**>>> from rest\_framework.renderers import JSONRenderer**

**>>> from myApp.models import Employee**

**>>> from myApp.serializers import EmployeeSerializer**

**>>> qs=Employee.objects.all()**

**>>> type(qs)**

**<class 'django.db.models.query.QuerySet'>**

**>>> eserializer=EmployeeSerializer(qs,many=True)**

**>>> type(eserializer)**

**<class 'rest\_framework.serializers.ListSerializer'>**

**>>> eserializer**

**EmployeeSerializer(<QuerySet [<Employee: Employee object (1)>, <Employee: Employee object (2)>, <Employee: Employee object (3)>, <Employee: Employee object (4)>]>, many=True):**

**eno = IntegerField()**

**ename = CharField(max\_length=60)**

**esal = FloatField()**

**eaddr = CharField(max\_length=100)**

**>>> eserializer.data**

**[OrderedDict([('eno', 101), ('ename', 'shreekanth'), ('esal', 70000.0), ('eaddr', 'bangalore')]), OrderedDict([('eno', 102), ('ename', 'shreenath'), ('esal', 10000.0), ('eaddr', 'Mumbai')]), OrderedDict([('eno', 103), ('ename', 'shreya'), ('esal', 30000.0), ('eaddr', 'Delhi')]), OrderedDict([('eno', 104), ('ename', 'Michael john'), ('esal', 50000.0), ('eaddr', 'Delhi')])]**

**>>> from rest\_framework.renderers import JSONRenderer**

**>>> json\_data=JSONRenderer().render(eserializer.data)**

**>>> json\_data**

**b'[{"eno":101,"ename":"shreekanth","esal":70000.0,"eaddr":"bangalore"},{"eno":102,"ename":"shreenath","esal":10000.0,"eaddr":"Mumbai"},{"eno":103,"ename":"shreya","esal":30000.0,"eaddr":"Delhi"},{"eno":104,"ename":"Michael john","esal":50000.0,"eaddr":"Delhi"}]'**

**Qs or Model Instance🡺dict[Serialization] dict🡺json data**

**DESERIALIZATION**

**Json data🡺dict🡺Db supported types[QS or Model instance]**

**The process of converting python native type(dict) into Db supported types is called deserialization**

**Ob🡺If this object contains a single record then the data type is model instance**

**Ob🡺If this object contains multiple records then the data type is query set**

**>>> emp=Employee.objects.get(id=1)**

**>>> type(emp)**

**<class 'myApp.models.Employee'>**

**>>> emp=Employee.objects.all()**

**>>> type(emp)**

**<class 'django.db.models.query.QuerySet'>**

**Scenario**

**Json data🡪dict 🡺JSONParser().parse(stream)**

**Stream is a byte string [b’……..’]**

**>>> import io**

**>>> from rest\_framework.parsers import JSONParser**

**>>>**

**>>> pdata=JSONParser().parse(stream)**

**>>> pdata**

**[{'eno': 101, 'ename': 'shreekanth', 'esal': 70000.0, 'eaddr': 'bangalore'}, {'eno': 102, 'ename': 'shreenath', 'esal': 10000.0, 'eaddr': 'Mumbai'}, {'eno': 103, 'ename': 'shreya', 'esal': 30000.0, 'eaddr': 'Delhi'}, {'eno': 104, 'ename': 'Michael john', 'esal': 50000.0, 'eaddr': 'Delhi'}]**

