so lets re write the prompt to actually give you a better insight of the pos   
this system should be used to geerate tickets for people paying by cash and carry out event verification..  
it should have agents but they should be under vendors ir there shud be a one to many relationship between the vendors and the agents... the vendor has the capability to create as many agents as he wants and fir each agent he creates he should assign them atleast one event that is upcoming under his events and with status == approved, so the egents should habe the ability to generate tickets and also verfy the tickest for those events that were assigned to them..  
in their Django app they should have a model that stores the username first name, last name , email, vendor, (and since the vendor can attach more than one event to these agaents that means they should have a connection to the events somehow but for those that have been assigned to them ).. also another table that stores the log of the different agents which ever action they do is recorded in this table since the vendor should be able to track wat they are doing ...

so these are the pages we need of the pos agents... and they should all extend the frontend/base.html pos dashboard(what they will see when they are logged in .. it should have the events that were assigned to them listed down and clickable , should have a log out button and a change password button since they can be able to change their password and also the history button, ), the change password(it should have a portion where they put in their old password provided by the system and also a new one and a confirm password for the new one with the final button of change password), history(this page will have the events that were assigned to the agent and wen pressed it should have a page that is divided into categories ie tickests generated, and tickets verified by you and under each we will have the total number for both.. therefore the system should be able to track what was created by who especially tickets and verification.. this is to help the vendor track the activity of the agents), POS page(this page will be called upon by the vendor.. this is where he is able to see, create the agents from.. so in here he will have a create button for the pos agent.., he will have a section of active agents where those assigned events whose start date or end dat hasn’t reached the current date will be displayed and another or inactive agents where agents whose events assigned have startdate or endo date past the current date are all these agents are clickable and wen pressed u get to the agent page ), agent page(this one is visible to the vendor here he can see wat that specific POS agent has done, how many ticktes he has created, how many he has verified under each event assigned and he can actually get a report for that listing down the ticket number for each portion separated for created, and also verified and the total amount of money they have collected for that event) so the vendor has the ability to activate and deactivate the agent when the agent who has been active is deactivated, the history stays input its just the agent wont be able to log in and and carry out actions..  
so these are the actions the vendor can do on the agent.. he can create, edit the agents events assigned, activate or deactivate the agent, delete the agent (but the tickets created by them stay in the system under the vendor), view the agents activities.. and all this is done from the agents page except the create.. and also he can see the overall amount the agent has made from all the tickets generations they have done for the events they have been assigned to   
 the agent can generate tickets for only the assigned events, verify tickets, login, logout, change password, view their history of wat they have done ..  
how many tickets have been generated by them, or verified by them, the total amount of money they have made under each event because whenever a ticket is generated, the amount under each category is collected and calcultated do on the tatal receipt i will need that amount saved up and shown..  
 so for all these functionalities i have stated.. u have to help me make sure we set up models for the storage of each and every aspect so that we can do a better data retrieval and manipulation .. for some we dont need to create new models but some we do  
now for instance the tickets have a model already in existence   
*# tickets/models.py*

**from** django.db **import** models

**from** events.models **import** Event, TicketCategory

**from** customers.models **import** Customer

**from** vendors.models **import** Vendor

**import** qrcode

**from** io **import** BytesIO

**from** django.core.files **import** File

**class** Ticket(models.Model):

    ENTITY\_TYPE\_CHOICES **=** [

        ('customer', 'Customer'),

        ('vendor', 'Vendor'),

    ]

    event **=** models.ForeignKey(Event, on\_delete**=**models.CASCADE, related\_name**=**'tickets')

    ticket\_category **=** models.ForeignKey(TicketCategory, on\_delete**=**models.CASCADE, null**=True**, blank**=True**)

    customer\_username **=** models.CharField(max\_length**=**255, blank**=True**, null**=True**)

    vendor **=** models.ForeignKey(Vendor, on\_delete**=**models.CASCADE, related\_name**=**'tickets')

    ticket\_number **=** models.CharField(max\_length**=**20, unique**=True**)

    purchase\_date **=** models.DateTimeField(auto\_now\_add**=True**)

    qr\_code **=** models.ImageField(upload\_to**=**'qr\_codes/', blank**=True**)

    entity\_type **=** models.CharField(max\_length**=**10, choices**=**ENTITY\_TYPE\_CHOICES)

    verified **=** models.BooleanField(default**=False**)

