THE BRIGADE SCHOOL@ JP NAGAR

PROJECT
Report
COMPUTER SCIENCE (083)

The TBS Box Office
Changing the way booking is done at the MLR Convention Center.

Name _____

Class _____

Roll No _____

THE BRIGADE SCHOOL
JP NAGAR BENGALURU

BONA FIDE CERTIFICATE

This is to certify that this project report entitled _____ _____ is a bona fide record of the project work done by _____ of class XII Reg. No_____ in the Computer Science Laboratory for the academic year 2023-24 This Project has been submitted in partial fulfilment of AISSCE for practical held at The Brigade School @ JP Nagar Bengaluru on _____

_____ _____

Internal Examiner **External Examiner**

School Stamp _____

Principal

ACKNOWLDEGMENT

"It is not possible to complete a project without assistance and encouragement of other people. This one is no exception"

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely fortunate to have got them all along to have my project completed. Whatever I have done is due to such guidance and assistance and I would not forget to thank them.

I would like to thank the principal, Ms Stella Parthasarathy, for being a constant source of support and encouragement.

I take this opportunity to express my profound gratitude and deep regards to my teacher Mrs. Divya Vikranth for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. Without your active guidance, cooperation, help and encouragement, I would not have made a head way in the project.

I extend my gratitude to The Brigade School@ J P Nagar for giving me this opportunity.

I also acknowledge with a deep sense of reverence, my gratitude towards my parents, my family members and friends, who have always supported me morally as well as economically.

Thanking You,

_____

INDEX

INTRODUCTION

In August of 2007, an idea struck three friends on holiday in South Africa. A few months on, their ideas had manifested into something more. Today, this idea is valued at over a billion USD with over two billion page views monthly. I am talking about the renowned BookMyShow website. It has always been a problem to get seats and tickets for a show. One would have to know the dates that the movie was to be shown, then hope to God that the tickets at their nearest theatre were not sold out and try to get tickets with their friends, praying that they would get seats that they wanted next to each other.

Today, anybody with access to the internet can seamlessly and easily book a ticket for a show and get verification online with none of these problems. The Brigade Schools have similar problems every year when holding an event for parents to attend. It is a hassle for the parents to get a good seat.

The school needs a system that allows each parent to have a seat for them to watch their children perform on stage and tackle the same problems that led to the creation of BookMyShow. There are also many specifications that the school requires and the program must be one that can flow with the needs of the school through the years ahead. Today, once again, three friends seek to solve this problem with some lines of code.

Getting a good seat in MLR for your child's event has always been a hassle. As it stands now, there is no way for parents to book/reserve their seats prior.Parents wait in line for hours to secure a good seat before others. It is quite competitive and exhaustive. If a parent enters late, they must look for an empty seat in the dark not knowing which seats they can occupy, all while blocking the view of the rest of the audience. The school has no way of tracking who is entering or knowing which seats have been occupied.

To make it easier for the school to manage and organize the crowds. To solve such problems. To ensure that each parent has a seat waiting for them as they enter and leave the auditorium with nothing but happy memories, we propose a website called 'TBS Box Office'.

The program provides the users with a hassle-free booking of seats of their preference for their child's events. It gives the school an easy way to keep track of the parents attending the events. It allows the parents to book seats remotely. For the school, all the seats aredistributed in a structured manner and are linked directly to the respective students' Unique Student Numbers.

MODULES AND PACKAGES

Virtualenv: A virtual environment within which we can run an instance.

Django: A batteries included package which is the backbone of the website.

Ezgmail: A package that sends emails through an API and helps automate the emails.

QRcode: A module that generates QR codes for our tickets.

DateTime: A module that helps us format dates.

JSON: A module that helps us convert dictionaries into JSON format to pass data to the front end.

Base64: A module that turns images into strings to be passed to the front end.

SPECIFICATIONS

Hardware:

A system containing minimum:

Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GH

64-bit operating system, x64-based processor

Software:

Python scripters such as PyCharm

Command Prompt/Windows Powershell

WindowsOS/MacOS/LinuxOS

FILES:

TBS_Box_Office

- events
  - __pycache__
    - __ init__.cpython-310.pyc (Created by Django)
    - admin.cpython-310.pyc (Created by Django)
    - apps.cpython-310.pyc (Created by Django)
    - models.cpython-310.pyc (Created by Django)
    - urls.cpython-310.pyc (Created by Django)
    - views.cpython-310.pyc (Created by Django)
  - migrations
    - __pycache__
      - __ init__.cpython-310.pyc (Created by Django)
      - admin.cpython-310.pyc (Created by Django)
      - apps.cpython-310.pyc (Created by Django)
      - models.cpython-310.pyc (Created by Django)
      - urls.cpython-310.pyc (Created by Django)
      - views.cpython-310.pyc (Created by Django)
    - __init__.py (Empty)
  - __init__.py (Empty)
  - admin.py (Created by Django)
  - apps.py (Created by Django)
  - models.py (Created by Django)
  - tests.py (Created by Django)
  - urls.py
  - views.py
- homepage
  - __pycache__
    - __ init__.cpython-310.pyc (Created by Django)
    - admin.cpython-310.pyc (Created by Django)
    - apps.cpython-310.pyc (Created by Django)
    - forms.cpython-310.pyc (Created by Django)

models.cpython-310.pyc (Created by Django)

urls.cpython-310.pyc (Created by Django)

views.cpython-310.pyc (Created by Django)

migrations

__pycache__

__ init__.cpython-310.pyc (Created by Django)

0001_initial.cpython-310.pyc (Created by Django)

0002_general.cpython-310.pyc (Created by Django)

0003_general_qrscanned_general_seatsbooked_and_more.cpython-310.pyc (Created by Django)

0004_linkage_maxseats.cpython-310.pyc (Created by Django)

0005_alter_linkage_maxseats.cpython-310.pyc (Created by Django)

0006_events_notifymail.cpython-310.pyc (Created by Django)

0007_events_siblingsbooked_alter_events_notifymail.cpython-310.pyc (Created by Django)

0008_alter_events_notifymail_alter_events_siblingsbooked.cpython-310.pyc (Created by Django)

__init__.py (Empty)

0001_inital.py (Created by Django)

0002_general.py (Created by Django)

0003_general_qrscanned_general_seatsbooked_and_more.py (Created by Django)

0004 _linkage_maxseats.py (Created by Django)

0005_alter_linkage_maxseats.py (Created by Django)

0006_events_notifymail.py (Created by Django)

0007_events_siblingsbooked_alter_events_notifymail.py (Created by Django)

0008_alter_events_notifymail_alter_events_siblingsbooked.py (Created by Django)

__init__.py (Empty)

admin.py

apps.py (Created by Django)

forms.py (Created by Django)

models.py

tests.py (Created by Django)

urls.py

views.py

new

__pycache__

__init__.cpython-310-pyc (Created by Django)

settings.cpython-310.pyc (Created by Django)

urls.cpython-310.pyc (Created by Django)

wsgi.cypython-310.pyc (Created by Django)

__init__.py (Empty)

asgi.py (Created by Django)

settings.py

urls.py

wsgi.py (Created by Django)

static (not included here, used for frontend design)

adminscript.js (javascript file for handling seats in admin pages.)

canceladminaudi.js (javascript file for admin's cancellation of seats.)

