**Request (some mata data) - HTTP REQUEST**

**Apache/nginx**

**Wsgi (web server gateway interface -> how web server communicate**

**To application)**

**Middleware**

**Setting.py -> ROOT\_URLCONF =”project.urls” (start urls) first come**

**urls.py**

**Urlpatterns list scan top to bottom**

**Find view from imported file - request is first parameter of view func**

**database (models.py ) perform some sql operations**

**return a http response**

**conf.py**

**path function get two args**

**def \_path(route, view, kwargs=None, name=None, Pattern=None):**

**Resolver.py**

**Check url pattern**

**def \_check\_callback(self):**

[**https://raturi.in/blog/custom-mixins-django-class-based-views/**](https://raturi.in/blog/custom-mixins-django-class-based-views/%5C)

**@action**

**from rest\_framework.decorators import action**

**@action(detail=True, methods=["get"], name="user\_posts")**

**def user\_posts(self, request, pk):**

**target\_user = get\_object\_or\_404(User, pk=pk)**

**data = []**

**if (**

**target\_user != request.user**

**and target\_user.is\_private**

**and not FollowRequest.is\_friends(target\_user, request.user)**

**):**

**return Response({"data": data})**

**queryset = self.queryset.filter(created\_by=target\_user)**

**page = self.paginate\_queryset(queryset)**

**if page is not None:**

**serializer = self.get\_serializer(page, many=True)**

**return self.get\_paginated\_response(serializer.data)**

**serializer = self.get\_serializer(queryset, many=True)**

**return Response(serializer.data)**

**My MiddleWARE**

**def my\_middle(get\_response):**

**print("one time")**

**def myfunction(request):**

**response = get\_response**

**print("After Response")**

**return response**

**return myfunction**

**"apps.organization.middle.my\_middle"**

**Django core**

[**https://docs.djangoproject.com/en/4.1/ref/settings/#core-settings**](https://docs.djangoproject.com/en/4.1/ref/settings/#core-settings)

**STATIC**

**urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT) + static(**

**settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT**

**)**

**STATIC\_URL = "/static/"**

**MEDIA\_URL = "/media/"**

**MEDIA\_ROOT = os.path.join(BASE\_DIR, "media")**

**STATIC\_ROOT = BASE\_DIR / "staticfiles"**

DJANGO\_SETTINGS\_MODULE = "RoboKidz.settings.production"

**All values Serializer**

|  | **from rest\_framework import serializers** |
| --- | --- |
|  | **from .models import Song, Artist** |
|  |  |
|  | **class ArtistSerializer(serialisers.ModelSerializer):** |
|  | **songs = serializers.PrimaryKeyRelatedField(** |
|  | **many=True,** |
|  | **read\_only=False,** |
|  | **queryset=Song.objects.all()** |
|  | **)** |

