# 2-Design

## 2.1-High Level Overview

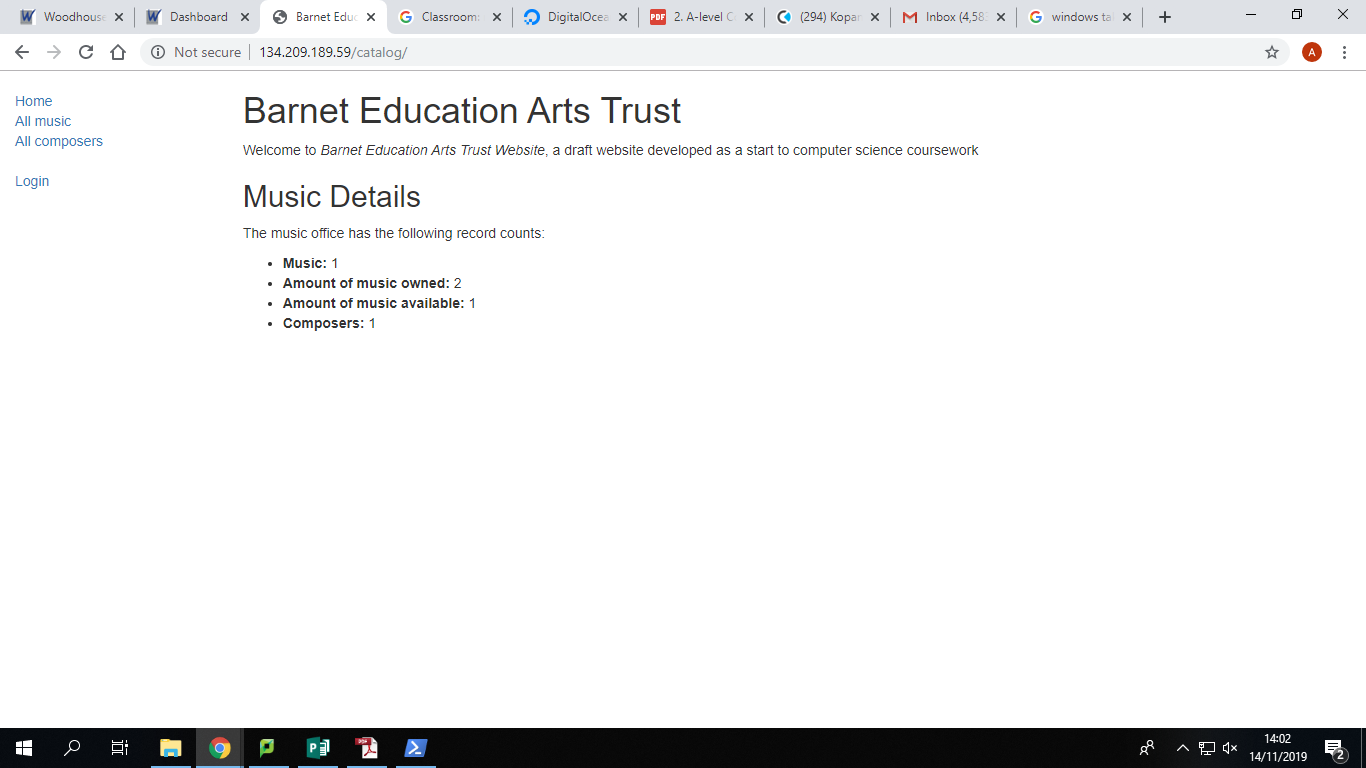
These flowcharts describe the general overview of how my website will work. It doesn’t have a registration page as this user is already created by the admin as it is part of a bigger picture.

### Main Menu

A close up of a map

Description automatically generated

This is what the logged out main menu looks like:

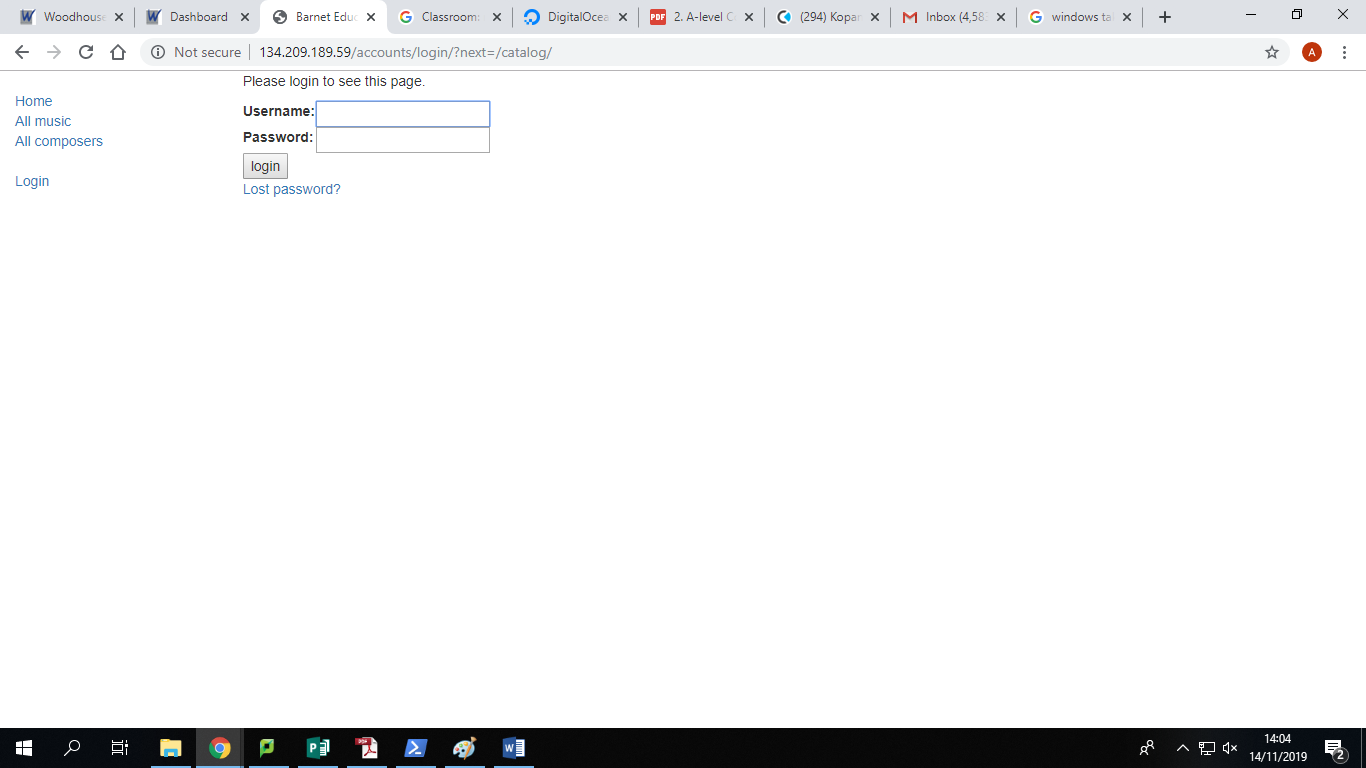


### Login

**A close up of a piece of paper

Description automatically generated**

This is what the login menu looks like:

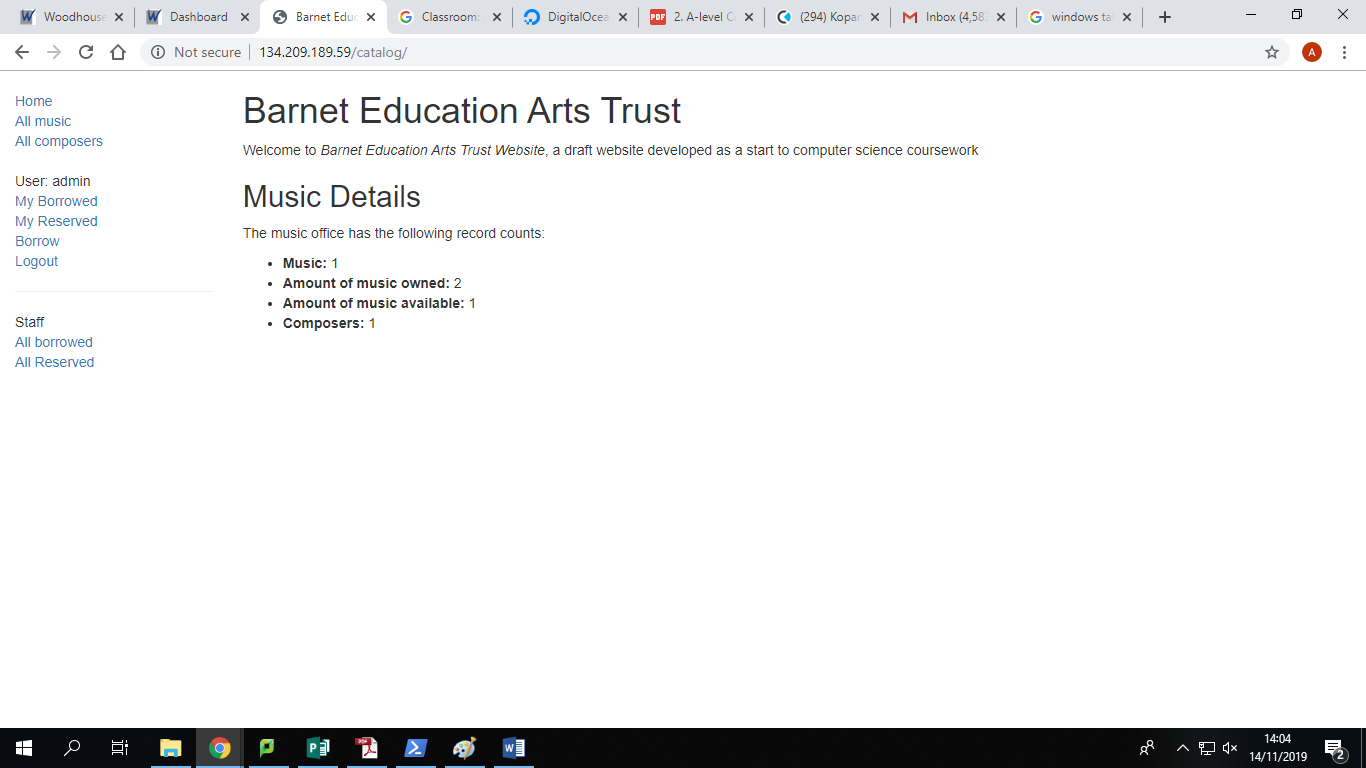


### Staff Main Menu Actions

**A close up of a mans face

Description automatically generated**

This is what the staff main menu looks like:

