PYTHON
ACADEMY

GUIA COMPLETO DJANGO REST FRAMEWORK 2025: APIS PROFISSIONAIS DO ZERO

Guia completo DRF 3.15 em 2025: serializers, viewsets, authentication, permissions, filtros, paginação, testes. Construa APIs REST profissionais com Django REST Framework do básico ao avançado.

PYTHONACADEMY.COM.BR

Salve salve Pythonista!

Django REST Framework (DRF) é o padrão de facto para construir **APIs REST profissionais** com Python e Django.

Em 2025, com a versão **3.15+**, o DRF continua sendo a escolha número 1 para desenvolvedores que precisam criar APIs robustas, escaláveis e bem documentadas.

Este é o **guia definitivo de Django REST Framework**: um artigo pilar que cobre desde a configuração inicial até técnicas avançadas de autenticação, filtros, paginação, versionamento e testes.

Se você quer dominar a criação de APIs REST profissionais com Django, este guia é para você! 🚀

Vá Direto ao Assunto...

- Do not remove this line (it will not be displayed)

---

1. Por Que Usar Django REST Framework em 2025?

Vantagens do DRF

✅ **Serialização Poderosa** Converte tipos Python complexos (QuerySets, Models) para JSON/XML automaticamente

✅ **Autenticação Robusta** Token, Session, OAuth2, JWT nativamente suportados

✅ **Browsable API** Interface web automática para testar endpoints

✅ **Validação Integrada** Validação de dados com Serializers (similar ao Pydantic)

✅ **Documentação Automática** OpenAPI/Swagger integration out-of-the-box

✅ **Performance** ViewSets e QuerySets otimizados para alta performance

DRF vs Outras Opções

Critério	DRF	FastAPI	Flask-RESTful
Maturidade	✅ 10+ anos	⚠️ 5 anos	✅ 8 anos
ORM Integrado	✅ Django ORM	❌ Depende SQLAlchemy	❌ Manual
Admin	✅ Django Admin	❌ Não	❌ Não
Async	⚠️ Parcial	✅ Completo	❌ Não
Documentação	✅ Excelente	✅ Excelente	⚠️ Média
Comunidade	✅ Muito grande	✅ Crescendo rápido	⚠️ Média
Ideal para	Apps completas	Microserviços	APIs simples

2. Instalação e Configuração

Criar Ambiente Virtual

```
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows
```

Instalar Django + DRF

```
pip install django==5.1 djangorestframework==3.15
pip install markdown django-filter  # Opcionais mas úteis
```

Criar Projeto Django

```
django-admin startproject myapi
cd myapi
python manage.py startapp products
```

Configurar settings.py

```python
# myapi/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # DRF
    'rest_framework',
    'django_filters',

    # Suas apps
    'products',
]

# Configurações DRF
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticatedOrReadOnly',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
        'rest_framework.filters.OrderingFilter',
    ],
}
```

📚 **Leia mais:** Configurando um Projeto DRF

3. Models e Serializers

Criar Model

```python
# products/models.py
from django.db import models
from django.contrib.auth.models import User

class Category(models.Model):
    name = models.CharField(max_length=100)
    slug = models.SlugField(unique=True)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name_plural = "Categories"

    def __str__(self):
        return self.name

class Product(models.Model):
    STATUS_CHOICES = [
        ('draft', 'Draft'),
        ('published', 'Published'),
        ('archived', 'Archived'),
    ]

    name = models.CharField(max_length=200)
    slug = models.SlugField(unique=True)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField(default=0)
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
        related_name='products')
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
        default='draft')
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-created_at']
        indexes = [
            models.Index(fields=['slug']),
            models.Index(fields=['status']),
        ]
    def __str__(self):
        return self.name
```

Serializers: Básico ao Avançado

Serializer Simples

```python
# products/serializers.py
from rest_framework import serializers
from .models import Product, Category

class CategorySerializer(serializers.ModelSerializer):
    products_count = serializers.SerializerMethodField()

    class Meta:
        model = Category
        fields = ['id', 'name', 'slug', 'created_at', 'products_count']
        read_only_fields = ['id', 'created_at']

    def get_products_count(self, obj):
        return obj.products.count()

class ProductSerializer(serializers.ModelSerializer):
    category_name = serializers.CharField(source='category.name',
     read_only=True)
    created_by_username =
     serializers.CharField(source='created_by.username',
     read_only=True)

    class Meta:
        model = Product
        fields = [
            'id', 'name', 'slug', 'description', 'price', 'stock',
            'category', 'category_name', 'status',
            'created_by', 'created_by_username',
            'created_at', 'updated_at'
        ]
        read_only_fields = ['id', 'created_at', 'updated_at',
         'created_by']

    def validate_price(self, value):
        """Validação customizada"""
        if value <= 0:
            raise
         serializers.ValidationError("Preço deve ser maior que zero")
        if value > 999999.99:
            raise serializers.ValidationError("Preço muito alto")
        return value
    def validate(self, data):
        """Validação de múltiplos campos"""
        if data.get('status') == 'published' and data.get('stock', 0)
        == 0:
            raise serializers.ValidationError(
                "Não é possível publicar produto sem estoque"
            )
        return data
```

