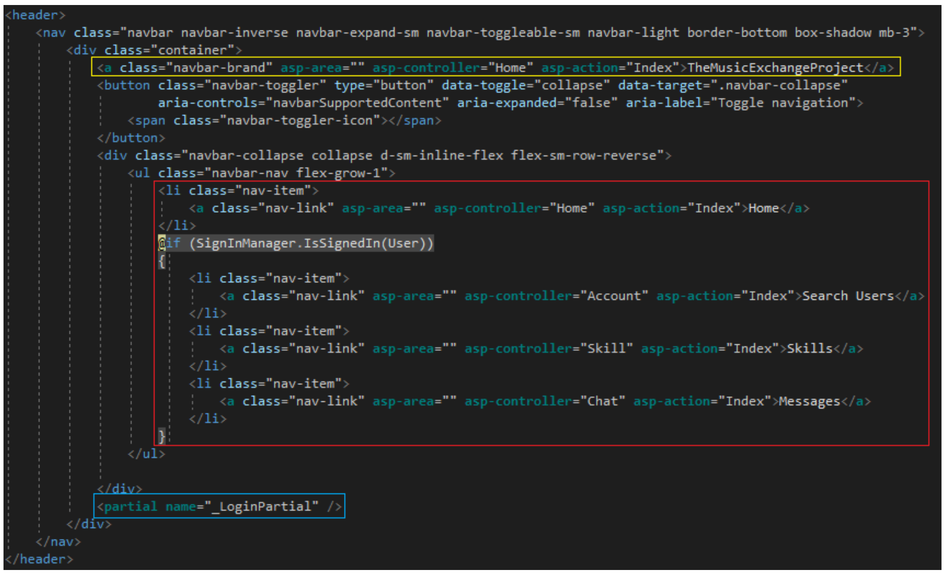
We are passionate about making your experience easy and stress free!
As soon as you connect with someone you can start chatting straight away!
Use our modern, convenient, instant messaging tool for chatting with your new friends.
Create an account or login to find some matches and start socialising today!

Figure 24: The Welcome Page on desktop.

As shown in *figure 24*, each section of the Welcome Page has a different styling, using different primary and accent colours from the colour scheme. This decision was made to make the welcome page distinctive and memorable, as well as providing a clear separation of information to encourage reading.

5.4.2 Navigation and Layout View

Whilst navigation to the three main sections can be done through the welcome page, the main tool is the navigation bar fixed to the top of the application. The navigation bar is created in the _Layout.cshtml file which is a layout view. A layout view is a feature of ASP.NET Core MVC that is used to contain common UI parts that remain the same throughout the application including logos, headers, navigation bars and footers [86]. It has been used to define a common site template, which is inherited in every view of the application providing a consistent look throughout and eliminating the need for duplicate code. The navigation bar implemented in the layout view can be seen in *figure 25* below.



```
<header>
    <nav class="navbar navbar-inverse navbar-expand-sm navbar-toggleable-sm navbar-light border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">TheMusicExchangeProject</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse"
                    aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                    </li>
                    @if (SignInManager.IsSignedIn(User))
                    {
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Account" asp-action="Index">Search Users</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Skill" asp-action="Index">Skills</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Chat" asp-action="Index">Messages</a>
                        </li>
                    }
                </ul>

            </div>
            <partial name="_LoginPartial" />
        </div>
    </nav>
</header>
```

Figure 25: Code implementing the navigation bar.

The navigation bar is implemented as the header of the page to ensure it is fixed to the top. The yellow box (figure 25) highlights how the logo which sits at the far left of the navigation bar was added and navigates to the welcome page. The red box (*figure 25*) creates the navigation items which can be used to access each area of the application.

Using dependency injection, the SignInManager has been injected to check if a user is signed in. This ensures the navigation items for the areas requiring membership are only displayed to a user with an account. The code inside the blue box (*figure 25*) renders a partial view, displaying the navigation items for registration, login, account management, and logout. A partial view is a reusable view, which can be used as a child view in multiple other views [87]. Similar to layout views, they are used to eliminate duplicate code by reusing the same partial view in multiple places.

The styling of the navigation bar uses a mixture of Bootstrap and CSS classes. Bootstrap makes the navigation bar responsive to screen size and will collapse the bar if the width of the screen is less than 576 pixels. This is useful to prevent the bar from occupying too much screen space on smaller devices. The CSS follows the colour scheme using the primary colour for the background of the bar and the accent colour for the navigation items. The logo uses the brighter tangerine colour from the scheme to make it highly visible and memorable. A labelled version of the navigation bar can be seen in *figure 26* below.

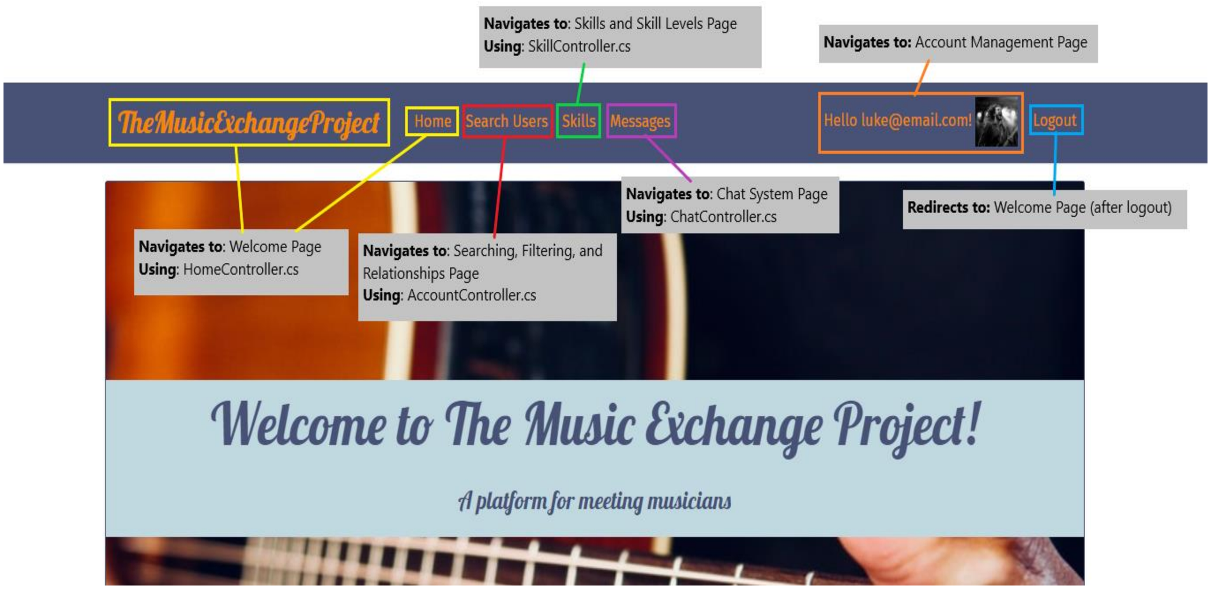

Navigates to: Skills and Skill Levels Page
Using: SkillController.cs
Navigates to: Account Management Page
TheMusicExchangeProject
Home
Search Users
Skills
Messages
Hello luke@email.com!
Logout
Navigates to: Chat System Page
Using: ChatController.cs
Redirects to: Welcome Page (after logout)
Navigates to: Welcome Page
Using: HomeController.cs
Navigates to: Searching, Filtering, and Relationships Page
Using: AccountController.cs
Welcome to The Music Exchange Project!
A platform for meeting musicians

Figure 26: Labelled navigation bar showing where each item leads.

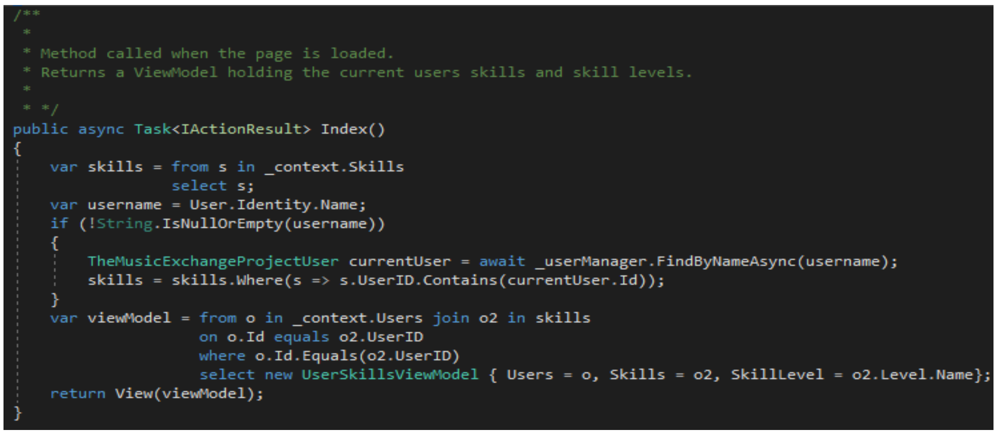
Figure 26 provides an overview of how the navigation bar works and where each item leads. Different controllers are used to power each area and display the relevant page.

5.5 Skills and Skill levels

The skills area of the application implements CRUD functionality through server-side scripting to provide functions for adding new skills, viewing skills, and deleting skills. This is a crucial part of the application that makes it possible for a user to be linked to the musical instruments they play and the levels they play at. The searching system for the application depends on the information that is added from this area. Each of the systems functions is outlined in the proceeding sections.

5.5.1 Viewing Skills

When the skills page is visited the Index view is displayed showing the current users skills and skill levels in lists structured as cards. This is accomplished by calling the Index method inside the SkillController, getting and returning the data to the view. *Figure 27* shows the implementation of this method.



```
/**
 *
 * Method called when the page is loaded.
 * Returns a ViewModel holding the current users skills and skill levels.
 *
 * */
public async Task<IActionResult> Index()
{
    var skills = from s in _context.Skills
                 select s;
    var username = User.Identity.Name;
    if (!String.IsNullOrEmpty(username))
    {
        TheMusicExchangeProjectUser currentUser = await _userManager.FindByNameAsync(username);
        skills = skills.Where(s => s.UserID.Contains(currentUser.Id));
    }
    var viewModel = from o in _context.Users join o2 in skills
                    on o.Id equals o2.UserID
                    where o.Id.Equals(o2.UserID)
                    select new UserSkillsViewModel { Users = o, Skills = o2, SkillLevel = o2.Level.Name};
    return View(viewModel);
}
```

Figure 27: Code for the Index method of the SkillController.

The method (*figure 27*) uses LINQ queries to get and merge data from the Users and Skills database tables. All data is read from the database by making a query to an instance of the applications DbContext (see Section 5.2). A ViewModel is then used to store the results of a join between the users table and skills table, specifically the current user's details, the current user's skills and corresponding skill levels.