Lol.jpeg (background picture for login page.)

script.js (javascript file for seat booking by user.)

styles.css (hallplan styles sheet.)

templates (not included here, used for frontend design)

adminaudi.html (admin's hallplan page.)(linked to adminscript.js)

audi.html (user's hallplan page.)(linked to script.js)

cancelaudi.html (admin's cancellation page.)(linked to canceladminaudi.js)

details.html (Page that shows details when QR code scanned.)

firstpage.html (Website homepage)

Grep.html (General Report page for admin.)

hi.html	(Login page)
hi2.html	(Events page)
info.html	(Event report page for admin.)
linktree.html	(Page having links to backend admin management)
main.html	(Main body of all HTML files)
notify.html	(Displays a message for users.)
siblings.html	(Displays a message for users.)
ticket.html	(Ticket page containing QR Code.)
credentials.json	(Created for Gmail API authorization.) (Must update when expires.)
db.sqlite3	(Database created by Django)
manage.py	(Created by Django)
Pipfile	(Created for virtual environment)
Pipfile.lock	(Created for virtual environment)
token.json	(Created for sending emails)

SOURCE CODE:

TBS_Box_Office\events\urls.py:

```
from django.urls import path
from . import views
urlpatterns=[
    path('',views.home,name='events'),
    path('book/<str:pk>/',views.hallplan,name='hallplan'),
    path('ticket/<str:pk>/',views.ticket,name='ticket'),
    path('cancel/<str:pk>/',views.cancel,name='cancel'),
    path('seatdetails/<str:pk>/',views.seatdetails,name='seatdetails'),
    path('resend/<str:pk>/',views.resend,name='resend'),
    path('report/<str:pk>/',views.report,name='info'),
    path('reserve/<str:pk>/',views.reserve,name='reserve'),
    path('notify/<str:pk>/',views.notify,name='notify'),
    path('cancelnotify/<str:pk>/',views.cancelnotify,name='cancelnotify'),
    path('cancelreserve/<str:pk>/',views.cancelreserve,name='cancelreserve'),
]
```

TBS_Box_Office\events\views.py:

```
from datetime import *
from django.utils import timezone
from io import BytesIO
import json
from django.http import HttpResponse
from django.shortcuts import render,redirect
from homepage.models import linkage, Family, events, General
from django.contrib.auth.decorators import login_required,user_passes_test
import urllib.parse
import qrcode
import base64
#from django.core.mail import send_mail
import ezgmail

@login_required(login_url='home') #just in case.
def home(request):
    context={'reports':False}
    if request.user.is_superuser:
myevents=events.objects.all()
        context["reports"]=True
    else:
        GI, created = General.objects.get_or_create(pk=1)
GI.logins+=1
GI.save()
myevents=[i.event for i in request.user.linkage_set.all()]#here I get only the events that belong to the user that has currently logged in
    context['Events']=myevents
    return(render(request,'hi2.html',context))

@login_required(login_url='home') #just in case.
```

```
def hallplan(request,pk):
    context={'pk':pk}
    if request.user.is_superuser:
        return(redirect(f'/events/seatdetails/{pk}/'))
    else:
        try:
            event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing that event(foriegn key)s event(attribute(name))
        except:
            return(redirect('home'))
event.entered=event.entered+1
event.save()
        context['allbooked']=(len(event.blocked.split(','))-1==476) #-1 is due to last comma being extra. This is so that we can see if all seats are taken.
curlinkage=linkage.objects.get(user=request.user,event=event)
        if curlinkage.seats:#if already booked, send to ticket.
            return(redirect(f"/events/ticket/{pk}"))
        #Checking if the user has siblings that have booked in the same event.
        fam=Family.objects.get(user=request.user)

sib=linkage.objects.filter(fami__Parent1=fam.Parent1,fami__Parent2=fam.Parent2,fami__Guardians=fam.Guardians,event=event)
maxseats=curlinkage.maxseats
        for i in sib:
            if i.seats is not None:
                if curlinkage.event.siblingsbooked and f"{curlinkage.user.username}:{i.user.username}" not in curlinkage.event.siblingsbooked.split(','):
curlinkage.event.siblingsbooked+=f"{curlinkage.user.username}:{i.user.username},"
curlinkage.event.save()
                    return(render(request,'siblings.html',{'i':i}))
                else:
                    return(render(request,'siblings.html',{'i':i}))
            if event not in [i.event for i in request.user.linkage_set.all()]: #just in case
```

```
            return(redirect('home'))
        if request.method == "POST":
                seats = request.POST.get('seats')  # Get selected seat IDs from POST data
                seats=seats.strip("")+','
                event=(request.user.linkage_set.get(event__event=pk)).event#refreshing, to check real time, if anyone has booked that justt before this person booked.
                if seats in event.blocked:#If somehow the seats have actually gotten booked just before the person clicked book, then reload the page.
                    return(redirect(f"/events/{event.red}"))
curlinkage.seats=seats
curlinkage.save()
                taken=event.blocked
                if taken == None:
                    taken=seats
                else:
                    taken+=seats
event.blocked=taken
event.save()
                GI, created = General.objects.get_or_create(pk=1)
GI.SeatsBooked+=1
GI.save()
                return(redirect(f"/events/ticket/{pk}"))
        #see if you can make this more efficient.

d={"maxseats":maxseats,"A":[],"B":[],"C":[],"D":[],"E":[],"F":[],"G":[],"H":[],"I":[],"J":[],"K":[],"L":[],"M":[],"N":[],"O":[],"AA":[],"BB":[],"CC":[],"DD":[],"EE":[],"FF":[]}
        #keeping default empty values for all so that it can easily be accesed in js without having to worry wether the key exists.
        s=event.blocked
        l=s.split(',')
l.pop()
        for i in l:
            for j in range(len(i)):
                if i[j].isdigit():
```

```
                try:
                    temp=d[i[:j]]
temp.append(int(i[j:]))
                    d[i[:j]]=temp
                except:
                    #thisshouldnt happen but just in case the admin has entered some bad data in the blocked.
                    d[i[:j]]=[int(i[j:])]
                break
        d=json.dumps(d).replace("'", '"') #replaces the " in the dict to "" so that we can pass it to the js (through the html.)
        context['blocked']=d
        context['event']=event
        return(render(request,'audi.html',context))


@login_required(login_url='home')
def ticket(request,pk):
    try:
        event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing that event(foriegn key)s event(attribute(name))
    except:
        return(redirect('home'))
curlinkage=linkage.objects.get(user=request.user,event=event)
    if event not in [i.event for i in request.user.linkage_set.all()]:
        return(redirect('home'))
    if curlinkage.seats==None:#if not booked, send home.
        return(redirect('home'))
    if event.notifymail and request.user.email in event.notifymail.split(','): #if the user has asked to notify and has now gotten a seat/seats, remove his name from notify.
        temp=event.notifymail
event.notifymail=temp.replace(request.user.email,'')
event.save()
    url="127.0.0.1:8000/"+urllib.parse.quote(f'details/{curlinkage}')#HERE, THE 127.0.0.1:8000 WILL HAVE TO BE CHANGED TO THE DOMAIN.
```

```
img=qrcode.make(url) #this is a PILImage.
    buffered = BytesIO()#this will be the image as a string. need the bytes as that way, i can encode the pilimage into bytes
    #and then decode that into a string.
img.save(buffered) #saving image into buffered. (as bytes prolly.)
image_str = base64.b64encode(buffered.getvalue()).decode() #making the image into a string that can be sent and rendered in ticket.

context={'Event':event,'EventDeets':event.Date,'desc':event.Desc,'seats':curlinkage.seats,'bookedwhen':curlinkage.whenbooked.strftime("%d/%m/%Y, %H:%M:%S"),'QR':image_str,'pk':pk,'email':request.user.email}
    if curlinkage.emailsent==None:#if email has not been sent, send an email.
        #Sending the email from suadnastorage.
        subject=f'Your ticket for {event} on {event.Date}'
        message=f"Ticket: \n\n Link: 127.0.0.1:8000/events/ticket/{urllib.parse.quote(pk)}/ \n\n Event: {event} \n\n Date and time: {event.Date} \n\n Description: {event.Desc} \n\n Your seats: {curlinkage.seats} \n\n For QR code, please visit the link. \n\n Booked on: {curlinkage.whenbooked.strftime('%d/%m/%Y, %H:%M:%S')} \n\n For any queries, please contact the school."
        #fromwhom='suadnastorage@gmail.com'
        recipients=request.user.email
ezgmail.send(recipients, subject=subject, body=message)
        GI, created = General.objects.get_or_create(pk=1)
GI.emailsent+=1
GI.save()
        #send_mail(subject, message, fromwhom, recipients,fail_silently=False)
curlinkage.emailsent=timezone.now()
curlinkage.save()
    return(render(request,'ticket.html',context))


@login_required(login_url='home')
def resend(request,pk):
    try:
        event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing that event(foriegn key)s event(attribute(name))
    except:
```

```
        return(redirect('home'))
curlinkage=linkage.objects.get(user=request.user,event=event)
    if event not in [i.event for i in request.user.linkage_set.all()]:
        return(redirect('home'))
    if curlinkage.seats==None:#if not booked, send home.
        return(redirect('home'))
    #Sending the email from suadnastorage.
    subject=f'Your ticket for {event} on {event.Date}'
    message=f"Ticket: \n\n Link: 127.0.0.1:8000/events/ticket/{urllib.parse.quote(pk)}/ \n\n Event: {event} \n\n Date and time: {event.Date} \n\n Description: {event.Desc} \n\n Your seats: {curlinkage.seats} \n\n For QR code, please visit the link. \n\n Booked on: {curlinkage.whenbooked.strftime('%d/%m/%Y, %H:%M:%S')} \n\n For any queries, please contact the school."
    #fromwhom='suadnastorage@gmail.com'
    recipients=request.user.email
ezgmail.send(recipients, subject=subject, body=message)
    GI, created = General.objects.get_or_create(pk=1)
GI.emailsent+=1
GI.save()
    #send_mail(subject, message, fromwhom, recipients,fail_silently=False)
curlinkage.emailsent=timezone.now()
curlinkage.save()
    return(redirect(f'/events/ticket/{pk}/'))


@login_required(login_url='home')
def cancel(request,pk):
    try:
        event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing that event(foriegn key)s event(attribute(name))
    except:
        return(redirect('home'))
curlinkage=linkage.objects.get(user=request.user,event=event)
    if event not in [i.event for i in request.user.linkage_set.all()]:
        return(redirect('home'))
```

```
    if curlinkage.seats==None:#if not booked, send home.
        return(redirect('home'))
    if curlinkage.seats not in event.blocked:#If its not there, we dont have to cancel anything. (all these are just fallbacks, JICs)
        return(redirect('home'))
event.blocked=event.blocked.replace(curlinkage.seats,'')#removing those seats from events blocked seats.
curlinkage.seats=None#removing seats from curlinkage.
curlinkage.emailsent=None
ezgmail.send(request.user.email, subject=f"Cancellation of your seats for {event} on {event.Date}", body=f"Your seats for {event} have been cancelled.")
curlinkage.save()
event.cancels=event.cancels+1
event.save()
    GI, created = General.objects.get_or_create(pk=1)
GI.Seatscancelled+=1
GI.save()


    if event.notifymail:
        for i in [j for j in event.notifymail.split(',') if j not in ' ,']:#now we send emails to all those who have been asked to be notified.
ezgmail.send(i, subject=f"Availability of seats for {event}.", body=f"This is to notify you that a seat may be available for {event} as someone has cancelled their seats.\n\n If you do not wish to be notified about this further, please sign in and click this link: 127.0.0.1:8000/events/cancelnotify/{urllib.parse.quote(pk)}")
    return(redirect('home'))#can change this if wanted.


@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def seatdetails(request,pk):
    event=(events.objects.get(event=pk))
deets={}


d={"A":[],"B":[],"C":[],"D":[],"E":[],"F":[],"G":[],"H":[],"I":[],"J":[],"K":[],"L":[],"M":[],"N":[],"O":[],"AA":[],"BB":[],"CC":[],"DD":[],"EE":[],"FF":[],"SA":[],"SB":[],"SC":[],"SD":[],"SE":[],"SF":[],"SG":[],"SH":[],"SI":[],"SJ":[],"SK":[],"SL":[],"SM":[],"SN":[],"SO":[],"SAA
```

