Programming a music loaning and catalogue website

Adam Bilkus

Candidate Number: 3085

Centre Number: 12290

Table of Contents

[1-Analysis 6](#_Toc33437883)

[1.1-Introduction 6](#_Toc33437884)

[1.1.1-Background 6](#_Toc33437885)

[1.1.2-Project Scope 6](#_Toc33437886)

[1.1.3- The Client/Supervisor 8](#_Toc33437887)

[1.1.4- The Prospective Users/Audience 8](#_Toc33437888)

[1.2-Investigation 9](#_Toc33437889)

[Existing Websites 9](#_Toc33437890)

[Questionnaire 10](#_Toc33437891)

[Prototyping 14](#_Toc33437892)

[Neural Network/Suggestions Algorithm 17](#_Toc33437893)

[1.3-Constraints/Hardware 21](#_Toc33437894)

[Hardware Constraints 21](#_Toc33437895)

[Software Constraints 21](#_Toc33437896)

[User’s Knowledge of Information Technology 22](#_Toc33437897)

[Access rights 22](#_Toc33437898)

[1.4 Limitations 22](#_Toc33437899)

[1.5-Objectives 23](#_Toc33437900)

[General Objectives 23](#_Toc33437901)

[Specific Objectives 23](#_Toc33437902)

[Optional Objectives (for if I have time) 24](#_Toc33437903)

[1.6-Proposed Solution 24](#_Toc33437904)

[Privacy Policy 26](#_Toc33437905)

[2-Design 33](#_Toc33437906)

[2.1-High Level Overview 33](#_Toc33437907)

[Main Menu 34](#_Toc33437908)

[Login 35](#_Toc33437909)

[Staff Main Menu Actions 36](#_Toc33437910)

[Non-Staff Main Menu Actions 37](#_Toc33437911)

[Logout 38](#_Toc33437912)

[All Music 38](#_Toc33437913)

[All Composers 39](#_Toc33437914)

[All Borrowed 41](#_Toc33437915)

[All Reserved 43](#_Toc33437916)

[My Borrowed 45](#_Toc33437917)

[My Reserved 47](#_Toc33437918)

[Borrow 49](#_Toc33437919)

[Reserve 51](#_Toc33437920)

[Overview of Chatbot 52](#_Toc33437921)

[Training 53](#_Toc33437922)

[Chat 54](#_Toc33437923)

[2.2-Data Dictionary 55](#_Toc33437924)

[Composers 56](#_Toc33437925)

[Genre 56](#_Toc33437926)

[Language 57](#_Toc33437927)

[Music 57](#_Toc33437928)

[musicinstance 57](#_Toc33437929)

[Borrowed 58](#_Toc33437930)

[Database Entity Relationship Diagram 59](#_Toc33437931)

[SQLite3 60](#_Toc33437932)

[Auth\_Group 60](#_Toc33437933)

[auth\_group\_permissions 60](#_Toc33437934)

[auth\_permissions 60](#_Toc33437935)

[auth\_user 61](#_Toc33437936)

[auth\_user\_groups 61](#_Toc33437937)

[auth\_user\_user\_permissions 61](#_Toc33437938)

[catalog\_borrowed 62](#_Toc33437939)

[catalog\_composer 62](#_Toc33437940)

[catalog\_genre 63](#_Toc33437941)

[catalog\_language 63](#_Toc33437942)

[catalog\_music 63](#_Toc33437943)

[catalog\_music\_genre 64](#_Toc33437944)

[catalog\_musicinstance 64](#_Toc33437945)

[django\_admin\_log 64](#_Toc33437946)

[django\_content\_type 65](#_Toc33437947)

[django\_migrations 65](#_Toc33437948)

[django\_session 65](#_Toc33437949)

[sqlite\_sequence 66](#_Toc33437950)

[Planned SQLite Queries 66](#_Toc33437951)

[Class Diagrams 66](#_Toc33437952)

[Admin 67](#_Toc33437953)

[Forms 67](#_Toc33437954)

[Models 68](#_Toc33437955)

[Main Algorithms 69](#_Toc33437956)

[Training() 69](#_Toc33437957)

[Chat() 71](#_Toc33437958)

[Suggestions() 72](#_Toc33437959)

[Data Structures 75](#_Toc33437960)

[Planned Security Measures 77](#_Toc33437961)

[3-Technical Solution 77](#_Toc33437962)

[Chatbot Code 79](#_Toc33437963)

[The dictionary this is calling from 83](#_Toc33437964)

[Models for Database 84](#_Toc33437965)

[Client-Server Model 91](#_Toc33437966)

[Admin 105](#_Toc33437967)

[Templates 109](#_Toc33437968)

[Overall page 109](#_Toc33437969)

[Indexes 116](#_Toc33437970)

[All Music (As the button only shows up if the music is available and you have gone to its detailed page and you are allowed to borrow music) 125](#_Toc33437971)

[Actual Music Detail Page with button to reserve 127](#_Toc33437972)

[Borrowed/Reserved Page for Users to see their current music 128](#_Toc33437973)

[Borrowed/Reserved For All 129](#_Toc33437974)

[Login 131](#_Toc33437975)

[Password Edit Form 132](#_Toc33437976)

[Password Reset Done 132](#_Toc33437977)

[Password Reset Confirmation 132](#_Toc33437978)

[Password Reset Complete 133](#_Toc33437979)

[Password Reset Email 133](#_Toc33437980)

[Borrowed Pie Chart to allow you to see the most borrowed pieces of music 133](#_Toc33437981)

[Borrowed List to see most borrowed music as a list 135](#_Toc33437982)

[Composer List 135](#_Toc33437983)

[Composer Detail 136](#_Toc33437984)

[Feedback Page to confirm pages 137](#_Toc33437985)

[Return Page 138](#_Toc33437986)

[Review Page 139](#_Toc33437987)

[Suggestions Page 140](#_Toc33437988)

# 1-Analysis

## 1.1-Introduction

### 1.1.1-Background

I will be creating a music loan website for Barnet Education Arts Trust. This will be based off library loaning online systems such as Barnet Libraries. Library Management systems are a way of keeping track of stock in a library. The first one was created over 20 years ago and hasn’t changed much.

Library management systems have been based off of stock tracking websites and let you know where items are and when they are going to be returned or if it is a shop when the stock is going to be delivered.

When I have finished coding my website, I will test it by giving it to my orchestra conductor to use for his music and test it for all of his bands. I will also test it across different operating systems and different devices.

I will be investigating how to send automatic emails from a server, how to record music, how to get ISBN numbers, possibly how to scan barcodes to get the code from it.

### 1.1.2-Project Scope

I will aim to finish this project over the course of several months. This table gives me a rough outline of every task I am going to need to complete, the time it will take, and the date it needs to be done by

|  |  |  |  |
| --- | --- | --- | --- |
| # | Task | Duration | Due Date |
| 1 | Introduction | 2 |  |
| 2 | Investigation | 2 |  |
| 3 | Constrains, requirements, limitations | 2 |  |
| 4 | Objectives | 1 |  |
| 5 | Proposed Solution | 1 |  |
| 6 | Finish Analysis |  | 13/10/2019 but add final changes before final deadline |
| 7 |  |  |  |
| 8 | Data Flow Diagrams | 3 |  |
| 9 | Data Dictionary | 1 |  |
| 10 | Design the database + Normalisation | 10 |  |
| 11 | Create Class Diagrams and class definitions | 1 |  |
| 12 | Pseudocode |  |  |
| 13 | Plan Security Measures |  |  |
| 14 | Plan test strategy |  |  |
| 15 | Finish Documented Design |  | 13/10/2019 added in a few changes along the way |
| 16 | Implement login system | 1 |  |
| 17 | Implement Reservation System | 2 |  |
| 18 | Implement Borrow System | 6 |  |
| 19 | Implement Returning System | 3 |  |
| 20 | Implement Suggestions Page | 10 |  |
| 21 | Implement AI and neural network on Google Cloud | 7 |  |
| 22 | Finish Implementation |  | 09/01/2020 after due to mocks |
| 23 | Test system on different OSs | 7 |  |
| 24 | Test using invalid Inputs | 1 |  |
| 25 | Get feedback from client | 1 |  |
| 26 | Finish Testing |  | 01/02/2020 after due to mocks |
| 27 | Did I meet all my objectives? | 1 |  |
| 28 | Go through feedback on game | 1 |  |
| 29 | How could it be improved | 1 |  |
| 30 | What could have been done differently | 1 |  |
| 31 | What was easy/hard? | 1 |  |
| 32 | Finish Evaluation |  | 11/02/2020 after due to mocks |
| 33 | Hand IN Whole Thing |  | 28/02/2020 |

### 1.1.3- The Client/Supervisor

My clients are Barnet Education Arts Trust and my dad. I have regular meetings with my dad and will meet when I can with a represented member of Barnet Education Arts Trust. Barnet Education Arts Trust are a music charity for the London Borough of Barnet. They run most music schools and lessons in the borough. As part of leading music centres/schools they have a music office. The music office is where all the music is kept when not on loan to conductors at these centres/schools. They don’t have an online system, at the moment, of keeping track of the music so are constantly losing music. For this reason, they have employed me to create this system for them.

### 1.1.4- The Prospective Users/Audience

The website is particularly made for Barnet Education Arts Trust, although it could be used with a few edits for any other music office. It will be designed in an easy way to be edited for whichever company will want to use it.

To achieve this, I will use OOP to create objects that can be edited and an admin page that is easy to use. This should make editing easy. It should also make it easy to maintain.

## 1.2-Investigation

### Existing Websites

1. **Barnet Library Page**

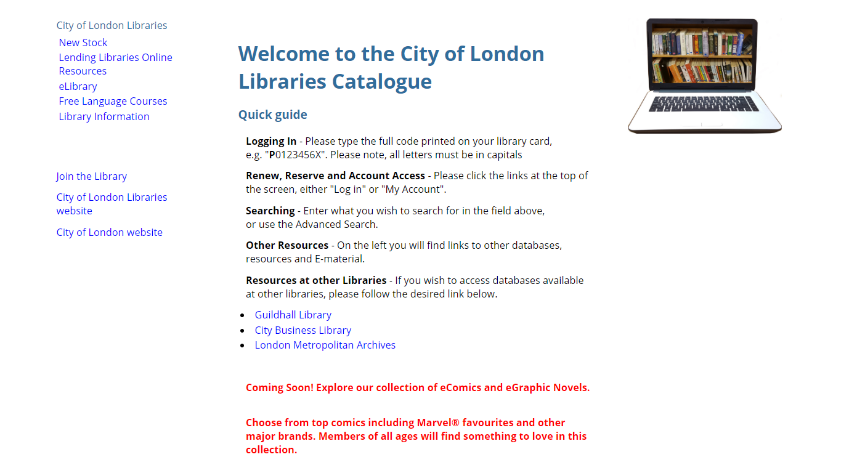
Barnet Libraries Website is the website for the catalogue of books in all the Barnet libraries. The website allows you to renew books and to see all of the books in the libraries, but you can’t reserve books off of it. They show who it is made for, what type of thing it is, new additions.



|  |  |
| --- | --- |
| Pros | Cons |
| -Clear buttons  -Easy to login and renew music  -Shows the new additions for different things | -No way to reserve stock  -Doesn’t show the genres as part of new things. |

In my project, I will replicate the clear buttons and showing new music. I will add a way to see all books. However, I will need to add a way to reserve and return music and show all the genres.

1. **City of London Library Page**



The City of London Library Page is where members of the City of London Library can reserve or renew music. You can also view all the books. It has queues for deciding who gets the books when they reserve it in order. It also has filters so you can filter by author, by year released, by genre, by whether non-fiction or fiction etc.

|  |  |
| --- | --- |
| Pros | Cons |
| -Clear buttons  -Easy to login and renew music  -Can renew Books  -Can reserve books  -Nice Welcome Page  -Multiple Filters | -Doesn’t show the new additions for different things |

I am going to try to copy everything but add in a way of showing all the new music with filters

### Questionnaire

I wrote a questionnaire to find out exactly what my users thought of the current manual loaning system as well as what else they would want from a new online system.

A screenshot of a cell phone

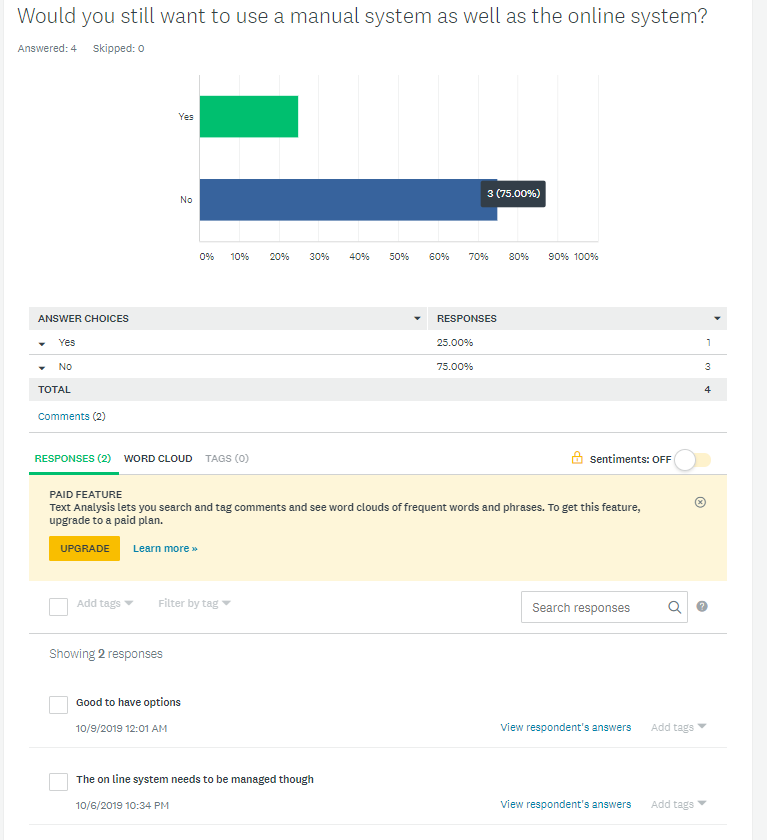
Description automatically generated

All of my users want an online catalogue and reservation system to replace the textbook that is currently in use.

A screenshot of a cell phone

Description automatically generated

Most of my users think that the catalogue part of the website is more important than the reservation system with 2/3 wanting the catalogue so that they can actually see what is available. The problem they have at the moment is having to go in hoping that there is a piece of music, but it may not be there when they arrive. You can’t have the reservation system without the catalogue, so it is the most important part. I will still need the reservation system as everyone in the previous question asked for it and have said it is very useful. Also, this is what will make my project A-Level Standard. The other parts will make the user’s use easier like other library systems.



Most of my users say that they wouldn’t want the manual system. This could be due to getting in the way or being absolutely rubbish. One user wants to still have the option of the manual system. This means that I will need to allow admins/librarians to input a user that is not themselves in the music being reserved and having to have someone go through the book everyday/week to add any music being reserved so that other people who want to reserve music online so that they can see what is still available.

A screenshot of a social media post

Description automatically generated

When I asked what else is needed my users all asked for a way to approximate standard of piece. This will mean looking into the way music is graded and finding out how to grade it. It also means editing the database so that it includes more information on the grading. One other thing asked for is a way for conductors to talk about the music. This will mean having a paragraph box that is allowed to be empty, but I will need to make sure only 1 user can edit this box at a time and set up a queue system.

A screenshot of a computer

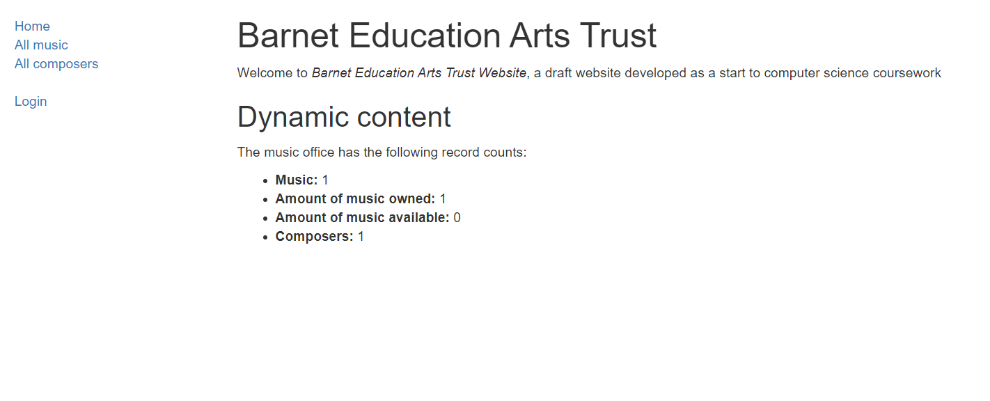
Description automatically generated

One user has said that the current system meets their needs meaning that all they need is a way of seeing current music and letting people know where the music is. They did say though that an online system that can be accessed remotely would be more convenient for them. The other 2 users have said that the system is not good for them. They want a catalogue that is kept up to date about where music is. They also want the capabilities/age recommendations for each piece.

### Prototyping

Over the summer I programmed a neural network in tflearn and TensorFlow to familiarise myself with them and how to create the different layers needed for a neural network. This isn’t doing what I need it to do for the final project, but it was just me learning. It is based off of the tutorial series starting with <https://www.youtube.com/watch?v=3zeg7H6cAJw>.

I also programmed a prototype of my website. This let me familiarise myself with Django. This is what it looked like:



In Django you have to create everything yourself except for the admin page which is created for you. You can edit this page to show you everything. You can use an automatically created view, but this is useless. I did this originally while learning. Each page and each button is based around views. These views can do everything. Pages can be created in views but you would have to pass information from the form to the view so you might as well do it in the view. Filtering is done in a form as it is easier to send the information back. You send information to a html page that will then pass information back to the view.

I have also written a very basic suggestions algorithm that only works on a csv file which I won’t be using. It looks at a user’s previous reviews and finds the most recent one. It then looks at the review and looks for user’s who have given the same review as you and will return you those user’s other reviewed pieces of music. In the final edition it will look at only the current user’s good reviews and then look for users who have similar reviews and return their best reviewed music unless you have borrowed/reserved/cancelled that piece of music before.

Here it is:

import csv

import sys

import random

number = 1

csv\_file = csv.reader(open('reviews.csv', "r"), delimiter=",")

print("Debug")

x=[]

for row in csv\_file:

x.append(row)

def get\_user\_reviews(x):

reviews = []

for i in x:

if i[2] == str(number):

reviews.append(i)

return reviews

def date\_check(reviews):

biggestdate = "01/01/0001"

reviewToUse = []

print(biggestdate)

print()

print()

print()

print(reviews)

for i in reviews:

print("Date Check")

print(i)

print("Test")

if i[3]>=biggestdate:

print(biggestdate)

print(i[3])

print()

print()

biggestdate = i[3]

reviewToUse.append(i)

return reviewToUse

def review\_check(reviewToUse):

print(reviewToUse)

stars = reviewToUse[0][3]

print(stars)

return stars

def music\_check(reviewToUse):

music = reviewToUse[0][1]

print(music + "Hello")

return music

def other\_reviews(number, stars, music, x):

otherReviews = []

choices = []

for i in x:

print(i)

print(str(stars))

print(str(number))

if i[1] == str(music) and i[3] == str(stars) and i[2]!= str(number):

otherReviews.append(i)

choices.append(i[0])

print("Hello")

return otherReviews, choices

def suggest(number, x):

reviews = get\_user\_reviews(x)

reviewToUse = date\_check(reviews)

stars = review\_check(reviewToUse)

music = music\_check(reviewToUse)

otherReviews, choices = other\_reviews(number ,stars, music, x)

print(otherReviews)

FinalChoices = []

if len(choices) > 0 and len(choices) <= 5:

for i in choices:

FinalChoices.append(i)

elif len(choices) > 5:

for i in range(0,5):

choice = random.choice(choices)

choices.remove(choice)

FinalChoices.append(choice)

for i in otherReviews:

for j in FinalChoices:

if i[0] == j:

print(i)

suggest(number, x)

The final edition can work straight on the database and is way shorter as it just uses a few queries rather than having to search through every single bit of the csv. It also links in to a view.

### Neural Network/Suggestions Algorithm

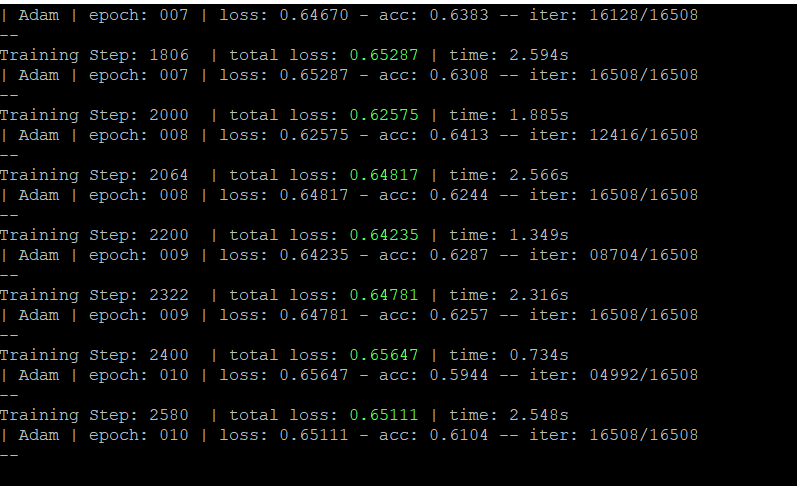
My current Neural Networks algorithms uses different levels. The first layer is the input layer. I then have fully connected levels. In this every neural is connected to every neuron in the previous layer. This uses a lot of memory and computation compared to convolutional layers where each neuron is only connected to a few nearby neurons. On the other hand, the fully connected layers are easier to program. You then train based off random data which over time keeps being used with a dropout rate where only useful data gets stored until it is only perfect and won’t be over trained. This is a possibility for my final AI but probably won’t be what I can use as I won’t have the information to use for it.

