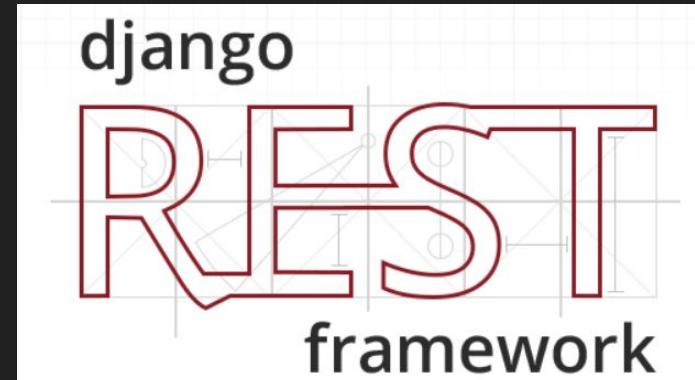


API REST Completa



Roteiro

- Models
- Serializers
- Views: por meio de Viewsets
- URLs: rotas
- Auto Docs: Swagger
- Paginação
- Filtros: django-filters
- Throttling: Limites de chamada
- Autenticação: Signup & Signin
- Autorização: Permissions
- CORS
- Upload

Models: Estado e Cidade

```
● ● ●  
1 from django.db import models  
2  
3  
4 class Estado(models.Model):  
5     nome = models.CharField(max_length=128, null=False, blank=False)  
6     sigla = models.CharField(max_length=2, null=False, blank=False)  
7  
8  
9 class Cidade(models.Model):  
10    nome = models.CharField(max_length=255, null=False, blank=False)  
11    estado = models.ForeignKey(  
12        Estado, on_delete=models.CASCADE, related_name='cidades')  
13
```

Serializers Básicos

```
from rest_framework.serializers import ModelSerializer
```

```
● ● ●  
1 class CidadeSerializer(ModelSerializer):  
2     class Meta:  
3         model = Cidade  
4         fields = '__all__'  
5  
6  
7 class EstadoSerializer(ModelSerializer):  
8     cidades = CidadeSerializer(many=True, read_only=True)  
9  
10    class Meta:  
11        model = Estado  
12        fields = ['id', 'nome', 'sigla', 'cidades']  
13
```

Serializers: Construtor e Atributos

```
1 class SeuModeloSerializer(serializers.ModelSerializer):
2
3     def __init__(self, instance=None, data=None, **kwargs):
4         """
5             Construtor da classe Serializer.
6
7             :param instance: Instância do modelo a ser serializada (opcional).
8             :param data: Dados a serem desserializados e validados (opcional).
9             :param kwargs: Outros argumentos opcionais.
10            """
11
12     # Chamada ao construtor da classe pai
13     super().__init__(instance=instance, data=data, **kwargs)
14
15     # Atributos do construtor
16     self.instance = instance # Instância do modelo a ser serializada
17     self.initial_data = data # Dados a serem desserializados e validados
18     self.partial = kwargs.pop('partial', False) # Indica se a atualização é parcial
19
20     # Contexto do serializer, útil para personalização e acesso a informações adicionais
21     self.context = kwargs.pop('context', {})
```