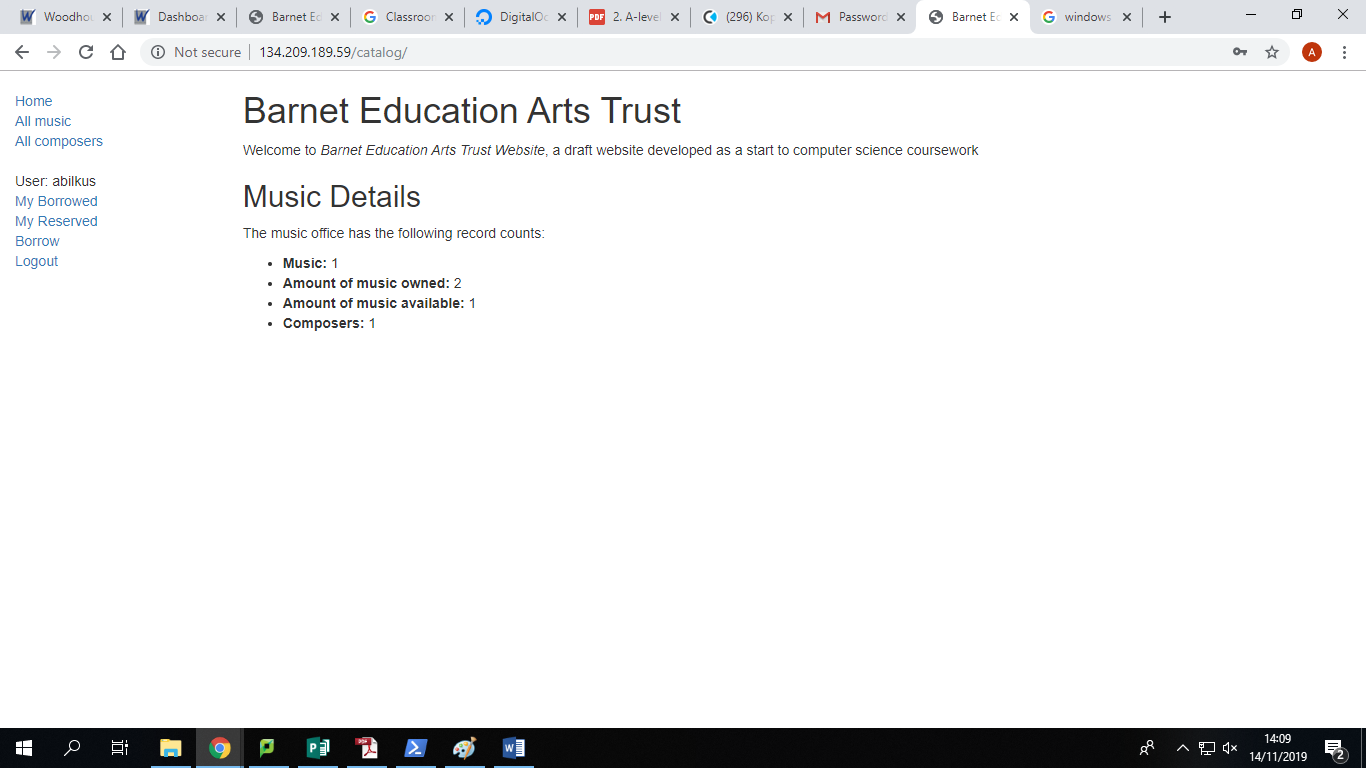
****

### Non-Staff Main Menu Actions

**A close up of a clock

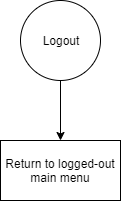
Description automatically generated**

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



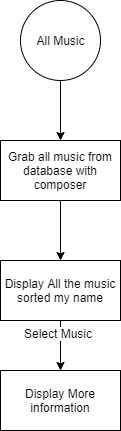
It is missing the staff actions

