Django Project Handover Document

# 1. Project Overview

This document provides a technical handover summary for the Django backend project.

# 2. Django Apps and Models

## 2.1 App: accounts

Models defined in this app:

from django.db import models  
  
# Create your models here.

## 2.2 App: pmc\_api

Models defined in this app:

from django.contrib.auth.models import User, Group  
from django.core.validators import MinLengthValidator, RegexValidator  
from django.contrib.gis.db import models  
from django.db.models import JSONField  
  
from pmc\_api.models\_choices import \*  
from pmc\_api.utils import validate\_latitude, validate\_longitude  
from rest\_framework import serializers  
import uuid  
import os  
from datetime import datetime, timedelta  
from django.utils import timezone  
from django.contrib.gis.geos import Point  
from simple\_history.models import HistoricalRecords  
  
class TblDivisions(models.Model):  
 # gid = models.AutoField()  
 division\_id = models.AutoField(primary\_key=True)  
 division\_name = models.CharField(max\_length=254)  
 division\_code = models.CharField(max\_length=254)  
  
 # geom = models.GeometryField(srid=0, blank=True, null=True)  
  
 def \_\_str\_\_(self):  
 return self.division\_name  
  
 class Meta:  
 managed = False  
 db\_table = 'tbl\_divisions'  
 verbose\_name\_plural = "Divisions"  
  
  
class TblDistricts(models.Model):  
 # gid = models.AutoField()  
 district\_id = models.IntegerField(primary\_key=True)  
 division = models.ForeignKey('TblDivisions', models.DO\_NOTHING)  
 district\_name = models.CharField(max\_length=254)  
 district\_code = models.CharField(max\_length=254)  
 short\_name = models.CharField(max\_length=3)  
 pitb\_district\_id = models.IntegerField(null=True, blank=True)  
 geom = models.GeometryField(srid=4326, null=True, blank=True)  
  
 # geom = models.GeometryField(srid=0, blank=True, null=True)  
  
 def \_\_str\_\_(self):  
 return self.district\_name  
  
 def get\_district\_by\_coordinates(lat, lon):  
 point = Point(lon, lat, srid=4326) # Create a Point in SRID 4326  
 district = TblDistricts.objects.filter(geom\_\_contains=point).first() # Check which district contains this point  
   
 if district:  
 return district.district\_name  
 return "District not found"  
  
 class Meta:  
 managed = False  
 db\_table = 'tbl\_districts'  
 ordering = ['district\_name', ]  
 verbose\_name\_plural = "Districts"  
 indexes = [  
 models.Index(fields=['district\_code'], name='idx\_district\_code'),  
 models.Index(fields=['short\_name'], name='idx\_district\_short\_name'),  
 ]  
  
  
  
class TblTehsils(models.Model):  
 # gid = models.AutoField()  
 tehsil\_id = models.AutoField(primary\_key=True)  
 district = models.ForeignKey(TblDistricts, models.DO\_NOTHING)  
 division = models.ForeignKey(TblDivisions, models.DO\_NOTHING)  
 tehsil\_name = models.CharField(max\_length=254)  
 tehsil\_code = models.CharField(unique=True, max\_length=254)  
  
 # geom = models.GeometryField(srid=0, blank=True, null=True)  
 # extent = models.CharField(max\_length=254, blank=True, null=True)  
 def \_\_str\_\_(self):  
 return self.tehsil\_name  
  
 class Meta:  
 managed = False  
 db\_table = 'tbl\_tehsils'  
 ordering = ['tehsil\_name', ]  
 verbose\_name\_plural = "Tehsils"  
 indexes = [  
 models.Index(fields=['district'], name='idx\_tehsil\_district'),  
 models.Index(fields=['tehsil\_code'], name='idx\_tehsil\_code'),  
 ]  
  
  
  
def default\_value\_uuid():  
 return uuid.uuid4()  
  
  
class ApplicantDetail(models.Model):  
 registration\_for = models.CharField(max\_length=10, choices=REG\_TYPE\_CHOICES, null=True)  
 first\_name = models.CharField(max\_length=255)  
 last\_name = models.CharField(max\_length=255, blank=True, null=True)  
 applicant\_designation = models.CharField(max\_length=255, blank=True, null=True)  
 gender = models.CharField(max\_length=100, choices=GENDER\_CHOICES)  
 cnic = models.CharField(max\_length=15, help\_text='XXXXX-XXXXXXX-X', validators=[  
 MinLengthValidator(15),  
 RegexValidator(  
 regex=r'^\d{5}-\d{7}-\d{1}$',  
 message="CNIC must be in the format XXXXX-XXXXXXX-X."  
 )  
 ], )  
 email = models.EmailField(max\_length=255, blank=True, null=True)  
 mobile\_operator = models.CharField(max\_length=15, choices=MOBILE\_NETWORK\_CHOICES, blank=True, null=True)  
 mobile\_no = models.CharField(max\_length=10, help\_text='3001234567', validators=[  
 MinLengthValidator(10), # Ensures minimum length is 10  
 RegexValidator(  
 regex=r'^\d{10}$',  
 message="Mobile number must be exactly 10 digits, e.g., '3001234567'."  
 )  
 ], )  
 application\_status = models.CharField(max\_length=20, choices=APPLICATION\_STATUS\_CHOICES, default='Created')  
 tracking\_number = models.CharField(max\_length=100, null=True)  
 remarks = models.TextField(null=True, blank=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 assigned\_group = models.CharField(max\_length=100, null=True, choices=USER\_GROUPS)  
 tracking\_hash = models.CharField(  
 max\_length=36, # Standard length for a UUID string  
 default=default\_value\_uuid,  
 editable=False,  
 unique=False  
 )  
 history = HistoricalRecords()  
 def \_\_str\_\_(self):  
 return f"{self.first\_name} {self.last\_name}"  
  
 def save(self, \*args, \*\*kwargs):  
 # Ensure the object is saved only once and the primary key is set correctly  
 is\_new\_record = self.pk is None # Check if this is a new record  
  