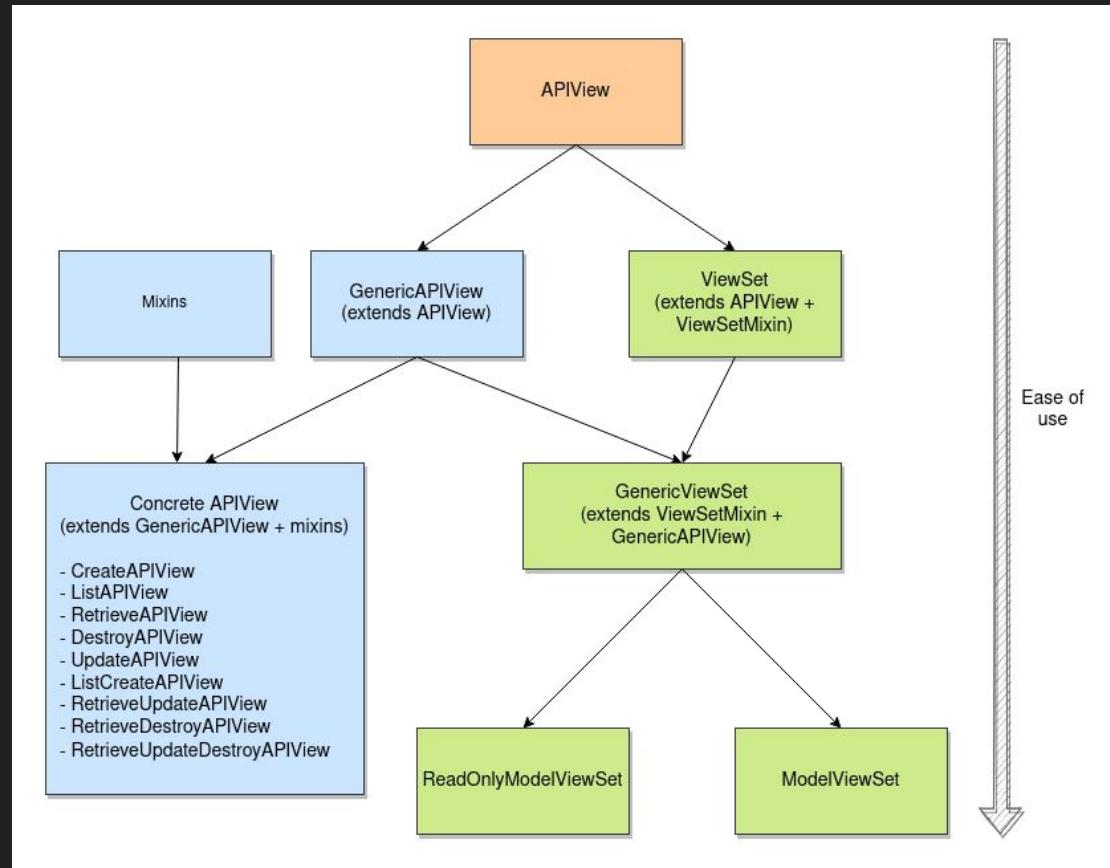
Serializers: Principais Métodos

```
4 class SeuModeloSerializer(serializers.ModelSerializer):
5     class Meta:
6         model = SeuModelo
7         fields = ['id', 'campo1', 'campo2', 'campo3']
8
9     def validate_campo1(self, value):
10        """
11            Valida o campo1.
12
13            Este método é chamado durante a validação do serializer.
14            Retorna o valor válido se a validação for bem-sucedida.
15            Caso contrário, lança uma exceção de validação.
16        """
17
18        if not value:
19            raise serializers.ValidationError(
20                "campo1 não pode estar vazio.")
21        return value
22
23    def validate(self, data):
24        """
25            Valida os dados do serializer.
26
27            Este método é chamado durante a validação do serializer,
28            após todos os campos individuais terem sido validados.
29            Retorna os dados válidos se a validação for bem-sucedida.
30            Caso contrário, lança uma exceção de validação.
31        """
32
33        if data['campo2'] == 'valor_invalido':
34            raise serializers.ValidationError(
35                "campo2 não pode ter o valor 'valor_invalido'.")
36
37        return data
```

```
4 class SeuModeloSerializer(serializers.ModelSerializer):
5
6     def create(self, validated_data):
7         """
8             Cria uma nova instância do modelo.
9
10            Este método é chamado quando o serializer
11            é usado para criar uma nova instância do modelo.
12            Retorna a instância do modelo criada
13            com base nos dados validados.
14        """
15
16        return SeuModelo.objects.create(**validated_data)
17
18    def update(self, instance, validated_data):
19        """
20            Atualiza uma instância existente do modelo.
21
22            Este método é chamado quando o serializer é usado
23            para atualizar uma instância existente do modelo.
24            Retorna a instância do modelo atualizada
25            com base nos dados validados.
26        """
27
28        instance.campo1 = validated_data.get('campo1', instance.campo1)
29        instance.save()
30
31        return instance
```