":[],"SBB":[],"SCC":[],"SDD":[],"SEE":[],"SFF":[],"RA":[],"RB":[],"RC":[],"RD":[],"RE":[]
,"RF":[],"RG":[],"RH":[],"RI":[],"RJ":[],"RK":[],"RL":[],"RM":[],"RN":[],"RO":[],"RAA":[],
"RBB":[],"RCC":[],"RDD":[],"REE":[],"RFF":[]}

#keeping default empty values for all so that it can easily be accesed in js without having to worry wether the key exists.

l=event.blocked.split(',')

l.pop()

for i in l: #traversing the list of booked seats

link= [k for k in linkage.objects.filter(event=event) if k.seats!=None and i in k.seats.split(',')] #contains only the linkage which booked that seat.

if link==[]: #reserved seats

for j in range(len(i)):

#rendering seats as usual

if i[j].isdigit():

try:

temp=d[f"R{i[:j]}"]

temp.append(int(i[j:]))

d[f"R{i[:j]}"]=temp

except:

#thisshouldnt happen but just in case the admin has entered some bad data in the blocked.

d[f"R{i[:j]}"]=[int(i[j:])]

break

deets[i]=None #if this seat is blocked but not through a linkage, then set it as none

eliflink!=[]: #Non reserved seats

link=link[0] #else, set deets[that seat] as the linkage which was used to book it.

deets[i]=json.dumps({"event":link.event.event,"USN":link.user.username,"group":f"{link.grp
.group}","family":f"{link.fami.Parent1}-{link.fami.Parent2}-
{link.fami.Guardians}".replace("None","").replace("--", "-").strip("-
"),"seats":link.seats,"whenbooked":str(link.whenbooked.strftime("%d/%m/%Y,
%H:%M:%S")),"whenmade":str(link.created.strftime("%d/%m/%Y,
%H:%M:%S")),"details":link.details,'emailsent':str(link.emailsent.strftime("%d/%m/%Y,
%H:%M:%S")),'scannedon':str(link.scanned)}).replace("'", "")