 # If it's a new record, set the primary key to None so that Django can auto-generate it  
 if is\_new\_record:  
 self.pk = None # Ensures auto-generation of primary key by Django  
  
 # Check if the application\_status is 'Submitted' for both new and existing records  
 if self.application\_status == 'Submitted':  
 # For existing records, set assigned\_group to 'LSO' if it's None or 'APPLICANT'  
 if not is\_new\_record:  
 existing\_record = ApplicantDetail.objects.filter(pk=self.pk).first()  
 if existing\_record and (existing\_record.assigned\_group is None or existing\_record.assigned\_group == 'APPLICANT'):  
 self.assigned\_group = 'LSO'  
 # For new records, set assigned\_group to 'LSO' if it's None or 'APPLICANT'  
 elif self.assigned\_group is None or self.assigned\_group == 'APPLICANT':  
 self.assigned\_group = 'LSO'  
   
 # Create a record in the ApplicationSubmitted model if it doesn't exist  
 if not ApplicationSubmitted.objects.filter(applicant=self).exists():  
 ApplicationSubmitted.objects.create(applicant=self)  
  
 # Check if a BusinessProfile exists for this applicant  
 if hasattr(self, 'businessprofile') and self.businessprofile:  
 business\_profile = self.businessprofile  
 district = business\_profile.district  
  
 # Ensure district and registration\_for exist  
 if district and self.registration\_for:  
 district\_code = district.short\_name or district.district\_name[:3].upper() or "XXX" # Use "XXX" if short\_name is missing  
 registration\_code = self.registration\_for[:3].upper() # First 3 letters of registration\_for  
 applicant\_id = str(self.id).zfill(3) # Zero-padded applicant ID  
  
 # Generate tracking\_number  
 self.tracking\_number = f"{district\_code}-{registration\_code}-{applicant\_id}"  
  
 # Save the instance only once, after all modifications  
 super().save(\*args, \*\*kwargs)  
  
  
 class Meta:  
 indexes = [  
 models.Index(fields=['application\_status'], name='idx\_app\_status'),  
 models.Index(fields=['assigned\_group'], name='idx\_assigned\_group'),  
 models.Index(fields=['created\_by'], name='idx\_created\_by'),  
 models.Index(fields=['tracking\_number'], name='idx\_tracking\_number'),  
 models.Index(fields=['application\_status', 'assigned\_group'], name='idx\_status\_group'),  
 ]  
  
class ApplicationSubmitted(models.Model):  
 applicant = models.OneToOneField(ApplicantDetail, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='submittedapplication')  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 history = HistoricalRecords()  
class BusinessProfile(models.Model):  
 entity\_type = models.CharField(  
 max\_length=20, choices=ENTITY\_TYPE\_CHOICES, default='Individual'  
 )  
 applicant = models.OneToOneField(ApplicantDetail, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='businessprofile')  
 tracking\_number = models.CharField(max\_length=100, unique=True, blank=True, null=True)  
 # If Individual  
 name = models.CharField(max\_length=255, blank=True, null=True)  
 ntn\_strn\_pra\_no\_individual = models.CharField(max\_length=20, blank=True, null=True)  
 # If Company/Corporation/Partnership  
 business\_name = models.CharField(max\_length=255, blank=True, null=True)  
 business\_registration\_type = models.CharField(  
 max\_length=50, choices=BUSINESS\_REGISTRATION\_CHOICES, blank=True, null=True  
 )  
 business\_registration\_no = models.CharField(max\_length=50, blank=True, null=True)  
 ntn\_strn\_pra\_no\_company = models.CharField(max\_length=20, blank=True, null=True) # Masking can be applied in forms  
 working\_days = models.IntegerField(choices=((5, 5), (6, 6), (7, 7)), default=5,  
 help\_text='working days in the week', blank=True, null=True)  
 commencement\_date = models.DateField(help\_text='Date since commencement of Business', blank=True, null=True)  
 no\_of\_workers = models.IntegerField(help\_text='Number of workers (including contract labour)', blank=True,  
 null=True)  
 # Address Detail  
 district = models.ForeignKey(TblDistricts, on\_delete=models.CASCADE, db\_column='district\_id',  
 verbose\_name="District", blank=True, null=True)  
 tehsil = models.ForeignKey(TblTehsils, on\_delete=models.CASCADE, db\_column='tehsil\_id',  
 verbose\_name="Tehsil", blank=True, null=True)  
 city\_town\_village = models.CharField(max\_length=256, help\_text="Name of City/Town or Village", blank=True,  
 null=True)  
 postal\_address = models.TextField(blank=True, null=True)  
 postal\_code = models.CharField(max\_length=10, blank=True, null=True)  
 location\_latitude = models.DecimalField(max\_digits=9, decimal\_places=6, validators=[validate\_latitude],  
 help\_text='Format: XX.XXXXXX, Range: 20.000000 to 40.000000, Unit: Decimal Degree',  
 blank=True, null=True)  
 location\_longitude = models.DecimalField(max\_digits=9, decimal\_places=6, validators=[validate\_longitude],  
 help\_text='Format: XX.XXXXXX, Range: 60.000000 to 80.000000,Unit: Decimal Degree',  
 blank=True, null=True)  
 # Contact Detail  
 email = models.EmailField(max\_length=255, blank=True, null=True)  
 mobile\_operator = models.CharField(max\_length=15, choices=MOBILE\_NETWORK\_CHOICES, blank=True, null=True)  
 mobile\_no = models.CharField(max\_length=10, help\_text='3001234567', validators=[  
 MinLengthValidator(10), # Ensures minimum length is 10  
 RegexValidator(  
 regex=r'^\d{10}$',  
 message="Mobile number must be exactly 10 digits, e.g., '3001234567'."  
 )  
 ], blank=True, null=True)  
 phone\_no = models.CharField(max\_length=12, help\_text='042-12345678', blank=True, null=True)  
 website\_address = models.URLField(blank=True, null=True)  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, related\_name='businessprofilecreatedby')  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return self.business\_name or self.name  
  