ViewModels are classes that contain the fields which are represented in a strongly-typed view [88]. They have been used throughout this application by the controllers to hold entities that come from different models or data sources and pass them to a view. A foreach loop assigns each card to a different skill and fills the list items with the skill name, description and skill level. *Figure 28* shows the cards generated from the data in the ViewModel.

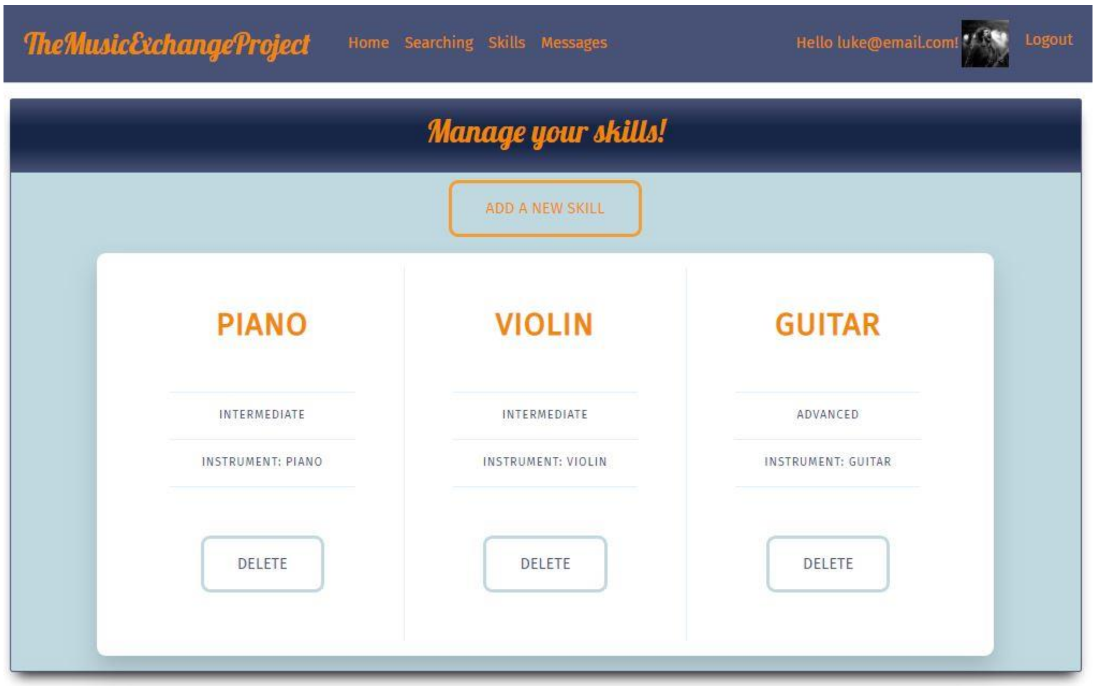


Figure 28: The skill management page on desktop.

All of the users' skills and levels are displayed with a delete option (*figure 28*). When the delete option is clicked the corresponding method in the SkillController is called with the SkillId as a parameter. This is necessary to correctly identify the skill to perform the operation on. The styling uses HTML, Bootstrap and CSS to display a maximum of three cards before adding any additional to a new line. When the page is resized to fit a mobile screen the cards are individually displayed.

5.5.2 Creating Skills

By default, no skills are linked to the user's account. Skills must be created before the account is visible in the search system. SkillController implements a GET and POST method for skill creation. When the 'Create new skill' button in the index view is clicked the HTTP GET method Create is called displaying the create view. *Figure 29* shows the design of the create form.

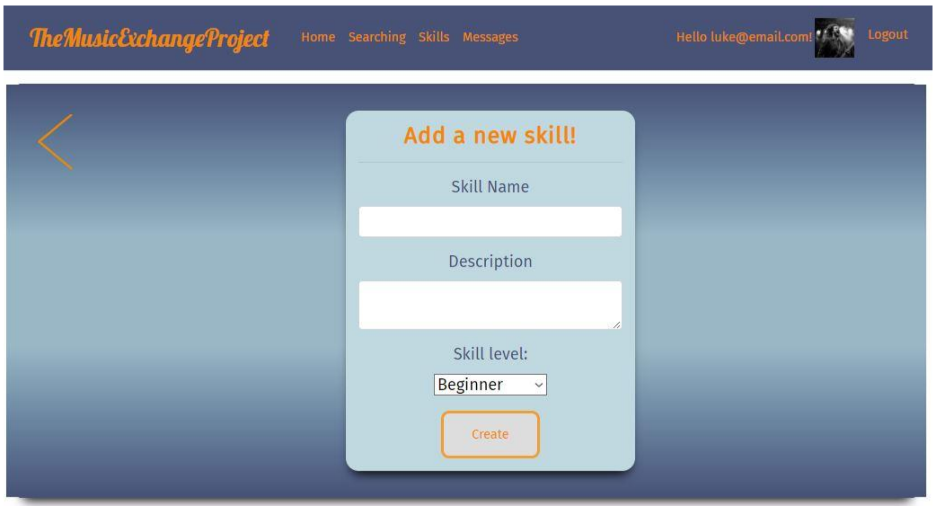


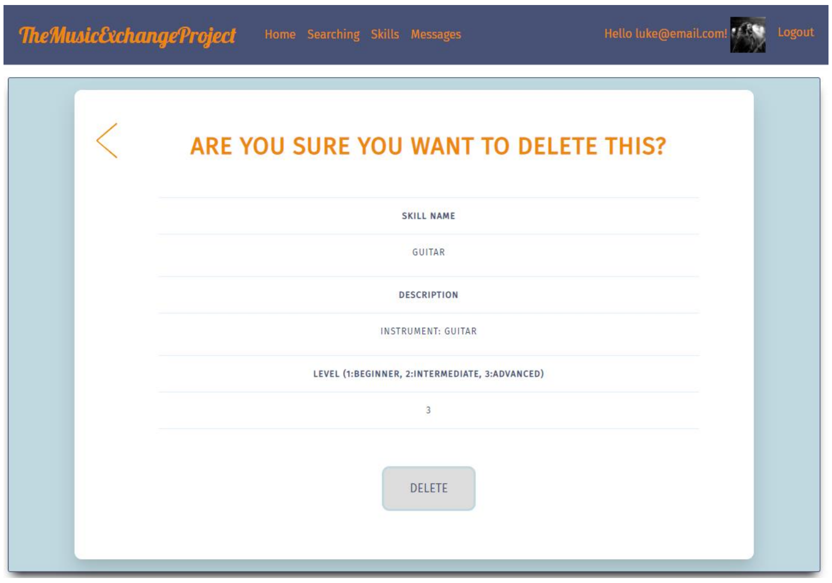
Figure 29: The skill creation form on desktop.

As demonstrated in *figure 29*, the view displays a form for getting the user input necessary to create a new skill in the database. Each field in the form is bound to an instance of the Skill.cs model, defining the attributes of the database table. When the form is submitted the HTTP POST method Create is called with the form data as the parameter. If the input inside the form fields satisfies the requirements of the model a LINQ query is used to add the new skill to the database.

The page uses the majority of the same CSS and Bootstrap classes as the registration and login forms. An arrow button (*figure 29, top left*) returns the client to the skill management page (*figure 28*) and cancels the transaction.

5.5.3 Deleting Skills

The client may choose to delete a current skill by clicking the corresponding delete button in the Index view. Once clicked, this calls the Delete GET method in the SkillController. The page displayed shows the full details of the skill and requests confirmation. This has been implemented to stop accidental deletion from occurring. If the deletion is confirmed, the DeleteConfirmed POST method in the SkillController is executed. If the deletion is cancelled, the user is returned to the Index page. *Figure 30* shows the design of the deletion form.



ARE YOU SURE YOU WANT TO DELETE THIS?

SKILL NAME

GUITAR

DESCRIPTION

INSTRUMENT: GUITAR

LEVEL (1:BEGINNER, 2:INTERMEDIATE, 3:ADVANCED)

3

Figure 30: The skill deletion form on desktop.

As demonstrated in *figure 30*, the view displays the current details of the skill and confirms the action when the delete button is clicked. The styling of the form uses the majority of the same CSS as the skill management page to provide a unified look. An arrow button identical to the one in *figure 29* is used to return the client to the skill management page (*figure 28*) and cancel the transaction.

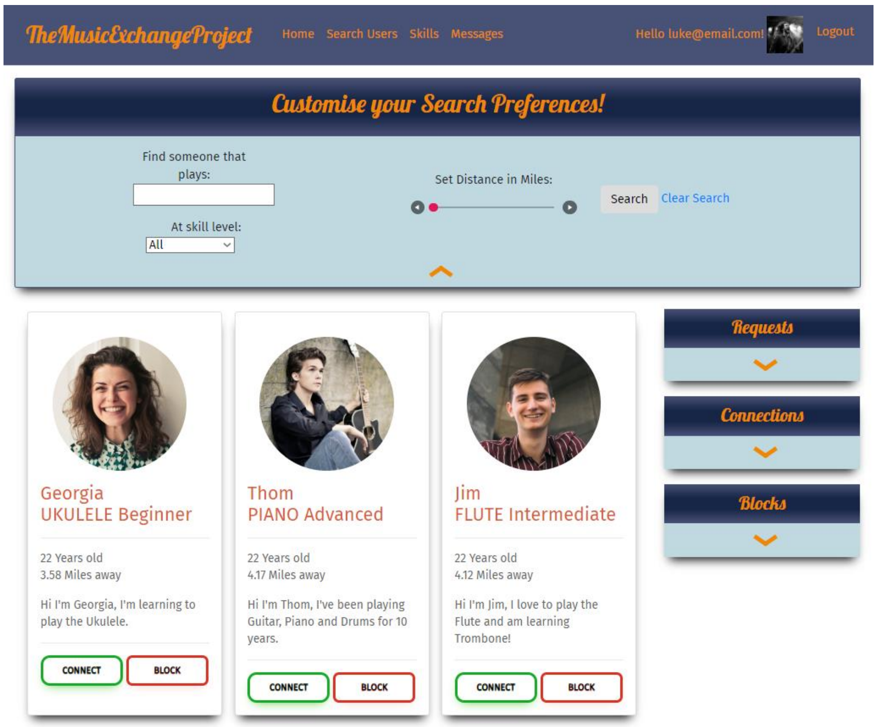
When deletion is confirmed the POST method DeleteConfirmed, is called with the SkillId as a parameter. Using LINQ it finds and removes the skill from the DbContext before redirecting to the skill management page (*figure 28*).

5.6 Searching, Filtering, and Relationships

The system for searching, filtering and forming relationships between users is arguably the most important area of the application. It allows the application to fulfil its aim of being a match making system for musicians and music hobbyists. The implementation of this system produced the largest controller, AccountController, due to the amount of functionality and features it contains. This section looks at the frontend design and how the user interacts with it, before breaking down the algorithms behind the features.