**SHOW IMG IN ADMIN**

**models.py**

**def image\_tag(self):**

**from django.utils.html import escape, mark\_safe**

**try:**

**return mark\_safe(**

**'<img src="%s" width="80px" height="80px"/>' % escape(self.qrcode.url)**

**)**

**except Exception as e:**

**pass**

**image\_tag.short\_description = "Image"**

**image\_tag.allow\_tags = True**

**Admin.py**

**@admin.register(QRCode)**

**class QrCodeAdmin(admin.ModelAdmin):**

**fields = ('category',)**

**list\_display = ("id", "qr\_sr\_no", "category", "printdate", "image\_tag") *#qrcode***

**readonly\_fields = ( "qr\_sr\_no", "image\_tag")**

**DOWNLOAD BOTO3**

**from django.test import TestCase**

**import os**

***# Create your tests here.***

**import os**

**import boto3**

**from boto3.session import Session**

**AWS\_ACCESS\_KEY\_ID = ""**

**AWS\_SECRET\_ACCESS\_KEY = ""**

**session = Session(aws\_access\_key\_id= AWS\_ACCESS\_KEY\_ID , aws\_secret\_access\_key="")**

**s3 = session.resource('s3')**

**Service\_Name = "S3"**

**bu = "clapzbucket"**

**mybucket = s3.Bucket(bu)**

**for i in mybucket.objects.all():**

**print(i)**

**USER MEDIA MODELVIEWSET URL**

**context={"request": self.context["request"]}**

**Access\_KEY\_ID = ""**

**Access\_Key = "”**

**Service\_Name = "S3"**

**Region\_name= "ap-south-1"**

**Request user**

**user = self.context["request"].user**

**Seilizer two**

**class CategorySerializer(serializers.ModelSerializer):**

**videos = serializers.SerializerMethodField()**

**def get\_videos(self, obj):**

**videos = obj.video\_category.all()**

**return VideoResponseserilizer(videos, many=True).data**

**class Meta:**

**model = Category**

**fields = ("id", "name", "videos")**

**Data Paload**

**class SubscriptionApiView(viewsets.ModelViewSet):**

**queryset = Subscription.objects.all()**

**serializer\_class = SubscriptionSerializer**

**def get\_serializer(self, \*args, \*\*kwargs):**

**if self.request.method in ["POST", "PATCH", "PUT"]:**

**serializer\_class = self.get\_serializer\_class()**

**else:**

**serializer\_class = SubscriptionResponseSerializer**

**kwargs.setdefault("context", self.get\_serializer\_context())**

**return serializer\_class(\*args, \*\*kwargs)**

**def create\_task\_payload(self, before, after):**

**output = []**

**for i in before.keys():**

**output.append({**

**"name":i,**

**"before":before[i],**

**"after":after[i]**

**})**

**print(output)**

**return output**

**def update(self, request, \*args, \*\*kwargs):**

**partial = kwargs.pop('partial', False)**

**instance = self.get\_object()**

**befor\_save\_data= Subscription.objects.filter(id=int(self.kwargs["pk"])).first()**

**befor\_save\_data = dict(SubscriptionSerializer(instance=befor\_save\_data).data)**

**serializer =SubscriptionSerializer(instance, data=request.data, partial=partial)**

**serializer.is\_valid(raise\_exception=True)**

**self.perform\_update(serializer)**

**after\_save\_data = dict(serializer.data)**

**print("------",type(befor\_save\_data))**

**print()**

**print("------",type(after\_save\_data))**

**newdatapayload = self.create\_task\_payload(befor\_save\_data, after\_save\_data)**

***# print(newdatapayload)***

**return Response(serializer.data)**

**My code**

**class SubscriptionApiView(viewsets.ModelViewSet):**

**queryset = Subscription.objects.all()**

**serializer\_class = SubscriptionSerializer**

**def get\_serializer(self, \*args, \*\*kwargs):**

**if self.request.method in ["POST", "PATCH", "PUT"]:**

**serializer\_class = self.get\_serializer\_class()**

**else:**

**serializer\_class = SubscriptionResponseSerializer**

**kwargs.setdefault("context", self.get\_serializer\_context())**

**return serializer\_class(\*args, \*\*kwargs)**

**def update(self, request, \*args, \*\*kwargs):**

**partial = kwargs.pop('partial', False)**

**instance = self.get\_object()**

**obj= Subscription.objects.filter(id=int(self.kwargs["pk"])).values()**

**olddata = SubscriptionSerializer(obj,many=True)**

**olddata = list(obj)**

**serializer =SubscriptionSerializer(instance, data=request.data, partial=partial)**

**serializer.is\_valid(raise\_exception=True)**

**self.perform\_update(serializer)**

**obj= Subscription.objects.filter(id=int(self.kwargs["pk"])).values()**

**newdata = list(obj.values())**

**olddict = {}**

**for i in olddata:**

**for key,values in i.items():**

**olddict[key] = values**

**print(olddict)**

**newdict = {}**

**for j in newdata:**

**for key,values in j.items():**

**olddict[key]= values**

**print(newdict)**

**def data(olddict,newdict):**

**kk =[]**

**for i in olddict.keys():**

**kk.append({**

**"name": i,**

**"before":olddict[i],**

**"after":newdict[i],**

**})**

**print(kk)**

**return kk**

**jj = data(olddict,newdict)**

**Data base**

**def *Insert*(request):**

**kk = Students.objects.filter(FirstName="sam").values()**

**lst = list(kk)**

**ja = Students.objects.filter(FirstName="sam")**

**kmm = [{'id': 2,**

**'FirstName': 'sam',**

**'LastName': 'kumar',**

**'Email': 'sam@gmail.om',**

**'Contact': 12333}]**

***for* i *in* kmm:**

**print(i)**

**OVERRIDE UPDATE method in modelviewset**

**class SubscriptionApiView(viewsets.ModelViewSet):**

**queryset = Subscription.objects.all()**

**serializer\_class = SubscriptionSerializer**

**def get\_serializer(self, \*args, \*\*kwargs):**

**if self.request.method in ["POST", "PATCH", "PUT"]:**

**serializer\_class = self.get\_serializer\_class()**

**else:**

**serializer\_class = SubscriptionResponseSerializer**

**kwargs.setdefault("context", self.get\_serializer\_context())**

**return serializer\_class(\*args, \*\*kwargs)**

**def update(self, request, \*args, \*\*kwargs):**

**partial = kwargs.pop('partial', False)**

**instance = self.get\_object()**

**before\_data= Subscription.objects.filter(id=int(self.kwargs["pk"])).values()**

**print("This is before data",before\_data)**

**serializer =SubscriptionSerializer(instance, data=request.data, partial=partial)**

**serializer.is\_valid(raise\_exception=True)**

**self.perform\_update(serializer)**

**after = serializer.data**

**print()**

**print()**

**print("This is Afetr Data", after)**

**return Response(serializer.data)**

**Payload**

**class VerifyEmail(generics.GenericAPIView):**

**permission\_classes = []**

**serializer\_class = PrentEmailSerializer**

**def get(self, request):**

**token = request.GET.get("token")**

**payload = jwt.decode(token, settings.SECRET\_KEY, algorithms=['HS256'])**

**user = User.objects.get(id=payload['user\_id'])**

**if not user.parent\_email\_verified:**

**user.parent\_email\_verified = True**

**user.save()**

**return Response({"email": "succefull activated"}, status=status.HTTP\_200\_OK)**

**else:**

**return Response(status=status.HTTP\_400\_BAD\_REQUEST)**

**Update**

**q = get\_object\_or\_404(MyModel,pk=some\_value)**

**q.field1 = 'some value'**

**q.save()**

**Expire plan celery**

**@shared\_task(bind =True)**

**def ExpirePlan(self):**

**try:**

**time = datetime.now()**

**my\_string1 = time.isoformat(' ', 'seconds')**

**print(my\_string1)**

**expire\_obj = Subscription.objects.filter(subscription\_close\_date = my\_string1)**

**print("------------------",expire\_obj)**

**for obj in expire\_obj:**

***# email = expire\_obj.email***

***# subject = 'Expire RoboKidz Portal '***

***# message = f' Your Portal acress Expire Today {time}'***

***# email\_from = settings.EMAIL\_HOST***

**obj.active = True**

**obj.save()**

***# send\_mail(subject, message, email\_from, [email])***

**print("Run Sucessfully")**

**except Exception as e:**

**print(e)**

***# @shared\_task(bind= True)***

***# def send\_mail\_task(self):***

***# # print("Mail sending.......")***

***# # subject = 'welcome to Celery world'***

***# # message = 'This is celery test'***

***# # email\_from = EMAIL\_HOST\_USER***

***# # recipient\_list = ['user1@yopmail.com' ]***

***# # send\_mail( subject, message, email\_from, recipient\_list )***

***# # return "Mail has been sent........"***

**app.conf.beat\_schedule = {**

**'ExpirePlan': {**

**'task': 'apps.mobile.tasks.ExpirePlan',**

**'schedule': 1.0, *#every 30 seconds it will be called***

***#'args': (2,) you can pass arguments also if rquired***

**}**

**}**

**app.autodiscover\_tasks()**

**@app.task(bind=True)**

**def debug\_task(self):**

**print(f'Request: {self.request!r}')**

**DELETE From django\_celery\_beat\_periodictask;**

**Modelviewset Pk**

**def retrieve(self, request, \*args, \*\*kwargs):**

**param = kwargs**

**pk = param['pk']**

**return Subscription.objects.filter(id=pk)**

**return**

***# return super().retrieve(request, \*args, \*\*kwargs)***

**Call API Data**

**def get\_products(request):**

**response = requests.get("https://fakestoreapi.com/products").json()**

**for i in response:**

**print(i['price'])**

**Products.objects.create(title=i['title'],price =i['price'],category=i['category'],description=i['description'], image=i['image'])**