**def** generate\_qr\_code(self):

        qr **=** qrcode.QRCode(

            version**=**1,

            error\_correction**=**qrcode.constants.ERROR\_CORRECT\_L,

            box\_size**=**10,

            border**=**4,

        )

        qr.add\_data(self.ticket\_number)

        qr.make(fit**=True**)

        img **=** qr.make\_image(fill**=**'black', back\_color**=**'white')

        buffer **=** BytesIO()

        img.save(buffer, format**=**'PNG')

        file\_name **=** **f**'{self.ticket\_number}.png'

        self.qr\_code.save(file\_name, File(buffer), save**=False**)

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

**if** **not** self.qr\_code:

            self.generate\_qr\_code()

        super().save(**\***args, **\*\***kwargs)

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

**return** **f**'{self.ticket\_number} - {self.event.title}'

    @staticmethod

**def** generate\_ticket\_number(event, vendor, category, entity, entity\_type):

**from** random **import** randint

        vendor\_code **=** vendor.storename[0].upper()

        vendor\_id **=** str(vendor.id).zfill(4)

        event\_code **=** event.title[0].upper()

        event\_id **=** str(event.id).zfill(4)

        category\_code **=** category.category\_title[:2].upper() **if** category **else** "OR"  *# OR for ordinary*

        category\_id **=** str(category.id).zfill(2) **if** category **else** "00"

*# Determine entity code and ID based on whether it's a customer or a vendor*

**if** entity\_type **==** 'customer':

            entity\_code **=** entity.username[0].upper()

            entity\_id **=** str(entity.id).zfill(3)

**elif** entity\_type **==** 'vendor':

            entity\_code **=** entity.storename[0].upper()

            entity\_id **=** str(entity.id).zfill(4)  *# Assuming vendor IDs may need to be 4 digits*

**else**:

**raise** ValueError("Invalid entity type. Must be 'customer' or 'vendor'.")

        random\_digits **=** str(randint(1000, 9999))

**return** **f**'{vendor\_code}{vendor\_id}{event\_code}{event\_id}{category\_code}{category\_id}{entity\_code}{entity\_id}{random\_digits}'

but since it doesn’t have the who verified part we can make it that when a ticket is generated, it is saved in both the pos agent tickets generated with all the information plus the name of the pos agent who generated and at the same time be saved in the tickets model so that during verification, even the vendor himself can verify the tickets made by the pos agents for his event that he assigned them and for verification the pos agent can verify any ticket for that event assigned to it (an event can be assigned more than one agent although it is optional)  
 now for the pos authentication, although they have an email they shuld use a username and a password to login, and amongst their popertives we should have the is\_posagent Boolean field to help in identification.. so the login view for hte pos agents should be using the updated backends.py

vendors/backends.py

**from** django.contrib.auth.backends **import** ModelBackend

**from** vendors.models **import** Vendor

**from** customers.models **import** Customer

**class** VendorOrCustomerModelBackend(ModelBackend):

**def** authenticate(self, request, username**=None**, password**=None**, **\*\***kwargs):

        is\_vendor\_login **=** kwargs.get('is\_vendor\_login', **False**)

**try**:

**if** is\_vendor\_login:

*# Authenticate as a vendor*

                vendor **=** Vendor.objects.get(username**=**username)

**if** vendor.check\_password(password) **and** vendor.is\_vendor:

**return** vendor

**else**:

*# Authenticate as a customer*

                customer **=** Customer.objects.get(username**=**username)