if link.scanned!=None: #scanned seats

for j in range(len(i)):

```
                #rendering seats as usual
                if i[j].isdigit():
                    try:
                        temp=d[f"S{i[:j]}"]
temp.append(int(i[j:]))
                        d[f"S{i[:j]}"]=temp
                    except:
                        #thisshouldnt happen but just in case the admin has entered some bad data
in the blocked.
                        d[f"S{i[:j]}"]=[int(i[j:])]
                    break
            else: #Booked but not scanned seats
                for j in range(len(i)):
                    #rendering seats as usual
                    if i[j].isdigit():
                        try:
                            temp=d[i[:j]]
temp.append(int(i[j:]))
                            d[i[:j]]=temp
                        except:
                            #thisshouldnt happen but just in case the admin has entered some bad data
in the blocked.
                            d[i[:j]]=[int(i[j:])]
                        break
    #deets now contains all the booked seats of that event: who booked them. Now, we take
this to JS (through hidden form) then there, we call the deets[selectedseat] which is the
linkage and then get all the data of the seat from there and display it.
    # d is just so that we can pass the same booked seats of that event in a nice form so aditi
can go make the booked things.
    context={"deets":json.dumps(deets).replace("""", ""),"blocked":json.dumps(d).replace("""",
""),"event":event}#replaces the " in the dict to "" so that we can pass it to the js (through the
html.)
    return(render(request,'adminaudi.html',context))
```

```
@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def report(request,pk):
    event=(events.objects.get(event=pk))
    links=[k for k in linkage.objects.filter(event=event) if k.seats!=None]
    context={'whenupdated':event.updatedon.strftime("%d/%m/%Y, %H:%M:%S"),'scanned':event.scanned,'blocked':len(event.blocked.split(',')),'pplsbooked':len(links),'clicked':event.entered,'cancelled':event.cancels,"event":event.event,'nowaiting':0,'nosiblings':0}
    if event.siblingsbooked:
        context['nosiblings']=len([i for i in event.siblingsbooked.strip('\' ,'').split(',') if i not in '\' ,'])
    if event.notifymail:
        context['notiymail']=len([i for i in event.notifymail.split(',') if i not in ' ,'])
deets=[]
    dates={}
    for link in links:
        if dates.get(str(link.whenbooked.date())):
            dates[str(link.whenbooked.date())]+=1
        else:
            dates[str(link.whenbooked.date())]=1

deets.append({"USN":link.user.username,"seats":link.seats,"group":f"{link.grp.group}","family":f"{link.fami.Parent1}-{link.fami.Parent2}-{link.fami.Guardians}".replace("None","").replace("--", "-").strip("-"),"whenbooked":str(link.whenbooked.strftime("%d/%m/%Y, %H:%M:%S")),"whenmade":str(link.created.strftime("%d/%m/%Y, %H:%M:%S")),"details":link.details,'emailsent':str(link.emailsent.strftime("%d/%m/%Y, %H:%M:%S")),'scan':str(link.scanned)})
    context['deets']=deets
    l=dates.items()
    dates=[]
    for i in l:
        temp=[]
        for j in i:
temp.append(j)
```

```
dates.append(temp)
    context['dates']=dates
return(render(request, 'info.html',context))


@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def reserve(request,pk): #this was a last minute addition, literally about to have the meeting when i realised this feature could be nice. It is just a workaround, there is a much more efficient way of doin this where it is just in the hallplan() itself.
    context={'pk':pk} #Here, were just blocking the seats as an admin.
    event=(events.objects.get(event=pk))
    if request.method == "POST":
        seats = request.POST.get('seats')  # Get selected seat IDs from POST data
        seats=seats.strip("")+','
        event=(events.objects.get(event=pk))#refreshing, to check real time, if anyone has booked that justt before this person booked.
        if seats in event.blocked:#If somehow the seats have actually gotten booked just before the person clicked book, then reload the page.
            return(redirect(f"/events/{event.red}"))
        taken=event.blocked
        if taken == None:
            taken=seats
        else:
            taken+=seats
event.blocked=taken
event.save()
        return(redirect('home'))
    #see if you can make this more efficient.

d={"maxseats":69420,"A":[],"B":[],"C":[],"D":[],"E":[],"F":[],"G":[],"H":[],"I":[],"J":[],"K":[],"L":[],"M":[],"N":[],"O":[],"AA":[],"BB":[],"CC":[],"DD":[],"EE":[],"FF":[]}
    #keeping default empty values for all so that it can easily be accesed in js without having to worry wether the key exists.
    s=event.blocked
    l=s.split(',')
```

```
l.pop()
    for i in l:
        for j in range(len(i)):
            if i[j].isdigit():
                try:
                    temp=d[i[:j]]
temp.append(int(i[j:]))
                    d[i[:j]]=temp
                except:
                    #thisshouldnt happen but just in case the admin has entered some bad data in the
blocked.
                    d[i[:j]]=[int(i[j:])]
                break
    d=json.dumps(d).replace("'", '"') #replaces the " in the dict to "" so that we can pass it to the
js (through the html.)
    context['blocked']=d
    context['event']=event
    return(render(request,'audi.html',context))


@login_required(login_url='home') #this function is another last minute added feauture. If all
seats are booked for that event,
def notify(request,pk): #it will send an email to whoever clicks notify my whenever someone
cancels and the event gets free.
    #this function is only to add the email ID to notify (in event model). the sending of the
emails will happen in cancel.
    context={}
    try:
        event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing
that event(foriegn key)s event(attribute(name))
    except:
        return(redirect('home'))
    #confirming that it is really full.
    context['allbooked']=(len(event.blocked.split(','))-1==476) #-1 is due to last comma being
extra. This is so that we can see if all seats are taken.
```

```
curlinkage=linkage.objects.get(user=request.user,event=event)
    #confirming that the criteria for inform is met. i.e They must not have aldready booked a ticket and they must not have siblings who have booked it.
    #there will be a check in ticket to remove any and all emails that have booked from inform.
    if curlinkage.seats:#if aldready booked, send to ticket.
        return(redirect(f"/events/ticket/{pk}"))
    #checking for siblings.
    fam=Family.objects.get(user=request.user)

sib=linkage.objects.filter(fami__Parent1=fam.Parent1,fami__Parent2=fam.Parent2,fami__Guardians=fam.Guardians,event=event)
    for i in sib:
        if i.seats is not None:
curlinkage.event.siblingsbooked+=f"{curlinkage.user.username}:{i.user.username},"
curlinkage.event.save()
            return(render(request,'siblings.html',{'i':i}))
        if event not in [i.event for i in request.user.linkage_set.all()]: #just in case
            return(redirect('home'))
    #checking if aldready asked for this.
    if event.notifymail and request.user.email not in event.notifymail.split(','):
        #adding to list.
event.notifymail+=request.user.email+','
event.save()
    context['email']=request.user.email
    return(render(request,'notify.html',context))
@login_required(login_url='home')
def cancelnotify(request,pk):
    context={}
    try:
        event=(request.user.linkage_set.get(event__event=pk)).event #event__event is accessing that event(foriegn key)s event(attribute(name))
    except:
        return(redirect('home'))
```

```
    if event.notifymail and request.user.email in event.notifymail.split(','): #if the user has asked to notify and has now gotten a seat/seats, remove his name from notify.
        temp=event.notifymail
event.notifymail=temp.replace(f"{request.user.email},","")
event.save()
    return(redirect('home'))


@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def cancelreserve(request,pk):
    context={'pk':pk} #Here, were just blocking the seats as an admin.
    event=(events.objects.get(event=pk))
    if request.method == "POST":
        seats = request.POST.get('seats')  # Get selected seat IDs from POST data
        seats=seats.strip("")+','
        taken=event.blocked
        for i in seats.strip(' ,').split(','):
            taken=taken.replace(f'{i},',"")#removing those seats from events blocked seats.
event.blocked=taken
event.save()
        return(redirect('home'))


d={"RA":[],"RB":[],"RC":[],"RD":[],"RE":[],"RF":[],"RG":[],"RH":[],"RI":[],"RJ":[],"RK":[],"RL":[],"RM":[],"RN":[],"RO":[],"RAA":[],"RBB":[],"RCC":[],"RDD":[],"REE":[],"RFF":[]}
    #keeping default empty values for all so that it can easily be accesed in js without having to worry wether the key exists.
    l=event.blocked.split(',')
l.pop()
    for i in l: #traversing the list of booked seats
        link= [k for k in linkage.objects.filter(event=event) if k.seats!=None and i in k.seats.split(',')] #contains only the linkage which booked that seat.
        if link==[]: #reserved seats
            for j in range(len(i)):
```

```
            #rendering seats as usual
            if i[j].isdigit():
                try:
                    temp=d[f"R{i[:j]}"]
temp.append(int(i[j:]))
                    d[f"R{i[:j]}"]=temp
                except:
                    #thisshouldnt happen but just in case the admin has entered some bad data in the blocked.
                    d[f"R{i[:j]}"]=[int(i[j:])]
                break
    if event.notifymail:
        for i in [j for j in event.notifymail.split(',') if j not in ' ,']:#now we send emails to all those who have been asked to be notified.
ezgmail.send(i, subject=f"Availability of seats for {event}.", body=f"This is to notify you that a seat may be available for {event} as someone has cancelled their seats.\n\n If you do not wish to be notified about this further, please sign in and click this link: 127.0.0.1:8000/events/cancelnotify/{urllib.parse.quote(pk)}")
    #deets now contains all the booked seats of that event: who booked them. Now, we take this to JS (through hidden form) then there, we call the deets[selectedseat] which is the linkage and then get all the data of the seat from there and display it.
    # d is just so that we can pass the same booked seats of that event in a nice form so aditi can go make the booked things.
    context={"blocked":json.dumps(d).replace("'", "")}#replaces the " in the dict to "" so that we can pass it to the js (through the html.)
    context['event']=event
    return(render(request,'cancelaudi.html',context))
```

TBS_Box_Office\homepage\admin.py:

```
from django.contrib import admin
from .models import events,linkage,Family,GroupEventLink,General
from django.contrib.auth.admin import UserAdmin
from django.urls import reverse
from django.http import HttpResponseRedirect
from django.contrib.auth.models import User
# Register your models here.


admin.site.register(events)
admin.site.register(GroupEventLink)


class CustomUserAdmin(UserAdmin):
    def response_change(self, request, obj) ->HttpResponseRedirect: # if you get an error on this line remove the type annotations
        try:
            if obj.me:#checking if users family has aldready been made
                return HttpResponseRedirect(reverse("admin:homepage_family_change", args=[obj.me.id]))
        except:
            Family.objects.create(user=obj) # creates a family object with everything None except the onetoone
            return HttpResponseRedirect(reverse("admin:homepage_family_change", args=[obj.me.id]))
admin.site.unregister(User)
admin.site.register(User, CustomUserAdmin)


admin.site.register(General)
admin.site.register(Family)
admin.site.register(linkage)
```

