

→ To generating or creating the our user defined views, in views.py.  
your following the some concepts

Eg:- APIView, Mixins, Generics, ViewSets, .....

these all concepts are using for converting data, from one form to another form without writing the more lines automatically

APIView:-

→ if we are using the APIView concepts then it is automatically checking the user entered data is json type or not, but not checking valid type or not  
→ To do database functionalities for id & non-id operations, we are going to use 'http' methods only

Ex:- class cbn(APIView):

def get(self, request):

≡

def post(self, request):

≡

→ To return response in django we are using 'Http' response method  
→ In restApi we are using Response()  
→ Response(), taking input as dictionary and converting to json type and return to browser or partner application  
→ In django we are sending the 'http Request' we are getting the 'HttpResponse' back  
→ But in RestApi we are sending only 'Request' object and getting the 'Response' object

Q write a program to do CRUD operations on database by using APIView Concepts

step-1: project name : APIview-project

step-2: app name : APIview-app

step-3: database name : APIview-db

step-4: 3.1: open mysql and run the command like below  
    > create database apiview-db;

step-4: configure database name in settings.py file, and add our application, REST-<sup>framework</sup>Application inside installed apps section

```
INSTALLED_APPS = [  
      
    'APIview-app',  
    'rest-framework',  
]
```

step-5: open project level --init.py file and write the below code for pymysql dependency problem

```
import pymysql  
pymysql.install_as_MySQLdb()
```

-- ~~init.py~~ --  
~~import pymysql~~

step-6:- open models.py file

from django.db import models

class employee (models.Model):

    eno = models.IntegerField(primary\_key=True)

    ename = models.CharField(max\_length=100)

    esal = models.IntegerField()

    def \_\_str\_\_(self):

        return self.ename



step-7:- create serializers.py inside our 'app'

```
from rest_framework import Serializers
from .models import employee
class employeeSerializer(Serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = '__all__'
```

step-8: open views.py

```
from django.shortcuts import render
from .models import Employee
from .serializers import employeeSerializer
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
```

#Non-Id based class operations

```
class employeeListView(APIView):
```

```
    def get(self, request):
```

```
        emp = Employee.objects.all()
```

```
        serializer = employeeSerializer(emp, many=True)
```

```
        return Response(serializer.data, status=status.HTTP_200_OK)
```

```
    def post(self, request):
```

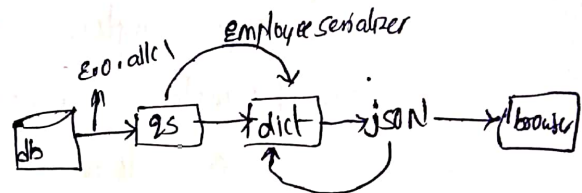
```
        (emp = Employee.objects.all())
```

```
        data = request.data
        serializer = employeeSerializer(emp, many=True)
```

```
        if serializer.is_valid():
```

```
            serializer.save()
```

```
            return Response(serializer.data, status=status.HTTP_201_CREATED)
```



else:

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

# id based operations

class employeeDetailview(APIView):

def get(self, request, id):

try:

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

except Employee.DoesNotExist:

return Response('Record not found')

else:

serializer = EmployeeSerializer(emp)

return Response(serializer.data, status=status.HTTP\_200\_OK)

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

try:

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

except Employee.DoesNotExist:

emp = None

return emp

def put(self, request, id):

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

if emp is None:

return Response('Record not available')

else:

serializer = EmployeeSerializer(emp, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_200\_OK)

else:

HTTP\_200\_OK



return Response (serializer . errors , status = status . HTTP\_400\_BAD\_REQUEST)

```
def delete (self , request , id):  
    emp = self . get . object _ by _ id (id) .  
    if emp is None :  
        return Response ('Record is not available')  
    else :  
        emp . delete ()  
        return Response ('Record deleted successfully' , status = status . HTTP_204_NO_CONTENT)
```

### open project url's .py

→ open project level url's py to calling application levels url's

from django . contrib import admin

from django . urls import path , include

url patterns = [

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

path ('api/' , include ('apiview-app . urls'))

]

→ create url's py in application level

→ Right click on app → New → python file → url's .py

from apiviewapp import views

from django . urls import path

url patterns = [

path ('emp/' , views . employee\_list\_view . as\_view ()) ,

path ('emp/<int: id>/' , views . employee\_detail\_view . as\_view ())

]

→ Execute 'makemigrations'

`py manage.py makemigrations`

→ Execute 'migrate' command

`py manage.py migrate`

→ Execute 'runserver' command

`py manage.py runserver`

→ click on url, it opens browser

127.0.0.1:2020/api/emp  
200 OK  
Alt: GET /api/emp/...  
Content-type: application/json  
Vary: accept  
[ ]

\* By default no data is database, so empty [ ] is displaying

\* To creating some data into database use 'post' method and send 'json' data

content {  
 2 'eno': 10  
 'ename': 'srinivas'  
 3 'esal': 10000  
}

\* Then this json object created in database

\* Now click get button to

[POST] ← get all existing data from db

→ To get id based records, execute 'url' like below

127.0.0.1:2020/api/emp/10  
200 OK  
[  
 {  
 'eno': 10,  
 'ename': 'Rohit',  
 'esal': 20000  
 }  
]  
media type: application/json  
Content: {  
 'eno': 10  
 'ename': 'Rohit',  
 'esal': 20000  
}  
[PUT]

\* To delete the data, click delete button, then it's display one confirmation popup box

Are you sure you want to delete this employee detail?

[Cancel]

[Delete]

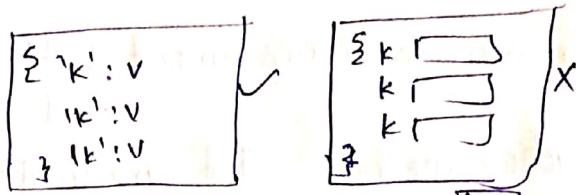
\* To delete the data click on 'Delete'

### DRAWBACKS :-

→ when we are using 'APIview' concept to developing the application it is reducing the more lines of code when compared to django related serialisation.



→ even though api view concept providing some burdens on developers  
eg:- it is not providing readymade templates, to (checking) create or update the data so, we need to provide data manually in the form of json object by following json rules



→ API view concept is not handling the "status" code properly. so as a developer we need to handle all the status codes respectively based on the requirement

→ it is by default not returning responses to 'partner' (another app) automatically. so manually by using response method we are returning responses properly

→ API view by default handling requested data is json & not only but it is not handling the requested data containing all fields data with valid types, that's why as a programmer we need to check manually by using "is-valid()"

→ Bcz of these all above burdens developers are not interested to use API view concept to developing the our application views

→ That's why we are choosing some other concepts like, "mrails" to developing the API's