Django REST\_API

Django REST\_API by Pretty Printed;

**Tutorial 1:Getting started with REST API(SERIALIZER):**  
1. Serializer:  
 To and from JSON. In the form of list and dictionary.

2. Routers:  
 Routers will generate URLS for both list and detail view.  
===========================================================  
App: Languages  
 language/**Models.py**from django .db import models  
  
class Language(models.Model):  
 name=models.CharField(max\_length=50)  
 paradigms=models.CharField(max\_length=50)

Language/**serializer.py**  
from rest\_framework import serializers  
from . models import language  
  
class LanguageSerializer(serializers.ModelSerializer):  
 class Meta:  
 model=Language  
 fields=[‘id’,’name’,’paradigm’]

Language/**views.py**  
from django.shortcuts import render  
from rest\_framework import viewsets  
from . models import language  
from .serializers import LanguageSerializer  
  
class LanguageView(viewsets.ModelViewSet): //HyperLinkedModelViewSet : for seeing the actual url for . //returned resource  
 queryset=Language.objects.all()  
 serializer\_class=LanguageSerializer

Language/**urls.py**from django.urls import path, include  
from . import views  
from rest\_framework import routers  
router=routers.DefaultRouter()  
router.register(‘languages’, views.LanguageView)

Urlpatterns=[  
 path(‘’,include(router.urls)),  
]

**B.Model Relationships(Foreign key relationship; Many to Many relationship):**  
 Language/**models.py**class Paradigm(models.Model):  
 name=models.CharField(max\_length=50)  
   
 def \_\_str\_\_(self):  
 return self.name  
class Programmer(models.Model):  
 name=models.CharField(max\_length=50)  
 paradigm=models.ForeignKey(Paradigm, on\_delete=models.CASCADE)  
 def \_\_str\_\_(self):  
 return self.name  
  
class Programmer(models.Model):  
 name=models.CharField(max\_length=50)  
 languages=models.ManyToManyField(Language)

Def \_\_str\_\_(self):  
 return self.name

Language/**serializers.py  
we already have serializer class for model Language; now we are going to build a serializer class for other models**class ParadigmSerializer(serializers.HyperLinkedModelSerializer):  
 class Meta:  
 model=Paradigm  
 fields=(‘id’,’url’,’name’)

Class ProgrammerSerializer(serializers.HyperLinkedModelSerializer):  
 class Meta:  
 model=Programmer  
 fields=(‘id’,’url’,’name’,’languages’)  
  
 Language/**views.py**//adding others models viewsets  
class ParadigmView(viewsets.ModelViewSets):  
 queryset=Paradigm.objects.all()  
 serializer\_class=ParadigmSerializer  
  
class ProgrammerView(viewsets.ModelViewSet):  
 queryset=Programmer.objects.all()  
 serializer\_class=ProgrammerSerializer

Language/**urls.py**

router= routers.DefaultRouter()  
router.register(‘languages’,views.LanguageView)  
router.register(‘paradigms’,views.ParadigmView)  
router.register(‘programmers’,views.ProgrammerView)  
  
urlpatterns=[  
 path(‘’,include(router.urls))  
]  
  
**Running this app in the browser; to add the information we start with the model Paradigm because it has no dependency with others;**

**c. How to use Django REST Permission: what the user is allowed to do;**1. **Login:**

**Main/URLs.py  
//to create the login panel in django rest home page**path(‘api-auth/’, include(‘rest\_framework.urls’))

2.**not to allow users to add the object in model; only viewing is allowed** **Language/views.py**from rest\_framework import permissions  
class LanguageView(viewsets.ModelViewSet):  
 queryset=Language.objects.all()  
 serializer\_class=LanguageSerializer  
 permission\_classes=(permissions.IsAuthenticatedorReadOnly,) //it is a tuple **this will bring the add option for the database objects; if we delete this (or comment ) this then the form will disappear  
=>we can do this in individual views   
 OR  
=>we can do this in settings.py for the entire app; all the models will have this activated  
 settings.py**REST\_FRAMEWORK={

‘DEFAULT\_PERMISSION\_CLASSES’=(‘rest\_framework.permissions.IsAuthenticatedOrReadOnly’,)///it is a string

} **D. JSON web tokens with Django REST Framework**It is used to verify some information without using database.  
Generally for api authentication you save the token in database and when that token changes you update in the database; but with JSON database you don’t have to use database for everything because the way to encode and decode the token is public.[**www.jwt.io**](http://www.jwt.io) **To do this we need to install a package called jwt  
=> pip install djangorestframework\_simplejwt  
=> it is pair because you get two tokens; one is access token and other is refresh token; when acces token is expired then you user refresh token to get new access token**

**STEP1:**  **settings.py** REST\_FRAMEWORK={

‘DEFAULT\_PERMISSION\_CLASSES’:(‘rest\_framework.permissions.IsAuthenticatedOrReadOnly’,)/

‘DEFAULT\_AUTHENTICATION\_CLASSES’:(‘rest\_framework\_simplejwt.authentication.JWTAuthentication’,)

}  
**STEP2: URLS.PY**//We will get two urls one for get the token and other to refresh the token  
from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView  
  
urlpatterns=[

Path(‘api/token/’,TokenObtainPairView.as\_view()),  
path(‘api/token/refresh/’,TokenRefreshView.as\_view())  
]