## Logout

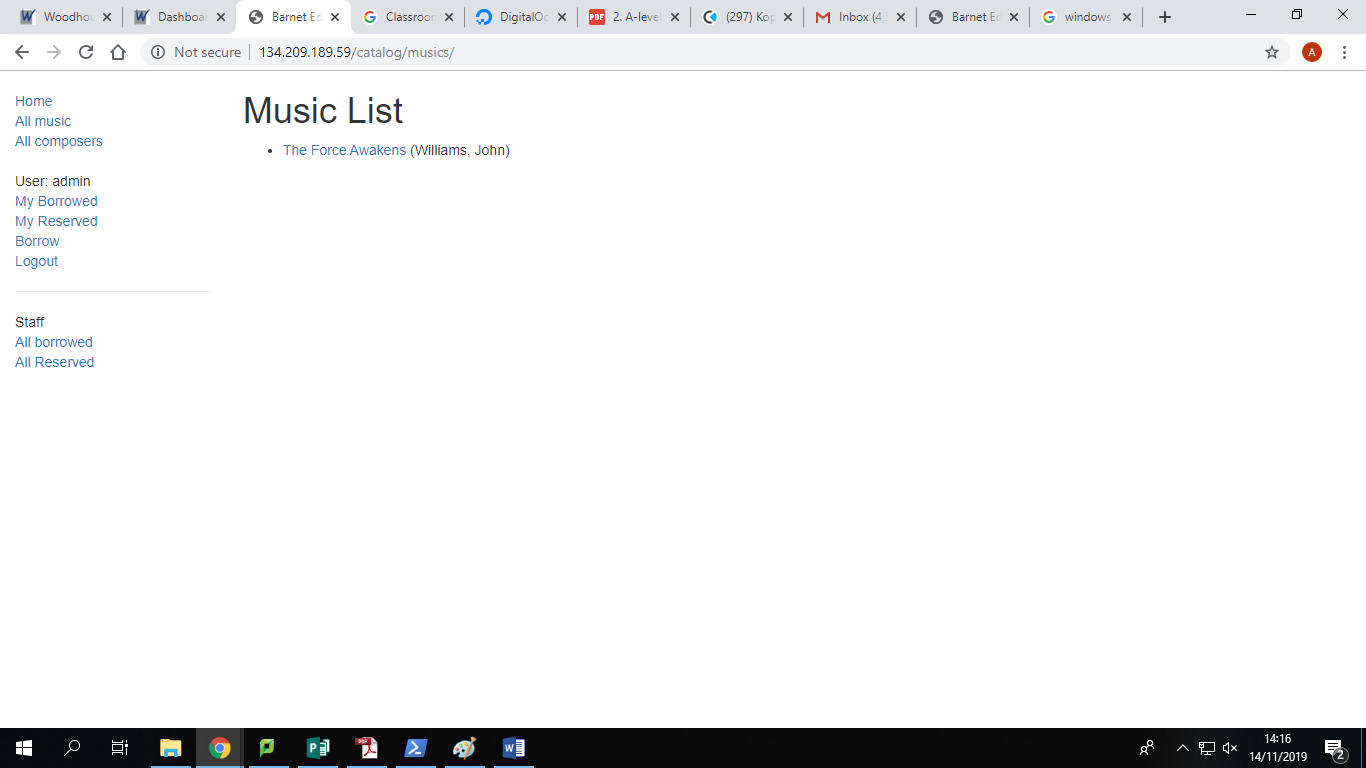
****

**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:**

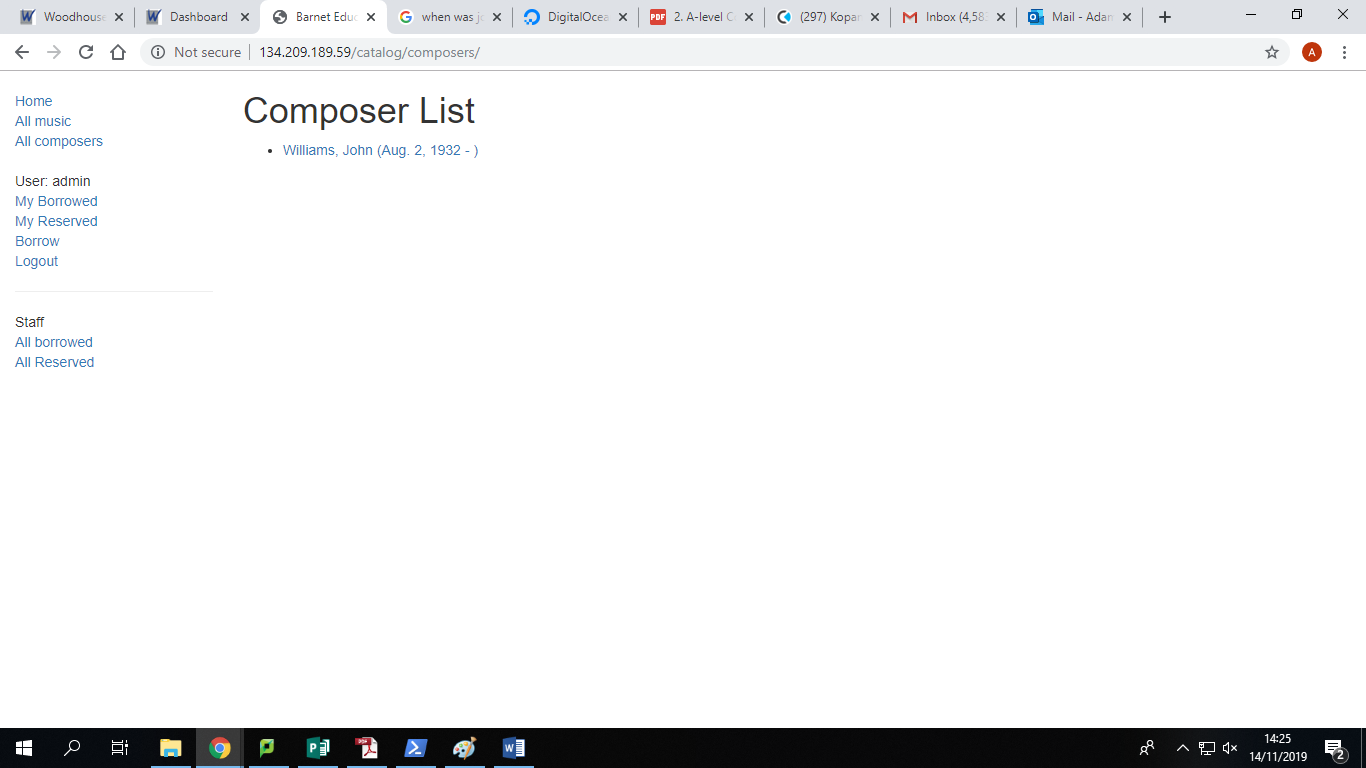
****

### All Composers

A screenshot of a cell phone

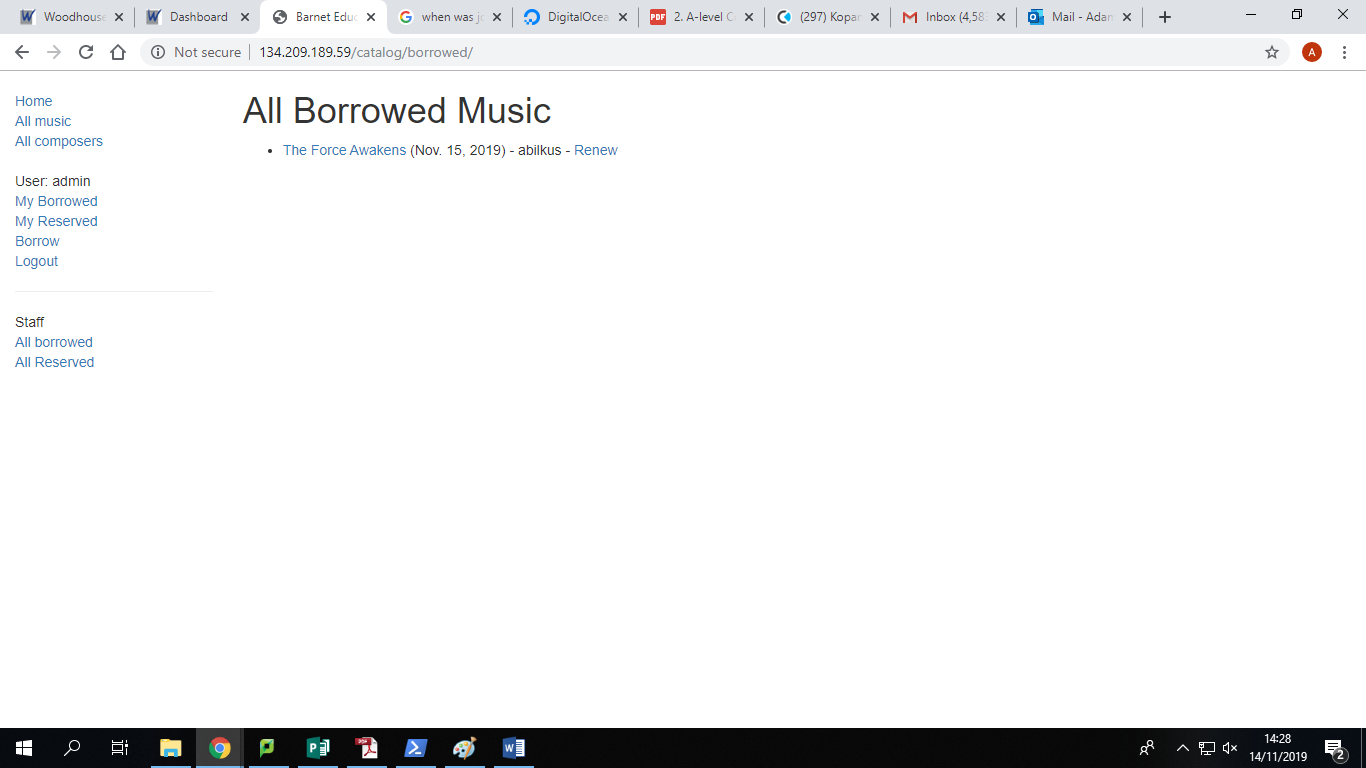
Description automatically generated

This is what the All Composers page looks like:

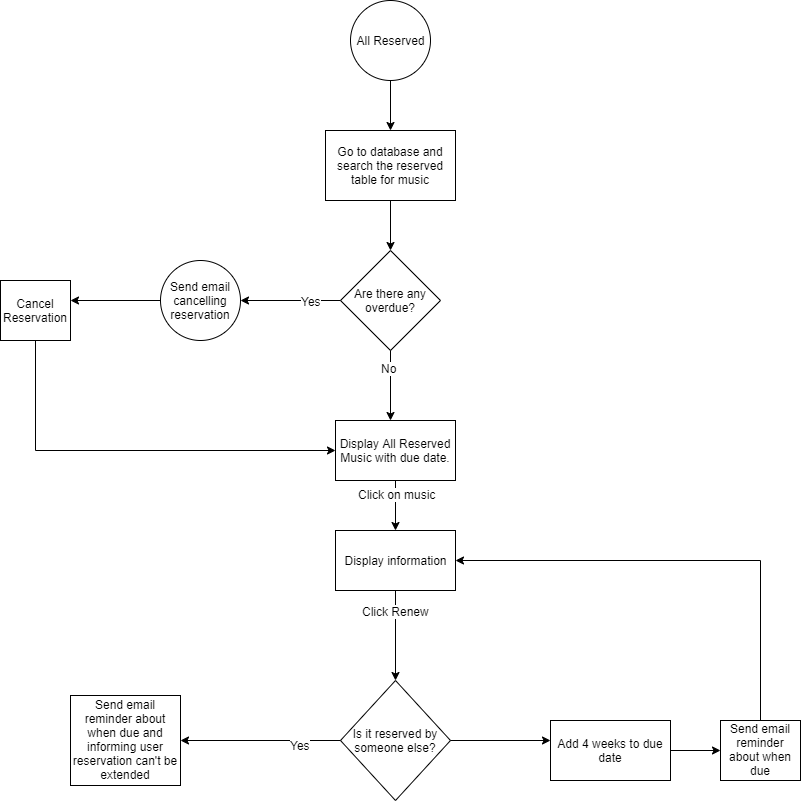


### All Borrowed

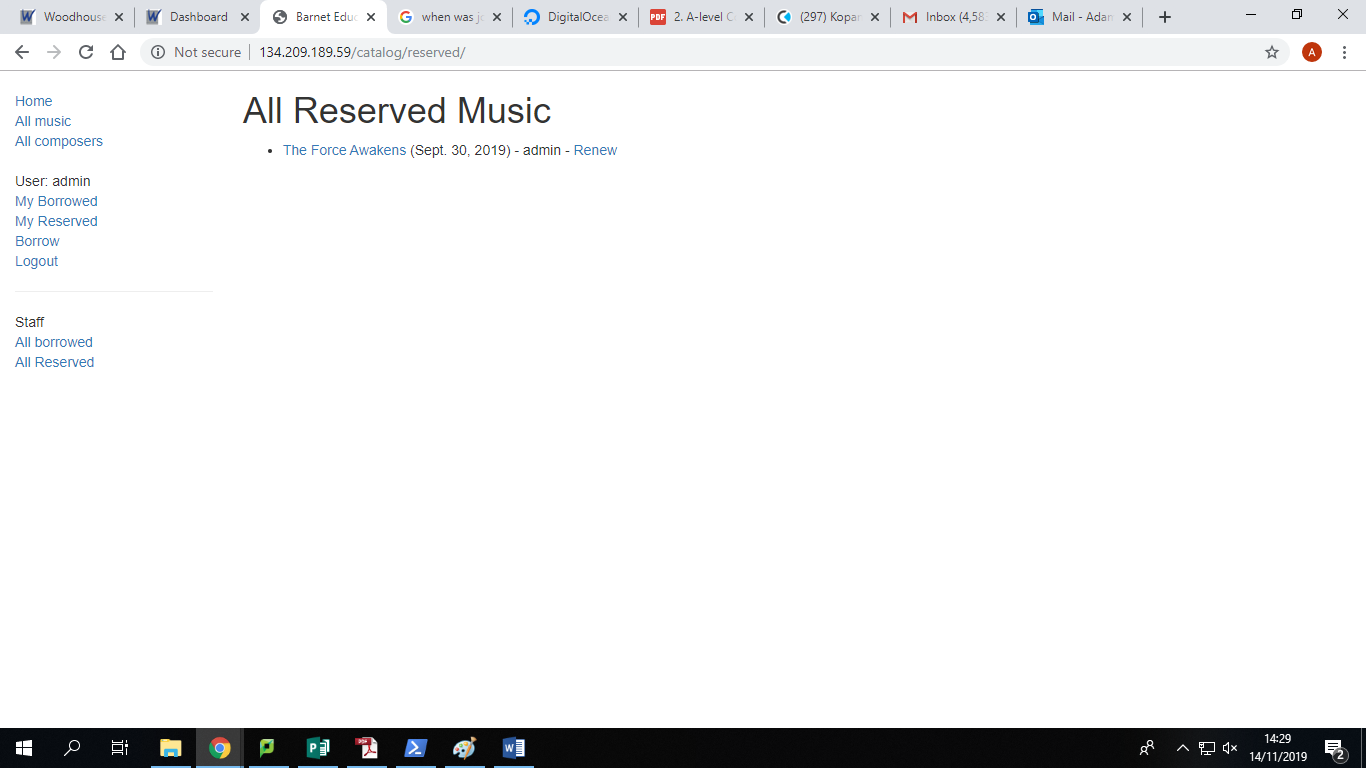
**The All Borrowed Page looks like this with music borrowed**