TBS_Box_Office\homepage\models.py:

```
from django.db import models as m
from django.contrib.auth.models import User
from django.db.models.signals import post_save,pre_delete
from django.dispatch import receiver
import urllib.parse


class General(m.Model):
    logins=m.BigIntegerField(default=0)
    logouts=m.BigIntegerField(default=0)
    emailsent=m.BigIntegerField(default=0)
    QRscanned=m.BigIntegerField(default=0)
    SeatsBooked=m.BigIntegerField(default=0)
    Seatscancelled=m.BigIntegerField(default=0)
    whenupdated=m.DateTimeField(auto_now=True)


class Family(m.Model): # FYI: It has all the attributes of User such as .is_authenticated etc... since it has inherited from User via onetoone
    user = m.OneToOneField(User, on_delete=m.CASCADE,related_name='me')# this is just to make a onetoone connection between a user and this.
    Parent1 = m.CharField(max_length=200)
    Parent2 = m.CharField(max_length=200, blank=True)
    Guardians = m.CharField(max_length=200, null=True, blank=True)


    def __str__(self):
        return f"{self.user.username}-{self.Parent1}-{self.Parent2}-{self.Guardians}".replace('None','').replace("--", "-").strip("-")


class events(m.Model):
    event=m.CharField(max_length=200)
    # attendees=m.ManyToManyField(User, through='linkage') #this linkage class that we call allows
```

```
    # #the many to many linkage while also allowing each link/pair to have its own unique attributes
    Date=m.DateTimeField()
    Desc=m.CharField(max_length=500,null=True)
    img=m.CharField(max_length=500,null=True)#image link to be rendered.
    red=m.CharField(max_length=100,default='default')#path to redirect on clicking (Usually the hall plan.(default)
    #If aldready booked, then to ticket page. But in an event like the carnival, directly to ticket page.)

blocked=m.CharField(max_length=10000,null=True,default='A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16,A17,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,B16,B17,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,')
    #blocking first three rows by default.pyt
    notifymail=m.CharField(max_length=1000000,default="",null=True,blank=True)#contains emails as email1,email2,... like seats booked.

siblingsbooked=m.CharField(max_length=1000000,default="",null=True,blank=True)#contains the usns of those that couldnt book because their siblings had booked.
    entered=m.BigIntegerField(default=0)
    scanned=m.BigIntegerField(default=0)
    cancels=m.BigIntegerField(default=0)
    updatedon=m.DateTimeField(auto_now=True)

    def __str__(self):
        return self.event

    def save(self, *args, **kwargs):
        if self.red=="default":
            self.red = f"book/{urllib.parse.quote(self.event)}/"
        super().save(*args, **kwargs)

    class Meta:
        unique_together=[['event']]
```

```
class GroupEventLink(m.Model): #With this, we are linking a group to an event so that we dont have to make a linkage for each user manually.
    #we only have to link the event to the group and then django should handle the induvidual attributes to be unique for each user.
    group = m.ForeignKey('auth.Group', on_delete=m.CASCADE)
    event = m.ForeignKey(events, on_delete=m.CASCADE)
    def __str__(self):
        return f"{self.event}-{self.group}"
    class Meta:
        unique_together=[['event','group']]


@receiver(post_save, sender=GroupEventLink)#calls the function whenever a GroupEventLink is created.
def create_linkages_for_group(sender, instance, created, **kwargs):
    if created:
        for user in instance.group.user_set.all():#makes a linkage for all users in that group to that event.
            if not linkage.objects.filter(event=instance.event,user=user).exists(): #if the user is aldready linked, do nothing.
                linkage.objects.create(event=instance.event, user=user,fami=Family.objects.get(user=user),grp=instance)#else, make the linkage


@receiver(pre_delete, sender=GroupEventLink)
def deleteold(sender, instance, **kwargs):
    event = instance.event
    group = instance.group
    others = GroupEventLink.objects.filter(event=event, group__user__in=group.user_set.all()).exclude(pk=instance.pk)
    # Checking if there are other groups of the user linked to the same event
    if not others.exists():
        # No other group links for the users and event, so delete related linkages
        linkages_to_delete = linkage.objects.filter(event=event, user__groups=group)
        linkages_to_delete.delete()
```

```
@receiver(post_save, sender=Family)
def Userchecklinkages(sender, instance, created, **kwargs):#to check if theres any new linkages to be made if a user is newely added (after the groupeventlink is added) or freshly added to that group.
    for grp in instance.user.groups.all():
        grpevents=grp.groupeventlink_set.all()
        for i in grpevents:
            if not linkage.objects.filter(event=i.event, user=instance.user).exists(): #if user aldready linked to event, leave
                linkage.objects.create(event=i.event, user=instance.user,fami=instance,grp=i)#else, link user to event using this.
class linkage(m.Model): #this is where we define the unique attributes to each link/pair.
    event = m.ForeignKey(events, on_delete=m.CASCADE)
    user=m.ForeignKey(User,on_delete=m.CASCADE)
    fami=m.ForeignKey(Family,on_delete=m.SET_NULL,null=True)
    grp=m.ForeignKey(GroupEventLink,on_delete=m.DO_NOTHING)
    seats=m.CharField(max_length=10,blank=True,null=True)#will be filled automatically when booked
    maxseats=m.IntegerField(default=2)
    whenbooked=m.DateTimeField(auto_now=True)
    created=m.DateTimeField(auto_now_add=True)
    #ticket=m.ImageField(blank=True,null=True)
    details=m.CharField(max_length=500,blank=True,null=True)#ticketdetails must be filled automatically when booked.
    emailsent=m.DateTimeField(blank=True,null=True)
    scanned=m.DateTimeField(blank=True,null=True)

    def __str__(self):
        return f"{self.user.username}-{self.event}"

    class Meta:#YET TO FINISH, ask group and finish.
        unique_together=[['event','user']]# a list of fields that should be associated together / unique together.
        #this means that we can put/ link a user in an event only once.
```

```
        #try adding this for seats etc also...


    #can acces with get, blah blah.
    #to add via code, can get that particular event and then say
event.user.add(<user>,through_defaults={'seats':'12,32'})
    #event.save()
    #if user.events.all() doesnt work to get the data, try user.events_set.all()
```

TBS_Box_Office\homepage\urls.py:

```
from django.urls import path,include
from . import views

urlpatterns=[
    path('',views.helloworld,name='TBS'),
    path('home/',views.loginPage, name='home'),
    path('events/',include('events.urls')),
    path('logout/',views.logoutPage,name='logout'),
    path('details/<str:pk>/',views.details,name='details'),
    path('reports/',views.report,name="Greport"),
    path('adminlinks/',views.adminlinks, name='adminlinks'),
]
```

TBS_Box_Office\homepage\views.py:

```
import datetime
from django.forms import DateTimeField
from django.shortcuts import render, redirect
from django.contrib import messages #this is what we import to get flash messages.
from django.contrib.auth.models import User
from .models import linkage,events,General
from django.contrib.auth import authenticate,login,logout #imported for the user login
from django.contrib.auth.decorators import login_required,user_passes_test #this is used to restrict pages that need login or a certain login.


def loginPage(request):
    if request.user.is_authenticated:
        return(redirect('events'))
    if request.method=="POST":
        username=request.POST.get('Fusn')
        password=request.POST.get('Fpwd')
        #email=request.POST.get('email') #given while signing in so not taking it again.
        try:
            user=User.objects.get(username=username)
        except:
            messages.error(request,'USN does not exist.')
        user = authenticate(request,username=username,password=password)
        if user is not None:
            login(request,user)
            return(redirect('events'))
        else:
            messages.error(request,"Username and password don't match.")

    context={}
    return(render(request,'hi.html',context))
```

```
@login_required(login_url='home') #just in case.
def logoutPage(request):
    GI, created = General.objects.get_or_create(pk=1)
    GI.logouts+=1
    GI.save()
    logout(request)
    return(redirect('home'))


def helloworld(request):
    return(render(request,'firstpage.html'))


@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def details(request,pk):
    link=linkage.objects.get(user=User.objects.get(username=pk.split('-',1)[0]),event=events.objects.get(event=pk.split('-',1)[-1]))
    #this only works if the usndoesnt have a hyphen in it and the rest of the code is exactly as is.

context={"USN":link.user.username,"seats":link.seats,"group":f"{link.grp.group}","family":f"{link.fami.Parent1}-{link.fami.Parent2}-{link.fami.Guardians}".replace("None","").replace("--", "-").strip("-"),"whenbooked":str(link.whenbooked),"whenmade":str(link.created),"details":link.details,'emailsent':str(link.emailsent),'event':link.event}
    if link.seats==None: #if no seats booked, change whenbooked to none. (default would be when created.)
        context['whenbooked']=None
    link.scanned=datetime.datetime.now()
    link.save()
    GI, created = General.objects.get_or_create(pk=1)
    GI.QRscanned+=1
    GI.save()
    a=link.event
    a.scanned=a.scanned+1
    a.save()
```

```
    return(render(request,'details.html',context))


@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def report(request):
    GI, created = General.objects.get_or_create(pk=1)

context={'logins':GI.logins,'logouts':GI.logouts,'emailsent':GI.emailsent,'QRscanned':GI.QRscanned,'Seatsbooked':GI.SeatsBooked,'Seatscancelled':GI.Seatscancelled,'whenupdated':GI.whenupdated}
    context['nosiblings']=sum([len([i for i in j.siblingsbooked.strip('\' ,'').split(',') if i not in '\' ,']) for j in events.objects.all() if j.siblingsbooked])
    context['nowaiting']=sum([len([i for i in j.notifymail.split(',') if i not in ' ,']) for j in events.objects.all() if j.notifymail])
    return(render(request,'Grep.html',context))
@user_passes_test(lambda u: u.is_superuser,login_url='home') #if user is admin, let them see this page.
def adminlinks(request):
    return(render(request,'linktree.html'))
```

TBS_Box_Office\new\settings.py:

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'django-insecure-n7pix!2*n+4o$374co6cw7jh9-
3u_(b43^m^n$wdt41o4j4nkt'

DEBUG = True

ALLOWED_HOSTS = []

#import ezgmail

#ezgmail.init(credentialsFile='confirm.json', tokenFile='token.json')

''' #tried using djangos inbuilt gmail sending method for a while but didnt work out. switched
to ezgmail instead.

