

# COMO CRIAR FORMULÁRIOS NO DJANGO (PYTHON)

Vamos conhecer a facilidade que é criar formulários em projetos Django utilizando Django Forms!

Gere ebooks como este com



Ebookr

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



Edita em Markdown em Tempo Real



Adicione Banners Promocionais

TESTE AGORA



PRIMEIRO CAPÍTULO 100% GRÁTIS

 **Artigo atualizado para Django 5.1** (Dezembro 2025) Django Forms e ModelForms permanecem estáveis. Validações e widgets funcionam perfeitamente.

Salve salve **Pythonista!**

Ou seria *Djangista*? Péssimo, esquece... 😊

Hoje vamos aprender a desenvolver Formulários web incríveis utilizando Django Forms!

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](#)

## Processamento de Formulários HTML no Django

O tratamento de formulários é uma tarefa que pode ser **bem complexa**.

Considere um formulário com diversos campos e diversas regras de validação: seu tratamento não é mais um processo simples.

Os *Forms* do Django são formas de descrever os elementos `<form>...</form>` das páginas HTML, simplificando e automatizando o processo de validação.

O Django trata três partes distintas dos formulários:

- Preparação dos dados tornando-os prontos para renderização
- Criação de formulários HTML para os dados
- Recepção e processamento dos formulários enviados ao servidor

Basicamente, queremos uma forma de renderizar em nosso *template* o código HTML:

```
<form action="/insere-funcionario/" method="post">
    <label for="nome">Your name: </label>
    <input id="nome" type="text" name="nome" value="">
    <input type="submit" value="Enviar">
</form>
```

Que, ao ser submetido ao servidor, tenha seus campos de entrada validados e inseridos no banco de dados.

No centro desse sistema de formulários do Django está a classe `Form`.

Nela, nós descrevemos os campos que estarão disponíveis no formulário HTML e os métodos de validação.

Para o formulário acima, podemos descrevê-lo da seguinte forma.

```
from django import forms

class InsereFuncionarioForm(forms.Form):
    nome = forms.CharField(
        label='Nome do Funcionário',
        max_length=100
    )
```

Nesse formulário:

- Utilizamos a classe `forms.CharField` para descrever um campo de texto.
- O parâmetro `label` descreve um rótulo para esse campo.
- `max_length` decreve o tamanho máximo que esse *input* pode receber (100 caracteres, no caso).

Existem diversos tipos de campos possíveis de serem utilizados, por exemplo:

- `BooleanField`: mapeia um campo booleano, resultando em um `checkbox` no HTML final.
- `DecimalField`: mapeia um campo decimal, resultando em um `<input type='number' ...>` no HTML final.
- `EmailField`: mapeia um campo de email, resultando em um `<input type='email' ...>` no HTML final.

Veja os diversos tipos de campos disponíveis [acessando aqui](#)

A classe `forms.Form` possui um método muito importante, chamado `is_valid()`.

Quando um formulário é submetido ao servidor, esse é um dos métodos que irá realizar a validação dos campos do formulário.

Se tudo estiver **OK**, ele colocará os dados do formulário no atributo `cleaned_data` (que pode ser acessado por você posteriormente para pegar alguma informação - como o nome que foi inserido pelo usuário no campo `<input name='nome'>` ).

Como o processo de validação do Django é bem complexo e para não prolongar muito o *post*, [acesse a documentação aqui](#) para saber mais.

## Criando seu primeiro Form

Vamos ver agora um exemplo mais complexo com um formulário de inserção de uma entidade exemplo Funcionário, com todos os campos.

Primeiro, vamos criar a seguinte modelagem da entidade `Funcionario` no arquivo `models.py` de sua aplicação:

```
from django.db import models

class Funcionario(models.Model):

    nome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    sobrenome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    cpf = models.CharField(
        max_length=14,
        null=False,
        blank=False
    )

    tempo_de_servico = models.IntegerField(
        default=0,
        null=False,
        blank=False
    )

    remuneracao = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        null=False,
        blank=False
    )
```

Em seguida, vamos criar um arquivo `forms.py` para guardar os formulários de nossa aplicação. Crie-o no mesmo `app` que criou o modelo `Funcionário`.

Dessa forma, e consultando a [documentação](#) dos possíveis campos do nosso formulário, nós podemos descrever um formulário de inserção da seguinte forma:

```
from django import forms

class InsereFuncionarioForm(forms.Form):
    nome = forms.CharField(
        required=True,
        max_length=255
    )

    sobrenome = forms.CharField(
        required=True,
        max_length=255
    )

    cpf = forms.CharField(
        required=True,
        max_length=14
    )

    tempo_de_servico = forms.IntegerField(
        required=True
    )

    remuneracao = forms.DecimalField(
        required=True
    )
```

Affff, o *Model* e o *Form* são quase iguais... Terei que reescrever os campos toda vez?  


**Claro que não, jovem!** Pra isso o Django criou o incrível *ModelForm* ! 😊

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 Forms com ModelForm

Com o `ModelForm` nós descrevemos os campos que queremos (atributo `fields`) e/ou os campos que não queremos (atributo `exclude`).

