Apostila 4: Formulários (Forms) e o Padrão "Post-Redirect-Get"

O "C" do CRUD (Create)

1. Introdução

O que é um Formulário Django?

Em HTML puro, você criaria um formulário assim:
<form method="POST"> <input type="text" name="titulo"> ... </form>
No back-end (na view), você teria que:

1. Pegar o dado manualmente (ex: request.POST.get('titulo')).
2. **Validar** o dado (O titulo não está vazio? Não é longo demais?).
3. Limpar o dado (proteger contra ataques XSS).
4. Salvar o dado no banco de dados.

Isso é muito trabalho e propenso a erros. O Django automatiza tudo isso.

O que é um ModelForm?
É uma classe mágica. Você diz a ela: "Construa um formulário para o meu Model de Tarefa".
O Django irá:

1. Olhar o models.py (ex: titulo = models.CharField(max_length=200)).
2. Criar um campo de formulário HTML (<input type="text" maxlength="200">).
3. Gerar automaticamente todas as regras de validação (ex: "Este campo é obrigatório", "Não pode ter mais de 200 caracteres").
4. Fornecer um método .save() que salva o objeto no banco de dados.

Nosso Objetivo: Adicionar um formulário na nossa página inicial para que o usuário possa criar uma nova Tarefa sem usar o /admin.

---

2. Módulo 1: Criando o ModelForm (15 min)

Assim como temos models.py e views.py, a convenção é criar um arquivo forms.py para nossos formulários.

Passo 1: Crie o arquivo core/forms.py
Dentro da sua pasta core, crie um novo arquivo:

```
core/
  ...
  forms.py    <-- CRIE ESTE ARQUIVO
  models.py
  views.py
  ...
```

Passo 2: Defina sua classe de Formulário
Abra core/forms.py e adicione o seguinte código:

Python

```python
# core/forms.py

from django import forms
from .models import Tarefa  # Importe o Model

# Esta classe herda de 'ModelForm'
class TarefaForm(forms.ModelForm):

    # A "mágica" acontece aqui, na classe 'Meta'
    class Meta:
        # 1. Diga ao form qual Model ele deve usar
        model = Tarefa

        # 2. Diga quais campos do Model devem virar campos no form
        # Nós só queremos que o usuário defina o 'titulo'.
        # 'concluida' (default=False) e 'criada_em' (auto_now_add=True)
        # serão preenchidos automaticamente pelo Model.
```

```
        fields = ['titulo']
```

É só isso. Você acabou de criar um formulário completo e seguro.

---

3. Módulo 2: Adicionando o Formulário ao Template (15 min)

Agora, vamos mostrar esse formulário no home.html.

Passo 1: Edite o templates/home.html
Precisamos de duas coisas:

1. A tag <form> do HTML.
2. Uma forma de "imprimir" os campos que o Django criou.

Abra templates/home.html e adicione o formulário dentro do {% block content %} (um bom lugar é logo abaixo do <h1>):

HTML

```
{% extends 'base.html' %}
...
{% block content %}
    <h1>Olá, {{ nome_usuario }}!</h1>

    <hr>
    <h3>Nova Tarefa:</h3>

    <form method="POST">
        {% csrf_token %}

        {{ form.as_p }}

        <button type="submit">Adicionar Tarefa</button>
    </form>
    <hr>
    <h2>Minha Lista de Tarefas...</h2>
    ... (o resto do seu arquivo) ...
{% endblock %}
```

Análise Rápida:

- method="POST": Diz ao navegador para **enviar** dados ao servidor (em vez de GET, que apenas **pede** dados).
- {% csrf_token %}: **ESSENCIAL**. É uma medida de segurança (Cross-Site Request Forgery). O Django *recusará* qualquer formulário POST sem ele.
- {{ form.as_p }}: Esta é a variável que virá da nossa view. .as_p diz ao Django: "Renderize os campos do formulário envolvendo-os em tags <p> (parágrafo)". (Você também pode usar .as_table ou .as_ul).