**return render(request,'show.html',{'response':response})**

**def get\_single(request,id):**

**response = requests.get("https://fakestoreapi.com/products/1").json()**

**return render(request,'one.html',{'response':response})**

**Export data as json file with all apps**

**python3 manage.py dumpdata users -o apps/users/fixtures/my.json**

**( python3 manage.py dumpdata users -o**

**)**

**Load json data in DB (when we already json file)**

**python3 manage.py loaddata apps/users/fixtures/role.json**

**Format in vs code – shift +alt + F ( window)**

**Shift + option+F - mac**

**Shift + alt +I - linux**

**#django rest change password (install django restassword)**

**from django\_rest\_passwordreset.views import ResetPasswordConfirm, GenericViewSet**

**from django\_rest\_passwordreset.views import ResetPasswordToken**

**Serilizer.py**

**class ChangePasswordSerializer(PasswordTokenSerializer):**

**confirm\_password = serializers.CharField(required=True)**

**def validate(self, attrs):**

**if attrs["password"] != attrs["confirm\_password"]:**

**raise serializers.ValidationError(**

**{"password": "Password fields didn't match."}**

**)**

**return attrs**

**views.py**

**class ResetPasswordConfirmView(ResetPasswordConfirm):**

**serializer\_class = ChangePasswordSerializer**

**"""**

**An Api ViewSet which provides a method to reset a password based on a unique token**

**"""**

**def post(self, request, reset\_password\_token, \*args, \*\*kwargs):**

**serializer = self.serializer\_class(data=request.data)**

**reset\_password\_token = ResetPasswordToken.objects.filter(**

**key=reset\_password\_token).first()**

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

**print("init")**

**if reset\_password\_token:**

**return super().post(request, \*args, \*\*kwargs)**

**else:**

**return Response({'message': "invaild Request or Token"})**

**Urls**

**path(**

**"password\_reset/",**

**include("django\_rest\_passwordreset.urls", namespace="password\_reset"),**

**),**

**path('confirm/<reset\_password\_token>/',views.ResetPasswordConfirmView.as\_view(),name="confirm\_passowrd"),**

**Email.py**

**from django.shortcuts import render,HttpResponse**

***# Create your views here.***

**from rest\_framework.decorators import api\_view**

**from rest\_framework.response import Response**

**from rest\_framework\_simplejwt.views import TokenObtainPairView**

**from rest\_framework import generics**

**from apps.organization.models import \***

**from rest\_framework.permissions import AllowAny, IsAuthenticated**

**from rest\_framework.decorators import api\_view, permission\_classes, authentication\_classes**

**from rest\_framework.authentication import SessionAuthentication**

**from .serializer import \***

**from . models import \***

***# Create your views here.***

**from my\_backend.settings.base import \***

**from django\_rest\_passwordreset.signals import reset\_password\_token\_created**

**from django\_rest\_passwordreset.signals import reset\_password\_token\_created**

**from django.core.mail import send\_mail**

**@receiver(reset\_password\_token\_created)**

**def password\_reset\_token\_created(sender, instance, reset\_password\_token, \*args, \*\*kwargs):**

***# email\_plaintext\_message = "http://127.0.0.1:8000{}?token={}".format(reverse('confirm/<token>/:confirm\_passowrd'), reset\_password\_token.key)***

***# # title:***

**email\_plaintext\_message = f"http://127.0.0.1:8000/confirm/{reset\_password\_token.key}"**

**subject ="Password Reset for {title}".format(title="Some website title")**

***# message:***

**message = email\_plaintext\_message**

***# from:***

**email\_from = EMAIL\_HOST**

***# to:***

**send\_mail(subject, message, email\_from, [reset\_password\_token.user.email])**

**print("send")**

**Django command fack data generate**

**Appname /management/commands**

***#dummy data***

**import random**

**from faker import Faker**

**fake = Faker()**

**fake.random.seed(4321)**

**from django.core.management.base import BaseCommand**

**class Command(BaseCommand):**

**help = "Display Info "**

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

**list = ["supervisor","reseller","resellerAgent","customerAdmin"]**

**for i in list:**

**UserRoles.objects.create(name=i )**

**print("roles add successfully")**

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

**for i in range(4):**

**name=fake.name()+str(random.randint(100,1234))**

**password = "admin@123"**

**email = fake.email()**

**choice = ["1","2","3","4"]**

**role = random.choice(choice)**

**cust = Customer.objects.create(name=name)**

**user = SeteraUser.objects.create(**

**first\_name=name,email=email,**

**role\_id=role,**

**customer\_id = cust.id)**

**user.set\_password(password)**

**user.save()**

**print("Data Inserted Successfully")**

**Fake data generate**

**import random**

**from random import shuffle**

**from faker import Faker**

**fake = Faker()**

**fake.random.seed(4321)**

**def insertdummydata(request):**

**for i in range(10):**

**name=fake.name()**

**print("-",name)**

**password = "sajal@123"**

**email = fake.email()**

**choice = ["1","2","3","4"]**

**role = random.choice(choice)**

**cust = Customer.objects.create(name=name)**

**user = SeteraUser.objects.create(**

**first\_name=name,email=email,**

**role\_id=role,**

**customer\_id = cust.id**

**)**

**user.set\_password(password)**

**user.save()**

**return HttpResponse("Done")**

**‘**

**Validate modelviewset(null)**

**def validate\_role(self, value):**

**if not value:**

**raise serializers.ValidationError(**

**{"role": "Please Select the User Role"})**

**return value**

**System celery**

**sudo systemctl start redis-server.service**

**Perform create**

**class QueryApiView(viewsets.ModelViewSet):**

**queryset = QueryifAny.objects.all()**

**serializer\_class = QuerySerializer**

**def perform\_create(self, serializer):**

**serializer.save(user=self.request.user, )**

**My profile**

**class MyProfileAPIView(generics.GenericAPIView):**

**serializer\_class = MyProfileSerializer**

**def get(self, request):**

**serializer = self.serializer\_class(request.user, context={"request": request})**

**return Response({"data": serializer.data})**

**Update otp**

**class MobileOtp(generics.CreateAPIView):**

**serializer\_class = PhoneModelSerilalizer**

**permission\_classes = ()**

**def post(self, request):**

**serializer = self.serializer\_class(data=request.data)**

**if serializer.is\_valid(raise\_exception=True):**

**mo =serializer.validated\_data['mobile']**

**if PhoneModel.objects.filter(mobile=mo).exists():**

**instance = PhoneModel.objects.get(mobile=mo)**

**instance.otp = random.randint(1000,9999)**

**instance.save()**

**content = {"mobile": instance.mobile, "otp": instance.otp}**

**return Response(content, status=status.HTTP\_201\_CREATED)**

**else:**

**instance = serializer.save()**

**content = {"mobile": instance.mobile, "otp": instance.otp}**

***# send\_otp(mobile,otp)***

**return Response(content, status=status.HTTP\_201\_CREATED)**

**return Response(**

**{"message", "invalid number"}, status=status.HTTP\_400\_BAD\_REQUEST**

**)**

**Scan wp**

**root@jarvis1:/home/shubpy*# wpscan --url https://www.shubpy.com / -e ap,at,cb ,dbe -o myscan.txt --random-user-agent***

**Reset Password**

**from django.db import models**

***# Create your models here.***

**from django.dispatch import receiver**

**from django.urls import reverse**

**from django\_rest\_passwordreset.signals import reset\_password\_token\_created**

**from django.core.mail import send\_mail**

**@receiver(reset\_password\_token\_created)**

**def password\_reset\_token\_created(sender, instance, reset\_password\_token, \*args, \*\*kwargs):**

**email\_plaintext\_message = "{}?token={}".format(reverse('password\_reset:reset-password-request'), reset\_password\_token.key)**