 class Meta:  
 indexes = [  
 models.Index(fields=['district'], name='idx\_district'),  
 models.Index(fields=['tehsil'], name='idx\_tehsil'),  
 models.Index(fields=['tracking\_number'], name='idx\_bp\_tracking\_number'),  
 ]  
  
  
class PlasticItems(models.Model):  
 item\_name = models.CharField(max\_length=255, unique=True) # single use plastic item name  
  
  
class Products(models.Model):  
 product\_name = models.CharField(max\_length=255, unique=True)  
  
  
class ByProducts(models.Model):  
 product\_name = models.CharField(max\_length=255, unique=True)  
  
  
class Producer(models.Model):  
 applicant = models.OneToOneField(ApplicantDetail, on\_delete=models.CASCADE)  
 tracking\_number = models.CharField(max\_length=100, blank=True, null=True)  
  
 # Registration details  
 registration\_required\_for = models.JSONField(blank=True, null=True) # Stores ManufacturingType[]  
 registration\_required\_for\_other = models.JSONField(blank=True, null=True) # Stores ManufacturingType[]  
 plain\_plastic\_sheets\_for\_food\_wrapping = models.JSONField(blank=True, null=True) # Stores SingleUseSheet[]  
 packaging\_items = models.JSONField(blank=True, null=True) # Stores PackagingItems[]  
  
 # Machine and capacity details  
 number\_of\_machines = models.CharField(max\_length=255, blank=True, null=True) # Stores string  
 total\_capacity\_value = models.FloatField(blank=True, null=True)  
  
 # Date of setting up  
 date\_of\_setting\_up = models.DateField(blank=True, null=True)  
  
 # Waste management  
 total\_waste\_generated\_value = models.FloatField(blank=True, null=True)  
 has\_waste\_storage\_capacity = models.CharField(max\_length=255, blank=True, null=True,  
 choices=[('Available', 'Available'),  
 ('Not Available', 'Not Available')])  
 waste\_disposal\_provision = models.CharField(max\_length=255, blank=True, null=True,  
 choices=[('Available', 'Available'),  
 ('Not Available', 'Not Available')])  
 registration\_required\_for\_other\_other\_text = models.CharField(max\_length=1024, blank=True, null=True)  
  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
 # raw material detail  
 # Documents  
 # flow\_diagram = models.FileField(upload\_to='diagrams/', blank=True, null=True)  
 # consent\_permit = models.FileField(upload\_to='permit/', blank=True, null=True)  
 history = HistoricalRecords()  
  
class RawMaterial(models.Model):  
 producer = models.ForeignKey(Producer, on\_delete=models.CASCADE)  
 material\_name = models.CharField(max\_length=255, unique=True)  
 material\_description = models.CharField(max\_length=255, blank=True, null=True)  
 material\_quantity\_value = models.FloatField(blank=True, null=True)  
 material\_quantity\_unit = models.FloatField(blank=True, null=True)  
 material\_utilized\_quantity\_value = models.FloatField(blank=True, null=True)  
 material\_utilized\_quantity\_unit = models.FloatField(blank=True, null=True)  
 material\_import\_bought = models.CharField(max\_length=255, blank=True, null=True, choices=IMPORT\_BOUGHT)  
 name\_seller\_importer = models.CharField(max\_length=255, blank=True, null=True)  
 is\_importer\_form\_filled = models.BooleanField(default=False)  
  
  
class Consumer(models.Model):  
 applicant = models.OneToOneField('ApplicantDetail', on\_delete=models.CASCADE)  
 registration\_required\_for = models.JSONField(default=list, blank=True) # Categories of Single Use Plastics  
 registration\_required\_for\_other = models.JSONField(default=list, blank=True) # Categories for Other Plastics  
 plain\_plastic\_sheets\_for\_food\_wrapping = models.JSONField(default=list, blank=True, null=True) # Additional Options  
 packaging\_items = models.JSONField(default=list, blank=True, null=True) # Additional Packaging Items  
 consumption = models.CharField(max\_length=100, blank=True, null=True) # Consumption (Kg per Day)  
 provision\_waste\_disposal\_bins = models.CharField(  
 max\_length=3, choices=[('Yes', 'Yes'), ('No', 'No')], default='No'  
 ) # Provision of Waste Disposal Bins  
 no\_of\_waste\_disposable\_bins = models.PositiveIntegerField(blank=True, null=True) # Number of Waste Disposal Bins  
 segregated\_plastics\_handed\_over\_to\_registered\_recyclers = models.CharField(  
 max\_length=3, choices=[('Yes', 'Yes'), ('No', 'No')], default='No'  
 ) # Segregated Plastics handed over to recyclers  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True,)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, related\_name='consumercreatedby')  
  
 registration\_required\_for\_other\_other\_text = models.CharField(max\_length=1024, blank=True, null=True)  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return self.applicant.first\_name  
  
  
class Collector(models.Model):  
 # Categories of Single Use Plastics  
 applicant = models.OneToOneField(ApplicantDetail, on\_delete=models.CASCADE, blank=True, null=True, )  
 registration\_required\_for = models.JSONField(  
 blank=True,  
 null=True,  
 help\_text="Categories of Single Use Plastics (e.g., ['Carry bags', 'Packaging except food'])"  
 )  
  