Se você rodar o servidor *agora*, a página dará um **erro**, pois a view ainda não está enviando a variável {{ form }}. Vamos corrigir isso.

---

4. Módulo 3: A View (GET vs. POST) (30 min)

Este é o passo mais importante. Nossa view home agora tem **duas responsabilidades**:

1. **Requisição GET**: Quando o usuário *visita* a página. Devemos mostrar a lista de tarefas E um formulário **vazio**.
2. **Requisição POST**: Quando o usuário *envia* o formulário. Devemos pegar os dados, validar, salvar no banco e, em seguida, mostrar a página novamente com a tarefa salva.

Passo 1: Edite core/views.py
Vamos reestruturar nossa view home:

Python

```
# core/views.py

from django.shortcuts import render, redirect  # 1. Importe o 'redirect'
from .models import Tarefa
from .forms import TarefaForm                  # 2. Importe nosso novo 'TarefaForm'

def home(request):

    # 3. Lógica de POST: Se o formulário foi enviado
    if request.method == 'POST':
        # Cria uma instância do form e preenche com os dados do POST
```

```
        form = TarefaForm(request.POST)

        # 4. O Django valida os dados (max_length, etc.)
        if form.is_valid():
            # 5. Salva o objeto no banco de dados!
            form.save()

            # 6. Redireciona de volta para a 'home'
            # Isso é o Padrão "Post-Redirect-Get" (PRG)
            return redirect('home')

        # Se o form NÃO for válido, o código continua e
        # o 'form' (com os erros) será enviado para o template

    # 7. Lógica de GET: Se o usuário apenas visitou a página
    else:
        form = TarefaForm()  # Cria um formulário vazio

    # 8. A busca de dados (fora dos 'ifs', pois é necessária sempre)
    todas_as_tarefas = Tarefa.objects.all().order_by('-criada_em') # Ordena pelas mais novas

    # 9. Atualize o contexto para incluir o formulário
    context = {
        'nome_usuario': 'Júnior',
        'tecnologias': ['Python', 'Django', 'Models', 'Forms'],
        'tarefas': todas_as_tarefas,
        'form': form,  # 10. Envie o 'form' (vazio ou com erros) para o template
    }

    return render(request, 'home.html', context)
```

Nota: Adicionei .order_by('-criada_em') para mostrar as tarefas mais novas primeiro. O - significa "ordem descendente".

O Padrão "Post-Redirect-Get" (PRG)

Por que usamos return redirect('home') após salvar?

Se apenas renderizássemos a página, o usuário estaria na mesma URL POST. Se ele apertasse F5 (atualizar), o navegador enviaria os dados do formulário *novamente*, criando uma tarefa duplicada.

Ao usar redirect('home'), nós forçamos o navegador do usuário a fazer uma nova requisição

GET para a página inicial. É uma prática de segurança fundamental.

---

5. Módulo 4: Testando o CRUD (Create) (10 min)

É hora do teste final.

1. Rode o servidor: python manage.py runserver
2. Acesse http://127.0.0.1:8000/.
3. Você deve ver sua lista de tarefas E o novo formulário "Nova Tarefa:" com um campo "Titulo".
4. **Teste 1 (Sucesso):** Digite uma nova tarefa (ex: "Entender Django Forms") e clique em "Adicionar Tarefa".
5. A página deve recarregar, e sua nova tarefa deve aparecer **no topo** da lista!
6. **Teste 2 (Validação):** Tente enviar o formulário com o campo "Titulo" *vazio*.
7. A página deve recarregar e (dependendo de como o .as_p renderiza) mostrará um erro, como "Este campo é obrigatório." (O Django impede que dados inválidos cheguem ao seu banco).
8. **Teste 3 (Admin):** Vá para http://127.0.0.1:8000/admin e veja suas tarefas. A tarefa que você criou pela página principal estará lá, salva corretamente.