Serializer Nested (Aninhado)

```python
class ProductDetailSerializer(serializers.ModelSerializer):
    """Serializer com relacionamentos aninhados"""
    category = CategorySerializer(read_only=True)
    category_id = serializers.PrimaryKeyRelatedField(
        queryset=Category.objects.all(),
        source='category',
        write_only=True
    )

    class Meta:
        model = Product
        fields = '__all__'

    def to_representation(self, instance):
        """Customizar resposta JSON"""
        data = super().to_representation(instance)

        # Adicionar campo calculado
        data['in_stock'] = instance.stock > 0
        data['formatted_price'] = f"R$ {instance.price:.2f}"

        return data
```

📚 **Leia mais:** Serializers no DRF

4. Views e ViewSets

Function-Based Views (FBV)

```python
# products/views.py
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework import status

@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])
def product_list(request):
    if request.method == 'GET':
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response(serializer.data)

    elif request.method == 'POST':
        serializer = ProductSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save(created_by=request.user)
            return Response(serializer.data,
        status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
        status=status.HTTP_400_BAD_REQUEST)
```

Class-Based Views (CBV)

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class ProductListView(APIView):
    """API para listar e criar produtos"""

    def get(self, request):
        products = Product.objects.select_related('category').all()
        serializer = ProductSerializer(products, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = ProductSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save(created_by=request.user)
            return Response(serializer.data,
        status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
        status=status.HTTP_400_BAD_REQUEST)
```

Generic Views

```python
from rest_framework import generics

class ProductListCreateView(generics.ListCreateAPIView):
    queryset = Product.objects.select_related('category', 'created_by')
    serializer_class = ProductSerializer

    def perform_create(self, serializer):
        serializer.save(created_by=self.request.user)

class ProductDetailView(generics.RetrieveUpdateDestroyAPIView):
    queryset = Product.objects.all()
    serializer_class = ProductDetailSerializer
    lookup_field = 'slug'
```

ViewSets (Recomendado)

```python
from rest_framework import viewsets
from rest_framework.decorators import action
from rest_framework.response import Response

class ProductViewSet(viewsets.ModelViewSet):
    """
    ViewSet completo para Product

    list:    GET /products/
    create:  POST /products/
    retrieve: GET /products/{id}/
    update:  PUT /products/{id}/
    partial_update: PATCH /products/{id}/
    destroy: DELETE /products/{id}/
    """
    queryset = Product.objects.select_related('category', 'created_by')
    serializer_class = ProductSerializer
    lookup_field = 'slug'
    filterset_fields = ['status', 'category']
    search_fields = ['name', 'description']
    ordering_fields = ['price', 'created_at', 'stock']

    def get_serializer_class(self):
        """Usar serializer diferente para detail"""
        if self.action == 'retrieve':
            return ProductDetailSerializer
        return ProductSerializer

    def perform_create(self, serializer):
        serializer.save(created_by=self.request.user)

    @action(detail=True, methods=['post'])
    def publish(self, request, slug=None):
        """Custom action: /products/{slug}/publish/"""
        product = self.get_object()

        if product.stock == 0:
            return Response(
                {'error': 'Produto sem estoque'},
                status=status.HTTP_400_BAD_REQUEST
            )
        product.status = 'published'
        product.save()

        serializer = self.get_serializer(product)
        return Response(serializer.data)

    @action(detail=False, methods=['get'])
    def low_stock(self, request):
        """Custom action: /products/low_stock/"""
        products = self.queryset.filter(stock__lt=10,
         status='published')
        serializer = self.get_serializer(products, many=True)
        return Response(serializer.data)
```

5. URLs e Routers

URLs Manual

```python
# products/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('products/', views.ProductListCreateView.as_view(),
         name='product-list'),
    path('products/<slug:slug>/', views.ProductDetailView.as_view(),
         name='product-detail'),
]
```

Routers (ViewSets)

