# Request

Cambian

Request.POST -> request.data (incluye archivo e imágenes enviados)

Request.GET -> query\_params

Dos variables dentro del request:

-accepted\_rendered

- accepted\_media\_type

# Return enpoints

From rest\_framework.response import Response

Siempre van a retornar un objeto Response

# serializador

From rest\_framework import serializer

El serializador es una clase que representan a un modelo y en específico que campos son los que se van a serializar, esta extiende de “serializer.ModelSerializer”

## Meta

Tiene una clase “Meta”

* model=ClaseModelo obvio
* exclude=(”campo1”, ”campo2”,…) tupla de campos que no se van a serializar
* fields=”\_\_all\_\_” o [ ”campo1”, ”campo2”,…] este o exclude, no ambos

# listas

Para las listas, se declara una clase que herede de “ViewSet”

* model=ClaseModelo obvio
* serielizerClass= clase serializador

luego un método para definir lo que pasa si se llama al get de este modelo y retornar una lista de las instancias disponibles ya serializadas

def list(self,request):

1ro se obtiene la lista usando al self.model y el request

lista=self.model.objects.all()

2do se serializa la lista usando a self.serielizerClass

listaSerializada=self.serielizerClass(lista, many=True) // de momento lo ultimo lo ignoro

3ro se retorna esta lista en una respuesta (Response)

Return Response(listaSerializada.data,status=200)

# APPS INSTALLES

Como buena práctica se puede dividir al APPS INSTALLES en el BASE\_APPS

, LOCAL\_APPS, THIRD\_APPS

INSTALLED\_APPS = BASE\_APPS + LOCAL\_APPS + THIRD\_APPS

# Settings dividido

El archivo settings también se puede sustituir por una carpeta y en esta poner a base, local y production .py

En base esta la mayor parte de la configuración y luego en local y production esta la configuración de las carpetas, y las bases de datos:

DEBUG, ALLOWED\_HOSTS, DATABASES, STATIC, MEDIA

# serializar una sola instancia

Si se va a serializar una sola instancia es

objetoSerializado=ClaseSerializadora(objetoSinSerializar)

# serializar varias instancias

Y si son varias se le agrega (…, many=True) como argumento a la llamada al constructor

listaSerializada=ClaseSerializadora(listaSinSerializar,many=True)

# APIView

From rest\_framework.views import APIView

Para responder a un GET o POST se usa una clase que herede de APIView y defina los métodos

Get o post (self,request,parametros) y retorne un Response

from rest\_framework.decorators import api\_view

# funciones apis (@api\_view)

Se puede ligar a funciones apis ya creadas, para ello se les agrega el decorador @api\_view([‘POST’, o+ ‘’GET’ …]), aunque se defina dentro del decorador el o los tipos de método, hay que comprobarlo internamente con el request.Metod==’GET…’

# Json a instancia

Para hacer el efecto contrario, ósea se obtiene del request un json y este se serializa para luego obtener el objeto

1ro- Se le pasa el request.data al argumento data de constructor de la case serializadora

objetoSerializador=ClaseSerializadora(data= request.data)

2do- luego se hace una mínima validación dond ese comprueba que el json obtenido almenos cumple con la estructura deseada

esValido= objetoSerializador.is\_valid()

3ro- en caso de ser valido se manada a realizas la conversión de los datos con el método

objetoSerializador.save()

ese método también al mismo tiempo guarda la información en la base de datos

4to- se pueden ver los errores en formato json, estos se almacenan en caso de ocurrir dentro del atributo

erroresJSON= objetoSerializador.errors

Este puede ser enviado también en el data de un Response

# update

Para el update se utiliza a PUT

1ro- instancia\_anterior=filtrar y obtener

2do- Variable\_Serializador=ClaseSerializador(instancia\_anterior,data=request.data)

# DELETE

Para el delete se utiliza DELETE

# status

from rest\_framework import status

de hay se pueden obtener la mayoría de los códigos de status que se utilizan en el response

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

# serializadores personalizados

Se pueden crear serializadores que no estén basados en modelos, y los campos disponibles para estos son los mismos que están disponibles para un modelo