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_HOST = 'smtp.gmail.com'

EMAIL_PORT = 587  # Port for TLS

EMAIL_USE_SSL = False  # Use SSL (True for SMTP over SSL)

EMAIL_USE_TLS = True

EMAIL_HOST_USER = 'suadnastorage@gmail.com'

EMAIL_HOST_PASSWORD = 'Suadna@temp69'

'''

#APPEND_SLASH=False #temp

#This is what I have added to ensure session logout after a certain time.

# Close the session when the user closes the browser

SESSION_EXPIRE_AT_BROWSER_CLOSE = True

# Set the session cookie age (in seconds) (change as you want)

SESSION_COOKIE_AGE = 1200 #20 mins.

# Save the session on every request (to reset the session cookie age timer.)

SESSION_SAVE_EVERY_REQUEST = True

# Application definition

INSTALLED_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',
```

```
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'homepage',
    'events',
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'new.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            BASE_DIR / 'templates', #telling django where we are storing all our templates. (a
public folder so that all can access.)
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
```

```
]


WSGI_APPLICATION = 'new.wsgi.application'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

AUTH_PASSWORD_VALIDATORS = [] #REMOVE THIS. IT IS ONLY FOR PRODUCTION. IT ALLOWS THE PASSWORD TO BE WEAK. the above password validators ensure security. by overwriting the list and making it empty, we remove all these safety checks. this is only temporary so that I can test with simple passwords. Do not forget to remove during production.

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
STATIC_URL = 'static/'
```

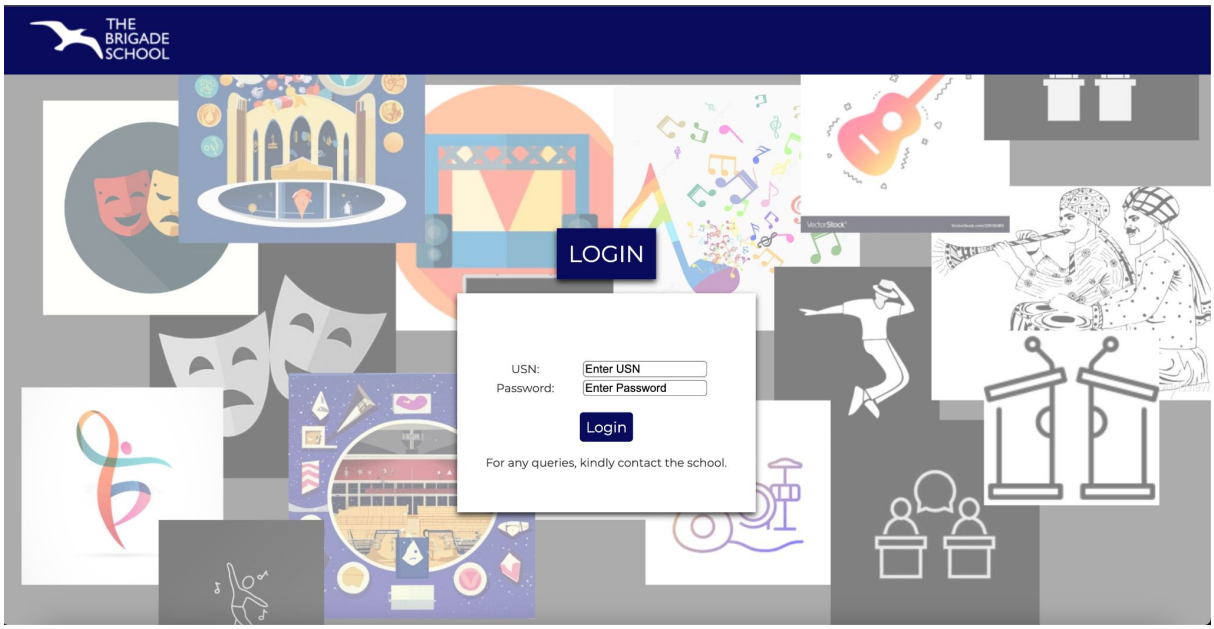
```
STATICFILES_DIRS=[#telling django where we are keeping all our static files so that it can handle the rest.
    BASE_DIR/'static'
]
#STATIC_ROOT=
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

TBS_Box_Office\new\urls.py:

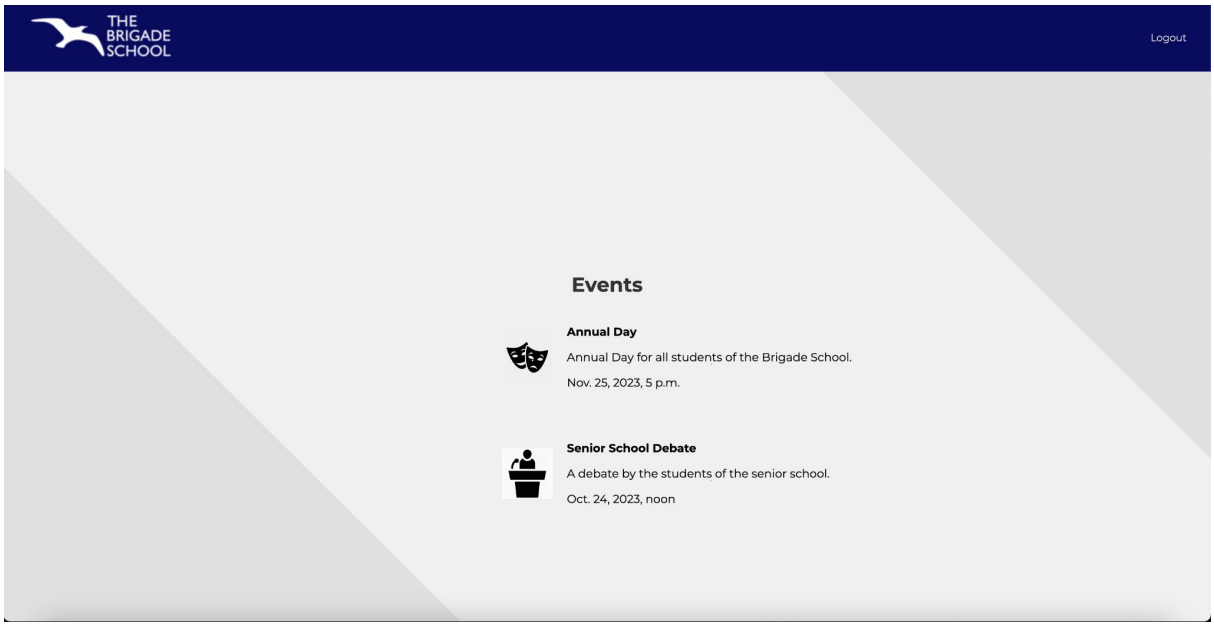
```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('homepage.urls')),
]
```