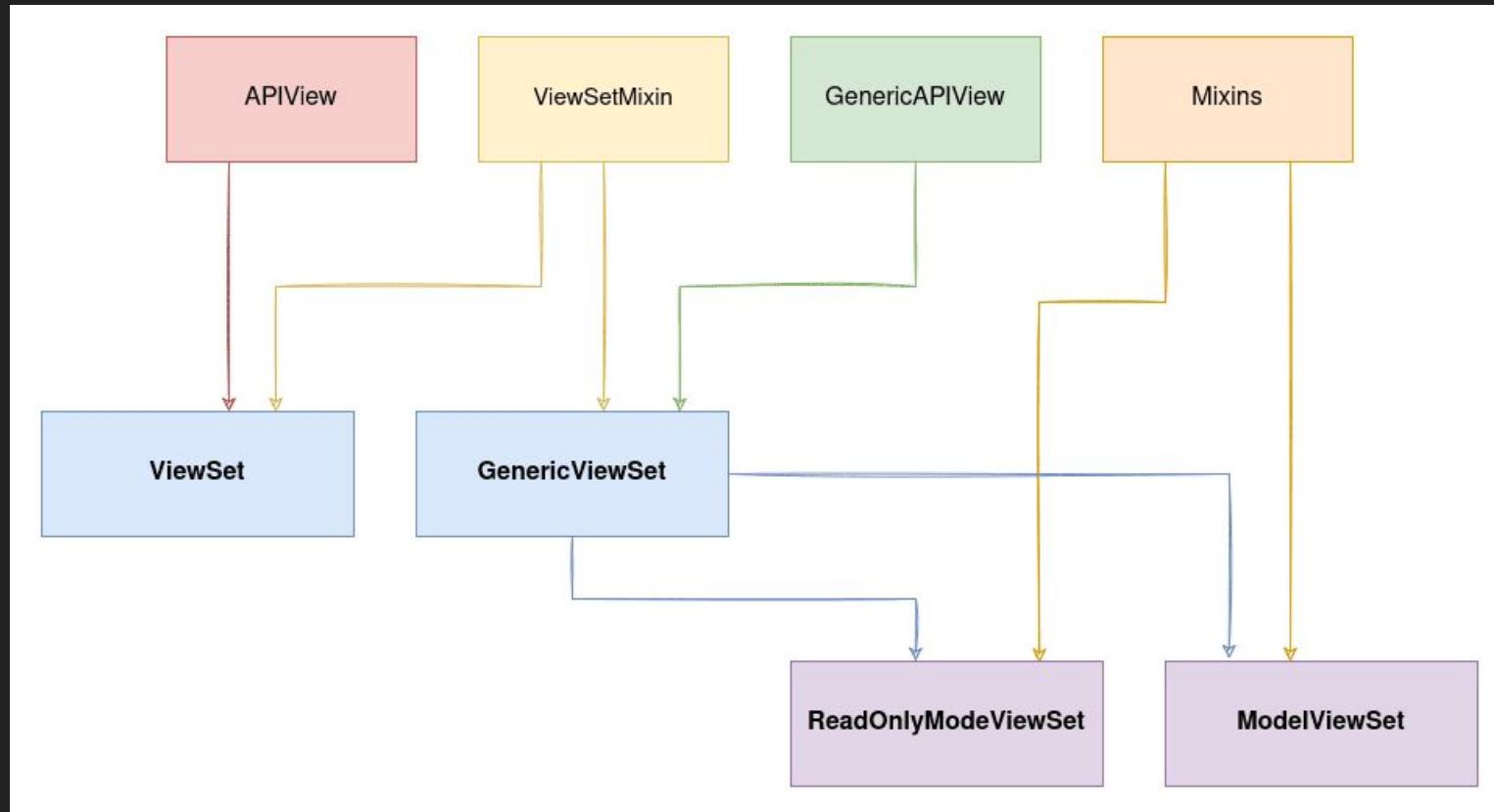
Hierarquia de Classes View do DRF

Consulte: https://github.com/encode/django-rest-framework/tree/master/rest_framework



Hierarquia de Classes View do DRF

Consulte: <https://testdriven.io/blog/drf-views-part-3/>



DRF APIView: handler e action

Method	List / Detail	Action
<code>post</code>	List	<code>create</code>
<code>get</code>	List	<code>list</code>
<code>get</code>	Detail	<code>retrieve</code>
<code>put</code>	Detail	<code>update</code>
<code>patch</code>	Detail	<code>partial_update</code>
<code>delete</code>	Detail	<code>destroy</code>

Views (vide Viewsets)

```
● ● ●  
1 from rest_framework import viewsets  
2 from .models import Cidade, Estado  
3 from .serializers import CidadeSerializer, EstadoSerializer  
4  
5  
6 class EstadoViewSet(viewsets.ModelViewSet):  
7     queryset = Estado.objects.all()  
8     serializer_class = EstadoSerializer  
9  
10  
11 class CidadeViewSet(viewsets.ModelViewSet):  
12     queryset = Cidade.objects.all()  
13     serializer_class = CidadeSerializer
```

