

# Por dentro do Django

Jorge Enrique  
Gabriel Melo  
Yuri Nogueira  
Giovanni Beralde

# Sobre Django

O Django é um framework web de alto nível em Python, criado em 2003 para aplicações jornalísticas online. Destaca-se pela simplicidade, eficiência e filosofia "baterias incluídas", com funcionalidades prontas para uso. Lançado como software livre em 2005, evoluiu com novos recursos e extensões da comunidade. Amplamente utilizado pela facilidade, documentação e eficiência no desenvolvimento de aplicações web complexas.

## Seus criadores



Adrian Holovaty



Simon Willison

# Empresas que utilizam o Django;



Instagram



Pinterest



Spotify



FireFox

# Quando usar o Django

O Django é um framework web Python que oferece um ambiente poderoso para o desenvolvimento de aplicações web e pode ser útil para :



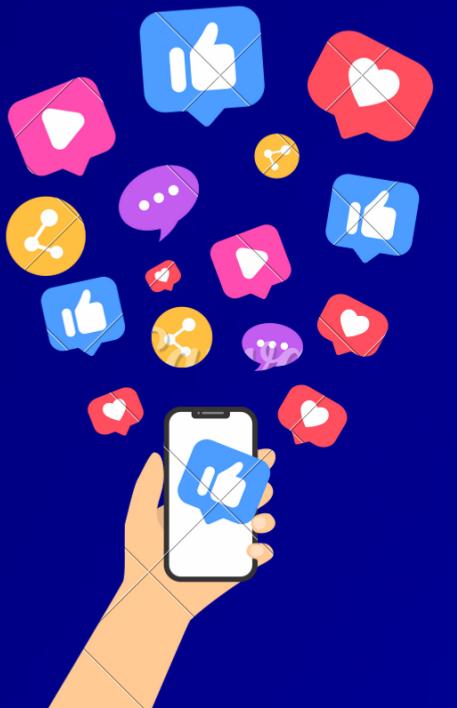
# Quando usar o Django

**1º Criação de sites web** - Ele foi projetado especificamente para essa finalidade. Com o Django, você pode desenvolver desde simples sites informativos até sistemas complexos e interativos



# Quando usar o Django

**2º Plataformas de redes sociais** - Muitas funcionalidades, como perfis de usuários, feeds de notícias e interações, podem ser implementadas com o Django



# Quando usar o Django

**3º Ambiente simplificado** - O Django abstrai processos repetitivos de configurações de ambiente de desenvolvimento, como gerenciamento de banco de dados, configuração de roteamento de URLs, autenticação e autorização



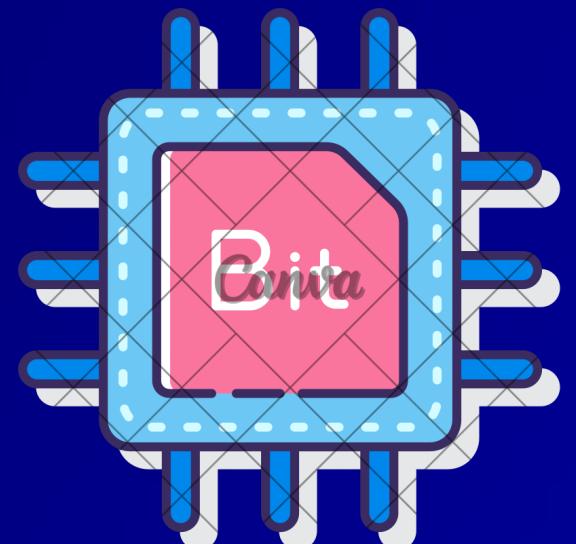
# Quando usar o Django

Se você deseja criar aplicações web escaláveis, com design limpo e desenvolvimento rápido, o Django é uma excelente escolha !