5.6.1 Design and Structure

The design of the user interface can be broken down into three areas. At the top of the page there is a tool for searching and filtering the user profiles by instrument, skill level, and distance. In the main body of the page, there are two sections; a left column displaying the profiles and a right column displaying connection requests, current connections and current blocks. A mixture of Bootstrap and CSS classes have been used to achieve the styling and ensure the page is resizable on different devices. A small amount of jQuery script is used to show and hide and the content. *Figures 31* and *32* show how the page looks on desktop and mobile.

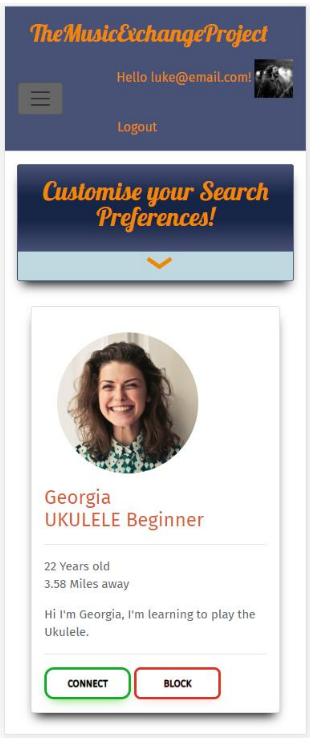


Georgia
UKULELE Beginner
22 Years old
3.58 Miles away
Hi I'm Georgia, I'm learning to play the Ukulele.

Thom
PIANO Advanced
22 Years old
4.17 Miles away
Hi I'm Thom, I've been playing Guitar, Piano and Drums for 10 years.

Jim
FLUTE Intermediate
22 Years old
4.12 Miles away
Hi I'm Jim, I love to play the Flute and am learning Trombone!

Figure 31: The searching page on desktop.



Georgia
UKULELE Beginner
22 Years old
3.58 Miles away
Hi I'm Georgia, I'm learning to play the Ukulele.

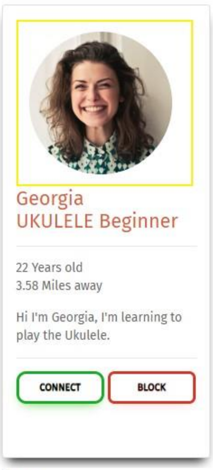
Figure 32: The searching page on mobile. (iPhone X dimensions).

When used, the searching and filtering tool (*figure 31*) manipulates the profiles that appear underneath in the left column. The left column is styled to occupy 75% of the screen space to ensure the profiles are the main focus on the page. When the width falls below 800 pixels this is changed to occupy 100% of the width, with the right column appearing underneath at the bottom of the page (*figure 32*). This ensures the system can be used on smaller devices without a decrease in visibility of the profile information.

5.6.2 Profiles and Relationships

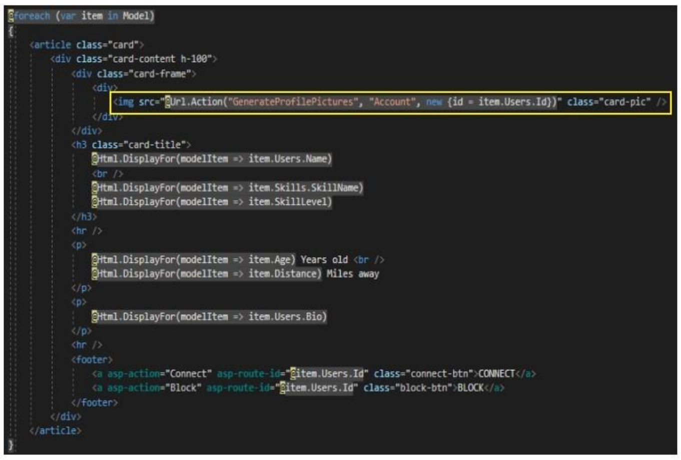
Each profile is structured as a card displaying the profile picture, name, skill and skill level, age, distance and bio of the user. This design was chosen so each card provides enough information to allow the current user to form an impression of who the person is before choosing to create a relationship between their accounts.

The profile picture is displayed using the GenerateProfilePictures method in the AccountController, as it needs to be converted from the byte array to an image file. *Figure 33* shows how each profile card is structured and how this method is called.


Georgia
UKULELE Beginner

22 Years old
3.58 Miles away

Hi I'm Georgia, I'm learning to play the Ukulele.


```
@foreach (var item in Model)
{
    <article class="card">
        <div class="card-content h-100">
            <div class="card-frame">
                <div>
                    <img src="@Url.Action("GenerateProfilePictures", "Account", new {id = item.Users.Id})" class="card-pic" />
                </div>
            </div>
            <h3 class="card-title">
                @Html.DisplayFor(modelItem => item.Users.Name)
                <br />
                @Html.DisplayFor(modelItem => item.Skills.SkillName)
                @Html.DisplayFor(modelItem => item.SkillLevel)
            </h3>
            <hr />
            <p>
                @Html.DisplayFor(modelItem => item.Age) Years old <br />
                @Html.DisplayFor(modelItem => item.Distance) Miles away
            </p>
            <p>
                @Html.DisplayFor(modelItem => item.Users.Bio)
            </p>
            <hr />
            <footer>
                <a asp-action="Connect" asp-route-id="@item.Users.Id" class="connect-btn">CONNECT</a>
                <a asp-action="Block" asp-route-id="@item.Users.Id" class="block-btn">BLOCK</a>
            </footer>
        </div>
    </article>
}
```

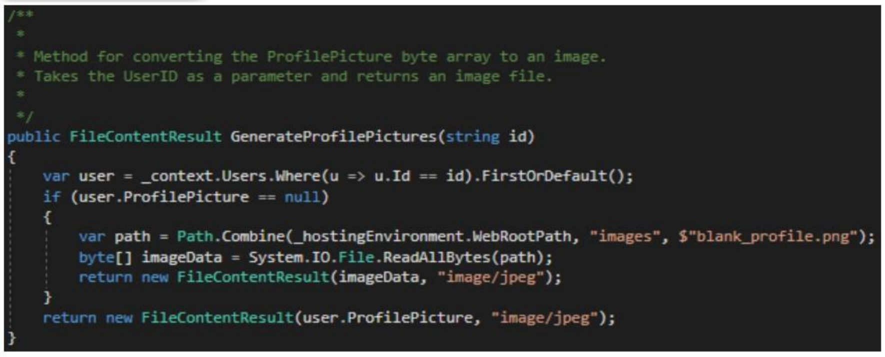

```
/**
 *
 * Method for converting the ProfilePicture byte array to an image.
 * Takes the UserID as a parameter and returns an image file.
 *
 */
public FileContentResult GenerateProfilePictures(string id)
{
    var user = _context.Users.Where(u => u.Id == id).FirstOrDefault();
    if (user.ProfilePicture == null)
    {
        var path = Path.Combine(_hostingEnvironment.WebRootPath, "images", $"blank_profile.png");
        byte[] imageData = System.IO.File.ReadAllBytes(path);
        return new FileContentResult(imageData, "image/jpeg");
    }
    return new FileContentResult(user.ProfilePicture, "image/jpeg");
}
```

Figure 33: The code for structuring profile cards and generating a profile picture.

The source of the profile picture is provided by using a UrlHelper action to call the GenerateProfilePictures method with the UserId as the parameter (*figure 33, yellow box in top right*). If a profile picture has been uploaded, the GenerateProfilePictures method will return the byte array as an image file. If a profile picture has not been uploaded or cannot be found, a default picture from the wwwroot is provided.

Each card provides the options for connecting with or blocking the user. When the connect option is clicked the Connect method in the AccountController is called, with the corresponding id of the user. *Figure 34* shows the implementation of the Connect method.

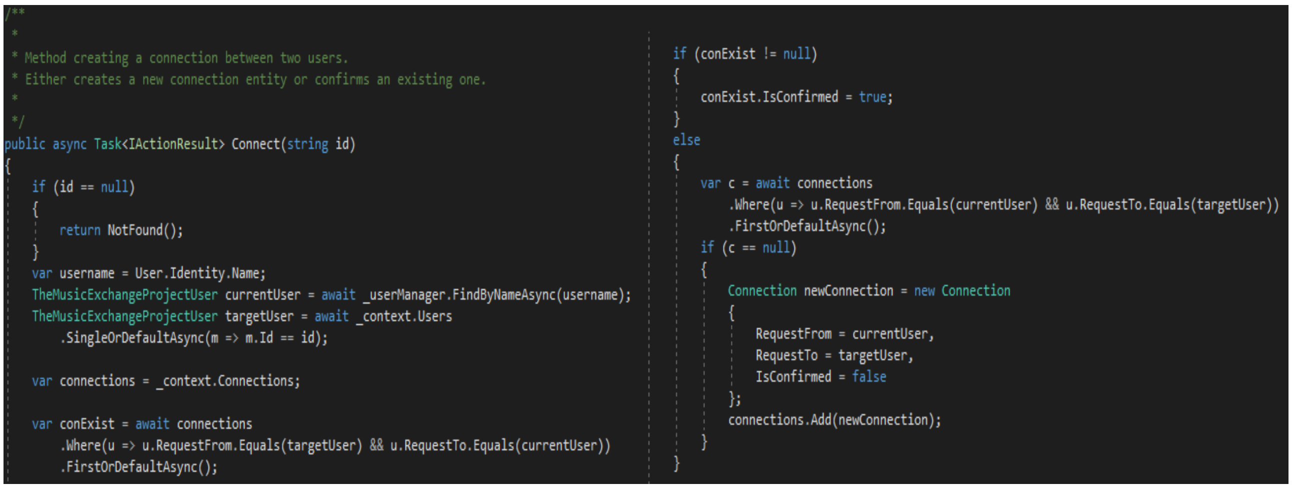
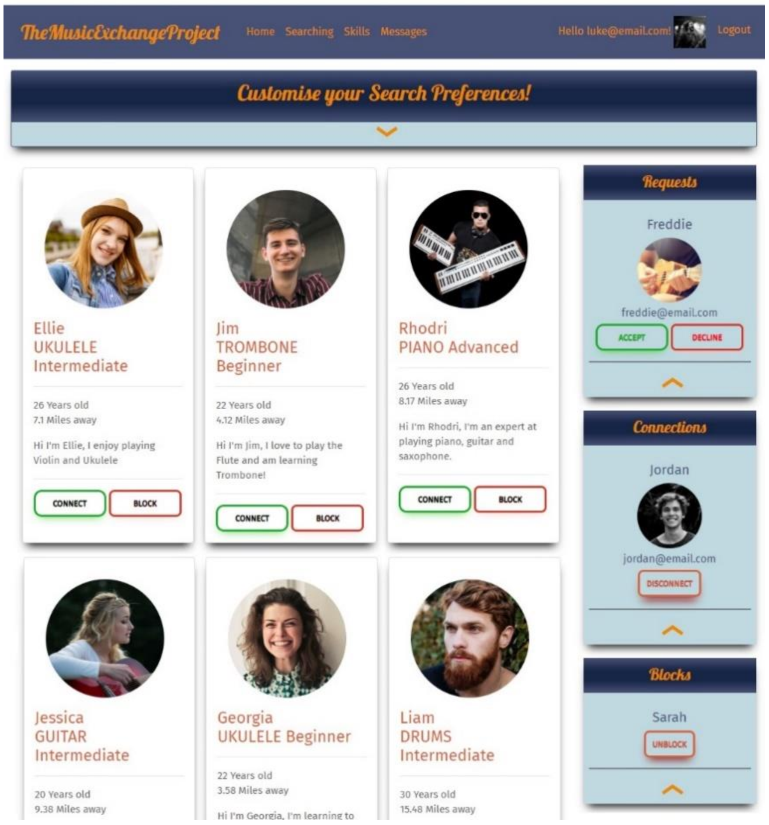

```
/**
 *
 * Method creating a connection between two users.
 * Either creates a new connection entity or confirms an existing one.
 *
 */
public async Task<IActionResult> Connect(string id)
{
    if (id == null)
    {
        return NotFound();
    }
    var username = User.Identity.Name;
    TheMusicExchangeProjectUser currentUser = await _userManager.FindByNameAsync(username);
    TheMusicExchangeProjectUser targetUser = await _context.Users
        .SingleOrDefaultAsync(m => m.Id == id);

    var connections = _context.Connections;

    var conExist = await connections
        .Where(u => u.RequestFrom.Equals(targetUser) && u.RequestTo.Equals(currentUser))
        .FirstOrDefaultAsync();

    if (conExist != null)
    {
        conExist.IsConfirmed = true;
    }
    else
    {
        var c = await connections
            .Where(u => u.RequestFrom.Equals(currentUser) && u.RequestTo.Equals(targetUser))
            .FirstOrDefaultAsync();
        if (c == null)
        {
            Connection newConnection = new Connection
            {
                RequestFrom = currentUser,
                RequestTo = targetUser,
                IsConfirmed = false
            };
            connections.Add(newConnection);
        }
    }
```
Figure 34: The code for the connect method, taken from the AccountController.