Viewset: sobrescrita de métodos

```
class SeuModeloViewSet(viewsets.ModelViewSet):
    serializer_class = SeuModeloSerializer

    def get_queryset(self):
        queryset = SeuModelo.objects.all()
        # Adicione lógica de filtragem personalizada, por exemplo:
        if self.request.user.is_authenticated:
            queryset = queryset.filter(owner=self.request.user)
        return queryset
```

Viewset: sobrescrita de métodos

```
class SeuModeloViewSet(viewsets.ModelViewSet):
    serializer_class = SeuModeloSerializer

    def get_object(self):
        pk = self.kwargs.get('pk')
        try:
            instance = SeuModelo.objects.get(pk=pk)
        except SeuModelo.DoesNotExist:
            raise NotFound("Objeto não encontrado")
        # Adicione lógica de controle de permissão personalizada, por exemplo:
        if not instance.public and instance.owner != self.request.user:
            raise PermissionDenied("Você não tem permissão para acessar este objeto")
        return instance
```

ViewSet: sobrescrita de métodos

```
class SeuModeloViewSet(viewsets.ModelViewSet):  
    queryset = SeuModelo.objects.all()  
    serializer_class = SeuModeloSerializer  
  
    def perform_create(self, serializer):  
        serializer.save(owner=self.request.user)
```

Viewset: sobrescrita de métodos

```
class SeuModeloViewSet(viewsets.ModelViewSet):  
    queryset = SeuModelo.objects.all()  
  
    def get_serializer_class(self):  
        if self.action == 'list':  
            return SerializerParaGetAll  
        elif self.action == 'retrieve':  
            return SerializerParaGetOne  
        return super().get_serializer_class()
```

Viewset: todos os métodos que podem ser sobreescritos

`get_queryset(self)`: Retorna o conjunto de consultas que será usado para buscar os objetos. Você pode querer sobreescriver este método para filtrar os resultados com base nas permissões do usuário ou adicionar lógica personalizada de filtragem.

`get_object(self)`: Retorna um objeto específico com base no parâmetro de URL `pk`. Você pode querer sobreescriver este método para adicionar lógica personalizada de obtenção de objetos ou controle de permissão.

`perform_create(self, serializer)`: Adiciona lógica personalizada antes de salvar um novo objeto criado. Por exemplo, você pode querer definir automaticamente alguns campos do objeto com base nos dados da solicitação.

`perform_update(self, serializer)`: Adiciona lógica personalizada antes de salvar um objeto atualizado. Por exemplo, você pode querer registrar as alterações feitas no objeto.

`perform_destroy(self, instance)`: Adiciona lógica personalizada antes de excluir um objeto. Por exemplo, você pode querer verificar se o objeto pode ser excluído com base em algumas condições específicas.

`get_serializer_class(self)`: Retorna a classe do serializador a ser usada com base na ação que está sendo executada. Você pode querer sobreescriver este método se precisar usar diferentes serializadores para diferentes ações.

`list(self, request), retrieve(self, request, pk=None), create(self, request), update(self, request, pk=None), partial_update(self, request, pk=None), destroy(self, request, pk=None)`: Esses são os métodos principais que correspondem às ações CRUD da API. Você pode querer sobreescriver qualquer um desses métodos para adicionar lógica personalizada, controle de permissão ou validação de dados.

Views: Custom Action

```
● ● ●  
1 class EstadoViewSet(viewsets.ModelViewSet):  
2     queryset = Estado.objects.all()  
3     serializer_class = EstadoSerializer  
4     pagination_class = CustomPagination  
5     filterset_class = EstadoFilter  
6     permission_classes = [  
7         permissions.isAuthenticatedOrReadOnly, IsOwnerOrReadOnly]  
8  
9     def get_serializer_class(self):  
10        return self.serializer_class if self.action not in ['upload_foto'] else None  
11  
12    @decorators.action(url_path='upload_foto', detail=True, methods=[HTTPMethod.PUT])  
13    def upload_foto():  
14        pass
```