**send\_mail(**

***# title:***

**"Password Reset for {title}".format(title="Some website title"),**

***# message:***

**email\_plaintext\_message,**

***# from:***

**"noreply@somehost.local",**

***# to:***

**[reset\_password\_token.user.email]**

**)**

**736674 Validation Modelviewset**

**def validate(self, attrs):**

**qty = attrs.get("quantity")**

**if qty <= 0 :**

**raise serializers.ValidationError("Please Choose a valid number between 0 to 100")**

**elif qty >= 100:**

**raise serializers.ValidationError("You Can print 100 Qr only at a time")**

**return super().validate(attrs)**

**Request.user in seriliazer**

**User = self.context["request"].user**

**Random Password generate**

**import random**

***# Random string of length 5***

**result\_str = ''.join((random.choice('abcdxyzpqr1234567890') for i in range(8)))**

**print(result\_str)**

**Manny to many fields**

**Modrls.py**

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

**return ",".join([str(p) for p in self.videoclass.all()])**

**Admin.py**

**def get\_hashtag(self,obj):**

**return [i.hashtag for i in obj.hashtag.all()]**

**class StudentDetailsAdmin(admin.ModelAdmin):**

**list\_display = ['first\_name',**

**'is\_verify',]**

**list\_editable = ['is\_verify']**

**def get\_queryset(self, request):**

**vuser = StudentDetails.objects.filter(is\_verify= False)**

**return vuser**

**@admin.register(PostHashTags)**

**class PostTagAdmin(admin.ModelAdmin):**

**list\_display = ('hashtag',)**

**@admin.register(Mypost)**

**class MyPostAdmin(admin.ModelAdmin):**

**list\_display = ('name',**

**'discription',**

**'get\_hashtag',**

**)**

**def get\_hashtag(self,obj):**

**return [i.hashtag for i in obj.hashtag.all()]**

**class ProfileAPIView(generics.ListAPIView):**

**serializer\_class = AllProfileSerializer**

**def get\_blocked\_user\_ids(self, data):**

**ids = []**

**for i in data:**

**ids.append(i["created\_by"])**

**ids.append(i["blocked\_user"])**

**return set(ids)**

**BAsic List**

**document = Document()**

**li = [3,5,6,7,8,9]**

**for i in li[3:5]:**

**print("-------------",i)**

**My doc**

**import qrcode**

**import random**

**import os, math**

**from docx.shared import Inches**

**import os**

***# import os***

***# file\_path = "/home/shubpy/CirtificATE/new1111111111.png"***

***# os.startfile(file\_path,'print')***

***# import subprocess, sys***

***# opener = "open" if sys.platform == "darwin" else "xdg-open"***

***# subprocess.call([opener, file\_path])***

**from docx import Document**

**document = Document()**

***# document.save("/home/shubpy/QRCODE/hello.docx")***

**cat = input("Enter the category : ")**

**code = int(input("Enter the range : "))**

**name=(cat[:3]).upper()**

**print(name)**

**res = []**

**for i in range(code):**

**no = random.randint(10000,100000)**

**imgname =f"RR{name}{no}"**

**print(imgname)**

**print(type(imgname))**

**img = qrcode.make(data=imgname)**

**path = img.save(f"/home/shubpy/QRCODE/qr/{imgname}.png")**

**pa = f"/home/shubpy/QRCODE/qr/"**

**for path in os.listdir(pa):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(pa, path)):**

**res.append(path)**

**document.add\_heading('Grid Images', 0)**

***# add paragraph and get run***

**p = document.add\_paragraph()**

**r = p.add\_run()**

***# j = int(input("How many print QR code from database :"))***

**for i in res:**

**print("------------------",i)**

**r.add\_picture(f"/home/shubpy/QRCODE/qr/{i}", width=Inches(1.46), height=Inches(1.45))**

**r.add\_text(" ")**

**document.save("/home/shubpy/QRCODE/doc/hello.docx")**

**2**

**Doc refer**

**from docx.shared import Inches**

***# add image with height and width in Inches***

**doc.add\_picture("images/emma-dau.jpg", width=Inches(5), height=Inches(3))**

**from docx.enum.text import WD\_ALIGN\_PARAGRAPH**

***# add picture***

**doc.add\_picture("images/emma-dau.jpg", width=Inches(5), height=Inches(3))**

***# Get image paragraph and align***

**last\_paragraph = doc.paragraphs[-1]**

**last\_paragraph.alignment = WD\_ALIGN\_PARAGRAPH.CENTER**

***# grid code here***

**from docx import Document**

**from docx.shared import Inches**

**import os**

***# create document***

**document = Document()**

***# add heading***

**document.add\_heading('Grid Images', 0)**

***# add paragraph and get run***

**p = document.add\_paragraph()**

**r = p.add\_run()**

***# iterate over each image in directory***

**for image\_name in os.listdir("images"):**

***# add image so it creates a row with 4 images***

**r.add\_picture(f"images/{image\_name}", width=Inches(1.46), height=Inches(1.45))**

**r.add\_text(" ") *# add space for image seperation***

***# save document***

**document.save('demo.docx')**

**from docx.shared import Inches, Cm**

**import os**

**from docx.enum.text import WD\_ALIGN\_PARAGRAPH**

**doc = Document() *# create doc***

**doc.add\_heading('Images in Table', 0) *# add heading***

***# create table with two rows and columns (Per row images are 3)***

**table = doc.add\_table(rows=0, cols=3, style="Table Grid")**

**image\_dir = "images"**

**images = os.listdir(image\_dir )**

**for i in range(4): *# show 4 rows of image (4x3 = 12 images)***

**image\_row = table.add\_row() *# add row to table for images***

**cap\_row = table.add\_row() *# add row to table for image name***

**for j in range(3): *# Iterate and get 3 images for row***

**image\_name = images.pop()**

***# add image to table***

**set\_cell\_margins(image\_row.cells[j], top=100, start=100, bottom=100, end=50)**

***# add image to cell and align center***

**paragraph = image\_row.cells[j].paragraphs[0]**

**paragraph.add\_run().add\_picture(f"{image\_dir }/{image\_name}", width=Inches(1.8), height=Inches(1.6))**

**paragraph.alignment = WD\_ALIGN\_PARAGRAPH.CENTER**

***# add caption to table***

**cap\_row.cells[j].text = image\_name**

**cap\_row.cells[j].paragraphs[0].alignment = WD\_ALIGN\_PARAGRAPH.CENTER**

***# save to file***

**doc.save("images-table.docx")**

**File size and read Basic**

**from django.test import TestCase**

**import os,math**

***# folder path***

**dir\_path = r'/home/shubpy/Videos/demo/'**

***# list to store files***

**res = []**

***# Iterate directory***

**for path in os.listdir(dir\_path):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(dir\_path, path)):**

**res.append(path)**

**for i in res:**

**file\_stats = os.stat(f"/home/shubpy/Videos/demo/{i}")**