Instead of using the neural networks algorithm I learnt over the Summer holidays, I will use a basic suggestions algorithm like what is used by Amazon and Netflix. There are 2 types of suggestions-based algorithms. One is called content-based filtering which is where you look at people have been recently been getting out and recommend music based off of it. The other is collaborative filtering systems which make recommendations based on user interactions. This would be when you see what other people got when they got out what you had.

I will first be building the content-based filtering system as this version doesn’t need the information from other users, so I only need one user. Once a few people are using the system and I have a bit more information stored I can then implement a collaborative filtering system so that I recommend based off of what other people have done.

For this suggestions I will first build a basic AI for content filtering using a tutorial from <https://towardsdatascience.com>

The current neural network program running looks as follows:



The code itself looks like this:

A screenshot of a computer

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

## 1.3-Constraints/Hardware

### Hardware Constraints

Server

I am building a website so it will need to be able to run on a normal server but one that can handle a bit of AI. This means that it shouldn’t use too much RAM or need a too powerful CPU. The specs needed are listed below.

|  |  |
| --- | --- |
| CPU | Intel Xeon |
| Graphics Card | Red Hat, Inc. QXL paravirtual graphic card |
| RAM | 100mb |

PC/Device connecting to website

You will need any machine that can handle a chrome page and has an internet connection.

### Software Constraints

I want to be able to run this on any machine so all they will need is a way of accessing the internet such as Google Chrome or Firefox. I will need to be able to test on different operating systems so I will need to borrow a Chromebook and an iPhone. I have a Windows Phone and a way of running Windows and Mac OS. Eventually, I want to be able to create an app that will be able to run on any Android Device able to run Android Marshmallow or after and on an IOS device running IOS 10 and above. This means that you are getting most devices.

### User’s Knowledge of Information Technology

My users will come from a range of IT backgrounds and may have little to no experience with computers. Therefore, I am aiming to make the website as accessible as possible. To do this, I will think carefully about UI design, making menus and buttons as clear as possible (e.g. using icons and large text). I will also include a chatbot eventually that will deal with any questions about how to use the website.

### Access rights

My users will have different permissions – librarians can: add/remove users; change reserved to borrowed then to returned. Admins can add/remove books, assign people user permissions and edit databases. Users can just see music and reserve available music. If I have time, I may add the option to have other user types.

## 1.4 Limitations

**Areas not touched / areas considered for future**

In the future there are other features I could add including having a link to a YouTube video for each set of music, but I don’t have the time for it and don’t have the expertise to be able to do it at this time. This would mean that the website would constantly be having to check YouTube for the best link and the most watched of the correct video and check the link it has to make sure that it is still working. This would slow the system down and require more RAM which I don’t have the money to upgrade.

I could also include a way for people to recommend music by making links from YouTube suggestions, but it takes a lot of time which I don’t have the time for. It would also mean creating an extra AI to work in the background and training this AI on top of the other would take a lot of resources which I don’t currently have.

## 1.5-Objectives

### General Objectives

My general objectives involve making sure that the website functions correctly, allows logins, allows reservations, allows librarians to change reserved to borrowed, allows returns and moves between pages correctly.

-The website functions as planned. The website has different pages with different links to other pages. The website allows reservations from previously created users. The users have a set group. These users will be able to see different pages depending on the group that they are part of.

-Users are able to be registered by the admin

-When the new user is registered it is hashed and sent to the database.

-The details are stored in an SQLite database

-Details of every music and how they are linked are saved in the database

-Details of every conductor is stored in the database

-Users are able to view the catalogue of music and composers

-Users are able to view the available music and reserve it

-Users can see their reserved/loaned music

-Different types of users.

-Visitors have to login

-Non-members can see all music and all composers but not reserve anything

-Members can do everything that a non-member can do but can also reserve only for themselves and see their suggestions

-Librarians can do everything a member can but can also reserve for others, allow users to actually borrow music, renew users music and can return music.

-Admins can see everything that a librarian can but can also edit the database completely and see all logs.

As part of the user types I will need permissions which will be outlined in design

### Specific Objectives

-The system should have a login page that allows you to login

-The system should have a main page with links to other pages such as:

-Login/Logout

-If it is login just give you a way to login as well as a way to get a new password

-If it is logout tells you that you are no longer logged

-Catalogue

-Has all music displaying the details including all the music instances. Also has a link to the composer

-Composers

-Displays all information about composer and links to music composed.

-Available

-My Reserved

-My Borrowed

### Optional Objectives (for if I have time)

1. Phone app to allow reservations to occur more easily as well as phone reminders about reservations and bookings
2. AI that is specific to each user and learns their normal music and gives them suggestions as they go along
3. Hard coded (or possibly AI based) chatbot to answer simple questions about how to use the reservation system
4. Calendaring
5. Suggestions algorithm that allows users to see what other users have given good reviews to based off of the fact that they gave a similar good review to the same piece of music
6. Filtering using JQuery. I chose this as it was the easiest to do in the time I had. I could have used Django-filter but it would have taken a long time and in the time constraints I couldn’t. JQuery allows easy lists similar to excel that will give me the filtering.
7. Adding in a separate user database to show that I could remove my user database and use BEAT’s instead.

## 1.6-Proposed Solution

I will be using a programming language called Python because it is a language I know fairly well, and it is a language with plenty of documentation. This will mean that I can easily learn the things I don’t know from reading about them and following tutorials. I tried to learn PHP from scratch in context to my project but couldn’t find anything.

I will be using a web framework called Django (<https://www.djangoproject.com/> or <https://github.com/django/django>). This library handles the backend of website building in Python. This includes talking to an SQLite3 database without the programmer having to know the precise SQL to do so. It also handles the running of the webpage and the admin functions. It also has extra parts do handle emails and creating of forms and the actual creation of the database. This means that you only have to know one programming language. I will however be using HTML so that I can use bootstrap (<https://getbootstrap.com/>) to make the website look nice. Django doing most things will allow me to focus more on complicated algorithms and building my AI for my suggestions page (and hopefully my chatbot). I tried using Flask but it has very limited resources and doesn’t provide you with much of a framework.

To implement my AI, I will be using a neural network library called TensorFlow and tflearn. These contain basic functions for creating layers for the network to learn from. This also allows it to learn off of itself to get better every time. Using this library will free up time working on my own functions to create layers as well as being able to not have to get it to teach it every time from new data. This will also allow it to function quicker and I can link it to the Google Cloud to do all the learning for me.

From my research, it is more suitable to run AI on a server where you get a GPU you can use rather than trying to host it on your machine which will never have a powerful enough GPU. Also, neural networks are the better way for me to go as they learn off of lists and are easy enough to program, they also take up far less space compared to deep learning. Deep learning would create more links but takes more time which I don’t have.

To run the server, I have chosen to run a virtual Ubuntu server on Digital Ocean. They offer a $5 tier and GitHub provides a free $50 as a student. For the web server I am just using Django. I have some familiarity with Apache but there is no need for it as Django can do everything. Eventually I would have to use Apache for the final running of the server but as it is still in development I will not be using it.

If I have time to create a phone app it will first be on android as I only have a windows laptop so can only run android studio. For this I would have to learn Kotlin. This is a java-based language made specifically for Android devices.

I would need to buy an iOS device and learn Swift to be able to create an app for apple watch and iPhone/iPad. This is a C based language aimed to work on just Apple Devices and only be written in MacOS/Linux.

For the database I am currently using SQLite3 as it is built into Django. It is an easy way to use python and databases without having to write hard SQL queries as instead you write 2 line queries which Django turns into sqlite3 queries for you. You also create readable models which have methods in them that are callable. I will for the moment create my own user model but in the future I can either import BEAT’s user table into my own or as they are using wordpress there is an easy API called django-wordpress that allows you to integrate WordPress’s user table into Django.

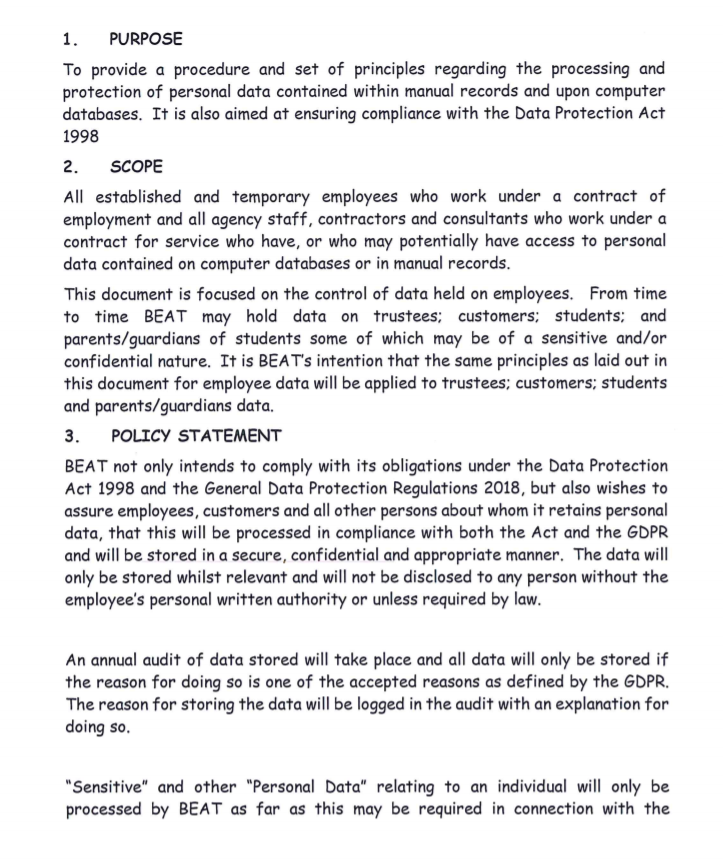
### Privacy Policy

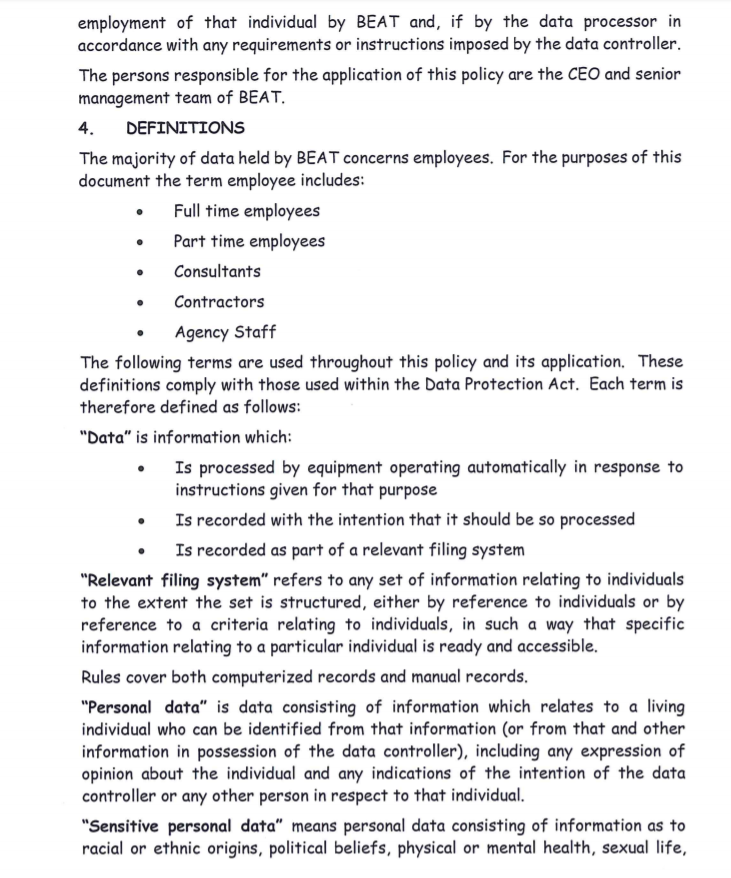
I am building a website and of 25th of May 2018 the GDPR laws came into force. This is the General Data Protection Regulation Act. It tells you what you can store and how long you should store it for. This may disappear in the future depending on Brexit but currently it is still necessary. My client currently has a privacy policy for their other websites so I will take this and update it.

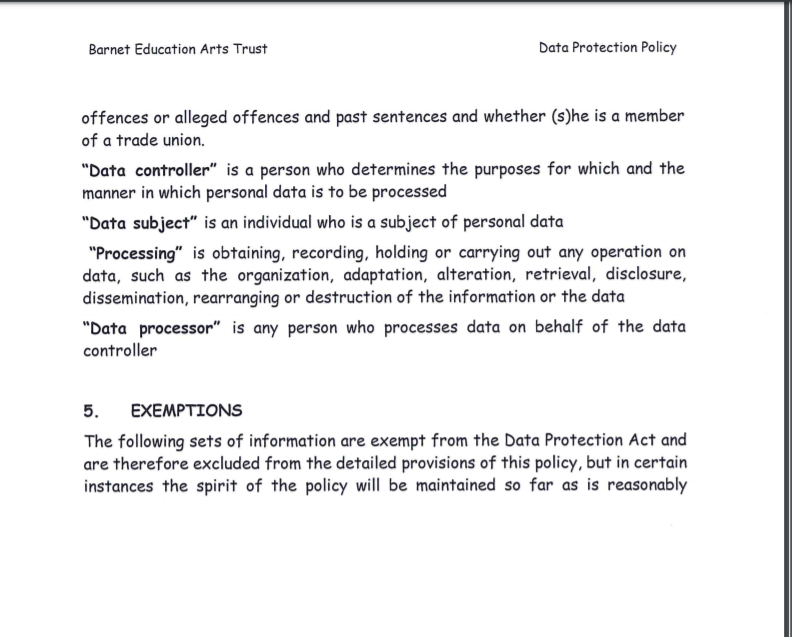
The things I need to include are:

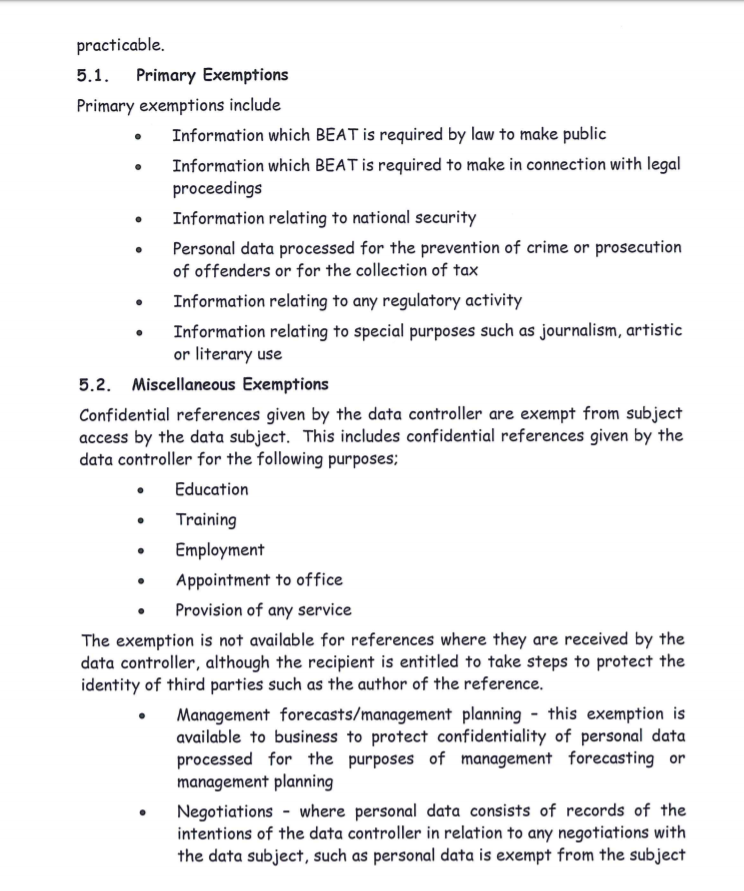
* What personal information I am collecting
* Why and how I am collecting it, and whether it’s mandatory
* What I am using the data for
* Who controls and processes the data
* What I retain, and where I store the data
* How I can keep the data secure
* Which third party companies can access the data
* Whether I am using cookies
* Whether I am using the data in automated decision-making
* Your users’ rights regarding their personal information

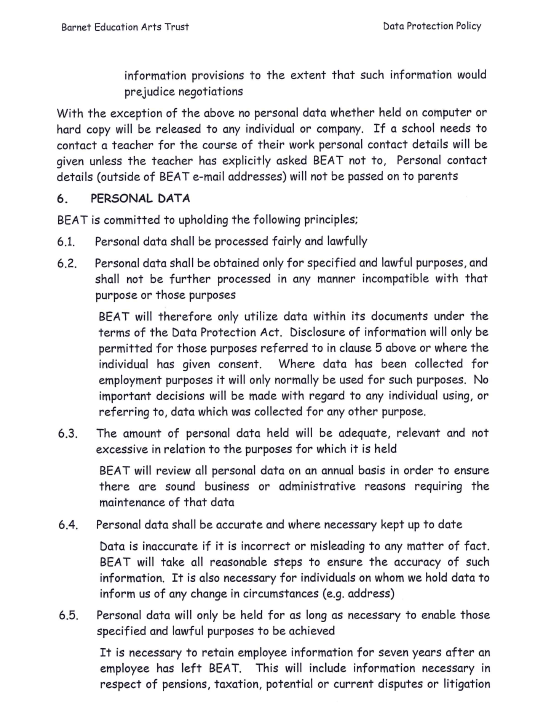
#### Current BEAT Privacy Policy

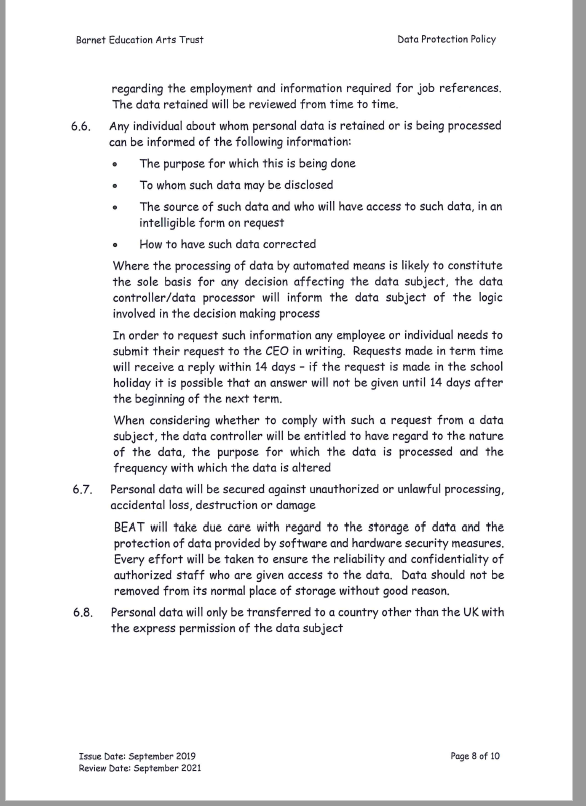


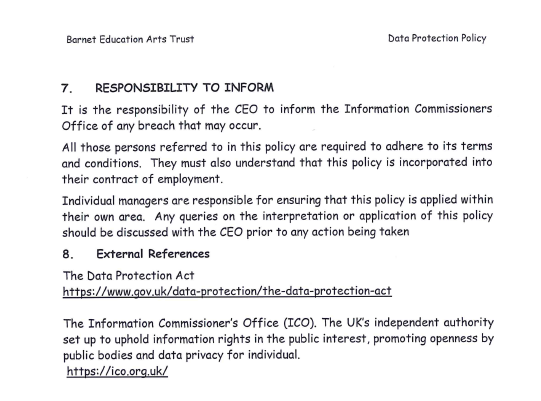












#### Bits to be Added to the Privacy Policy

**Using Your Data for Automated Decision-Making**

We collect information about what you take out to be able to make automated decisions of what else you might want to reserve and what to suggest you get out next. If you don’t want us using this information just let us know by email at admin@gshs.tech

# 2-Design

## 2.1-High Level Overview

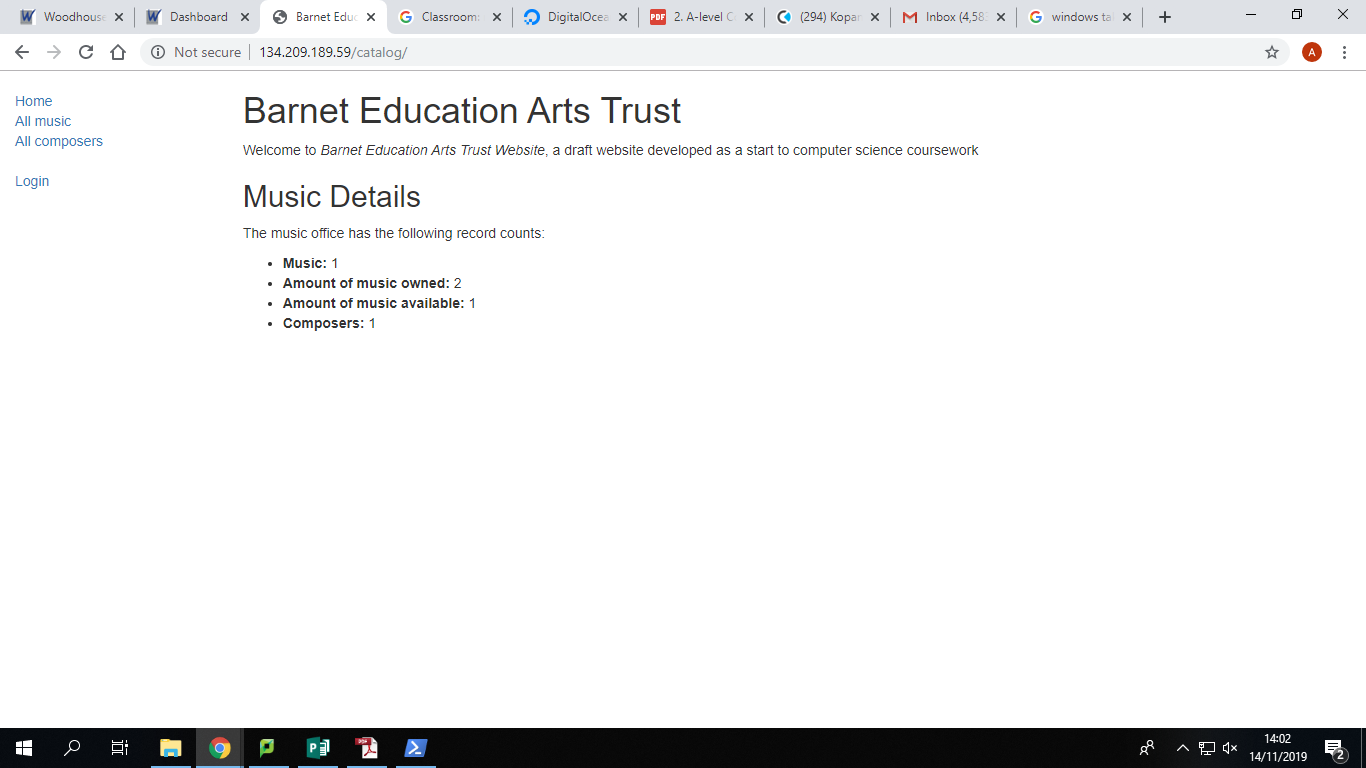
These flowcharts describe the general overview of how my website will work. It doesn’t have a registration page as this user is already created by the admin as it is part of a bigger picture.