 # Categories for Other Plastics  
 registration\_required\_for\_other = models.JSONField(  
 blank=True,  
 null=True,  
 help\_text="Categories for Other Plastics (e.g., ['Plastic Utensils', 'PET Bottles'])"  
 )  
  
 # Source of Disposal  
 selected\_categories = models.JSONField(  
 blank=True,  
 null=True,  
 help\_text=(  
 "Source of Disposal, with details for each category. "  
 "Example: [{'category': 'Recycler', 'address': '123 Street Name'}, {'category': 'Landfill Site', 'address': '456 Another St'}]"  
 )  
 )  
  
 # Collection details  
 total\_capacity\_value = models.FloatField(  
 blank=True,  
 null=True,  
 help\_text="Collection in Kg per day"  
 )  
 number\_of\_vehicles = models.PositiveIntegerField(  
 blank=True,  
 null=True,  
 help\_text="Number of vehicles for collection"  
 )  
 number\_of\_persons = models.PositiveIntegerField(  
 blank=True,  
 null=True,  
 help\_text="Number of persons for collection"  
 )  
 registration\_required\_for\_other\_other\_text = models.CharField(max\_length=1024, blank=True, null=True)  
  
 # Metadata  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, related\_name='collectorcreatedby')  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return f"Collector ID: {self.id}, Collection Capacity: {self.total\_capacity\_value} Kg/day"  
  
  
class Recycler(models.Model):  
 applicant = models.OneToOneField(ApplicantDetail, on\_delete=models.CASCADE)  
 selected\_categories = models.JSONField(default=list) # Stores categories and their waste details  
 plastic\_waste\_acquired\_through = models.JSONField(  
 default=list, # Default to an empty list  
 blank=True  
 )  
  
 has\_adequate\_pollution\_control\_systems = models.CharField(  
 max\_length=10,  
 choices=[('Yes', 'Yes'), ('No', 'No')],  
 default='No'  
 )  
  
 pollution\_control\_details = models.TextField(blank=True, null=True)  
  
 registration\_required\_for\_other\_other\_text = models.CharField(max\_length=1024, blank=True, null=True)  
  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, related\_name='recyclercreatedby')  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return self.applicant.first\_name  
  
 # Method to sum up wasteCollection from JSONField  
 def get\_total\_waste\_collected(self):  
 return sum(  
 float(item.get("wasteCollection", 0) or 0) for item in self.selected\_categories  
 )  
  
 # Method to sum up wasteDisposal from JSONField  
 def get\_total\_waste\_disposed(self):  
 return sum(  
 float(item.get("wasteDisposal", 0) or 0) for item in self.selected\_categories  
 )  
  
  
class ApplicationAssignment(models.Model):  
 applicant = models.ForeignKey(ApplicantDetail, on\_delete=models.CASCADE, related\_name='applicationassignment')  
 assigned\_group = models.CharField(max\_length=100, null=True, choices=USER\_GROUPS)  
 remarks = models.TextField(null=True)  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='applicationassignmentupdatedby')  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='applicationassignmentcreatedby')  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return self.applicant.first\_name  
  
 class Meta:  
 indexes = [  
 models.Index(fields=['applicant'], name='idx\_applicant\_assignment'),  
 models.Index(fields=['assigned\_group'], name='idx\_assigned\_group\_assignment'),  
 models.Index(fields=['assigned\_group', 'created\_by'], name='idx\_group\_created\_assignment'),  
 ]  
  
def upload\_to\_with\_uuid(instance, filename):  
 """  
 Generates a unique filename by prepending a UUID to the original filename.  
 """  
 original\_name, ext = os.path.splitext(filename) # Separate the original name and extension  
 unique\_filename = f"{uuid.uuid4()}\_{original\_name}{ext}" # Prepend UUID and keep original name  
 return os.path.join('media/documents/', unique\_filename)  
  
class ApplicantDocuments(models.Model):  
 applicant = models.ForeignKey(ApplicantDetail, on\_delete=models.CASCADE, related\_name='applicationdocument')  
 document = models.FileField(upload\_to=upload\_to\_with\_uuid) # Use custom upload\_to  
 document\_description = models.CharField(max\_length=255)  
 updated\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='applicationdocumentupdatedby')  
 updated\_at = models.DateTimeField(auto\_now=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True,  
 related\_name='applicationdocumentcreatedby')  
 history = HistoricalRecords()  
  
 def \_\_str\_\_(self):  
 return self.applicant.first\_name  
  
 class Meta:  
 indexes = [  
 models.Index(fields=['applicant'], name='idx\_document\_applicant'),  
 models.Index(fields=['created\_by'], name='idx\_document\_created\_by'),  
 ]  
  