**s =file\_stats.st\_size**

**size\_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")**

**i = int(math.floor(math.log(s, 1024)))**

**p = math.pow(1024, i)**

**s = round(s/ p, 2)**

**print( f"%s %s" % (s, size\_name[i]))**

**from django.test import TestCase**

**import os,math**

***# folder path***

**dir\_path = r'/home/shubpy/Videos/demo/'**

***# list to store files***

**res = []**

***# Iterate directory***

**for path in os.listdir(dir\_path):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(dir\_path, path)):**

**res.append(path)**

**for i in res:**

***# file\_stats = os.stat(f"/home/shubpy/Videos/demo/{i}")***

***# s =file\_stats.st\_size***

**size\_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")**

**i = int(math.floor(math.log(1e+12, 1024)))**

**p = math.pow(1024, i)**

**s = round(1e+12/ p, 2)**

**print( f"%s %s" % (s, size\_name[i]))**

**File size (st\_size=8615175)**

**from django.test import TestCase**

**import os**

***# folder path***

**dir\_path = r'/home/shubpy/Videos/demo/'**

***# list to store files***

**res = []**

***# Iterate directory***

**for path in os.listdir(dir\_path):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(dir\_path, path)):**

**res.append(path)**

**for i in res:**

**file\_stats = os.stat(f"/home/shubpy/Videos/demo/{i}")**

**print(file\_stats)**

**User read files from directoers**

**from django.test import TestCase**

**import os**

***# folder path***

**dir\_path = r'/home/shubpy/Videos/demo/'**

***# list to store files***

**res = []**

***# Iterate directory***

**for path in os.listdir(dir\_path):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(dir\_path, path)):**

**res.append(path)**

**print(res)**

**Read QRcode Image**

**import glob**

**import cv2**

**import pandas as pd**

**import pathlib**

**print("-----------------")**

**def read\_qr\_code(filename):**

**"""Read an image and read the QR code.**

**Args:**

**filename (string): Path to file**

**Returns:**

**qr (string): Value from QR code**

**"""**

**try:**

**img = cv2.imread(filename)**

**detect = cv2.QRCodeDetector()**

**value, points, straight\_qrcode = detect.detectAndDecode(img)**

**return value**

**except:**

**return**

**value = read\_qr\_code('/home/shubpy/projects/SOW/ROBO-SERVER/RoboKidz/RoboKidz/media/images/RRHAN29570.jpg')**

**print(value)**

**QR CODE BASIC**

**from django.test import TestCase**

**import os**

***# folder path***

**dir\_path = r'/home/shubpy/Videos/demo/'**

***# list to store files***

**res = []**

***# Iterate directory***

**for path in os.listdir(dir\_path):**

***# check if current path is a file***

**if os.path.isfile(os.path.join(dir\_path, path)):**

**res.append(path)**

**print(res)**

**Main —---------------------------------**

**import qrcode**

**import random**

**cat = input("Enter the category : ")**

**code = int(input("Enter the range : "))**

**name=(cat[:3]).upper()**

**print(name)**

**for i in range(code):**

**no = random.randint(10000,100000)**

**imgname =f"RR{name}{no}"**

**print(imgname)**

**print(type(imgname))**

**img = qrcode.make(data=imgname)**

**img.save(f"{imgname}.png")**

**from enum import Enum**

**import random**

**cat = input("Enetr the catagory name : ")**

**category =(cat[:3]).upper()**

**code = int(input("Eneter the num :"))**

**title="Robokidz API"**

**print("this is tpe : ",type(title))**

**import qrcode**

**obj\_qr = qrcode.QRCode(**

**version = 1,**

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

**box\_size = 10,**

**border = 4,**

**)**

***# using the add\_data() function***

**for i in range(code):**

**no =random.randint(10000,100000)**

**obj\_qr.add\_data(res)**

***# using the make() function***

**obj\_qr.make(fit = True)**

***# using the make\_image() function***

**qr\_img = obj\_qr.make\_image(fill\_color = "cyan", back\_color = "black")**

***# saving the QR code image***

**qr\_img.save(f"RR{category}{no}.png")**

**Queyset values**

def get\_queryset(self):

h = Post.objects.filter(created\_by=self.request.user)

j = [i.is\_approved for i in h]

print("This is jj values ---",j)

**Swaggers Setup (run migrations is madentory)**

pip install django-rest-swagger

pip install django-rest-swagger

pip install drf-yash

from django.urls import path,include,re\_path

from rest\_framework\_swagger.views import get\_swagger\_view

schema\_view = get\_swagger\_view(title='Rest API Sajal ViEW')

from rest\_framework import permissions

from drf\_yasg.views import get\_schema\_view

from drf\_yasg import openapi

schema\_view = get\_schema\_view(

openapi.Info(

title="Robokidz API",

default\_version="v1",

description="Robokidz api documentation LCM",

),

public=True,

permission\_classes=(permissions.AllowAny,),

)

urlpatterns = [

re\_path(

r"^doc(?P<format>\.json|\.yaml)$",

schema\_view.without\_ui(cache\_timeout=0),

name="schema-json",

), *# <-- Here*

path(

"doc/",

schema\_view.with\_ui("swagger", cache\_timeout=0),

name="schema-swagger-ui",

), *# <-- Here*

path(

"redoc/", schema\_view.with\_ui("redoc", cache\_timeout=0), name="schema-redoc"

),

path('admin/', admin.site.urls),

path('',include('apps.user.urls'))

]

There is one error occur if

Refer code

**Retun 2 serilizrs data**

class SearchAPIView(generics.GenericAPIView):

permission\_classes = (IsAuthenticated,)

serializer\_class = SearchSerializer

def post(self, request, \*args, \*\*kwargs):

serializer = self.serializer\_class(data=request.data)

serializer.is\_valid()

*# search\_val = serializer.validated\_data["search"]*

search\_val = request.data["search"]

posts = Post.objects.filter(

Q(caption\_\_icontains=search\_val,is\_approved=True )

| Q(description\_\_icontains=search\_val, is\_approved=True)

| Q(created\_by\_\_mobile\_\_icontains=search\_val, is\_approved=True)

| Q(created\_by\_\_username\_\_icontains=search\_val,is\_approved=True)

| Q(created\_by\_\_name\_\_icontains=search\_val, is\_approved=True)

| Q(created\_by\_\_id\_\_contains=search\_val,is\_approved=True)

| Q(created\_by\_\_email\_\_icontains=search\_val,is\_approved=True)

| Q(created\_by\_\_first\_name\_\_icontains=search\_val,is\_approved=True)

)

users = User.objects.filter(

Q(user\_type\_\_icontains=search\_val)

| Q(username\_\_icontains=search\_val)

| Q(email\_\_icontains=search\_val)

| Q(school\_\_name=search\_val)

| Q(first\_name\_\_icontains=search\_val)

| Q(name\_\_icontains=search\_val)

