Single endpoint for all operations

**url('api1/',views.AllStudent.as\_view()),**

**url('api2/(?P<id>\d+)/$',views.ParticularStudent.as\_view())**

**All these methods use upto 2 endpoints but we are supposed to use only**

**1 end point to perform all the CURD operation with a single api call**

**SINGLE API FOR ALL CRUD METHODS**

**models.py**

**from django.db import models**

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

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

**rollno=models.IntegerField()**

**marks=models.FloatField()**

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

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

**return self.name**

**admin.py**

**from django.contrib import admin**

**from myApp.models import Student**

**# Register your models here.**

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

**l=['id','name','rollno','marks','address']**

**admin.site.register(Student,StudentAdmin)**

**forms.py**

**from django import forms**

**from myApp.models import Student**

**class StudentForm(forms.ModelForm):**

**def clean\_marks(self):**

**m=self.cleaned\_data['marks']**

**if m<35:**

**raise Exception("Min marks must be 35 ")**

**return m**

**class Meta:**

**model=Student**

**fields="\_\_all\_\_"**

**mixins.py**

**from django.core.serializers import serialize**

**from django.http import HttpResponse**

**import json**

**class SerilizeMixin(object):**

**def fun(self,qs):**

**json\_data=serialize('json',qs)**

**pdata=json.loads(json\_data)**

**final\_list=[]**

**for ob in pdata:**

**required\_info=ob['fields']**

**final\_list.append(required\_info)**

**json\_data=json.dumps(final\_list)**

**return json\_data**

**class HttpResponseMixin(object):**

**def render\_to\_http\_response(self,json\_data,status=200):**

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

**myApp/utils.py**

**import json**

**def is\_json(data):**

**try:**

**p\_data=json.loads(data)**

**valid=True**

**except Exception:**

**valid=False**

**return valid**

**urls.py**

**from django.contrib import admin**

**from django.conf.urls import url**

**from myApp import views**

**urlpatterns = [**

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

**url('api/',views.StudentCBV.as\_view())**

**]**

**views.py**

**from django.shortcuts import render**

**from myApp.models import Student**

**from myApp.forms import StudentForm**

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

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

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

**from myApp.utils import is\_json**

**from myApp.mixins import SerilizeMixin,HttpResponseMixin**

**import json**

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

**class StudentCBV(View,HttpResponseMixin,SerilizeMixin):**

**def get\_rec\_by\_id(self,id):**

**try:**

**stud=Student.objects.get(id=id)**

**except Exception:**

**stud=None**

**return stud**

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

**data=request.body #data is a json data it may be empty or it may contain id**

**valid=is\_json(data)**

**if not valid:**

**json\_data=json.dumps({'msg':'invalid'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**pdata=json.loads(data)**

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

**if id is not None:**

**stud=self.get\_rec\_by\_id(id)**

**if stud is None:**

**json\_data=json.dumps({'msg':'invalid record'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**json\_data=self.fun([stud])**

**return self.render\_to\_http\_response(json\_data)**

**qs=Student.objects.all() # if id is None**

**json\_data=self.fun(qs)**

**return self.render\_to\_http\_response(json\_data)**

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

**data=request.body**

**valid=is\_json(data)**

**if not valid:**

**json\_data=json.dumps({'msg':'invalid'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**p\_data=json.loads(data)**

**form=StudentForm(p\_data)**

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

**form.save(commit=True)**

**json\_data=json.dumps({'msg':'Successfully created the record'})**

**return self.render\_to\_http\_response(json\_data)**

**if form.errors():**

**json\_data=json.dumps({'msg':'Form submission error'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

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

**data=request.body**

**valid=is\_json(data)**

**if not valid:**

**json\_data=json.dumps({'msg':'Not a json data'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**provided\_data=json.loads(data)**

**id=provided\_data.get('id',None)**

**if id is None:**

**json\_data=json.dumps({'msg':'invalid id'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**stud=self.get\_rec\_by\_id(id)**

**if stud is None:**

**json\_data=json.dumps({'msg':'invalid record'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**original\_data={'name':stud.name,'rollno':stud.rollno,'marks':stud.marks,'address':stud.address}**

**original\_data.update(provided\_data)**

**form=StudentForm(original\_data,instance=stud)**

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

**form.save(commit=True)**

**json\_data=json.dumps({'msg':'Updation is successful'})**

**return self.render\_to\_http\_response(json\_data)**

**if form.errors:**

**json\_data=json.dumps({'msg':'Error in form submission'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

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

**data=request.body**

**valid=is\_json(data)**

**if not valid:**

**json\_data=json.dumps({'msg':'Not a json data'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**provided\_data=json.loads(data)**