# Validación serializer

Para la validación de los campos de un serializer, primero por cada campo se busca que exista un método:

def validate\_nombreCampo(self,value):

if validar:

raise serializers.ValidationError(“descripion”)

return value # si todo esta bien

y luego se llama si no hay errores a

def validate(self,data):

valorAEvaluar=data[‘’NombreAtributo”]

#lo validamos y disparamos el ValidationError

Return data #si todo esta bien

Para casos donde en se quiere pasar mas datos a la validación (antes de realizarse) o simplemente poder acceder a los a todos los datos existentes desde un método de validación, se le pasas al constructor del Serilizador otro argumento context= dic (puede ser el mismo data)

1ro- Variable\_serializador=ClaseSerilizador(data=data,context=data u otro dic)

2do- Luego

def validate\_nombreCampo(self,value):

dato=self.context[“dato”]

if validar:

raise serializers.ValidationError(“descripion”)

return value # si todo esta bien

# sobrescribir save

Para sobreescribir la manera en que se guarda una instancia en la base de datos (solo al crear)

Def créate(self,validate\_data):

Instancia=ClaseModelo(\*\*validate\_data)

Return instancia

# Modelo a partir de dic data

Para crear una instancia de un modelo rápido teniendo un dic con los valores de sus atributos

Instancia=ClaseModelo(\*\*dic\_conLosAtributos)

# Sobrescribir editar

Para sobrescribir el método de salvar al editar (solo al editar)

Def update(self,instance,validate\_data):

Instance=super().update(instance,validate\_data) # si se quiere aprovecha rla edición automática de los campos

Instance.Atributo=validate\_data.get(“Atributo”, Instance.Atributo) # medoto del dic para que retorne un valor por defecto

Instance.save()

Return Instance

# Representación de solo campos seleccionados

Para de un filtro a modelos obtener solo un objeto con solo ciertos campos

Instancia\_reducida=Clase\_Modelo.objects.filter(….).values(“Atributo1”,”Atributo2”)

# to\_represntation

Para manipular la forma en que se ve una instancia (por ejemplo para quitar la contraseña de la vista)

Def to\_represntation(self,instance):

Dato=instance[“atributo”]

Return diccionario

# ListAPIView

from rest\_framework import generics

Para las listas usar a vistas (clases) que hereden de generics. ListAPIView

Class VistaLista(generics. ListAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# mostrar datos de campos id serializer

## forma completa

Cuando se quiere en ves de mostrar un atributo que sea llave como el id, sino mostrar toda la información de esta entidad, hay que declarar ese atributo en el serializador y darle como valor el serializador que representa a ese campo.

Class ClaseSerializador(serializer.ModelSerializer):

atributoLlave=ClaseSerializadoraAtributo()

## str

Si lo que se quiere es mostrar su forma str por defecto (lo del atributo llave), se utiliza el tipo de campo se serializador serializer.StringRelatedField()

Class ClaseSerializador(serializer.ModelSerializer):

atributoLlave= serializer.StringRelatedField()

# CreateAPIView

Para la vista de crear usar a vistas (clases) que hereden de generics. CreateAPIView

Class VistaCrear(generics.CreateAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# RetrieveAPIView

Para la vista de ver una entidad usar a vistas (clases) que hereden de generics. RetrieveAPIView

Class VistaVer(generics. RetrieveAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# DestroyAPIView

Para la vista de eliminar una entidad usar a vistas (clases) que hereden de generics.DestroyAPIView

Class VistaEliminar(generics.DestroyAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# UpdateAPIView

Para la vista de editar una entidad usar a vistas (clases) que hereden de generics. UpdateAPIView

Class VistaEditar(generics. UpdateAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

de manera adicional la vista editar admite el método PATCH (pasándole también el id) y lo que hace es que retorna la información (tipo get)

# ListCreateAPIView

Para la vista de crear y listar usar a vistas (clases) que hereden de generics. ListCreateAPIView

Class VistaListarCrear(generics. ListCreateAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# RetrieveUpdateDestroyAPIView

Para la vista de eliminar, editar y ver una entidad usar a vistas (clases) que hereden de generics. RetrieveUpdateDestroyAPIView

Class VistaEliminar(generics. RetrieveUpdateDestroyAPIView):

serializer\_class = ClaseSerializadora

def get\_queryset(self):  
 return self.serializer\_class.Meta.model.objects.all()

