**1. Autenticação**

A autenticação verifica **quem é o usuário** que está fazendo a requisição. Ela se preocupa em identificar o usuário (ou determinar que ele não está autenticado).

**Como funciona?**

1. **Requisição recebida pelo servidor**:
   * Quando um cliente (ex.: navegador, aplicativo ou outra API) faz uma requisição, ela geralmente inclui um cabeçalho de autenticação (ex.: um token, uma sessão ou um nome de usuário/senha).
2. **DRF chama o backend de autenticação**:
   * O DRF verifica as classes de autenticação definidas na configuração do projeto (via DEFAULT\_AUTHENTICATION\_CLASSES em settings.py).
3. **Backends disponíveis**:
   * O DRF já vem com implementações padrão, como:
     + **SessionAuthentication**: Usa sessões e cookies (útil para aplicativos web baseados no Django).
     + **TokenAuthentication**: Usa um token enviado no cabeçalho da requisição.
     + **BasicAuthentication**: Usa o esquema HTTP Basic Auth (transmite username/password).
4. **Resultado**:
   * Se a autenticação for bem-sucedida, o DRF anexa o usuário autenticado ao objeto da requisição (request.user).
   * Se a autenticação falhar, o DRF retorna uma resposta de erro (geralmente um status HTTP 401 - Unauthorized).

**2. Permissões**

Depois que o usuário é autenticado, o próximo passo é verificar **o que ele pode fazer**. Aqui, o DRF avalia as permissões para determinar se o usuário tem direito de acessar o recurso solicitado.

**Como funciona?**

1. **Requisição chega ao sistema de permissões**:
   * O DRF usa as classes definidas em DEFAULT\_PERMISSION\_CLASSES (ou personalizadas por view).
2. **Verificação de permissões**:
   * O DRF verifica se o usuário tem acesso ao recurso baseado nas classes de permissão.
3. **Exemplos de permissões padrão no DRF**:
   * **AllowAny**: Qualquer um pode acessar (não exige autenticação).
   * **IsAuthenticated**: Apenas usuários autenticados podem acessar.
   * **IsAdminUser**: Apenas usuários administradores têm acesso.
   * **IsAuthenticatedOrReadOnly**: Usuários autenticados têm acesso total; os não autenticados podem apenas realizar operações de leitura (GET).
4. **Resultado**:
   * Se o usuário tiver permissão, o DRF processa a requisição normalmente.
   * Caso contrário, uma resposta de erro (geralmente HTTP 403 - Forbidden) é retornada.

**Fluxo Resumido**

1. **Cliente faz uma requisição**.
2. **Autenticação**: O DRF verifica quem é o usuário (ou se ele não é autenticado).
3. **Permissões**: O DRF verifica se o usuário pode realizar a ação solicitada.
4. **Resposta**:
   * Se ambas as etapas forem bem-sucedidas, a requisição é processada.
   * Caso contrário, uma mensagem de erro é retornada.

**Exemplo de Código**

Configuração das classes de autenticação e permissão no settings.py:

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.TokenAuthentication',

'rest\_framework.authentication.SessionAuthentication',

],

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.IsAuthenticated',

],

}

**Uso de permissões em uma view:**

from rest\_framework.permissions import IsAuthenticated, IsAdminUser

from rest\_framework.views import APIView

from rest\_framework.response import Response

class ExampleView(APIView):

permission\_classes = [IsAuthenticated] # Apenas usuários autenticados podem acessar

def get(self, request):

return Response({"message": "Bem-vindo, você está autenticado!"})

**Resumo Didático**

* **Autenticação** = Verifica *quem você é*.
* **Permissão** = Verifica *o que você pode fazer*.
* O DRF usa **classes configuráveis** para ambas as etapas, permitindo customizações para atender diferentes necessidades do projeto.

