Function based views for creating APIs :-

REST Framework also allows us to work with regular function based views.

→ It provides a set of simple decorators that wrap our function based views to ensure and they receive an instance of request, rather than usual django 'HttpRequest.'

→ It also return a response instead of django 'HttpResponse' and allow you to configuring how the request is processed

@api_view() :-

```
@api_view (http_method_names =['GET'])
```

⇒ the main functionality of Api_View decorator, which takes a list of http methods, that our ~~b~~ view should responde to

→ which is available in

```
from rest_framework.decorators import api_view
```

Eg:-
```
@api_view()
def helloworld(request):
        return Response ({"message": "Hello, world"})
```

→ By default @api_view() have 'get' method

→ this view will use the default renderers, parsers, authentication classes etc specified in the settings

405 method not allowed :-

→ By default @api_view() , only 'GET' methods.

→ other methods except 'get()' with response with '405' method not allowed

→ If you want to provide other methods behaviour also and we have change default behaviour of @api-view decorator, then we should specify what methods we want to use inside @api-view decorator

Example :-

```
@api_view(['GET','post'])
def hello_world(request):
        if request.method =='post':
                return Response({"message": "Got some data!", "data": request.data})
        else request.method == 'GET':
                return Responce({"message": "Hello, world!"})
```

project :-

create a project to performing the CURD operations by using the function based views

step-1: function based project.

step-2: function based app

step-3: functionbaseddb

step-4 : add appname & 'rest_framework' in the 'INSTALLED APPS', configuring the db details inside db section

step-5 : open product.py and write the logic

```
from django.db import models
class product (Models.Model):
        Product_id = models.IntegerField(primary_key=True)
        product_name = models.CharField(max_length=10)
        product_price = models.DecimalField(max_digits=10,
                                        decimal_places=3)
        product_color = models.CharField(max_~~le~~length=10)
```

```
def __str__(self):
        return self.product_name
```

step-6: Create serializers.py in app level and write the below code

```
from rest_framework import serializers
from.models import product
class product_serializer(serializers.ModelSerializer):
    class Meta:
        model = product
        fields = "__all__"
```

step-7: open views.py and create FBV

```
from.models import product
from.serializers import productSerializer
~~from~~ django
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

# create non-id based view
@api_view(['GET','POST'])
def productdataView(request):
        if request.method=='GET':
                products = product.objects.all()
                serializer = productSerializer(product, many=True)
                return Response(serializer.data,.status=status.HTTP_200_OK)
        elif request.method=='post':
                serializer = productserializer(data=request.data)
                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)


@api-view ['GET', 'PUT', 'DELETE'])
def productdetail (request, PK):
    # retrive, update or delete a code of product
    try:
        products = product.objects.get (pk=pk)
    except:
        return Response (status = status. HTTP-400-NOT-FOUND)

    if request.method == 'GET':
        serializer = productserializer (products)
        return Response (serializer.data)
    elif request.method == 'PUT':
        serializer = productserializer (products, data = request.data)
        if serializer.is_valid():
            serializer.save()
            return Response (serializer.data)
        return Response (Serializer.errors, status = status. HTTP-400-BAD-
                                                                REQUEST)
    elif request.method == 'DELETE':
        products.delete()
        return Response (status = status. HTTP-204-NO-CONTENT)
```

step-7:- Create url's.py in app level and write below code

```
from django.conf.urls import url
from .import views

urlpatterns=[
        url(r'^product/$', views.productlist),
        url(r'^product/(?P<pk>[0-9]+)', views.productdetail)
  ]
```

step-8: open project level url's.py

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import url, include
urlpatterns=[
      path('admin/', admin.site.urls),
      url(r'^api/', include('Function_based-app.urls'))
```