

# **Programação de Sistemas para Internet**

**Prof. Diego Cirilo**

**Aula 09: Forms**

# Entrada de dados no sistema

- Como enviar dados para o sistema?
- Formulários - *Forms*

# HTML Forms

- Definidos pela tag `<form></form>` ;
- Os dados são preenchidos nos elementos `<input>` ;
- Os *inputs* podem ter vários tipos ( `type` ):
  - `<input type="button">` ;
  - `<input type="checkbox">` ;
  - `<input type="color">` ;
  - `<input type="date">` ;
  - `<input type="datetime-local">` ;
  - `<input type="email">` .

# Inputs

- Cont.:
  - `<input type="file">` ;
  - `<input type="hidden">` ;
  - `<input type="image">` ;
  - `<input type="month">` ;
  - `<input type="number">` ;
  - `<input type="password">` ;
  - `<input type="radio">` .

# Inputs

- Cont.:
  - `<input type="range">` ;
  - `<input type="reset">` ;
  - `<input type="search">` ;
  - `<input type="submit">` ;
  - `<input type="tel">` ;
  - `<input type="text">` ;
  - `<input type="time">` ;
  - `<input type="url">` ;
  - `<input type="week">` .

# Labels

- Labels são as legendas ou nome do campo;
- `<label for="id do campo">Conteúdo do label</label>` ;
- Ex.:

```
<form>  
  <input type="text" id="nome" name="nome">  
  <label for="nome">Nome: </label>  
</form>
```

# Submit

- O `input` do tipo `submit` envia os dados do formulário:
- `<input type=submit value="Enviar">`

# Name

- O atributo `name` é importante pois é como o dado vai ser referenciado no servidor.

# Method/Action

- O atributo `action` indica o caminho/url para onde o form será enviado.;
- O `method` pode ser `POST` ou `GET` :
- `GET` :
  - Os dados são enviados na URL;
  - Utilizado quando a ação não modifica dados no sistema;
  - É útil pois permite salvar a URL com conteúdo;
  - Exemplos: pesquisa, ordenação, filtros, etc.
- `POST` :
  - Os dados são enviados na requisição HTTP;
  - Deve ser utilizado sempre que a ação modifique dados no sistema.



# Recebendo os dados de um form

- Quando o usuário clica no `submit` os dados do form são enviados para o servidor;
- Se o `method` for `GET` os dados vão como uma *query string* na própria URL;
- Ex. `http://localhost:8000?nome=Maria&idade=23` ;
- Se o `method` for `POST` os dados vão na requisição (`request`) HTTP;
- Quando usar `GET` ou `POST` ?

# Cross-site Request Forgery

- Falsificação de requisição entre sites;
- Aplicações maliciosas podem fazer requisições de usuários "logados";
- Pode ser prevenido com um token (ficha) de segurança;
- O Django já faz isso com a tag `{% csrf_token %}` ;
- É uma chave que o Django gera quando renderiza o form, e é enviado em um `input hidden` ;
- Quando o Django recebe a requisição, ele compara se o token é o mesmo que ele gerou.

# Recebendo os dados de um form

- Caso o `action` não seja definido, o formulário é enviado para a mesma URL onde o formulário foi carregado;
- No Django, esses dados são acessados na `view`, através dos *QueryDicts* `request.POST`, `request.GET` e `request.FILES`;
- Ex. `request.POST['nome']` ou `request.POST.get('nome')`;
- É possível utilizar os dados diretamente assim, porém o Django conta com uma ferramenta mais completa.

# Django Forms

- O Django permite a descrição de forms diretamente no Python;
- As vantagens são a possibilidade de validação, redução da quantidade de HTML, segurança, integração com os Models, etc.;
- Maior facilidade de manutenção do código também.

# Definindo um Form no Django

- Criamos um arquivo `forms.py` na pasta do `app` :
- [Tipos de Dados](#);
- Usam uma sintaxe parecida com a dos Models;
- Ex.:

```
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=100, label="Seu Nome")
    email = forms.EmailField(label="Seu Email")
    message = forms.CharField(widget=forms.Textarea, label="Mensagem")
```

# Widgets

- Os widgets são os componentes que renderizam os `inputs` do HTML;
- Os fields já tem widgets padrão, mas é possível customizar;
- [Tipos de Widgets.](#)

# Model Forms

- Normalmente os Forms trabalham com dados de algum Model;
- É possível criar um Form "automaticamente" com os dados de um Model;
- Ex.:

```
from django import forms
from .models import MeuModel

class MeuModelForm(forms.ModelForm): # É um padrão usar o Form no nome da classe
    class Meta:
        model = MeuModel # Modelo associado
        fields = ['campo1', 'campo2', 'campo3'] # Campos incluídos no formulário
```

# Usando o Django Forms nas views

- O form pode ser instanciado vazio ou com dados;
- Um form vazio é utilizado para um formulário de criação (cadastro, por exemplo);
- Um form com dados pode ser utilizado para receber/tratar/armazenar ou realizar a alteração dos dados.