Figure 34 demonstrates how the method uses id and an instance of UserManager to assign the current and target users associated with the connection. The Connections table in the database is queried to check if the target user has already attempted to connect with the current user. If this is the case, the existing connection is confirmed by setting the IsConfirmed Boolean to true. This ensures all connections are mutual and halves the number of records in the database. If there are no existing records of a connection between the users, a new unconfirmed connection entity is added to the database. This new connection will not be interpreted as valid until the target user mutually connects or accepts the request.

When one of the arrows in the searching page (*figure 35*) is clicked jQuery script is executed to toggle the CSS of the content from hidden to shown and transition the arrow to face the opposite direction. *Figure 35* contains a screenshot of the searching page with the requests, connections and blocks lists toggled to shown.



Hi I'm Ellie, I enjoy playing Violin and Ukulele

Hi I'm Jim, I love to play the Flute and am learning Trombone!

Hi I'm Rhodri, I'm an expert at playing piano, guitar and saxophone.

Hi I'm Georgia, I'm learning to

Figure 35: The searching page with all lists open on desktop.

All connection requests appear in the first list of the right column with the options accept and decline (*figure 35*). When the accept option is chosen, the Connect method explained previously is run to confirm it. This means the user now appears in the connections list and communication between the two users through the chat system is made possible. The decline option in the requests list and disconnect button in the connections list both use the Disconnect method in the AccountController to remove the connection from the database.

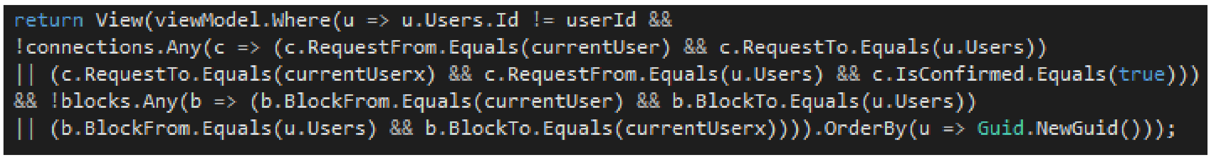
The Disconnect method works in a similar way to the Connect method, but removes the connection from the database instead of adding it. The method checks if there is an existing connection where the requestFrom matches the current user and the requestTo matches the target user, and vice versa. If a matching record is found it is removed from the database. This stops communication from being possible and removes the user from the requests and connections lists. When the block option is clicked, the Block method in the AccountController is called with the corresponding id of the user.

The method uses id and an instance of UserManager to assign the current and target users associated with the block. Unlike a connection, a block is not a mutual decision and is therefore always created as a new entity and stored in the Blocks table. To ensure there are no cases where two people are simultaneously blocked and connected, existing connections are removed. All changes are saved to the database and the page is updated to show the block in the blocks list. If the user wishes to unblock someone, they simply click the unblock button next to the name in the list, which calls the Unblock method in the AccountController. Unblock removes the record from the database and makes the account visible in the search system.

5.6.3 Displaying, Searching, and Filtering Accounts

When the search user's page is visited or the filtering tool is executed, the Index method of the AccountController is called. This method implements an algorithm for deciding what accounts will be displayed in the view. Due to its complexity, the method requires access to the largest number of tables in comparison to the rest of the application. The method uses the Tables, Users, Skills (and SkillLevels through Skills), Connections, and Blocks. It also uses three ViewModels, UserSkillsViewModel for storing and mapping data to the profile cards, two instances of UserConnectionsViewModel for storing and mapping data to the requests and connections lists, and UserBlocksViewModel for storing and mapping data to the blocks list.

When no filters are applied to the page, the algorithm will show all accounts except; the current user, who the current user has sent a connection request to or fully established a connection with, and those that the user has blocked or been blocked by. This ensures no duplicated connect or block relationships can be formed. The logic of this is applied to the UserSkillsViewModel before returning it to the view. *Figure 36* shows the snippet of code determining this logic.

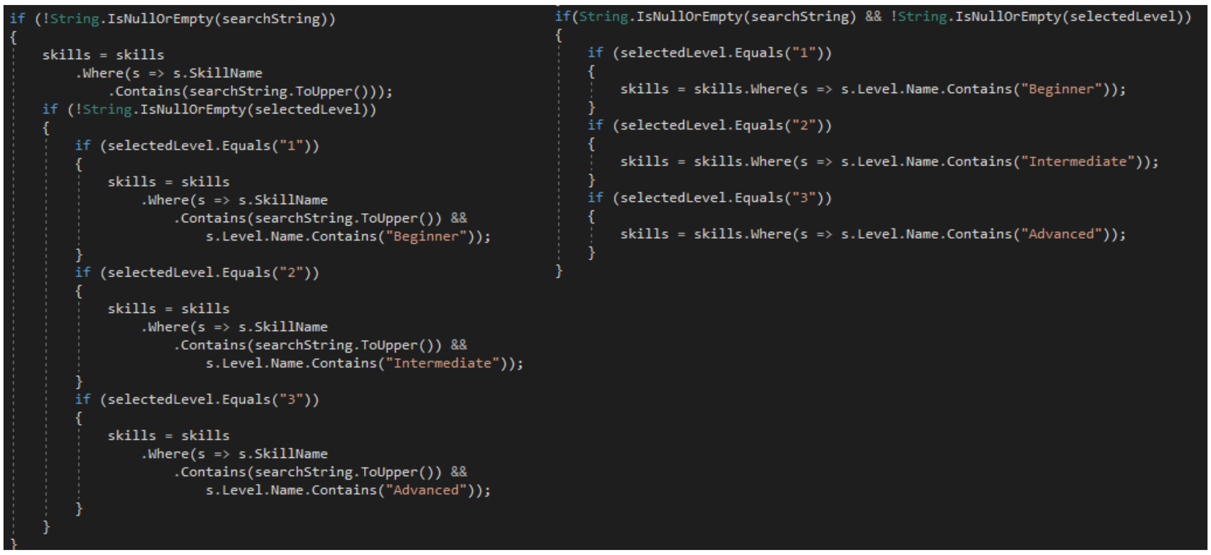


```
return View(viewModel.Where(u => u.Users.Id != userId &&
!connections.Any(c => (c.RequestFrom.Equals(currentUser) && c.RequestTo.Equals(u.Users))
|| (c.RequestTo.Equals(currentUserx) && c.RequestFrom.Equals(u.Users) && c.IsConfirmed.Equals(true)))
&& !blocks.Any(b => (b.BlockFrom.Equals(currentUser) && b.BlockTo.Equals(u.Users))
|| (b.BlockFrom.Equals(u.Users) && b.BlockTo.Equals(currentUserx)))).OrderBy(u => Guid.NewGuid()));
```

Figure 36: Code snippet showing the logic behind displaying accounts.

The logic described in *figure 36* is implemented with a series LINQ and lambda expressions, using conditional and/or Booleans. After filtering the data in the ViewModel, it is then given a random order to ensure different accounts will be shown on each visit.

When a search filter preference is submitted, the Index method is called with the inputted values as its parameters. The conditional statements displayed in *figure 37* are used to get all skill records matching the parameters from the Skills database table.



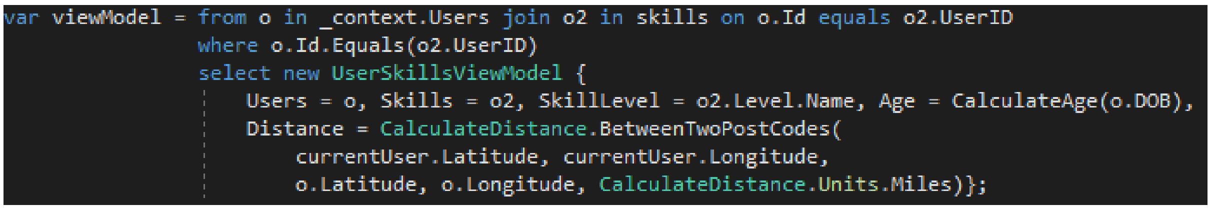
```
if (!String.IsNullOrEmpty(searchString))
{
    skills = skills
        .Where(s => s.SkillName
            .Contains(searchString.ToUpper()));
    if (!String.IsNullOrEmpty(selectedLevel))
    {
        if (selectedLevel.Equals("1"))
        {
            skills = skills
                .Where(s => s.SkillName
                    .Contains(searchString.ToUpper()) &&
                        s.Level.Name.Contains("Beginner"));
        }
        if (selectedLevel.Equals("2"))
        {
            skills = skills
                .Where(s => s.SkillName
                    .Contains(searchString.ToUpper()) &&
                        s.Level.Name.Contains("Intermediate"));
        }
        if (selectedLevel.Equals("3"))
        {
            skills = skills
                .Where(s => s.SkillName
                    .Contains(searchString.ToUpper()) &&
                        s.Level.Name.Contains("Advanced"));
        }
    }
}
```

```
if(String.IsNullOrEmpty(searchString) && !String.IsNullOrEmpty(selectedLevel))
{
    if (selectedLevel.Equals("1"))
    {
        skills = skills.Where(s => s.Level.Name.Contains("Beginner"));
    }
    if (selectedLevel.Equals("2"))
    {
        skills = skills.Where(s => s.Level.Name.Contains("Intermediate"));
    }
    if (selectedLevel.Equals("3"))
    {
        skills = skills.Where(s => s.Level.Name.Contains("Advanced"));
    }
}
```

Figure 37: Code snippet showing the logic behind filtering accounts by skill and skill level.

