



PYTHON
ACADEMY

COMO CRIAR MIDDLEWARES NO DJANGO (PYTHON)

Aprenda a criar Middlewares no Django para interceptar e processar requisições vindas dos usuários.

[PYTHONACADEMY.COM.BR](https://pythonacademy.com.br)

Gere ebooks como este com



em <https://ebookr.ai>

Crie ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!



Capas gerados por IA



Infográficos feitos por IA



Edite em Markdown em Tempo Real



Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

✓ **Artigo atualizado para Django 5.1** (Dezembro 2025) *Middlewares permanecem essenciais. Django 5.1 traz suporte aprimorado para async middlewares.*

Fala **Dev!** Na paz?!

Nesse post vamos aprender a criar *Middlewares* no Django, que são trechos de código para tratamento prévio ou posterior das requisições que chegam e saem do seu sistema Django.

Ficou confuso? Então bora pro post! 😊

Mas antes, um aviso!

Esse post faz parte do contexto da **Série Django** aqui da Python Academy!

Já se liga nos outros posts:

[Django: Introdução ao *framework*](#)

[Django: A Camada *Model*](#)

[Django: A Camada *View*](#)

[Django: A Camada *Template*](#)

Agora sim, bora!

O que são *Middlewares*

Middlewares são trechos de códigos que podem ser executados antes ou depois do processamento de requisições/respostas pelo Django.

É uma forma que os desenvolvedores, nós, temos para alterar como o Django processa algum dado de entrada ou de saída.

💡 **Django 5.1:** Suporte completo para **async middlewares**, permitindo operações assíncronas não-bloqueantes. Ideal para aplicações de alta concorrência.

Se você olhar no arquivo `settings.py`, nós temos a lista `MIDDLEWARE` com diversos *middlewares* pré-configurados:

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

Por exemplo, o *middleware* `AuthenticationMiddleware` é responsável por adicionar a variável `user` a todas as requisições.

Dessa forma, você pode, por exemplo, mostrar o usuário logado no seu *template*:

```
{% raw %}  
<li>  
    <a href="{% url 'profile' id=user.id %}">Olá, {{ user.email }}</a>  
</li>  
{% endraw %}
```

Vamos ver agora o ciclo de vida de um *Middleware* dentro do Django.

Ciclo de vida de um *Middleware* no Django

Um *middleware* é um método *callable* (que implementa o método `__call__()`).

Esse método recebe uma **requisição** e retorna uma **resposta**, assim como uma *View*, podendo ser escrito como **função** ou como **classe**.

Desde Django 3.1+, middlewares assíncronos são totalmente suportados, permitindo melhor performance em aplicações que usam `async/await`.

Um exemplo de *middleware* escrito como função é:

E como Classe:

Como cada *Middleware* é executado de maneira encadeada, do topo da configuração `MIDDLEWARE` (do *settings.py*) para o fim, a saída de um é a entrada do próximo.

O método `get_response()` pode ser a própria *View*, caso ela seja a última configurada no `MIDDLEWARE` do *settings.py*, ou o próximo *middleware* da cadeia.

Utilizando a construção do *middleware* via Classe, nós temos três métodos importantes, que veremos em seguida!

O método `process_view`

Assinatura: `process_view(request, func, args, kwargs)`

Esse método é chamado logo antes do Django executar a *View* que vai processar a requisição e possui os seguintes parâmetros:

- `request` é o objeto `HttpRequest`.

- `func` é a própria *view* que o Django está para chamar ao final da cadeia de *middlewares*.
- `args` é a lista de parâmetros posicionais que serão passados à *view*.
- `kwargs` é o *dict* contendo os argumentos nomeados (*keyword arguments*) que serão passados à *view*.

Esse método deve retornar `None` ou um objeto `HttpResponse` :

- Caso retorne `None`, o Django entenderá que deve continuar a cadeia de *Middlewares*.
- Caso retorne `HttpResponse`, o Django entenderá que a resposta está pronta para ser enviada de volta e não vai se preocupar em chamar o resto da cadeia de *Middlewares*, nem a *view* que iria processar a requisição.

O método `process_exception`

Assinatura: `process_exception(request, exception)`

Esse método é chamada quando uma *view* lança uma exceção e deve retornar ou `None` ou `HttpResponse`. Caso retorne um objeto `HttpResponse`, o Django irá aplicar o *middleware* de resposta e o de *template*, retornando a requisição ao *browser*.

- `request` é o objeto `HttpRequest`
- `exception` é a exceção propriamente dita lançada pela *view* (`Exception`).

O método `process_template_response`

Assinatura: `process_template_response(request, response)`

Esse método é chamado logo após a *view* ter terminado sua execução, caso a resposta tenha uma chamada ao método `render()` indicando que a resposta possui um *template*.

Possui os seguintes parâmetros:

- `request` é um objeto `HttpRequest`.
- `response` é o objeto `TemplateResponse` retornado pela *view* ou por outro *middleware*.

Agora vamos criar um *middleware* um pouco mais complexo para exemplificar o que foi dito aqui!