**>>> stream**

**<\_io.BytesIO object at 0x0000018F9677CE00>**

**Json\_data must be converted into bytes form for processing**

**parse() is the method of JSONParser class,it converts bytes stream of json data into python dict.**

**Post processing step after serialization**

**json\_data=JSONRenderer().render(eserializer.data)**

**🡺converts dict into json data**

**QS or model instance🡺dict[serialization]**

**Dict🡺json string**

**Pre processing step before deserialization**

**pdata=JSONRenderer().render(eserializer.data)**

**🡪converts json stream into py dict**

**Json string🡺dict🡪DB supported types**

**Dict🡺DB supported types(deserialization)**

**render()🡪dict to json data [post processing after serialization]**

**parse()🡪json datastream to dict [pre processing before deserialization]**

**USE CASE**

**1.PARTNER APPLICATION NEEDS ALL RECORDS IN JSON FORM**

**STEP 1:Employee Records[Query set]**

**STEP 2:Qs is converted into dict[serializat]ion]**

**STEP 3:dict🡪json data by using render(){post processing}**

**o/p🡺json data**

**2.PARTNER APPLICATION IS SENDING JSON DATA TO CREATE NEW EMPLOYEE ENTRY**

**STEP 1:collect json data**

**STEP 2:Json data is converted into dict by using parse(){pre processing}**

**STEP 3:dict is converted into DB supported types{Deserialization}**

**o/p:Query set or model instance**

**UPDATION**

**For updation we must override a method update for this,we need 2 entities**

**1.Existing instance**

**2.Provided data**

**We should update the instance with validated data provided by partner application**

**instance.eno=validated\_data.get(‘eno’,instance.eno)**

**eno🡪provided by partner application[test.py]**

**if test.py provides any eno,consider the corresponding value and assign this to existing instance.else[if no eno is provided then consider the current eno]**