Rotas com *sub-recursos*:
<https://github.com/alanjds/drf-nested-routers>

Rotas (urls)

```
● ● ●
1 from django.urls import path, include
2 from rest_framework.routers import DefaultRouter
3 from rest_framework_nested import routers
4 from .viewsets import EstadoViewSet, CidadeViewSet
5
6 # Roteador raiz para /estados
7 router = DefaultRouter()
8 router.register(r'estados', EstadoViewSet)
9
10 estado_router = routers.NestedDefaultRouter(
11     router, r'estados', lookup='estado')
12 estado_router.register(r'cidades', CidadeViewSet,
13                         basename='estado-cidades')
14
15 urlpatterns = [
16     path('', include(router.urls)),
17     path('', include(estado_router.urls)),
18 ]
```

Swagger Docs

```
pip install drf-yasg
```

Swagger Docs

```
● ● ●
1 from django.urls import path, include
2 from rest_framework import permissions
3 from drf_yasg.views import get_schema_view
4 from drf_yasg import openapi
5
6 schema_view = get_schema_view(
7     openapi.Info(
8         title="Cidades API Documentation",
9         default_version='v1',
10        description="API para Cidades e Estados",
11        terms_of_service="https://www.google.com/policies/terms/",
12        contact=openapi.Contact(email="rogerio410@gmail.com"),
13        license=openapi.License(name="BSD License"),
14    ),
15    public=True,
16    permission_classes=(permissions.AllowAny,),
17 )
18
19
20 urlpatterns = [
21     path('swagger/', schema_view.with_ui('swagger',
22             cache_timeout=0), name='schema-swagger-ui'),
23     path('redoc/', schema_view.with_ui('redoc',
24             cache_timeout=0), name='schema-redoc'),
25     path('', include('core.urls')),
26 ]
27
```

Swagger Auto Docs

The screenshot shows a web browser window displaying the Swagger Auto Docs interface at `localhost:8000/swagger/`. The page lists several API endpoints for managing states and cities, each with a method, URL, and a dropdown menu for actions like 'Copy' and 'Edit'.

- GET /estados/** (estados_list) - A blue button.
- POST /estados/** (estados_create) - A green button.
- GET /estados/{estado_pk}/cidades/** (estados_cidades_list) - A blue button.
- POST /estados/{estado_pk}/cidades/** (estados_cidades_create) - A green button.
- GET /estados/{estado_pk}/cidades/{id}/** (estados_cidades_read) - A blue button.
- PUT /estados/{estado_pk}/cidades/{id}/** (estados_cidades_update) - An orange button.
- PATCH /estados/{estado_pk}/cidades/{id}/** (estados_cidades_partial_update) - A green button.
- DELETE /estados/{estado_pk}/cidades/{id}/** (estados_cidades_delete) - A red button.

Redoc Auto Docs

localhost:8000/redoc/#tag/estados/operation/estados_list

Search...

estados

- GET** estados_list
- POST** estados_create
- GET** estados_cidades_list
- POST** estados_cidades_create
- GET** estados_cidades_read
- PUT** estados_cidades_update
- PATCH** estados_cidades_partial_up...
- DEL** estados_cidades_delete

Responses

✓ 200

RESPONSE SCHEMA: application/json

Array [

- id integer (ID)
- nome string (Nome) [1 .. 128] characters
- sigla string (Sigla) [1 .. 2] characters
- cidades > Array of objects (Cidade)

]

API docs by Redocly

Paginação

```
● ● ●  
1 from rest_framework import viewsets, pagination  
2  
3  
4 class CustomPagination(pagination.PageNumberPagination):  
5     page_size = 10  
6     max_page_size = 100  
7     page_size_query_param = 'take'  
8     page_query_param = 'page'  
9  
10  
11 class EstadoViewSet(viewsets.ModelViewSet):  
12     queryset = Estado.objects.all()  
13     serializer_class = EstadoSerializer  
14     pagination_class = CustomPagination  
15
```

Paginação

GET /estados/

Parameters

Name	Description
page <small>integer (query)</small>	A page number within the paginated result set. page
take <small>integer (query)</small>	Number of results to return per page. take

Execute

Filtros de Consulta (query)

```
pip install django-filter
```

https://django-filter.readthedocs.io/en/stable/guide/rest_framework.html

Filtros

```
● ● ●  
1 from django_filters import rest_framework as filters  
2 from .models import Estado, Cidade  
3  
4  
5 class EstadoFilter(filters.FilterSet):  
6     nome = filters.CharFilter(  
7         field_name='nome', lookup_expr='icontains')  
8  
9     class Meta:  
10        model = Estado  
11        fields = ['nome']  
12  
13  
14 class CidadeFilter(filters.FilterSet):  
15     nome = filters.CharFilter(  
16         field_name='nome', lookup_expr='icontains')  
17  
18     class Meta:  
19        model = Cidade  
20        fields = ['nome']
```