Para isso, utilizamos a classe interna `Meta` para incluirmos esses metadados na nossa classe.

Metadado (no caso do `Model` e do `Form`) é tudo aquilo que não será transformado em campo, como `model`, `fields`, `ordering` etc ([mais sobre Meta options](#))

Nosso `ModelForm`, pode ser descrito da seguinte forma:

```
from django import forms

class InsereFuncionarioForm(forms.ModelForm):
    class Meta:
        # Modelo base
        model = Funcionario

        # Campos que estarão no form
        fields = [
            'nome',
            'sobrenome',
            'cpf',
            'remuneracao'
        ]

        # Campos que não estarão no form
        exclude = [
            'tempo_de_servico'
        ]
```

**Viu que simples!** 😊

Podemos utilizar apenas o campo `fields`, apenas o campo `exclude` ou os dois juntos.

Mesmo utilizando os atributos `fields` e `exclude`, ainda podemos adicionar outros campos, independente dos campos do modelo `Funcionário`.

O resultado será um formulário com todos os campos presentes no `fields`, menos os campos do `exclude` mais os campos adicionados.

Ficou confuso? Então vamos ver o exemplo:

```

from django import forms

class InsereFuncionarioForm(forms.ModelForm):

    chefe = forms.BooleanField(
        label='Chefe?',
        required=True,
    )

    biografia = forms.CharField(
        label='Biografia',
        required=False,
        widget=forms.TextArea
    )

    class Meta:
        # Modelo base
        model = Funcionario

        # Campos que estarão no form
        fields = [
            'nome',
            'sobrenome',
            'cpf',
            'remuneracao'
        ]

        # Campos que não estarão no form
        exclude = [
            'tempo_de_servico'
        ]

```

Isso vai gerar um formulário com:

- Todos os campos contidos em `fields` menos os campos contidos em `exclude`
- O campo `chefe`, renderizado como um `checkbox` (`<input type='checkbox' name='chefe' ...>`)

• Uma área de texto para biografia  
( <textarea name='biografia' ...></textarea> )

**Mas como biografia será um <textarea .../> se ele é um CharField ?**

Calma lá.... Você vai descobrir na seção BÔNUS! 😊

💡 Estou construindo o **Ebookr.ai**, uma plataforma onde você cria ebooks profissionais com IA sobre qualquer assunto — do zero ao PDF pronto, com capas e infográficos gerados automaticamente. Dá uma olhada!

**Ebookr**

**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

# Juntando tudo

Poderíamos criar uma `CreateView` para renderizar esse `Form` da seguinte forma:

```
from django.views.generic import CreateView
from django.urls import reverse_lazy

class FuncionarioCreateView(CreateView):
    template_name = "website/cria.html"
    model = Funcionario
    form_class = InsereFuncionarioForm
    success_url = reverse_lazy("website:lista_funcionarios")
```

Nela:

- `template_name` : Aponta para o HTML base para renderização do form.
- `model` : O modelo associado à essa `View`.
- `form_class` : `Form` que será disponibilizado no *template*, definido ali em cima.
- `success_url` : URL para redirecionar o usuário, em caso de sucesso.

Já para o template HTML, temos duas estratégias:

- Deixar que o Django renderize tudo: aqui você não tem muito controle de como será o layout do resultado final.
- Construir o código HTML: eu prefiro essa opção para construir algo mais bonito no final.

Vamos ao resultado de ambas alternativas.

Na primeira, o código HTML poderia ser assim:

```

<form method="post">
    <!-- Não se esqueça dessa tag -->
    {% raw %}{% csrf_token %}{% endraw %}

    <hr>
    <div class="text-right">
        <a href="{% raw %}{% url 'website:lista_funcionarios' %}{% endraw %}" class="btn btn-outline-primary">Voltar</a>
        <button class="btn btn-primary">Enviar</button>
    </div>
</form>

```

O que resultaria no seguinte HTML, após renderizado:

**Cadastro de Funcionário**

Complete o formulário abaixo para cadastrar um novo Funcionário.

Nome	
Sobrenome	
CPF	
Tempo de Serviço	0
Remuneração	
Chefe?	<input type="checkbox"/>
Biografia	

**Voltar** **Enviar**

Pouco código, contudo...

**Meio “feio” pra mostrar pra um cliente, não concorda? 😊**

Já utilizando a segunda estratégia, poderíamos construir parte do HTML resultante, tendo um controle maior sobre o layout.

Para isso, gosto de utilizar a biblioteca [Django Widget Tweaks](#).

Ela possibilita adicionar classes CSS ao campos do formulário (coisa que o Django, por padrão, não permite).

