Update operation/put request by using Serializers:

To handle put requests, we have to override update() method in the serializer class.

from rest\_framework import serializers

from testapp.models import Employee

class EmployeeSerializer(serializers.Serializer):

eno=serializers.IntegerField()

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

esal=serializers.FloatField()

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

def create(self,validated\_data):

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

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

views.py

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

json\_data=request.body

stream=io.BytesIO(json\_data)

data=JSONParser().parse(stream)

id=data.get('id')

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

#serializer=EmployeeSerializer(emp,data=data)

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

if serializer.is\_valid():

serializer.save()

msg={'msg':'Resource Updated Succesfully'}

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

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

else:

json\_data=JSONRenderer().render(serializer.errors)

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

Note: By default for update operation, we have to provide all fields. If any field is missing, then we will get ValidationError.

If we don't want to provide all fields, then we have to use 'partial' attribute.

serializer = EmployeeSerializer(emp,data=data)

In this case we have to provide all fields for updation

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

In this case we have to provide only required fields but not all.

Note: By using serializers, we can perform get(),post() and put() operations. There is role of serializers in delete operation.

3) Validations by using Serializers

We can implement validations by using the following 3 ways

1) Field Level Validations

2) Object Level Validations

3) By using validators

1) Field Level Validations

Syntax: validate\_fieldname(self,value):

Eg: To check esal should be minimum 5000

class EmployeeSerializer(serializers.Serializer):

....

def validate\_esal(self,value):

if value<5000:

raise serializers.ValidationError('Employee Salaray Should be Minimum 5000')

return value

2) Object Level Validations:

If we want to perform validations for multiple fields simultaneously then we should go for object level validations.

Eg: If ename is 'ramesh' then salary should be minimum 60000

def validate(self,data):

ename=data.get('ename')

esal=data.get('esal')

if ename.lower()=='ramesh':

if esal<60000:

raise serializers.ValidationError('ramesh Salary should be minimum 60K')

return data

Use Cases:

1) First entered pwd and re-entered pwd must be same.

2) First entered account number and re-entered account number must be same

These validations we can implement at object level.

3) Validations by using Validator Field:

def multiples\_of\_1000(value):

if value % 1000 != 0:

raise serializers.ValidationError('Salary should be multiples of 1000s')

class EmployeeSerializer(serializers.Serializer):

...

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

..

Note: If we implement all 3 types of validations then the order of priority is

1) validations by using validator

2) validations at field level

3) validations at object level

Complete Application for Serializers:

serializers.py

from rest\_framework import serializers

from testapp.models import Employee

def multiples\_of\_1000(value):

print('validations by using validator')

if value % 1000 != 0:

raise serializers.ValidationError('Salary should be multiples of 1000s')

class EmployeeSerializer(serializers.Serializer):

eno=serializers.IntegerField()

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

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

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

def validate\_esal(self,value):

print('validations at field level')

if value<5000:

raise serializers.ValidationError('Employee Salaray Should be Minimum 5000')

return value

def validate(self,data):

print('validations at object level')

ename=data.get('ename')

esal=data.get('esal')

if ename.lower()=='ramesh':

if esal<60000:

raise serializers.ValidationError('ramesh Salary should be minimum 60K')

return data

def create(self,validated\_data):

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

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

views.py

from django.shortcuts import render

from django.views.generic import View

import io

from rest\_framework.parsers import JSONParser

from app1.models import Employee

from app1.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)

data=JSONParser().parse(stream)

id=data.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)

data=JSONParser().parse(stream)

serializer=EmployeeSerializer(data=data)

if serializer.is\_valid():

serializer.save()

msg={'msg':'Resource Created Succesfully'}

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

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

json\_data=JSONRenderer().render(serializer.errors)

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

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

json\_data=request.body

stream=io.BytesIO(json\_data)

data=JSONParser().parse(stream)

id=data.get('id')

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

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

if serializer.is\_valid():

serializer.save()

msg={'msg':'Resource Updated Succesfully'}

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

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

json\_data=JSONRenderer().render(serializer.errors)

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

test.py

import requests

import json

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

ENDPOINT='api/'

def update\_resource(id):

new\_data={

'id':id,

# 'eno':700,

'ename':'ramesh123',

'esal':15000,

# 'eaddr':'Hyd'

}

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

print(r.status\_code)

# print(r.text)

print(r.json())

update\_resource(3)

ModelSerializers:

 If our serializable objects are Django model objects, then it is highly recommended to go for ModelSerializer.

 ModelSerializer class is exactly same as regular serializer classe except the following differences

1) The fields will be considered automatically based on the model and we are not required to specify explicitly.

2) It provides default implementation for create() and update() methods.

Note: ModelSerializer won't provide any extra functionality and it is just for typing shortcut.

We can define ModelSerializer class as follows:

1) class EmployeeSerializer(serializers.ModelSerializer):

2) class Meta:

3) model=Employee

4) fields='\_\_all\_\_'

Here we are not required to specify fields and these will be considered automatically

based on Model class. We are not required to implement create() and update() methods,

because ModelSerializer class will provide these methods.

\*\*\*Note: If we want to define validations for any field then that particular field we have to declare explicitly.