Aplicando Filtros à View

```
● ● ●  
1 from .filters import EstadoFilter, CidadeFilter  
2  
3  
4 class EstadoViewSet(viewsets.ModelViewSet):  
5     queryset = Estado.objects.all()  
6     serializer_class = EstadoSerializer  
7     pagination_class = CustomPagination  
8     filterset_class = EstadoFilter  
9  
10  
11 class CidadeViewSet(viewsets.ModelViewSet):  
12     queryset = Cidade.objects.all()  
13     serializer_class = CidadeSerializer  
14     pagination_class = CustomPagination  
15     filterset_class = CidadeFilter  
16
```

Throttling

<https://www.djangoproject.org/api-guide/throttling/>

Throttling:

(settings.py)

```
1
2 REST_FRAMEWORK = {
3     'DEFAULT_FILTER_BACKENDS': (
4         'django_filters.rest_framework.DjangoFilterBackend',
5     ),
6     'DEFAULT_THROTTLE_CLASSES': [
7         # Throttling para usuários anônimos
8         'rest_framework.throttling.AnonRateThrottle',
9         # Throttling para usuários autenticados
10        'rest_framework.throttling.UserRateThrottle',
11    ],
12    'DEFAULT_THROTTLE_RATES': {
13        'anon': '100/day',
14        'user': '1000/day',
15    },
16 }
```

Autenticação & Autorização

Signup

Signin (Token e JWT Token)

Autenticação: Signup e Signin(token)

1. DRF Token Auth App

a. `'rest_framework.authtoken',`

i. `migrate (model Token)`

2. Serializer

3. View

4. Rotas (urls)

DRF AuthToken App

```
● ● ●  
1 INSTALLED_APPS = [  
2     'django.contrib.admin',  
3     'django.contrib.auth',  
4     'django.contrib.contenttypes',  
5     'django.contrib.sessions',  
6     'django.contrib.messages',  
7     'django.contrib.staticfiles',  
8     'rest_framework',  
9     'rest_framework.authtoken',  
10    'drf_yasg',  
11    'django_filters',  
12    'core',  
13 ]
```

Signup: Serializer

```
● ● ●  
1 from django.contrib.auth.models import User  
2  
3  
4 class UserSerializer(serializers.ModelSerializer):  
5     password = serializers.CharField(write_only=True)  
6  
7     class Meta:  
8         model = User  
9         fields = ['username', 'password', 'email']  
10  
11    def create(self, validated_data):  
12        user = User.objects.create_user(**validated_data)  
13        return user
```

Signup: View e Rota (usando APIView)



```
1 class SignupAPIView(APIView):
2
3     def post(self, request):
4         serializer = UserSerializer(data=request.data)
5         serializer.is_valid(raise_exception=True)
6         serializer.save()
7         return Response(data=serializer.data, status=status.HTTP_201_CREATED)
8
```

Signup: View e Rota (Usando GenericViewSet)



```
1 class SignupAPIView(mixins.CreateModelMixin, viewsets.GenericViewSet):  
2     queryset = User.objects.all()  
3     serializer_class = UserSerializer
```

Rotas: Signup e Signin

```
1 from .viewsets import EstadoViewSet, CidadeViewSet, SignupAPIView  
2  
3 # Roteador  
4 router = DefaultRouter()  
5 router.register(r'signup', SignupAPIView)  
6  
7 urlpatterns = [  
8     path('', include(router.urls)),  
9     path('', include(estado_router.urls)),  
10    path('signin', ObtainAuthToken.as_view()),  
11 ]  
12
```



1 Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b

Vincular Objetos com seu Dono



```
1 from rest_framework import viewsets, permissions
2 from .models import SeuModelo
3 from .serializers import SeuModeloSerializer
4
5 class SeuModeloViewSet(viewsets.ModelViewSet):
6     queryset = SeuModelo.objects.all()
7     serializer_class = SeuModeloSerializer
8     permission_classes = [permissions.IsAuthenticated]
9
10    def perform_create(self, serializer):
11        # Extra params de .save() são incluidos em validated_data
12        serializer.save(usuario=self.request.user)
13
```

