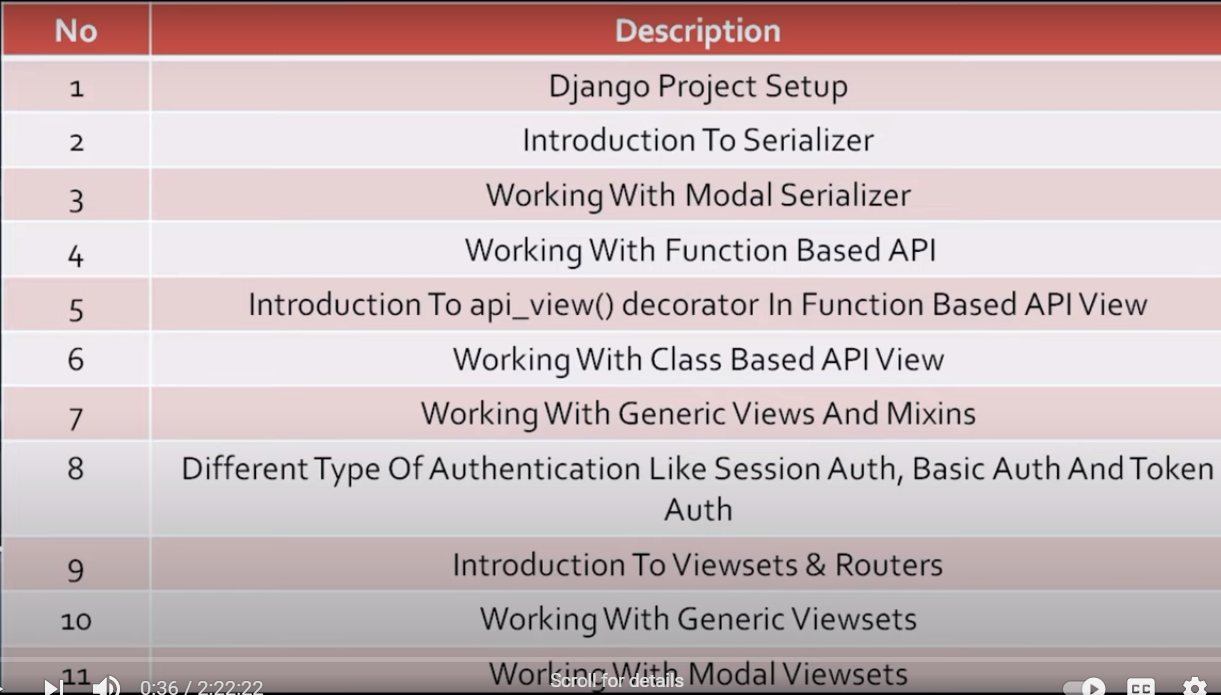
**Django Rest framework**



A restful Api acts a translator between two machines communicating over a web service.

We can say it’s an Api working on restful web service.

Rest Api return json files which can be interpreted by a large variety of devices.

* **Serializer**

Before sending data to client, we need to serialize it into json and Api’s end result is always json. Api communicate between multiple technologies which take json as their input.

Models and serializer files.

* Models.py

from django.db import models  
  
# Create your models here.  
  
class Article(models.Model):  
 title = models.CharField(max\_length=100)  
 author = models.CharField(max\_length=100)  
 email = models.CharField(max\_length=100)  
 date = models.DateTimeField(auto\_now\_add=True)  
  
 def \_\_str\_\_(self):  
 return self.title

* Serializer.py

from rest\_framework import serializers  
from .models import Article  
  
class ArticleSerializer(serializers.Serializer):  
 title = serializers.CharField(max\_length=100)  
 author = serializers.CharField(max\_length=100)  
 email = serializers.CharField(max\_length=100)  
 date = serializers.DateTimeField()  
  
# this below 2 methods are only required if we have serializer class only and when we have modes.serializer class  
  
 def create(self, validated\_data):  
 return Article.objects.create(validated\_data)  
  
 def update(self, instance, validated\_data):  
 instance.title = validated\_data.get('title',instance.title)  
 instance.author = validated\_data.get('author', instance.author)  
 instance.email = validated\_data.get('email', instance.email)  
 instance.date = validated\_data.get('date', instance.date)  
 instance.save()  
 return instance

Steps of serialization:

1. Create model object
2. Create serializer object and pass model object to it.
3. Create Json Renderer object and pass serializer object to it.