****

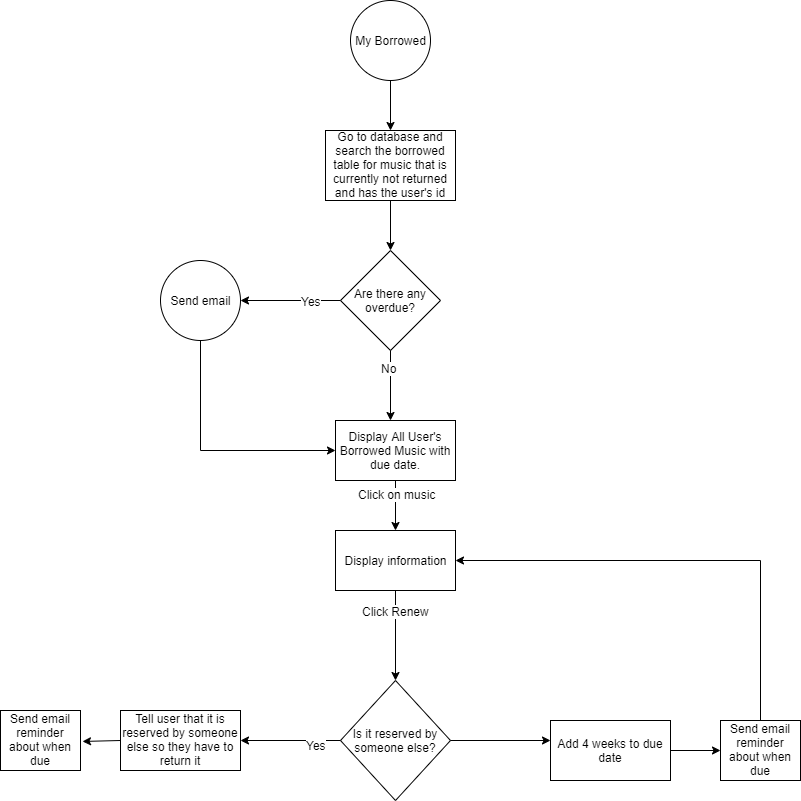
### All Reserved



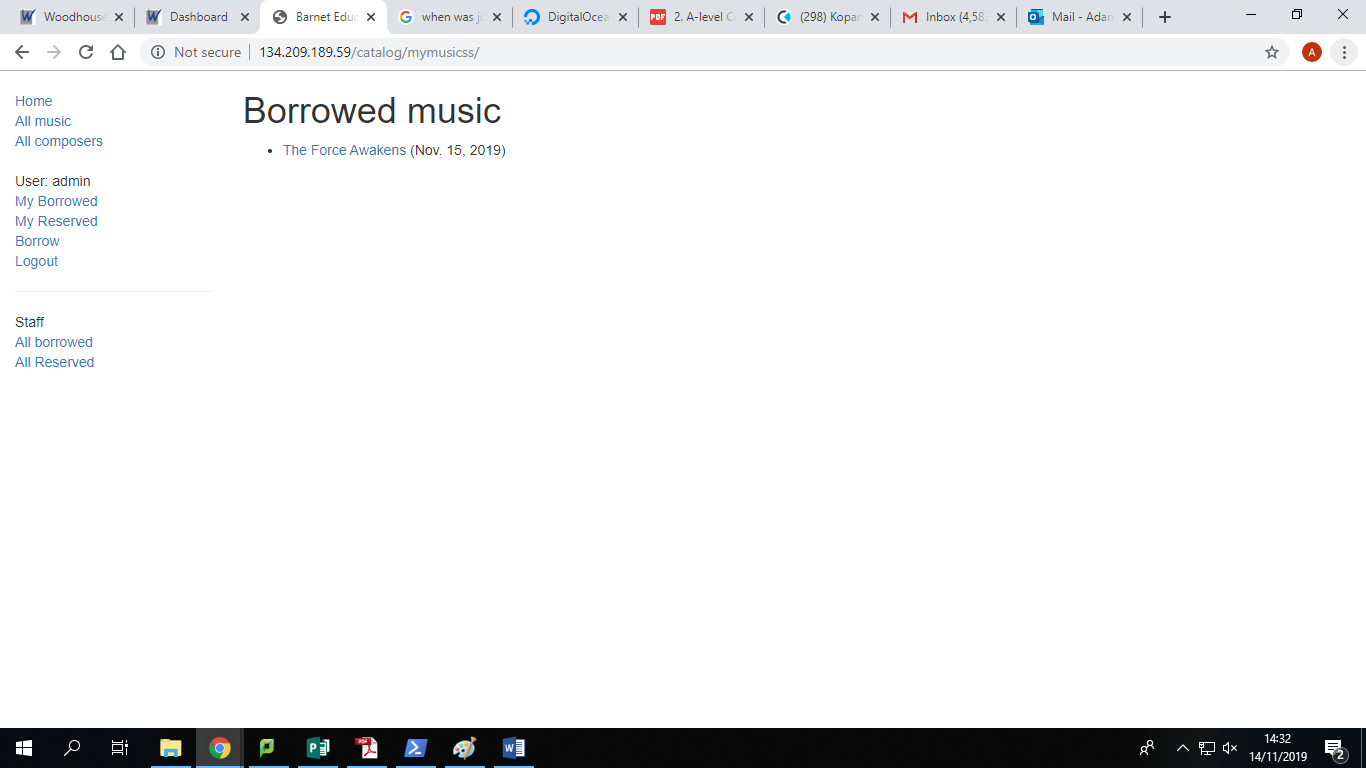
This is what the All Reserved Page looks like with reservations

