**Throttling**

O **throttling** no Django Rest Framework (DRF) é uma funcionalidade que limita a **quantidade de requisições** que um cliente pode fazer em um determinado intervalo de tempo. É útil para prevenir abuso, proteger o servidor contra sobrecarga e implementar limites por usuário, por IP ou outros critérios.

**Como o Throttling Funciona no DRF?**

1. **Objetivo**:
   * Controlar a taxa de requisições feitas por um cliente a um endpoint da API.
   * Impor limites por **IP**, **usuário** ou outros identificadores.
2. **Classes de Throttling**: O DRF fornece classes de throttling que definem diferentes estratégias para aplicar essas limitações.
3. **Como é aplicado?**
   * O DRF verifica se a requisição ultrapassa os limites configurados.
   * Se o cliente atingir o limite, o DRF retorna:

HTTP/1.1 429 Too Many Requests

* + Com uma mensagem indicando que o cliente precisa aguardar antes de tentar novamente.

**Classes de Throttling Disponíveis**

O DRF possui algumas classes de throttling padrão:

1. **AnonRateThrottle**:
   * Limita as requisições feitas por **usuários anônimos** (não autenticados).
   * Usa o endereço IP do cliente como identificador.
2. **UserRateThrottle**:
   * Limita as requisições feitas por **usuários autenticados**.
   * Usa o identificador do usuário autenticado.
3. **ScopedRateThrottle**:
   * Permite definir diferentes limites de taxa com base no **escopo do endpoint**.
   * Útil quando diferentes endpoints requerem limites específicos.
4. **Custom Throttles**:
   * Você pode criar classes personalizadas para implementar lógica específica.

**Configuração no settings.py**

Para habilitar o throttling, configure o atributo DEFAULT\_THROTTLE\_CLASSES e DEFAULT\_THROTTLE\_RATES no settings.py:

REST\_FRAMEWORK = {

'DEFAULT\_THROTTLE\_CLASSES': [

'rest\_framework.throttling.AnonRateThrottle',

'rest\_framework.throttling.UserRateThrottle',

],

'DEFAULT\_THROTTLE\_RATES': {

'anon': '10/minute', # Limite para usuários anônimos

'user': '100/hour', # Limite para usuários autenticados

},

}

**Exemplo Prático**

**1. Limitar Requisições para Anônimos e Usuários Autenticados**

* Usuários anônimos podem fazer no máximo 10 requisições por minuto.
* Usuários autenticados podem fazer no máximo 100 requisições por hora.

Requisição anônima excessiva:

HTTP/1.1 429 Too Many Requests

Content-Type: application/json

{

"detail": "Request was throttled. Expected available in 30 seconds."

}

**2. Usando ScopedRateThrottle**

1. Defina um escopo específico para throttling no settings.py:

REST\_FRAMEWORK = {

'DEFAULT\_THROTTLE\_CLASSES': [

'rest\_framework.throttling.ScopedRateThrottle',

],

'DEFAULT\_THROTTLE\_RATES': {

'low\_rate': '5/minute', # Escopo com limite baixo

'high\_rate': '50/minute', # Escopo com limite alto

},

}

Configure o escopo na view ou ViewSet:

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework.throttling import ScopedRateThrottle

class LowRateView(APIView):

throttle\_classes = [ScopedRateThrottle]

throttle\_scope = 'low\_rate' # Usará o limite definido no escopo 'low\_rate'

def get(self, request):

return Response({"message": "Você acessou um endpoint com baixa taxa!"})

**Como o DRF Identifica o Cliente?**

* **AnonRateThrottle**:
  + Identifica clientes anônimos pelo endereço IP.
* **UserRateThrottle**:
  + Identifica clientes autenticados pelo ID do usuário (request.user.id).

**Criando uma Classe de Throttling Personalizada**

Você pode criar um throttling customizado para atender a casos específicos, como limites baseados em parâmetros ou outros identificadores.

Exemplo: Limitar requisições por um parâmetro de API (ex.: um campo api\_key na URL):

from rest\_framework.throttling import BaseThrottle

class APIKeyThrottle(BaseThrottle):

def allow\_request(self, request, view):

# Obter a chave da requisição

api\_key = request.query\_params.get('api\_key')

if not api\_key:

return False # Bloqueia se não houver API Key

# Verificar se a chave atingiu o limite (simulação)

if api\_key == "bloqueado":

return False

return True

def wait(self):

# Tempo de espera até a próxima requisição permitida (opcional)

return 60

Use essa classe na view:

class APIKeyProtectedView(APIView):

throttle\_classes = [APIKeyThrottle]

def get(self, request):

return Response({"message": "Requisição permitida!"})

**Resumo**

| **Classe** | **Limita Por** | **Usado Para** |
| --- | --- | --- |
| AnonRateThrottle | IP do cliente | Controlar requisições de usuários anônimos. |
| UserRateThrottle | ID do usuário | Limitar requisições de usuários autenticados. |
| ScopedRateThrottle | Escopo configurado | Limites por endpoint ou conjunto de endpoints. |
| **Custom Throttle** | Lógica personalizada | Casos específicos (ex.: por chave API). |

* O **throttling** protege sua API contra uso excessivo, abusos e sobrecarga.
* Classes padrão cobrem casos comuns, mas podem ser personalizadas conforme necessário.
* Configure limites apropriados no settings.py e implemente lógicas específicas usando **throttles personalizados**.