**if** customer.check\_password(password) **and** customer.is\_customer:

**return** customer

**except** (Vendor.DoesNotExist, Customer.DoesNotExist):

**return** **None**

**def** get\_user(self, user\_id):

**try**:

**return** Vendor.objects.get(pk**=**user\_id)

**except** Vendor.DoesNotExist:

**try**:

**return** Customer.objects.get(pk**=**user\_id)

**except** Customer.DoesNotExist:

**return** **None  
  
and vendors/views.py**

to add the posagent bit..(dont change the name of the class just add the functionality to cater for the pos agent authentication login)

so these are the the models for events, and vendors and tickets   
vendors   
**from** django.db **import** models

**from** django.contrib.auth.models **import** AbstractBaseUser, BaseUserManager, PermissionsMixin

**from** django.utils **import** timezone

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

**class** VendorManager(BaseUserManager):

**def** create\_user(self, username, email, first\_name, last\_name, storename, store\_phone, password**=None**):

**if** **not** email:

**raise** ValueError('Users must have an email address')

        user **=** self.model(

            username**=**username,

            email**=**self.normalize\_email(email),

            first\_name**=**first\_name,

            last\_name**=**last\_name,

            storename**=**storename,

            store\_phone**=**store\_phone,

        )

        user.set\_password(password)

        user.save(using**=**self.\_db)

**return** user

**def** create\_superuser(self, username, email, first\_name, last\_name, storename, store\_phone, password**=None**):

        user **=** self.create\_user(

            username,

            email,

            first\_name,

            last\_name,

            storename,

            store\_phone,

            password**=**password

        )

        user.is\_admin **=** **True**

        user.is\_superuser **=** **True**

        user.is\_staff **=** **True**

        user.save(using**=**self.\_db)

**return** user

**class** Vendor(AbstractBaseUser, PermissionsMixin):

    username **=** models.CharField(max\_length**=**30, unique**=True**)

    email **=** models.EmailField(unique**=True**)

    first\_name **=** models.CharField(max\_length**=**30)

    last\_name **=** models.CharField(max\_length**=**30)

    storename **=** models.CharField(max\_length**=**50)

    store\_phone **=** models.CharField(max\_length**=**15)

    verification\_code **=** models.CharField(max\_length**=**6, null**=True**, blank**=True**)

    verification\_code\_created\_at **=** models.DateTimeField(null**=True**, blank**=True**)

    is\_active **=** models.BooleanField(default**=True**)

    is\_admin **=** models.BooleanField(default**=False**)

    is\_superuser **=** models.BooleanField(default**=False**)

    is\_staff **=** models.BooleanField(default**=False**)

    is\_vendor **=** models.BooleanField(default**=True**)

    objects **=** VendorManager()

    USERNAME\_FIELD **=** 'username'

    REQUIRED\_FIELDS **=** ['email', 'first\_name', 'last\_name', 'storename', 'store\_phone']

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

**return** self.username

**def** has\_perm(self, perm, obj**=None**):

**return** **True**

**def** has\_module\_perms(self, app\_label):

**return** **True**

tickets  
*# tickets/models.py*

**from** django.db **import** models

**from** events.models **import** Event, TicketCategory

**from** customers.models **import** Customer

**from** vendors.models **import** Vendor

**import** qrcode

**from** io **import** BytesIO

**from** django.core.files **import** File

**class** Ticket(models.Model):

    ENTITY\_TYPE\_CHOICES **=** [

        ('customer', 'Customer'),

        ('vendor', 'Vendor'),

    ]

    event **=** models.ForeignKey(Event, on\_delete**=**models.CASCADE, related\_name**=**'tickets')

    ticket\_category **=** models.ForeignKey(TicketCategory, on\_delete**=**models.CASCADE, null**=True**, blank**=True**)

    customer\_username **=** models.CharField(max\_length**=**255, blank**=True**, null**=True**)

    vendor **=** models.ForeignKey(Vendor, on\_delete**=**models.CASCADE, related\_name**=**'tickets')

    ticket\_number **=** models.CharField(max\_length**=**20, unique**=True**)

    purchase\_date **=** models.DateTimeField(auto\_now\_add**=True**)

    qr\_code **=** models.ImageField(upload\_to**=**'qr\_codes/', blank**=True**)

    entity\_type **=** models.CharField(max\_length**=**10, choices**=**ENTITY\_TYPE\_CHOICES)

    verified **=** models.BooleanField(default**=False**)

