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': 'naresh', '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 program):

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

Note: dumpdata option provides support only for 3 formats

1) json(default)

2) XML

3) YAML (YAML Ain't Markup Language)

YAML is a human readable data serialization language, which is supported by multiple languages.

How to Create Resource from partner application by using API

(POST Request):

test.py(Partner Application)

import json

import requests

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

ENDPOINT='api/'

def create\_resource():

new\_emp={

'eno':600,

'ename':'Shiva',

'esal':6000,

'eaddr':'Chennai',

}

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

print(r.status\_code)

print(r.text)

print(r.json())

create\_resource()

Note: For POST Requests, compulsory CSRF verification should be done.If it fails our request will be aborted

How to disable CSRF Verification:

--> Just for our testing purposes we can disable CSRF verification, but not recommended in production environment.

--> We can disable CSRF verification at Function level, class level or at project level

1) To disable at Function/Method Level:

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

@csrf\_exempt

def my\_view(request):

body

This approach is helpful for Function Based Views(FBVs)

2) To disable at class level:

If we disable at class level then it is applicable for all methods present inside that class.

This approach is helpful for class based views(CBVs).

Code:

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

from django.utils.decorators import method\_decorator

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

class EmployeeListCBV(SerializeMixin,View):

3) To disable at Project globally:

Inside settings.py file comment the following middleware

'django.middleware.csrf.CsrfViewMiddleware

post() Method Logic:

--> Inside post method we can access data sent by partner application by using

request.body.

--> First we have to check whether this data is json or not

How to Check Data is json OR not?

utils.py:

import json

def is\_json(data):

try:

real\_data=json.loads(data)

valid=True

except ValueError:

valid=False

return valid

views.py:

from testapp.utils import is\_json

...

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

class EmployeeListCBV(HttpResponseMixin,SerializeMixin,View):

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

data=request.body

if not is\_json(data):

return self.render\_to\_http\_response(json.dumps({'msg':'plz send valid json

data only'}),status=400)

json\_data=json.dumps({'msg':'post method'})

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

Creating Model Based Form to hold Employee Data:

forms.py

from django import forms

from testapp.models import Employee

class EmployeeForm(forms.ModelForm):

class Meta:

model=Employee

fields='\_\_all\_\_'

To validate emp Salary:

from django import forms

from testapp.models import Employee

class EmployeeForm(forms.ModelForm):

#validations

def clean\_esal(self):

inputsal=self.cleaned\_data['esal']

if inputsal < 5000:

raise forms.ValidationError('The minimum salary should be 5000')

return inputsal

class Meta:

model=Employee

fields='\_\_all\_\_'

views.py

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

from django.utils.decorators import method\_decorator

from testapp.utils import is\_json

from testapp.forms import EmployeeForm

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

class EmployeeListCBV(HttpResponseMixin,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')

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

data=request.body

if not is\_json(data):

return self.render\_to\_http\_response(json.dumps({'msg':'plz send valid json

data only'}),status=400)

empdata=json.loads(request.body)

form=EmployeeForm(empdata)

if form.is\_valid():

obj = form.save(commit=True)

return self.render\_to\_http\_response(json.dumps({'msg':'resource created su

ccessfully'}))

if form.errors:

json\_data=json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data,status=400)

Performing Update Operation (put() Method):

Partner Application(test.py)

import requests

import json

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

ENDPOINT='api/'

def update\_resource():

new\_data={

eaddr:'Ameerpet',

}

r=requests.put(BASE\_URL+ENDPOINT+'8/',data=json.dumps(new\_data))

print(r.status\_code)

# print(r.text)

print(r.json())

update\_resource()

views.py

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

class EmployeeCRUDCBV(SerializeMixin,HttpResponseMixin,View):

def get\_object\_by\_id(self,id):

try:

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

except Employee.DoesNotExist:

emp=None

return emp

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

obj=self.get\_object\_by\_id(id)

if obj is None:

json\_data=json.dumps({'msg':'No matched record found, Not possible to per

form updataion'})

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

data=request.body

if not is\_json(data):

return self.render\_to\_http\_response(json.dumps({'msg':'plz send valid json

data only'}),status=400)

new\_data=json.loads(data)

old\_data={

'eno':obj.eno,

'ename':obj.ename,

'esal':obj.esal,

'eaddr':obj.eaddr,

}

for k,v in new\_data.items():

old\_data[k]=v

form=EmployeeForm(old\_data,instance=obj)

if form.is\_valid():

form.save(commit=True)

json\_data=json.dumps({'msg':'Updated successfully'})

return self.render\_to\_http\_response(json\_data,status=201)

if form.errors:

json\_data=json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data,status=400

Note:

1) form = EmployeeForm(old\_data)

form.save(commit=True)

The above code will create a new record

2) form = EmployeeForm(old\_data,instance=obj)

form.save(commit=True)

The above code will perform updations to existing object instead of creating new object.

Performing Delete Operation:

partner application(test.py)

import requests

import json

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

ENDPOINT='api/'

def delete\_resource():

r=requests.delete(BASE\_URL+ENDPOINT+'9/')

print(r.status\_code)

# print(r.text)

print(r.json())

delete\_resource()

views.py

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

obj=self.get\_object\_by\_id(id)

if obj is None:

json\_data=json.dumps({'msg':'No matched record found, Not possible to per

form deletion'})

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

status,deleted\_item=obj.delete()

if status==1:

json\_data=json.dumps({'msg':'Resource Deleted successfully'})

return self.render\_to\_http\_response(json\_data,status=201)

json\_data=json.dumps({'msg':'unable to delete ...plz try again'})

return self.render\_to\_http\_response(json\_data,status=500)

Note:

1) obj.delete() returns a tuple with 2 values.

2) The first value represents the status of the delete. If the deletion is success then its

value will be 1.

3) The secord value represents the deleted object.

Problem with Our Own Web API Framework:

The following are endpoints for CRUD operations in the above application

1) To get a particular resource: api/id/

2) To get all resources : api/

3) To create(post) a resource: api/

4) To update a resource: api/id

5) To delete a resource: api/id

According industry standard, in all rest api frameworks, endpoint should be same for all

CRUD operations. In our application we are violating this rule.

\*\*\*For all CRUD operations, ENDPOINT should be same