The left side of *figure 37* checks if an instrument preference has been entered, and gets all the skills that have a name equal to the search string from the database. If a level has been selected, this is filtered further to get the instances of the skill at a chosen skill level. The right side of *figure 37* gets all skills at a chosen skill level.

Using a LINQ query, the resulting skill model is joined with the users table to create an instance of UserSkillsViewModel holding the account data. *Figure 38* shows the full query.

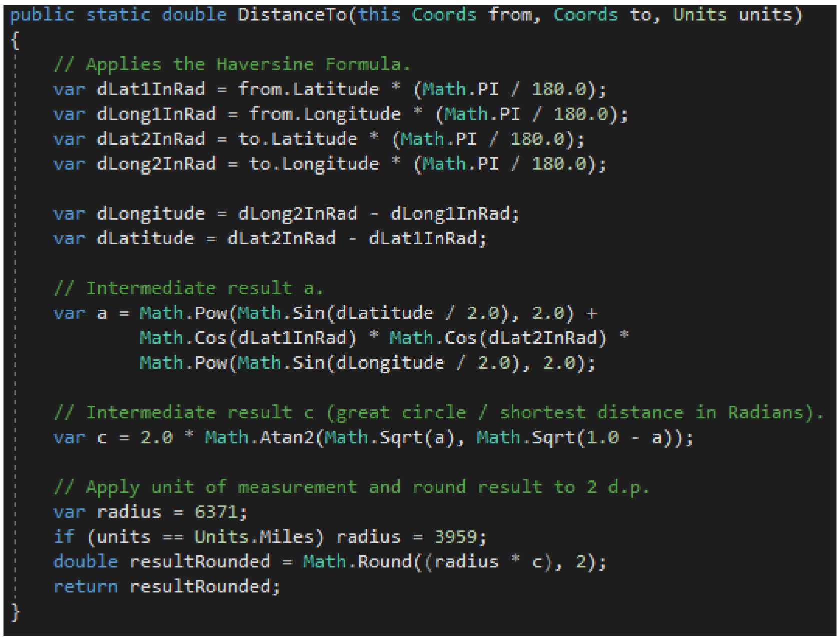


```
var viewModel = from o in _context.Users join o2 in skills on o.Id equals o2.UserID
                where o.Id.Equals(o2.UserID)
                select new UserSkillsViewModel {
                    Users = o, Skills = o2, SkillLevel = o2.Level.Name, Age = CalculateAge(o.DOB),
                    Distance = CalculateDistance.BetweenTwoPostCodes(
                        currentUser.Latitude, currentUser.Longitude,
                        o.Latitude, o.Longitude, CalculateDistance.Units.Miles)};
```

Figure 38: Code snippet showing how the UserSkillsViewModel is used to store account details.

The ViewModel now holds all the details necessary for displaying the accounts. When no filters are selected the full skill records are used instead of model in *figure 38*. The age is calculated using the method CalculateAge which takes the DOB of the user as input, compares it to the current date, and returns the age in years. The distance is calculated using the methods defined in the static class CalculateDistance.

CalculateDistance's method BetweenTwoPostCodes is individually called with the latitude and longitude coordinates of the current user, each user in the UserSkillsViewModel, and the units (miles or kilometres) that the calculation should be in. BetweenTwoPostCodes structures the input as coordinates and returns the result of another method called DistanceTo. *Figure 39* shows the code for the DistanceTo method.



```
public static double DistanceTo(this Coords from, Coords to, Units units)
{
    // Applies the Haversine Formula.
    var dLat1InRad = from.Latitude * (Math.PI / 180.0);
    var dLong1InRad = from.Longitude * (Math.PI / 180.0);
    var dLat2InRad = to.Latitude * (Math.PI / 180.0);
    var dLong2InRad = to.Longitude * (Math.PI / 180.0);

    var dLongitude = dLong2InRad - dLong1InRad;
    var dLatitude = dLat2InRad - dLat1InRad;

    // Intermediate result a.
    var a = Math.Pow(Math.Sin(dLatitude / 2.0), 2.0) +
            Math.Cos(dLat1InRad) * Math.Cos(dLat2InRad) *
            Math.Pow(Math.Sin(dLongitude / 2.0), 2.0);

    // Intermediate result c (great circle / shortest distance in Radians).
    var c = 2.0 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a));

    // Apply unit of measurement and round result to 2 d.p.
    var radius = 6371;
    if (units == Units.Miles) radius = 3959;
    double resultRounded = Math.Round((radius * c), 2);
    return resultRounded;
}
```

Figure 39: Code snippet showing the DistanceTo method in the CalculateDistance class.

Distance from the coordinate values is calculated using the Haversine Formula (*figure 39*). The Haversine Formula is a popular and frequently used formula for calculating geographical distance. It uses the latitude and longitude coordinates of two different points to compute the great-circle or shortest distance between them on a spherical surface [89].

The highest calculated distance is mapped to the maximum value of the slider (*figure 31*) allowing the user to display all accounts within a chosen radius. The slider is added using the Syncfusion [90] JavaScript UI controls package. The currently selected value is displayed with a tooltip and used to filter the accounts held in the ViewModel when the search button (*figure 31*) is clicked.

5.7 Chat System

To implement the real-time chat system, the tutorial "Build a group chat app using .net core" [91] was followed and purposed for use in this application. Extra features were also added outside of the scope of the tutorial. The chat system uses server-side scripting through four controllers, ChatController, GroupController, MessageController, and AuthController. Each providing an element of the systems functionality.

The chat system also uses a large amount of client-side scripting through jQuery and Ajax requests. To add real-time functionality to the system, Pusher sits as a real-time layer between the server and client, maintaining persistent connections to the clients. This section explains how each element of the chat system works in regards to the front/backend and clarifies the role of Pusher.

5.7.1 Chat Interface

When the chat system is accessed by the user, the interface is displayed using ChatController to display the corresponding Index view. The controller makes requests to the database to get all the groups the current user belongs to and provides a list of the current connections for use in creating new chats. The controller uses the UserGroupViewModel to combine the records from the UserGroups and MessageGroups tables. This enables access to the users associated with each message group. The chat interface can be seen in *figure 40* below.

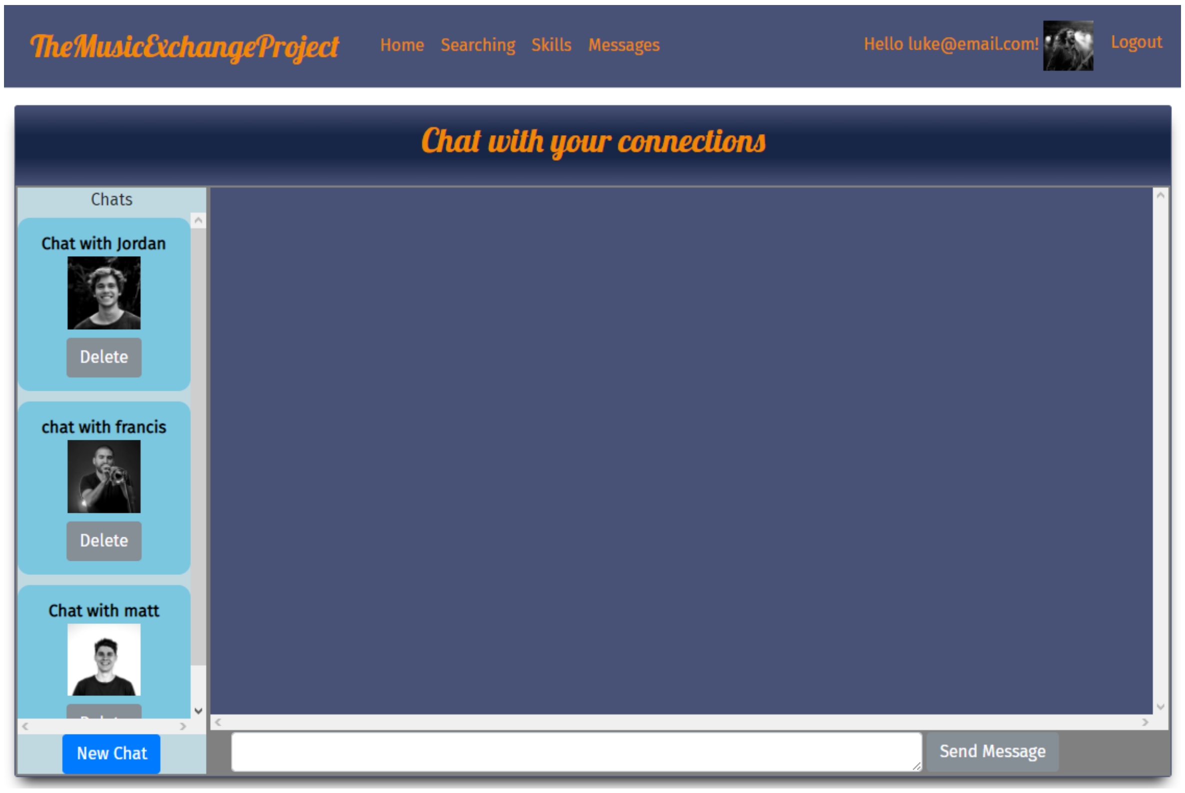


Figure 40: The chat interface on desktop.

As show *figure 40*, the left sidebar is used to display the list of chats the user is currently engaged in and provides the option to delete the chat or create a new one. To create a new chat the "New Chat" button is clicked and a hidden modal form which is contained in the view is displayed to the user. *Figure 41* shows the design of the form.

