def has\_permission(*self*, *request*, *view*):

O método has\_permission(self, request, view) no Django Rest Framework (DRF) é utilizado nas **classes de permissão** para determinar se o usuário que está fazendo a requisição tem permissão para acessar o endpoint da API. É um dos principais pontos de controle de acesso no DRF.

**Como funciona?**

1. **Papel**:
   * O método é chamado antes de o endpoint ser acessado.
   * Ele verifica se o usuário tem **permissão global** para acessar a view (não está relacionado diretamente a objetos específicos).
2. **Parâmetros**:
   * request: O objeto da requisição, que contém informações como o usuário autenticado (request.user), o token (se houver), e outros dados da requisição.
   * view: A instância da view que está sendo acessada. Isso permite verificar metadados ou configurações específicas da view.
3. **Retorno**:
   * O método retorna:
     + **True**: Se o usuário tiver permissão para acessar o recurso.
     + **False**: Caso contrário (o DRF retornará um erro HTTP 403 - Forbidden).

**Exemplo Prático**

Vamos criar uma classe de permissão personalizada chamada IsAdminOrReadOnly. Ela permite que usuários autenticados como administradores façam qualquer operação, enquanto outros usuários podem apenas ler (GET).

from rest\_framework.permissions import BasePermission

class IsAdminOrReadOnly(BasePermission):

"""

Permite acesso completo apenas para administradores.

Usuários não administradores podem apenas visualizar (GET).

"""

def has\_permission(self, request, view):

# Permitir acesso total para usuários admin

if request.user and request.user.is\_staff:

return True

# Permitir apenas leitura para outros usuários

if request.method in ['GET', 'HEAD', 'OPTIONS']:

return True

# Negar para outros métodos

return False

**Passo a Passo do Funcionamento**

1. **Usuário faz uma requisição**:
   * Por exemplo, POST /api/ebooks/ para criar um eBook.
2. **DRF chama o método has\_permission**:
   * A classe de permissão configurada na view verifica a permissão do usuário.
3. **Verificação de permissão**:
   * O método avalia o tipo de requisição (ex.: GET ou POST) e o status do usuário (ex.: is\_staff).
4. **Resposta baseada no retorno**:
   * Se o método retornar True, a requisição continua.
   * Se retornar False, o DRF retorna:

HTTP/1.1 403 Forbidden

**Aplicação na View**

Para usar a classe de permissão na view, basta configurá-la no atributo permission\_classes:

from rest\_framework.viewsets import ModelViewSet

from .models import Ebook

from .serializers import EbookSerializer

from .permissions import IsAdminOrReadOnly

class EbookViewSet(ModelViewSet):

queryset = Ebook.objects.all()

serializer\_class = EbookSerializer

permission\_classes = [IsAdminOrReadOnly]

**Dicas Importantes**

1. **Ordem das permissões**:
   * Se houver múltiplas permissões na view, todas são avaliadas. O usuário precisa passar por todas para acessar o recurso.
2. **Comparação com has\_object\_permission**:
   * Enquanto has\_permission verifica permissões **globais**, o método has\_object\_permission(self, request, view, obj) é usado para verificar permissões específicas para **objetos individuais**.
3. **Permissões genéricas**:
   * Você pode reutilizar has\_permission para implementar comportamentos amplos, como limitar acesso a métodos HTTP, checar autenticação, ou aplicar políticas baseadas em IP.

**Exemplo com Restrição por Método HTTP**

Uma classe de permissão que permite apenas requisições GET e POST:

class AllowGetAndPostOnly(BasePermission):

def has\_permission(self, request, view):

# Permitir apenas GET e POST

return request.method in ['GET', 'POST']

**Resumo**

* O método has\_permission verifica **se o usuário tem permissão global** para acessar um endpoint.
* Ele é chamado antes do processamento da requisição.
* Use-o para validar permissões com base no usuário, no método HTTP ou em qualquer outra lógica global.
* Se precisar verificar permissões relacionadas a um **objeto específico**, combine com o método has\_object\_permission.

Essa flexibilidade torna o DRF poderoso na criação de APIs seguras e personalizáveis!