### Main Menu

A close up of a map

Description automatically generated

This is what the logged out main menu looks like:

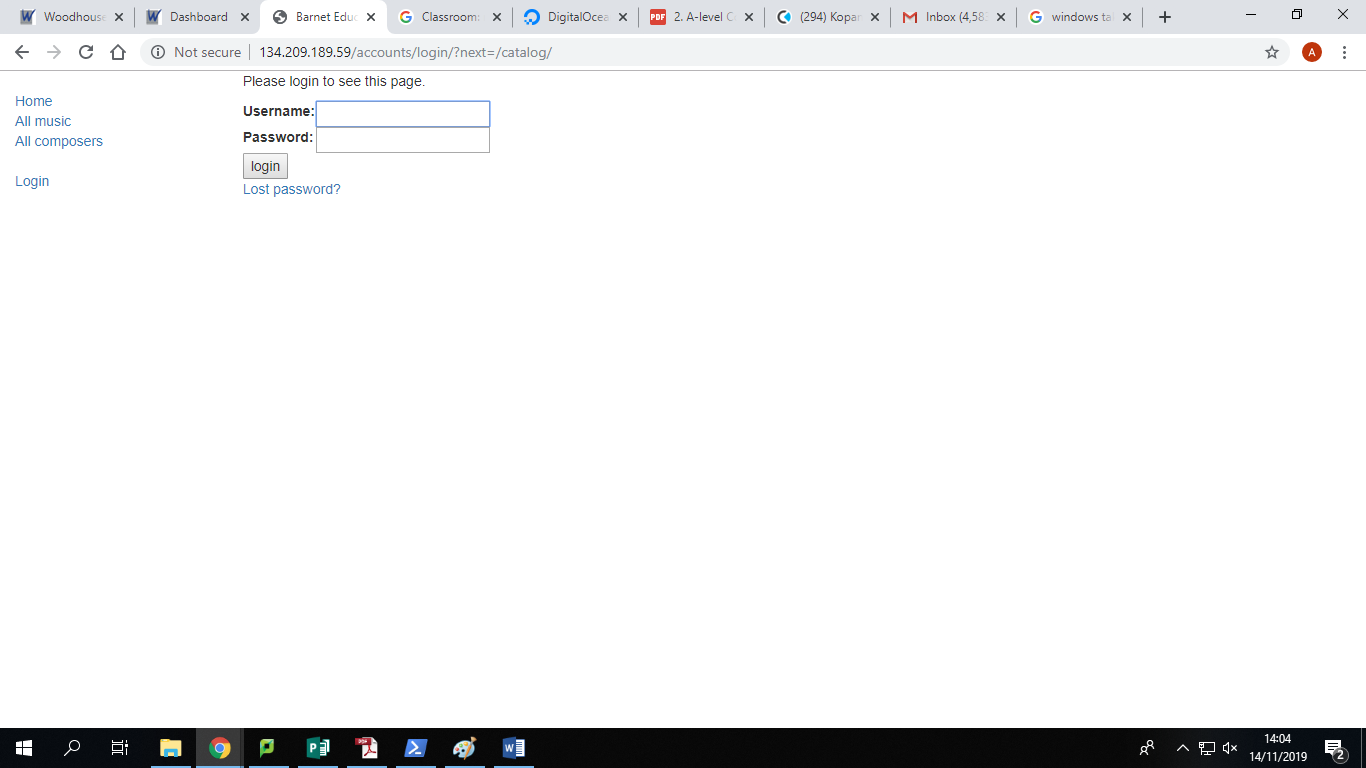


### Login

**A close up of a piece of paper

Description automatically generated**

This is what the login menu looks like:

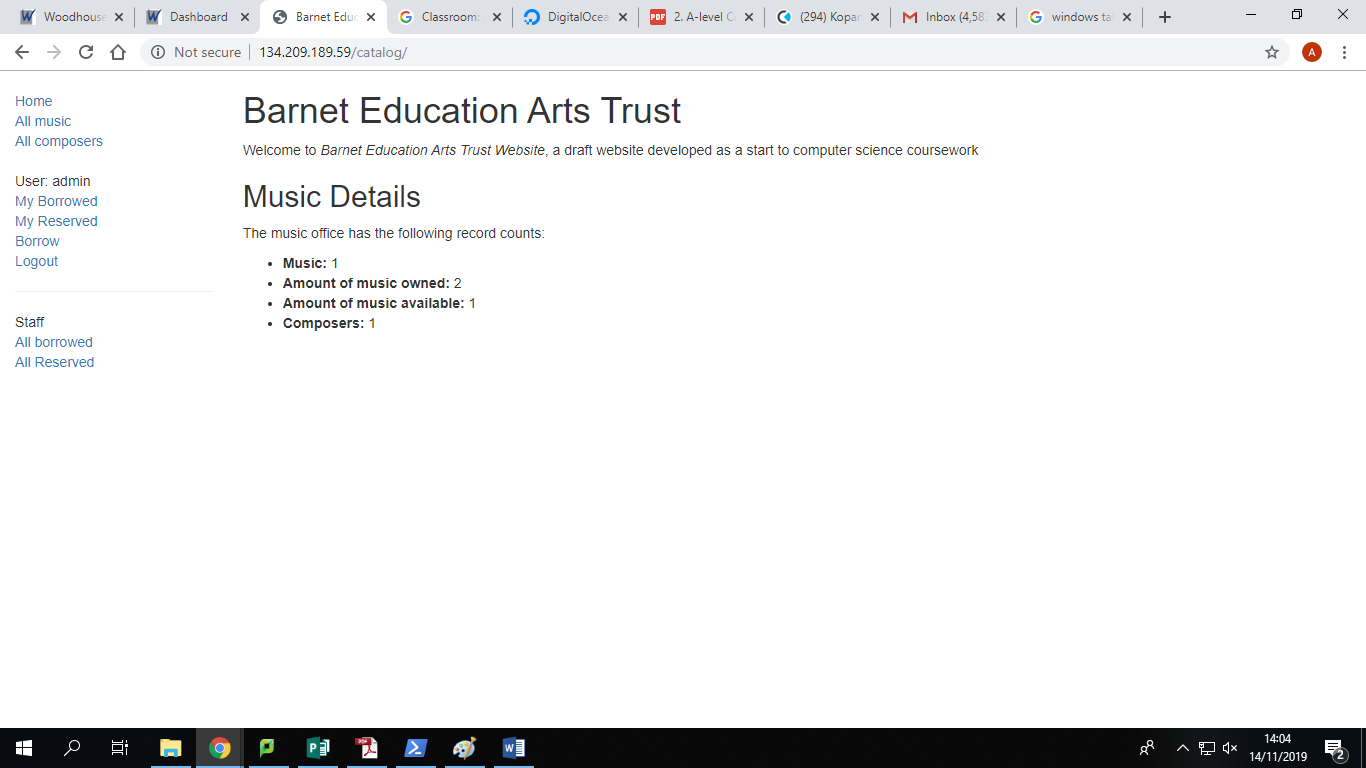


### Staff Main Menu Actions

**A close up of a mans face

Description automatically generated**

This is what the staff main menu looks like:

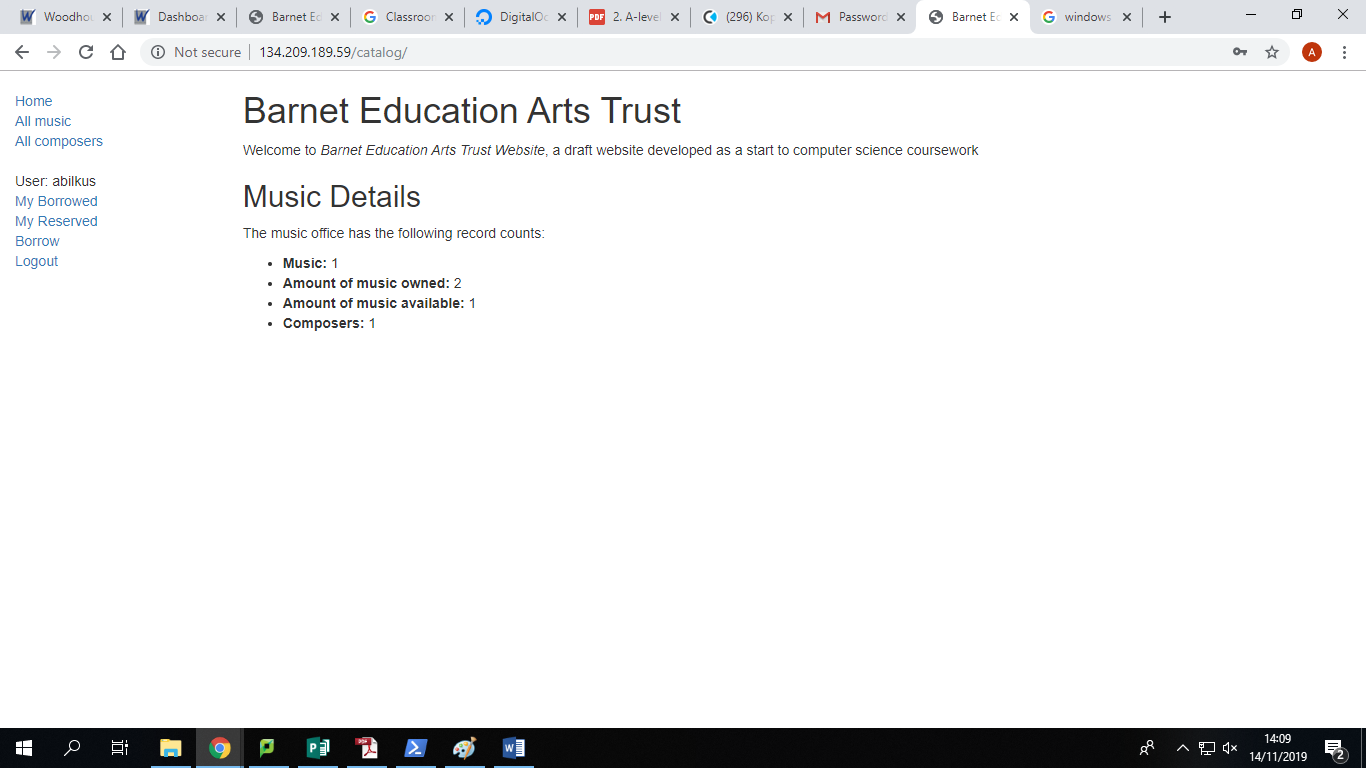
****

### Non-Staff Main Menu Actions

**A close up of a clock

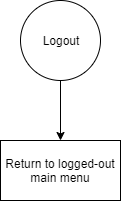
Description automatically generated**

This is what the non-staff main menu looks like:



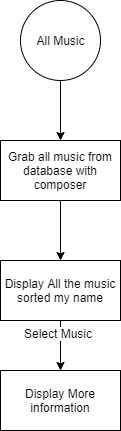
It is missing the staff actions

## Logout

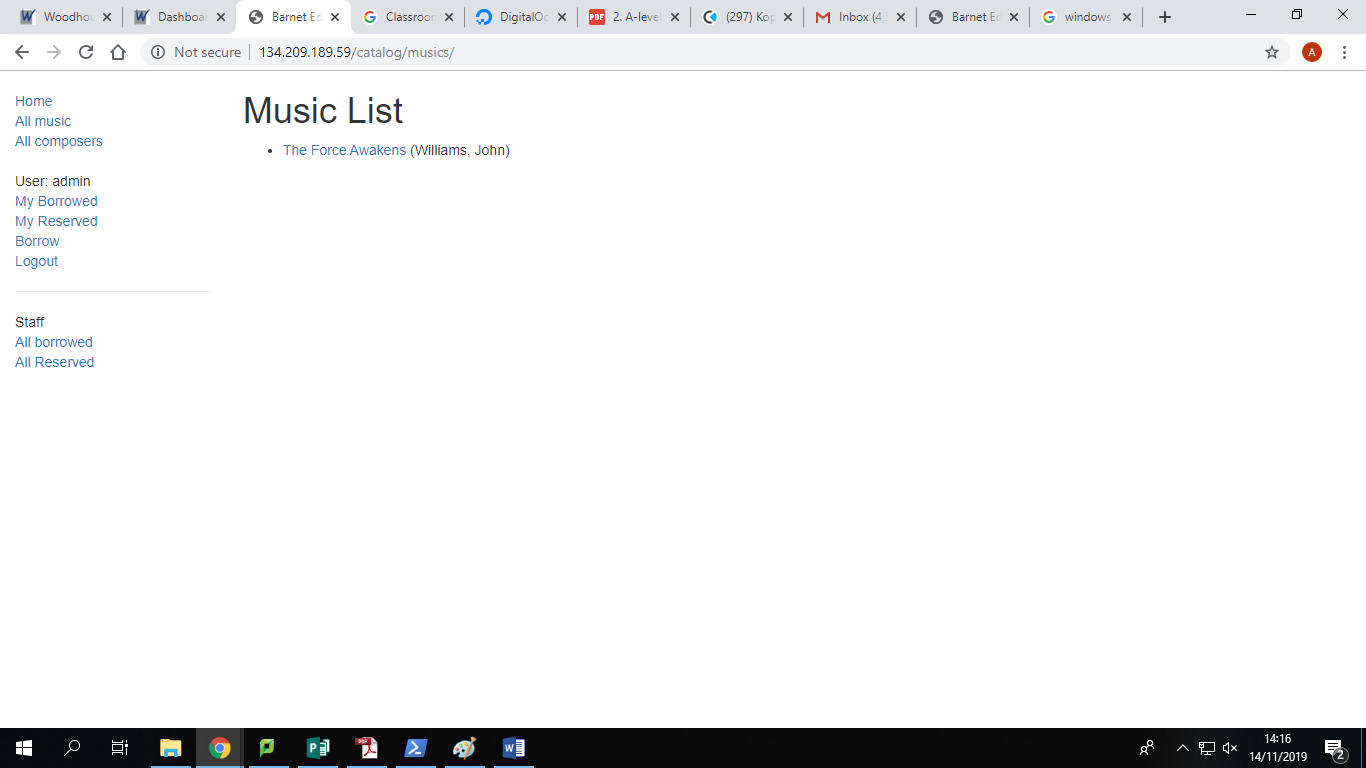
****

**The logout screen just takes you back so has no screen of its own**

### All Music

****

**This is what the All Music Screen Looks Like:**

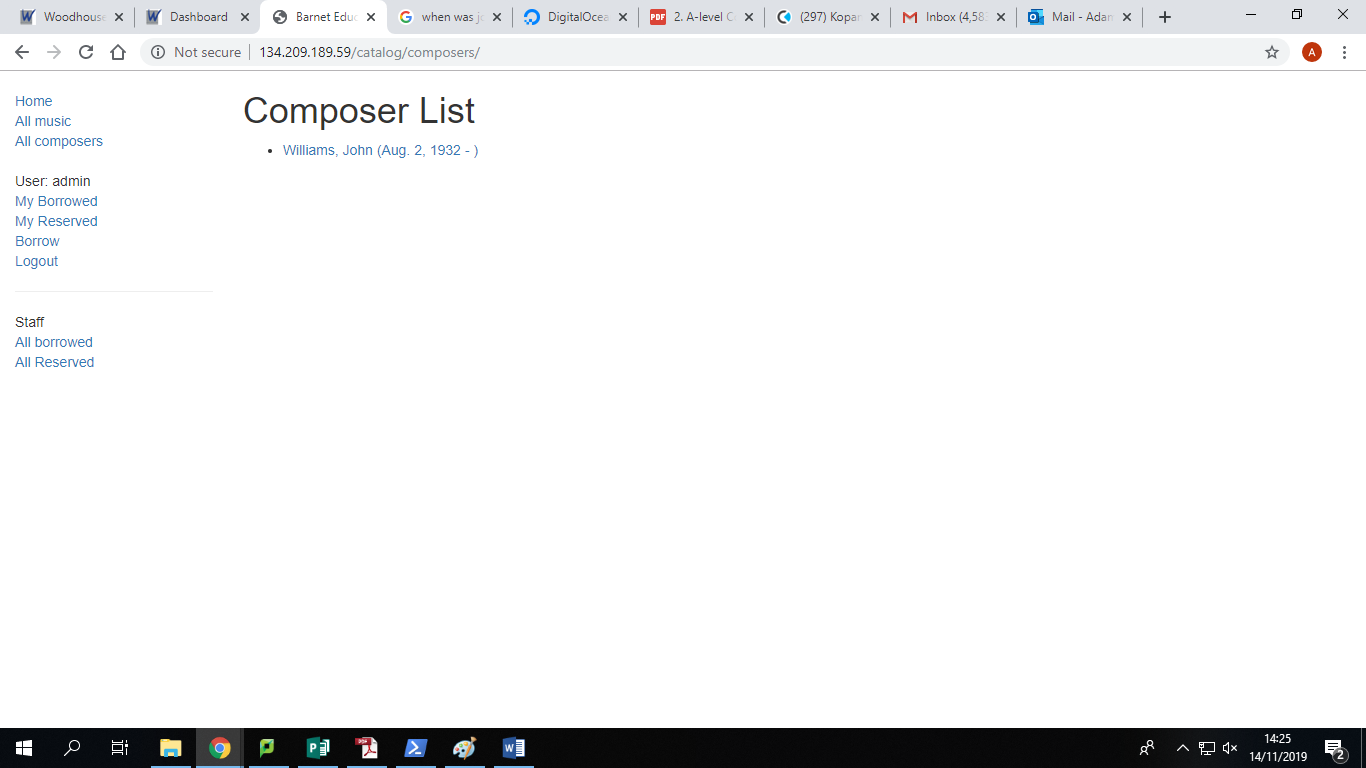
****

### All Composers

A screenshot of a cell phone

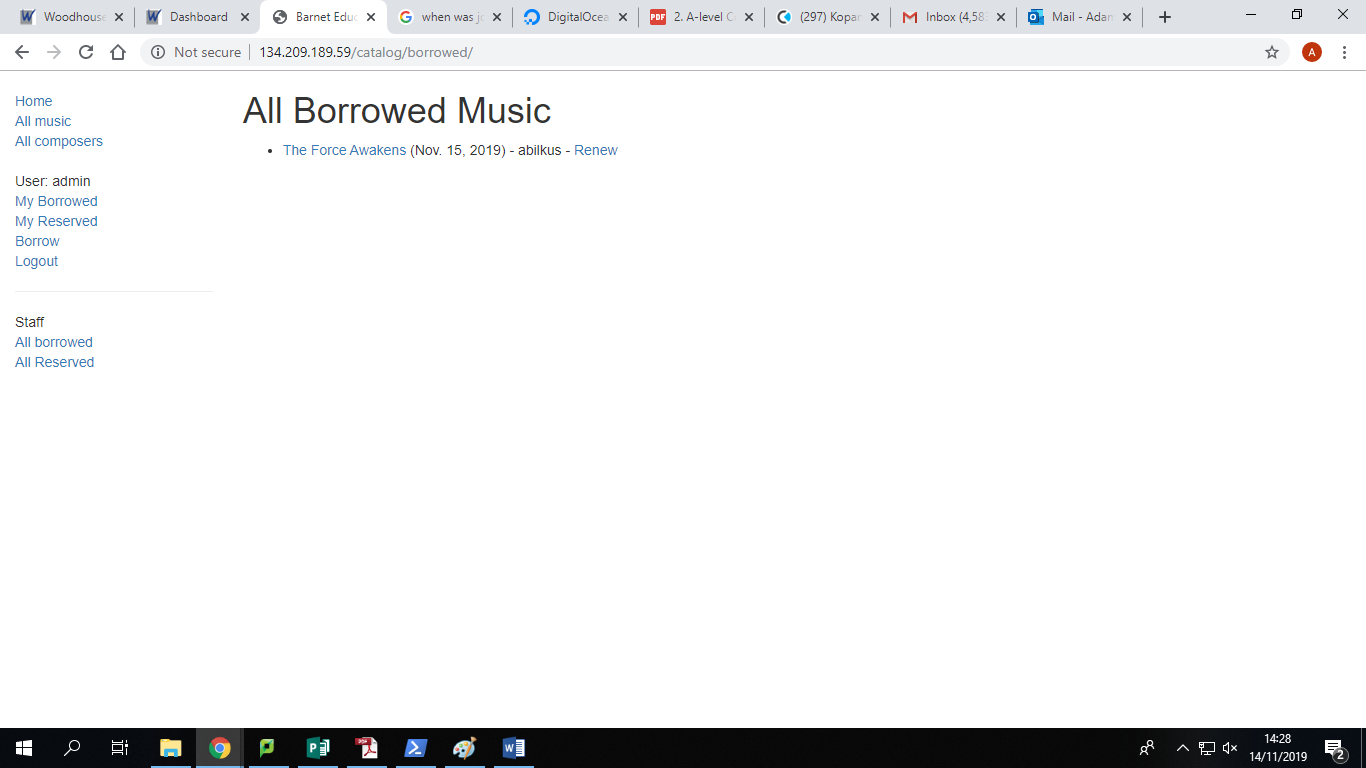
Description automatically generated

This is what the All Composers page looks like:

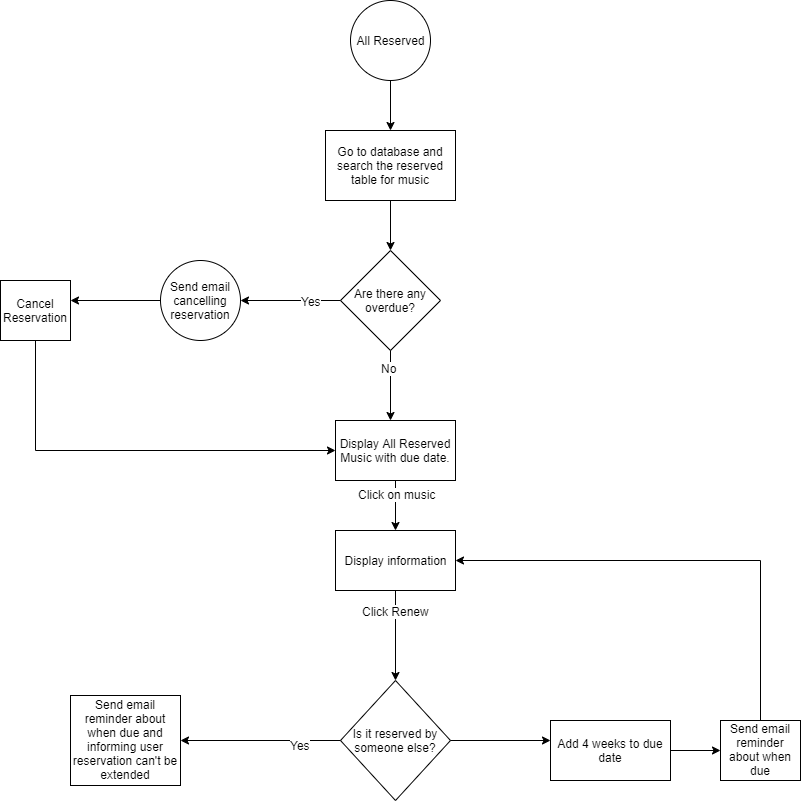


### All Borrowed

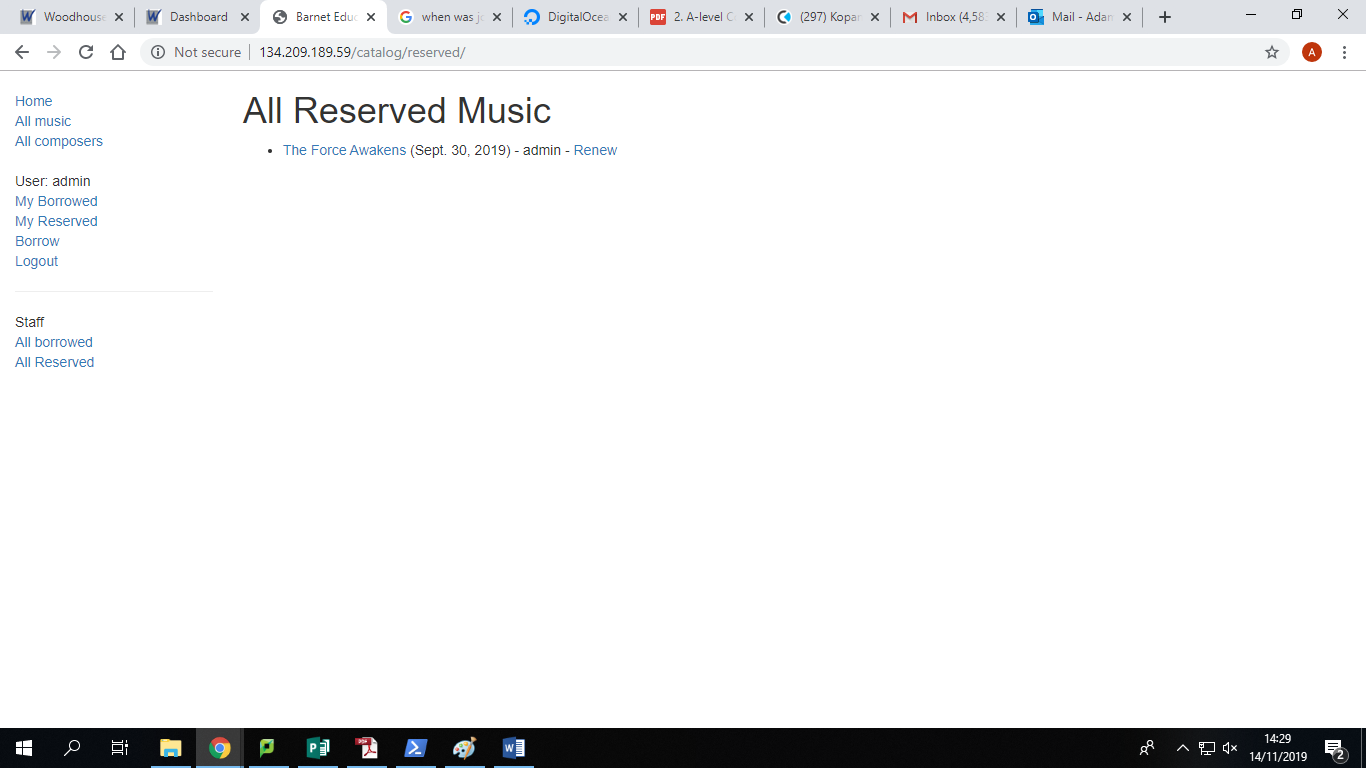
**The All Borrowed Page looks like this with music borrowed**

****

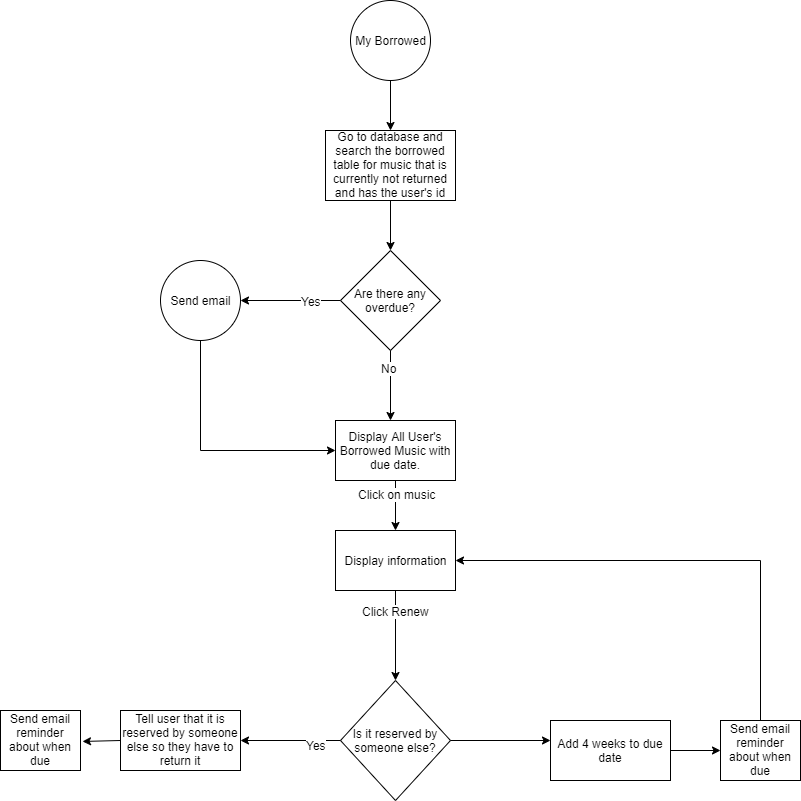
### All Reserved



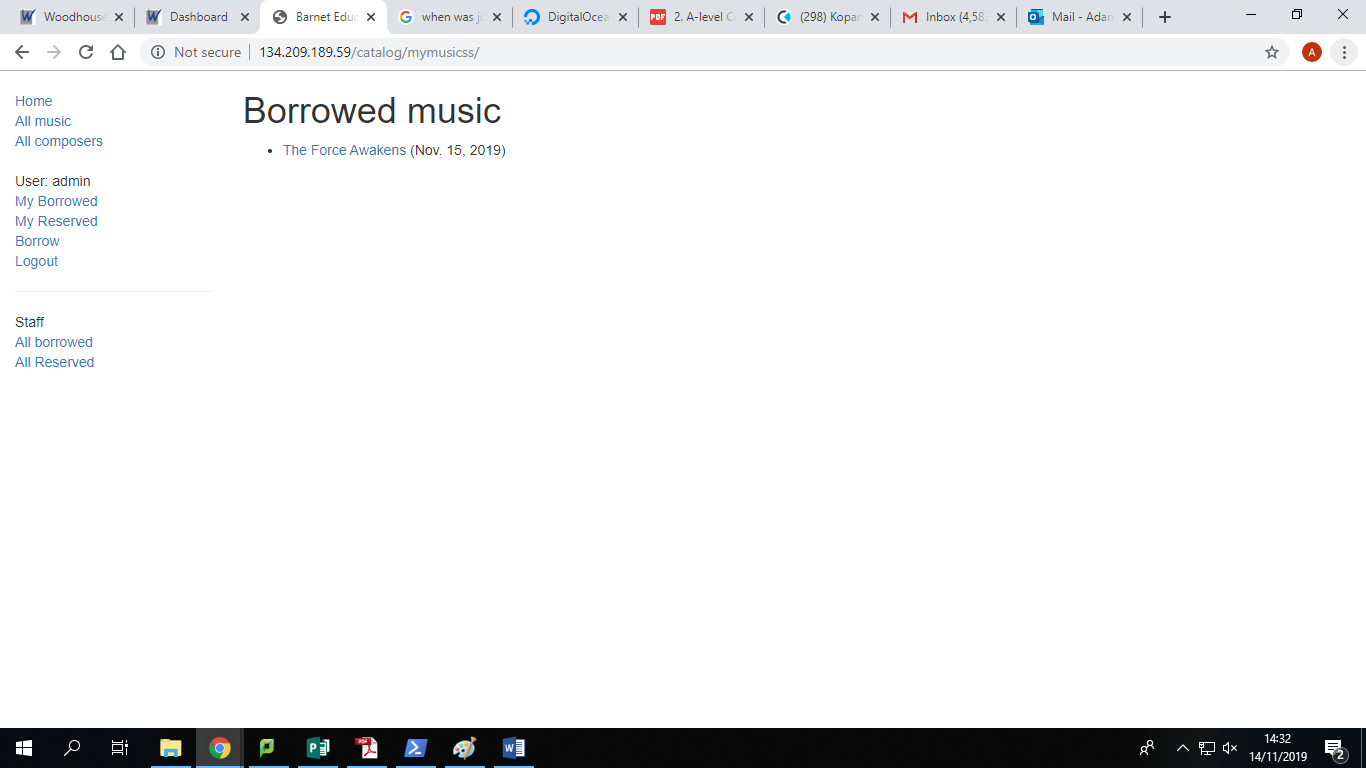
This is what the All Reserved Page looks like with reservations

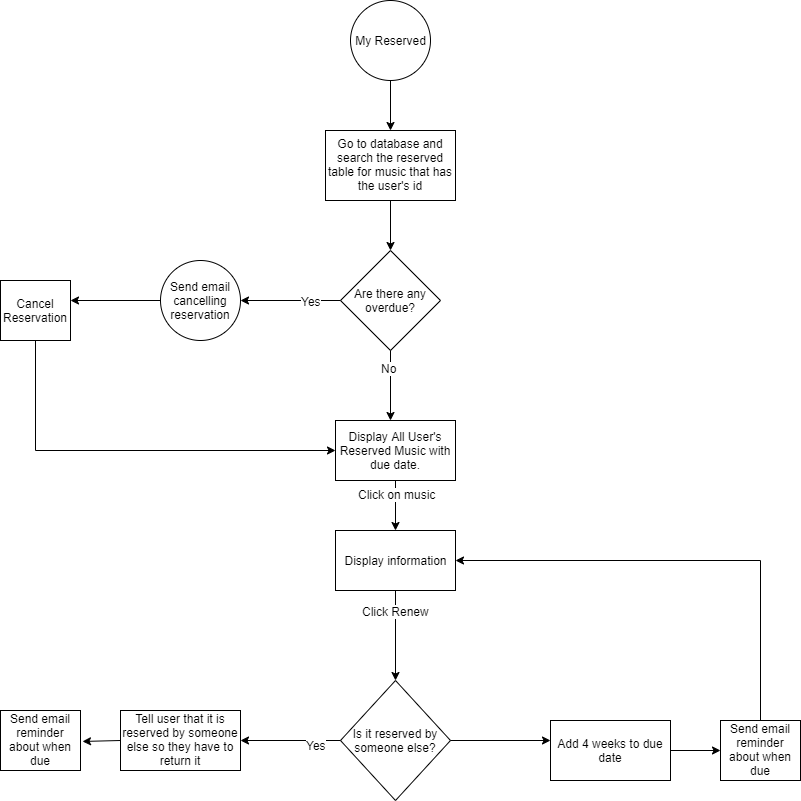


### My Borrowed

****

This is what the My Borrowed page looks like with bookings:

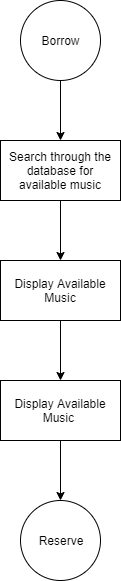


My Reserved****

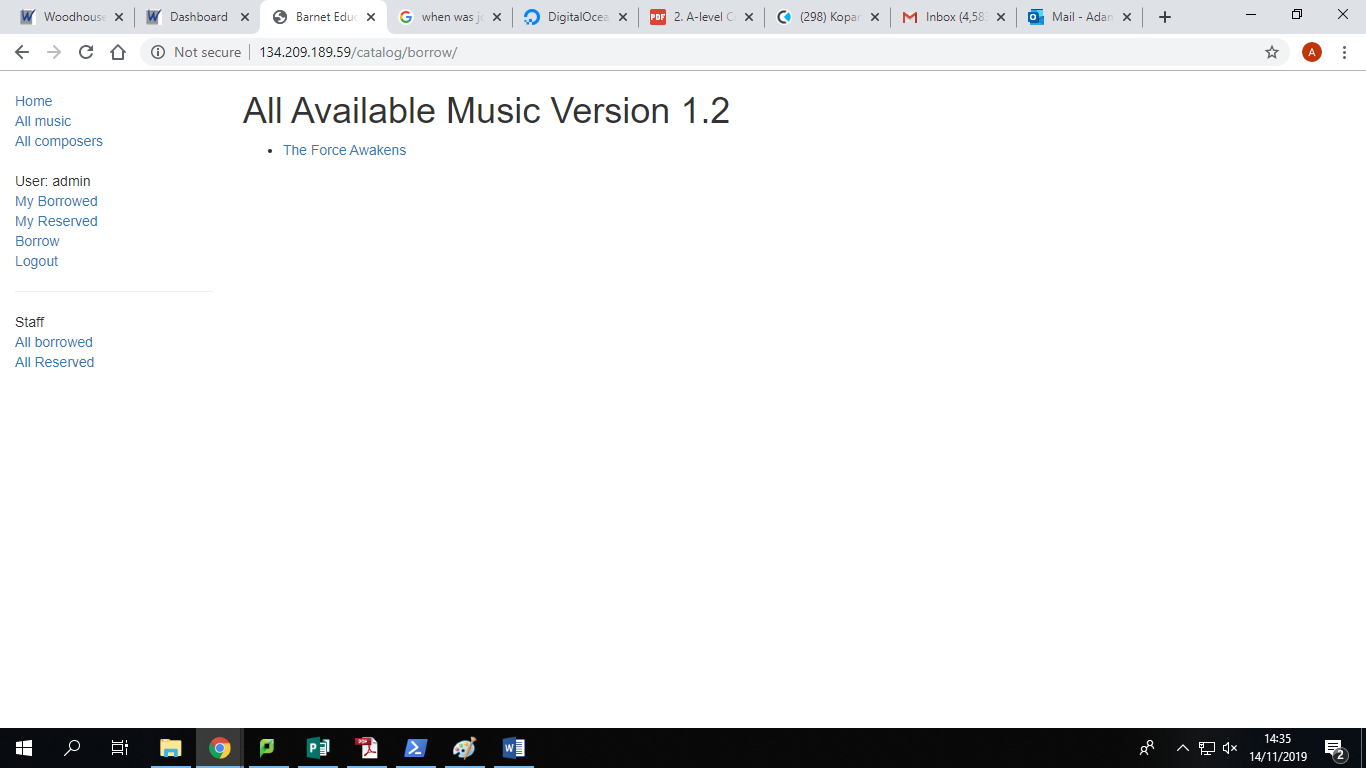
**This is what the My Reserved page looks like with reservations:**

****

### Borrow

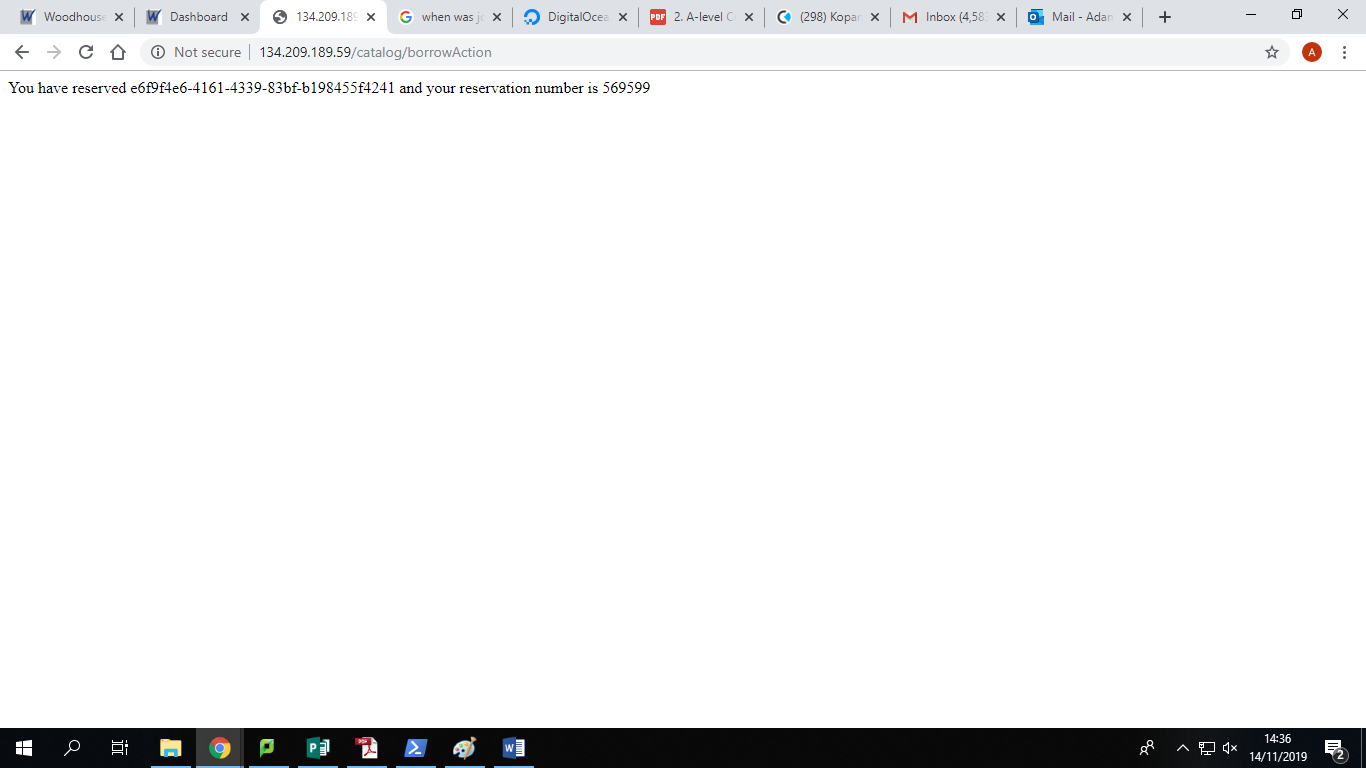
****

**This is what the borrow page looks like:**

****

### Reserve

**The reservation function doesn’t have a page but it could be shown with the reservation occurring:**

****

**The chatbot doesn’t yet have a page as it should be on the index page but I haven’t worked out how to implement it.**

### Overview of Chatbot

**A close up of a logo

Description automatically generated**

### Training

A screenshot of a cell phone

Description automatically generated

### Chat

**A close up of a map

Description automatically generated**

All of the pages will become prettier but that occurs at the end when I have more time.