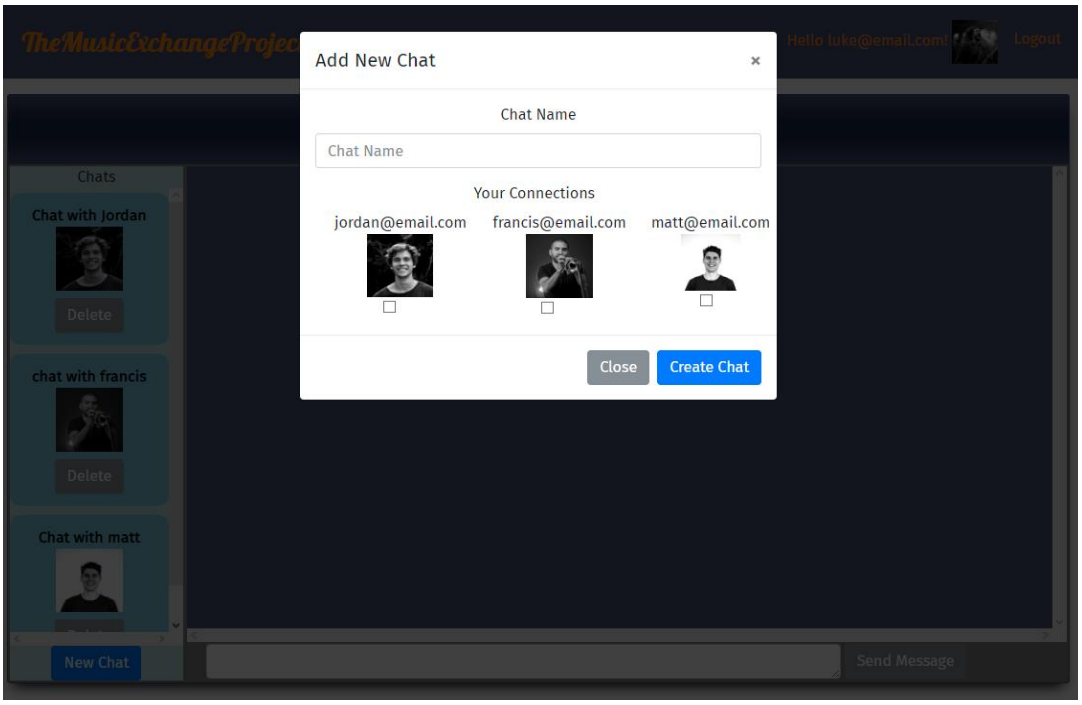
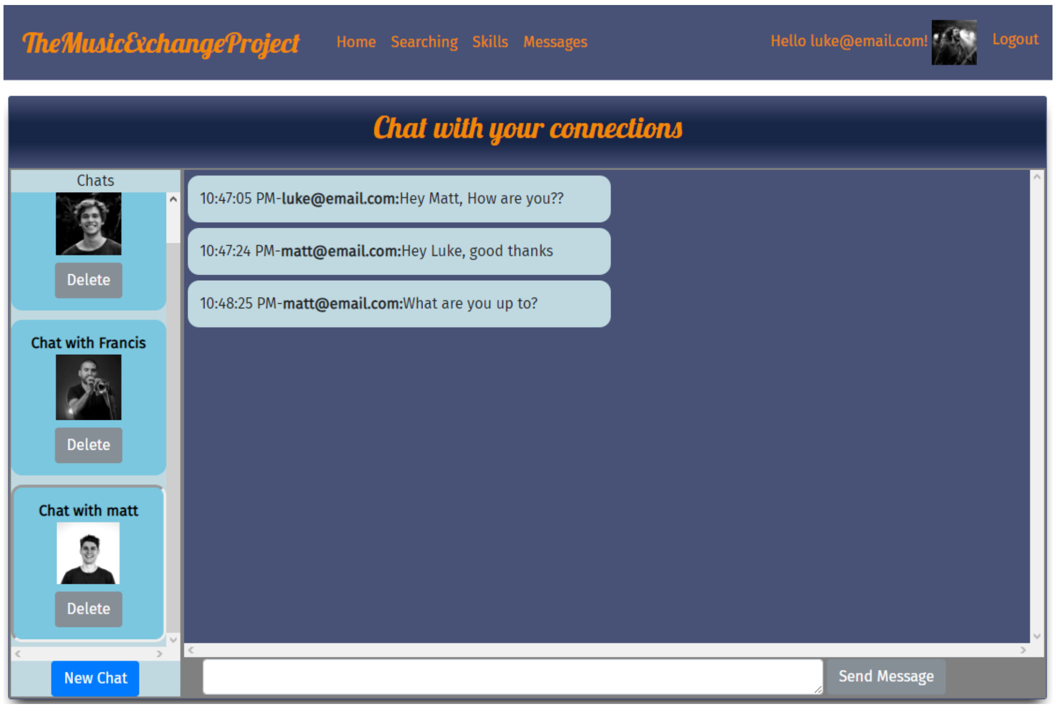


Figure 41: The modal form for creating a new chat.

Using the form (*figure 41*) the new chat is given a name and the person to be added is selected from the list of connections. If the close button is clicked, the form is toggled to hidden as it was previously. If the create chat button is clicked (after satisfying the form) a jQuery function executes an Ajax POST request to the GroupController's Create method with the group name and user as the parameters. This method creates a new message group in the MessageGroups table and adds records of the users and the group to the UserGroups table. If the method is successful it returns a success JSON indicating that the form can be hidden.

A chat is deleted when the corresponding delete button (*figure 40*) is clicked. The button calls the Delete method in the ChatController with the id of the group as a parameter and removes the group and all its messages from the database. If the method returns successfully the jQuery function deleteGroup is called to remove the HTML object from the interface. All evidence of the chat has now been removed from the front and back end of the application.

When a chat is clicked the associated messages are displayed in the main body of the interface (*figure 42*). Once a chat is selected, it is given a CSS effect to appear visibly selected and the input box at the bottom of the interface can be used to send a new message to the other user in the chat.



10:47:05 PM-**luke@email.com:**Hey Matt, How are you??

10:47:24 PM-**matt@email.com:**Hey Luke, good thanks

10:48:25 PM-**matt@email.com:**What are you up to?

Figure 42: The chat interface on desktop with previous messages displayed.

To display the messages for a particular group, client-side script executes an Ajax GET request to the MessageController's GetById method with the group id as the parameter. This returns all the messages with the matching id from the Messages table of the database. Once this has been received, the function uses a foreach loop to iterate through each message and structure the attributes as html div objects, containing the time sent, the username of the person and the message text. Once all the associated messages have been marked as divs, they are appended to the body of the chat view displaying them to the user.

To add a new message, the user adds the desired text to the input box located at the bottom of the interface and hits the "Send Message" button. When this button is clicked it calls a jQuery function which executes an Ajax POST request to the MessageController's Create method with the message content, UserName and GroupId as the parameters. The function then adds the message to the Messages table in the database and returns a success JSON if it has been successful. Once success has been confirmed, the function structures the message as a div in the same format as the previous paragraph and appends it to the body of the chat view to be displayed alongside the other messages.

5.7.2 Real-time functionality

Currently, the previously explained implementations would provide a working system, however the other user would not be able to see updates to the messages or groups in real-time. To add real-time functionality the Pusher API is used to implement public and private channels for communication to occur, listen for messages and trigger events when groups and messages are created or deleted.

In this application, private channels are used to restrict the channels an unauthenticated user can subscribe to ensuring that they only have access to the channels corresponding to the chats they are in. When a user wants to subscribe to a private channel a request is made to Pusher to check if the user has the necessary authentication. To do this the middleware route pusher/auth is added to the Startup file which is used by Pusher to make a request to an API endpoint AuthController. Pusher calls the ChannelAuth method of the controller with a channel name and the socket id of the user it wants to authenticate. The method queries the UserGroups table of the database to check if the user is in the group they are trying to gain authentication for. If they belong to the corresponding group access to the private channel is authenticated.

To listen for new chats the Pusher javascript library is used to subscribe to a public channel representing the chat system. When a new chat is created an event is triggered to display the newly created chat to the users subscribed to the channel. This is implemented in the Create method of the GroupController and functions by creating and returning the new chat. Once the chat is returned the jQuery function reloadGroup is called to change the display in the left side bar.

To listen for chat deletion a similar method to create is used for the same public channel. When a chat is deleted an event is triggered after it has been successfully removed from the database by the Delete method of the ChatController. The jQuery function reloadGroup is called to change the display in the left side bar for all subscribers of the chat channel.

To listen for new messages the system checks if the user is subscribed to the private channel linked to currently selected group. When a new message is added an event is triggered to display the newly added message to the users subscribed to the channel. This is implemented in the Create method of the MessageController and functions by creating and returning the new message. Once a new message is returned, the channel is bound to an event and the message is appended to the body of the chat for all subscribers of the channel.

5.8 Testing

Throughout the implementation of this project testing was performed as each new feature or piece of functionality was added. A great deal of care was taken to ensure each feature worked as expected. As the functionality of the application is almost entirely based on how the client interacts with other accounts in the system it was important to populate the database before the system could be tested. Initially, the class DbInitializer was implemented to seed the database with 20 users, 40 skills, and 3 skill levels. This class is run when the application is launched and populates the corresponding tables if they are empty.

Each of the 20 users has been given their own identity resembling a real person in the system. Each account is created with a different name, email, bio, DOB, postcode, and profile picture. Using the postcode lists on doogal.co.uk [92], the postcodes were randomly selected from different SA districts to be used in testing the distances between users. Profile pictures for the users were selected from the stock photo website pexels.com [93].

Once the accounts were created the skill levels “Beginner”, “Intermediate”, and “Advanced” were seeded as each skill has to be attributed to a skill level. 5 instances of the following 8 skill types were seeded, “Guitar”, “Saxophone”, “Flute”, “Violin”, “Piano”, “Trombone”, “Drums”, “Ukulele”, each linking to a skill level and user. The application is now reasonably populated and can be used for testing the functionality of the different areas.

Testing began by systematically attempting to visit every possible route in the application as an unauthenticated user and checking if the application automatically redirects to the login page. Attempting to reach every endpoint and discovering that all redirections occurred as expected has meant none of the areas possess obvious security flaws. Following this, each route was visited as an authenticated user to check if the navigation between pages was correctly established for users that are signed in. Successfully travelling between the pages proved the areas of the application had been correctly integrated.

After the routing tests the next testing method was attempting to use all the features and functionality provided by each the pages. It was important to check that the logic of the algorithms was correct and the methods produced the desired output. To test the search preferences different combinations of filters were used and the resulting profiles were observed. This led to the discovery of oversights during development such as profiles being displayed in the same order and certain filter combinations not working. To test the relationship system the developer simultaneously signed in as two accounts and used each function, observing the changes to the page and the resulting data stored in the database.

To test the skill system each of the CRUD functions was performed and the resulting output on screen and in the database was observed. To test the chat interface the developer simultaneously signed in as two accounts and performed a mock conversation in real time, as well as using and observing the behaviour of each function. This led to the discovery of various issues such as the background of the create creation modal form not disappearing as expected, the delete function not happening in real-time, and various styling issues.

Several of the technical issues that were encountered and corresponding solutions will be discussed in the following section.

5.9 Technical Issues

Technical issues were encountered throughout the development of this project and solutions were found for the vast majority. Some sort of issue has occurred in every area of the application for a range of different reasons. This section discusses some of the issues and their solutions below, before covering some existing issues which have not been solved at the time of writing this dissertation.