*# | Q(mobile=search\_val)*

| Q(state\_\_icontains=search\_val)

| Q(city\_\_icontains=search\_val)

| Q(grade\_\_name=search\_val)

| Q(parent\_email=search\_val)

| Q(address\_\_icontains=search\_val)

| Q(workspace\_\_icontains=search\_val)

| Q(position\_\_icontains=search\_val)

| Q(qualification\_\_icontains=search\_val)

)

posts\_data = [PostResponseSerializer(posts, context={"request": request}, many=True).data, AllProfileSerializer(users, context={"request": request}, many=True).data]

return Response({"posts": posts\_data})

class CrateCirtificates(viewsets.ModelViewSet):

queryset=Certificate.objects.all()

serializer\_class = CreateCertificateSerializer

class GenrateUserCirtificate(viewsets.ModelViewSet):

serializer\_class=GenerateUserCertificateSerilizer

queryset=PostCertificate.objects.all()

from PIL import Image, ImageDraw, ImageFont

import datetime

import uuid

class GenerateUserCertificateSerilizer(serializers.ModelSerializer):

class Meta:

model = PostCertificate

fields = ["post", "user", "follow", "certificate", "created\_by", "updated\_by"]

print(fields)

def create(self, validated\_data):

post = validated\_data["post"].id

user = validated\_data["user"].id

follow = validated\_data["follow"].id

certificate = validated\_data["certificate"].id

certificate = validated\_data["certificate"].certificate

img = Image.open(certificate)

now = datetime.datetime.now()

today = now.strftime("%d-%m-%Y")

*# conditions here*

image\_editable = ImageDraw.Draw(img)

font = ImageFont.truetype("E:/PythonPillow/Fonts/FreeMono.ttf", 48)

image\_editable.text(

(500, 500), validated\_data["user"].first\_name, fill=(0, 0, 0), font=font

)

image\_editable.text( (530, 700), "Outstanding followers ", fill=(0, 0, 0), font=font )

image\_editable.text((950, 800), "Follow", fill=(0, 0, 0), font=font)

font = ImageFont.truetype("E:/PythonPillow/Fonts/FreeMono.ttf", 38)

image\_editable.text((500, 900), today, fill=(0, 0, 0), font=font)

img.show()

file\_name = uuid.uuid4().hex

img.save(f"RoboKidz/media/usercirtificate/Cirtificate{file\_name}.png")

validated\_data["certificate\_image"] = f"/usercirtificate/Cirtificate{file\_name}.png"

RA = PostCertificate.objects.create(\*\*validated\_data)

return RA

**Validate foreign key values**

class GenerateUserCertificateSerilizer(serializers.ModelSerializer):

class Meta:

model = PostCertificate

fields = ["post", "user", "follow", "certificate"]

print(fields)

def create(self, validated\_data):

Image = validated\_data["certificate"].certificate

print("----------------------",Image)

img\_info = PostCertificate.objects.create(\*\*validated\_data)

return img\_info

**Update python3**

[**https://www.debugpoint.com/install-python-3-11-ubuntu/**](https://www.debugpoint.com/install-python-3-11-ubuntu/)

**sudo apt install software-properties-common**

**sudo add-apt-repository ppa:deadsnakes/ppa**

**sudo apt update**

**sudo apt install python3.11**

**sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 2**

**sudo update-alternatives --config python3**

**curl -sS https://bootstrap.pypa.io/get-pip.py | python3.11**

<https://www.debugpoint.com/install-python-3-11-ubuntu/>

class EquipmentType(models.Model):

equipment\_type = models.CharField(verbose\_name="Equipment Type", max\_length=50, unique=True)

def \_\_unicode\_\_(self):

return self.equipment\_type

class EquipmentManufacturer(models.Model):

manufacturer\_name = models.CharField(verbose\_name="Manufacturer Name", max\_length=50, unique=True)

def \_\_unicode\_\_(self):

return self.manufacturer\_name

class EquipmentInfo(models.Model):

equipment\_type = models.ForeignKey(EquipmentType, verbose\_name="Equipment Type")

part\_identifier = models.CharField(verbose\_name="Machine ID (alias)", max\_length=25)

manufacturer\_name = models.ForeignKey(EquipmentManufacturer, verbose\_name="Manufacturer Name")

serial\_number = models.CharField(verbose\_name="Serial Number", max\_length=25)

date\_of\_manufacture = models.DateField(verbose\_name="Date of Manufacture", default=date.today)

is\_active = models.BooleanField(verbose\_name="Is Active", default=True)

def \_\_unicode\_\_(self):

return self.part\_identifier

serializers.py

class EquipmentTypeSerializer(serializers.ModelSerializer):

class Meta:

model = EquipmentType

fields = ('id', 'equipment\_type',)

class EquipmentManufacturerSerializer(serializers.ModelSerializer):

class Meta:

model = EquipmentManufacturer

fields = ('id', 'manufacturer\_name',)

class EquipmentInfoSerializer(serializers.ModelSerializer):

class Meta:

model = EquipmentInfo

fields = ('id', 'equipment\_type', 'part\_identifier', 'manufacturer\_name','serial\_number', 'date\_of\_manufacture', 'is\_active')

equipment\_type = EquipmentTypeSerializer(many=False)

manufacturer\_name = EquipmentManufacturerSerializer(many=False)

def create(self, validated\_data):

equipment\_type = validated\_data.pop('equipment\_type')

manufacturer\_name = validated\_data.pop('manufacturer\_name')

equipment\_info = EquipmentInfo.objects.create(\*\*validated\_data)

return equipment\_info

Assuming I already have relevant EquipmentType and EquipmentManufacturer objects created, I would like to add another EquipmentInfo object. What is the appropriate way to set up my EquipmentInfo serializer so that I can pass in information such as

{

"equipment\_type":{

"equipment\_type":"already\_created",

},

"part\_identifier":"something\_new",

"manufacturer\_name":{

"manufacturer\_name":"already\_created"

},

"serial\_number":"WBA1",

"date\_of\_manufacture": "1900-01-01",

"is\_active":true

}

or even better:

{

"equipment\_type":"already\_created",

"part\_identifier":"something\_new",

"manufacturer\_name":"already\_created",

"serial\_number":"WBA1",

"date\_of\_manufacture": "1900-01-01",

"is\_active":true

}

*# @api\_view(['GET'])*

*# @permission\_classes([IsAuthenticated])*

*# @authentication\_classes([SessionAuthentication, JWTAuthentication])*

*# def getRoutes(request):*

*# routes = [*

*# '/api/token/',*

*# '/api/register/',*

*# '/api/token/refresh/',*

*# '/api/organization/'*

*# ]*

*# return Response(routes)*

Urls

*# path('', views.getRoutes),*

**NOTIFICATION**

class notification(viewsets.ModelViewSet):

serializer\_class = NotificationSerializer

queryset = Notification.objects.all()

http\_method\_names: List[str] = ["get", "patch"]

def get\_serializer(self, \*args, \*\*kwargs):

if self.request.method in ["POST","PATCH"]:

serializer\_class = self.get\_serializer\_class()

else:

serializer\_class = NotificationResponseSerializer

kwargs.setdefault("context", self.get\_serializer\_context())

return serializer\_class(\*args, \*\*kwargs)

def get\_queryset(self):

return Notification.objects.filter(notified\_to=self.request.user.id)