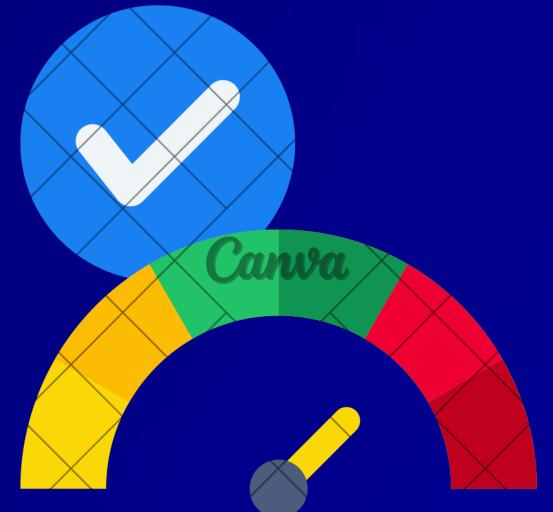
# Quando NÃO usar o Django

**1º Projetos pequenos e simples** - se você está criando um site ou aplicação web muito pequeno, com poucas páginas e funcionalidades básicas, frameworks mais leves podem ser mais adequados



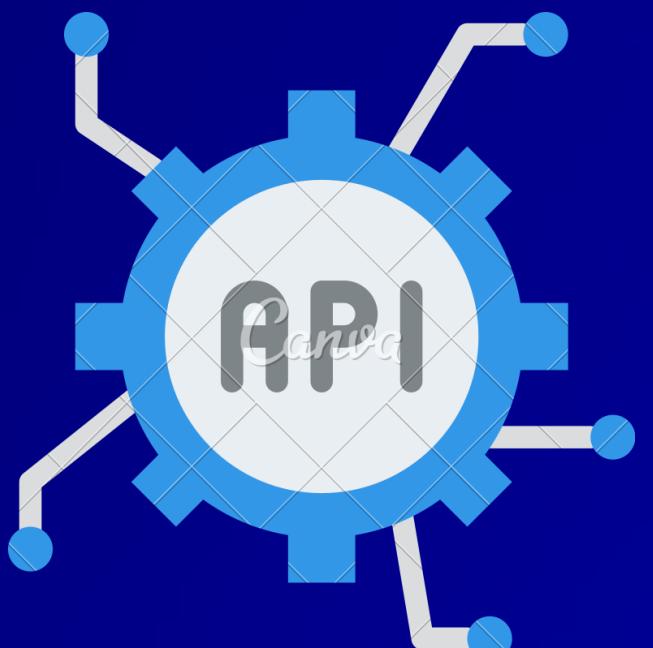
# Quando NÃO usar o Django

**2º Performance extrema:** Se o seu projeto exige extrema otimização de performance, o Django pode não ser a melhor opção. Frameworks mais minimalistas ou linguagens de programação de baixo nível podem ser mais adequados para esses cenários como Go, Rust ou C++



# Quando NÃO usar o Django

**3º APIs ultraleves:** Se você está construindo uma API ultraleve (por exemplo, para microserviços), o Django pode ser excessivo. Frameworks como o FastAPI ou Flask podem ser mais adequados para APIs simples e rápidas.



# Autenticação e administração do sistema

O Django fornece um sistema de autenticação e autorização bastante completo, que permite o gerenciamento de usuários, grupos e permissões



# Autenticação

**Autenticação:** A autenticação é o processo de verificar a identidade de um usuário. Quando um usuário fornece um nome de usuário e senha, o sistema de autenticação do Django verifica se a combinação é válida. Se for válida, o usuário é autenticado e liberado para acessar outras páginas.