**id=provided\_data.get('id',None)**

**if id is None:**

**json\_data=json.dumps({'msg':'Invalid id'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**stud=self.get\_rec\_by\_id(id)**

**if stud is None:**

**json\_data=json.dumps({'msg':'invalid record'})**

**return self.render\_to\_http\_response(json\_data,status=404)**

**status,item=stud.delete()**

**if status==1:**

**print("Deleted successfully")**

**else:**

**print("Deletion error")**

**At project level test.py**

**import json**

**import requests**

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

**ENDPOINT='api/'**

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

**data={} #you need all records**

**if id is not None:**

**data={'id':id} # A record with a particular id**

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

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

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

**def create\_resource():**

**stud\_data={'name':'Anand','rollno':201,'marks':88,'address':'Delhi'}**

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

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

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

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

**stud\_data={'id':id,'name':'New user','marks':88,'address':'Unknown place'}**

**response=requests.put(BASE\_URL+ENDPOINT,data=json.dumps(stud\_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--->To retrive the records \n 2--->To create a new record 3--->To update a record \n 4---->To delete a record\n")**

**print("Enter your choice")**

**ch=int(input())**

**if ch==1:**

**get\_resource()**

**elif ch==2:**

**create\_resource()**

**elif ch==3:**

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

**update\_resource(id)**

**elif ch==4:**

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

**delete\_resource(id)**

**Developing web apis by using third party django rest framework**

**The browsable API ensures that all the endpoints you create in your API are able to respond both with machine readable (ie JSON) and human readable (ie HTML) representations.This feature is used to generate HTML o/p for different resource.It facilitates interaction with rest web services through any web browser.It helps us to use web browser to surf through the API and can make different HTTP requests.**

**Specialty of DRF**

**1.It provides browsable apis**

**2.Security token authentication**

**3.In built serializers that support ORM and non-ORM**

**4.Extensive documentation**

**5.Used by Mozilla,redhat**

**Installation**

**1.pip install djangorestframework**

**2.pip install markdown # used for browsable apis**

**3.pip install django-filter # filtering support**

**4.pip freeze>requirements.txt**

**settings.py**

**INSTALLED\_APPS=[**

**…**

**‘myApp’,**

**‘rest\_framework’**

**]**

**from django.urls import path,include**

**Mapping django’s url pattern to rest\_framework’s url pattern**

**path(‘api/’,include(‘rest\_framework.urls’))**

**DRF SERIALIZERS**

**Perform three activities 1.serialization 2.Deserialization 3.Validation**

**Note:**

**class StudentForm(forms.Form):**

**sname=forms.CharField()**

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

**sname= serializers.CharField(max\_length)**

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

**class Meta:**

**model=Student**

**fields=”\_\_all\_\_”**

**urls.py**

**from django.contrib import admin**

**from django.urls import path,include**

**urlpatterns = [**

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

**path('api-auth/',include('rest\_framework.urls'))**

**]**

**models.py**

**from django.db import models**

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

**eno=models.IntegerField()**

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

**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)**

**myApp/serializers.py**

**from rest\_framework import serializers**

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

**eno=serializers.IntegerField()**

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

**esal=serializers.FloatField()**

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

**makemigrations,migrate,create superuser and runserver**

**The process of converting complex types(query set or model instance) to dict[python native form]🡪Serialization**

**Ex:If we convert an employee object [querySet/model instance] to dict,then we can easily convert into json form**

**Py manage.py shell**

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

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

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

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

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

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

**>>> emp**

**<Employee: Employee object (1)>**

**>>> emp.eno**

**101**

**>>> emp.ename**

**'shreekanth'**

**>>> eserializer=EmployeeSerializer(emp)**

**>>> eserializer.data**

**{'eno': 101, 'ename': 'shreekanth', 'esal': 70000.0, 'eaddr': 'bangalore'}**

**>>> type(emp)**

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

**>>> type(eserializer)**

**<class 'myApp.serializers.EmployeeSerializer'>**

**Converting python native type(dict) to Json data**

**Note**

**class One:**

**def fun(self):**

**print("inside fun")**

**One().fun()**

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

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

**>>> json\_data**

**b'{"eno":101,"ename":"shreekanth","esal":70000.0,"eaddr":"bangalore"}'**