```python
# myapi/urls.py
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from products.views import ProductViewSet, CategoryViewSet

# Router automático
router = DefaultRouter()
router.register(r'products', ProductViewSet, basename='product')
router.register(r'categories', CategoryViewSet, basename='category')

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include(router.urls)),
    path('api-auth/', include('rest_framework.urls')),  # Login/Logout
]

# URLs geradas automaticamente:
# GET    /api/products/
# POST   /api/products/
# GET    /api/products/{slug}/
# PUT    /api/products/{slug}/
# PATCH  /api/products/{slug}/
# DELETE /api/products/{slug}/
# POST   /api/products/{slug}/publish/
# GET    /api/products/low_stock/
```

6. Autenticação e Permissões

Autenticação: Token

```
# settings.py
INSTALLED_APPS += ['rest_framework.authtoken']

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

```
python manage.py migrate
python manage.py drf_create_token <username>
```

```
# Criar token automaticamente para novos usuários
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Autenticação: JWT

```
pip install djangorestframework-simplejwt
```

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}

from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
}
```

```python
# urls.py
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns += [
    path('api/token/', TokenObtainPairView.as_view(),
         name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(),
         name='token_refresh'),
]
```

Permissões Customizadas

```python
# products/permissions.py
from rest_framework import permissions

class IsOwnerOrReadOnly(permissions.BasePermission):
    """Apenas dono pode editar"""

    def has_object_permission(self, request, view, obj):
        # Leitura permitida para todos
        if request.method in permissions.SAFE_METHODS:
            return True

        # Escrita apenas para dono
        return obj.created_by == request.user

class IsAdminOrReadOnly(permissions.BasePermission):
    """Apenas admin pode criar/editar/deletar"""

    def has_permission(self, request, view):
        if request.method in permissions.SAFE_METHODS:
            return True
        return request.user and request.user.is_staff
```

```python
# Usar nas views
class ProductViewSet(viewsets.ModelViewSet):
    permission_classes = [IsOwnerOrReadOnly]
    # ou
    permission_classes = [IsAdminOrReadOnly]
```

7. Filtros, Busca e Ordenação

```
pip install django-filter
```

```python
# products/filters.py
from django_filters import rest_framework as filters
from .models import Product

class ProductFilter(filters.FilterSet):
    min_price = filters.NumberFilter(field_name='price',
        lookup_expr='gte')
    max_price = filters.NumberFilter(field_name='price',
        lookup_expr='lte')
    name = filters.CharFilter(lookup_expr='icontains')
    created_after = filters.DateTimeFilter(field_name='created_at',
        lookup_expr='gte')

    class Meta:
        model = Product
        fields = ['category', 'status']
```

```python
# products/views.py
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import filters

class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter,
        filters.OrderingFilter]
    filterset_class = ProductFilter
    search_fields = ['name', 'description', 'category__name']
    ordering_fields = ['price', 'created_at', 'stock']
    ordering = ['-created_at']
```

Exemplos de uso:

```
GET /api/products/?status=published
GET /api/products/?min_price=50&max_price=500
GET /api/products/?search=notebook
GET /api/products/?ordering=-price
GET /api/products/?category=1&status=published&ordering=price
```

📚 **Leia mais:** Filtros, Busca e Ordenação no DRF

8. Paginação

PageNumberPagination

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

Paginação Customizada

```
# myapi/pagination.py
from rest_framework.pagination import PageNumberPagination

class CustomPagination(PageNumberPagination):
    page_size = 20
    page_size_query_param = 'page_size'
    max_page_size = 100
```

```
# products/views.py
class ProductViewSet(viewsets.ModelViewSet):
    pagination_class = CustomPagination
```

Resposta paginada:

```
{
    "count": 125,
    "next": "http://api.example.com/products/?page=2",
    "previous": null,
    "results": [
        {...},
        {...}
    ]
}
```

9. Versionamento de API

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_VERSIONING_CLASS': 'rest_framework.versioning.URLPathVersioning',
    'DEFAULT_VERSION': 'v1',
    'ALLOWED_VERSIONS': ['v1', 'v2'],
}
```

```
# urls.py
urlpatterns = [
    path('api/<str:version>/', include(router.urls)),
]
```

```
# views.py
class ProductViewSet(viewsets.ModelViewSet):
    def get_serializer_class(self):
        if self.request.version == 'v2':
            return ProductSerializerV2
        return ProductSerializer
```

10. Testes com DRF

Testes Básicos