## 2.2-Data Dictionary

I will be storing data such as user details, bookings, reservations, composers, music, music instances and past bookings in an SQLite3 database as this data needs to be stored long-term

User table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | Integer | Infinite | Giving each user a unique key | 01 | Auto-increment |
| password | varchar | 128 | Holding the hashed password | fafaffarrea | Password Management |
| last\_login | Datetime | 8 bytes | Holding the last\_login time of the user | YYYY-MM-DD HH:MM:SS | Datetime checker |
| is\_superuser | Bool | 1 | Checking whether they are the admin | True | Null |
| username | varchar | 150 | Holding the username of the user | Abilkus | Not empty between 3 and 15 characters |
| first\_name | Varchar | 30 | Holding the first name of the user | Adam | Not empty between 2 and 30 characters |
| email | Varchar | 254 | Holding email address of user | [adam@Bilkus.com](mailto:adam@Bilkus.com) | N/A |
| is\_staff | Bool | 1 | Tells you whether they can access parts of the admin page | True/False | n/a |
| is\_active | Bool | 1 | Tells you whether a user is active | True/False | n/a |
| date\_joined | Datetime | 8 | Date user joined | YYYY-MM-DD HH:MM:SS | Date |
| last\_name | Varchar | 150 | Last name of user | Bilkus | Between 2 and 150 characters |

### Composers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | integer |  | Unique id for each composer | 01 | Is integer |
| first\_name | varchar | 100 | First name of the composer |  |  |
| last\_name | varchar | 100 | Last name of composer |  |  |
| date\_of\_birth | date |  | Date of birth of user |  |  |
| date\_of\_death | date |  | Date of death of the user |  |  |

### Genre

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | integer |  | Unique id to each genre | 01 | Is numerical |
| name | varchar | 200 | Name of each genre | film | Minimum 1 letter |

### Language

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | integer |  | Unique id to each language | 01 | Is numerical |
| name | varchar | 200 | Name of each language | French | Minimum 1 letter or not applicable |

### Music

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | integer |  | Unique id to each set of music | 01 | Is numerical |
| title | varchar | 200 | Title of each set of music | Watermelon man | Minimum 1 letter |
| Summary | Text |  | Lets you give a short summary of the song |  |  |
| Composer\_id | Integer |  | Foreign Key to link to composer table | 01 | Numerical |
| Language\_id | Integer |  | Foreign key to link to language table | 01 | Numerical |
| Genre\_id | Integer |  | Foreign key to link to genre table | 01 | Numerical |

### musicinstance

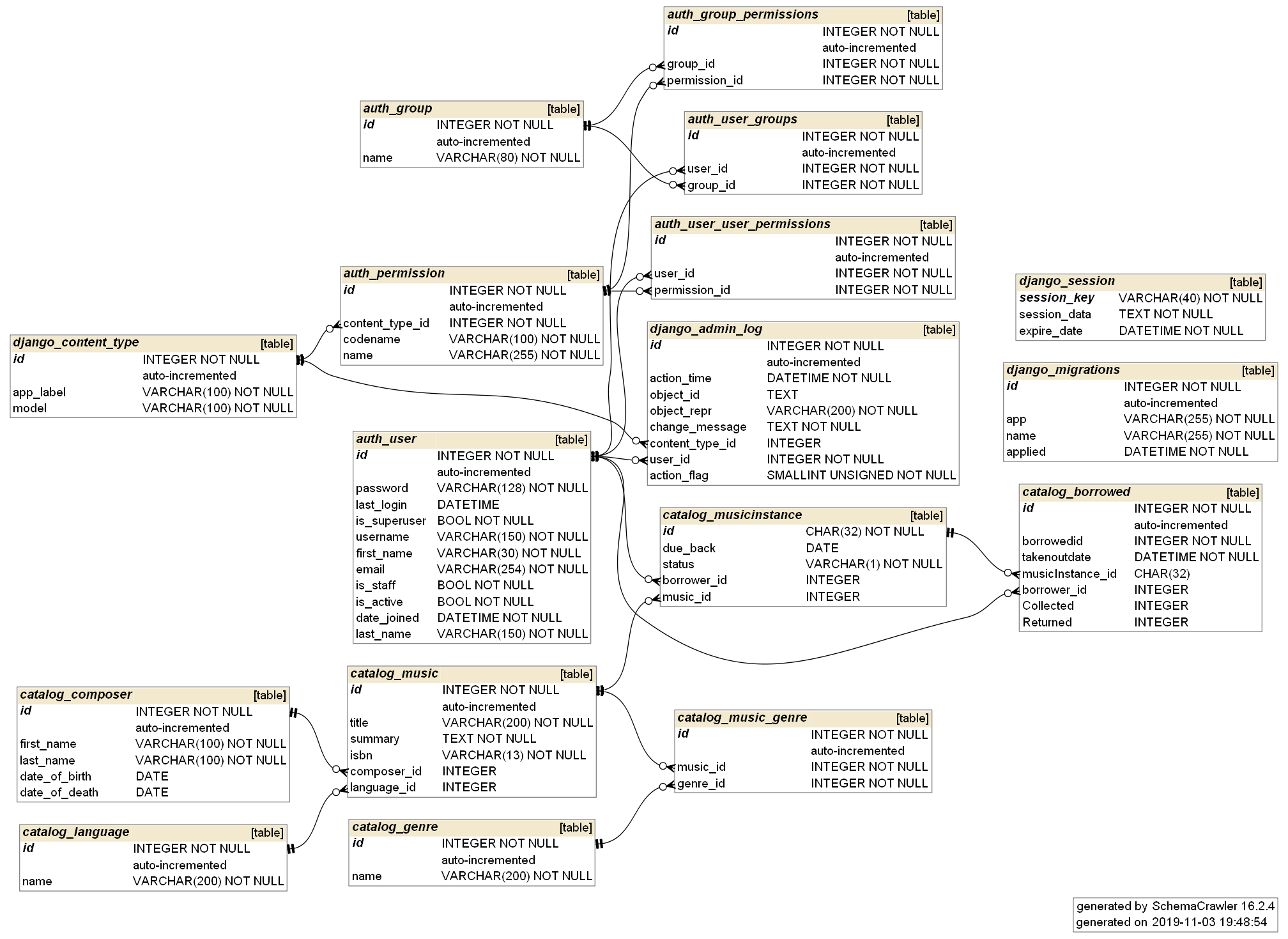
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | char |  | Unique id to each music instance (UUID) |  | Is numerical |
| due\_back | Date |  | Due back date | YYYY-MM-DD | Correct Date Format |
| status | varchar | 1 | Stores whether it is reserved/on loan/out for maintenance/available | R | Checks it is only 1 of 4 characters |
| Booking\_id | integer |  | Foreign Key to Reservation/Booking Table |  |  |
| music\_id | Integer |  | Foreign Key to Music Table | 01 |  |

### Borrowed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | char |  | Unique id to each music instance (UUID) | 01 | Is numerical |
| Borrowed id | Char | 16 | It’s the reservation id that is emailed | aaaaaaaaaaaaaaaa | Definitely 16 characters |
| takenoutdate | Date | 1 | Gives you the date it was originally reserved/taken out | R |  |
| Musicinstance\_id | integer |  | Foreign Key to music instance table | 01 |  |
| borrower\_id | Integer |  | Foreign Key to user table | 01 |  |
| Collected | Bool | 1 | Checks that the reservation was picked up | T |  |
| Returned | Bool | 1 | Checks that it has been returned |  |  |

## Database Entity Relationship Diagram

This image shows my original idea for my database



## SQLite3

This is the SQLite that creates my tables in their original form

## Auth\_Group

CREATE TABLE "auth\_group" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"name" varchar(80) NOT NULL UNIQUE

);

## auth\_group\_permissions

CREATE TABLE "auth\_group\_permissions" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"group\_id" integer NOT NULL,

"permission\_id" integer NOT NULL,

FOREIGN KEY("group\_id") REFERENCES "auth\_group"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("permission\_id") REFERENCES "auth\_permission"("id") DEFERRABLE INITIALLY DEFERRED

);

## auth\_permissions

CREATE TABLE "auth\_permission" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"content\_type\_id" integer NOT NULL,

"codename" varchar(100) NOT NULL,

"name" varchar(255) NOT NULL,

FOREIGN KEY("content\_type\_id") REFERENCES "django\_content\_type"("id") DEFERRABLE INITIALLY DEFERRED

);

## auth\_user

CREATE TABLE "auth\_user" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"password" varchar(128) NOT NULL,

"last\_login" datetime,

"is\_superuser" bool NOT NULL,

"username" varchar(150) NOT NULL UNIQUE,

"first\_name" varchar(30) NOT NULL,

"email" varchar(254) NOT NULL,

"is\_staff" bool NOT NULL,

"is\_active" bool NOT NULL,

"date\_joined" datetime NOT NULL,

"last\_name" varchar(150) NOT NULL

);

## auth\_user\_groups

CREATE TABLE "auth\_user\_groups" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"user\_id" integer NOT NULL,

"group\_id" integer NOT NULL,

FOREIGN KEY("group\_id") REFERENCES "auth\_group"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("user\_id") REFERENCES "auth\_user"("id") DEFERRABLE INITIALLY DEFERRED

);

## auth\_user\_user\_permissions

CREATE TABLE "auth\_user\_user\_permissions" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"user\_id" integer NOT NULL,

"permission\_id" integer NOT NULL,

FOREIGN KEY("permission\_id") REFERENCES "auth\_permission"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("user\_id") REFERENCES "auth\_user"("id") DEFERRABLE INITIALLY DEFERRED

);

## catalog\_borrowed

CREATE TABLE "catalog\_borrowed" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"borrowedid" INTEGER NOT NULL,

"takenoutdate" datetime NOT NULL,

"musicInstance\_id" char(32),

"borrower\_id" INTEGER,

"Collected" INTEGER DEFAULT 0,

"Returned" INTEGER DEFAULT 0,

FOREIGN KEY("borrower\_id") REFERENCES "auth\_user"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("musicInstance\_id") REFERENCES "catalog\_musicinstance"("id") DEFERRABLE INITIALLY DEFERRED

);

## catalog\_composer

CREATE TABLE "catalog\_composer" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"first\_name" varchar(100) NOT NULL,

"last\_name" varchar(100) NOT NULL,

"date\_of\_birth" date,

"date\_of\_death" date

);

## catalog\_genre

CREATE TABLE "catalog\_genre" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"name" varchar(200) NOT NULL

);

## catalog\_language

CREATE TABLE "catalog\_genre" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"name" varchar(200) NOT NULL

);

## catalog\_music

CREATE TABLE "catalog\_music" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"title" varchar(200) NOT NULL,

"summary" text NOT NULL,

"isbn" varchar(13) NOT NULL,

"composer\_id" integer,

"language\_id" integer,

FOREIGN KEY("composer\_id") REFERENCES "catalog\_composer"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("language\_id") REFERENCES "catalog\_language"("id") DEFERRABLE INITIALLY DEFERRED

);

## catalog\_music\_genre

CREATE TABLE "catalog\_music\_genre" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"music\_id" integer NOT NULL,

"genre\_id" integer NOT NULL,

FOREIGN KEY("music\_id") REFERENCES "catalog\_music"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("genre\_id") REFERENCES "catalog\_genre"("id") DEFERRABLE INITIALLY DEFERRED

);

## catalog\_musicinstance

CREATE TABLE "catalog\_musicinstance" (

"id" char(32) NOT NULL,

"due\_back" date,

"status" varchar(1) NOT NULL,

"borrower\_id" integer,

"music\_id" integer,

PRIMARY KEY("id"),

FOREIGN KEY("borrower\_id") REFERENCES "auth\_user"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("music\_id") REFERENCES "catalog\_music"("id") DEFERRABLE INITIALLY DEFERRED

);

## django\_admin\_log

CREATE TABLE "django\_admin\_log" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"action\_time" datetime NOT NULL,

"object\_id" text,

"object\_repr" varchar(200) NOT NULL,

"change\_message" text NOT NULL,

"content\_type\_id" integer,

"user\_id" integer NOT NULL,

"action\_flag" smallint unsigned NOT NULL,

FOREIGN KEY("content\_type\_id") REFERENCES "django\_content\_type"("id") DEFERRABLE INITIALLY DEFERRED,

FOREIGN KEY("user\_id") REFERENCES "auth\_user"("id") DEFERRABLE INITIALLY DEFERRED

);

## django\_content\_type

CREATE TABLE "django\_content\_type" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"app\_label" varchar(100) NOT NULL,

"model" varchar(100) NOT NULL

);

## django\_migrations

CREATE TABLE "django\_migrations" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"app" varchar(255) NOT NULL,

"name" varchar(255) NOT NULL,

"applied" datetime NOT NULL

);

## django\_session

CREATE TABLE "django\_session" (

"session\_key" varchar(40) NOT NULL,

"session\_data" text NOT NULL,

"expire\_date" datetime NOT NULL,

PRIMARY KEY("session\_key")

);

## sqlite\_sequence

CREATE TABLE "sqlite\_sequence" (

"name" TEXT,

"seq" TEXT

);

## Planned SQLite Queries

With Django I create queries and it will translate them into sqlite queries.

I only have a few queries and they mostly look like this:

*def get\_queryset(self):*

*return MusicInstance.objects.filter(borrower=self.request.user).filter(status\_\_exact='o').order\_by('due\_back')*

*This grabs the musicinstances, filters by user, filters by onloan and then orders by when they are due back.*

All of them just change whether the borrower is the user or all and the status.

# Class Diagrams

These show how different classes are linked to each other in each part of my code.

## Admin

A screenshot of a cell phone screen with text

Description automatically generated

## Forms

A screenshot of a cell phone

Description automatically generated

## A screenshot of a computer screen Description automatically generatedModels

# Main Algorithms

## Training()

This takes the information from the json file and shortens the questions down to the main words. It then will learn these questions and link them to the tag. It then links the tags with the answers. It will then save all of this information into 2 files. 1 pickle (serialized file) and 1 tflearn file for the neural network to use. This part at the moment will take seconds but as you add more questions and answers will take longer. This also can’t learn new questions and answers by itself it will have to email me to receive an appropriate answer.

Here is the code:

*words = []*

*labels = []*

*docs\_x = []*

*docs\_y = []*

*for intent in data["intents"]:*

*for pattern in intent["patterns"]:*

*wrds = nltk.word\_tokenize(pattern)*

*words.extend(wrds)*

*docs\_x.append(wrds)*

*docs\_y.append(intent["tag"])*

*if intent["tag"] not in labels:*

*labels.append(intent["tag"])*

*words = [stemmer.stem(w.lower()) for w in words if w != "?"]*

*words = sorted(list(set(words)))*

*labels = sorted(labels)*

*training = []*

*output = []*

*out\_empty = [0 for \_ in range(len(labels))]*

*for x, doc in enumerate(docs\_x):*

*bag = []*

*wrds = [stemmer.stem(w.lower()) for w in doc]*

*for w in words:*

*if w in wrds:*

*bag.append(1)*

*else:*

*bag.append(0)*

*output\_row = out\_empty[:]*

*output\_row[labels.index(docs\_y[x])] = 1*

*training.append(bag)*

*output.append(output\_row)*

*training = numpy.array(training)*

*output = numpy.array(output)*

*with open("data.pickle", "wb") as f:*

*pickle.dump((words, labels, training, output), f)*

## Chat()

This takes the model and uses it to chat with the user. It checks the questions and checks if it knows any similar questions above 90% accuracy. If it knows the question then it will respond with a random answer from the tag. If not it will tell the user that it doesn’t understand the question and then will send me (the admin) the question to add to the json file.

Here is the code:

It works by creating a list of main words associated to questions and answers.

It then searches through this each time and looks for how similar it is.

It then will either print out a random answer from the list or tell you that it doesn’t know what is going on.

*def bag\_of\_words(s, words):*

*bag = [0 for \_ in range(len(words))]*

*s\_words = nltk.word\_tokenize(s)*

*s\_words = [stemmer.stem(word.lower()) for word in s\_words]*

*for se in s\_words:*

*for i, w in enumerate(words):*

*if w == se:*

*bag[i] = 1*

*return numpy.array(bag)*

*def chat():*

*print("Start talking with the bot (type quit to stop)!")*

*while True:*

*inp = input("You: ")*

*if inp.lower() == "quit":*

*break*

*results = model.predict([bag\_of\_words(inp, words)])*

*results\_index = numpy.argmax(results)*

*tag = labels[results\_index]*

*for tg in data["intents"]:*

*if tg['tag'] == tag:*

*responses = tg['responses']*

*print(random.choice(responses))*

## Suggestions()

This will work by looking at ratings of music you have and look for the highest rated ones. Then it will look at people who have given similar ratings to this music and look for their highest rated music and give it to you. Eventually it will use a better AI system but I haven’t got the time to learn this in the time. I won’t be able to implement the piece of code until I have enough data from my users which will take a few years.

Here is a piece of code similar to what I want to do.

This was similar but not exactly what I did. Below it is my final version

import numpy as np

import pandas as pd

ratings\_data = pd.read\_csv("ratings.csv")

ratings\_data.head()

movie\_names=pd.read\_csv("movies.csv")

movie\_names.head()

movie\_data=pd.merge(ratings\_data, movie\_names, on='movieId')

movie\_data.head()

movie\_data.groupby('title')['rating'].mean().sort\_values(ascending=False).head()

movie\_data.groupby('title')['rating'].count().sort\_values(ascending=False).head()

ratings\_mean\_count = pd.DataFrame(movie\_data.groupby('title')['rating'].mean())

ratings\_mean\_count['rating\_counts'] = pd.DataFrame(movie\_data.groupby('title')['rating'].count())

ratings\_mean\_count.head()

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('dark')

%matplotlib inline

plt.figure(figsize=(8,6))

plt.rcParams['patch.force\_edgecolor'] = True

sns.jointplot[x='rating', y='rating\_counts', data=ratings\_mean\_count, alpha=0.4]

user\_movie\_rating = movie\_data.pivot\_table(index='userId', columns='title', values='rating')

user\_movie\_rating.head()

This uses a few CSV files but I am hoping to eventually use my database system to implement it. It might take some time so I will try and change these csvs into just one file that is normalised.

@staticmethod

def suggestionsForUser(user):

latestGoodReviews = Review.objects.filter(user=user).filter(rating\_\_gte=6).order\_by('-reviewDate')

numberOfGoodReviews = latestGoodReviews.count()

if numberOfGoodReviews == 0:

return []

numberOfCandidates = 0

compatibleUsers = set()

for goodReview in latestGoodReviews:

otherReviews = Review.objects.filter(music = goodReview.music).filter(rating\_\_gte=goodReview.rating - 1).filter(rating\_\_lte = goodReview.rating + 1)

for otherReview in otherReviews:

if otherReview.user == user:

continue

compatibleUsers.add(otherReview.user)

itemDict = {}

for user in compatibleUsers:

positiveReviews = Review.objects.filter(user=user).filter(rating\_\_gte = 7)

for positiveReview in positiveReviews:

if MusicInstanceReservation.objects.filter(musicInstance\_\_music=positiveReview.music,userid=user).exists():

continue

currentVal = itemDict.get(positiveReview.music.id)

if (currentVal == None):

currentVal = 1

else:

currentVal = currentVal + 1

itemDict[positiveReview.music] = currentVal

nSuggestions = 0

suggestions = []

for k, v in sorted(itemDict.items(), key=lambda item: item[1],reverse = True):

nSuggestions += 1

if (nSuggestions > 4):

break

suggestions.append(k)

return suggestions

# Data Structures

I only have 1 main type of data structure. For the chatbot I am using dictionaries as it makes it easier for the neural network to learn what questions are linked to what answers to make it easier to output answers.

Here it is:

*{"intents": [*

*{"tag": "greeting",*

*"patterns": ["Hi", "Is anyone there?", "Hello", "Good day", "Whats up"],*

*"responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"],*

*"context\_set": ""*

*},*

*{"tag": "goodbye",*

*"patterns": ["cya", "See you later", "Goodbye", "I am Leaving", "Have a Good day"],*

*"responses": ["Sad to see you go :(", "Talk to you later", "Goodbye!"],*

*"context\_set": ""*

*},*

*{"tag": "age",*

*"patterns": ["how old", "how old is tim", "what is your age", "how old are you", "age?"],*

*"responses": ["I am 18 years old!", "18 years young!"],*

*"context\_set": ""*

*},*

*{"tag": "name",*

*"patterns": ["what is your name", "what should I call you", "whats your name?"],*

*"responses": ["You can call me Postel.", "I'm Jon Postel but you can call me Postel!", "I'm Postel aka The "God" of the Internet."],*

*"context\_set": ""*

*},*

*{"tag": "work",*

*"patterns": ["How does it work", "How do you work", "What can I do?"],*

*"responses": ["I have a set of preprogrammed questions that I have learnt! If I don't know an answer I ask my creator.", "You can just ask me questions."],*

*"context\_set": ""*

*},*

*{"tag": "robot",*

*"patterns": ["are you a bot", "are you a chatbot", "are you real"],*

*"responses": ["I am a chatbot hear to help you. I can answer some questions and if I don't know it I will email the admin and they will get back to you with a response"],*

*"context\_set": ""*

*},*

*{"tag": "emotions",*

*"patterns": ["how's yur day been", "how are you", "how you doing", "whats up"],*

*"responses": ["I'm good", "I'm freezing start playing some games with me. ;}", "Sad as noone plays games with me :(", "Happy to see a new face. :)"],*

*"context\_set":""*

*}*

*]*

*}*

This is just a general AI at the moment but once I have spoken to my client I will update it to make it more specific to the system.

# Planned Security Measures

Client-side, the information stored on the user’s side is just their username and password and only if they want it to. Cookies are unavoidable. The web browser used will store the information in a secure way.

Server side, I will be using Django security which hashes password before putting them in the database. Django also stops SQL injection by taking whatever the user enters and turning it into a query of its own.

# 3-Technical Solution

My system was programmed in python, JavaScript and html. I am running the website on digital ocean using a web server. My chatbot links into discord for the viewing of this and this is hosted on their own webpage.