In the early stages of its implementation many issues occurred within the searching system. The first approach to displaying the lists for the user’s requests, connections and blocks was to store a list in the UserSkillsViewModel bound to the view. This led to inefficiencies in accessing the items that were linked to the current user as the full lists for each user were being stored. To overcome this issue, lists were created specifically for the current user when the page is visited and sent straight from the controller to the view using the dynamic property ViewBag.

Another issue was encountered when trying to convert a postcode to latitude and longitude coordinates. Initially, the approach was to get the latitude and longitude coordinates from the

Postcodes.io API when the CalculateDistance class is called. By doing this the application was exceeding the API transfer limit and it was therefore throwing an error whenever the application attempted to calculate the distance. To overcome this and reduce API requests, it was decided that the latitude and longitude coordinates would be calculated and stored on registration and account modification.

The issue of the background of the modal form for chat creation not disappearing was a small but annoying bug that would ruin the immersion of real-time chat functionality by requiring a full page refresh to fix. The bug would leave a grey tint on the screen that could not be closed and would overlay and stop any page interactions from being possible. The developer attempted to fix this issue by applying styling to the CSS classes to remove the background. This solution did not work so an alternative solution was implemented using jQuery to remove the references to the CSS classes after account creation.

The styling and frontend design of the application was personally found to be the most challenging aspect of the implementation. At the time of writing this dissertation most issues with styling have been solved and only minor issues still exist. Using a layout view has been beneficial in enforcing an application wide layout and implementing the navigation bar, however it has also caused styling issues for some of the pages.

In the layout view, each page is rendered as the content within the main body of the page. However, when the searching page is rendered only the form for the filtering tool is actually considered to be inside this container, with the profile cards and lists not inheriting the same styling. This may not appear as though it is a large issue, but has meant the background of the page is difficult to style as the dimensions of the objects are not considered to be that of the container. This has caused a multitude of frustrating problems such as struggling to add pagination to the cards, meaning they have all been displayed on a single page.

Describing every issue encountered and its solution would be a dissertation length document in itself and recalling them all would be an impossible task. Most issues were simply initial mistakes due to human error, oversights of user interaction, or gaps in knowledge. The produced application contains little remaining technical issues, with the only existing ones relating to the styling rather than the functionality.

6: Evaluation

This section of the document will begin by reflecting on what has been achieved during the course of this project, from a personal, educational and technical perspective. Following this, it will identify the challenges that were encountered and how they have been mitigated. Finally, the future direction for this project will be discussed.

6.1 Achievements

Overall, this project has been a steep learning curve which has pushed the limits the developer's knowledge and understanding of web application development. Although, the journey has not been without flaws, the outcome of this project has been a success and the work has gone beyond what was initially scoped. All the basic requirements were fulfilled and many of the extra requirements were implemented. The objectives for this project have been met, TheMusicExchangeProject is a web application that can search for musicians using a range of customisable preferences, building relationships between users, and facilitating social interactions.

A multitude of powerful, unique features have been implemented, with each page fulfilling its intended purpose and providing a sophisticated level of functionality. The database is well designed, efficiently storing and managing the data behind each area of the application. The server-side script is extensive and uses 8 different controllers to liaise between the 8 database tables and the 10 main views. The client-side is visually appealing and uses jQuery, Pusher, Ajax calls, Bootstrap, CSS and HTML, to create a real-time messaging service and give the application its unique, crafted, resizable interface.

Although progress fell behind schedule in the early stages of the development process, through hard work and dedication, TheMusicExchangeProject managed to meet the majority of its deadlines once the technology choices were changed and a second plan was constructed. The evaluation and selection of the software development methodologies played a big role in this project and aided in its success. Most of the issues discovered during the implementation and testing of the product were solved without much difficulty, with only minor styling changes remaining.

This project has improved the developer's knowledge of web application development and provided the opportunity to build a better understanding and confidence using the selected software languages and frameworks.

6.2 Risks and Mitigation

In the beginning of this project, a clear analysis of potential risks and mitigation strategies was carried out for the developer to refer to if a risk became apparent. The risk of the initial technology choices exceeding the developer's skill level was identified as having a reasonable chance of occurring and started becoming a possibility during the early development stages of this project. The possibility of the risk occurring was considered when progress began to fall behind schedule due to a lack of experience with the chosen technology.

A mitigation strategy was planned when the risk first became apparent and was put into place from the start of development. The developer attempted to mitigate the risk before implementation began by setting aside large amounts of time to learn and practice with the initial technology choices. When the projects development began to progress passed the basic concepts, the difficulty increased which lead to progress occupying more time and falling dangerously behind schedule. After consulting the timeline and realising the possibility of incompletion was becoming apparent it was time to put the second part of the mitigation strategy into action.

The second part of the mitigation strategy involved making the drastic decision to change the technology choices to those the developer had more experience using. This was a difficult decision but clearly the correct one as the project was completed to a high standard within the timeframe. Identifying and mitigating this risk early on in the development process was crucial to the project's success. If it was ignored or the technology choices were changed at a later time the project could have remained incomplete or been delivered at a lower quality.

6.3 Future Development

Although developing TheMusicExchangeProject was a challenging experience it was also highly rewarding. Given more time there are many features that could have been added and existing areas where functionality could have been improved. Once the academic side of the project has concluded, it would definitely be of interest to continue development as a personal project without the presence of time constraints.

It is unlikely that the application will ever be released as a professional product, but the experience and knowledge the process has provided will stay with the developer for future projects. The project has expanded personal knowledge of web application development tenfold and the learning experience has fuelled the developer's passion for web application development.

Acknowledgements

The completion of this project would not have been possible without the advice given and time committed by Dr. Liam O'Reilly. Without Liam's help many of the functions would have taken a lot more time and difficulty to implement. Liam's words of encouragement and frequent meetings were exactly what was needed to ensure progress stayed on track throughout the process.

Many friends and family members deserve to be thanked for their help and emotional support through the successes and struggles faced in this project. A massive thanks to Georgia for spending a large amount of her own time proof reading sections and offering advice throughout the process. Without the constant assurance and reminders to take breaks this project would have caused much more stress and anxiety.

A final thanks go to Swansea University and the excellent lecturers and staff members. The experience of doing a Computer Science degree has provided the skills and knowledge that will stay with the developer for life.

References

[1] MacDonald, R., Kreutz, G. and Mitchell, L. eds., 2013. *Music, health, and wellbeing*. Oxford University Press. chp2. [Accessed 16 April 2019]
[2] ABRSM. 2014. Making Music Report: The Statistics. [Online] Available at: https://gb.abrsm.org/en/making-music/4-the-statistics/. [Accessed 16 April 2019].
[3] Bidvine. 2019. Cost to hire a Guitar Teacher on Bidvine. [Online] Available at: https://www.bidvine.com/guitar-lessons/price-guide. [Accessed 16 April 2019].
[4] Avgeriou, P., Zdun, U. 2005, Architectural patterns revisited a pattern language. In: 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, pp. 1–39. [Accessed 16 April 2019].
[5] Herberto Graca. 2017. Architectural Styles vs. Architectural Patterns vs. Design Patterns. [Online] Available at: https://herbertograca.com/2017/07/28/architectural-styles-vs-architectural-patterns-vs-design-patterns/. [Accessed 16 April 2019].
[6] Code Academy. 2019. MVC: Model, View, Controller. [Online] Available at: https://www.codecademy.com/articles/mvc. [Accessed 16 April 2019].
[7] Code Academy. 2019. What is REST?. [Online] Available at: https://www.codecademy.com/articles/what-is-rest. [Accessed 16 April 2019].
[8] Sam Deering. 2012. Do you know what a REST API is?. [Online] Available at: https://www.sitepoint.com/developers-rest-api/. [Accessed 16 April 2019].
[9] Fielding, R., 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD. Irvine: University of California. Chp 5. Available from: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed 16 April 2019]
[10] Stackify. 2017. What are CRUD Operations?. [Online] Available at: https://stackify.com/what-are-crud-operations/. [Accessed 16 April 2019].
[11] Martin, J. 1983. Managing the Data Base Environment. A James Martin book, p 381. [Accessed 16 April 2019].
[12] Stephen Watts. 2018. REST vs CRUD: Exploring The Differences?. [Online] Available at: https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/. [Accessed 16 April 2019].
[13] Jingjing Li and Chunlin Peng. 2012. "jQuery-based Ajax general interactive architecture," *2012 IEEE International Conference on Computer Science and Automation Engineering*, Beijing. pp. 304-306. [Accessed 17 April 2019].
[14] Flint. 2019. Flint Website Discover page. [Online] Available at: https://www.flint.cool/Discover. [Accessed 17 April 2019].
[15] Hendrix Media LLC. 2019. Hendrix Website Profile page. [Online] Available at: https://www.gohendrix.com/profiles. [Accessed 18 April 2019].
[16] Jamseek. 2019. Jamseek Home Page. [Online] Available at: https://www.jamseekapp.co.uk/en/index. [Accessed 18 April 2019].
[17] Jamseek. 2019. Jamseek About Page. [Online] Available at: https://www.jamseekapp.co.uk/en/index#about. [Accessed 18 April 2019].
[18] Jamseek Twitter Account. 2019. @JamseekLondon. [Online] Available at: https://twitter.com/JamseekLondon. [Accessed 18 April 2019].
[19] Valade, J. 2004. PHP 5 for Dummies, John Wiley & Sons, Incorporated, Hoboken. Available from: ProQuest Ebook Central. [Accessed 19 April 2019].
[20] Martinez, J. 2012. C# 5 First Look, Packt Publishing Ltd, Olton. Available from: ProQuest Ebook Central. [Accessed 19 April 2019].
[21] W3Schools. 2019. “Introduction to SQL”. [Online] Available at: https://www.w3schools.com/sql/sql_intro.asp. [Accessed 19 April 2019].