Python manage.py shell

In [5]: from rest\_app1.models import Article

In [6]: from rest\_app1.serializer import ArticleSerializer

In [7]: from rest\_framework.renderers import JSONRenderer

In [8]: from rest\_framework.parsers import JSONParser

In [9]: b = Article(title='article title2', author ='Prawez', email='Prawez@email.com')

In [10]: b.save()

In [11]: serializer = ArticleSerializer(b)

In [12]: serializer.data

Out[12]: {'id': 2, 'title': 'article title2', 'author': 'Prawez', 'email': 'Prawez@email.com'}

In [13]: content = JSONRenderer().render(serializer.data)

In [14]: content

Out[14]: b'{"id":2,"title":"article title2","author":"Prawez","email":"Prawez@email.com"}'

In [15]: serializer = ArticleSerializer(Article.objects.all(), many=True)

In [16]: serializer.data

Out[16]: [OrderedDict([('id', 1), ('title', 'Article Title'), ('author', 'parwez'), ('email', 'parvez@gmail.com')]), OrderedDict([('id', 2), ('title', 'article title2'), ('author', 'Prawez'), ('email', 'Prawez@email.com')])]

* **Model Serializer**
* Serializer.py

from rest\_framework import serializers  
from .models import Article  
  
class ArticleSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Article  
 fields = ['id','title','author','email']

Python manage.py shell

In [1]: from rest\_app1.serializer import ArticleSerializer

In [2]: serializer = ArticleSerializer()

In [3]: print(repr(serializer))

ArticleSerializer():

id = IntegerField(label='ID', read\_only=True)

title = CharField(max\_length=100)

author = CharField(max\_length=100)

email = CharField(max\_length=100)

* **FunctionBasedApiView**

**Views.py**

from django.shortcuts import render  
from django.http import HttpResponse,JsonResponse  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
  
# Create your views here.  
  
@csrf\_exempt  
def article\_list(request):  
 if request.method == 'GET':  
 articles = Article.objects.all()  
 serializer = ArticleSerializer(articles, many=True)  
 return JsonResponse(serializer.data,safe=False)  
  
 if request.method == 'POST':  
 data = JSONParser().parse(request)  
 serializer = ArticleSerializer(data=data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return JsonResponse(serializer.data,status=201)  
  
 return JsonResponse(serializer.errors,status=400)  
  
@csrf\_exempt  
def article\_detail(request,pk):  
 try:  
 article = Article.objects.get(pk=pk)  
 except Article.DoesNotExist:  
 return HttpResponse(status=404)  
 if request.method=="GET":  
 serializer = ArticleSerializer(article)  
 return JsonResponse(serializer.data)  
  
 elif request.method=="PUT":  
 data = JSONParser().parse(request)  
 serializer = ArticleSerializer(article, data=data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return JsonResponse(serializer.data)  
  
 return JsonResponse(serializer.errors, status=400)  
  
 elif request.method == 'DELETE':  
 article.delete()  
 return HttpResponse(status=204)

**Urls.py**

from django.contrib import admin  
from django.urls import path  
from . import views  
  
urlpatterns = [  
 path('article/', views.article\_list, name='article\_list'),  
 path('detail/<str:pk>',views.article\_detail, name='article\_detail'),  
  
]

* **Api\_view() Decorator in Function based Api View**

Views.py

from django.shortcuts import render  
from django.http import HttpResponse,JsonResponse  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.decorators import api\_view  
from rest\_framework.response import Response  
from rest\_framework import status  
  
  
# Create your views here.  
  
@api\_view(['GET','POST'])  
def article\_list(request):  
 if request.method == 'GET':  
 articles = Article.objects.all()  
 serializer = ArticleSerializer(articles, many=True)  
 return Response(serializer.data)  
  
 if request.method == 'POST':  
  
 serializer = ArticleSerializer(data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return JsonResponse(serializer.data,status=status.HTTP\_201\_CREATED)  
 return JsonResponse(serializer.errors,status=status.HTTP\_400\_BAD\_REQUEST)  
  
@api\_view(['GET','PUT','DELETE'])  
def article\_detail(request,pk):  
 try:  
 article = Article.objects.get(pk=pk)  
 except Article.DoesNotExist:  
 return HttpResponse(status=status.HTTP\_404\_NOT\_FOUND)  
 if request.method=="GET":  
 serializer = ArticleSerializer(article)  
 return Response(serializer.data)  
  
 elif request.method=="PUT":  
 serializer = ArticleSerializer(article, data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
  
 return JsonResponse(serializer.errors, status=status.HTTP\_404\_NOT\_FOUND)  
  
 elif request.method == 'DELETE':  
 article.delete()  
 return Response(status=status.HTTP\_204\_NO\_CONTENT)