Here is a list of techniques I used in my implementation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Technique/Algorithm | Bands | Type of Script | Name | Page | Description |
| Dictionaries | A | JavaScript | intents.json | 5 | Used to teach the chatbot |
| Parsing json | B | Python | chatbot.py | 2 | Bringing in the questions and answers from the json dictionaries to teach the chatbot |
| Client-server model | A | Python | views.py |  | This is used to call information straight from my database into the website to be used later on |
| Complex Data Model | A | Python/SQLite3 | models.py | 6 | This is where I create all my tables with multiple crossovers |
| Neural Networks | A | Python | chatbot.py | 3 | This is what actually grabs the information from the parsing, splits it into words and then teaches the chatbot |
| File Reading and Writing | B | Python | chatbot.py | 4 | Saving the model |
| API calls | C | Python | index.html | 23 | This is where I call an API for discord |
| Reservations |  | Python | ReserveAction | 14 | This is where the actual action of reserving a piece of music occurs and saves to database |
| Borrowing |  | Python | BorrowAction | 15 | This is where the actual action of borrowing a piece of music occurs and saves to database |
| Returning |  | Python | ReturnAction | 16 | This is where the actual action of returning a piece of music occurs and saves to database |
| Renewal |  | Python | forms.py | 17 | This is where I do my renewal of music |

## Chatbot Code

I created my chatbot originally as a terminal based chat system and then realised that it needed implementing into my website. This meant trying to find an API to easily implement this. From looking at different chat systems, the easiest way of doing it was using a discord server with my bot installed on it. It would then be called from a html page as an API to look good on my webpage. Th

This is what the code looks like:

import nltk

from nltk.stem.lancaster import LancasterStemmer

stemmer = LancasterStemmer()

import numpy

import tflearn

import tensorflow

import random

import json

import pickle

This is parsing my json file

with open("intents.json") as file:

data = json.load(file)

words = []

labels = []

docs\_x = []

docs\_y = []

for intent in data["intents"]:

for pattern in intent["patterns"]:

wrds = nltk.word\_tokenize(pattern)

words.extend(wrds)

docs\_x.append(wrds)

docs\_y.append(intent["tag"])

if intent["tag"] not in labels:

labels.append(intent["tag"])

words = [stemmer.stem(w.lower()) for w in words if w != "?"]

words = sorted(list(set(words)))

This is sorting my labels. I could have created my own sorting algorithm but for what I needed it was just easier to allow python to do it. It also works quicker.

labels = sorted(labels)

training = []

output = []

out\_empty = [0 for \_ in range(len(labels))]

for x, doc in enumerate(docs\_x):

bag = []

This makes the words shorter to store into my page

wrds = [stemmer.stem(w.lower()) for w in doc]

for w in words:

if w in wrds:

bag.append(1)

else:

bag.append(0)

output\_row = out\_empty[:]

output\_row[labels.index(docs\_y[x])] = 1

training.append(bag)

output.append(output\_row)

training = numpy.array(training)

output = numpy.array(output)

with open("data.pickle", "wb") as f:

pickle.dump((words, labels, training, output), f)

This creates a clean diagram for the neural network to work on

tensorflow.reset\_default\_graph()

net = tflearn.input\_data(shape=[None, len(training[0])])

net = tflearn.fully\_connected(net, 8)

net = tflearn.fully\_connected(net, 8)

net = tflearn.fully\_connected(net, len(output[0]), activation="softmax")

net = tflearn.regression(net)

model = tflearn.DNN(net)

model.fit(training, output, n\_epoch=1000, batch\_size=8, show\_metric=True)

model.save("model.tflearn")

def bag\_of\_words(s, words):

bag = [0 for \_ in range(len(words))]

s\_words = nltk.word\_tokenize(s)

s\_words = [stemmer.stem(word.lower()) for word in s\_words]

for se in s\_words:

for i, w in enumerate(words):

if w == se:

bag[i] = 1

return numpy.array(bag)

def chat(x):

results = model.predict([bag\_of\_words(x, words)])

results\_index = numpy.argmax(results)

tag = labels[results\_index]

for tg in data["intents"]:

if tg['tag'] == tag:

responses = tg['responses']

return random.choice(responses)

import discord

client = discord.Client()

This is an event so looks for when someone is typing a question and then it will respond

@client.event

async def on\_message(message):

if message.author.id == client.user.id:

This makes sure the bot doesn’t respond to itself

return

x = message.content

y= chat(x)

await message.channel.send(y)

keep\_alive()

This is running it in a flask app that just runs a small websocket just for the chatbot and keeps it running all the time

This just runs my chatbot inside of discord

client.run("NjQyMDY3Nzg4NzY0MTUxODA5.XdlGnA.Ydtxl4Q7\_z1v9rIiM4curuvnNdc")

### The dictionary this is calling from

These are just all the questions and answers for a general chatbot. I will be editing this when it goes live

{"intents": [

{"tag": "greeting",

"patterns": ["Hi", "Is anyone there?", "Hello", "Good day", "Whats up"],

"responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"],

"context\_set": ""

},

{"tag": "goodbye",

"patterns": ["cya", "See you later", "Goodbye", "I am Leaving", "Have a Good day"],

"responses": ["Sad to see you go :(", "Talk to you later", "Goodbye!"],

"context\_set": ""

},

{"tag": "age",

"patterns": ["how old", "how old is tim", "what is your age", "how old are you", "age?"],

"responses": ["I am 18 years old!", "18 years young!"],

"context\_set": ""

},

{"tag": "name",

"patterns": ["what is your name", "what should I call you", "whats your name?"],

"responses": ["You can call me Postel.", "I'm Jon Postel but you can call me Postel!", "I'm Postel aka The God of the Internet."],

"context\_set": ""

},

{"tag": "work",

"patterns": ["How does it work", "How do you work", "What can I do?"],

"responses": ["I have a set of preprogrammed questions that I have learnt! If I don't know an answer I ask my creator.", "You can just ask me questions."],

"context\_set": ""

},

{"tag": "robot",

"patterns": ["are you a bot", "are you a chatbot", "are you real"],

"responses": ["I am a chatbot hear to help you. I can answer some questions and if I don't know it I will email the admin and they will get back to you with a response"],

"context\_set": ""

},

{"tag": "emotions",

"patterns": ["how's yur day been", "how are you", "how you doing", "whats up"],

"responses": ["I'm good", "I'm freezing start playing some games with me. ;}", "Sad as noone plays games with me :(", "Happy to see a new face. :)"],

"context\_set":""

}

]

}

# Models for Database

from django.db import models

from django.core.mail import send\_mail

from django.utils.timezone import now

# Create your models here.

from django.utils import timezone

from django.urls import reverse # To generate URLS by reversing URL patterns

from django.contrib.auth.models import User

import django\_filters

from django.utils.crypto import get\_random\_string

import uuid # Required for unique music instances

from datetime import date,timedelta

daysToReserve = 14

daysToBorrow = 122

class Genre(models.Model):

"""Model representing a musical genre (e.g. Jazz, Classical, Pop)."""

name = models.CharField(

max\_length=200,

help\_text="Enter a musical genre (e.g. Jazz, Classical, Pop)"

)

def \_\_str\_\_(self):

"""String for representing the Model object (in Admin site etc.)"""

return self.name

class Language(models.Model):

"""Model representing a Language (e.g. English, French, Japanese, etc.)"""

name = models.CharField(max\_length=200,

help\_text="Enter the music's natural language (e.g. English, French, Japanese etc.)")

def \_\_str\_\_(self):

"""String for representing the Model object (in Admin site etc.)"""

return self.name

class Composer(models.Model):

atomic = False

"""Model representing an author."""

first\_name = models.CharField(max\_length=100)

last\_name = models.CharField(max\_length=100)

date\_of\_birth = models.DateField(null=True, blank=True)

date\_of\_death = models.DateField('died', null=True, blank=True)

class Meta:

ordering = ['last\_name', 'first\_name']

def get\_absolute\_url(self):

"""Returns the url to access a particular author instance."""

return reverse('composer\_detail', args=[str(self.id)])

def \_\_str\_\_(self):

"""String for representing the Model object."""

return '{0}, {1}'.format(self.last\_name, self.first\_name)

class Music(models.Model):

atomic = False

"""Model representing a piece of music (but not a specific copy of that music)."""

title = models.CharField(max\_length=200)

composer = models.ForeignKey(Composer, on\_delete=models.SET\_NULL, null=True)

# Foreign Key used because music can only have one composer but composers can have multiple sets of music

# Composer as a string rather than object because it hasn't been declared yet in file.

summary = models.TextField(max\_length=1000, help\_text="Enter a brief description of the book")

barcode = models.CharField('barcode', max\_length=13,

help\_text='the library unique reference code for this piece')

genre = models.ManyToManyField(Genre, help\_text="Select a genre for this music")

# ManyToManyField used because a genre can contain many sets of music and a Music can cover many genres.

# Genre class has already been defined so we can specify the object above.

language = models.ForeignKey(Language, on\_delete=models.SET\_NULL, null=True)

def display\_genre(self):

"""Creates a string for the Genre. This is required to display genre in Admin."""

return ', '.join([genre.name for genre in self.genre.all()[:3]])

display\_genre.short\_description = 'Genre'

def get\_absolute\_url(self):

"""Returns the url to access a particular book instance."""

return reverse('music-detail', args=[str(self.id)])

def \_\_str\_\_(self):

"""String for representing the Model object."""

return self.title

class Meta:

permissions = (

("can\_browse\_catalog", "Can see what the music library has to offer"),

("can\_see\_availability", "Can see whether a piece is available"),

("can\_self\_reserve", "Can make a reservation for themself"),

("can\_any\_reserve", "Can make reservations on behalf of other users"),

("can\_issue", "Can set reservation as borrowed"),

("can\_return", "Can mark borrowed music as now returned"),

("can\_see\_reservation\_detail", "Can see who has reserved or borrowed an instance"),

("is\_admin", "Can see admin pages"),

)

from django.contrib.auth.models import User # Required to assign User as a borrower

class MusicInstance(models.Model):

atomic = False

"""Model representing a specific copy of a book (i.e. that can be borrowed from the library)."""

id = models.UUIDField(primary\_key=True, default=uuid.uuid4,

help\_text="Unique ID for this particular book across whole library")

music = models.ForeignKey(Music, on\_delete=models.SET\_NULL, null=True)

due\_back = models.DateField(null=True, blank=True)

borrower = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True)

def reserve(self,user,\*\*kwargs):

instance = self

now = kwargs.get('dateOverride') # to use when creating test data

if now == None:

now = timezone.now()

print("Making a reservation as at: %s" % (str(now)))

reservationnumber = get\_random\_string(length=6, allowed\_chars='1234567890')

reservationnumber = int(reservationnumber)

instance.status = 'r'

instance.due\_back = now + timedelta(days=daysToReserve)

instance.borrower = user

instance.save()

reservation = MusicInstanceReservation(borrowedid = reservationnumber, musicInstance=instance , duedate = instance.due\_back, takenoutdate = now, userid=user)

activity = ActivityLog(activityCode = 'res', music=instance.music,musicInstance=instance,composer=instance.music.composer,user=user)

activity.save()

reservation.save()

return reservationnumber,instance

@property

def is\_overdue(self):

if self.due\_back and date.today() > self.due\_back:

send\_mail(

'Music overdue',

'Your Music is overdue',

'adam@Bilkus.com',

['adam@Bilkus.com'],

fail\_silently = False,

)

print("hELLO")

return True

return False

LOAN\_STATUS = (

('d', 'Maintenance'),

('o', 'On loan'),

('a', 'Available'),

('r', 'Reserved'),

)

status = models.CharField(

max\_length=1,

choices=LOAN\_STATUS,

blank=True,

default='a',

help\_text='Music availability')

class Meta:

ordering = ['due\_back']

permissions = (("can\_mark\_returned", "Set book as returned"),)

def \_\_str\_\_(self):

"""String for representing the Model object."""

return '{0} ({1} {2})'.format(self.id, self.music.title, self.music.composer.last\_name)

class MusicInstanceReservation(models.Model):

borrowedid = models.IntegerField()

musicInstance = models.ForeignKey(MusicInstance, on\_delete=models.SET\_NULL, null=True)

userid= models.ForeignKey(User, null=True, on\_delete=models.SET\_NULL)

takenoutdate=models.DateTimeField(null=True,blank=True)

returneddate=models.DateTimeField(null=True,blank=True)

duedate = models.DateTimeField(null=True,blank=True)

returned = models.BooleanField(default=False)

takenout = models.BooleanField(default=False)

cancelled = models.BooleanField(default=False)

def save(self,\*args,\*\*kwargs):

if self.duedate == None:

self.duedate = timezone.now + timedelta(days=daysToReserve)

if self.takenoutdate == None:

self.takenoutdate = timezone.now()

return super(MusicInstanceReservation,self).save(\*args,\*\*kwargs)

def cancel(self,user,\*\*kwargs):

instance = self.musicInstance

instance.status = 'a'

instance.due\_back = None

instance.borrower = None

instance.save()

reservation = self

reservation.cancelled = True

reservation.save()

activity = ActivityLog(activityCode = 'can', music=instance.music,musicInstance=instance,composer=instance.music.composer,user=user)

activity.save()

def borrow(self,user,\*\*kwargs):

reservation = self

instance = self.musicInstance

instance.status = 'o'

instance.due\_back = date.today() + timedelta(days = daysToBorrow)

instance.save()

reservation.due\_back = instance.due\_back

reservation.takenout = True

reservation.takenoutdate = date.today()

reservation.returned = False

reservation.save()

activity = ActivityLog(activityCode = 'bor', music=instance.music,musicInstance=instance,composer=instance.music.composer,user=user)

activity.save()

return instance

def renew(self,user,\*\*kwargs):

reservation = self

instance = self.musicInstance

instance.due\_back = date.today() + timedelta(days = daysToBorrow)

instance.save()

activity = ActivityLog(activityCode = 'ren', music=instance.music,musicInstance=instance,composer=instance.music.composer,user=user)

activity.save()

reservation.duedate = instance.due\_back

def returns(self,user,\*\*kwargs):

reservation = self

instance = self.musicInstance

instance.status = 'a'

instance.due\_back = None

instance.borrower = None

instance.save()

reservation.returned = True

reservation.returneddate = date.today()

reservation.save()

activity = ActivityLog(activityCode = 'ret', music=instance.music,musicInstance=instance,composer=instance.music.composer,user=user)

activity.save()

return reservation.userid

def hasExpired(self):

return not self.takenout and not self.returned and not self.cancelled and (self.duedate < timezone.now())

@staticmethod

def cancelExpiredReservations(user):

for res in MusicInstanceReservation.objects.all():

if res.hasExpired():

print("Cancelling an expired reservation")

res.cancel(user)

ACTIVITY\_CODE = (

('res', 'Reserve'),

('bor', 'Borrow'),

('can', 'Cancel Res'),

('ret', 'Return'),

('ren', 'Renew'),

)

class ActivityLog(models.Model):

activityTimestamp = models.DateTimeField(default=timezone.now)

activityCode = models.CharField(

max\_length=4,

choices=ACTIVITY\_CODE,

null=False,

help\_text='The activity being logged')

music = models.ForeignKey(Music, on\_delete=models.SET\_NULL, null=True)

musicInstance = models.ForeignKey(MusicInstance, on\_delete=models.SET\_NULL, null=True)

musicInstanceReservation = models.ForeignKey(MusicInstanceReservation, on\_delete=models.SET\_NULL, null=True)

composer = models.ForeignKey(Composer, on\_delete=models.SET\_NULL, null=True)

user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)

class Review(models.Model):

music = models.ForeignKey(Music, on\_delete=models.SET\_NULL, null=True)

user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)

rating = models.IntegerField()

reviewDate = models.DateTimeField(default=timezone.now)

@staticmethod

def suggestionsForUser(user):

latestGoodReviews = Review.objects.filter(user=user).filter(rating\_\_gte=6).order\_by('-reviewDate')

numberOfGoodReviews = latestGoodReviews.count()

if numberOfGoodReviews == 0:

return []

numberOfCandidates = 0

compatibleUsers = set()

for goodReview in latestGoodReviews:

otherReviews = Review.objects.filter(music = goodReview.music).filter(rating\_\_gte=goodReview.rating - 1).filter(rating\_\_lte = goodReview.rating + 1)

for otherReview in otherReviews:

if otherReview.user == user:

continue

compatibleUsers.add(otherReview.user)

itemDict = {}

for user in compatibleUsers:

positiveReviews = Review.objects.filter(user=user).filter(rating\_\_gte = 7)

for positiveReview in positiveReviews:

if MusicInstanceReservation.objects.filter(musicInstance\_\_music=positiveReview.music,userid=user).exists():

continue

currentVal = itemDict.get(positiveReview.music.id)

if (currentVal == None):

currentVal = 1

else:

currentVal = currentVal + 1

itemDict[positiveReview.music] = currentVal

nSuggestions = 0

suggestions = []

for k, v in sorted(itemDict.items(), key=lambda item: item[1],reverse = True):

nSuggestions += 1

if (nSuggestions > 4):

break

suggestions.append(k)

return suggestions

In this set of code I am laying out how my tables should look in Django form but they are very similar to SQLite3. Django then uses the models function to turn these into SQLite3 form. You then migrate this to allow it to create the tables.

## Client-Server Model

import random

import datetime

from datetime import date,timedelta

import time

from django.shortcuts import render

from django.views import View

from django.http import HttpResponse

from django.template import loader

from django.db.models import Exists, OuterRef, Q, Count

from django.db.models.functions import Lower

from django.core.mail import send\_mail

from django.contrib.auth.models import User

from django.contrib.auth.decorators import permission\_required

from django.contrib.auth.mixins import PermissionRequiredMixin,AccessMixin

from django.contrib.auth import get\_user\_model

from django.views import generic

from django.views.generic.base import TemplateView

from django.contrib.auth.mixins import LoginRequiredMixin

from django.shortcuts import get\_object\_or\_404

from django.http import HttpResponseRedirect,HttpResponseForbidden

from django.urls import reverse

from django.contrib import messages

import django\_filters

from django.views.generic.edit import CreateView, UpdateView, DeleteView, FormView

from django.urls import reverse\_lazy

from django.shortcuts import render

from django.core.exceptions import PermissionDenied

from django.contrib.auth.models import Group

from django.utils.decorators import method\_decorator

from django\_ajax.decorators import ajax

from django\_ajax.mixin import AJAXMixin

from catalog.forms import \*

from catalog.models import Music, Composer, MusicInstance, Genre, MusicInstanceReservation,ActivityLog,Review

from pprint import pprint

#from rest\_framework import serializers

def is\_in\_group(user,group\_name):

group = Group.objects.get(name=group\_name)

return True if group in user.groups.all() else False

# The home page will depend on the logged in user and which group they belong to

class HomePageView(TemplateView):

def get\_template\_names(self):

if not self.request.user.is\_authenticated:

return "visitorindex.html"

if is\_in\_group(self.request.user,"Nonmember"):

return "nonmemberindex.html"

if is\_in\_group(self.request.user,"Librarian"):

return "librarianindex.html"

if is\_in\_group(self.request.user,"Member"):

return "memberindex.html"

return super().get\_template\_names()

template\_name = 'index.html'

# calendarStartDate event\_list

'''

{

title: 'All Day Event',

start: '2020-02-01'

},

{

title: 'Long Event',

start: '2020-02-07',

end: '2020-02-10'

},

{

groupId: '999',

title: 'Repeating Event',

start: '2020-02-09T16:00:00'

},

{

groupId: '999',

title: 'Repeating Event',

start: '2020-02-16T16:00:00'

},

{

title: 'Conference',

start: '2020-02-11',

end: '2020-02-13'

},

{

title: 'Meeting',

start: '2020-02-12T10:30:00',

end: '2020-02-12T12:30:00'

},

{

title: 'Lunch',

start: '2020-02-12T12:00:00'

},

{

title: 'Meeting',

start: '2020-02-12T14:30:00'

},

{

title: 'Birthday Party',

start: '2020-02-13T07:00:00'

},

{

title: 'Click for Google',

url: 'http://google.com/',

start: '2020-02-28'

}

'''

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

user = None

# Generate counts of some of the main objects

num\_music = Music.objects.all().count()

num\_instances = MusicInstance.objects.all().count()

# Available copies of books

num\_instances\_available = MusicInstance.objects.filter(status\_\_exact='a').count()

num\_composers = Composer.objects.count() # The 'all()' is implied by default.