Vincular Objetos com seu Dono (via MIXIN)



```
1 from rest_framework import generics, permissions
2
3 class UserOwnedMixin:
4     def perform_create(self, serializer):
5         serializer.save(usuario=self.request.user)
6
7 class UserOwnedCreateAPIView(UserOwnedMixin, generics.CreateAPIView):
8     permission_classes = [permissions.IsAuthenticated]
```

Cada usuário acessa apenas seus dados

```
● ● ●  
1 class SeuModeloListView(generics.ListAPIView):  
2     serializer_class = SeuModeloSerializer  
3     permission_classes = [IsAuthenticated]  
4  
5     def get_queryset(self):  
6         user = self.request.user  
7         return SeuModelo.objects.filter(usuario=user)  
8  
9 class SeuModeloDetalhesView(generics.RetrieveAPIView):  
10    queryset = SeuModelo.objects.all()  
11    serializer_class = SeuModeloSerializer  
12    permission_classes = [IsAuthenticated]  
13  
14    def get_object(self):  
15        user = self.request.user  
16        obj = super().get_object()  
17        if obj.usuario == user:  
18            return obj  
19        else:  
20            raise Http404
```

Cada usuário acessa apenas seus dados

```
● ● ●  
1 class UserFilterBackend(filters.BaseFilterBackend):  
2     def filter_queryset(self, request, queryset, view):  
3         return queryset.filter(usuario=request.user)  
4  
5 class SeuModeloListView(generics.ListAPIView):  
6     serializer_class = SeuModeloSerializer  
7     permission_classes = [IsAuthenticated]  
8     filter_backends = [UserFilterBackend]  
9  
10    def get_queryset(self):  
11        return SeuModelo.objects.all()  
12  
13 class SeuModeloDetalhesView(generics.RetrieveAPIView):  
14     queryset = SeuModelo.objects.all()  
15     serializer_class = SeuModeloSerializer  
16     permission_classes = [IsAuthenticated]  
17     filter_backends = [UserFilterBackend]
```

Cada usuário acessa apenas seus dados (MIXIN)

```
● ● ●
1 from rest_framework import permissions
2
3 class UserOwnedMixin:
4     def get_queryset(self):
5         queryset = super().get_queryset()
6         return queryset.filter(usuario=self.request.user)
7
8     def get_object(self):
9         obj = super().get_object()
10        if obj.usuario == self.request.user:
11            return obj
12        else:
13            raise Http404
14
15 class UserOwnedListAPIView(UserOwnedMixin, generics.ListAPIView):
16     permission_classes = [permissions.IsAuthenticated]
17
18 class UserOwnedRetrieveAPIView(UserOwnedMixin, generics.RetrieveAPIView):
19     permission_classes = [permissions.IsAuthenticated]
20
```

JWT

A JSON Web Token authentication plugin for the Django REST Framework.

Setup passa a passo consultar: <https://django-rest-framework-simplejwt.readthedocs.io/en/latest/>

```
pip install djangorestframework-simplejwt[crypto]
```

Simple JWT Token

```
● ● ●  
1 from django.urls import path, include  
2 from rest_framework.routers import DefaultRouter  
3  
4 from rest_framework_simplejwt.views import (  
5     TokenObtainPairView,  
6     TokenRefreshView,  
7 )  
8  
9 # Roteador  
10 # ...  
11  
12 urlpatterns = [  
13     # ...  
14     path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),  
15     path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),  
16 ]  
17
```



1 Authorization: Bearer eyJhbGciOiJIU...eyJhbGciOiJIU

Autorização

DRF tem controle baseado em Permissions, e antes do corpo de cada view, todas as Permission são avaliadas, retornando um exception em caso de falha de qualquer uma delas:

- 401 Unauthorized (caso seja problemas de não-logado)
- 403 Forbidden (caso seja problema de permissão)

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

DRF Autorização



```
1 REST_FRAMEWORK = {  
2     'DEFAULT_AUTHENTICATION_CLASSES': [  
3         'rest_framework.authentication.TokenAuthentication',  
4     ],  
5     'DEFAULT_PERMISSION_CLASSES': [  
6         # 'rest_framework.permissions.AllowAny',  
7         'rest_framework.permissions.IsAuthenticated',  
8     ]  
9 }
```

401 Error: Unauthorized
Undocumented Response body

```
{  
    "detail": "Authentication credentials were not provided."  
}
```

