# 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 | Type of Script | Name | Page | Description |
| Dictionaries | Javascript | intents.json | 5 | Used to teach the chatbot |
| Parsing json | Python | chatbot.py | 2 | Bringing in the questions and answers from the json dictionaries to teach the chatbot |
| Client-server model | Python | views.py |  | This is used to call information straight from my database into the website to be used later on |
| Complex Data Model | Python/SQLite3 | models.py | 6 | This is where I create all my tables with multiple crossovers |
| Neural Networks | Python | chatbot.py | 3 | This is what actually grabs the information from the parsing, splits it into words and then teaches the chatbot |
| API calls | 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()

from keep\_alive import keep\_alive

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

class Genre(models.Model):

"""Model representing a book genre (e.g. Science Fiction, Non Fiction)."""

name = models.CharField(

max\_length=200,

help\_text="Enter a book genre (e.g. Science Fiction, French Poetry etc.)"

)

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 Music(models.Model):

atomic = False

"""Model representing a book (but not a specific copy of a book)."""

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")

isbn = models.CharField('ISBN', max\_length=13,

help\_text='13 Character <a href="https://www.isbn-international.org/content/what-isbn'

'">ISBN number</a>')

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

import uuid # Required for unique music instances

from datetime import date

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)

@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})'.format(self.id, self.music.title)

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 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(default=timezone.now)

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

returned = models.CharField(max\_length=5, blank=True)

takenout = models.CharField(max\_length=5, blank = True)

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

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

from django.shortcuts import render

from django.views import View

# Create your views here.

from django.http import HttpResponse

from .models import Music, Composer, MusicInstance, Genre, MusicInstanceReservation

from django.template import loader

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

from datetime import date

from django.contrib.auth.models import User

from django.db.models import Exists, OuterRef

import datetime

from datetime import timedelta

from django.core.mail import send\_mail

from django.contrib.auth.models import User

def index(request):

"""View function for home page of site."""

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

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

# Render the HTML template index.html with the data in the context variable.

return render(

request,

'index.html',

context={'num\_music': num\_music, 'num\_instances': num\_instances,

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

'num\_visits': num\_visits},

)

from django.views import generic

class MusicListView(generic.ListView):

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

model = Music

paginate\_by = 10

class MusicDetailView(generic.DetailView):

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

model = Music

class ComposerListView(generic.ListView):

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

model = Composer

paginate\_by = 10

class ComposerDetailView(generic.DetailView):

"""Generic class-based detail view for an author."""

model = Composer

from django.contrib.auth.mixins import LoginRequiredMixin

class BorrowedUser(LoginRequiredMixin, generic.ListView):

model = Music

template\_name = 'catalog/music\_list\_borrowed\_user.html'

paginate\_by = 10

def get\_queryset(self):

is\_borrowed = MusicInstance.objects.filter(music=OuterRef('pk'),status\_\_exact='o').filter(borrower=self.request.user)

return Music.objects.annotate(is\_borrowed=Exists(is\_borrowed)).filter(is\_borrowed=True)

def BorrowedMusicDetail(request, pk):

if request.user.is\_authenticated:

username = request.user

template = loader.get\_template("catalog/borrowed\_music.html")

music=Music.objects.get(pk=pk)

reserved=music.musicinstance\_set.filter(status\_\_exact = 'o').filter(borrower=username)

context= {"music":music,"available":reserved}

return HttpResponse(template.render(context,request))

class ReservedUser(LoginRequiredMixin, generic.ListView):

model = Music

template\_name = 'catalog/music\_list\_reserved\_user.html'

paginate\_by = 10

def get\_queryset(self):

is\_borrowed = MusicInstance.objects.filter(music=OuterRef('pk'),status\_\_exact='r').filter(borrower=self.request.user)

return Music.objects.annotate(is\_borrowed=Exists(is\_borrowed)).filter(is\_borrowed=True)

def ReservedMusicDetail(request, pk):

if request.user.is\_authenticated:

username = request.user

template = loader.get\_template("catalog/reserved\_music.html")

music=Music.objects.get(pk=pk)

reserved=music.musicinstance\_set.filter(status\_\_exact = 'r').filter(borrower=username)

context= {"music":music,"available":reserved}

return HttpResponse(template.render(context,request))

# Added as part of challenge!

from django.contrib.auth.mixins import PermissionRequiredMixin

class LoanedMusicAllListView(PermissionRequiredMixin, generic.ListView):

"""Generic class-based view listing all books on loan. Only visible to users with can\_mark\_returned permission."""

model = MusicInstance

permission\_required = 'catalog.can\_mark\_returned'

template\_name = 'catalog/musicinstance\_list\_borrowed\_all.html'

paginate\_by = 10

def get\_queryset(self):

return MusicInstance.objects.filter(status\_\_exact='o').order\_by('due\_back')