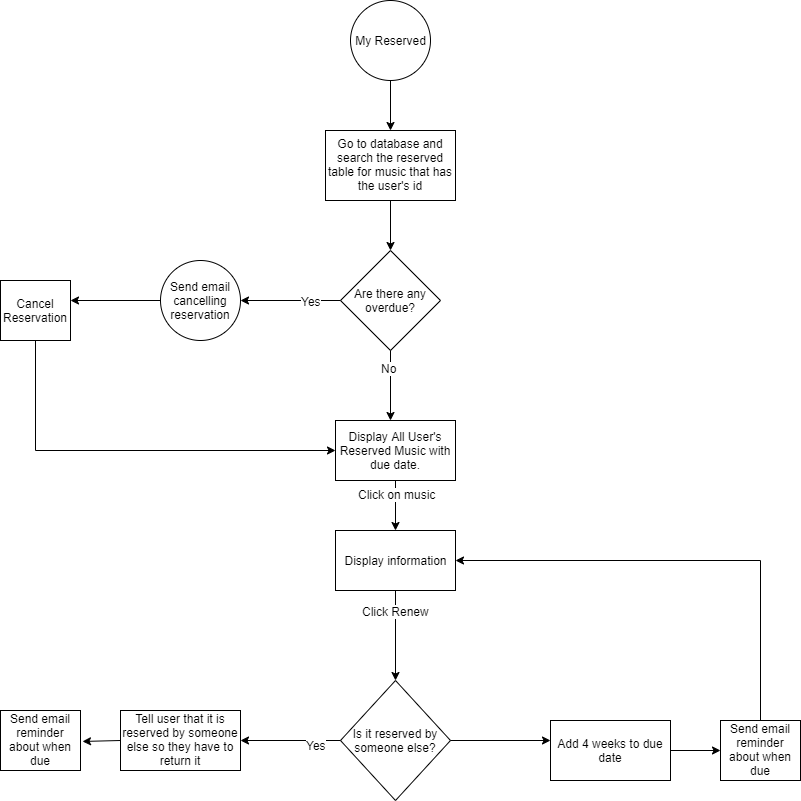


### My Borrowed

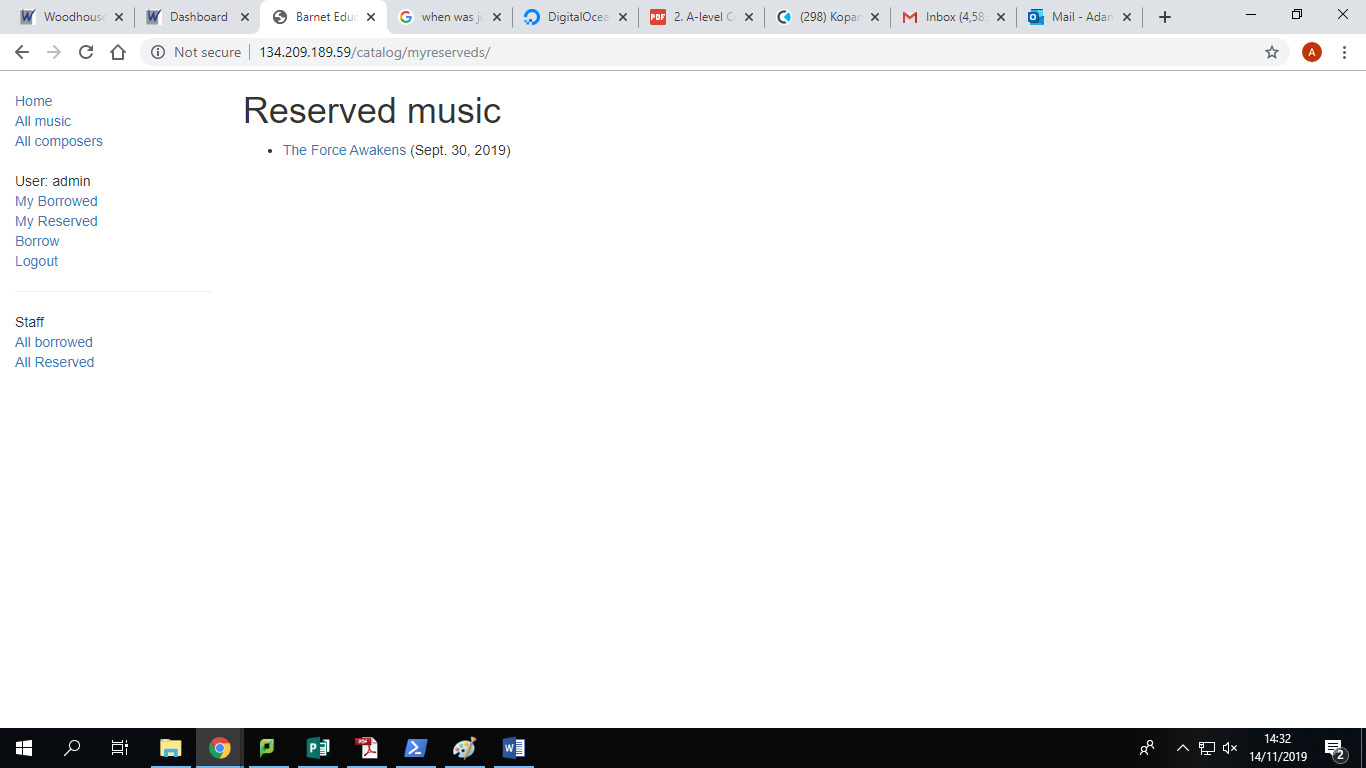
****

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

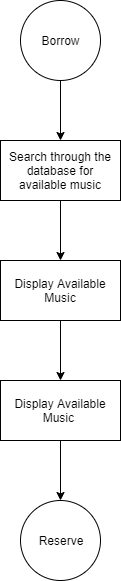


My Reserved****

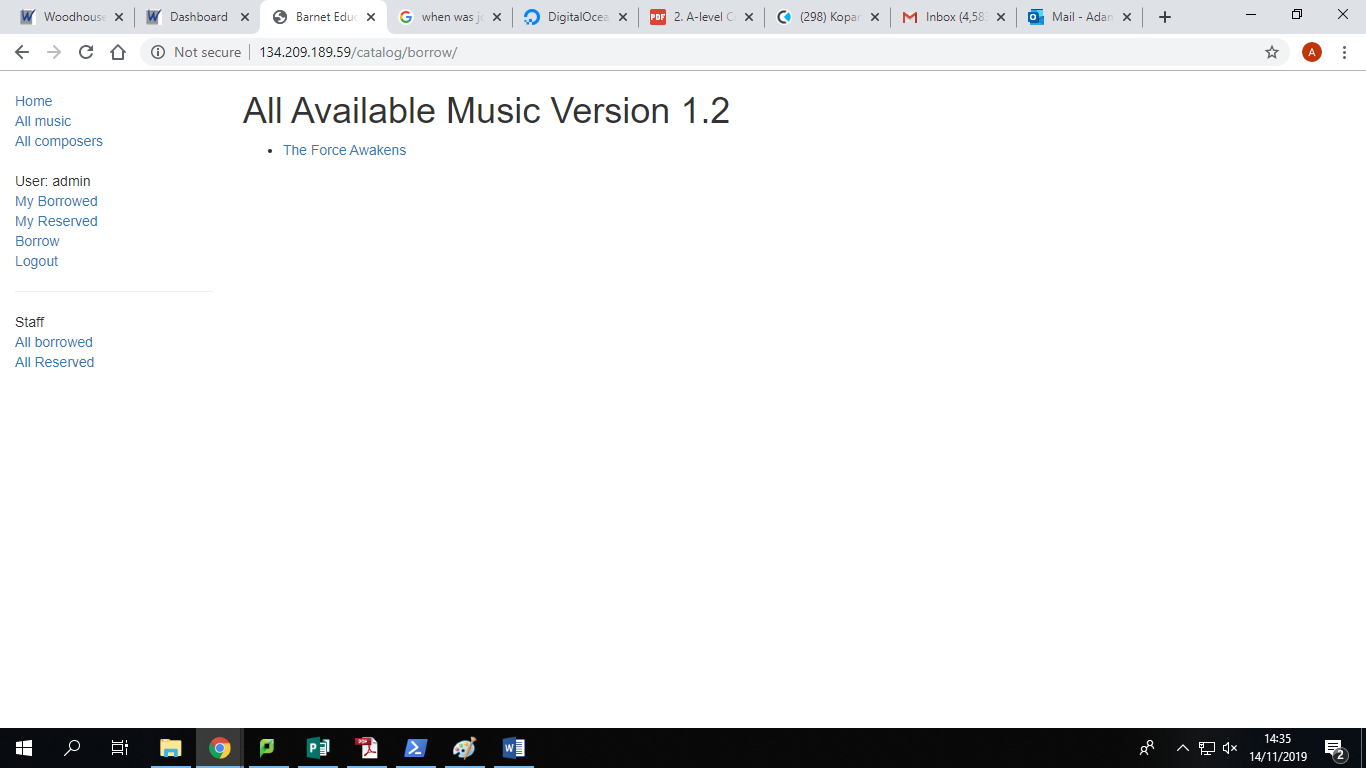
**This is what the My Reserved page looks like with reservations:**