Dessa forma, e usando um pouco de Bootstrap, podemos construir nosso formulário da seguinte forma:

```

<!-- Carrega as tags do Django Widget Tweaks para utilização -->
{% raw %}{% load widget_tweaks %}{% endraw %}

<form method="post">
    <!-- Não se esqueça dessa tag -->
    {% raw %}{% csrf_token %}{% endraw %}

    <!-- Nome -->
    <div class="input-group mb-3">
        <div class="input-group-prepend"><span class="input-group-text">Nome</span></div>
        {% raw %}{% render_field form.nome class+="form-control" %}{%
        endraw %}
    </div>

    <!-- Sobrenome -->
    <div class="input-group mb-3">
        <div class="input-group-prepend"><span class="input-group-text">Sobrenome</span></div>
        {% raw %}{% render_field form.sobrenome class+="form-control" %}{%
        endraw %}
    </div>

    <!-- CPF -->
    <div class="input-group mb-3">
        <div class="input-group-prepend"><span class="input-group-text">CPF</span></div>
        {% raw %}{% render_field form.cpf class+="form-control" %}{%
        endraw %}
    </div>

    <!-- Tempo de Serviço -->
    <div class="input-group mb-3">
        <div class="input-group-prepend"><span class="input-group-text">Tempo de Serviço</span></div>
        {% raw %}{% render_field form.tempo_de_servico class+="form-
control" %}{% endraw %}
    </div>

    <!-- Remuneração -->
    <div class="input-group mb-3">
        <div class="input-group-prepend"><span class="input-group-
text">Remuneração</span></div>

```

```

    {% raw %}{% render_field form.remuneracao class+="form-control" %}
    {% endraw %}

</div>

<!-- Chefe -->
<div class="input-group mb-3">
    <div class="input-group-prepend"><span class="input-group-text">Chefe?</span></div>
    {% raw %}{% render_field form.chefe class+="form-control" %}{%
    endraw %}
</div>

<!-- Biografia -->
<div class="input-group mb-3">
    <div class="input-group-prepend"><span class="input-group-text">Biografia</span></div>
    {% raw %}{% render_field form.biografia class+="form-control" %}{%
    endraw %}
</div>
<hr>
<div class="text-right">
    <a href="{% raw %}{% url 'website:lista_funcionarios' %}{%
    endraw %}" class="btn btn-outline-primary">Voltar</a>
    <button class="btn btn-primary">Enviar</button>
</div>
</form>

```

O que resultaria na seguinte tela:

Cadastro de Funcionário

Complete o formulário abaixo para cadastrar um novo **Funcionário**.

Nome

Sobrenome

CPF

Tempo de Serviço

Remuneração

Chefe?

Biografia

***Mais bonito pra colocar no portifólio, né?***

## BÔNUS: Alterando o comportamento padrão com Widgets

Assim como é possível definir atributos nos modelos, os campos do formulário também são customizáveis.

Veja que o campo `biografia` do nosso `InsereFuncionarioForm` é do tipo `CharField`, portanto deveria ser renderizado como um campo `<input type='text' ...>`.

Contudo, eu modifiquei o campo setando o atributo `widget` com `forms.TextArea`, assim:

```
biografia = forms.CharField(  
    label='Biografia',  
    required=False,  
    widget=forms.TextArea  
)
```

Assim, ele não mais será um simples *input*, mas será renderizado como um `<textarea></textarea>` no nosso *template*!

O Django possui diversos tipos de `Widgets` disponíveis, por exemplo:

- Podemos esconder um campo `CharField` com o `widget HiddenInput`.
- Podemos transformar um campo `CharField` para ser renderizado como campo de senha com o `widget PasswordInput`.
- Podemos usar o `widget FileInput` para fazer um campo para upload de arquivos!

As possibilidades são infinitas!

## Conclusão

Nesse *post*, vimos o poder dos **Django Forms** e sua flexibilidade!

Vimos como podemos construí-lo a partir da modelagem do nosso sistema através dos `ModelForms`.

Também vimos o quanto simples é alterar o comportamento de um campo do formulário através dos `Widgets`.

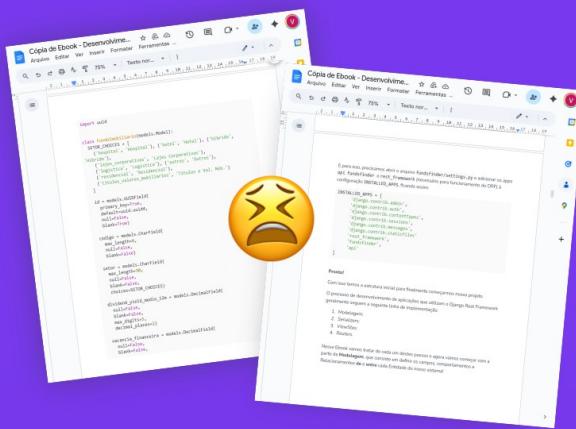
É isso por hoje dev! Nos vemos na próxima! 😊

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



**Adicione Banners Promocionais**

Deixe que nossa IA faça o trabalho pesado



 Edite em Markdown em Tempo Real

**TESTE AGORA**



 PRIMEIRO CAPÍTULO 100% GRÁTIS