'''class ReservedMusicAllListView(PermissionRequiredMixin, generic.ListView):

model = Music

permission\_required = 'catalog.can\_mark\_returned'

template\_name = 'catalog/musicinstance\_list\_reserved\_all.html'

paginate\_by = 10

def get\_queryset(self):

return MusicInstance.objects.filter(status\_\_exact='r').order\_by('due\_back')

'''

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

from django.http import HttpResponseRedirect

from django.urls import reverse

import datetime

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

# from .forms import RenewmusicForm

from catalog.forms import RenewMusicForm

@permission\_required('catalog.can\_mark\_returned')

def renew\_music\_librarian(request, pk):

"""View function for renewing a specific musicInstance by librarian."""

music\_instance = get\_object\_or\_404(MusicInstance, pk=pk)

# If this is a POST request then process the Form data

if request.method == 'POST':

# Create a form instance and populate it with data from the request (binding):

form = RenewMusicForm(request.POST)

# Check if the form is valid:

if form.is\_valid():

# process the data in form.cleaned\_data as required (here we just write it to the model due\_back field)

music\_instance.due\_back = form.cleaned\_data['renewal\_date']

music\_instance.save()

# redirect to a new URL:

return HttpResponseRedirect(reverse('all-borrowed'))

# If this is a GET (or any other method) create the default form

else:

proposed\_renewal\_date = datetime.date.today() + datetime.timedelta(weeks=3)

form = RenewMusicForm(initial={'renewal\_date': proposed\_renewal\_date})

context = {

'form': form,

'music\_instance': music\_instance,

}

return render(request, 'catalog/music\_renew\_librarian.html', context)

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

from django.urls import reverse\_lazy

from .models import Composer

class ComposerCreate(PermissionRequiredMixin, CreateView):

model = Composer

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

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

permission\_required = 'catalog.can\_mark\_returned'

class ComposerUpdate(PermissionRequiredMixin, UpdateView):

model = Composer

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

permission\_required = 'catalog.can\_mark\_returned'

class ComposerDelete(PermissionRequiredMixin, DeleteView):

model = Composer

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

permission\_required = 'catalog.can\_mark\_returned'

# Classes created for the forms challenge

class MusicCreate(PermissionRequiredMixin, CreateView):

model = Music

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

permission\_required = 'catalog.can\_mark\_returned'

class MusicUpdate(PermissionRequiredMixin, UpdateView):

model = Music

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

permission\_required = 'catalog.can\_mark\_returned'

class MusicDelete(PermissionRequiredMixin, DeleteView):

model = Music

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

permission\_required = 'catalog.can\_mark\_returned'

class Reserve(generic.ListView):

model = Music

template\_name = 'catalog/music\_list\_available\_all.html'

paginate\_by = 10

def get\_queryset(self):

is\_available = MusicInstance.objects.filter(music=OuterRef('pk'),status\_\_exact='a')

return Music.objects.annotate(is\_available=Exists(is\_available)).filter(is\_available=True)

class Borrow(generic.ListView):

model = Music

template\_name = 'catalog/music\_list\_reserved\_all.html'

paginate\_by = 10

def get\_queryset(self):

is\_reserved = MusicInstance.objects.filter(music=OuterRef('pk'),status\_\_exact='r')

return Music.objects.annotate(is\_reserved=Exists(is\_reserved)).filter(is\_reserved=True)

class Return(generic.ListView):

model = Music

template\_name = 'catalog/music\_list\_borrowed\_all.html'

paginate\_by = 10

def get\_queryset(self):

is\_borrowed = MusicInstance.objects.filter(music=OuterRef('pk'),status\_\_exact='o')

return Music.objects.annotate(is\_borrowed=Exists(is\_borrowed)).filter(is\_borrowed=True)

def BorrowMusicDetail(request, pk):

template = loader.get\_template("catalog/borrow\_music.html")

music=Music.objects.get(pk=pk)

reserved=music.musicinstance\_set.filter(status\_\_exact = 'r')

context= {"music":music,"available":reserved}

return HttpResponse(template.render(context,request))

def ReturnMusicDetail(request, pk):

template = loader.get\_template("catalog/return\_music.html")

music=Music.objects.get(pk=pk)

borrowed=music.musicinstance\_set.filter(status\_\_exact = 'o')

context= {"music":music,"available":borrowed}

return HttpResponse(template.render(context,request))

def ReserveMusicDetail(request, pk):

template = loader.get\_template("catalog/reserve\_music.html")

music=Music.objects.get(pk=pk)

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

context= {"music":music,"available":available}

return HttpResponse(template.render(context,request))

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

def ReserveAction(request):

whichCopy= request.POST['reservebutton']