# unir el crud completo ViewSet

https://www.django-rest-framework.org/api-guide/viewsets/

Para unir el crud completo

class ClaseViewSet(viewsets.ViewSet):

def list(self, request):

pass

def create(self, request):

pass

def retrieve(self, request, pk=None):

pass

def update(self, request, pk=None):

pass

def partial\_update(self, request, pk=None):

pass

def destroy(self, request, pk=None):

pass

# routers

<https://www.django-rest-framework.org/api-guide/routers/>

Los ViewSet se registran (url) mediante routers

Esos se declaran en algún archivo aparte, seria buena costumbre llamarlo routers.py

from rest\_framework import routers

router = routers.SimpleRouter()

router.register(r'users', UserViewSet,basename=”suurl”)

router.register(r'accounts', AccountViewSet)

urlpatterns = router.urls

# JWT

## Authtoken

### Login

Para tokens trae por defecto a la aplicación 'rest\_framework.authtoken'

http://www.developerpe.com/programaci%C3%B3n/autenticacion-en-django-rest-framework/

from rest\_framework.authtoken.models import Token

from rest\_framework.authtoken.views import ObtainAuthToken

from rest\_framework import serializers

from django.contrib.sessions.models import Session

from datetime import datetime

class UserTokenSerializer(serializers.ModelSerializer):

    class Meta:

        model = User

        fields = ('username','email','name','last\_name')

poner en Apps/user/views.py

class Login2(ObtainAuthToken):

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

        reinciar\_token=True

        loguin\_serielizer=self.serializer\_class(data=request.data,context={'request':request})

        if loguin\_serielizer.is\_valid():

            user=loguin\_serielizer.validated\_data["user"]

            if user.is\_active:

                token,created=Token.objects.get\_or\_create(user=user)

                user\_serializer=UserTokenSerializer(user)

                if created:

                    return Response({

                        'token':token.key

                        ,'user':user\_serializer.data,

                        'message':'Incio de sesion exitoso'

                    },status=status.HTTP\_201\_CREATED)

                else:

                    if reinciar\_token:

                        all\_sessions=Session.objects.filter(expire\_date\_\_gte=datetime.now())

                        if all\_sessions.exists():

                            for session in all\_sessions:

                                session\_data=session.get\_decoded()

                                if user.id==int(session\_data.get('\_auth\_user\_id')):

                                    session.delete()

                        token=Token.objects.create(user=user)

                        return Response({

                            'token':token.key

                            ,'user':user\_serializer.data,

                            'message':'Incio de sesion exitoso'

                        },status=status.HTTP\_201\_CREATED)

                    else:

                        token.delete()

                        return Response({

                            'error':'Ya se ha iniciado sesion con este usuario'

                        },status=status.HTTP\_409\_CONFLICT)

            else:

                return Response({

                    'error':'Usuario no activo'

                    },status=status.HTTP\_401\_UNAUTHORIZED)

        else:

                return Response({

                    'error':'Nombre de usuario o contraseña incorrecto'

                    },status=status.HTTP\_400\_BAD\_REQUEST)

### Logout

poner en Apps/user/views.py

class Logout2(APIView):

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

        token=request.GET.get('token')

        token=Token.objects.filter(key=token).first()

        if token:

            user=token.user

            all\_sessions=Session.objects.filter(expire\_date\_\_gte=datetime.now())

            if all\_sessions.exists():

                for session in all\_sessions:

                    session\_data=session.get\_decoded()

                    if user.id==int(session\_data.get('\_auth\_user\_id')):

                        session.delete()

            token.delete()

            return Response({

                            'token\_message':'token eliminado'

                            ,'session\_message':'sesiones de usuario elminadas',

                        },status=status.HTTP\_200\_OK)

        return Response({

                    'error':'no se han enctroda las credenciales correctas'

                    },status=status.HTTP\_400\_BAD\_REQUEST)

### Petición Frontend

En el header

Authorization: Token 401f7ac837da42b97f613d789819ff93537bee6a

### Tiempo de vida token

poner en Apps/user/ authentication.py

from datetime import timedelta

from django.utils import timezone

from django.conf import settings

from rest\_framework.response import Response

from rest\_framework.authentication import TokenAuthentication

from rest\_framework.exceptions import AuthenticationFailed

class ExpiringTokenAuthentication(TokenAuthentication):

    def expires\_in(self,token):

