

forms.py

O tratamento de formulário é uma tarefa que pode ser bem complexa. Considere um formulário com diversos campos e diversas regras da validação: seu tratamento não é um processo simples.

Os **forms** do Django são formas de descrever, em **código python**, os formulários das páginas HTML, simplificando e automatizando seu processo de criação e 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 seguinte código:

```
<form action="{% url 'funcionario' %}" method="post">
    <label for="nome">YOUR NAME:</label>
    <input id="nome" type="text" name="nome" value="">
    <input type="submit" value="Enviar">
</form>
```

E que, ao ser submetido ao servidor, tenha seus campos de entrada validados e, em caso de validação positiva - sem erros, seja inserindo no banco de dados.

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

Nela, nós descrevemos os campos que estão disponíveis no formulário **HTML**.

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** descreve o tamanho máximo que esse **input** pode receber (100 caracteres, no caso).

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 que foi inserido pelo usuário no campo `<input name="nome">`).

Vamos ver agora um exemplo mais complexo com um formulário de inserção de um funcionário com todos os campos. Para isso, crie o arquivo **forms.py** no app **core**.

Podemos descrever um form de inserção assim:

```
from django import forms

class InsereFuncionarioForm(forms.Form):
    nome = forms.CharField(
        required="True",
        max_length=200
    )
    cps = forms.CharField(
        required="True",
        max_length=14
    )
    tempo_servico = forms.IntegerField(
        required="True",
    )
    remuneracao = forms.DecimalField()
```

Mas o **models** e o **forms** são quase iguais, terei que reescrever os campos toda vez?

Claro que não!!! Por isso o Django nos presenteou com o incrível **ModelForm**.

Com o **ModelForm** nós configuramos de qual **models** o Django deve pegar os campos. A partir do atributo **fields**, nós dizemos quais campos nós queremos e através do campo **exclude**, os campos que não queremos.

Para fazer essa configuração, utilizamos os metadados da classe interna **Meta**. Metadado (no caso do **Model** e do **Form**) é tudo aquilo que não será transformado em campo, como **model**, **fields**, **ordering** etc.

Assim, nosso **ModelForms**, pode ser descrito da seguinte forma:

```
From django import forms

Class InsereFuncionarioFom(forms.ModelForm):
    Class Meta:
        Model = Funcionario
        Fields = [
            'nome', 'cpf', 'remuneracao'
        ]
        Exclude = ['tempo_servico']
```

Podemos utilizar apenas o campo `fields`, apenas o `exclude` ou dos dois juntos e mesmo ao utilizá-los, ainda podemos adicionar outros campos, **independente** dos campos do **Model**.

O resultado será um formulário com todos os campos presentes no **fields**, menos os campos do **exclude** mais os outros campos que adicionarmos.

Então, vamos ver um exemplo que utiliza todos os atributos e ainda adiciona novos campos ao formulário:

```
From django import forms

Class InsereFuncionarioForm(forms.ModelForm):
    Chefe = forms.BooleanField(
        Label = "chefe?",
        Require = True,
    )

    Biografia = forms.CharField(
        Label= : 'Biografia',
        Required: 'False',
        Widget: forms.TextArea
    )

    Class Meta:
        Model = Funcionario
        Fields = [
            'nome',
            'cpf',
            'remuneracao'
        ]
        Exclude = [
            'Tempo_servico'
        ]
```

Isso vai gerar um formulário com:

- Todos os campos contidos em **fields**;

- Serão retirados os campos contidos em `exclude`;
- O campo `forms.BooleanField`, como um `CheckBox` (`<input type="checkbox" name="chefe"...>`)
- `Biografia` como uma área de texto (`<textarea name="biografia"...></textarea>`).

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

Veja que o campo **biografia** é um tipo **CharField**, portanto deveria ser renderizado como um campo `<input type="text"...>`.

Contudo, modificamos o campo configurando o atributo **widget** com **`forms.TextArea`**.

Com isso, ele não será um simples **input**, mas será renderizado como um `<textarea></textarea>` no nosso **template**.