# Autenticação

## exemplo de autenticação:

```
from django.contrib.auth import authenticate, login
from django.http import HttpResponseRedirect

def login_view(request):
    # Verifica se a requisição é do tipo POST
    if request.method == 'POST':
        # Obtém o nome de usuário e a senha do formulário de login
        username = request.POST['username']
        password = request.POST['password']

        # Autentica o usuário
        user = authenticate(request, username=username, password=password)

        # Se o usuário é autenticado com sucesso, faz o login e redireciona para a página inicial
        if user is not None:
            login(request, user)
            return HttpResponseRedirect('Usuário logado com sucesso.')

        # Se a autenticação falhar, retorna uma mensagem de erro
        else:
            return HttpResponseRedirect('Falha na autenticação. Por favor, tente novamente.')

    # Se a requisição não é do tipo POST, retorna uma mensagem de erro
    else:
        return HttpResponseRedirect('Método inválido. Use um método POST.')
```

# Administração do Sistema

**Administração:** O Django vem com um módulo de administração pronto para uso que fornece uma interface de usuário para administrar o conteúdo do site. Os administradores do site podem usar essa interface para adicionar, alterar e excluir registros no banco de dados.



# Administração do Sistema

Este código define um modelo chamado Blog com dois campos: title para o título do blog e content para o conteúdo do blog. Isso permite que você crie objetos Blog no seu aplicativo Django e armazene esses dados no banco de dados..

**models.py:**

```
from django.db import models

# A classe Blog é definida como um modelo Django.
# Um modelo Django é a representação de uma tabela de banco de dados em Python.
class Blog(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
```

# Administração do Sistema

Uma vez que você tenha executado essa linha de código, poderá acessar o modelo Blog através do painel administrativo do Django. Você poderá visualizar, adicionar, editar e excluir instâncias do modelo Blog diretamente através do navegador.

admin.py:

```
from django.contrib import admin
from .models import Blog

# Registra o modelo Blog no módulo de administração
admin.site.register(Blog)
```

# Principais Características

- MVT (Model-View-Template): Padrão arquitetural semelhante ao MVC, com uma abordagem um pouco diferente.
- Admin Site: Interface administrativa automática gerada para gerenciar modelos de dados.
- ORM (Object-Relational Mapping): Permite interagir com o banco de dados através de classes Python.
- Formulários: Facilita a criação e validação de formulários HTML.
- Template Engine: Sistema de templates que simplifica a criação de páginas web.

# Como utilizar Django

1. Instalação: **pip install django**
2. Criar um projeto: **django-admin startproject nome\_projeto**
3. Criar uma aplicação: **python manage.py startapp nome\_aplicacao**
4. Definir modelos de dados em **models.py**
5. Criar as migrações: **python manage.py makemigrations** e **python manage.py migrate**
6. Criar views em **views.py** e URLs em **urls.py**
7. Executar o servidor de desenvolvimento: **python manage.py runserver**

# Aplicações Models Django

O models tem como objetivo tornar a codificação mais simples e prática através de suas ferramentas, de certa forma permitindo que seja feita de forma mais intuitiva, auxiliando o CRUD, além de ajudar no gerenciamento do Django de forma mais eficiente e conveniente."

# exemplo models

```
from django.db import models

class User(models.Model):

    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=100)

    def __str__(self):
        return self.username
```

# exemplo models

Esse Código tem como objetivo servir como um pequeno banco de dados para as informações do usuário de para que possam ser usadas para adicionar novos processos de autenticação. Além disso, ele possui ferramentas que julgam as informações do usuário, atuando como um pequeno gerenciador de dados.

# Variaveis dinamicas

Variáveis dinâmicas são um tipo de variável que permite que um valor seja alterado durante a execução do programa. Entre as linguagens de programação, o Python oferece uma maior liberdade, que é aprimorada pelo Django, que não necessita de vastas especificações para criação de webs.

# Variável dinâmica {{nome}}

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemplo de Variável</title>
</head>
<body>
    <h1>Olá, {{ nome }}!</h1>
</body>
</html>
```

Esse é um exemplo Python simples que usa {{nome}} para substituir o nome. Quando o template for aberto no navegador, {{nome}} será substituído pelo valor que quisermos.

# Aplicações Template Django

Os templates do Django são fundamentais para a manutenção de uma página web, oferecendo otimização e flexibilidade para sua evolução.

Eles possibilitam um ambiente mais limpo e organizado, facilitando mudanças posteriores e simplificando o desenvolvimento com suas ferramentas.

# Template {% for %} Django

O {% for %} tem como função criar uma interação com uma lista HTML.

Ela serve para criar loops que mostra pro django que queremos percorrer todos os itens de uma lista, de acordo com como você o utiliza no código.

# exemplo { % for %}