**def** generate\_qr\_code(self):

        qr **=** qrcode.QRCode(

            version**=**1,

            error\_correction**=**qrcode.constants.ERROR\_CORRECT\_L,

            box\_size**=**10,

            border**=**4,

        )

        qr.add\_data(self.ticket\_number)

        qr.make(fit**=True**)

        img **=** qr.make\_image(fill**=**'black', back\_color**=**'white')

        buffer **=** BytesIO()

        img.save(buffer, format**=**'PNG')

        file\_name **=** **f**'{self.ticket\_number}.png'

        self.qr\_code.save(file\_name, File(buffer), save**=False**)

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

**if** **not** self.qr\_code:

            self.generate\_qr\_code()

        super().save(**\***args, **\*\***kwargs)

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

**return** **f**'{self.ticket\_number} - {self.event.title}'

    @staticmethod

**def** generate\_ticket\_number(event, vendor, category, entity, entity\_type):

**from** random **import** randint

        vendor\_code **=** vendor.storename[0].upper()

        vendor\_id **=** str(vendor.id).zfill(4)

        event\_code **=** event.title[0].upper()

        event\_id **=** str(event.id).zfill(4)

        category\_code **=** category.category\_title[:2].upper() **if** category **else** "OR"  *# OR for ordinary*

        category\_id **=** str(category.id).zfill(2) **if** category **else** "00"

*# Determine entity code and ID based on whether it's a customer or a vendor*

**if** entity\_type **==** 'customer':

            entity\_code **=** entity.username[0].upper()

            entity\_id **=** str(entity.id).zfill(3)

**elif** entity\_type **==** 'vendor':

            entity\_code **=** entity.storename[0].upper()

            entity\_id **=** str(entity.id).zfill(4)  *# Assuming vendor IDs may need to be 4 digits*

**else**:

**raise** ValueError("Invalid entity type. Must be 'customer' or 'vendor'.")

        random\_digits **=** str(randint(1000, 9999))

**return** **f**'{vendor\_code}{vendor\_id}{event\_code}{event\_id}{category\_code}{category\_id}{entity\_code}{entity\_id}{random\_digits}'

and the events  
**from** django.db **import** models

**from** django.conf **import** settings

**from** django.contrib.contenttypes.models **import** ContentType

**from** django.contrib.contenttypes.fields **import** GenericForeignKey

**from** vendors.models **import** Vendor  *# This remains as is*

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

**from** django.utils **import** timezone

**class** Event(models.Model):

    STATUS\_CHOICES **=** [

        ('pending', 'Pending'),

        ('approved', 'Approved'),

        ('rejected', 'Rejected'),

    ]

    vendor **=** models.ForeignKey(Vendor, on\_delete**=**models.CASCADE, related\_name**=**'events')

    poster **=** models.ImageField(upload\_to**=**'event\_posters/')

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

    description **=** models.TextField()

    category **=** models.CharField(max\_length**=**100)

    start\_date **=** models.DateTimeField()

    end\_date **=** models.DateTimeField(null**=True**, blank**=True**)

    venue\_name **=** models.CharField(max\_length**=**100)

    regular\_price **=** models.DecimalField(max\_digits**=**10, decimal\_places**=**2)

    sale\_price **=** models.DecimalField(max\_digits**=**10, decimal\_places**=**2, null**=True**, blank**=True**)

    tickets\_available **=** models.PositiveIntegerField(null**=True**, blank**=True**)

    tickets\_sold **=** models.PositiveIntegerField(default**=**0)

    status **=** models.CharField(max\_length**=**10, choices**=**STATUS\_CHOICES, default**=**'pending')

    adminaction **=** models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete**=**models.SET\_NULL, null**=True**, blank**=True**, editable**=False**)  *# Stores the admin who approved/rejected*