Noti signal

from django.db.models.signals import post\_save, pre\_save

*# from django.contrib.auth.models import User*

from django.dispatch import receiver

from apps.post.models import Like, LikeComment, Post, Comment

from .models import Notification

from apps.user.models import FollowRequest

from django.db.models import signals

@receiver(post\_save, sender=Like)

def post\_save\_like(sender, instance, created, \*\*kwargs):

if instance.created\_by==instance.post.created\_by:

return False

Notification.objects.create(

performed\_by=instance.created\_by,

notified\_to=instance.post.created\_by,

message=instance.created\_by.username + " " +" Liked on your posts.",

message\_key="post like",

)

@receiver(post\_save, sender=LikeComment)

def post\_save\_like\_comment(sender, instance, created, \*\*kwargs):

if instance.created\_by==instance.comment.created\_by:

return False

Notification.objects.create(

performed\_by=instance.created\_by,

notified\_to=instance.comment.created\_by,

message=instance.created\_by.username + " " +" Liked on one your comment. ",

message\_key="comment like",

)

@receiver(post\_save, sender=FollowRequest)

def post\_save\_follow\_request(sender, instance, created, \*\*kwargs):

if instance.status == "pending":

Notification.objects.create(

performed\_by=instance.created\_by,

notified\_to=instance.receiver,

message=instance.created\_by.username + " " +"send your follow request",

message\_key="follow-request-sent",

)

@receiver(post\_save, sender=FollowRequest)

def post\_save\_accept\_follow\_request(sender, instance, created, \*\*kwargs):

if instance.status == "accepted":

Notification.objects.create(

performed\_by=instance.created\_by,

notified\_to=instance.receiver,

message=instance.created\_by.username + " " + "accepted your follow request",

message\_key="follow-request-accepted",

)

@receiver(post\_save, sender=Comment)

def post\_save\_comment(sender, instance, created, \*\*kwargs):

if instance.created\_by==instance.post.created\_by:

return False

Notification.objects.create(

performed\_by=instance.created\_by,

notified\_to=instance.post.created\_by,

message=instance.created\_by.username +" "+ "commented on your post. ",

message\_key="comment send",

)

**MOdelview set Crud**

***from* django.shortcuts *import* render**

***from* rest\_framework *import* viewsets,status**

***from* rest\_framework.response *import* Response**

***from* rest\_framework.authtoken.models *import* Token**

***from* django.contrib.auth.models *import* User**

***from* django.db *import* transaction**

***from* adminapp.serializers *import* UserSerializer**

***from* myapp.authentication.authentication *import* cTokenAuthentication**

***from* rest\_framework.permissions *import* DjangoModelPermissions**

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

**# *Create your views here.***

**class adminLogin(viewsets.ModelViewSet):**

**queryset = User.objects.all()**

**http\_method\_names = ['post']**

**def *create*(self, request, \*args, \*\*kwargs):**

**data = request.data**

***try*:**

***with* transaction.atomic():**

**userData = User.objects.get(**

**username = data["username"],**

**)**

***if* userData.is\_superuser == 0:**

***return* Response({"message":"Only admin can login", "status": False,**

**"response": "fail", }, status=status.HTTP\_400\_BAD\_REQUEST)**

***if* userData.check\_password(data["password"]):**

***if* Token.objects.filter(user=userData).exists():**

**Token.objects.get(user=userData).delete()**

**userData.last\_login = timezone.now()**

**userData.save()**

**token=Token.objects.create(user=userData)**

***return* Response({"username":data['username'], "message": "Login Successfully.Please use token for further process",**

