DJANGO Rest Framework

Make a folder in your computer and run below commands

py -m pip install --user virtualenv
py -m venv venv
.\venv\Scripts\activate

pip freeze > requirements.txt (it will create requirements.txt in your folder)

simple apiview using function based views :-

```
from rest_framework.request import Request from restframework.response
import Response from rest_framework.decorators import APIView, api_view,
permission_classes


 @api_view(http_method_names=["GET", "POST"])
 @permission_classes([AllowAny])
 def homepage(request: Request):

    if request.method == "POST":
         data = request.data

         response = {"message": "Hello World", "data": data}

         return Response(data=response,
status=status.HTTP_201_CREATED)

    response = {"message": "Hello World"}
    return Response(data=response, status=status.HTTP_200_OK)


 post=[
 {
 'id':1,
 'title':'django',
 'duration':'2 hours'},
 {
 'id':2,
 'title':'python',
 'duration':'1 hour'}
 }
```

```
]


@api_view(http_method_names=['GET']) def
post_detail(request:Request):
return Response(data=posts,status=status.HTTP_200_OK)

@api_view(http_method_names=['GET']) def
post_details(request:Request,post_index:int):
Post=posts[post_index]    if post:
       return Response(data=post,status=status.HTTP__OK)
else:
        return Response( data={'error':'post is not available
'},status=status.HTTP_404_Not_Found)


@api_view(http_method_names=['GET']) def
post_list(request:Request):
     return Response(data=posts,status= status.HTTP__OK)


Urls.py :-

Path('homepage/',views.homepage,name='homepage'),
Path('posts/',views.post_list,name='post_list'),
Path('<int:post_index>',views.post_details,name='post_details'),

Serializers:-

Serializers.py:
 from restframework import serializers
from post.models import PostModel

Class PostSerializers(serializers.Modelserializer):
class Meta:
```

```
            model=PostModel
fields=”__all” models.py :-  from
django.contrib import models


Class PostModel(models.Model):
      id=models.IntegerField(unique=True)
name=models.CharField(max_length=60)       description=
models.CharField(max_length=600)
       def __str__(self):
return self.name


Views.py :-  from restframework,request import
Request from restframework.response import
Response from restframework.decoratores import
api_view from post.models import PostModel from
post.serializers import PostSerializers




@api_view(http_method_names=[‘GET’,’POST’]) def
listposts (request:Request):     if
request.method == ‘POST’:
data=request.data
serializers=PostSerializers(data=data)
if serializers.is_valid():
serializers.save()               response={
                  ‘message’:’post list fetched’,
‘data’: serializers.data }               return
Response(data=response
,status=status.HTTP_201_CREATED)
else:
              response={
                    ‘error’: serializers.errors,
}               return Response(data=response
,status=status.HTTP_400_BAD_REQUEST)

else:
```

```
      post=PostModel.objects.all()
serializers=PostSerializer(instance=post,many=True)       response={
             'message':'post list fetched',
'data': serializers.data   }        return
Response(data=response
,status=status.HTTP_200_OK)
```

urls.py :-

```
Path('list/',views.listposts,name='list')
```

django fixtures

to laod to data base from fixtures ---->> python manage.py loaddata albums (in the same app)

to dump to json file from db --> python manage.py dumpdata albums > data.json

python manage.py loaddata fixtures\question.json (in diffrent app)

python manage.py dumpdata appname.ModelName > specific_file.json (specific table data)