# User Profile Model (OneToOne with User)  
class UserProfile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE) # One-to-One with User  
 district = models.ForeignKey(TblDistricts, on\_delete=models.CASCADE, db\_column='district\_id',  
 verbose\_name="District", blank=True, null=True, related\_name='userprofile')  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return f"{self.user.username} - {self.district.short\_name if self.district else 'No District'}"  
  
class GroupSerializer(serializers.ModelSerializer):  
 district\_id = serializers.SerializerMethodField()  
 district\_name = serializers.SerializerMethodField()  
 class Meta:  
 model = Group  
 fields = ['id', 'name', 'district\_id', 'district\_name']  
  
 def get\_district\_id(self, obj):  
 user = self.context['request'].user  
 try:  
 return user.userprofile.district.district\_id # Fetch district\_id from UserProfile  
 except UserProfile.DoesNotExist:  
 return None # Return None if no district assigned  
  
 def get\_district\_name(self, obj):  
 user = self.context['request'].user  
 try:  
 return user.userprofile.district.district\_name # Fetch district\_name from UserProfile  
 except UserProfile.DoesNotExist:  
 return None # Return None if no district assigned  
  
  
class PSIDTracking(models.Model):  
 # Input data fields  
 applicant = models.ForeignKey('ApplicantDetail', on\_delete=models.CASCADE, related\_name='psid\_tracking', null=True, blank=True)  
 dept\_transaction\_id = models.CharField(max\_length=50)  
 due\_date = models.DateField()  
 expiry\_date = models.DateTimeField()  
 amount\_within\_due\_date = models.DecimalField(max\_digits=10, decimal\_places=2)  
 amount\_after\_due\_date = models.DecimalField(max\_digits=10, decimal\_places=2, null=True, blank=True)  
 consumer\_name = models.CharField(max\_length=255)  
 mobile\_no = models.CharField(max\_length=15)  
 cnic = models.CharField(max\_length=13)  
 email = models.EmailField(null=True, blank=True)  
 district\_id = models.IntegerField()  
 amount\_bifurcation = JSONField() # Stores bifurcation data as JSON  
  
 # Response data fields  
 consumer\_number = models.CharField(max\_length=50, unique=True, null=True, blank=True, verbose\_name="PSID")  
 status = models.CharField(max\_length=50, default="Pending")  
 message = models.TextField(null=True, blank=True)  
  
 # New fields for payment details  
 payment\_status = models.CharField(max\_length=10, default="UNPAID") # UNPAID or PAID  
 amount\_paid = models.DecimalField(max\_digits=10, decimal\_places=2, null=True, blank=True)  
 paid\_date = models.DateField(null=True, blank=True)  
 paid\_time = models.TimeField(null=True, blank=True)  
 bank\_code = models.CharField(max\_length=10, null=True, blank=True)  
   
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, null=True, blank=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 history = HistoricalRecords()  
 def \_\_str\_\_(self):  
 return f"PSID {self.consumer\_number or 'Pending'} - {self.dept\_transaction\_id}"  
  
class ApplicantFieldResponse(models.Model):  
 applicant = models.ForeignKey('ApplicantDetail', on\_delete=models.CASCADE, related\_name='field\_responses')  
 field\_key = models.CharField(max\_length=255) # Key from `keyToTitleMapping`  
 response = models.CharField(max\_length=3, choices=[('Yes', 'Yes'), ('No', 'No')], default='Yes')  
 comment = models.TextField(null=True, blank=True) # Only populated if response is 'No'  
   
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, null=True, blank=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return f"{self.field\_key} - {self.response}"  
  
class ApplicantManualFields(models.Model):  
 applicant = models.OneToOneField(  
 ApplicantDetail,  
 on\_delete=models.CASCADE,  
 related\_name='manual\_fields'  
 )  
 # Latitude & Longitude  
 latitude = models.DecimalField(  
 max\_digits=9,  
 decimal\_places=6,  
 null=True,  
 blank=True  
 )  
 longitude = models.DecimalField(  
 max\_digits=9,  
 decimal\_places=6,  
 null=True,  
 blank=True  
 )  
  
 # Producer-related Fields  
 list\_of\_products = models.TextField(null=True, blank=True)  
 list\_of\_by\_products = models.TextField(null=True, blank=True)  
 raw\_material\_imported = models.TextField(null=True, blank=True)  
 seller\_name\_if\_raw\_material\_bought = models.CharField(max\_length=255, null=True, blank=True)  
 self\_import\_details = models.TextField(null=True, blank=True)  
 raw\_material\_utilized = models.TextField(null=True, blank=True)  
 compliance\_thickness\_75 = models.CharField(  
 max\_length=3,  
 choices=[('Yes', 'Yes'), ('No', 'No')],  
 null=True,  
 blank=True  
 )  
 valid\_consent\_permit\_building\_bylaws = models.CharField(  
 max\_length=3,  
 choices=[('Yes', 'Yes'), ('No', 'No')],  
 null=True,  
 blank=True  
 )  
 stockist\_distributor\_list = models.TextField(null=True, blank=True)  
  
 # Consumer-related Field  
 procurement\_per\_day = models.CharField(  
 max\_length=100,  
 null=True,  
 blank=True,  
 help\_text="Procurement in Kg per day"  
 )  
  
 # Recycler-related Fields  
 no\_of\_workers = models.PositiveIntegerField(null=True, blank=True)  
 labor\_dept\_registration\_status = models.CharField(  
 max\_length=3,  
 choices=[('Yes', 'Yes'), ('No', 'No')],  
 null=True,  
 blank=True  
 )  
 occupational\_safety\_and\_health\_facilities = models.TextField(null=True, blank=True)  
 adverse\_environmental\_impacts = models.TextField(null=True, blank=True)  
  