```
{% for item in lista %}  
    {{ item }}  
{% endfor %}
```

O {{ item }} é uma variável temporária que serve para representar cada item desta lista. Durante o loop, essa variável é usada para acessar e exibir os dados de cada item, mas ela não retém esses dados por muito tempo só ate o loop acabar. É tambem e onde cada item é inserido para ser utilizado na página da web.

# Template { % if % }

O { % if % } é usada para realizar verificações condicionais para determinar um resultado desejado de acordo com as especificações do autor do código, assim verificando uma ampla quantidade de informações de acordo com seu código.

# Exemplo { % if % }

```
{% if product.price < 50 %}  
    <p>Este é um produto com preço especial!</p>  
{% endif %}
```

Ele ira checar cada item da lista protudo e quando for ativado ira imprimir a mensagem “Este é um produto com preço especial” ele e muito utilizado para fazer verificação de dados para paginas de login e outras que tenham que interagir com itens com caracteristicas especificas.

# Template {% block %}

```
{% extends 'base.html' %}

{% block title %}Minha Página{% endblock %}

{% block content %}
    <h2>Bem-vindo à Minha Página</h2>
    <p>Esta é uma página específica do meu site.</p>
{% endblock %}
```

O {% block %} serve para criar áreas em um arquivo que podem ser alteradas ou ampliadas dentro de template filho. É uma maneira eficiente de permitir o uso de partes específicas de templates, pois essas partes do código podem ser definidas como blocos que podem ser adicionados.

# Template { % include % }

O { % include % } no Django tem como objetivo incluir outros arquivos dentro do arquivo principal do template, permitindo que você economize tempo e mantenha o código mais organizado, evitando repetições desnecessárias.

# Exemplo { % include % }

```
<html>
<head>
    <title>Minha Página</title>
</head>
<body>
    {% include 'header.html' %}

    <main>
        <p>Conteúdo da minha página.</p>
    </main>

    {% include 'footer.html' %}
</body>
</html>
```

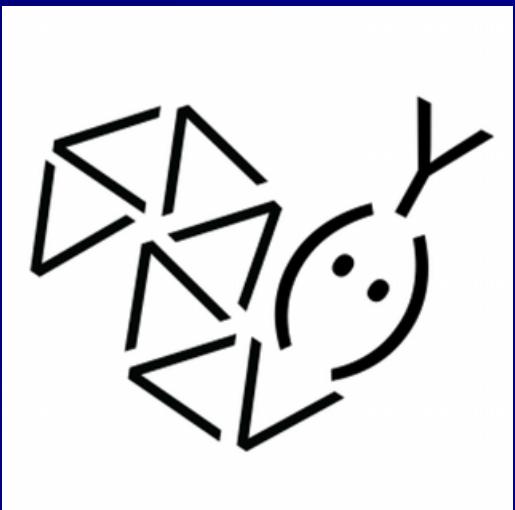
Quando você usa o código "**{% include 'header.html' %}**" e "**{% include 'footer.html' %}**", ele irá pegar o conteúdo dos arquivos "header.html" e "footer.html" e incluí-los nos locais escolhidos no template, sem precisar de mudanças tão drásticas e extensas no código.

# Exemplo de Hospedagem Gratuita

Existem várias plataformas onde você pode hospedar gratuitamente suas aplicações Django. Aqui estão algumas delas:

# Exemplo de Hospedagem Gratuita

PythonAnywhere: É uma plataforma baseada em nuvem que permite ter uma instância de servidor para todas as suas necessidades de desenvolvimento Python<sup>1</sup>. Você pode configurar um servidor web totalmente funcional com apenas alguns cliques



# Exemplo de Hospedagem Gratuita

A2 Hosting: Oferece servidores Super que carregam páginas até 20 vezes mais rápido do que o hosting padrão



# Exemplo de Hospedagem Gratuita

Vercel: oferece um plano gratuito chamado “Hobby” que pode ser usado para hospedar suas aplicações<sup>1</sup>. Este plano inclui suporte para mais de 35 frameworks, integração automática de CI/CD com Git, funções serverless e uma base de dados inicial<sup>1</sup>.



# vercel

1- Após a conclusão do código, você deve ir ao arquivo `settings.py` que fica localizada em seu projeto e na dentro da função `ALLOWED_HOSTS[]` colocar sua extensão do seu framework (nesse caso colocar `.vercel.app`)