Ah, antes de seguir! Está curtindo esse conteúdo? Que tal levá-lo pra onde quiser?

Então já baixe nosso **ebook GRÁTIS de Desenvolvimento Web com Python e Django!**



Criando um *Middleware* no Django

Vamos supor que queremos um *middleware* que filtre requisições e só processe aquelas que venham de uma determinada lista de IP's.

O que precisamos fazer é abrir o cabeçalho de todas as requisições que chegam no nosso servidor e verificar se o IP de origem bate com a nossa lista de IP's.

Para isso, colocamos a lógica no método `process_view`, da seguinte forma:


```

class FiltraIPMiddleware:

    def __init__(self, get_response=None):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        return response

    def process_view(request, func, args, kwargs):
        # Lista de IPs autorizados
        ips_autorizados = ['127.0.0.1']

        # IP do usuário
        ip = request.META.get('REMOTE_ADDR')

        # Verifica se o IP do cliente está na lista de IPs autorizados
        if ip not in ips_autorizados:
            # Se usuário não autorizado > HTTP 403: Não Autorizado
            return HttpResponseForbidden("IP não autorizado")

        # Se for autorizado, não fazemos nada
        return None

```

Depois disso, precisamos registrar nosso *middleware* no arquivo de configurações *settings.py* (na configuração `MIDDLEWARE`):

```
MIDDLEWARE = [
    # Middlewares do próprio Django
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # Nosso Middleware
    'helloworld.middlewares.FiltraIPMiddleware',
]
```

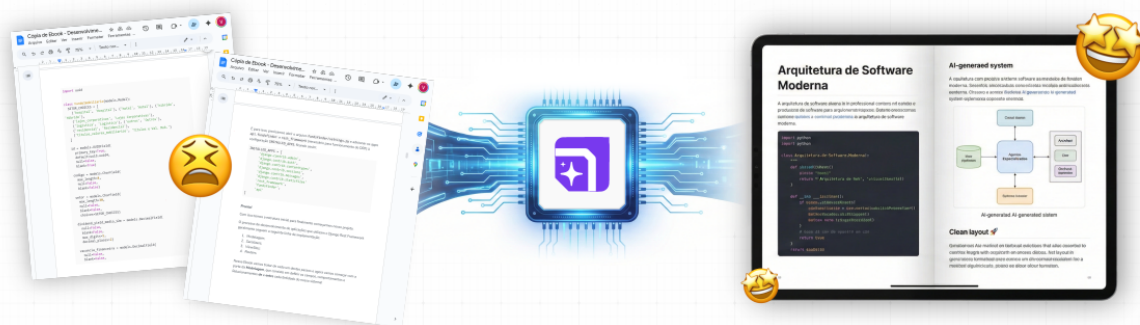
Agora, podemos testar seu funcionamento alterando a lista `ips_autorizados`:

- Coloque `ips_autorizados = ['127.0.0.1']` e tente acessar alguma URL da nossa aplicação: devemos conseguir acessar normalmente nossa aplicação, pois como estamos executando o servidor localmente, nosso IP será 127.0.0.1 e, portanto, passaremos no teste.
- Coloque `ips_autorizados = []` e tente acessar alguma URL da nossa aplicação: deve aparecer a mensagem de “**IP não autorizado**”, pois nosso IP (127.0.0.1) não está autorizado a acessar o servidor.



Estou desenvolvendo o [Ebookr.ai](https://ebookr.ai), uma plataforma que transforma suas ideias em ebooks profissionais usando IA — com geração de capa, infográficos e exportação em PDF. Te convido a conhecer!


Crie Ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

... e deixe que nossa IA faça o trabalho pesado!

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS 

 Capas gerados por IA

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

Conclusão

Nesse *post* vimos como é simples criar *Middlewares* no Django.

Vimos como podemos adicionar tratamento prévio ao processamento de requisições e posterior ao processamento de respostas.

Seus casos de uso são bem específicos, contudo é bom saber que temos essa possibilidade!

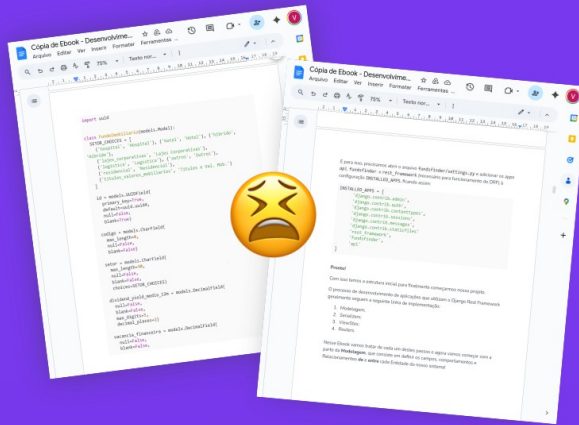
É isso dev, nos vemos na próxima! 😊

Não se esqueça de conferir!



Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA



Infográficos feitos por IA



Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado



Edite em Markdown em Tempo Real

TESTE AGORA



PRIMEIRO CAPÍTULO 100% GRÁTIS