 # Optional Timestamps / Audit Fields  
 created\_by = models.ForeignKey(  
 User,  
 on\_delete=models.SET\_NULL,  
 null=True,  
 blank=True,  
 related\_name='applicantmanualfields\_created'  
 )  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_by = models.ForeignKey(  
 User,  
 on\_delete=models.SET\_NULL,  
 null=True,  
 blank=True,  
 related\_name='applicantmanualfields\_updated'  
 )  
 updated\_at = models.DateTimeField(auto\_now=True)  
 history = HistoricalRecords()  
 def \_\_str\_\_(self):  
 return f"Manual Fields for {self.applicant} (ID: {self.id})"  
  
class ApplicantFee(models.Model):  
 applicant = models.ForeignKey(ApplicantDetail, on\_delete=models.CASCADE, related\_name="applicantfees")  
 fee\_amount = models.DecimalField(max\_digits=10, decimal\_places=2)  
 is\_settled = models.BooleanField(default=False) # Indicates if the fee is settled  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 reason = models.TextField(blank=True, null=True) # Reason or purpose for the fee (optional)  
 history = HistoricalRecords()  
 class Meta:  
 ordering = ['-created\_at'] # Order by latest fee first  
  
 def \_\_str\_\_(self):  
 return f"Fee for {self.applicant} - Rs. {self.fee\_amount}"  
  
 class Meta:  
 indexes = [  
 models.Index(fields=['applicant'], name='idx\_fee\_applicant'),  
 models.Index(fields=['is\_settled'], name='idx\_fee\_is\_settled'),  
 ]  
  
class ServiceConfiguration(models.Model):  
 service\_name = models.CharField(max\_length=100, unique=True)  
 base\_url = models.URLField(help\_text="Base endpoint of the service")  
 auth\_endpoint = models.URLField(help\_text="Authentication endpoint")  
 generate\_psid\_endpoint = models.URLField(help\_text="PSID generation endpoint")  
 transaction\_status\_endpoint= models.URLField(help\_text="Transaction Status endpoint", null=True, blank=True)  
 # If you also store credentials  
 client\_id = models.CharField(max\_length=200)  
 client\_secret = models.CharField(max\_length=500)  
  
 updated\_at = models.DateTimeField(auto\_now=True)  
 history = HistoricalRecords()  
   
 def \_\_str\_\_(self):  
 return self.service\_name  
  
class ExternalServiceToken(models.Model):  
 service\_name = models.CharField(max\_length=100)  
 access\_token = models.TextField()  
 expires\_at = models.DateTimeField()  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 history = HistoricalRecords()  
   
 def is\_expired(self):  
 # Give a little buffer (e.g. 30 seconds) to account for clock skew  
 return timezone.localtime() > self.expires\_at  
   
class ApiLog(models.Model):  
 """  
 Stores metadata about API calls made by our system, including request/response data.  
 """  
 service\_name = models.CharField(max\_length=100)  
 endpoint = models.CharField(max\_length=500)  
 request\_data = models.JSONField(null=True, blank=True)  
 response\_data = models.JSONField(null=True, blank=True)  
 status\_code = models.PositiveIntegerField(null=True, blank=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
  
 def \_\_str\_\_(self):  
 return f"{self.service\_name} - {self.endpoint} - {self.created\_at.strftime('%Y-%m-%d %H:%M:%S')}"  
   
  
class License(models.Model):  
 # If you want to store the specific role (Producer, Stockist, etc.)  
 # as a text field, you can do so directly or use choices:  
  
 license\_for = models.CharField(  
 max\_length=50,  
 default="producer", # or whichever default you want  
 verbose\_name="License For",  
 help\_text="Type of license issued (e.g., Producer, Stockist, Distributor, etc.)"  
 )  
  
 license\_number = models.CharField(  
 max\_length=100,  
 unique=False, # or True, depending on your rules  
 verbose\_name="License Number"  
 )  
  
 license\_duration = models.CharField(  
 max\_length=50,  
 verbose\_name="License Duration",  
 help\_text="e.g., '3 Years'"  
 )  
  
 owner\_name = models.CharField(  
 max\_length=200,  
 verbose\_name="Owner’s Name"  
 )  
  
 business\_name = models.CharField(  
 max\_length=200,  
 verbose\_name="Business Name"  
 )  
  
 types\_of\_plastics = models.CharField(  
 max\_length=200,  
 verbose\_name="Types of Plastics",  
 help\_text="e.g., 'ABC, DEF'"  
 )  
   
 particulars = models.CharField(  
 max\_length=200,  
 verbose\_name="Particulars",  
 help\_text="e.g., 'ABC, DEF'"  
 )  
   
 fee\_amount = models.DecimalField(max\_digits=10, decimal\_places=2)  
   
 address = models.CharField(  
 max\_length=300,  
 verbose\_name="Address"  
 )  
  
 date\_of\_issue = models.DateField(  
 verbose\_name="Date of Issue",  
 help\_text="e.g., '10.01.2025'"  
 )  
  
 # applicant\_id could be an integer if you have no Applicant model,  
 # or a ForeignKey if you do have an Applicant model  
 applicant\_id = models.IntegerField(  
 verbose\_name="Applicant ID",  
 help\_text="Link this license to an applicant record"  
 )  
 # If you have an Applicant model, do instead:  
 # applicant = models.ForeignKey(Applicant, on\_delete=models.CASCADE)  
  
