MIXINS :-

→ RestApi is providing "mixing modules". this module providing several classes to performing the data base functionalities.

Ex:- ~~import mixins~~

from rest_framework.mixins import ListModel mixin,
CreateModel mixin,....

→ Mixins ~~module~~ class providing action methods that are used to provide the basic behaviour of respective classes

→ mixin classes providing action methods rather than defining the handler methods such as List(), retrieve(), create(), update() and destroy()

→ By using both handler methods and action methods in class based views to handle ~~the~~ corresponding request like get, post, get, delete

Eg:-

```
     handler method
     ↑
def get(self, request):
    return self.list(request)
                ↓
          action method
```

→ mixins module directly not supporting all the data base functionalities for this it is taking another module support

Ex:- generics
↓
this module providing "GenericAPIView" class for supporting the "browsable Api"

→ Display all mixin classes with respective action methods and handler methods

→ Mixins automatically checking data is JSON or not and also Valid type or not

→ it automatically providing 'status codes' based on response

→ it automatically returns 'response' based on the request

mixin_class	purpose	action.methods	handler methods
ListModelMixin	getting all	.list()	get()
CreateModelMixin	create one	.create()	post()
RetrieveModelMixin	getting one	.retrieve()	get()
UpdateModelMixin	update one	.update()	put()
DestroyModelMixin	delete one	.destroy()	delete()

→ it automatically provide ready made template to perform .CURD operations

→ In mixins we should be representing all our database model table information (instance) by using "queryset" field

→ for converting database data into dictionary and then JSON and also reverse process we should be "serializer_class" field with user defined serializer class

Eg:-
```
class Emp (mixin.ListModelMix, mixins.CreateModelmixin)
queryset = modelname. objects. all()
serializer_class = User defined serializerclass
```

project:-

create a project to performing the database operations by using mixins concept

step-1 : project name : mixins_project

step-2 : app name : mixins_app

step-3 : data base : mixinsdb

step-4 : configure database in settings.py and add our 'app' name and rest_framework inside "INSTALLED_APPS" : [- - - -

- - - -

'rest_framework']

Scanned by CamScanner

step-5:-
open models.py and Create our model

```
from django.db import models
class Emp (models.Model):
    eno = models.IntegerField(primary_key=True)
    ename = models.CharField(max_length=30)
    esal  = models.DecimalField(max_digits=10, decimal_places=2)
   created = models.DateTimeField(auto_now_add=True)
   modified = models.DateTimeField(auto_now ~~add~~ = True)
```

not modified

we can modify

step-7:-

create serializer.py inside our app

```
from rest_framework import serializer
from . mixins_app import models
class EmpSerializer (Serializer.ModelSerializer)
    class Meta:
        model = Emp
        fields = '__all__'
```

step-8:- open views.py and write the Id & Non-Id based Views

```
from .models import Emp
from .serializer import EmpSerializer
from rest_framework import mixins, ge
# Non-Id based clay
class Emplistview(mixins.ListModelMixins, mixins.CreateModel
                                   mixins, generic.GenericAPIView):
        queryset = Emp.objects.all()
```

Scanned by CamScanner

```
    serializer_class = EmpSerializer
    def get (self, request):
        return self.list(request)
    def post (self, request):
        return self.create(request)
#Id based operations
class  Empdetailview(mixins.RetriveModelMixins,mixins.UpdateModel
                     Mixins, mixins.DestroyModelMixins, generic.
                                                  GenericAPIView):
    queryset = Emp.objects.all()
    serializer_class = EmpSerializer
    def get(self,request ,pk):
        return self.retrieve (request)

    def put (self, request, pk):
        return self.update(request)

    def delete(self, request ,pk):
        return self.destroy(request)
```

step-9 :- open project url's.py

```
from django.urls import path,include
from django.contrib import admin
urlpatterns = [
    path ('admin/', admin.site.urls) ,
    path ( 'api/', include ('mixins-app.urls')),
]
```

step-10:- ~~open urls~~ create url's.py in app level and write the below code

```
from django.urls import path
from mixins_app import views
urlpatterns = [

    path('empl', views.EmplistView.as_view())
    path ('emp/<int:pk/', views.Empdetailview.as_view()]
```

step-11:- execute makemigrations, migrate, runserver commands

DRAWBACKS:-

* By using mixins we are getting only individual classes foreach requirement
  Ex:- To gettingall records we are using ListModelMixin
  for creating we are using CreateModelMixin ....
* but mixins not providing combination classes to providing above two operations at a time
* using mixins we need to write handler, & action method combination to performing the required functionalities

```
Ex:- def get (self, request, pk):
         return self.retrive(request)
```

* If we don't want to use this combination methods and if we want to use combination classes then we should go for generic concepts