OUTPUT SCREENSHOTS

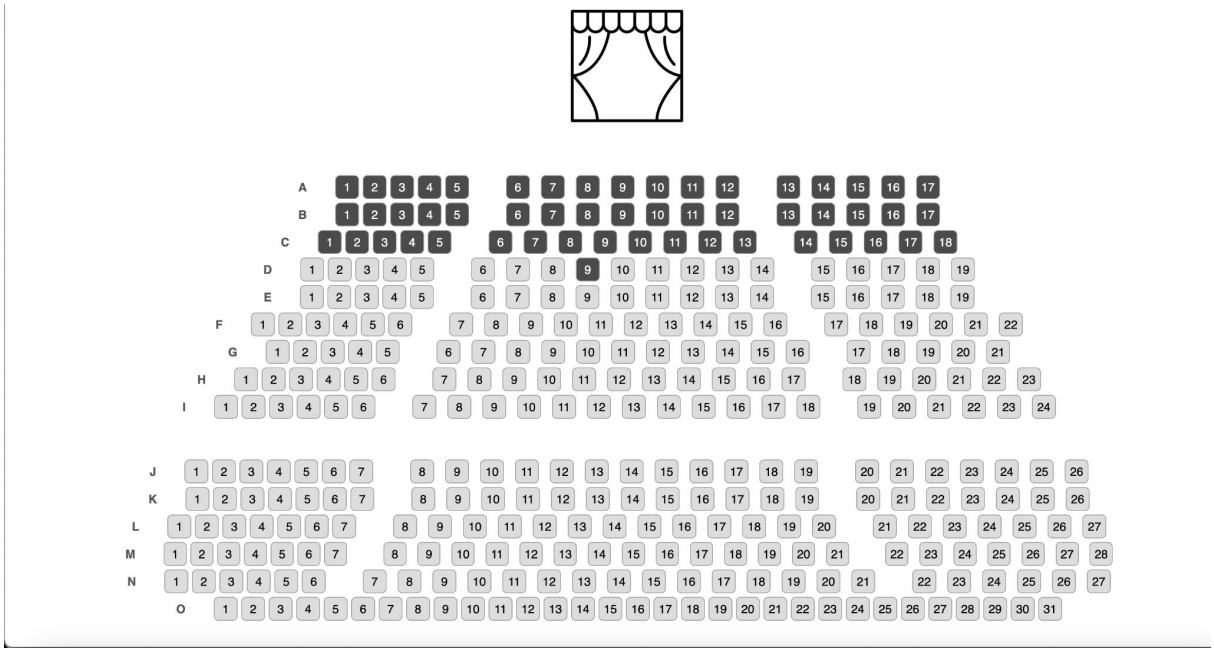
Login Page



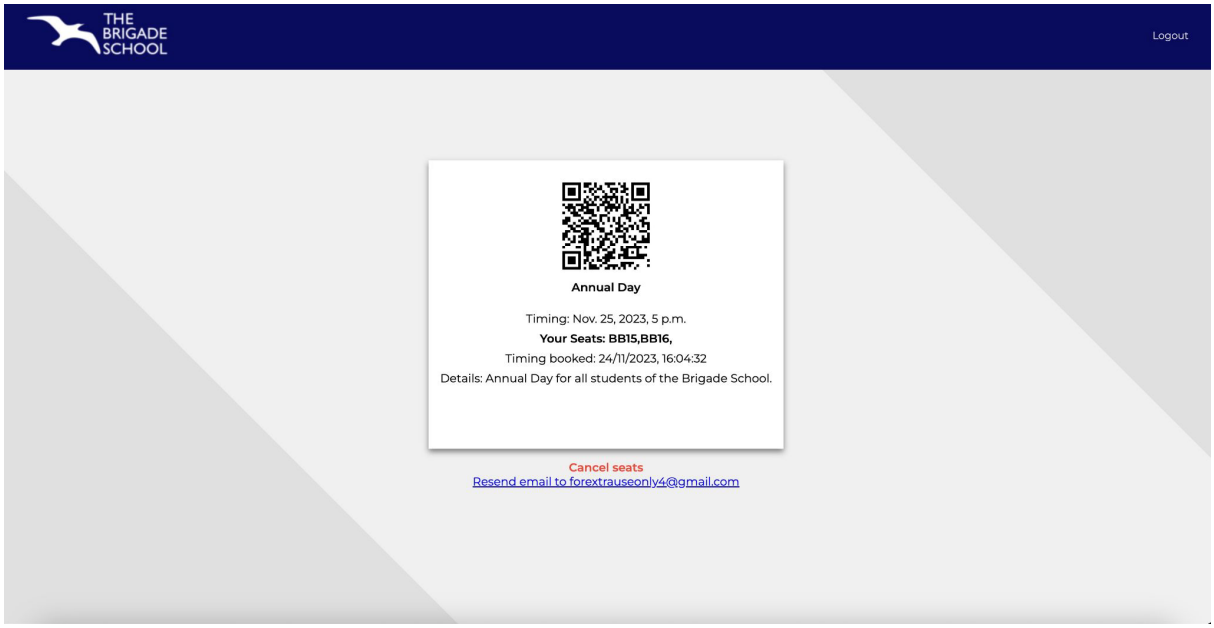
Events Page



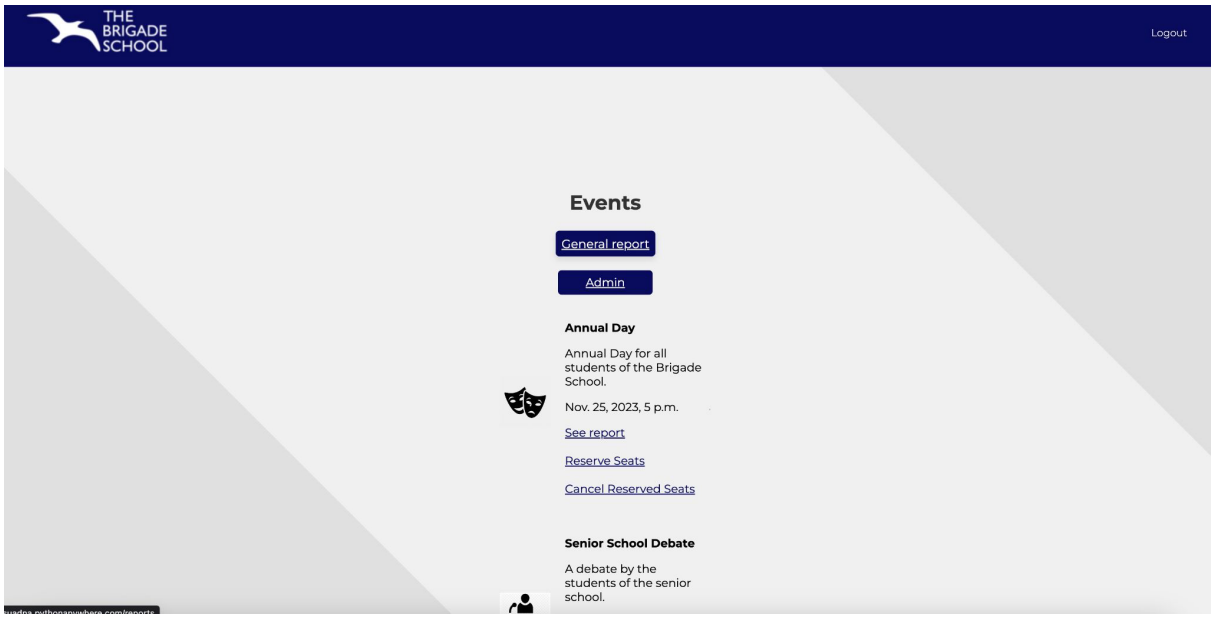
Seat Selection



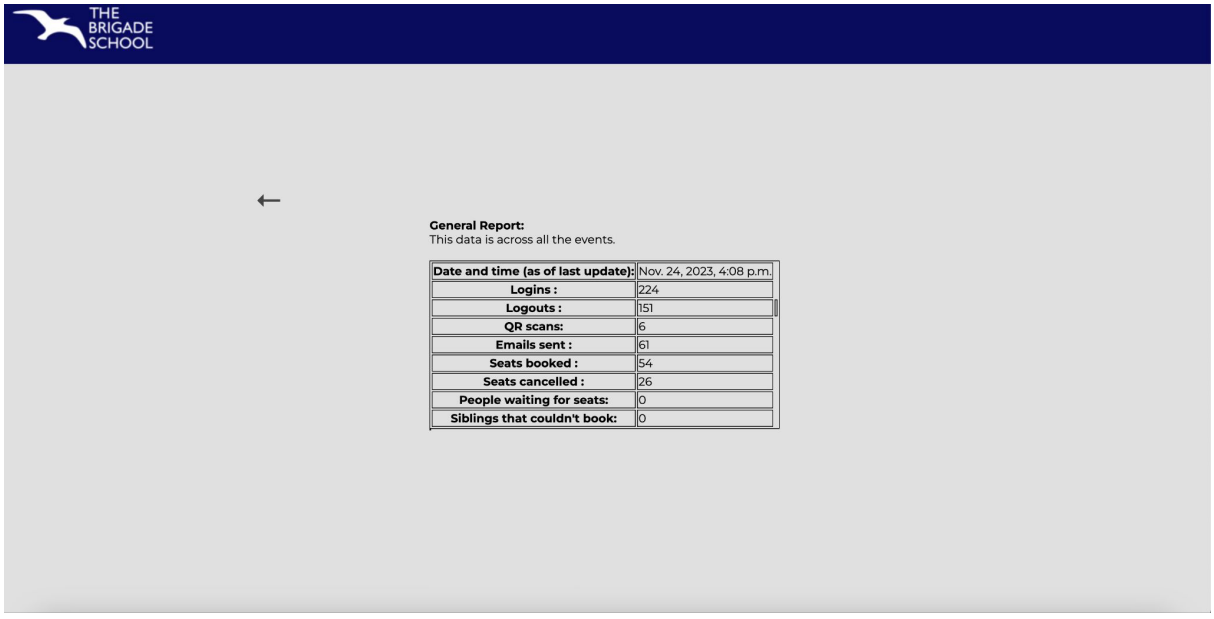
Ticket Page



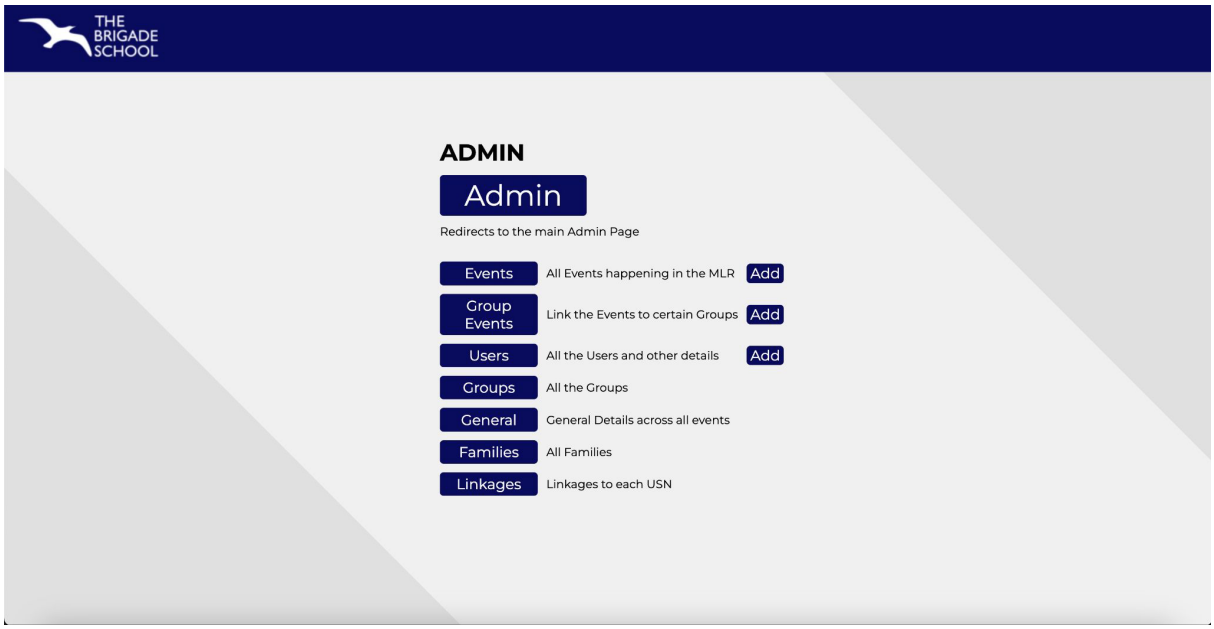
Admin Page



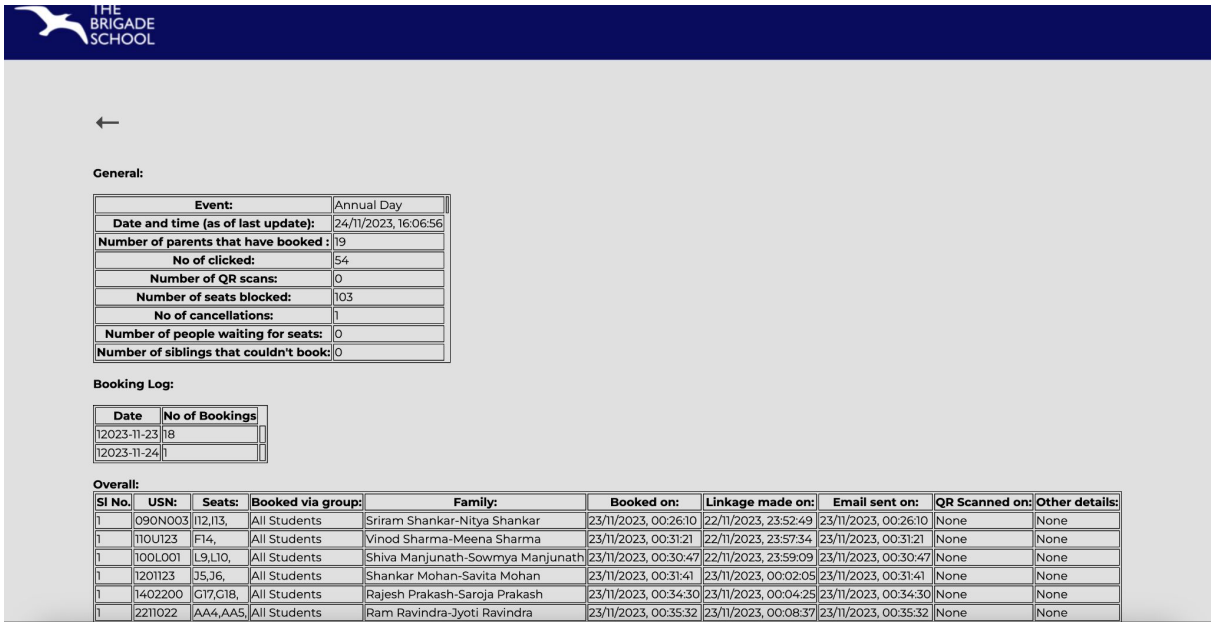
General Report



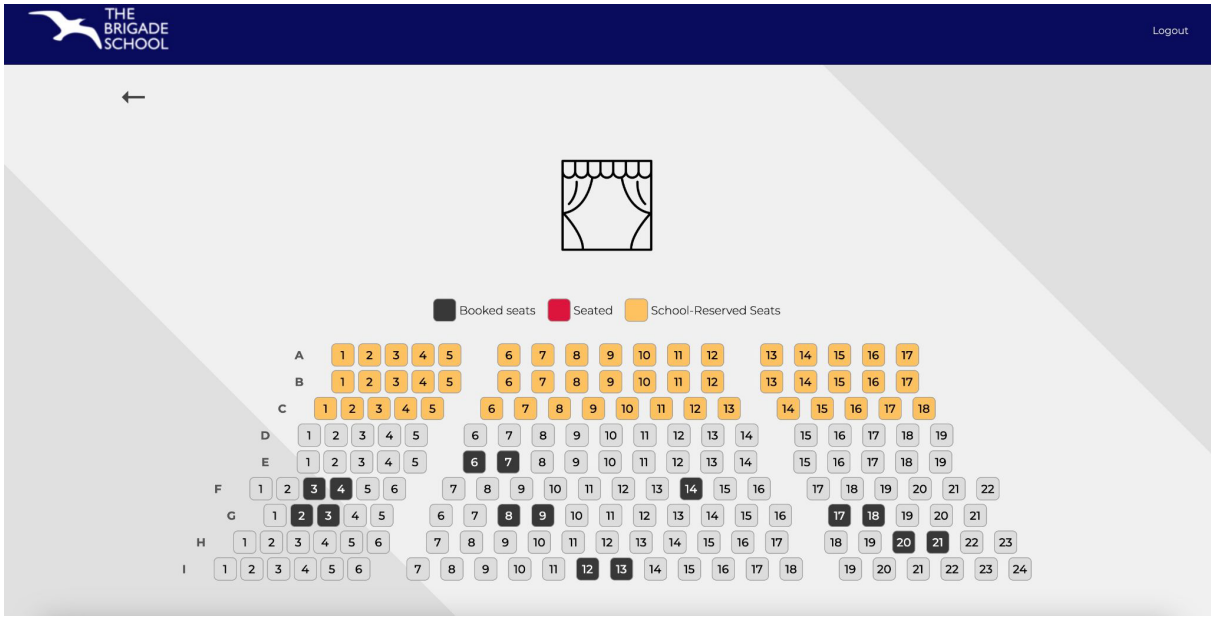
Admin Features



Event Report



Auditorium Status



FUTURE ENHANCEMENTS

- Enhancing report generation which includes offloading all details of events that are over and related linkages onto a separate database.
- Allowing for repeated events instead of deleting and recreating events.
- Automating the credentials.JSON refresh.
- Setting up an API for the website to allow for faster, more efficient transfer of data between frontend and backend.
- Being able to keep updated with school databases in real time.
- Upgrading the style sheets to be able to format various themes.
- Making all the pages more mobile friendly.

WEBILOGRAPHY

geeksforgeeks.org

codepen.io

automatetheboringstuff.com

chat.openai.com

Django documentation

Youtube.com