        # return left time of token

        time\_elapsed = timezone.now() - token.created

        left\_time = timedelta(seconds=settings.TOKEN\_EXPIRED\_AFTER\_SECONDS) - time\_elapsed

        return left\_time

    def is\_token\_expired(self,token):

        # return True if token is alive or False if token is expired

        return self.expires\_in(token) < timedelta(seconds = 0)

    def token\_expire\_handler(self,token):

        """

        Return:

            \* is\_expire     : True if token is alive, False if token is expired

            \* token         : New token or actual token

        """

        is\_expire = self.is\_token\_expired(token)

        if is\_expire:

            user = token.user

            token.delete()

            token = self.get\_model().objects.create(user=user)

        return token

    def authenticate\_credentials(self,key):

        """

        Return:

            \* user      : Instance User that sended request

            \* token     : New Token or actual token for user

            \* message   : Error message

            \* expired   : True if token is alive or False if token is expired

        """

        user = None

        try:

            token = self.get\_model().objects.select\_related('user').get(key=key)

            token = self.token\_expire\_handler(token)

            user = token.user

        except self.get\_model().DoesNotExist:

            pass

        return user

### Validar Vista

#### Mixin

Hay que crear los métodos que se sobrescribirán del View a modo de interfaz

poner en Apps/user/ authentication\_mixins.py

from rest\_framework import status, authentication, exceptions

from rest\_framework.response import Response

from rest\_framework.renderers import JSONRenderer

from rest\_framework.authentication import get\_authorization\_header

from apps.users.authentication import ExpiringTokenAuthentication

class Authentication(authentication.BaseAuthentication):

    user = None

    def get\_user(self,request):

        """

        Return:

            \* user      : User Instance or

            \* message   : Error Message or

            \* None      : Corrup Token

        """

        token = get\_authorization\_header(request).split()

        if token:

            try:

                token = token[1].decode()

            except:

                return None

            token\_expire = ExpiringTokenAuthentication()

            user = token\_expire.authenticate\_credentials(token)

            if user != None:

                self.user = user

                return user

        return None

    def authenticate(self, request):

        self.get\_user(request)

        if self.user is None:

            raise exceptions.AuthenticationFailed('No se han enviado las credenciales.')

        return (self.user, 1)

#### Vista

En este caso se le llamo Authentication

##### Como interfaz

class ProductViewSet(Authentication,viewsets.ModelViewSet):

##### authentication\_classes

o también en

class ProductViewSet(viewsets.ModelViewSet):

    authentication\_classes=[Authentication]

##### funciones ( @authentication\_classes )

o en funciones

@api\_view(['GET'])

@authentication\_classes([SessionAuthentication, BasicAuthentication, Authentication])

@permission\_classes([IsAuthenticated])

def example\_view(request, format=None):

##### Global

En el settings, para poder realizar esto la clase debe de heredar de authentication.BaseAuthentication y a su ves sobrescribir el método

def authenticate(self, request):

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.BasicAuthentication',

'rest\_framework.authentication.SessionAuthentication',

]

}

# cors

Algunos servidores tienen una política de seguridad que se tiene que asegurar de comprobar de donde se originan las peticiones, para eso hay que usar django-cors-headers

<https://pypi.org/project/django-cors-headers/>

* pip install django-cors-headers

INSTALLED\_APPS = [

...,

"corsheaders",

...,

]

MIDDLEWARE = [

...,

"corsheaders.middleware.CorsMiddleware",

"django.middleware.common.CommonMiddleware",#encima de este

...,

]

Direcciones permitidas

CORS\_ALLOWED\_ORIGINS = [

"https://example.com",

"https://sub.example.com",

"http://localhost:8080",

"http://127.0.0.1:9000",

]

#Todos los siguientes son opcionales

CORS\_ALLOWED\_ORIGIN\_REGEXES = [

r"^https://\w+\.example\.com$",

]

CORS\_ALLOW\_METHODS = (

"DELETE",

"GET",

"OPTIONS",

"PATCH",

"POST",

"PUT",

)

CORS\_ALLOW\_HEADERS = (

"accept",

"authorization",

"content-type",

"user-agent",

"x-csrftoken",

"x-requested-with",

)