 # Status field to indicate active/inactive  
 is\_active = models.BooleanField(  
 default=True,  
 verbose\_name="Is Active",  
 help\_text="Indicates whether the license is active."  
 )  
   
 # Audit fields  
 created\_at = models.DateTimeField(  
 auto\_now\_add=True,  
 verbose\_name="Created At"  
 )  
  
 created\_by = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
  
 # If you want to track the user who created it, use a ForeignKey to settings.AUTH\_USER\_MODEL  
 models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True)  
   
 history = HistoricalRecords()  
 class Meta:  
 # If you need a constraint that each license\_number can only appear once per date\_of\_issue:  
 # unique\_together = ("license\_number", "date\_of\_issue")  
 ordering = ["-created\_at"] # or whichever ordering you prefer  
  
  
 def formatted\_date\_of\_issue(self):  
 # Return dd.mm.yyyy  
 return self.date\_of\_issue.strftime("%d.%m.%Y")  
   
 def types\_of\_plastics\_truncated(self):  
 """  
 Truncate the text to the first `max\_length` characters.  
 If a comma exists within these characters, truncate up to the last comma.  
 Otherwise, truncate at `max\_length`.  
  
 Args:  
 text (str): The input string to be truncated.  
 max\_length (int): The maximum number of characters to retain.  
  
 Returns:  
 str: The truncated string.  
 """  
 text = self.types\_of\_plastics  
 max\_length=71  
 if not text:  
 return ""  
  
 if len(text) <= max\_length:  
 return text  
  
 substring = text[:max\_length]  
 last\_comma\_index = substring.rfind(',')  
  
 if last\_comma\_index != -1:  
 return substring[:last\_comma\_index]  
 else:  
 return substring  
   
 def license\_for\_formatted(self):  
 return "Stockist/Distributor/Supplier" if self.license\_for == "Consumer" else self.license\_for or "Not Specified"  
   
 def \_\_str\_\_(self):  
 return f"{self.license\_number} ({self.license\_for})"   
  
def upload\_affidavits\_to\_with\_uuid(instance, filename):  
 """  
 Generates a unique filename by prepending a UUID to the original filename.  
 """  
 original\_name, ext = os.path.splitext(filename) # Separate the original name and extension  
 unique\_filename = f"{uuid.uuid4()}\_{original\_name}{ext}" # Prepend UUID and keep original name  
 return os.path.join('media/affidavits/', unique\_filename)  
  
def upload\_inspections\_to\_with\_uuid(instance, filename):  
 """  
 Generates a unique filename by prepending a UUID to the original filename.  
 """  
 original\_name, ext = os.path.splitext(filename) # Separate the original name and extension  
 unique\_filename = f"{uuid.uuid4()}\_{original\_name}{ext}" # Prepend UUID and keep original name  
 return os.path.join('media/inspections/', unique\_filename)  
  
class InspectionReport(models.Model):  
 business\_name = models.CharField(max\_length=255)  
 business\_type = models.CharField(max\_length=50)  
  
 license\_number = models.CharField(max\_length=50, blank=True, null=True)  
  
 violation\_found = models.JSONField(blank=True, null=True)  
 violation\_type = models.JSONField(blank=True, null=True)  
 action\_taken = models.JSONField(blank=True, null=True)  
  
 plastic\_bags\_confiscation = models.FloatField(blank=True, null=True)  
 confiscation\_other\_plastics = models.JSONField(blank=True, null=True)  
 total\_confiscation = models.FloatField(blank=True, null=True)  
  
 other\_single\_use\_items = models.JSONField(blank=True, null=True)  
  
 latitude = models.FloatField(blank=True, null=True)  
 longitude = models.FloatField(blank=True, null=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
  
 # ✅ New Fields Added  
 inspection\_date = models.DateField(blank=True, null=True)  
 fine\_amount = models.FloatField(blank=True, null=True)  
 fine\_recovery\_status = models.CharField(  
 max\_length=20,   
 choices=[("Pending", "Pending"), ("Partial", "Partial"), ("Recovered", "Recovered")],  
 blank=True,  
 null=True  
 )  
 fine\_recovery\_date = models.DateField(blank=True, null=True)  
 recovery\_amount = models.FloatField(blank=True, null=True)  
 de\_sealed\_date = models.DateField(blank=True, null=True)  
 fine\_recovery\_breakup = models.JSONField(blank=True, null=True)  
  
 # ✅ Affidavit (File Upload)  
 affidavit = models.FileField(upload\_to=upload\_affidavits\_to\_with\_uuid, blank=True, null=True)  
 district = models.ForeignKey(TblDistricts, on\_delete=models.CASCADE, db\_column='district\_id',  
 verbose\_name="District", blank=True, null=True, related\_name='inspectionreport')  
  
 # ✅ Created By User (New Field)  
 created\_by = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True, related\_name="inspections")  
 history = HistoricalRecords()  
  
 confiscation\_receipt = models.FileField(  
 upload\_to=upload\_inspections\_to\_with\_uuid,  
 null=True,  
 blank=True  
 )  
  
 payment\_challan = models.FileField(  
 upload\_to=upload\_inspections\_to\_with\_uuid,  
 null=True,  
 blank=True  
 )  
  
 receipt\_book\_number = models.CharField(  
 max\_length=100,  
 null=True,  
 blank=True  
 )  
  
 receipt\_number = models.CharField(  
 max\_length=100,  
 null=True,  
 blank=True  
 )  
  
 def save(self, \*args, \*\*kwargs):  
 """Ensure all unique 'other\_single\_use\_items' are stored in the snapshot"""  
 super().save(\*args, \*\*kwargs) # Save report first  
  
