# 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 %}