using custome serializers :-

```
class ObjectTracking(models.Model):
    created_at = models.DateTimeField(auto_now_add=True,null=True)
    updated_at = models.DateTimeField(auto_now=True,null=True)

    class Meta:
        abstract = True
        ordering = ('-created_at',)
```

```
# Create your models here.
class Question(ObjectTracking):
    title = models.TextField(null=False, blank=False)
    status = models.CharField(default='inactive', max_length=10)
    created_by = models.ForeignKey(User, null=True, blank=True,
on_delete=models.CASCADE)

    start_date = models.DateTimeField(null=True, blank=True)
    end_date = models.DateTimeField(null=True, blank=True)
    #tags = models.ManyToManyField(Tag)
```

```
    # comments = GenericRelation(Comment, related_query_name="question")

    # objects = QuestionManager()


    def __str__(self):
        return self.title

    @property
    def choices(self):
        return self.choice_set.all()
```

```
class Choice(models.Model):
    question = models.ForeignKey(Question,
on_delete=models.CASCADE,related_name='choices',null=True)
    text = models.TextField(null=True, blank=True)

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
```

```
        return self.text

    @property
    def votes(self):
        return self.answer_set.count()
```

```
from .models import Question

class ChoiceSerializers(serializers.ModelSerializer):
    id=serializers.IntegerField(required=False)
    class Meta:
        model=Choice
        fields="__all__"
        read_only_fields=('question',)

class QuestionSerializers(serializers.ModelSerializer):
    choices = ChoiceSerializers( many=True)
```

```
    #question=serializers.SerializerMethodField('question')
    class Meta:
        model=Question
        fields=['id','title','status','created_by','start_date','end_date','choic
es']

    def create(self, validated_data):
        choices = validated_data.pop('choices')
        #tags = validated_data.pop('tags')
        question = Question.objects.create(**validated_data)
        for choice in choices:
```

```
            Choice.objects.create(**choice, question=question)
        #question.tags.set(tags)
        return question

    def update(self, instance, validated_data):
        choices = validated_data.pop('choices')
        instance.title = validated_data.get("title", instance.title)
        instance.save()
        keep_choices = []
        #existing_ids=[c.id for c in instance.choices]
        for choice in choices:
```

```
            if "id" in choice.keys():
                if Choice.objects.filter(id=choice["id"]).exists():
                    c = Choice.objects.get(id=choice["id"])
                    c.text = choice.get('text', c.text)
                    c.save()
                    keep_choices.append(c.id)
                else:
                    continue
            else:
                c = Choice.objects.create(**choice, question=instance)
                keep_choices.append(c.id)
```

```
        for choice in instance.choices.all():
            if choice.id not in keep_choices:
                choice.delete()
        return instance



class QuestionListSerializer(serializers.ModelSerializer):
      class Meta:
          model=Question
```

```
          fields="__all__"



class QuestionViewsets(viewsets.ModelViewSet):
    serializer_class=QuestionSerializers

    def get_queryset(self):
        queryset=Question.objects.all()
        return queryset
```

```
    # def get_serializer_class(self):
    #     if self.action == 'list':

    #         return QuestionListSerializer
    #     else:
    #          return QuestionSerializers


class ChoiceViewsets(viewsets.ModelViewSet):
    serializer_class=ChoiceSerializers
```

```
    def get_queryset(self):
        queryset=Choice.objects.all()
        return queryset
```

Update using django restframework

```
class AlbumSerializer(serializers.ModelSerializer):

    #artist=serializers.CharField( source='artist.first_name'  ,read_only=True)

    class Meta:
        model = Album
        fields = ('id', 'artist', 'name', 'release_date', 'num_stars',)
```

```
class MusicianSerializer(serializers.ModelSerializer):
    album_musician = AlbumSerializer(read_only=True, many=True)




    class Meta:
        model = Musician
        fields = ('id', 'first_name', 'last_name', 'instrument','album_musician')
```

```
    def update(self,instance,**validated_data):
        instance=self.first_name(instance.first_name,validated_data)
        instance=self.last_name(instance.last_name,validated_data)
        instance=self.instrument(instance.instrument,validated_data)
        instance.save()
```

```
class AlbumView(generics.RetrieveUpdateDestroyAPIView):
   serializer_class = AlbumSerializer
```

```
    queryset = Album.objects.all()

from rest_framework.response import Response

class UpdateMusicianView(generics.GenericAPIView):
    serializer_class=MusicianSerializer
    def put(self,pk,request):
        musican=Musician.objects.filter(id=pk)
        if musican:
            serializer = self.serializer_class(musican, data=request.data,
partial=True)
```

```
            serializer.is_valid(raise_exception=True)
            serializer.save()
            return Response(serializer.data,many=True)
```