**"token":token.key,**

**"status": True, "response": "success", }, status=status.HTTP\_200\_OK)**

***except* Exception *as* error:**

***return* Response({"message":str(error), "status": False,**

**"response": "fail", }, status=status.HTTP\_400\_BAD\_REQUEST)**

**class UserViewSet(viewsets.ModelViewSet):**

**authentication\_classes = [cTokenAuthentication]**

**permission\_classes = [DjangoModelPermissions]**

**queryset = User.objects.all()**

**serializer\_class=UserSerializer**

**http\_method\_names = ['post','get','put']**

**def *create*(self, request, \*args, \*\*kwargs):**

**data = request.data**

***try*:**

***with* transaction.atomic():**

**userData = User.objects.create(**

**username = data["username"],**

**email = data["email"],**

**first\_name = data["first\_name"],**

**last\_name = data["last\_name"],**

**is\_active = True,**

**)**

**userData.set\_password(data["password"])**

**userData.save()**

***return* Response({"username":data['username'], "message": "Your registration has been successfully completed.",**

**"status": True, "response": "success", }, status=status.HTTP\_200\_OK)**

***except* Exception *as* error:**

***return* Response({"message":str(error), "status": False,**

**"response": "fail", }, status=status.HTTP\_400\_BAD\_REQUEST)**

**def *update*(self, request, \*args, \*\*kwargs):**

**data = request.data**

***try*:**

***with* transaction.atomic():**

**userData = User.objects.filter(id=int(self.kwargs['pk'])).update(**

**email = data["email"],**

**first\_name = data["first\_name"],**

**last\_name = data["last\_name"],**

**)**

**user=User.objects.get(id=int(self.kwargs['pk']))**

**user.set\_password(data["password"])**

**user.save()**

***return* Response({"message": "User updated successfully",**

**"status": True, "response": "success", }, status=status.HTTP\_200\_OK)**

***except* Exception *as* error:**

***return* Response({"message":str(error), "status": False,**

**"response": "fail", }, status=status.HTTP\_400\_BAD\_REQUEST)**

**def list(self, request, \*args, \*\*kwargs):**

***try*:**

***with* transaction.atomic():**

**userData = User.objects.all()**

**data=self.serializer\_class(userData,many=True).data**

***return* Response({"data":data,"message": "list",**

**"status": True, "response": "success", }, status=status.HTTP\_200\_OK)**

***except* Exception *as* error:**

***return* Response({"message":str(error), "status": False,**

**"response": "fail", }, status=status.HTTP\_400\_BAD\_REQUEST)**

**Two serializer view Response**

**class SubscriptionApiView(viewsets.ModelViewSet):**

**queryset = Subscription.objects.all()**

**serializer\_class = SubscriptionSerializer**

**def get\_serializer(self, \*args, \*\*kwargs):**

**if self.request.method in ["POST", "PATCH"]:**

**serializer\_class = self.get\_serializer\_class()**

**else:**

**serializer\_class = SubscriptionResponseSerializer**

**kwargs.setdefault("context", self.get\_serializer\_context())**

**return serializer\_class(\*args, \*\*kwargs)**

**Signals**

***from* django.shortcuts *import* render,HttpResponse**

***from* . models *import* \***

**# *Create your views here.***

***from* rest\_framework *import* generics**

***from* rest\_framework.response *import* Response**

***from* . serializer *import* \***

**def *Insert*(request):**

**kk = Students.objects.filter(FirstName="sam").values()**

**lst = list(kk)**

**ja = Students.objects.filter(FirstName="sam")**

**kmm = [{'id': 2,**

**'FirstName': 'sam',**

**'LastName': 'kumar',**

**'Email': 'sam@gmail.om',**

**'Contact': 12333}]**

**ll = {}**

***for* i *in* kmm:**

***for* key,values *in* i.items():**

**ll["before " +key] = values**

**print(ll)**

**# *Driver code***

**# *jam = list(kk)***

**# *print("=====================",jam)***

***return* render(request,"app\index.html")**

**class Mystudents(generics.ListCreateAPIView):**

**queryset = Students.objects.all()**

**serializer\_class = StudentSerilizer**

**def *create*(self,request,\*args,\*\*kwargs):**

**serializer = StudentSerilizer(data =request.data)**

***if* serializer.is\_valid():**

**kk = serializer.save()**

**# *jam =Students.objects.filter(id=1)***

**# *print("------------------------------",jam)***

**jj = serializer.data**

**print(jj)**

**print("-------------",type(jj))**

***for* key,values *in* jj.items():**

**print(key,values)**

***return* Response(serializer.data )**

***else*:**

***return* Response({'msg':'mmm'})**

**Forget passwsword custom**

**def Change\_password(email,token):**

**try:**

**subject = 'Reset Password'**

**message = f"http://127.0.0.1:8000/confirm/{token}"**

**email\_from = EMAIL\_HOST**

**send\_mail(subject, message, email\_from, [email])**

**print("send ----------------------------")**

**except Exception as e:**

**print("not send ",e)**

**import uuid**

**class ForgetPasswordSerializer(serializers.Serializer):**

**email = serializers.EmailField()**

**def validate(self, attrs):**

**email = attrs.get('email')**

***# print(email)***

**obj = User.objects.filter(email = email).exists()**

**if not obj:**

**raise ValidationError("Email Not Exists")**

**return super().validate(attrs)**

**class VerifyPasswordSerializer(serializers.Serializer):**

**password = serializers.CharField(label=\_("Password"), style={'input\_type': 'password'})**

**confirm\_password = serializers.CharField(label=\_("confirm\_password"), style={'input\_type': 'password'})**

**token = serializers.CharField()**

**def validate\_password(self, value):**

**data = self.get\_initial()**

**password = data.get('confirm\_password')**

**confirm\_password = value**

**if password != confirm\_password:**

**raise ValidationError('Passwords and Confirm password not matching')**

**return value**

**class ForgetApiView(generics.CreateAPIView):**

**serializer\_class = ForgetPasswordSerializer**

**permission\_classes = ()**

**def create(self, request, \*args, \*\*kwargs):**

**token = uuid.uuid1()**

**serializer = self.serializer\_class(data=request.data)**

**if serializer.is\_valid(raise\_exception=True):**

**email = serializer.data['email']**

**print("This is email , ", email)**

**reset\_user = User.objects.filter(email =email).first()**

**if EmailToken.objects.filter(user = reset\_user.id ).exists():**

**instance = EmailToken.objects.get()**

**instance.email\_token = token**

**token = instance.email\_token**

**email = instance.user.email**

**Change\_password(email,token)**

**instance.save()**

**Change\_password(instance.user.email,instance.email\_token )**

**else:**

**obj = EmailToken.objects.create(email\_token = token,user\_id = reset\_user.id)**

**token = obj.email\_token**

**email = obj.user.email**

**Change\_password(email,token)**

**return Response({"message", "Email Send Succefully"} )**

**return Response(**

**{"message", "Not valid Email"}, status=status.HTTP\_400\_BAD\_REQUEST )**

**class VerifyResetPassword(generics.CreateAPIView):**

**queryset = ""**

**serializer\_class = VerifyPasswordSerializer**

**permission\_classes = ()**

**def post(self, request,token, \*args, \*\*kwargs):**

**serializer = VerifyPasswordSerializer(data=request.data)**

**if serializer.is\_valid(raise\_exception=True):**

**token = serializer.data['token']**

**newpassword = serializer.data['password']**

**print()**

**print(token)**

**obj\_pass = EmailToken.objects.filter(email\_token = token).first()**

***# print(obj\_pass)***

**if obj\_pass:**

**updatepassword = User.objects.filter(id = obj\_pass.user.id).first()**

**updatepassword.set\_password(newpassword)**

**updatepassword.save()**

**obj\_pass.delete()**

**return Response({"message", "Password Update Successfully"}, status=status.HTTP\_202\_ACCEPTED)**

**return Response({"message", "Not valid Email"}, status=status.HTTP\_400\_BAD\_REQUEST )**

**class MobileChangePassword(generics.CreateAPIView):**

**queryset = ""**

**serializer\_class = MobileChangePasswordSerializer**

**permission\_classes = ()**

**def post(self, request, \*args, \*\*kwargs):**

**serializer = MobileChangePasswordSerializer(data=request.data)**

**if serializer.is\_valid(raise\_exception=True):**

**otp = serializer.data['otp']**

**newpassword = serializer.data['password']**

**obj\_pass = PhoneModel.objects.filter(otp = otp).first()**

**if obj\_pass:**

**updatepassword = User.objects.filter(mobile = obj\_pass.mobile).first()**

**updatepassword.set\_password(newpassword)**

**updatepassword.save()**

**obj\_pass.delete()**

**p = obj\_pass.mobile**

**return Response({"message", "Password Update Successfully"}, status=status.HTTP\_202\_ACCEPTED)**

**return Response({"message", "Not valid Email"}, status=status.HTTP\_400\_BAD\_REQUEST )**

**CORS HEADER**

**CORS\_ORIGIN\_ALLOW\_ALL = True**

**CORS\_ALLOWED\_ORIGINS = [**

**"http://localhost:3000",**

**"http://localhost:8000",**

**"http://127.0.0.1:3000",**

**"http://127.0.0.1:8000",**

**]**

**CORS\_ORIGIN\_WHITELIST = (**

**"http://localhost:8000",**

**"http://127.0.0.1:8000",**

**"http://localhost:3000",**

**"http://127.0.0.1:3000",**

**)**

**CORS\_ALLOW\_HEADERS = [**

**"access-control-allow-origin",**

**"content-type",**

**"authorization",**

**]**

**CORS\_ALLOW\_HEADERS = [**

**'accept',**

**'accept-encoding',**

**'authorization',**

**'content-type',**

**'dnt',**

**'origin',**

**'user-agent',**

**'x-csrftoken',**

**'x-requested-with',**

**'Access-Control-Allow-Origin',**

**]**