 # Fetch or create the single record in `SingleUsePlasticsSnapshot`  
 snapshot, created = SingleUsePlasticsSnapshot.objects.get\_or\_create(id=1)  
  
 # Ensure other\_single\_use\_items is a list before updating the snapshot  
 if isinstance(self.other\_single\_use\_items, list):  
 snapshot.update\_snapshot(self.other\_single\_use\_items)  
  
 def \_\_str\_\_(self):  
 return f"{self.business\_name} - {self.business\_type}"  
  
class SingleUsePlasticsSnapshot(models.Model):  
 plastic\_items = models.JSONField(default=list) # Store unique items  
  
 def update\_snapshot(self, new\_items):  
 """Ensure the snapshot contains all unique items"""  
 current\_items = set(self.plastic\_items) # Convert to a set for uniqueness  
 current\_items.update(new\_items) # Add new items  
 self.plastic\_items = list(current\_items) # Convert back to a list  
 self.save()  
  
 def \_\_str\_\_(self):  
 return f"Snapshot of {len(self.plastic\_items)} Single Use Plastic Items"  
  
def upload\_plastic\_committee\_to\_with\_uuid(instance, filename):  
 """  
 Generates a unique filename by prepending a UUID to the original filename.  
 """  
 original\_name, ext = os.path.splitext(filename) # Separate the original name and extension  
 unique\_filename = f"{uuid.uuid4()}\_{original\_name}{ext}" # Prepend UUID and keep original name  
 return os.path.join('media/plastic\_committee/', unique\_filename)  
  
class DistrictPlasticCommitteeDocument(models.Model):  
 district = models.ForeignKey(TblDistricts, on\_delete=models.CASCADE, related\_name="committee\_documents")  
 document\_type = models.CharField(  
 max\_length=50, choices=[('Notification', 'Notification'), ('Minutes of Meeting', 'Minutes of Meeting')]  
 )  
 title = models.CharField(max\_length=1024, blank=True, null=True)  
 document = models.FileField(upload\_to=upload\_plastic\_committee\_to\_with\_uuid)  
 uploaded\_at = models.DateTimeField(auto\_now\_add=True)  
 uploaded\_by = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)  
 document\_date = models.DateField(null=True, blank=True) # New field for document date  
  
 history = HistoricalRecords()  
 def \_\_str\_\_(self):  
 return f"{self.district.district\_name} - {self.document\_type} ({self.uploaded\_at.date()})"  
  
  
# models.py  
class AuditLog(models.Model):  
 ACTION\_CHOICES = [  
 ("create", "Create"),  
 ("update", "Update"),  
 ("delete", "Delete"),  
 ("login", "Login"),  
 ("logout", "Logout"),  
 ]  
  
 user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)  
 action = models.CharField(max\_length=10, choices=ACTION\_CHOICES)  
 model\_name = models.CharField(max\_length=255, null=True, blank=True)  
 object\_id = models.CharField(max\_length=255, null=True, blank=True)  
 description = models.TextField()  
 ip\_address = models.GenericIPAddressField(null=True, blank=True)  
 timestamp = models.DateTimeField(auto\_now\_add=True)  
  
 def \_\_str\_\_(self):  
 return f"{self.timestamp} - {self.user} - {self.action}"  
  
class AccessLog(models.Model):  
 user = models.ForeignKey(User, null=True, on\_delete=models.SET\_NULL)  
 model\_name = models.CharField(max\_length=100)  
 object\_id = models.CharField(max\_length=100)  
 method = models.CharField(max\_length=10) # GET, POST  
 ip\_address = models.GenericIPAddressField(null=True, blank=True)  
 timestamp = models.DateTimeField(auto\_now\_add=True)  
 endpoint = models.CharField(max\_length=255)  
  
  
def upload\_student\_card(instance, filename):  
 import uuid, os  
 ext = os.path.splitext(filename)[1]  
 return f"media/competition/student\_cards/{uuid.uuid4()}{ext}"  
  
class CompetitionRegistration(models.Model):  
 COMPETITION\_CHOICES = [  
 ('poster', 'Poster'),  
 ('painting', 'Painting'),  
 ('3d\_model', '3D Model')  
 ]  
 CATEGORY\_CHOICES = [  
 ('School', 'School/College'),  
 ('University', 'University')  
 ]  
 full\_name = models.CharField(max\_length=255)  
 institute = models.CharField(max\_length=255)  
 grade = models.CharField(max\_length=50)  
 category = models.CharField(max\_length=20, choices=CATEGORY\_CHOICES)  
 competition\_type = models.CharField(max\_length=20, choices=COMPETITION\_CHOICES)  
 mobile = models.CharField(max\_length=10)  
 student\_card\_front = models.ImageField(upload\_to=upload\_student\_card)  
 student\_card\_back = models.ImageField(upload\_to=upload\_student\_card, null=True, blank=True)  
 photo\_object = models.ImageField(upload\_to=upload\_student\_card, null=True, blank=True)  
 registration\_id = models.CharField(max\_length=50, unique=True, editable=False)  
 created\_at = models.DateTimeField(default=timezone.now)  
  
 def save(self, \*args, \*\*kwargs):  
 if not self.registration\_id:  
 import uuid  
 self.registration\_id = str(uuid.uuid4())[:8].upper()  
 super().save(\*args, \*\*kwargs)  
  
 def \_\_str\_\_(self):  
 return f"{self.full\_name} - {self.competition\_type}"