# Number of visits to this view, as counted in the session variable.

num\_visits = self.request.session.get('num\_visits', 0)

self.request.session['num\_visits'] = num\_visits+1

can\_reserve = False

if self.request.user.has\_perm('catalog.can\_issue'):

can\_reserve = True

xxx = (

{'can\_reserve':can\_reserve ,'num\_music': num\_music, 'num\_instances': num\_instances,

'num\_instances\_available': num\_instances\_available, 'num\_composers': num\_composers,

'num\_visits': num\_visits})

context.update(xxx)

context['calendarStartDate'] = date.today().strftime("%Y-%m-%d")

statusq = Q(status\_\_exact = 'r') | Q(status\_\_exact = 'o')

if self.request.user.has\_perm('catalog.can\_any\_reserve'):

instances = MusicInstance.objects.filter(statusq)

else:

instances = MusicInstance.objects.filter(statusq, borrower\_id = self.request.user.id)

events = []

for event in instances:

eventtext = '{title:" ' + str(event.music.title) + '\\n user: ' + str(event.borrower) + '",start:"' + event.due\_back.strftime("%Y-%m-%d") + '"},'

events.append(eventtext)

context['event\_list'] = events

return context

def render\_to\_response(self,context,\*\*kwargs):

return super().render\_to\_response(context);

# These next views are the various ways of seeing our data

class MusicListView(PermissionRequiredMixin,generic.ListView):

"""Generic class-based view for a list of music."""

model = Music

paginate\_by = 10

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

return context

def has\_permission(self):

if not self.request.user.is\_authenticated:

print("musiclistview not authenticated")

return False

if not self.request.user.has\_perm('catalog.can\_browse\_catalog'):

print("musiclistview user lacks can\_browse\_catalog")

return False

return True

class MusicListGridView(PermissionRequiredMixin,TemplateView):

"""Generic class-based view for a list of music."""

template\_name = "catalog/music\_list\_grid.html"

def has\_permission(self):

if not self.request.user.is\_authenticated:

print("musiclistview not authenticated")

return False

if not self.request.user.has\_perm('catalog.can\_browse\_catalog'):

print("musiclistview user lacks can\_browse\_catalog")

return False

return True

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

#queryjson = serializers.serialize('json',music.objects().all())

x = list(Music.objects.values('id','title','composer\_\_last\_name','genre\_\_name','language\_\_name').order\_by(Lower('composer\_\_last\_name')))

queryjson = json.dumps(x)

context['queryjson'] = queryjson

return context

class MusicDetailView(PermissionRequiredMixin,generic.DetailView):

"""Generic class-based detail view for a book."""

model = Music

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_browse\_catalog'):

return False

return True

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

music=kwargs['object']

print(music)

available=music.musicinstance\_set.filter(status\_\_exact = 'a')

navailable = available.count()

context['music'] = music

context['firstavailable'] = available.first()

context['navailable'] = navailable

context['show\_reserve\_button'] = navailable > 0 and (self.request.user.has\_perm('catalog.can\_self\_reserve') or self.request.user.has\_perm('catalog.can\_any\_reserve'))

context['form'] = GetUserForm(initial={'user': self.request.user})

return context

# Clean up composer just like we did for music

class ComposerListView(PermissionRequiredMixin,generic.ListView):

"""Generic class-based list view for a list of authors."""

model = Composer

paginate\_by = 10

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_browse\_catalog'):

return False

return True

class ComposerDetailView(generic.DetailView):

"""Generic class-based detail view for a composer """

model = Composer

class BorrowedOrReservedByUser(PermissionRequiredMixin, generic.ListView):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_self\_reserve'):

return False

return True

template\_name = "catalog/borrowed\_or\_reserved\_by\_user.html"

context\_object\_name = 'instances'

paginate\_by = 10

def get\_queryset(self, \*\*kwargs):

statusq = Q(status\_\_exact = 'r') | Q(status\_\_exact = 'o')

instances = MusicInstance.objects.filter(statusq, borrower\_id = self.request.user.id)

'''for i in instances:

if instance.statusq == 'r' and (date.today() - instance.due\_back) < 0:

'''

return instances

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

return context

class BorrowedOrReservedByAll(PermissionRequiredMixin, generic.ListView):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_any\_reserve'):

return False

return True

template\_name = "catalog/borrowed\_or\_reserved\_by\_all.html"

context\_object\_name = 'instances'

paginate\_by = 10

def get\_queryset(self, \*\*kwargs):

statusq = Q(status\_\_exact = 'r') | Q(status\_\_exact = 'o')

instances = MusicInstance.objects.filter(statusq)

return instances

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

return context

import json

class BorrowedPie(PermissionRequiredMixin,TemplateView):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_any\_reserve'):

return False

return True

template\_name = "catalog/borrowed\_pie.html"

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

history = ActivityLog.objects.filter(activityCode='bor').values('music\_\_title','music\_\_composer\_\_last\_name','activityCode').annotate(events=Count('id')).order\_by('-events')

chartDataArray = []

for record in history:

dict = {}

dict['x'] = record['music\_\_title'] + ' ' + record['music\_\_composer\_\_last\_name'] + ' ' + record['activityCode']

dict['value'] = str(record['events'])

chartDataArray.append(dict)

context['chartData'] = json.dumps(chartDataArray)

''' [

{x: "Beethoven", value: 25},

{x: "Williams", value: 8},

{x: "Mozart", value: 12},

{x: "Bruch", value: 11},

];

'''

return context

class BorrowedList(PermissionRequiredMixin,TemplateView):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_any\_reserve'):

return False

return True

template\_name = "catalog/borrowed\_list.html"

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

history = ActivityLog.objects.filter(activityCode='bor').values('music\_\_title','music\_\_composer\_\_last\_name','activityCode').annotate(events=Count('id')).order\_by('-events')

chartDataArray = []

for record in history:

row = [ record['music\_\_title'] + ' ' + record['music\_\_composer\_\_last\_name'],record['events'] ]

chartDataArray.append(row)

context['chartData'] = json.dumps(chartDataArray)

return context

# Now the post actions

class ReserveAction(PermissionRequiredMixin,FormView) :

template\_name = 'catalog/music\_detail.html'

form\_class = GetUserForm

success\_url = '/catalog/feedback'

def get\_context\_data(self, \*\*kwargs):

print('ReserveAction getcontextdata')

context = super().get\_context\_data(\*\*kwargs)

whichCopy= self.request.POST['reservebutton']

print('Reserve action copy is ' + str(whichCopy))

context['music'] = MusicInstance.objects.get(id = whichCopy).music

return context

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_self\_reserve'):

if not self.request.user.has\_perm('catalog.can\_any\_reserve'):

return False

return True

def form\_valid(self, form):

request = self.request

print("form reserveAction is valid")

whichCopy= request.POST['reservebutton']

instance = MusicInstance.objects.get(id = whichCopy)

user=form.cleaned\_data['user']

reservationnumber,instance = instance.reserve(user) #dateOverride= to override the date here

emailAddress= request.user.email

send\_mail(

'Music Reserved',

'Your Borrowed id is: ' + str(reservationnumber),

'adam@Bilkus.com',

[emailAddress])

messages.info(self.request,"Reservation successful: Your reservation number is %s" % (reservationnumber))

return super().form\_valid(form)

def postNotUsed(self,request,\*args,\*\*kwargs):

whichCopy= request.POST['reservebutton']

instance = MusicInstance.objects.get(id = whichCopy)

reservationnumber,instance = instance.reserve(request.user) #dateOverride= to override the date here

emailAddress= request.user.email

send\_mail(

'Music Reserved',

'Your Borrowed id is: ' + str(reservationnumber),

'adam@Bilkus.com',

[emailAddress])

messages.info(self.request,"Reservation successful: Your reservation number is %s" % (reservationnumber))

return HttpResponseRedirect("/catalog/feedback")

class CancelReserveAction(PermissionRequiredMixin, View):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_self\_reserve'):

return False

return True

def post(self,request,\*args,\*\*kwargs):

whichCopy= request.POST['cancelReservation']

instance = MusicInstance.objects.get(id = whichCopy)

reservation = MusicInstanceReservation.objects.get(musicInstance = instance, takenout = False, returned = False, cancelled = False)

reservationnumber = reservation.borrowedid

reservation.cancel(request.user)

emailAddress= request.user.email

send\_mail(

'Music Reservation has been cancelled',

'Your Borrowed id is: ' + str(reservationnumber),

'adam@Bilkus.com',

[emailAddress])

messages.info(self.request,"Reservation number %s has been cancelled" % (reservationnumber))

return HttpResponseRedirect("/catalog/feedback")

class BorrowInstanceAction(PermissionRequiredMixin, View):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_issue'):

return False

return True

def post(self, request, \*args, \*\*kwargs):

whichCopy = request.POST['instanceId']

instance = MusicInstance.objects.get(id = whichCopy)

reservation = MusicInstanceReservation.objects.get(musicInstance = instance, takenout = False, cancelled = False)

instance=reservation.borrow(request.user)

userid = reservation.userid\_id

user = User.objects.get(id = str(userid))

reservationnumber = reservation.borrowedid

email = user.email

send\_mail(

'Music Borrowed',

'Your Borrowed id is: ' + str(reservationnumber),

'adam@Bilkus.com',

[email])

messages.info(self.request, "The borrowing was successful: %s has borrowed %s" % (user, whichCopy))

return HttpResponseRedirect("/catalog/feedback")

class RenewInstanceAction(PermissionRequiredMixin, View):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_issue'):

return False

return True

def post(self,request,\*args,\*\*kwargs):

whichCopy = request.POST['instanceId']

instance = MusicInstance.objects.get(id = whichCopy)

reservation = MusicInstanceReservation.objects.get(musicInstance = instance, takenout = True, returned = False)

reservation.renew(request.user)

userid = reservation.userid\_id

user = User.objects.get(id = str(userid))

email = user.email

send\_mail(

'Music Returned',

'Your reservation: ' + str(id) +' has been returned',

'adam@Bilkus.com',

[email])

messages.info(self.request, "Return Successful: %s has returned %s" % (user, whichCopy))

return HttpResponseRedirect("/catalog/feedback")

class ReturnInstanceAction(PermissionRequiredMixin, View):

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_issue'):

return False

return True

def post(self,request,\*args,\*\*kwargs):

whichCopy = request.POST['instanceId']

instance = MusicInstance.objects.get(id = whichCopy)

reservation = MusicInstanceReservation.objects.get(musicInstance = instance, takenout = True, returned = False, cancelled=False)

user = reservation.returns(request.user)

email = user.email

send\_mail(

'Music Returned',

'Your reservation: ' + str(id) +' has been returned',

'adam@Bilkus.com',

[email])

messages.info(self.request, "Return Successful: %s has returned %s" % (user.userid, whichCopy))

return HttpResponseRedirect("/catalog/feedback/" + str(reservation.id))

class RoutineMaintenance(PermissionRequiredMixin,View):

def has\_permission(self):

if not self.request.user.is\_superuser:

return False

return True

def get(self,request,\*args,\*\*kwargs):

MusicInstanceReservation.cancelExpiredReservations(request.user)

return HttpResponse("Routine maintenance has run")

class ReviewMusic(FormView):

template\_name = 'catalog/review\_music.html'

form\_class = ReviewMusicForm

success\_url = '/catalog/feedback'

def form\_valid(self, form):

musicreservationkey=self.kwargs['pk']

musicreservation=MusicInstanceReservation.objects.get(id=musicreservationkey)

music = musicreservation.musicInstance.music

user = musicreservation.userid

rating=form.cleaned\_data['rating']

if rating == 0:

messages.warning(self.request,"You chose not to review this item - please do so in future!")

return super().form\_valid(form)

review = Review(user=user,music=music,rating=rating)

review.save()

messages.info(self.request,'Thank you for your rating')

return super().form\_valid(form)

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

musicreservationkey=self.kwargs['pk']

musicreservation=MusicInstanceReservation.objects.get(id=musicreservationkey)

music = musicreservation.musicInstance.music

user = musicreservation.userid

context['music'] = music

context['user'] = user

context['reservation'] = musicreservation

return context

class CreateRandomMusic(PermissionRequiredMixin,View):

def has\_permission(self):

if not self.request.user.is\_superuser:

return False

return True

def get(self,request,\*args,\*\*kwargs):

items = ['Symphony no: 3','Bagatelle in G','Minuet and Rondo','Concerto for Strings']

allComposers = Composer.objects.all()

for composer in allComposers:

print("creating random music for %s" % (composer.last\_name))

for itemname in items:

m = Music(

title = itemname,

composer = composer,

summary = 'Automatically generated',

barcode = '12345')

m.save()

return HttpResponse("Random music created")

class CreateRandomMusicInstances(PermissionRequiredMixin,View):

def has\_permission(self):

if not self.request.user.is\_superuser:

return False

return True

def get(self,request,\*args,\*\*kwargs):

allMusic = Music.objects.all()

for music in allMusic:

for i in random.choice(range(1,5)):

print('Creating %d instances for %s' % (i,music.title))

m = MusicInstance(

music=music

)

m.save()

return HttpResponse("Random music created")

class CreateRandomReviews(PermissionRequiredMixin,View):

def has\_permission(self):

if not self.request.user.is\_superuser:

return False

return True

def get(self,request,\*args,\*\*kwargs):

allMusic = Music.objects.all()

for userid in range(6,9): # member1 is user id 6

for music in allMusic:

myRating = random.choice(range(-3,11))

if myRating < 1:

continue

newRating = Review(user\_id=userid,music=music,rating=myRating)

newRating.save()

print("Saved rating for music id " + str(music.id))

return HttpResponse("Random reviews created ")

class FeedbackView(TemplateView):

template\_name = 'catalog/feedback.html'

class SuggestionsView(PermissionRequiredMixin,TemplateView):

template\_name = 'catalog/suggestions.html'

def has\_permission(self):

if not self.request.user.is\_authenticated:

return False

if not self.request.user.has\_perm('catalog.can\_self\_reserve'):

return False

return True

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

userId =self.kwargs['pk']

user = User.objects.get(id=userId)

context['suggestions'] = Review.suggestionsForUser(user)

context['user'] = user

return context

**The classes below are included for if I ever wanted to create separate pages for the admin to create everything instead of just sending them to the specific already built admin page**

'''

class ComposerCreate(CreateView):

model = Composer

fields = '\_\_all\_\_'

initial = {'date\_of\_death': '05/01/2018'}

class ComposerUpdate(UpdateView):

model = Composer

fields = ['first\_name', 'last\_name', 'date\_of\_birth', 'date\_of\_death']

class ComposerDelete(DeleteView):

model = Composer

success\_url = reverse\_lazy('composers')

class MusicCreate(CreateView):

model = Music

fields = '\_\_all\_\_'

class MusicUpdate(UpdateView):

model = Music

fields = '\_\_all\_\_'

class MusicDelete(DeleteView):

model = Music

success\_url = reverse\_lazy('musics')

class MusicFilter(django\_filters.FilterSet):

name = django\_filters.CharFilter(lookup\_expr='iexact')

class Meta:

model = Music

fields = ['genre', 'language']

'''

## Admin

In Django you can create an admin page which allows you to easily browse the database if you are a staff member. This means that you can then change reservations and bookings even if the page is down. It also allows admins to add users as I am not implementing a user registration page as these users are created before hand by BEAT for other resources and this website will be linked in with it.

Here is the code:

from django.contrib import admin

from import\_export.admin import ImportExportActionModelAdmin,ExportActionMixin,ImportExportMixin

from django.contrib.auth.models import User

from catalog.models import Composer, Genre, Music, MusicInstance, Language, MusicInstanceReservation,ActivityLog,Review

admin.site.register(Genre)

admin.site.register(Language)

class MusicInline(admin.TabularInline):

"""Defines format of inline music insertion (used in composerAdmin)"""

model = Music

@admin.register(Composer)

class ComposerAdmin(ImportExportMixin,admin.ModelAdmin):

"""Administration object for Composer models.

Defines:

- fields to be displayed in list view (list\_display)

- orders fields in detail view (fields),

grouping the date fields horizontally

- adds inline addition of music in composor view (inlines)

"""

list\_display = ('last\_name',

'first\_name', 'date\_of\_birth', 'date\_of\_death')

fields = ['first\_name', 'last\_name', ('date\_of\_birth', 'date\_of\_death')]

inlines = [MusicInline]

from\_encoding = 'utf-8'

class MusicsInstanceInline(admin.TabularInline):

"""Defines format of inline instance insertion (used in ComposerAdmin)"""

model = MusicInstance

class MusicAdmin(ImportExportMixin,admin.ModelAdmin):

"""Administration object for Music models.

Defines:

- fields to be displayed in list view (list\_display)

- adds inline addition of music instances in music view (inlines)

"""

list\_display = ('title', 'composer', 'display\_genre')

inlines = [MusicsInstanceInline]

admin.site.register(Music, MusicAdmin)

@admin.register(MusicInstance)

class MusicInstanceAdmin(admin.ModelAdmin):

"""Administration object for musicInstance models.

Defines:

- fields to be displayed in list view (list\_display)

- filters that will be displayed in sidebar (list\_filter)

- grouping of fields into sections (fieldsets)

"""

list\_display = ('music', 'status', 'borrower', 'due\_back', 'id')

list\_filter = ('status', 'due\_back')

fieldsets = (

('Availability', {

'fields': ('status', 'due\_back', 'borrower')

}),

)

@admin.register(MusicInstanceReservation)

class MusicInstanceReservationAdmin(admin.ModelAdmin):

model = MusicInstanceReservation

@admin.register(ActivityLog)

class ActivityLogAdmin(admin.ModelAdmin):

model = ActivityLog

@admin.register(Review)

class ReviewAdmin(ImportExportMixin,admin.ModelAdmin):

model = Review;

from django.contrib import admin

from import\_export import resources

from catalog.models import Composer

class ComposerResource(resources.ModelResource):

class Meta:

model = Composer

class MusicResource(resources.ModelResource):

class Meta:

model = Music

class GenreResource(resources.ModelResource):

class Meta:

model = Genre

admin.site.unregister(User)

class UserResource(resources.ModelResource):

class Meta:

model = User

class UserAdmin(ImportExportMixin,admin.ModelAdmin):

"""Administration object

Defines:

- fields to be displayed in list view (list\_display)

- filters that will be displayed in sidebar (list\_filter)

- grouping of fields into sections (fieldsets)

"""

pass

admin.site.register(User, UserAdmin)

URLS

This is where I setup all of the URLS for my pages

from django.urls import path

from catalog import views

# The home page is the only one available to non-logged in users

# It displays differently depending on whether you are logged in

urlpatterns = [

path('',views.HomePageView.as\_view(),name='index'),

path('feedback/', views.FeedbackView.as\_view(), name='feedback'),

]

# From now on you must be logged in

# These patterns correspond to the functions available to anyone whether or not a member

urlpatterns += [

path('musicListOld/', views.MusicListView.as\_view(), name='musics'),

path('musicList/', views.MusicListGridView.as\_view(), name='musicgrid'),

path('musicDetail/<int:pk>', views.MusicDetailView.as\_view(), name='music-detail'),

path('composerList/', views.ComposerListView.as\_view(), name='composers'),

path('composerDetail/<int:pk>',

views.ComposerDetailView.as\_view(), name='composer\_detail'),

]

# These patterns implement functions only available to members who therefore have reservation rights

urlpatterns += [

path('suggestions/<int:pk>', views.SuggestionsView.as\_view(), name='suggestions'),

# the button which actually makes a reservation

path('reserveAction/', views.ReserveAction.as\_view(), name='reserveAction'),

# list of music which has been borrowed or reserved by the user

path('borrowedOrReservedByUser/', views.BorrowedOrReservedByUser.as\_view(), name='my-borrowed'),

path('cancelReserveAction/',views.CancelReserveAction.as\_view(),name='cancelReserveAction'),

]

# These are only available to librarians

urlpatterns += [

path('borrowAction/', views.BorrowInstanceAction.as\_view(), name='borrowAction'),

path('renewAction/', views.RenewInstanceAction.as\_view(), name='renewAction'),

path('returnAction/', views.ReturnInstanceAction.as\_view(), name='returnAction'),

path('reviewMusic/<int:pk>', views.ReviewMusic.as\_view(), name='reviewMusic'),

path('borrowedOrReservedByAll/', views.BorrowedOrReservedByAll.as\_view(), name='all-borrowed'),

path('borrowedPie/', views.BorrowedPie.as\_view(), name='borrowedPie'),

path('borrowedList/', views.BorrowedList.as\_view(), name='borrowedList'),

path('routineMaintenance/',views.RoutineMaintenance.as\_view(),name='routineMaintenance'),

path('createRandomMusic/',views.CreateRandomMusic.as\_view(),name='createRandomMusic'),

path('createRandomMusicInstances/',views.CreateRandomMusicInstances.as\_view(),name='createRandomMusicInstances'),

path('createRandomReviews/',views.CreateRandomReviews.as\_view(),name='createRandomReviews'),

]

'''

# Add URLConf to create, update, and delete composers only needed for if I wanted to create separate pages but I didn’t have time

urlpatterns += [

path('composer/create/', views.ComposerCreate.as\_view(), name='composer\_create'),

path('composer/<int:pk>/update/', views.ComposerUpdate.as\_view(), name='composer\_update'),

path('composer/<int:pk>/delete/', views.ComposerDelete.as\_view(), name='composer\_delete'),

]

# Add URLConf to create, update, and delete music

urlpatterns += [

path('music/create/', views.MusicCreate.as\_view(), name='music\_create'),

path('music/<int:pk>/update/', views.MusicUpdate.as\_view(), name='music\_update'),

path('music/<int:pk>/delete/', views.MusicDelete.as\_view(), name='music\_delete'),

]

'''

## Templates

Here is the html code/Django logic

### Overall page

This code overlays the whole website by giving meaning to different pages.

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

{% block title %}<title>Barnet Education Arts Trust</title>{% endblock %}

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

<!-- Add additional CSS in static file -->

{% load static %}

<link rel="stylesheet" href="{% static 'css/styles.css' %}">

<style>

body {

background-color: white;

}

.backgroundimage {

background-image: url("/static/images/background.jpg");

background-size: cover;

}

</style>

<link rel="shortcut icon" href="/static/images/favicon.ico" type="image/x-icon">

<link rel="icon" href="/favicon.ico" type="image/x-icon">

</head>

<body style="height:100%">

<a href="/catalog/">

<h1 role='textbox' align="center"><img src="/static/images/BEAT-logo.png" height="150"

alt="Beat Logo"></img></h1>

</a>

<div class="container-fluid backgroundimage" >

<div class="row">

<div class="col">

{% if messages %}

<div class="messages" style="background: white;">

{% for message in messages %}

<div class="alert alert-{{ message.tags }}">{{ message }}</div>

{% endfor %}

</div>

{% endif %}

</div>

</div>

<div class="row">

<div class="col-sm-2">

</div>

<div class="col-sm-10">

<h1>Welcome to the BEAT booking system</h1>

</div>

</div>

<div class="row" >

<div class="col-sm-2" style="background:white;" >

{% block sidebar %}

<nav class="nav navbar-expand flex-column">

<a class="nav-link" href="/catalog">Home</a>

{% if not user.is\_authenticated %}

<a class="nav-link" href="/accounts/login">Login</a>

{% else %}

<a class="nav-link" href="/accounts/logout">Logout as {{user}}</a>

<nav class="nav-link">

Browse and Reserve Catalogue

</nav>

<nav class="nav-item pl-2">

<a class="nav-link" href="/catalog/musicList/">See all music</a>

<a class="nav-link" href="/catalog/composerList/">See all composers</a>

</nav>

<nav class="nav-link">

Manage reservations

</nav>

<nav class="nav-item pl-2">

{% if perms.catalog.can\_self\_reserve %}

<a class="nav-link" href="/catalog/suggestions/{{user.id}}">Suggestions</a>

</nav>

{% endif %}

<nav class="nav-item pl-2">

{% if perms.catalog.can\_self\_reserve %}

<a class="nav-link" href="/catalog/borrowedOrReservedByUser">Your reservations</a>

</nav>

{% endif %}

{% if perms.catalog.can\_any\_reserve %}

<nav class="nav-link">

Librarian functions

</nav>

<nav class="nav-item pl-2">

<a class="nav-link" href="/catalog/borrowedOrReservedByAll">All reservations</a>

</nav>

<nav class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Reports

</a>

<div class="dropdown-menu" aria-labelledby="navbarDropdown">

<a class="dropdown-item" href="/catalog/borrowedPie/">Borrowed by Piece (pie)</a>

<a class="dropdown-item" href="/catalog/borrowedList/">Borrowed by Piece List</a>

</div>

</nav>

{% endif %}

{% if perms.catalog.is\_admin %}

<nav class="nav-link">

Administration

</nav>

<nav class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Admin Functions

</a>

<div class="dropdown-menu" aria-labelledby="navbarDropdown">

<a class="dropdown-item" href="/admin/catalog/composer/">Edit Composer</a>

<a class="dropdown-item" href="/admin/catalog/composer/add/">Create Composer</a>

<a class="dropdown-item" href="/admin/catalog/musicinstance/">Edit Music Instance</a>

<a class="dropdown-item" href="/admin/catalog/musicinstance/">Delete Music Instance</a>

<a class="dropdown-item" href="/admin/catalog/music/add/">Create Music</a>

<a class="dropdown-item" href="/admin/catalog/music/">Edit Music</a>

<a class="dropdown-item" href="/admin/catalog/genre/add/">Create Genre</a>

<a class="dropdown-item" href="/admin/catalog/genre/">Edit Genre</a>

<a class="dropdown-item" href="/admin/catalog/language/">Edit Language</a>

<a class="dropdown-item" href="/admin/catalog/language/">Add Language</a>

</div>

</nav>

{% endif %}

{% endif %}

</nav>

{% endblock %}

</div>

<div class ="col-sm-1"></div>

<div class="col-sm-8 " style="background:white;" >

{% block content %}{% endblock %}

{% block pagination %}

{% if is\_paginated %}

<div class="pagination">

<span class="page-links">

{% if page\_obj.has\_previous %}

<a href="{{ request.path }}?page={{ page\_obj.previous\_page\_number }}">previous</a>

{% endif %}

<span class="page-current">

Page {{ page\_obj.number }} of {{ page\_obj.paginator.num\_pages }}.