Deixa-me ver se entendi. Imagine que eu desenvolvi uma api que fornece acesso a páginas de PDF de um ebook. Quero passar essa api para ser usado pelo sistema de e-learning (AVA) de uma faculdade. Tenho que criar um usuário no Admin do DRF e gerar um token para que na chamada da api este token seja fornecido. É isso?

**1. Configurando a API com TokenAuthentication**

No seu caso, você quer que a API só seja acessada por um sistema de e-learning (AVA) autenticado. Para isso, você pode usar o esquema de autenticação por token do DRF.

1. **Ativando o TokenAuthentication**: No arquivo settings.py, você precisa configurar as classes de autenticação para usar tokens:

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.TokenAuthentication',

],

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.IsAuthenticated',

],

}

Aqui:

* **TokenAuthentication**: Exige que cada requisição inclua um token.
* **IsAuthenticated**: Garante que apenas usuários autenticados possam acessar os endpoints da API.

**2. Criando o Usuário e Gerando o Token**

Para permitir que o AVA acesse sua API, você cria um usuário e associa um token a ele:

1. **Criar o usuário no painel admin do Django**:
   * Acesse o admin (/admin).
   * Vá até a seção de "Usuários" e crie um usuário que represente o AVA (por exemplo, ava\_faculdade).
   * Você pode criar o usuário com permissões mínimas, já que ele só será usado para autenticação.
2. **Gerar um token para o usuário**: O DRF oferece um modelo de token que pode ser usado para autenticar requisições. Você pode gerar o token de duas formas:
   * **No painel admin** (se você ativar a seção de tokens):
     + Ative o modelo Token no admin adicionando rest\_framework.authtoken ao INSTALLED\_APPS.
     + Rode a migração:

python manage.py migrate

*  Acesse o admin, encontre a seção de tokens e crie um para o usuário ava\_faculdade.

 **Via terminal do Django**: No terminal interativo do Django, gere um token para o usuário:

from rest\_framework.authtoken.models import Token

from django.contrib.auth.models import User

user = User.objects.get(username="ava\_faculdade")

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

print(token.key)

**3. Uso do Token na Requisição**

Agora que o token foi gerado, o AVA deve incluí-lo no cabeçalho de cada requisição para autenticar o acesso à API.

1. **Exemplo de requisição com o token**: O sistema AVA deve enviar o token no cabeçalho Authorization usando o esquema Token.

GET /api/ebooks/1/pages/ HTTP/1.1

Host: sua-api.com

Authorization: Token seu\_token\_aqui

1. **Como a API valida**: Quando o DRF recebe essa requisição:
   * Ele verifica o cabeçalho Authorization.
   * Valida o token enviado.
   * Identifica o usuário associado ao token.
   * Se tudo estiver correto, o usuário será autenticado e a permissão concedida.

**4. Protegendo os Endpoints**

Você pode proteger os endpoints para garantir que apenas usuários autenticados acessem:

from rest\_framework.permissions import IsAuthenticated

from rest\_framework.views import APIView

from rest\_framework.response import Response

class EbookPagesView(APIView):

permission\_classes = [IsAuthenticated] # Apenas usuários autenticados podem acessar

def get(self, request, ebook\_id, page\_number):

# Lógica para retornar a página do ebook

return Response({"page": f"Conteúdo da página {page\_number} do ebook {ebook\_id}"})

**Fluxo Completo**

1. **Você configura a API com TokenAuthentication e IsAuthenticated.**
2. **Cria um usuário para o AVA no painel admin.**
3. **Gera um token para esse usuário.**
4. **Fornece o token para o AVA.**
5. **O AVA inclui o token nas requisições enviadas à API.**
6. **O DRF valida o token e permite o acesso, ou retorna um erro de autenticação.**

**Cenários Especiais**

* **Expiração de tokens**: Tokens gerados pelo DRF por padrão não expiram. Se você precisar de tokens temporários, pode implementar uma solução customizada ou usar bibliotecas como o **Django Rest Framework Simple JWT**.
* **Revogação de tokens**: Se o token do AVA for comprometido, você pode revogá-lo deletando o token associado ao usuário.