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

reservationnumber = int(reservationnumber)

if request.user.is\_authenticated:

username = request.user

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

a.status = 'r'

a.due\_back = datetime.date.today() + timedelta(days=122)

a.borrower = username

a.save()

b=request.user

c= b.email

print(c)

p = MusicInstanceReservation(borrowedid = reservationnumber, musicInstance=a , takenoutdate = date.today(), userid=username, takenout= False)

p.save()

send\_mail(

'Music Reserved',

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

'adam@Bilkus.com',

[c])

return HttpResponse( ("You have reserved %s and your reservation number is %s") % (whichCopy, reservationnumber))

def BorrowAction(request):

whichCopy= request.POST['borrowbutton']

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

a.status = 'o'

a.due\_back = date.today() + timedelta(days=42)

a.save()

p = MusicInstanceReservation.objects.get(musicInstance = a, takenout = False)

p.due\_back = a.due\_back

p.takenout = True

p.takenoutdate=date.today()

p.returned=False

p.save()

userid = p.userid\_id

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

username=user.username

reservationnumber = p.borrowedid

email = user.email

send\_mail(

'Music Borrowed',

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

'adam@Bilkus.com',

[email])

return HttpResponse( ("%s has borrowed %s") % (username, whichCopy))

def bview(request):

return HttpResponse("Hello World")

def ReturnAction(request):

whichCopy= request.POST['returnbutton']

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

a.status = 'a'

a.due\_back = None

a.borrower = None

a.save()

p = MusicInstanceReservation.objects.get(musicInstance = a, takenout=True, returned= False)

c=p.borrowedid

p.returneddate= date.today()

p.returned = True

p.save()

d = p.userid\_id

e = User.objects.get(id = str(d))

f = e.email

send\_mail(

'Music Returned',

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

'adam@Bilkus.com',

[f])

return HttpResponse( ("You have returned %s ") % (whichCopy))

Most of the code here is called a view which is where I do a search of the database when a link is clicked then return it to a html file with Django logic to return to the user in a neat webpage.

### Forms

Forms are where I grab an idea from a view and can do work in the background without returning any information to the webpage.

from django.core.exceptions import ValidationError

from django.utils.translation import ugettext\_lazy as \_

import datetime # for checking renewal date range.

from django import forms

class RenewMusicForm(forms.Form):

"""Form for a librarian to renew books."""

renewal\_date = forms.DateField(

help\_text="Enter a date between now and 4 weeks (default 3).")

def clean\_renewal\_date(self):

data = self.cleaned\_data['renewal\_date']

# Check date is not in past.

if data < datetime.date.today():

raise ValidationError(\_('Invalid date - renewal in past'))

# Check date is in range librarian allowed to change (+4 weeks)

if data > datetime.date.today() + datetime.timedelta(weeks=4):

raise ValidationError(

\_('Invalid date - renewal more than 4 weeks ahead'))

# Remember to always return the cleaned data.

return data

## 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

# Register your models here.

from .models import Composer, Genre, Music, MusicInstance, Language, MusicInstanceReservation

admin.site.register(Genre)

admin.site.register(Language)

class MusicInline(admin.TabularInline):

"""Defines format of inline book insertion (used in AuthorAdmin)"""

model = Music

@admin.register(Composer)

class ComposerAdmin(admin.ModelAdmin):

"""Administration object for Author 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 books in author 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]

class MusicsInstanceInline(admin.TabularInline):

"""Defines format of inline book instance insertion (used in BookAdmin)"""

model = MusicInstance

class MusicAdmin(admin.ModelAdmin):

"""Administration object for Book models.

Defines:

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

- adds inline addition of book instances in book 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 BookInstance 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

## URLS

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

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

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

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

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

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

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

]

urlpatterns += [

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

path('reservedMusicDetailUser/<int:pk>', views.ReservedMusicDetail, name='reserved-music'),

path('borrowedMusicDetailUser/<int:pk>', views.BorrowedMusicDetail, name = 'borrowed-music'),

path('borrowed/', views.Return.as\_view(), name='return'), # Added for challenge

path('returnAction',views.ReturnAction, name='return\_action'),

path('borrowedMusicDetail/<int:pk>', views.ReturnMusicDetail, name='return-music'),

path('myreserveds/', views.ReservedUser.as\_view(), name='my-reserved'),

path('reserved/', views.Borrow.as\_view(), name='borrow'),

path('reservedMusicDetailAll/<int:pk>', views.BorrowMusicDetail,name='borrow-music'),

path('borrowAction',views.BorrowAction,name='borrow\_action')

]

# Add URLConf for librarian to renew a book.

urlpatterns += [

path('music/<uuid:pk>/renew/', views.renew\_music\_librarian, name='renew-music-librarian'),