**def** is\_sold\_out(self):

**return** self.tickets\_available **is** **not** **None** **and** self.tickets\_sold **>=** self.tickets\_available

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

**return** self.title

**class** TicketCategory(models.Model):

    event **=** models.ForeignKey(Event, on\_delete**=**models.CASCADE, related\_name**=**'ticket\_categories')  *# Mandatory, linked to Event*

    category\_title **=** models.CharField(max\_length**=**100)  *# Required field*

    category\_price **=** models.DecimalField(max\_digits**=**10, decimal\_places**=**2)  *# Required field*

    category\_tickets\_available **=** models.PositiveIntegerField()  *# Required field*

    category\_tickets\_sold **=** models.PositiveIntegerField(default**=**0)  *# New field to track tickets sold in this category*

**def** is\_category\_sold\_out(self):

**return** self.category\_tickets\_sold **>=** self.category\_tickets\_available

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

**return** **f**'{self.category\_title} - {self.event.title}'

**class** Cart(models.Model):

    vendor **=** models.ForeignKey('vendors.Vendor', on\_delete**=**models.CASCADE, null**=True**, blank**=True**)

    customer **=** models.ForeignKey('customers.Customer', on\_delete**=**models.CASCADE, null**=True**, blank**=True**)

    event **=** models.ForeignKey(Event, on\_delete**=**models.CASCADE)

    added\_at **=** models.DateTimeField(auto\_now\_add**=True**)

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

**return** **f**'{self.get\_user()} - {self.event.title}'

**def** get\_user(self):

**return** self.vendor **if** self.vendor **else** self.customer

**class** ActionLog(models.Model):

    admin\_user **=** models.ForeignKey(get\_user\_model(), on\_delete**=**models.CASCADE)  *# Link to the user who performed the action*

    event **=** models.ForeignKey(Event, on\_delete**=**models.CASCADE)  *# Direct reference to Event model*

    action **=** models.CharField(max\_length**=**50)  *# E.g., 'approved' or 'rejected'*

    timestamp **=** models.DateTimeField(default**=**timezone.now)  *# Time of the action*

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

**return** **f**"{self.admin\_user} {self.action} {self.event.title} at {self.timestamp}"

and the creation of tickets usually works for only two entities vendors and customers, i want to add another too but let me show yyou the view that carries this out  
  
  
**def** prepare\_event\_data(event\_id):

    event **=** get\_object\_or\_404(Event, id**=**event\_id)

    categories **=** event.ticket\_categories.all()

**if** event.is\_sold\_out():

**return** **None**, {'template': 'tickets/sold\_out.html', 'context': {'event': event}}

    ticket\_data **=** [

        {'category': category,

         'tickets\_remaining': category.category\_tickets\_available **-** category.category\_tickets\_sold}

**for** category **in** categories

    ]

    ordinary\_ticket\_data **=** **None**

**if** **not** categories.exists():

        ordinary\_remaining **=** event.tickets\_available **-** event.tickets\_sold

        ordinary\_ticket\_data **=** {'price': event.sale\_price, 'tickets\_remaining': ordinary\_remaining}

**return** {'event': event, 'ticket\_data': ticket\_data, 'ordinary\_ticket\_data': ordinary\_ticket\_data}, **None**

**def** get\_user\_entity(request):

**if** hasattr(request.user, 'is\_vendor') **and** request.user.is\_vendor:

**return** 'vendor', request.user

**elif** hasattr(request.user, 'is\_customer') **and** request.user.is\_customer:

**return** 'customer', request.user

**else**:

**return** **None**, **None**

**def** process\_ticket\_purchase(event, ticket\_data, ordinary\_ticket\_data, tickets\_info, entity, entity\_type):

    ticket\_details **=** []

    total\_tickets **=** 0

    total\_price **=** 0

**for** data **in** ticket\_data:

        category **=** data['category']

        quantity **=** int(tickets\_info.get(**f**'quantity\_{category.id}', 0))