**models.py**

**from django.db import models**

**class Employee(models.Model):**

**eno=models.IntegerField()**

**ename=models.CharField(max\_length=6)**

**esal=models.FloatField()**

**eaddr=models.CharField(max\_length=100)**

**admin.py**

**from django.contrib import admin**

**from myApp.models import Employee**

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

**l=['id','eno','ename','esal','eaddr']**

**admin.site.register(Employee,EmployeeAdmin)**

**serializers.py**

**from rest\_framework import serializers**

**from myApp.models import Employee**

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

**def multiples\_of\_1000(value):**

**print("validation by validator attribute")**

**if value%1000!=0:**

**raise Exception("Emp sal must be multiple of 1000")**

**def create(self,validated\_data):**

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

**#validated data containes kye:value pairs**

**def update(self,instance,validated\_data):**

**instance.eno=validated\_data.get('eno',instance.eno)**

**instance.ename=validated\_data.get('ename',instance.ename)**

**instance.esal=validated\_data.get('esal',instance.esal)**

**instance.eaddr=validated\_data.get('eaddr',instance.eaddr)**

**instance.save()**

**return instance**

**eno=serializers.IntegerField()**

**ename=serializers.CharField(max\_length=6)**

**esal=serializers.FloatField(validators=[multiples\_of\_1000])**

**eaddr=serializers.CharField(max\_length=100)**

**test.py**

**import requests**

**import json**

**BASE\_URL='http://127.0.0.1:8000/'**

**ENDPOINT='api/'**

**def get\_resource(id=None):**

**data={}**

**if id is not None:**

**data={'id':id}**

**response=requests.get(BASE\_URL+ENDPOINT,data=json.dumps(data))**

**print(response.status\_code)**

**print(response.json())**

**def create\_resource():**

**new\_emp={'eno':999,'ename':'abhi','esal':200,'eaddr':'Mumbai'}**

**response=requests.post(BASE\_URL+ENDPOINT,data=json.dumps(new\_emp))**

**print(response.status\_code)**

**print(response.json())**

**def update\_resource(id):**

**new\_data={'id':id,'esal':100000,'eaddr':'Mysore'}**

**response=requests.put(BASE\_URL+ENDPOINT,data=json.dumps(new\_data))**

**print(response.status\_code)**

**print(response.json())**

**def delete\_resource(id):**

**data={'id':id}**

**response=requests.delete(BASE\_URL+ENDPOINT,data=json.dumps(data))**

**print(response.status\_code)**

**print(response.json())**

**print("Enter 1--->get a resource\n 2--->create a resource 3-->update resource 4--->delete resource")**

**ch=int(input("Enter your option"))**

**if ch==1:**

**get\_resource()**

**if ch==2:**

**create\_resource()**

**if ch==3:**

**id=int(input("Enter id"))**

**update\_resource(id)**

**if ch==4:**

**id=int(input("Enter id"))**

**delete\_resource(id)**

**views.py**

**from django.shortcuts import render**

**from django.views.generic import View**

**from rest\_framework.parsers import JSONParser**

**import io**

**from myApp.models import Employee**

**from myApp.serializers import EmployeeSerializer**

**from rest\_framework.renderers import JSONRenderer**

**from django.http import HttpResponse**

**from django.views.decorators.csrf import csrf\_exempt**

**from django.utils.decorators import method\_decorator**

**@method\_decorator(csrf\_exempt,name='dispatch')**

**class EmployeeCRUDCBV(View):**

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

**json\_data=request.body**

**stream=io.BytesIO(json\_data)**

**pdata=JSONParser().parse(stream)**

**id=pdata.get('id',None)**

**if id is not None:**

**emp=Employee.objects.get(id=id)**

**serializer=EmployeeSerializer(emp)**

**json\_data=JSONRenderer().render(serializer.data)**

**return HttpResponse(json\_data,content\_type='application/json')**

**qs=Employee.objects.all()**

**serializer=EmployeeSerializer(qs,many=True)**

**json\_data=JSONRenderer().render(serializer.data)**

**return HttpResponse(json\_data,content\_type='application/json')**

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

**json\_data=request.body**

**stream=io.BytesIO(json\_data)**

**pdata=JSONParser().parse(stream)**

**serializer=EmployeeSerializer(data=pdata)**

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

**serializer.save() #internally calls create method\_decorator**

**msg={'msg':'resource created successfully'}**

**json\_data=JSONRenderer().render(msg)**

**return HttpResponse(json\_data,content\_type='application/json')**

**else:**

**msg={'msg':'error in creation'}**

**json\_data=JSONRenderer().render(msg)**

**return HttpResponse(json\_data,content\_type='application/json')**

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

**json\_data=request.body**

**stream=io.BytesIO(json\_data)**

**pdata=JSONParser().parse(stream)**

**id=pdata.get('id')**

**emp=Employee.objects.get(id=id)**

**serializer=EmployeeSerializer(emp,data=pdata,partial=True)**

**#update emp with data**

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

**serializer.save() #internally calls update method**

**msg={'msg':'resource updated successfully'}**

**json\_data=JSONRenderer().render(msg)**

**return HttpResponse(json\_data,content\_type='application/json')**

**else:**

**msg={'msg':'error in updation'}**

**json\_data=JSONRenderer().render(msg)**

**return HttpResponse(json\_data,content\_type='application/json')**

**#by default put() accepts complete data not partial data**

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