Com essa abordagem, você garante que apenas o sistema AVA autenticado tenha acesso às páginas do ebook!

Nas permissões padrão no DRF você não citou a DjangoModelPermissions

Você está absolutamente certo! O DjangoModelPermissions é uma das permissões padrão do Django Rest Framework (DRF) e merece destaque. Vamos entender como ela funciona e onde pode ser aplicada.

**DjangoModelPermissions**

O DjangoModelPermissions é uma classe de permissão que conecta os sistemas de permissões do Django com o DRF. Ele verifica as permissões associadas ao modelo do banco de dados antes de conceder acesso a um endpoint.

**Como funciona?**

* O DjangoModelPermissions verifica se o usuário autenticado tem as permissões associadas ao modelo para a ação que ele está tentando realizar.
* As permissões no Django são definidas automaticamente quando você cria um modelo. Para cada modelo, o Django cria as permissões:
  + add\_<model>: Para criar objetos.
  + change\_<model>: Para editar objetos.
  + delete\_<model>: Para excluir objetos.
  + view\_<model>: Para visualizar objetos (a partir do Django 2.1).

**Exemplo de configuração**

Imagine que você tenha um modelo Ebook para gerenciar os e-books da API. Você quer garantir que apenas usuários com permissões específicas possam acessar os endpoints relacionados.

1. **Definindo permissões no modelo**: No modelo Ebook, as permissões são criadas automaticamente, mas você também pode definir permissões personalizadas:

from django.db import models

class Ebook(models.Model):

title = models.CharField(max\_length=255)

author = models.CharField(max\_length=255)

class Meta:

permissions = [

("special\_permission", "Can perform a special action"),

]

**Configurando a permissão na view**: Você pode usar o DjangoModelPermissions na sua APIView ou ViewSet:

from rest\_framework.permissions import DjangoModelPermissions

from rest\_framework.viewsets import ModelViewSet

from .models import Ebook

from .serializers import EbookSerializer

class EbookViewSet(ModelViewSet):

queryset = Ebook.objects.all()

serializer\_class = EbookSerializer

permission\_classes = [DjangoModelPermissions]

1. **Como o DRF verifica as permissões**:
   * Se o usuário autenticado tiver as permissões adequadas no Django Admin (ou via outro sistema de gerenciamento de permissões), ele poderá realizar a ação.
   * Caso contrário, o DRF retorna um erro de **403 Forbidden**.

**Exemplo Prático**

Vamos supor que você tenha um administrador no Django Admin que deseja configurar permissões para diferentes usuários:

1. Acesse o **Admin do Django**.
2. Edite o grupo ou o usuário diretamente.
3. Conceda permissões específicas relacionadas ao modelo Ebook, como:
   * add\_ebook
   * change\_ebook
   * delete\_ebook
   * view\_ebook

Usuários que pertencem a um grupo com essas permissões poderão acessar os endpoints correspondentes.

**Configuração Global (opcional)**

Se você quiser que todas as views da API usem o DjangoModelPermissions por padrão, basta configurá-lo no settings.py:

REST\_FRAMEWORK = {

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.DjangoModelPermissions',

],

}

**Quando usar o DjangoModelPermissions?**

* Quando sua API está fortemente integrada ao sistema de modelos do Django.
* Se você já gerencia permissões de usuários e grupos no painel administrativo do Django.
* Quando deseja evitar implementar permissões customizadas, aproveitando o que já existe no sistema de permissões do Django.

**Resumo**

* O DjangoModelPermissions é útil para aproveitar o sistema robusto de permissões do Django no DRF.
* Ele verifica as permissões associadas ao modelo para as ações add, change, delete e view.
* É uma solução prática para APIs que compartilham as mesmas permissões do painel administrativo.

Obrigado por apontar! O DjangoModelPermissions é uma classe poderosa e merece estar no radar. 🚀