* **Class based Api view**

Views.py

from django.shortcuts import render  
from django.http import HttpResponse, JsonResponse  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.decorators import api\_view  
from rest\_framework.response import Response  
from rest\_framework import status  
from rest\_framework.views import APIView  
  
# Create your views here.  
  
class ArticleApiView(APIView):  
  
 def get(self, request):  
 articles = Article.objects.all()  
 serializer = ArticleSerializer(articles, many=True)  
 return Response(serializer.data)  
  
 def post(self, request):  
 serializer = ArticleSerializer(data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return JsonResponse(serializer.data, status=status.HTTP\_201\_CREATED)  
 return JsonResponse(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)  
  
class ArticleDetails(APIView):  
  
 def get\_object(self, id):  
 try:  
 return Article.objects.get(id=id)  
 except Article.DoesNotExist:  
 return HttpResponse(status=status.HTTP\_404\_NOT\_FOUND)  
  
 def get(self,request, id):  
 article = self.get\_object(id)  
 serializer = ArticleSerializer(article)  
 return Response(serializer.data)  
  
 def put(self, request, id):  
 article = self.get\_object(id)  
 serializer = ArticleSerializer(article, data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
  
 return JsonResponse(serializer.errors, status=status.HTTP\_404\_NOT\_FOUND)  
  
 def delete(self,request, id):  
 article = self.get\_object(id)  
 article.delete()  
 return Response(status=status.HTTP\_204\_NO\_CONTENT)

Urls.py

from django.contrib import admin  
from django.urls import path  
from . import views  
  
urlpatterns = [  
path('article/', views.ArticleApiView.as\_view(), name='article\_list'),  
path('detail/<str:id>', views.ArticleDetails.as\_view(), name='article\_detail'),  
]

* **Generic Views and Mixins**

from django.shortcuts import render  
from django.http import HttpResponse, JsonResponse  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.decorators import api\_view  
from rest\_framework.response import Response  
from rest\_framework import status  
from rest\_framework.views import APIView  
from rest\_framework import generics  
from rest\_framework import mixins  
  
# Create your views here.  
  
class GenericApiViews(generics.GenericAPIView, mixins.ListModelMixin, mixins.CreateModelMixin, mixins.UpdateModelMixin,  
 mixins.RetrieveModelMixin,mixins.DestroyModelMixin):  
 serializer\_class = ArticleSerializer  
 queryset = Article.objects.all()  
  
 lookup\_field = 'id'  
  
 def get(self, request,id=None):  
 if id:  
 return self.retrieve(request)  
 else:  
 return self.list(request)  
  
 def post(self,request):  
 return self.create(request)  
  
 def put(self, request, id=None):  
 return self.update(request,id)  
  
 def delete(self, request, id):  
 return self.destroy(request, id)

Urls.py

from django.contrib import admin  
from django.urls import path  
from . import views  
  
urlpatterns = [  
path('generic/article/<int:id>/', views.GenericApiViews.as\_view(), name='article\_list'),  
path('detail/<str:id>', views.ArticleDetails.as\_view(), name='article\_detail'),  
]

* **Authentication**
* Basic Authentication
* Session Authentication
* Token Authentication

The request.user property will typically be set to an instance of the contrib.auth package's User class.

The request.auth property is used for any additional authentication information, for example, it may be used to represent an authentication token that the request was signed with.

**Note:** Don't forget that **authentication by itself won't allow or disallow an incoming request**, it simply identifies the credentials that the request was made with.

## [**BasicAuthentication:**](https://www.django-rest-framework.org/api-guide/authentication/#basicauthentication) This authentication scheme uses [HTTP Basic Authentication](https://tools.ietf.org/html/rfc2617), signed against a user's username and password. Basic authentication is generally only appropriate for testing.

If successfully authenticated, BasicAuthentication provides the following credentials.

* request.user will be a Django User instance.
* request.auth will be None.

Unauthenticated responses that are denied permission will result in an HTTP 401 Unauthorized response with an appropriate WWW-Authenticate header

**Note:** If you use BasicAuthentication in production you must ensure that your API is only available over https. You should also ensure that your API clients will always re-request the username and password at login, and will never store those details to persistent storage.