[22] Microsoft, 2017, "Language Integrated Query (LINQ). [Online] Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/. [Accessed 19 April 2019].
[23] LINQPad, 2019, Why LINQ beats SQL. [Online] Available at: https://www.linqpad.net/WhyLINQBeatsSQL.aspx. [Accessed 19 April 2019].
[24] Microsoft, 2017, LINQ to SQL. [Online] Available at: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/. [Accessed 19 April 2019].
[25] Kurniawan, B 2015, HTML : A Beginner's Tutorial, Brainy Software, Vancouver. Available from: ProQuest Ebook Central. [Accessed 19 April 2019].
[26] W3Schools, 2019, "CSS Introduction". [Online] Available at: https://www.w3schools.com/css/css_intro.asp. [Accessed 19 April 2019].
[27] MDN web docs, 2019, "CSS3". [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3. [Accessed 19 April 2019].
[28] MDN web docs, 2019, "What is JavaScript?". [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed 19 April 2019].
[29] TechTerms, 2013, Framework Definition. [Online] Available at: https://techterms.com/definition/framework. [Accessed 21 April 2019].
[30] Webopedia, 2019, Programming Language. [Online] Available at: https://www.webopedia.com/TERM/P/programming_language.html. [Accessed 21 April 2019].
[31] Carey Wodehouse. 2019. What Is a Framework? [Online] Available at: https://www.upwork.com/hiring/development/understanding-software-frameworks/. [Accessed 21 April 2019].
[32] Tutorialspoint. 2019. Laravel - Overview. [Online] Available at: https://www.tutorialspoint.com/laravel/laravel_overview.htm. [Accessed 21 April 2019].
[33] Saunier, R 2014, Getting Started with Laravel 4, Packt Publishing Ltd, Birmingham. Available from: ProQuest Ebook Central. [21 April 2019].
[34] Steve Smith. 2018. Overview of ASP.NET Core MVC. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2. [Accessed 21 April 2019].
[35] Freeman, A. 2016, *Pro ASP.NET Core MVC*. Berkeley, CA: Apress (ITpro Collection). Available at: http://search.ebscohost.com/login.aspx?direct=true&AuthType=cookie,ip,shib,uid&db=nlebk&AN=1174486&site=ehost-live&scope=site [Accessed: 21 April 2019].
[36] Bootstrap. 2019. Bootstrap Home Page. [Online] Available at: https://getbootstrap.com. [Accessed 21 April 2019].
[37] W3Schools. 2019. Bootstrap 4 Get Started. [Online] Available at: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp. [Accessed 21 April 2019].
[38] Vue.js. 2019. What is Vue.js?. [Online] Available at: https://vuejs.org/v2/guide/. [Accessed 21 April 2019].
[39] Tutorials Point. 2019. VueJS - Overview. [Online] Available at: https://www.tutorialspoint.com/vuejs/vuejs_overview.htm. [Accessed 21 April 2019].
[40] jQuery. 2019. What is jQuery?. [Online] Available at: https://jquery.com. [Accessed 21 April 2019].
[41] W3Schools. 2019. jQuery Introduction. [Online] Available at: https://www.w3schools.com/jquery/jquery_intro.asp. [Accessed 21 April 2019].

[42] Pusher Community. 2019. What is Pusher?. [Online] Available at: https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html. [Accessed 21 April 2019].
[43] Pusher. 2019. Channels Overview. [Online] Available at: https://pusher.com/docs/channels. [Accessed 21 April 2019].
[44] Postcodes.io. 2019. Postcodes.io Home Page. [Online] Available at: https://postcodes.io. [Accessed 22 April 2019].
[45] Google Fonts. 2019. About Google Fonts. [Online] Available at: https://fonts.google.com/about. [Accessed 22 April 2019].
[46] Microsoft. 2019. Welcome to the Visual Studio IDE. [Online] Available at: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2017. [Accessed 22 April 2019].
[47] Microsoft. 2019. Visual Studio IDE documentation. [Online] Available at: https://docs.microsoft.com/en-us/visualstudio/ide/?view=vs-2017. [Accessed 22 April 2019].
[48] NuGet. 2019. NuGet Gallery. [Online] Available at: https://www.nuget.org. [Accessed 22 April 2019].
[49] Microsoft. 2018. An introduction to NuGet. [Online] Available at: https://docs.microsoft.com/en-gb/nuget/what-is-nuget. [Accessed 22 April 2019].
[50] Microsoft. 2019. Introduction to Identity on ASP.NET Core. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.2&tabs=visual-studio. [Accessed 22 April 2019].
[51] Microsoft. 2018. Advanced Features of Visual Studio. [ONLINE] Available at: https://docs.microsoft.com/en-us/visualstudio/ide/advanced-feature-overview?view=vs-2017. [Accessed 22 April 2019].
[52] Atlassian. 2018. What is Bitbucket?. [Online] Available at: https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket. [Accessed 22 April 2019].
[53] Git. 2019. 1.1 Getting Started - About Version Control. [Online] Available at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. [Accessed 22 April 2019].
[54] MySQL. 2019. 1.3.1 What is MySQL?. [Online] Available at: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. [Accessed 22 April 2019].
[55] Microsoft. 2018. Advanced feature Overview: Connect to databases. [Online] Available at: https://docs.microsoft.com/en-us/visualstudio/ide/advanced-feature-overview?view=vs-2017#connect-to-databases. [Accessed 22 April 2019].
[56] Gantt.com. 2019. What is a Gantt Chart?. [Online] Available at: https://www.gantt.com. [Accessed 23 April 2019].
[57] GanttPRO. 2018. GanttPRO Chart Creator. [Online] Available at: https://app.ganttpro.com. [Accessed 1 October 2018].
[58] TeamGantt. 2019. Online Gantt Chart Software. [Online] Available at: https://www.teamgantt.com. [Accessed 21 January 2019].
[59] Cambridge Dictionary. 2019. Meaning of methodology in English. [Online] Available at: https://dictionary.cambridge.org/dictionary/english/methodology. [Accessed 23 April 2019].
[60] Gerald D. Everett; Raymond McLeod, 2007, "The Software Development Life Cycle," in *Software Testing: Testing Across the Entire Software Development Life Cycle*, IEEE, pp.29-58. [Accessed 23 April 2019].
[61] Bassil, Y., 2012. A simulation model for the waterfall software development life cycle. *arXiv preprint arXiv:1205.6904*. [Accessed 23 April 2019].

[62] Existek. 2017. SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral. [Online] Available at: https://existek.com/blog/sdlc-models/. [Accessed 23 April 2019].
[63] Leau, Y.B., Loo, W.K., Tham, W.Y. and Tan, S.F., 2012. Software development life cycle AGILE vs traditional approaches. In *International Conference on Information and Network Technology* (Vol. 37, No. 1, pp. 162-167). [Accessed 23 April 2019].
[64] Balaji, S. and Murugaiyan, M.S., 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, *2*(1), pp.26-30. [Accessed 23 April 2019].
[65] Alshamrani, A. and Bahattab, A., 2015. A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. *International Journal of Computer Science Issues (IJCSI)*, *12*(1), p.106. [Accessed 23 April 2019].
[66] Visual Paradigm. 2019. What is Entity Relationship Diagram (ERD)? [Online] Available at: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/. [Accessed 24 April 2019].
[67] Visual Paradigm. 2019. Entity Relationship Diagram Online Tool. [Online] Available at: https://online.visual-paradigm.com. [Accessed 24 April 2019].
[68] NAz, K.A.Y.A. and Epps, H., 2004. Relationship between color and emotion: A study of college students. *College Student J*, *38*(3), p.396. [Accessed 24 April 2019].
[69] Jerry Cao, Creative Bloq. 2018. 12 colours and the emotions they evoke. [Online] Available at: https://www.creativebloq.com/web-design/12-colours-and-emotions-they-evoke-61515112. [Accessed 24 April 2019].
[70] Jennifer Bourn. 2011. Color Meaning: Meaning of the Color Purple. [Online] Available at: https://www.bourncreative.com/meaning-of-the-color-purple/. [Accessed 24 April 2019].
[71] Coolors. 2019. Colour Scheme Generator. [Online] Available at: https://coolors.co/f18805-d95d39-c0d9e1-485277-291c1b. [Accessed 24 April 2019].
[72] Colour Blind Awareness. 2019. Types of Colour Blindness. [Online] Available at: http://www.colourblindawareness.org/colour-blindness/types-of-colour-blindness/. [Accessed 24 April 2019].
[73] Jakob Nielsen. 1994. 10 Usability Heuristics for User Interface Design. [Online] Available at: https://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed 24 April 2019].
[74] Learn Entity Framework Core. 2018. What is Entity Framework Core?. [Online] Available at: https://www.learnentityframeworkcore.com/#what-is-entity-framework-core. [Accessed 26 April 2019].
[75] Microsoft. 2019. IdentityDbContext Class. [Online] Available at: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identitydbcontext?view=aspnetcore-2.2. [Accessed 26 April 2019].
[76] Learn Entity Framework Core. 2018. DbContext. [Online] Available at: https://www.learnentityframeworkcore.com/dbcontext. [Accessed 26 April 2019].
[77] Learn Entity Framework Core. 2018. DbSet. [Online] Available at: https://www.learnentityframeworkcore.com/dbset. [Accessed 26 April 2019].
[78] Microsoft. 2016. Relationships. [Online] Available at: https://docs.microsoft.com/en-us/ef/core/modeling/relationships. [Accessed 26 April 2019].
[79] Learn Entity Framework Core. 2018. Migrations. [Online] Available at: https://www.learnentityframeworkcore.com/migrations. [Accessed 26 April 2019].
[80] Microsoft. 2019. Identity model customization in ASP.NET Core. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-2.2. [Accessed 26 April 2019].

[81] Rick Anderson, Microsoft. 2019. Introduction to Identity on ASP.NET Core. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.2&tabs=visual-studio. [Accessed 26 April 2019].
[82] Rick Anderson, Microsoft. 2019. Tag Helpers. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.2. [Accessed 26 April 2019].
[83] ColorSpace. 2019. Color Gradient Generator. [Online] Available at: https://mycolor.space. [Accessed 26 April 2019].
[84] Microsoft. 2019. SignInManager Class. [Online] Available at: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.signinmanager-1?view=aspnetcore-2.2. [Accessed 26 April 2019].
[85] Per Christensson. 2007. What are static and dynamic Web pages?. [Online] Available at: https://pc.net/helpcenter/answers/static_and_dynamic_web_pages. [Accessed 27 April 2019].
[86] TutorialsTeacher. 2019. Layout View. [Online] Available at: https://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc. [Accessed 27 April 2019].
[87] TutorialsTeacher. 2019. Partial View. [Online] Available at: https://www.tutorialsteacher.com/mvc/partial-view-in-asp.net-mvc. [Accessed 27 April 2019].
[88] Shailendra Chauhan. 2014. Understanding ViewModel in ASP.NET MVC. [Online] Available at: https://www.dotnettricks.com/learn/mvc/understanding-viewmodel-in-aspnet-mvc. [Accessed 30 April 2019].
[89] GIS Map. 2019. Haversine Formula – Calculate geographic distance on earth. [Online] Available at: https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/. [Accessed 30 April 2019].
[90] Syncfusion. 2019. Syncfusion JavaScript UI Controls demos. [Online] Available at: https://ej2.syncfusion.com/home/. [Accessed 30 April 2019].
[91] Pusher. 2018. Build a group chat app using .net core. [Online] Available at: https://pusher.com/tutorials/group-chat-net#making-our-messaging-realtime. [Accessed 5 May 2019].
[92] Doogal.co.uk. 2019. SA1 Postcodes. [Online] Available at: https://www.doogal.co.uk/UKPostcodes.php?Search=SA1. [Accessed 6 May 2019].
[93] Pexels.com. 2019. Free Stock Photos. [Online] Available at: https://www.pexels.com. [Accessed 6 May 2019].