**if** quantity **>** 0:

**if** category.is\_category\_sold\_out() **or** category.category\_tickets\_sold **+** quantity **>** category.category\_tickets\_available:

**return** **None**, {'template': 'events/sold\_out.html', 'context': {'event': event}}

            ticket\_details **+=** generate\_tickets(event, category, quantity, entity, entity\_type)

            total\_tickets **+=** quantity

            total\_price **+=** quantity **\*** category.category\_price

            category.category\_tickets\_sold **+=** quantity

            category.save()

**if** **not** ticket\_data **and** ordinary\_ticket\_data:

        quantity **=** int(tickets\_info.get('quantity\_ordinary', 0))

**if** quantity **>** 0:

**if** event.is\_sold\_out() **or** event.tickets\_sold **+** quantity **>** event.tickets\_available:

**return** **None**, {'template': 'events/sold\_out.html', 'context': {'event': event}}

            ticket\_details **+=** generate\_tickets(event, **None**, quantity, entity, entity\_type)

            total\_tickets **+=** quantity

            total\_price **+=** quantity **\*** event.sale\_price

    event.tickets\_sold **+=** total\_tickets

    event.save()

**return** {'ticket\_details': ticket\_details, 'total\_tickets': total\_tickets, 'total\_price': total\_price}, **None**

**def** generate\_tickets(event, category, quantity, entity, entity\_type):

    tickets **=** []

**for** \_ **in** range(quantity):

        ticket\_number **=** Ticket.generate\_ticket\_number(event**=**event, vendor**=**event.vendor, category**=**category, entity**=**entity, entity\_type**=**entity\_type)

        qr\_image **=** generate\_qr\_code(ticket\_number)

        ticket **=** Ticket.objects.create(

            event**=**event,

            ticket\_category**=**category,

            customer\_username**=**entity.username,

            vendor**=**event.vendor,

            ticket\_number**=**ticket\_number,

            qr\_code**=**qr\_image,

            entity\_type**=**entity\_type

        )

        tickets.append({

            'ticket\_number': ticket\_number,

            'category': category.category\_title **if** category **else** 'Ordinary',

            'price': category.category\_price **if** category **else** event.sale\_price,

            'qr\_code\_url': ticket.qr\_code.url

        })

**return** tickets

**def** generate\_qr\_code(ticket\_number):

    qr **=** qrcode.QRCode(version**=**1, error\_correction**=**qrcode.constants.ERROR\_CORRECT\_L, box\_size**=**10, border**=**4)

    qr.add\_data(ticket\_number)

    qr.make(fit**=True**)

    img **=** qr.make\_image(fill\_color**=**"black", back\_color**=**"white")

    buffer **=** BytesIO()

    img.save(buffer)

**return** File(buffer, name**=f**'{ticket\_number}\_qr.png')

@login\_required

**def** buy\_ticket(request, event\_id):

    event\_data, error **=** prepare\_event\_data(event\_id)

**if** error:

**return** render(request, error['template'], error['context'])

**if** request.method **==** 'POST':

        msisdn **=** request.POST.get('msisdn')

        entity\_type, entity **=** get\_user\_entity(request)

**if** **not** entity:

**return** render(request, 'tickets/error.html', {'message': 'User is not authorized to buy tickets.'})

        purchase\_data, error **=** process\_ticket\_purchase(

            event\_data['event'], event\_data['ticket\_data'], event\_data['ordinary\_ticket\_data'], request.POST, entity, entity\_type

        )