</span>

{% if page\_obj.has\_next %}

<a href="{{ request.path }}?page={{ page\_obj.next\_page\_number }}">next</a>

{% endif %}

</span>

</div>

{% endif %}

{% endblock %}

</div>

<div class ="col-sm-1"></div>

</div>

<div class="row" >

<p>&nbsp</p>

</div>

</div>

</body>

</html>

### Indexes

#### Librarian

{% extends "base\_generic.html" %}

{% block content %}

<h1 align="center">Home Page</h1>

<p>Welcome to <em>Barnet Education Arts Trust Website</em>, a draft website developed as a start to computer science coursework</p>

You are logged in as a librarian

{% if can\_reserve %}

This user can make reservations

{% else %}

This user can only browse

{% endif %}

{% if perms.catalog.can\_self\_reserve %}

{% endif %}

<h2>Music Details</h2>

<p>The music office has the following record counts:</p>

<ul>

<li><strong>Music:</strong> {{ num\_music }}</li>

<li><strong>Amount of music owned:</strong> {{ num\_instances }}</li>

<li><strong>Amount of music available:</strong> {{ num\_instances\_available }}</li>

<li><strong>Composers:</strong> {{ num\_composers }}</li>

</ul>

<div id="calendar"></div>

<link href='https://unpkg.com/@fullcalendar/core@4.3.1/main.min.css' rel='stylesheet' />

<link href='https://unpkg.com/@fullcalendar/daygrid@4.3.0/main.min.css' rel='stylesheet' />

<link href='https://unpkg.com/@fullcalendar/timegrid@4.3.0/main.min.css' rel='stylesheet' />

<script src='https://unpkg.com/@fullcalendar/core@4.3.1/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/interaction@4.3.0/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/daygrid@4.3.0/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/timegrid@4.3.0/main.min.js'></script>

<script>

document.addEventListener('DOMContentLoaded', function() {

var calendarEl = document.getElementById('calendar');

var calendar = new FullCalendar.Calendar(calendarEl, {

plugins: [ 'interaction', 'dayGrid', 'timeGrid' ],

defaultView: 'dayGridMonth',

defaultDate: '{{ calendarStartDate}}',

header: {

left: 'prev,next today',

center: 'title',

right: 'dayGridMonth,timeGridWeek,timeGridDay'

},

events: [

{% for event in event\_list %}

{{event|safe}}

{% endfor %}

]

});

calendar.render();

});

</script>

{% endblock %}

#### Non-member Index

{% extends "base\_generic.html" %}

{% block content %}

<h1 align="center">Home Page</h1>

<p>Welcome to <em>Barnet Education Arts Trust Website</em>, a draft website developed as a start to computer science coursework</p>

You are logged in but not yet a member, so you can't borrow anything

{% if can\_reserve %}

This user can make reservations

{% else %}

This user can only browse

{% endif %}

{% if perms.catalog.can\_self\_reserve %}

CAN SELF RESERVE THROUGH TEMPLATE

{% endif %}

<h2>Music Details</h2>

<p>The music office has the following record counts:</p>

<ul>

<li><strong>Music:</strong> {{ num\_music }}</li>

<li><strong>Amount of music owned:</strong> {{ num\_instances }}</li>

<li><strong>Amount of music available:</strong> {{ num\_instances\_available }}</li>

<li><strong>Composers:</strong> {{ num\_composers }}</li>

</ul>

<div id="chatbot" style="display:none">

<iframe src="https://titanembeds.com/embed/642070935620550666?css=42" height="300" width="400" frameborder="0" theme=MetroEdge></iframe>

<button type="button" onclick="closeChatbot()" class="btn btn-primary">End Chat</button>

</div>

<div id="chatbotbutton">

<button type="button" onclick="showChatbot()" class="btn btn-primary">Chat with a help engine</button>

</div>

<script>

function showChatbot() {

let chatbot = document.getElementById("chatbot");

let chatbotbutton = document.getElementById("chatbotbutton");

chatbot.style.display = "block";

chatbotbutton.style.display = "none";

}

function closeChatbot() {

let chatbot = document.getElementById("chatbot");

let chatbotbutton = document.getElementById("chatbotbutton");

chatbotbutton.style.display = "block";

chatbot.style.display = "none";

}

</script>

{% endblock %}

#### Member Index

{% extends "base\_generic.html" %}

{% block content %}

<h1 align="center">Home Page</h1>

<p>Welcome to <em>Barnet Education Arts Trust Website</em>, a draft website developed as a start to computer science coursework</p>

You are logged in as a member

<!--

{% if can\_reserve %}

This user can make reservations

{% else %}

This user can browse

{% endif %}

{% if perms.catalog.can\_self\_reserve %}

CAN SELF RESERVE THROUGH TEMPLATE

{% endif %}

-->

<h2>Music Details</h2>

<p>The music office has the following record counts:</p>

<ul>

<li><strong>Music:</strong> {{ num\_music }}</li>

<li><strong>Amount of music owned:</strong> {{ num\_instances }}</li>

<li><strong>Amount of music available:</strong> {{ num\_instances\_available }}</li>

<li><strong>Composers:</strong> {{ num\_composers }}</li>

</ul><div id="chatbot" style="display:none">

<iframe src="https://titanembeds.com/embed/642070935620550666?css=42" height="300" width="400" frameborder="0" theme=MetroEdge></iframe>

<button type="button" onclick="closeChatbot()" class="btn btn-primary">End Chat</button>

</div>

<div id="chatbotbutton">

<button type="button" onclick="showChatbot()" class="btn btn-primary">Chat with a help engine</button>

</div>

<div id="calendar"></div>

<link href='https://unpkg.com/@fullcalendar/core@4.3.1/main.min.css' rel='stylesheet' />

<link href='https://unpkg.com/@fullcalendar/daygrid@4.3.0/main.min.css' rel='stylesheet' />

<link href='https://unpkg.com/@fullcalendar/timegrid@4.3.0/main.min.css' rel='stylesheet' />

<script src='https://unpkg.com/@fullcalendar/core@4.3.1/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/interaction@4.3.0/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/daygrid@4.3.0/main.min.js'></script>

<script src='https://unpkg.com/@fullcalendar/timegrid@4.3.0/main.min.js'></script>

<script>

function showChatbot() {

let chatbot = document.getElementById("chatbot");

let chatbotbutton = document.getElementById("chatbotbutton");

chatbot.style.display = "block";

chatbotbutton.style.display = "none";

}

function closeChatbot() {

let chatbot = document.getElementById("chatbot");

let chatbotbutton = document.getElementById("chatbotbutton");

chatbotbutton.style.display = "block";

chatbot.style.display = "none";

}

</script>

<script>

document.addEventListener('DOMContentLoaded', function() {

var calendarEl = document.getElementById('calendar');

var calendar = new FullCalendar.Calendar(calendarEl, {

plugins: [ 'interaction', 'dayGrid', 'timeGrid' ],

defaultView: 'dayGridMonth',

defaultDate: '{{ calendarStartDate}}',

header: {

left: 'prev,next today',

center: 'title',

right: 'dayGridMonth,timeGridWeek,timeGridDay'

},

events: [

{% for event in event\_list %}

{{event|safe}}

{% endfor %}

]

});

calendar.render();

});

</script>

{% endblock %}

#### Visitor Index

{% extends "base\_generic.html" %}

{% block content %}

<h1 align="center">Home Page</h1>

<p>Welcome to <em>Barnet Education Arts Trust Website</em>, a draft website developed as a start to computer science coursework</p>

<p>

You are here as a visitor. In order to browse the catalogue to see the music available to borrow, you need to login or register

</p>

{% endblock %}

The Admin Doesn’t need a separate index as they see everything on the homepage that the librarian sees and the extra links they see are all in the base\_generic.html file

### All Music (As the button only shows up if the music is available and you have gone to its detailed page and you are allowed to borrow music)

{% extends "base\_generic.html" %}

{% block content %}

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.11.4/themes/redmond/jquery-ui.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/free-jqgrid/4.15.5/css/ui.jqgrid.min.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/free-jqgrid/4.15.5/jquery.jqgrid.min.js"></script>

<h2>List of all music pieces in the system</h2>

<h4>Click on a column name to sort by it, or enter in the box text to filter</h3>

<h4>Click on a row to select it for more detail or to reserve</h4>

<table id="grid"></table>

<script>

$(function() {

"use strict";

let data = {};

{% if queryjson %}

data = {{queryjson|safe}};

{% endif %}

$('#grid').jqGrid({

toppager:true,pager:true,rowNum:15,

colModel:[

{name: "id",width:1,hidden:true},

{name: "title",label:'Title',width:300},

{name: "composer\_\_last\_name",label:'Composer'},

{name: "genre\_\_name",label:'Genre',width:50},

{name: "language\_\_name",label:'Language',width:80},

],

data: data,

guiStyle: "bootstrap4",

ondblClickRow: function(rowId,iRow,iCol,e) {

let grid = $("#grid");

let id = grid.jqGrid('getCell',rowId,'title');

console.log("clicked row",rowId,id);

alert("clicked row",rowId,id);

},

}).jqGrid('filterToolbar').jqGrid('setGridParam',{

onSelectRow: function(rowId,e) {

let grid = $("#grid");

let id = grid.jqGrid('getCell',rowId,'id');

window.location.href = '/catalog/musicDetail/' + id;

}

});

})

</script>

{% endblock %}

### Actual Music Detail Page with button to reserve

{% extends "base\_generic.html" %}

{% block content %}

<h1>Title: {{ music.title }}</h1>

<p><strong>Composer:</strong> <a href="{% url 'composer\_detail' music.composer.pk %}">{{ music.composer }}</a></p>

<p><strong>Summary:</strong> {{ music.summary }}</p>

<p><strong>ISBN:</strong> {{ music.isbn }}</p>

<p><strong>Language:</strong> {{ music.language }}</p>

<p><strong>Genre:</strong> {% for genre in music.genre.all %}{{genre}}{% if not forloop.last %}, {% endif %}{% endfor %}</p>

<div style="margin-left:20px;margin-top:20px">

{% if navailable == 0 %}

<h3>There are no copies available to borrow at the moment</h3>

{% else %}

{% if show\_reserve\_button %}

<form action="{%url 'reserveAction' %}" method="post">

{% csrf\_token %}

<button type="submit" name="reservebutton" value="{{firstavailable.id}}">Reserve a copy</button>

{% if perms.can\_any\_reserve %}

{{form}}

{% endif %}

</form>

{% endif %}

{% endif %}

</div>

{% endblock %}

### Borrowed/Reserved Page for Users to see their current music

{% extends "base\_generic.html" %}

{% block content %}

<h1>Reserved or borrowed music for {{user}}</h1>

<form action="{%url 'cancelReserveAction' %}" method="post">

{% csrf\_token %}

<ul>

{% for musicinstance in instances %}

<li>

{{musicinstance.music.title}}

{% if musicinstance.status == 'r' %}

<button type="submit" name="cancelReservation" value="{{musicinstance.id}}">Cancel this reservation</button>

{% else %}

<label>Borrowed until {{ musicinstance.due\_back }}</label>

{% endif %}

</li>

{% endfor %}

</ul>

</form>

{% endblock %}

This allows you to cancel a reservation online without having to see a librarian.

### Borrowed/Reserved For All

{% extends "base\_generic.html" %}

{% block content %}

<h1>All reserved or borrowed music</h1>

<ul>

{% for musicinstance in instances %}

<li>

{{musicinstance.music.title}}

{{musicinstance.borrower}}

{% if musicinstance.status == 'r' %}

<label>Reserved until {{ musicinstance.due\_back }}</label>

<form style="display:inline" action="{%url 'cancelReserveAction' %}" method="post">

{% csrf\_token %}

<button type="submit" name="cancelReservation" value="{{musicinstance.id}}">Cancel this reservation</button>

</form>

<form style="display:inline" action="{%url 'borrowAction' %}" method="post">

{% csrf\_token %}

<button type="submit" name="instanceId" value="{{musicinstance.id}}">Check out as borrowed</button>

</form>

{% else %}

<label>Borrowed until {{ musicinstance.due\_back }}</label>

<form style="display:inline" action="{%url 'returnAction' %}" method="post">

{% csrf\_token %}

<button type="submit" name="instanceId" value="{{musicinstance.id}}">Return</button>

</form>

<form style="display:inline" action="{%url 'renewAction' %}" method="post">

{% csrf\_token %}

<button type="submit" name="instanceId" value="{{musicinstance.id}}">Renew</button>

</form>

{% endif %}

</li>

{% endfor %}

</ul>

{% endblock %}

This includes the button for returning the music and a link to the returning action page. It also allows librarians to renew music.

### Login

{% extends "base\_generic.html" %}

{% block content %}

{% if form.errors %}

<p>Your username and password didn't match. Please try again.</p>

{% endif %}

{% if next %}

{% if user.is\_authenticated %}

<p>Your account doesn't have access to this page. To proceed,

please login with an account that has access.</p>

{% else %}

<p>Please login to see this page.</p>

{% endif %}

{% endif %}

<form method="post" action="{% url 'login' %}">

{% csrf\_token %}

<table>

<tr>

<td>{{ form.username.label\_tag }}</td>

<td>{{ form.username }}</td>

</tr>

<tr>

<td>{{ form.password.label\_tag }}</td>

<td>{{ form.password }}</td>

</tr>

</table>

<input type="submit" value="login" />

<input type="hidden" name="next" value="{{ next }}" />

</form>

{# Assumes you setup the password\_reset view in your URLconf #}

<p><a href="{% url 'password\_reset' %}">Lost password?</a></p>

{% endblock %}

### Password Edit Form

{% extends "base\_generic.html" %}

{% block content %}

<form action="" method="post">{% csrf\_token %}

{% if form.email.errors %}{{ form.email.errors }}{% endif %}

<p>{{ form.email }}</p>

<input type="submit" class='btn btn-default btn-lg' value="Reset password" />

</form>

{% endblock %}

### Password Reset Done

{% extends "base\_generic.html" %}

{% block content %}

<p>We've emailed you instructions for setting your password. If they haven't arrived in a few minutes, check your spam folder.</p>

{% endblock %}

### Password Reset Confirmation

{% extends "base\_generic.html" %}

{% block content %}

{% if validlink %}

<p>Please enter (and confirm) your new password.</p>

<form action="" method="post">

<div style="display:none">

<input type="hidden" value="{{ csrf\_token }}" name="csrfmiddlewaretoken">

</div>

<table>

<tr>

<td>{{ form.new\_password1.errors }}

<label for="id\_new\_password1">New password:</label></td>

<td>{{ form.new\_password1 }}</td>

</tr>

<tr>

<td>{{ form.new\_password2.errors }}

<label for="id\_new\_password2">Confirm password:</label></td>

<td>{{ form.new\_password2 }}</td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="Change my password" /></td>

</tr>

</table>

</form>

{% else %}

<h1>Password reset failed</h1>

<p>The password reset link was invalid, possibly because it has already been used. Please request a new password reset.</p>

{% endif %}

{% endblock %}

### Password Reset Complete

{% extends "base\_generic.html" %}

{% block content %}

<h1>The password has been changed!</h1>

<p><a href="{% url 'login' %}">log in again?</a></p>

{% endblock %}

### Password Reset Email

Someone asked for password reset for email {{ email }}. Follow the link below:

{{ protocol}}://{{ domain }}{% url 'password\_reset\_confirm' uidb64=uid token=token %}

### Borrowed Pie Chart to allow you to see the most borrowed pieces of music

{% extends "base\_generic.html" %}

{% block content %}

<h2>Chart of activities</h2>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{{ music.get\_absolute\_url }}">{{ music.title }}</a> ({{music.composer}})

</li>

{% endfor %}

</ul>

{% else %}

{% endif %}

<script src="https://cdn.anychart.com/js/8.0.1/anychart-core.min.js"></script>

<script src="https://cdn.anychart.com/js/8.0.1/anychart-pie.min.js"></script>

<div id="container" style="width: 100%; height: 100%"></div>

<script>

let data = {{chartData|safe}}

let chart = anychart.pie();

chart.data(data);

chart.container('container');

chart.draw();

</script>

{% endblock %}

### Borrowed List to see most borrowed music as a list

{% extends "base\_generic.html" %}

{% block content %}

<h2>Music by times borrowed</h2>

<script src="https://cdn.anychart.com/js/8.0.1/anychart-base.min.js"></script>

<div id="container" style="width: 100%; height: 100%"></div>

<script>

let data = {{chartData|safe}}

let chart = anychart.bar();

chart.data(data);

chart.container('container');

chart.draw();

</script>

{% endblock %}

### Composer List

{% extends "base\_generic.html" %}

{% block content %}

<h1>Composer List</h1>

{% if composer\_list %}

<ul>

{% for composer in composer\_list %}

<li>

<a href="{{ composer.get\_absolute\_url }}">

{{ composer }} ({{composer.date\_of\_birth}} - {% if composer.date\_of\_death %}{{composer.date\_of\_death}}{% endif %})

</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There are no composers available.</p>

{% endif %}

{% endblock %}

### Composer Detail

{% extends "base\_generic.html" %}

{% block content %}

<h1>Composer: {{ composer }} </h1>

<p>{{composer.date\_of\_birth}} - {% if composer.date\_of\_death %}{{composer.date\_of\_death}}{% endif %}</p>

<div style="margin-left:20px;margin-top:20px">

<h4>Music</h4>

<dl>

{% for music in composer.music\_set.all %}

<dt><a href="{% url 'music-detail' music.pk %}">{{music}}</a> ({{music.musicinstance\_set.all.count}})</dt>

<dd>{{music.summary}}</dd>

{% endfor %}

</dl>

</div>

{% endblock %}

### Feedback Page to confirm pages

{% extends "base\_generic.html" %}

{% block content %}

{% if messages %}

<div class="messages" style="background: white;">

{% for message in messages %}

<div class="alert alert-{{ message.tags }}">{{ message }}</div>

{% endfor %}

</div>

{% endif %}

{% endblock %}

### Return Page

{% extends "base\_generic.html" %}

{% block content %}

<h1>Title: {{ music.title }}</h1>

<p><strong>Composer:</strong> <a href="{% url 'composer\_detail' music.composer.pk %}">{{ music.composer }}</a></p>

<p><strong>Summary:</strong> {{ music.summary }}</p>

<p><strong>ISBN:</strong> {{ music.isbn }}</p>

<p><strong>Language:</strong> {{ music.language }}</p>

<p><strong>Genre:</strong> {% for genre in music.genre.all %}{{genre}}{% if not forloop.last %}, {% endif %}{% endfor %}</p>

<div style="margin-left:20px;margin-top:20px">

<h4>Copies</h4>

<form action="{%url 'return\_action' %}" method="post">

{% csrf\_token %}

<table>

<thead></thead>

<tbody>

{% for copy in available %}

<tr>

<td>{{copy.id}}</td>

<td><button type="submit" name="returnbutton" value="{{copy.id}}">Return</button></td>

<td><input type="hidden" name="userid" value="{{user.id}}"

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% endblock %}

### Review Page

{% extends "base\_generic.html" %}

{% block content %}

<h1>Title: Review of {{ music.title }} by {{ user.username }} ({{user.first\_name}} {{user.last\_name}})</h1>

You have just returned the above piece to the library on behalf of the above user.

Please help other users and provide their rating out of 10 where 1 means terrible and 10 brilliant.

<p>

Many thanks

</p>

<div style="margin-left:20px;margin-top:20px">

<form action="/catalog/reviewMusic/{{reservation.id}}" method="post">

{% csrf\_token %}

{{ form }}

<input type="submit" value="Submit">

</form>

</div>

{% endblock %}

### Suggestions Page

{% extends "base\_generic.html" %}

{% block content %}

<h2>Based on previous reviews, we suggest the following pieces for user {{user.first\_name}} {{user.last\_name}}:</h2>

<ul>

{% for suggestion in suggestions %}

<li><a href="/catalog/musicDetail/{{suggestion.id}}">{{suggestion.title}} {{suggestion.composer.last\_name}}</a></li>

{% endfor %}

</ul>

{% endblock %}