هنگامیکه دیتایی را میخواهید به کلاس مربوطه serializer بدهید باید آن دیتا را به متغیر data بدهید

my\_ser = {"phone\_number":"099085sdsd37607" , "first\_name":"ahamd" , "last\_name":"moshtaghi"}

my\_data = ProfileSerializers(data = my\_ser)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه از serializers.Serializer در استفاده میکنید حتما باید تابع create ساخته شود به صورت زیر :

    def create(self , validated\_data):

        return Profiles.objects.create(\*\*validated\_data)

\*\*validated\_data -----> به صورت دیکشنری

هنگام update هم حتما باید تابع update را implement کنیم

Regexfield حتما مطالعه شود

اولین آرگومانی که متود init هر کلاسی در سریالازر دارد instance میباشد به این صورت که برای بروزرسانی دیتاهایی که در دیتابیس دخیره شده هست میتوانیم به instance بدهیم مثلا:

My\_profiel = Profiles.objects.get(id=3)

My\_ser = ProfilesSerializers(instance = my\_profile , data = my\_data)

هنگامیکه از متد is\_valid() استفاده میکنیم ممکن است خروجی false دریافت کنیم . برای اینکه متوجه شویم کدام فیلد مشکل دارد باید از متد errors() استفاده شود که کاملا توضیح میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

posts = PostsSerializers( many = True )

serializers = ProfileSerializers(profiles , many = True)

many=True به این معنی است که serializer میتواند چندین objects را دریافت و نمایش دهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه از متد post استفاده میکنیم در veiws لازم است request را به json تبدیل کنیم

برای این کار کافیست از دستورات زیر استفاده کنیم

from rest\_framework.parsers import JSONParser

data = JSONParser().parse(request)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در serializer اگر بخواهیم متوجه شویم مقدار فیلدی درست وارد شده است کافیست از روش زیر استفاده کنیم مثلا برای (phone\_number) :

import re

from rest\_framework.exceptions import ValidationError

def validate\_phone\_number(self , value):

phone\_number\_pattern = re.compile("09\d{9}$")

if phone\_number\_pattern.match(value):

return value

raise ValidationError("{phone\_number is not correct}")

اگر بخواهیم برای فیلد دیگری این کار را انجام دهیم باید از قانون زیر استفاده کنیم:

def validate\_”namefield”(self , value)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه بخواهیم دو مدل را از طریق کلید خارجی به هم متصل کنیم لازم است و همچنین از serializer هم استفاده کنیم حتما باید از related\_name استفاده کنیم

class Posts(models.Model):

author = models.ForeignKey(Profiles , related\_name='posts' , on\_delete=models.CASCADE)

اکنون میتوانیم در serializer که مربوط به profiles است postهای مربوطه را اینچنین نمایش دهیم

posts = PostsSerializers( many = True , read\_only=True)

“posts----------->related\_name=”post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه بخواهیم از طریق کلید خارجی و متد POST مدل دیگری بسازیم (profiles , posts) بدیل اینکه profiles همان لحطه ساخته میشود لازم است برای post متد create را دوباره نویسی کنیم که میتوانیم از قانون زیر پیروی کنیم :

post = PostsSerializers(write\_only=True)

class ProfileSerializers(serializers.ModelSerializer):

    def create(self , validate\_data):

        post = validate\_data.pop("post")

        my\_author = Profiles.objects.create(\*\*validate\_data)

        my\_post = Posts.objects.create(\*\*post , author=my\_author)

        return my\_author

اکنون لازم است فیلد author که فیلد مربوط به مدل post میباشد و با استفاده از کلید خارجی به مدل profiles وصل شده است ویژگی دیگری به آن اضافه کنیم این ویژگی required میباشد که این گونه اضافه میشود :

class PostsSerializers(serializers.ModelSerializer):

    class Meta:

        model = Posts

        fields = '\_\_all\_\_'

        extra\_kwargs = {'author':{'required':False}}

در واقع این ویژگی میگوید لازم نیست این فیلد را داخل ورودی inputها داشته باشیم

گاهی اوقات لازم نیست مدلی که از طریق کلید خارجی به مدل اصلی ما متصل شده است تمام فیلدهایش نمایش داده شود و فقط لازم است مثلا بگوید کدام postها مربوط به کدام profile میباشد برای همین کار از primarykeyrelatedfield به صورت زیر استفاده کنید :

class ProfileSerializers(serializers.ModelSerializer):

    posts = serializers.PrimaryKeyRelatedField( many=True , read\_only=True)