```
19  
20 ALLOWED_HOSTS = [".vercel.app"]  
21
```



# vercel

**2- Em seguida, crie um novo arquivo chamado vercel.json na pasta raiz do seu projeto. Este arquivo irá conter as configurações do seu projeto Vercel. Você pode usar o seguinte modelo para o seu arquivo vercel.json:**

```
setup > {} vercel.json > ...
1  {
2    "builds": [
3      {
4        "src": "djangoprojectname/wsgi.py",
5        "use": "@vercel/python",
6        "config": { "maxLambdaSize": "15mb", "runtime": "python3.9" }
7      },
8      {
9        "src": "/(.*)",
10       "dest": "djangoprojectname/wsgi.py"
11     }
12   ]
13 }
```



# vercel

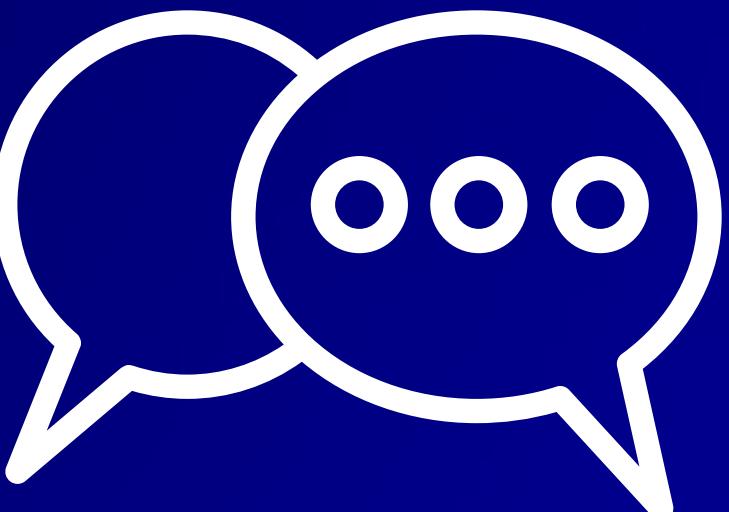
3-No arquivo `wsgi.py`, você deve definir a variável `app` como o objeto `application`. Isso irá garantir que todas as aplicações que você criou sejam apresentadas como `app`. Você pode usar o seguinte código no seu arquivo `wsgi.py`:

```
16     application = get_wsgi_application()  
17       
18     app = application
```



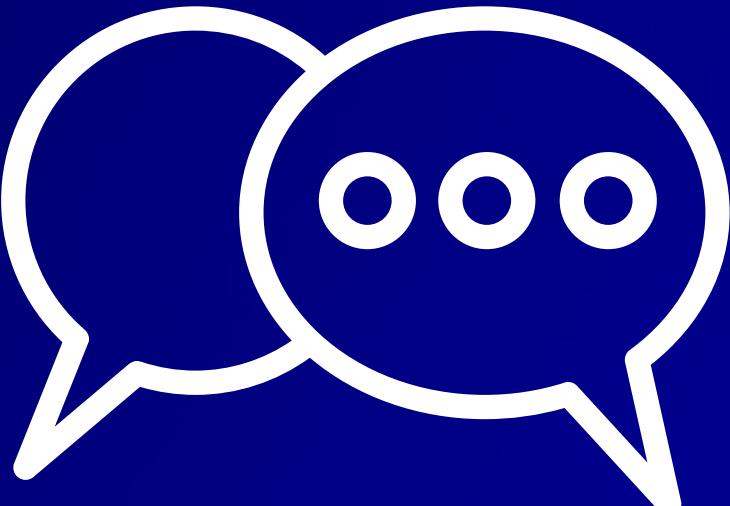
# como ele “conversa” com o DB e o front-end

O Django se comunica com o banco de dados e o frontend de maneiras diferentes:



# como ele “conversa” com o DB

Comunicação com o Banco de Dados: O Django usa um ORM (Object-Relational Mapping) para interagir com o banco de dados. O ORM permite que você interaja com seu banco de dados, como se estivesse usando SQL. Mas, em vez disso, você usa Python



# como ele “conversa” com o DB

Vamos supor que temos um modelo chamado Blog

Definindo um modelo:

```
from django.db import models

# A classe Blog é definida como um modelo Django.
# Um modelo Django é a representação de uma tabela de banco de dados em Python.
class Blog(models.Model):
    # title é definido como um campo CharField, que é traduzido em um campo VARCHAR no banco de
    # dados.
    title = models.CharField(max_length=200)
    # content é definido como um campo TextField, que é traduzido em um campo TEXT no banco de dados.
    content = models.TextField()
```

# como ele “conversa” com o DB

Vamos supor que temos um modelo chamado Blog

Criando um novo registro no banco de dados:

```
# Uma nova instância do modelo Blog é criada.  
# Os argumentos passados para Blog() são usados para inicializar os campos correspondentes.  
blog = Blog(title='Meu primeiro blog', content='Este é o conteúdo do meu primeiro blog.')  
# O método save() é chamado na instância do modelo.  
# Isso cria um novo registro no banco de dados correspondente à instância do modelo.  
blog.save()
```

# como ele “conversa” com o DB

Vamos supor que temos um modelo chamado Blog

Buscando registros do banco de dados:

```
# O método all() do manager objects do modelo Blog é chamado.  
# Isso retorna um QuerySet representando todos os registros do banco de dados para o modelo Blog.  
blogs = Blog.objects.all()  
  
# O método first() do manager objects do modelo Blog é chamado.  
# Isso retorna o primeiro registro do banco de dados para o modelo Blog.  
first_blog = Blog.objects.first()  
  
# O método get() do manager objects do modelo Blog é chamado com um argumento de palavra-chave.  
# Isso retorna o (class) Blog anco de dados para o modelo Blog que corresponde ao argumento de  
palavra-chave.  
specific_blog = Blog.objects.get(title='Meu primeiro blog')
```

# como ele “conversa” com o DB

Vamos supor que temos um modelo chamado Blog

Atualizando um registro no banco de dados:

```
# Um registro específico do banco de dados para o modelo Blog é recuperado.  
blog = Blog.objects.get(title='Meu primeiro blog')  
# O campo content da instância do modelo é atualizado.  
blog.content = 'Este é o conteúdo atualizado do meu primeiro blog.'  
# O método save() é chamado na instância do modelo.  
# Isso atualiza o registro do banco de dados correspondente à instância do modelo.  
blog.save()
```

# como ele “conversa” com o DB

Vamos supor que temos um modelo chamado Blog

Excluindo um registro do banco de dados:

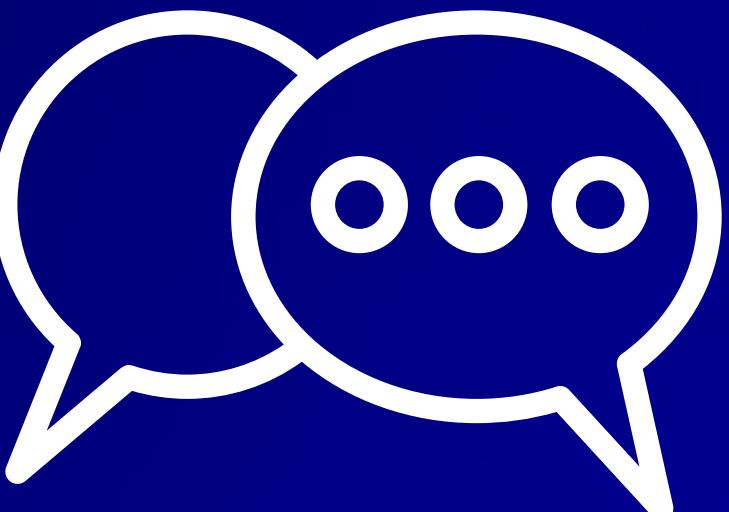
```
# Um registro específico do banco de dados para o modelo Blog é recuperado.  
blog = Blog.objects.get(title='Meu primeiro blog')  
# O método delete() é chamado na instância do modelo.  
# Isso exclui o registro do banco de dados correspondente à instância do modelo.  
blog.delete()
```

# como ele “conversa” com o DB

Esses códigos demonstram como o Django usa o ORM para interagir com o banco de dados. Você pode realizar operações CRUD (Criar, Ler, Atualizar, Deletar) no banco de dados usando Python, em vez de escrever consultas SQL diretamente. Isso torna o código mais legível e fácil de manter.

# como ele “conversa” com o front-end

Comunicação com o Frontend: O Django pode gerar HTML dinamicamente usando o sistema de templates. Você escreve templates usando uma combinação de HTML, tags de template do Django e variáveis de contexto<sup>2</sup>. As variáveis de contexto são passadas de sua view para o template



# como ele “conversa” com o front-end

Vamos supor que temos uma view chamada blog\_view e um template chamado blog.html.

views.py:

```
from django.shortcuts import render
from .models import Blog

def blog_view(request):
    # Buscando todos os blogs do banco de dados
    blogs = Blog.objects.all()

    # As variáveis de contexto são passadas para o template
    context = {'blogs': blogs}

    # A função render combina o template com o contexto e retorna o HTML resultante
    return render(request, 'blog.html', context)
```

# como ele “conversa” com o front-end

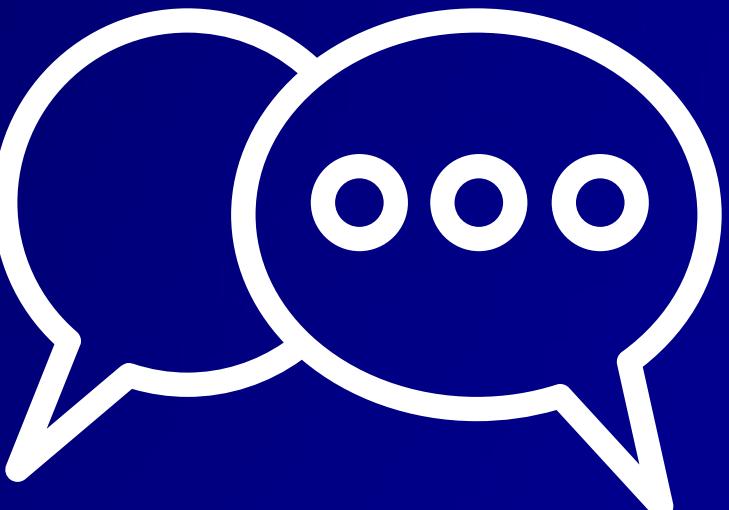
Vamos supor que temos uma view chamada blog\_view e um template chamado blog.html.

blog.html:

```
{% for blog in blogs %} <!-- Inicia um loop for que percorre cada 'blog' na lista de 'blogs' -->
|   <h2>{{ blog.title }}</h2> <!-- Exibe o título do 'blog' atual dentro de uma tag h2 -->
|   <p>{{ blog.content }}</p> <!-- Exibe o conteúdo do 'blog' atual dentro de uma tag p -->
{% endfor %} <!-- Termina o loop for -->
```

# como ele “conversa” com o front-end

No arquivo `views.py`, a função `blog_view` é definida. Esta função busca todos os blogs do banco de dados e os passa para o template `blog.html` através do dicionário `context`.



# Desenvolvimento do Projeto Django