# Usando o Django Forms nas views

```
from .forms import MeuModelForm

def minha_view(request):
    if request.method == 'POST':
        form = MeuForm(request.POST)
        if form.is_valid():
            form.save() # Salva os dados no banco (ModelForm)
            return render(request, 'success.html')
    else:
        form = MeuForm()

    context = {
        'form': form,
    }
    return render(request, 'form_template.html', context) # passa o form no context
```

# Editando os dados na view

- O método `save()` já chama o processo de escrita no BD;
- Podemos usar a opção `commit=False` para evitar que os dados sejam escritos no BD;
- Com esses dados em uma variável, podemos realizar algum processamento e aí sim, escrever;

```
...
temporario = form.save(commit=False) # Recupera os dados do form sem salvar
temporario.usuario = request.user
temporario.save()
...
```

# Upload de arquivos

- Usando um form do Django, passamos o `request.FILES` além do `request.POST` :
  - `form = ComArquivoForm(request.POST, request.FILES)` .

# Validação

- O método `is_valid` executa as validações do form;
- O Django possui validações padrão para cada tipo de dado;
- É possível criar suas próprias validações;
- Quando uma validação falha, uma exceção *ValidationError* é gerada.

# Criando Validações

- Podemos validar um campo definindo um método `clean_nomedocampo` , por exemplo para validar um campo CPF:

```
...
def clean_cpf(self):
    data = self.cleaned_data["cpf"]
    if (cpf não é valido): # escreva a validação de cpf
        raise ValidationError("CPF inválido!") # gera o erro

    return data # sempre retorne data
```

- É possível processar os dados, mudar maiúsculas/minúsculas, etc.

# Criando Validações

- Exemplo

```
...
def clean_estado(self):
    data = self.cleaned_data["estado"]
    estados_validos = ['RN', 'CE', 'PB']
    if (not data in estados_validos): # verifica se o valor existe na lista
        raise ValidationError("O estado informado não é permitido!") # gera o erro

    return data # sempre retorne data
```

# Mensagens de Erro

- Caso haja um *ValidationError* na validação de um Form, o Django retorna um atributo `errors` em cada field;
- Ex. `form.cpf.errors` ;
- Quando usamos o form completo no template, os erros já são carregados, desde que a view envie o form validado!

# Mensagens de Erro

- Exemplo ( `views.py` ):

```
def mensagem(request):
    if request.method == "POST":
        form = MensagemForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('mensagens')
        else:
            context["form"] = form
    else:
        context["form"] = MensagemForm()

    return render(request, "contact.html", context)
```



# Customização

- Os Forms diretamente do Django não tem estilo;
- Há algumas possibilidades para estilizar um Form no Django:
  - Acessando os fields (`form.nomedofield`) diretamente no template e criando o estilo diretamente;
  - Customizando os `widgets` diretamente em `forms.py` passando atributos como `class` e `style` (`inline`;
  - Usando bibliotecas que auxiliam nessa tarefa, como o Django Crispy Forms e o Django Widget Tweaks.

# Customização direto no template

- Exemplo de um campo *Nome* usando Bootstrap:

```
<div class="mb-3">
  <label for="id_nome" class="form-label">{{ form.nome.label }}</label>
  {{ form.nome|add_class:"form-control" }}
  {% if form.nome.errors %}
    <div class="text-danger">
      {% for error in form.nome.errors %}
        <small>{{ error }}</small>
      {% endfor %}
    </div>
  {% endif %}
</div>
```

# Customização do Widget

- Exemplo de customização de um campo no `forms.py` :

```
nome = forms.CharField(  
    label="Nome",  
    max_length=100,  
    widget=forms.TextInput(attrs={  
        'class': 'form-control', # do bootstrap, mas pode adicionar as suas  
        'placeholder': 'Enter your name'  
    })  
)
```

# Django Crispy Forms

- É um pacote que traz várias funcionalidades para estilizar os forms diretamente no Python;
- A vantagem é padronizar essa estilização, facilitar o reuso e simplificar os templates;
- Deve ser instalado com o pip `pip install django-crispy-forms` ;
- É necessário também instalar um pacote de estilos, na disciplina usaremos o Bootstrap 5: `pip install crispy-bootstrap5` ;
- Não esqueça de adicionar ao `requirements.txt` ;
- [Documentação](#).

# Django Crispy Forms

- Além de instalar os pacotes, devemos adicionar ao `INSTALLED_APPS` do `settings.py` e a configuração do template;
- Ex. ( `settings.py` ):

```
INSTALLED_APPS = (  
    ...  
    "crispy_forms",  
    "crispy_bootstrap5",  
    ...  
)  
  
CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5"  
  
CRISPY_TEMPLATE_PACK = "bootstrap5"
```

# Django Crispy Form Filter

- O jeito mais básico de usar é com o filtro de template;
- Carregue os filtros no início do template:

```
{% load crispy_forms_filters %}
```

- Use com `{{ form | crispy }}`;
- Vantagem: simplicidade;
- Desvantagem: usa o Template Pack diretamente, sem permitir mais customizações.

# Django Crispy FormHelper

- O `FormHelper` é a classe utilizada para estilizar o Form;
- Há inúmeras possibilidades;
- [Documentação](#).

# Django Crispy Forms

- No form ( forms.py ):

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.helper = FormHelper()
    self.helper.form_method = 'post'
    self.helper.layout = Layout(
        Row(
            Column('name', css_class='col-md-6'),
            Column('email', css_class='col-md-6'),
            css_class='row'
        ),
        Row(
            Column('message', css_class='col-12'),
            css_class='row'
        ),
        Submit('submit', 'Enviar', css_class='btn btn-primary')
    )
```



# Django Crispy Forms

- No template:

```
{% load crispy_forms_tags %}

<form method="post">
  {% csrf_token %}
  {% crispy form %}
</form>
```

# Dúvidas?