دقت شود posts که در بالا تعریف شده است حتما باید هم نام با related\_name ی باشد که در مدل post تعریف شده است

class Posts(models.Model):

    author = models.ForeignKey(Profiles , related\_name='posts' , on\_delete=models.CASCADE)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در restfull ما اگر بخواهیم دیتایی رو بگیریم باید حتما از متد GET استفاده کنیم . برای ساخت کلاس حتما باید از متد POST استفاده کنیم در ادامه به توضیح بقیه ی متد ها میپردازیم

گرفتن دیتا ----< GET read -🡪GET

ساخت کلاس -----< POST create-🡪post

آپدیت کردن کلی یک object -----< PUT update--🡪PUT

آپدیت کردن بخشی از یک object -----< PATCH partial-update--🡪 PATCH

و برای حذف یک object ----< DELETE remove --🡪 DELETE

در نظر داشته باشید که میتوانید برای آپدیت از POST استفاده کرد اما اگر این کار را انجام دهید دیگر rest شما دیگر restfull نیست و دیگر نمیتوان به آن restfull گفت زیرا از قوانین restfull خارج شده اید

Restfull یک سری قوانین هست که با رعایت آنها کار راحت تر و استاندارد تر میشود

تفاوت HTTP request و request این هست که HTTP request ها برای خود جنگو هستند یعنی request هایی که از طریق سایت دریافت میشود و request معمولی در API ها وجود دارند . در Django rest framework قابلیتی وجود دارد تا بتوانیم بین این دو درخواست تفاوت قائل شویم , چگونه : با استفاده از decorator

در زیر نحوه استفاده از آن را مشاهده میکنید :

from rest\_framework.decorators import api\_view

@api\_view

def profile\_list(request):

…

روش دیگری هم برای این کار وجود دارد که با استفاده از کلاس های آماده rest میتوان این کار را انجام داد و خود کلاس تمام این موارد را بررسی میکند

from rest\_framework.views import APIView

from rest\_framework.response import Response

class ProfilesView(APIView):

    def get (self , request):

        profiles = Profiles.objects.all()

        serializers = ProfileSerializers(profiles , many = True)   #many = True ----> several objects

        return Response(serializers.data)

در اینجا متد برای متد get نوشته شده است

Generic api view ها کلاس هایی هستند که میتوانیم از آنها ارث بری کنیم و کارمان را آسان تر انجام دهیم به مثال زیر توجه کنید:

from rest\_framework import generics

class Profilesview (generics.ListAPIView):

    serializer\_class = ProfileSerializers

    queryset = Profiles.objects.all()

در این مثال کلاس Profilesview از کلاس ListAPIView ارث بری کرده است وظیفه کلاس ListAPIView این است که لیستی از آبجکت ها را برگرداند در زیر خود کلاس ListAPIView را مشاهده میکنید

class ListAPIView(mixins.ListModelMixin,

                  GenericAPIView):

    """

    Concrete view for listing a queryset.

    """

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

        return self.list(request, \*args, \*\*kwargs)

همان طور که انتظار میرود این کلاس فقط متد get را دارد چون قرار است فقط لیستی از آبجکت ها را برگرداند

تنها کاری که لازم است در کلاس Profilesview انجام شود این است که دو مقدار serializer\_class و queryset مقداردهی شود

انواع مختلف دیگری از کلاس های generic وجود دارد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_