```python
# products/tests.py
from rest_framework.test import APITestCase, APIClient
from rest_framework import status
from django.contrib.auth.models import User
from .models import Product, Category

class ProductAPITestCase(APITestCase):

    def setUp(self):
        """Executado antes de cada teste"""
        self.client = APIClient()
        self.user = User.objects.create_user(
            username='testuser',
            password='testpass123'
        )
        self.category = Category.objects.create(
            name='Eletrônicos',
            slug='eletronicos'
        )
        self.product = Product.objects.create(
            name='Notebook',
            slug='notebook',
            description='Notebook de teste',
            price=3000.00,
            stock=10,
            category=self.category,
            created_by=self.user,
            status='published'
        )

    def test_list_products(self):
        """Teste GET /api/products/"""
        response = self.client.get('/api/products/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['results']), 1)

    def test_create_product_authenticated(self):
        """Teste POST /api/products/ (autenticado)"""
        self.client.force_authenticate(user=self.user)

        data = {
            'name': 'Mouse',
            'slug': 'mouse',
            'description': 'Mouse gamer',
            'price': 150.00,
            'stock': 50,
            'category': self.category.id,
            'status': 'draft'
        }

        response = self.client.post('/api/products/', data,
format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Product.objects.count(), 2)

    def test_create_product_unauthenticated(self):
        """Teste POST sem autenticação"""
        data = {'name': 'Test'}
        response = self.client.post('/api/products/', data)
        self.assertEqual(response.status_code,
status.HTTP_401_UNAUTHORIZED)

    def test_update_product(self):
        """Teste PATCH /api/products/{slug}/"""
        self.client.force_authenticate(user=self.user)

        data = {'price': 2500.00}
        response = self.client.patch(
            f'/api/products/{self.product.slug}/',
            data,
            format='json'
        )

        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.product.refresh_from_db()
        self.assertEqual(float(self.product.price), 2500.00)

    def test_delete_product(self):
        """Teste DELETE /api/products/{slug}/"""
        self.client.force_authenticate(user=self.user)

        response = self.client.delete(f'/api/products/
{self.product.slug}/')
```

```python
        self.assertEqual(response.status_code,
         status.HTTP_204_NO_CONTENT)
        self.assertEqual(Product.objects.count(), 0)

    def test_filter_by_status(self):
        """Teste filtros"""
        response = self.client.get('/api/products/?status=published')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['results']), 1)

    def test_search(self):
        """Teste busca"""
        response = self.client.get('/api/products/?search=Notebook')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['results']), 1)
```

Executar Testes

```bash
# Todos os testes
python manage.py test

# Testes específicos
python manage.py test products.tests.ProductAPITestCase

# Com coverage
pip install coverage
coverage run --source='.' manage.py test
coverage report
coverage html  # Gera relatório HTML
```

11. Documentação Automática (OpenAPI/Swagger)

```
pip install drf-spectacular
```

```python
# settings.py
INSTALLED_APPS += ['drf_spectacular']

REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
}

SPECTACULAR_SETTINGS = {
    'TITLE': 'Products API',
    'DESCRIPTION': 'API REST para gerenciamento de produtos',
    'VERSION': '1.0.0',
    'SERVE_INCLUDE_SCHEMA': False,
}
```

```python
# urls.py
from drf_spectacular.views import SpectacularAPIView,
        SpectacularSwaggerView

urlpatterns += [
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    path('api/docs/', SpectacularSwaggerView.as_view(url_name='schema'),
        name='swagger-ui'),
]
```

Acesse: `http://localhost:8000/api/docs/`

12. CORS (Cross-Origin Resource Sharing)

```
pip install django-cors-headers
```

```python
# settings.py
INSTALLED_APPS += ['corsheaders']

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    # ...
]

# Desenvolvimento
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://localhost:8080",
]

# Produção (mais restritivo)
CORS_ALLOWED_ORIGINS = [
    "https://meusite.com",
]

# Ou permitir tudo (NÃO recomendado em produção)
CORS_ALLOW_ALL_ORIGINS = True
```

13. Throttling (Rate Limiting)

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ],
    'DEFAULT_THROTTLE_RATES': {
        'anon': '100/day',
        'user': '1000/day'
    }
}
```

Throttling Customizado

```
from rest_framework.throttling import UserRateThrottle

class BurstRateThrottle(UserRateThrottle):
    rate = '60/minute'

class SustainedRateThrottle(UserRateThrottle):
    rate = '1000/day'

class ProductViewSet(viewsets.ModelViewSet):
    throttle_classes = [BurstRateThrottle, SustainedRateThrottle]
```