****

### Borrow

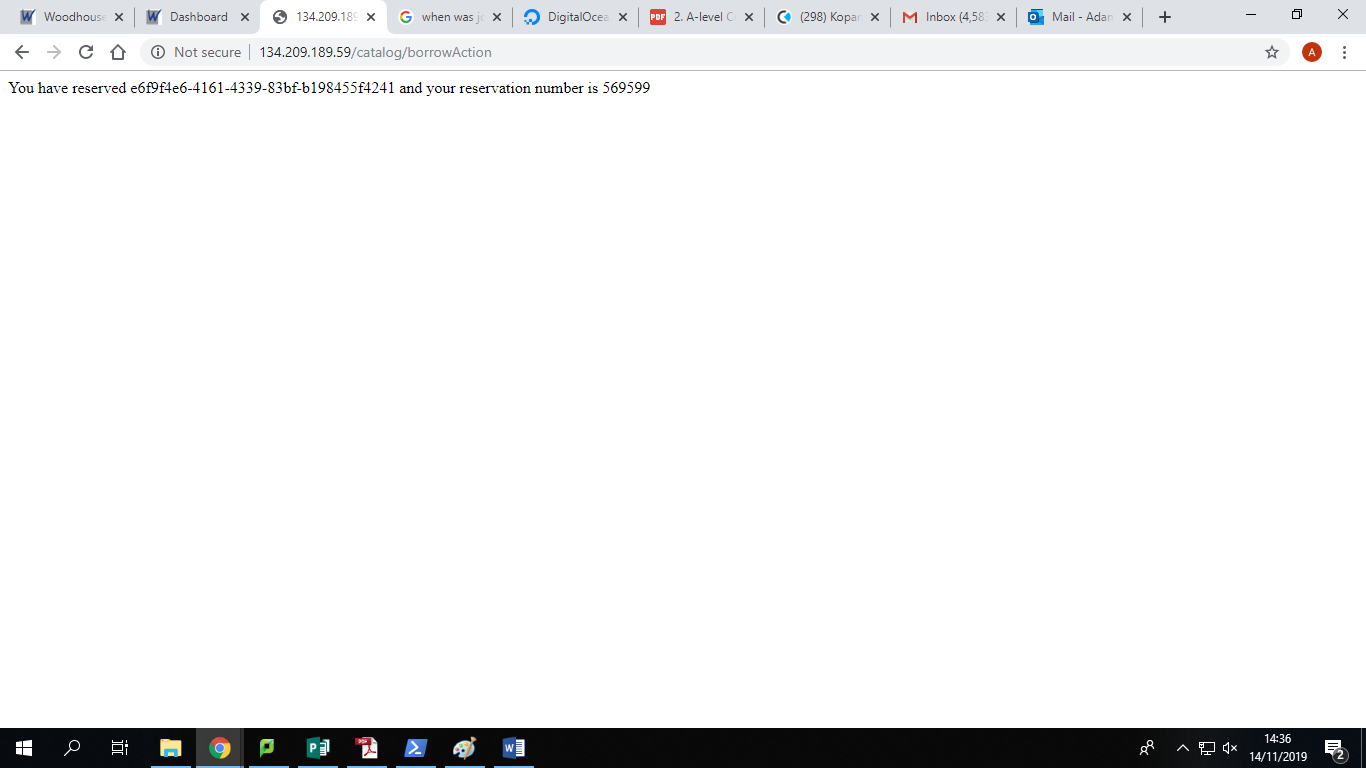
****

**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

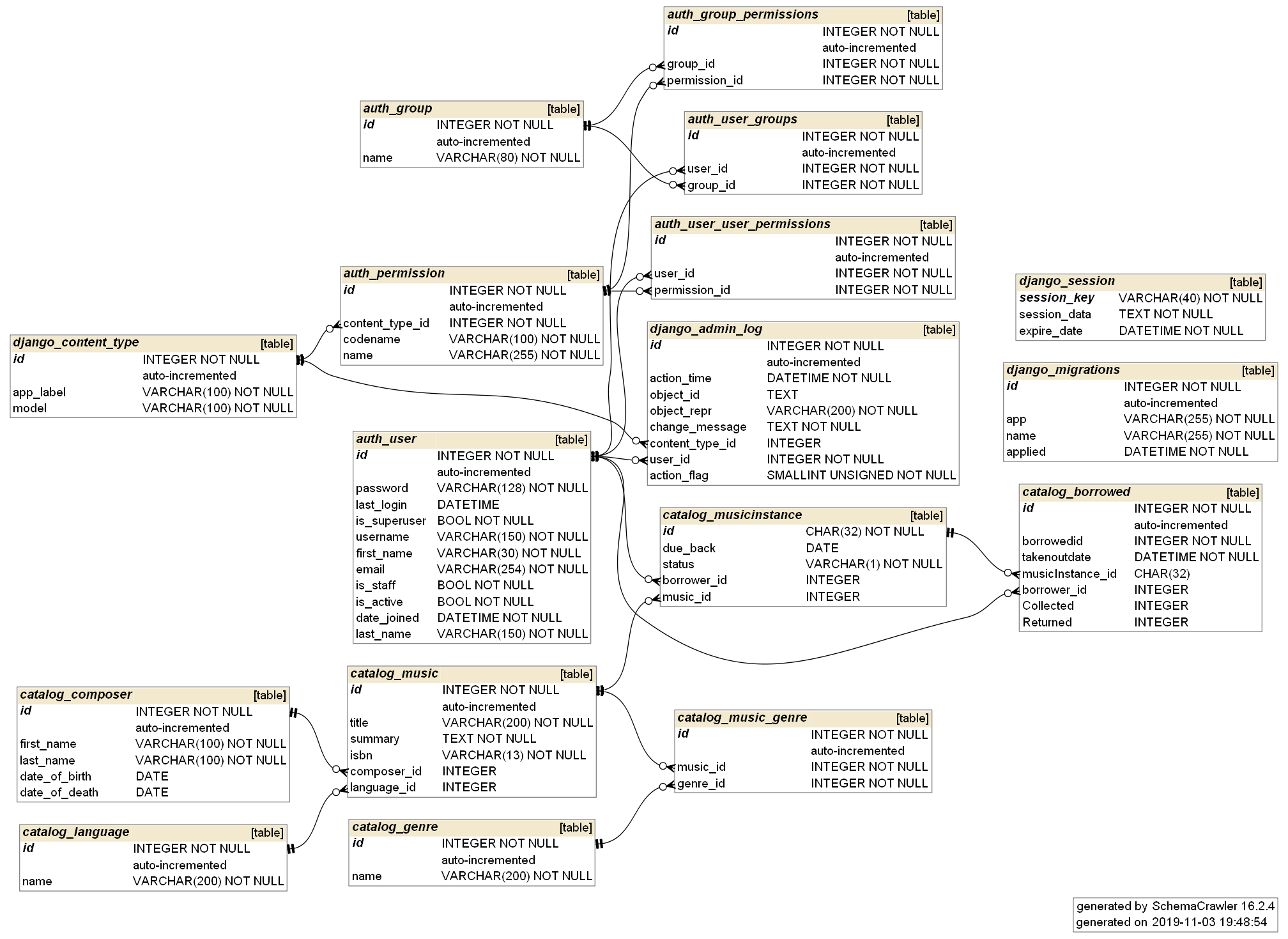
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Field Type | Field size | Purpose | Example data | Validation |
| id | char |  | Unique id to each music instance (UUID) |  | Is numerical |
| due\_back | Date |  | Due back date | YYYY-MM-DD | Correct Date Format |
| status | varchar | 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.