**json\_data=request.body**

**stream=io.BytesIO(json\_data)**

**pdata=JSONParser().parse(stream)**

**id=pdata.get('id')**

**emp=Employee.objects.get(id=id)**

**emp.delete()**

**msg={'msg':'Resource deleted successfully'}**

**json\_data=JSONRenderer().render(msg)**

**return HttpResponse(json\_data,content\_type='application/json')**

**settings.py**

**INSTALLED\_APPS = [**

**….**

**'myApp',**

**'rest\_framework'**

**]**

**Validations by using serializers**

**1.Field level validation**

**2.Object level validation**

**3.By using validators**

**Field level validation**

**validate\_field\_name()**

**serializers.py**

**from rest\_framework import serializers**

**from myApp.models import Employee**

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

**def validate\_esal(self,value): #is called by serializers.is\_valid()**

**if value<5000:**

**raise serializers.ValidationError(“Sal cannot be <5000”)**

**def create(self,validated\_data):**

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

**#validated data containes kye:value pairs**

**def update(self,instance,validated\_data):**

**instance.eno=validated\_data.get('eno',instance.eno)**

**instance.ename=validated\_data.get('ename',instance.ename)**

**instance.esal=validated\_data.get('esal',instance.esal)**

**instance.eaddr=validated\_data.get('eaddr',instance.eaddr)**

**instance.save()**

**return instance**

**eno=serializers.IntegerField()**

**ename=serializers.CharField(max\_length=40)**

**esal=serializers.FloatField()**

**eaddr=serializers.CharField(max\_length=100)**

**object level validation: use the method validate()**

**serializers.py**

**def validate(self,data):# data total data**

**ename=data.get(‘ename’)**

**esal=data.get(‘esal’)**

**if esal<5000:**

**raise serializers.ValidationError(“Sal cannot be <5000”)**

**return data**

**when is\_valid() is called,both field level and object level implementations get executed,**

**3.By using validators**

**Ename’s max\_length=4,**

**In serializers.py**

**ename=serializers.CharField(max\_length=6)**

**models.py**

**ename=models.CharField(max\_length=6)**

**now if we try to create a resource then we get the following error**

**{'msg': 'error in creation'}**

**That means max\_length is an inbuilt validator**

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

**def multiples\_of\_1000(value):**

**print("validation by validator attribute")**

**if value%1000!=0:**

**raise Exception("Emp sal must be multiple of 1000")**

**….**

**esal=serializers.FloatField(validators=[multiples\_of\_1000])**

**Model Serializers**

**For normal serializers,**

**1.We have to specify all fields explicitly**

**2.Implement create(),update() explicitly**

**Serializers.py**

**class EmployeeSerializer(serializers.Modelserializer):**

**class Meta:**

**model=Employee**

**fields=”\_\_all\_\_”**

**test.py**

**….**

**Model serialization by default provides create(),update() methods**

**DRF VIEWS**

**DRF provides 2 classes to define business logic 1.APIView 2.ViewSet**

**class EmployeeCRUDCBV(APIView|ViewSet):**

**1.API view:Child class view ,get(),post() etc.we have write code explicitly and the complete control is over code.clear execution flow,best suited for operations like working with multiple data sources,other third party apis,map urls and views explicitly**

**2.Viewset:list(),create(),retrieve(),update(),destroy(),partial\_update().Most of the code gets generated automatically.B est suited for simple operations .urls are implemented by using router concept.Not required to map urls.py**

**We use browsable api to check api functionality**

**views.py**

**from django.shortcuts import render**

**from rest\_framework.views import APIView**

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

**from myApp.serializers import NameSerializer**

**class TestApiView(APIView):**

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

**colors=['red','blue','black','green']**

**return Response({'msg':'Happy bday','colors':colors})**

**#Response converts dict to json data**

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

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

**#data by browsable api interface**

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

**name=serializer.data.get('name')**

**msg="Hello {} happy bday -:".format(name)**

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

**else:**

**return Response(serializer.errors,status=404)**

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

**return Response({'msg':'put method called'})**

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

**return Response({'msg':'patch method called'})**

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

**return Response({'msg':'delete method called'})**

**urls.py**

**from django.contrib import admin**

**from django.urls import path**

**from myApp.views import TestApiView**

**urlpatterns = [**

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

**path('api/',TestApiView.as\_view())**

**]**

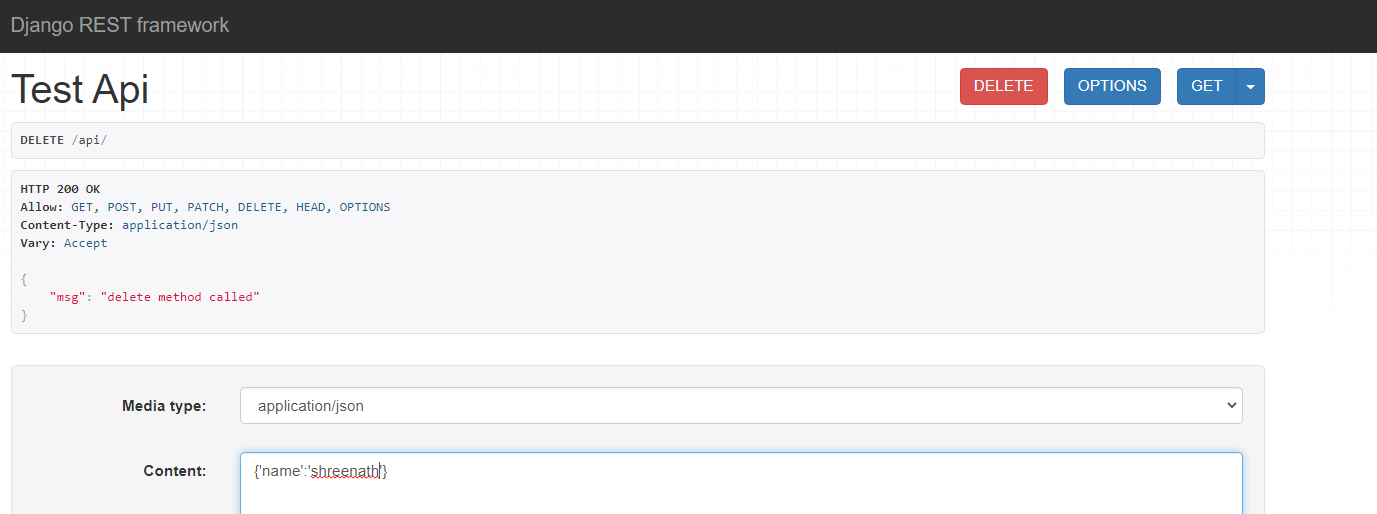
**serializers.py**

**from rest\_framework import serializers**

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

**name=serializers.CharField(max\_length=30)**

**perform make make migrations,migrate,run server and send request**

****