Writing CBV by using Mixin class:

mixins.py:

from django.http import JsonResponse

class JsonResponseMixin(object):

def render\_to\_json\_response(self,context,\*\*kwargs):

return JsonResponse(context,\*\*kwargs)

CBV:

from testapp.mixins import JsonResponseMixin

class JsonCBV2(JsonResponseMixin,View):

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

employee\_data={'eno':100,'ename':'Naresh','esal':1000,'eaddr':'Hyderabad'}

return self.render\_to\_json\_response(employee\_data)

Performing database CRUD operations by using web api without

REST Framework:

models.py

from django.db import models

# Create your models here.

class Employee(models.Model):

eno=models.IntegerField()

ename=models.CharField(max\_length=64)

esal=models.FloatField()

eaddr=models.CharField(max\_length=64)

admin.py

from django.contrib import admin

from testapp.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

list\_display=['id','eno','ename','esal','eaddr']

admin.site.register(Employee,EmployeeAdmin)

views.py

from django.shortcuts import render

from django.views.generic import View

from testapp.models import Employee

import json

from django.http import HttpResponse

# Create your views here.

class EmployeeCRUDCBV(View):

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

emp=Employee.objects.get(id=2)

data={

'eno':emp.eno,

'ename':emp.ename,

'esal':emp.esal,

'eaddr':emp.eaddr,

}

json\_data=json.dumps(data)

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

Without Hardcoding id Value

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

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

....

urls.py

from django.contrib import admin

from django.urls import path

from app1 import views

urlpatterns = [

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

path('get/<int:id>/',views.EmployeeCRUDCBV.as\_view())

]

test.py

import requests

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

ENDPOINT='api/'

n=input('Enter required id:')

r=requests.get(BASE\_URL+ENDPOINT+n+'/')

data=r.json()

print(data)

Serialization:

The process of converting object from one form to another form is called serialization.

Eg: converting python dictionary object to json

json\_data = json.dumps(data)

Serialization by using django.core.serializers Module:

django provides inbuilt module serializers to perform serialization very easily.This module contains serialize() function for this activity.

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

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

json\_data=serialize('json',[emp,],fields=('eno','ename'))

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

If we are not specifying fields attribute, then all fields will be included in json data. For security reasons, if we don't want to provide access to some fields then this fields attribute is very helpful.

Note: Here exclude attribute is not allowed

To get all Records:

class EmployeeListCBV(View):

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

qs=Employee.objects.all()

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

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

urls.py

path('list/',views.EmployeeListCBV.as\_view())

test.py

import requests

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

ENDPOINT='list/'

r=requests.get(BASE\_URL+ENDPOINT)

data=r.json()

print(data)

Note: In the output we are getting some extra meta information also.

[{'model': 'testapp.employee', 'pk': 1, 'fields': {'eno': 100, 'ename': 'naresh', 'esal': 1000.0,

'eaddr': 'Mumbai'}}, {'model': 'testapp.employee', 'pk': 2, 'fields': {'eno': 200, 'ename':

'Suresh', 'esal': 2000.0, 'eaddr

': 'Hyderabad'}}, {'model': 'testapp.employee', 'pk': 3, 'fields': {'eno': 300, 'ename':

'Rajesh', 'esal': 3000.0, 'eaddr': 'Hyderabad'}}, {'model': 'testapp.employee', 'pk': 4, 'fields': {'eno': 400, 'ename': 'Ramesh', '

esal': 4000.0, 'eaddr': 'Bangalore'}}

How to get only Original Database Data:

class EmployeeListCBV(View):

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

qs=Employee.objects.all()

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

pdict=json.loads(json\_data)

final\_list=[]

for obj in pdict:

final\_list.append(obj['fields'])

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

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

Output:

[{'eno': 100, 'ename': 'Naresh', 'esal': 1000.0, 'eaddr': 'Mumbai'}, {'eno': 200, 'ename':

'Suresh', 'esal': 2000.0, 'eaddr': 'Hyderabad'}, {'eno': 300, 'ename': 'remash', 'esal': 3000.0,

'eaddr': 'Hyderabad'}, {'eno': 400, 'ename': 'rajesh', 'esal': 4000.0, 'eaddr': 'Bangalore'}]

Seperate serialization Code into SerializeMixin:

mixins.py

from django.core.serializers import serialize

import json

class SerializeMixin(object):

def serialize(self,qs):

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

pdict=json.loads(json\_data)

final\_list=[]

for obj in pdict:

final\_list.append(obj['fields'])

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

return json\_data

views.py

class EmployeeListCBV(SerializeMixin,View):

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

qs=Employee.objects.all()

json\_data=self.serialize(qs)

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

We can also use mixin to get a particular record data as follows

class EmployeeCRUDCBV(SerializeMixin,View):

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

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

json\_data=self.serialize([emp,])

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

Output: [{'eno': 200, 'ename': 'ramesh', 'esal': 2000.0, 'eaddr': 'Hyderabad'}]

Error Handling in the API:

It is not recommended to display our django error information directly to the partner

applications.Hence it is highly recommened to perform error handling.

class EmployeeCRUDCBV(SerializeMixin,View):

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

try:

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

except Employee.DoesNotExist:

json\_data=json.dumps({'msg':'Specified Record Not Found'})

else:

json\_data=self.serialize([emp,])

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

Status Codes:

Status code represents the status of HttpResponse. The following are various possible

status codes.

1XX Informational

2XX Successful

3XX Redirection

4XX Client Error

5XX Server Error

Exception Handling in Partner Application (Python Script):

import requests

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

ENDPOINT='api/'

r=requests.get(BASE\_URL+ENDPOINT+'1/')

# if r.status\_code in range(200,300):

if r.status\_code==requests.codes.ok:

data=r.json()

print(data)

else:

print('Something goes wrong')

print('Status Code:',r.status\_code)

How to add Status Code to HttpResponse explicitly:

By using status attribute

Eg: HttpResponse(json\_data,content\_type='application/json',status=403)

How to render HttpResponse By using Mixin:

from django.http import HttpResponse

class HttpResponseMixin(object):

def render\_to\_http\_response(self,data,status=200):

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

views.py:

class EmployeeCRUDCBV(SerializeMixin,HttpResponseMixin,View):

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

try:

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

except Employee.DoesNotExist:

json\_data=json.dumps({'msg':'Specified Record Not Available'})

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

else:

json\_data=self.serialize([emp,])

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

How to use dumpdata Option:

We can dump our database data either to the console or to the file by using dumpdata

option. This option provides support for json and xml formats. The default format is json.

We can write this data to files also.

Commands:

1) py manage.py dumpdata testapp.Employee

Print data to the console in json format without identation

2) py manage.py dumpdata testapp.Employee --indent 4

Print data to the console in json format with identation

3) py manage.py dumpdata testapp.Employee >emp.json --indent 4

Write data to emp.json file instead of displaying to the console

4) py manage.py dumpdata testapp.Employee --format json >emp.json --indent 4

We are specifying format as json explicitly

5) py manage.py dumpdata testapp.Employee --format xml --indent 4

Print data to the console in xml format with identation

6) py manage.py dumpdata testapp.Employee --format xml > emp.xml --indent 4

Write data to emp.xml file instead of displaying to the console