14. Boas Práticas DRF

✅ Use ViewSets para CRUD Completo

```python
# ✅ Bom
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer

# ❌ Evite repetir código
class ProductListView(APIView): ...
class ProductDetailView(APIView): ...
class ProductCreateView(APIView): ...
```

✅ Otimize Queries

```python
# ✅ Bom - select_related para ForeignKey
Product.objects.select_related('category', 'created_by')

# ✅ Bom - prefetch_related para ManyToMany
Product.objects.prefetch_related('tags')

# ❌ Ruim - N+1 queries
for product in Product.objects.all():
    print(product.category.name)  # Query para cada produto!
```

✅ Valide Dados no Serializer

```python
# ✅ Bom
class ProductSerializer(serializers.ModelSerializer):
    def validate_price(self, value):
        if value <= 0:
            raise serializers.ValidationError("Preço inválido")
        return value

# ❌ Evite validação na view
```

✅ Use Permissions Adequadas

```python
# ✅ Bom - granular
class ProductViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticatedOrReadOnly]

    def get_permissions(self):
        if self.action in ['create', 'update', 'destroy']:
            return [IsAdminUser()]
        return super().get_permissions()

# ❌ Ruim - tudo aberto
permission_classes = [AllowAny]
```

✅ Versione sua API

```
# ✅ Bom
/api/v1/products/
/api/v2/products/

# ❌ Evite quebrar a API sem aviso
```

15. Deploy e Produção

Checklist de Produção

✅ **DEBUG = False**

```
DEBUG = False
ALLOWED_HOSTS = ['meusite.com', 'api.meusite.com']
```

✅ **Use PostgreSQL**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'myapi_db',
        'USER': 'postgres',
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

✅ **Segurança**

```
SECRET_KEY = os.environ.get('SECRET_KEY')
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
```

✅ **Logging**

```
LOGGING = {
    'version': 1,
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'filename': '/var/log/django/api.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'INFO',
        },
    },
}
```

✅ Servir com Gunicorn + Nginx

```
pip install gunicorn
gunicorn myapi.wsgi:application --bind 0.0.0.0:8000
```

16. Quando Usar DRF vs FastAPI

Cenário	Use DRF	Use FastAPI
App Django existente	✅ Sim	❌ Não
Precisa Django Admin	✅ Sim	❌ Não
API pura (microserviço)	⚠️ OK	✅ Melhor
Async/WebSocket	⚠️ Limitado	✅ Excelente
Time já conhece Django	✅ Sim	⚠️ Depende
Performance máxima	⚠️ Boa	✅ Excelente
Documentação auto	✅ drf-spectacular	✅ Built-in

---

17. Recursos e Próximos Passos

Documentação Oficial

📚 Django REST Framework Docs 📚 Django 5.1 Docs

Posts Relacionados

🔗 Construção de APIs com DRF 🔗 Configurando Projeto DRF 🔗 Serializers no DRF 🔗 Filtros, Busca e Ordenação 🔗 Guia Completo Django 2025

Continue Aprendendo

1. **GraphQL com Graphene-Django**
2. **WebSockets com Django Channels**
3. **Celery para tarefas assíncronas**
4. **Cache com Redis**
5. **CI/CD com GitHub Actions**

Conclusão

Neste **guia completo de Django REST Framework**, você aprendeu:

✅ **Por que usar DRF** - Vantagens e comparações ✅ **Instalação e setup** - Configuração completa ✅ **Models e Serializers** - Validação e serialização ✅ **Views e ViewSets** - FBV, CBV, Generic, ViewSets ✅ **Autenticação** - Token, JWT, permissões customizadas ✅ **Filtros e paginação** - django-filter, search, ordering ✅ **Testes** - APITestCase completo ✅ **Documentação** - OpenAPI/Swagger ✅ **Produção** - Deploy, segurança, boas práticas

Principais lições:

- **ViewSets** são a forma mais eficiente de criar APIs CRUD
- **Serializers** fazem validação automática de dados

- **Permissions** garantem segurança granular
- **Filtros** tornam a API flexível e poderosa
- **Testes** são essenciais para APIs confiáveis
- **DRF** é ideal para apps Django que precisam de APIs REST

Próximos passos:

- Implemente autenticação JWT em seu projeto
- Adicione testes com cobertura > 80%
- Configure documentação Swagger
- Otimize queries com select_related/prefetch_related
- Deploy em produção com Docker + Gunicorn + Nginx

Django REST Framework é a escolha certa quando você precisa de APIs REST robustas, bem documentadas e escaláveis com Django!

Até a próxima! 🚀