**if** error:

**return** render(request, error['template'], error['context'])

        context **=** {

            'event': event\_data['event'],

            'vendor': event\_data['event'].vendor,

            'ticket\_details': purchase\_data['ticket\_details'],

            'total\_price': purchase\_data['total\_price'],

            'total\_tickets': purchase\_data['total\_tickets'],

            'customer': entity,

            'ticket\_data': event\_data['ticket\_data'],

            'ordinary\_ticket\_data': event\_data['ordinary\_ticket\_data'],

            'msisdn': msisdn

        }

**return** render(request, 'tickets/ticket\_success.html', context)

**return** render(request, 'tickets/buy\_ticket.html', event\_data)

@login\_required

**def** download\_ticket\_pdf(request, ticket\_number):

    ticket **=** get\_object\_or\_404(Ticket, ticket\_number**=**ticket\_number)

    event **=** ticket.event  *# Get the associated event*

    vendor **=** event.vendor  *# Get the associated vendor*

*# Create absolute URLs for QR code and poster*

    qr\_code\_url **=** request.build\_absolute\_uri(ticket.qr\_code.url)

    poster\_url **=** request.build\_absolute\_uri(event.poster.url)  *# Ensure you use 'event.poster.url'*

    context **=** {

        'ticket': ticket,

        'event': event,

        'vendor': vendor,

        'qr\_code\_url': qr\_code\_url,  *# Add absolute URL for QR code*

        'poster\_url': poster\_url,  *# Add absolute URL for poster*

    }

    template\_path **=** 'tickets/ticket\_pdf.html'

    response **=** HttpResponse(content\_type**=**'application/pdf')

    response['Content-Disposition'] **=** **f**'attachment; filename="{ticket.ticket\_number}\_ticket.pdf"'

    template **=** get\_template(template\_path)

    html **=** template.render(context)

    pisa\_status **=** pisa.CreatePDF(html, dest**=**response)

**if** pisa\_status.err:

**return** HttpResponse('We had some errors <pre>' **+** html **+** '</pre>')

**return** response

**def** generate\_pdf\_for\_ticket(template\_src, context\_dict):

    template **=** get\_template(template\_src)

    html **=** template.render(context\_dict)

    result **=** BytesIO()

    pdf **=** pisa.pisaDocument(BytesIO(html.encode("UTF-8")), result)

**if** **not** pdf.err:

**return** HttpResponse(result.getvalue(), content\_type**=**'application/pdf')

**return** **None**

and this is the verify ticket view  
@login\_required

@vendor\_required

**def** verify\_ticket(request, event\_id):

    event **=** get\_object\_or\_404(Event, id**=**event\_id)

    vendor **=** request.user

*# Ensure the correct tickets are being filtered by event and vendor*

    tickets **=** Ticket.objects.filter(event**=**event, vendor**=**vendor)

**if** request.method **==** 'POST':

        ticket\_number **=** request.POST.get('ticket\_number', '').strip()

*# Case-insensitive search and check if the ticket exists*

        ticket **=** tickets.filter(ticket\_number\_\_iexact**=**ticket\_number).first()

**if** ticket:

**if** ticket.verified:

**return** JsonResponse({'status': 'error', 'message': 'This ticket has already been verified.'})

**else**:

*# Mark the ticket as verified*

                ticket.verified **=** **True**

                ticket.save()

*# Safely retrieve the category title*

                ticket\_category\_title **=** 'N/A'

**if** ticket.ticket\_category:

                    ticket\_category\_title **=** ticket.ticket\_category.category\_title

**return** JsonResponse({

                    'status': 'success',

                    'ticket\_category': ticket\_category\_title,

                    'customer\_username': ticket.customer\_username **or** 'N/A',

                    'purchase\_date': ticket.purchase\_date.strftime('%Y-%m-%d %H:%M:%S'),

                })

**else**:

**return** JsonResponse({'status': 'error', 'message': 'Ticket not found.'})

**return** render(request, 'tickets/verify\_ticket.html', {'event': event})

since the action is part of the pos i need you to have the information so that u can add asomething to cater for when the entity is a pos agent  
and i need a decorator for a pos so that some things are only accessed by them

so for all the pages styling should be done using bootsrap css only and the make sure we have all the necsarry pages , urls, and views for this to work