]

# Add URLConf to create, update, and delete authors

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 books

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'),

]

urlpatterns += [

path('reserve/',views.Reserve.as\_view(),name='reserve'),

path('reserveMusicDetail/<int:pk>', views.ReserveMusicDetail,name='reserve-music'),

path('reserveAction',views.ReserveAction,name='reserve\_action')

]

## 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">

<a href="http://smartdailysaver.com/catalog/">

<h1 role='textbox' align="center"><img src="/static/images/BEAT-logo.png" width="200" height="200" alt="Italian Trulli"></img></h1>

</a>

<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://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

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

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

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

{% load static %}

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

<style>

body {

background-color: white;

}

html {

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>

<div class="container-fluid">

<div class="row">

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

{% block sidebar %}

<ul class="sidebar-nav">

<li><a href="{% url 'index' %}">Home</a></li>

<li><a href="{% url 'musics' %}">All music</a></li>

<li><a href="{% url 'composers' %}">All composers</a></li>

</ul>

<ul class="sidebar-nav">

{% if user.is\_authenticated %}

<li>User: {{ user.get\_username }}</li>

<li><a href="{% url 'my-borrowed' %}">My Borrowed</a></li>

<li><a href="{% url 'my-reserved' %}">My Reserved</a></li>

<li><a href="{% url 'reserve' %}">Reserve</a></li>

<li><a href="{% url 'logout'%}?next={{request.path}}">Logout</a></li>

{% else %}

<li><a href="{% url 'login'%}?next={{request.path}}">Login</a></li>

{% endif %}

</ul>

{% if user.is\_staff %}

<hr />

<ul class="sidebar-nav">

<li>Staff</li>

{% if perms.catalog.can\_mark\_returned %}

<li><a href="{% url 'return' %}">All borrowed</a></li>

<li><a href="{% url 'borrow' %}">All Reserved</a></li>

{% endif %}

</ul>

{% endif %}

{% endblock %}

</div>

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

{% 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>

</div>

</body>

</html>

This includes a few bits of logic allowing you to do requests for going to other webpages.

### Index/Homepage

{% 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>

<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>

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

This line is the API call to an embed for my chatbot.

{% endblock %}

### Borrow 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 'borrow\_action' %}" method="post">

{% csrf\_token %}

<table>

<thead></thead>

<tbody>

{% for copy in available %}

<tr>

<td>{{copy.id}}</td>

<td><button type="submit" name="borrowbutton" value="{{copy.id}}">Borrow</button></td>

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% endblock %}

This includes a button for borrowing and a link to the borrow page

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

{% 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>

{% csrf\_token %}

<table>

<thead></thead>

<tbody>

{% for copy in available %}

<tr>

<td>{{copy.id}} {{copy.due\_back}}</td>

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% endblock %}

### Reserve 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 'reserve\_action' %}" method="post">

{% csrf\_token %}

<table>

<thead></thead>

<tbody>

{% for copy in available %}

<tr>

<td>{{copy.id}}</td>

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

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% endblock %}

This includes the button for reserving and a link to the reservation action page

### Reserved Page for users to see their current reservations

{% 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>

{% csrf\_token %}

<table>

<thead></thead>

<tbody>

{% for copy in available %}

<tr>

<td>{{copy.id}} {{copy.due\_back}}</td>

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% 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>

</tr>

{% endfor %}

</tbody>

</table>

</form>

</div>

{% endblock %}

This includes the button for returning the music and a link to the returning action page

### Available Music

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

{% block content %}

<h1>All Available Music Version 1.2</h1>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{% url 'reserve-music' music.pk %}">{{music.title}}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There is no music available.</p>

{% endif %}

{% endblock %}

### All Borrowed Music

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

{% block content %}

<h1>All Borrowed Music Version 1.2</h1>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{% url 'return-music' music.pk %}">{{music.title}}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There is no music borrowed.</p>

{% endif %}

{% endblock %}

### All Reserved Music

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

{% block content %}

<h1>All Reserved Music Version 1.2</h1>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{% url 'borrow-music' music.pk %}">{{music.title}}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There is no music reserved.</p>

{% endif %}

{% endblock %}

### User Borrowed Music

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

{% block content %}

<h1>User Reserved Music Version 1.2</h1>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{% url 'borrowed-music' music.pk %}">{{music.title}}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There is no music borrowed.</p>

{% endif %}

{% endblock %}

### User Reserved Music

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

{% block content %}

<h1>User Reserved Music Version 1.2</h1>

{% if music\_list %}

<ul>

{% for music in music\_list %}

<li>

<a href="{% url 'borrowed-music' music.pk %}">{{music.title}}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There is no music reserved.</p>

{% endif %}

{% endblock %}

### 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 %}