## [**TokenAuthentication:**](https://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication) This authentication scheme uses a simple token-based HTTP Authentication scheme. Token authentication is appropriate for client-server setups, such as native desktop and mobile clients.

[**SessionAuthentication:**](https://www.django-rest-framework.org/api-guide/authentication/#sessionauthentication) This authentication scheme uses Django's default session backend for authentication. Session authentication is appropriate for AJAX clients that are running in the same session context as your website.

Views.py

from django.shortcuts import render  
from django.http import HttpResponse, JsonResponse  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.decorators import api\_view  
from rest\_framework.response import Response  
from rest\_framework import status  
from rest\_framework.views import APIView  
from rest\_framework import generics  
from rest\_framework import mixins  
from rest\_framework.authentication import BasicAuthentication,SessionAuthentication, TokenAuthentication  
from rest\_framework.permissions import IsAuthenticated  
  
# Create your views here.  
  
class GenericApiViews(generics.GenericAPIView, mixins.ListModelMixin, mixins.CreateModelMixin, mixins.UpdateModelMixin,  
 mixins.RetrieveModelMixin,mixins.DestroyModelMixin):  
  
 serializer\_class = ArticleSerializer  
 queryset = Article.objects.all()  
  
 lookup\_field = 'id'  
 *#authentication\_classes = [SessionAuthentication, BasicAuthentication]*  
authentication\_classes = [TokenAuthentication]  
 permission\_classes = [IsAuthenticated]  
  
  
 def get(self, request,id=None):  
 if id:  
 return self.retrieve(request)  
 else:  
 return self.list(request)  
  
 def post(self,request):  
 return self.create(request)  
  
 def put(self, request, id=None):  
 return self.update(request,id)  
  
 def delete(self, request, id):  
 return self.destroy(request, id)

settings.py

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'rest\_framework',  
 'rest\_app1',  
 'rest\_framework.authtoken',  
]

* **Viewsets & Routers**

Django rest framework allows us to combine a set of related views in a single class that we can call viewsets.

View sets are basically class-based views that does not provide any method handler such as get or post.

Views.py

from django.shortcuts import render  
from django.http import HttpResponse, JsonResponse  
from rest\_framework.parsers import JSONParser  
from .models import Article  
from .serializer import ArticleSerializer  
from django.views.decorators.csrf import csrf\_exempt  
from rest\_framework.decorators import api\_view  
from rest\_framework.response import Response  
from rest\_framework import status  
from rest\_framework.views import APIView  
from rest\_framework import generics  
from rest\_framework import mixins  
from rest\_framework.authentication import BasicAuthentication,SessionAuthentication, TokenAuthentication  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework import viewsets  
from django.shortcuts import get\_object\_or\_404  
  
# Create your views here.  
  
class ArticleViewSet(viewsets.ViewSet):  
  
 def list(self, request):  
 article = Article.objects.all()  
 serializer = ArticleSerializer(article, many=True)  
 return Response(serializer.data)  
  
 def create(self, request):  
 serializer = ArticleSerializer(data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return JsonResponse(serializer.data, status=status.HTTP\_201\_CREATED)  
 return JsonResponse(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)  
  
 def retrieve(self, request, pk=None):  
 queryset = Article.objects.all()  
 article = get\_object\_or\_404(queryset, pk=pk)  
 serializer = ArticleSerializer(article)  
 return Response(serializer.data)  
  
 def update(self, request, pk=None):  
 article = Article.get\_objects(pk=pk)  
 serializer = ArticleSerializer(article, data=request.data)  
  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
  
 return JsonResponse(serializer.errors, status=status.HTTP\_404\_NOT\_FOUND)

Urls.py

urlpatterns = [  
 path('viewset/', include(router.urls)),  
 path('viewset/<int:pk>/', include(router.urls)),]

* **Generic Viewsets**

Views.py

*# Create your views here.*

class ArticleViewSet(viewsets.GenericViewSet, mixins.ListModelMixin, mixins.CreateModelMixin, mixins.UpdateModelMixin,  
 mixins.RetrieveModelMixin, mixins.DestroyModelMixin):  
 serializer\_class = ArticleSerializer  
 queryset = Article.objects.all()

* **Model Viewsts**

Views.py

*# Create your views here.*  
*class* ArticleViewSet(viewsets.ModelViewSet):  
 serializer\_class = ArticleSerializer  
 queryset = Article.objects.all()