DRF Autorização

Permissão Personalizada

1. Criar Subclasse de **BasePermission**
2. Implementar os Métodos:

```
.has_permission(self, request, view)
```

```
from rest_framework import permissions

class BlocklistPermission(permissions.BasePermission):
    """
    Global permission check for blocked IPs.
    """

    def has_permission(self, request, view):
        ip_addr = request.META['REMOTE_ADDR']
        blocked = Blocklist.objects.filter(ip_addr=ip_addr).exists()
        return not blocked
```

```
.has_object_permission(self, request, view, obj)
```

```
class IsOwnerOrReadOnly(permissions.BasePermission):
    """
    Object-level permission to only allow owners of an object to edit it.
    Assumes the model instance has an `owner` attribute.
    """

    def has_object_permission(self, request, view, obj):
        # Read permissions are allowed to any request,
        # so we'll always allow GET, HEAD or OPTIONS requests.
        if request.method in permissions.SAFE_METHODS:
            return True

        # Instance must have an attribute named `owner`.
        return obj.owner == request.user
```

DRF Autorização

- Permissão por View (CBV)

```
11 class EstadoViewSet(viewsets.ModelView):
12     queryset = Estado.objects.all()
13     serializer_class = EstadoSerializer
14     pagination_class = CustomPagination
15     filterset_class = EstadoFilter
16     permission_classes = [permissions.]
```

DRF Autorização

- Permissão por Function View

```
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def example_view(request, format=None):
    content = {
        'status': 'request was permitted'
    }
    return Response(content)
```

CORS

<https://github.com/adamchainz/django-cors-headers>

upload

Ajustes nos Model



```
1 from django.db import models
2
3
4 class Estado(models.Model):
5     nome = models.CharField(max_length=128, null=False, blank=False)
6     sigla = models.CharField(max_length=2, null=False, blank=False)
7     foto = models.ImageField(upload_to='estados', null=True, blank=True)
8
9     def __str__(self):
10         return self.nome
```

Configurar MEDIA



settings.py

```
1 # Media (upload)
2 MEDIA_URL = '/media/'
3 MEDIA_ROOT = BASE_DIR / 'media'
```



urls.py

```
1 urlpatterns = [
2     path('', include(router.urls)),
3     path('', include(estado_router.urls)),
4     # path('signin', ObtainAuthToken.as_view()),
5     path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
6     path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
7 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
8
```

Serializers: criação de um Específico para Upload

```
● ● ●  
1 class EstadoSerializer(serializers.ModelSerializer):  
2     cidades = CidadeSerializer(many=True, read_only=True)  
3     foto = serializers.ImageField(read_only=True)  
4  
5     class Meta:  
6         model = Estado  
7         fields = ['id', 'nome', 'sigla', 'foto', 'cidades']  
8  
9  
10 class UploadFotoEstadoSerializer(serializers.ModelSerializer):  
11  
12     foto = serializers.ImageField(required=True)  
13  
14     class Meta:  
15         model = Estado  
16         fields = ['foto']
```

Custom Action para upload: PUT /upload_foto

```
● ● ●
1 class EstadoViewSet(viewsets.ModelViewSet):
2     # ...
3
4     parser_classes = [parsers.JSONParser, parsers.MultipartParser]
5
6     def get_serializer_class(self):
7         if self.action not in ['upload_foto']:
8             return UploadFotoEstadoSerializer
9         return self.serializer_class
10
11     @decorators.action(url_path='upload_foto', detail=True, methods=[HTTPMethod.PUT])
12     def upload_foto(self, request, pk):
13         estado = Estado.objects.get(pk=pk)
14         serializer = UploadFotoEstadoSerializer(
15             instance=estado, data=request.data)
16         serializer.is_valid(raise_exception=True)
17         serializer.save()
18         return Response(EstadoSerializer(serializer.instance).data)
```

Request Body tipo "multipart"

The screenshot shows a POST request to `http://localhost:8000/estados/1/upload_foto/`. The response status is `200 OK` with `31.5 ms` latency and `89 B` size, timestamped "Just Now". The request body contains a file named `foto_piaui.png`, which is a file type. The response body is a JSON object representing a state:

```
1 {  
2   "id": 1,  
3   "nome": "Piauí",  
4   "sigla": "PI",  
5   "foto":  
6     "/media/estados/foto_piaui.png",  
7   "cidades": []  
}
```

FIM

