Apostila 5: Update e Delete (O "U" e "D" do CRUD)

Concluindo a Funcionalidade

1. Introdução

O Desafio: Como podemos deletar ou atualizar *uma* tarefa específica?

Quando um usuário clica em "Deletar" ao lado da tarefa "Lavar a louça", como a nossa view sabe que é para deletar o objeto Tarefa com pk=1 (primary key) e não o objeto com pk=2?

A resposta é: **URLs Dinâmicas**.

Vamos criar URLs que contenham o ID (a pk) da tarefa. Exemplo:

- /tarefa/1/concluir/
- /tarefa/1/deletar/

Importante: GET vs. POST para Mudanças

Uma regra de ouro da web: **NUNCA** altere ou delete dados usando uma requisição GET.

- **Por quê?** Requisições GET podem ser acionadas por links, motores de busca (como o Google) e até mesmo por navegadores que pré-carregam páginas. Você não quer que o Google "visite" seu link tarefa/1/deletar/ e apague sua tarefa.
- **A Solução:** Qualquer ação que *mude* dados (Update, Delete) DEVE ser feita usando uma requisição POST.

Isso significa que nossos botões "Concluir" e "Deletar" não serão links (<a>), mas sim mini-formulários <form> com um botão.

Nosso Objetivo: Adicionar botões "Marcar como Concluída" e "Deletar" para cada item da nossa lista de tarefas.

---

2. Módulo 1: Adicionando Botões ao Template (20 min)

Vamos começar pelo template. Vamos adicionar os botões na nossa lista de tarefas.

Abra templates/home.html e modifique o *loop* ({% for %}) das tarefas:

HTML

```
...
  <h2>Minha Lista de Tarefas (do Banco de Dados):</h2>

  <ul style="list-style: none; padding: 0;">
    {% for tarefa in tarefas %}
      <li style="margin-bottom: 15px; border: 1px solid #ccc; padding: 10px;">

        {{ tarefa.titulo }}

        {% if tarefa.concluida %}
          <strong style="color: green;">(✓ Concluída)</strong>
        {% else %}
          <strong style="color: orange;">(☐ Pendente)</strong>
        {% endif %}

        <br>
        <small>Criada em: {{ tarefa.criada_em|date:"d/m/Y H:i" }}</small> <div style="margin-top:
10px;">

          <form action="{% url 'deletar_tarefa' tarefa.pk %}" method="POST" style="display: inline;">
            {% csrf_token %}
            <button type="submit">Deletar</button>
          </form>

          {% if not tarefa.concluida %}
            <form action="{% url 'concluir_tarefa' tarefa.pk %}" method="POST" style="display:
inline;">
              {% csrf_token %}
              <button type="submit">Marcar como Concluída</button>
```

```
                    </form>
                {% endif %}
            </div>
        </li>
    {% empty %}
        <li>Você ainda não tem tarefas cadastradas.</li>
    {% endfor %}
  </ul>
...
```

Análise:

- {% url 'deletar_tarefa' tarefa.pk %}: Esta é a tag de URL dinâmica. Ela diz ao Django: "Procure uma URL com o nome deletar_tarefa e preencha a parte dinâmica dela com o valor de tarefa.pk (a Primary Key da tarefa)".
- style="display: inline;": Apenas para os dois formulários ficarem lado a lado.
- {% if not tarefa.concluida %}: É uma boa prática. Não precisamos mostrar o botão "Concluir" se a tarefa já está concluída.
- |date:"d/m/Y H:i": É um "filtro" de template, um bônus para formatar nossa data de forma mais legível.

Se você tentar rodar o servidor agora, ele vai **quebrar**! O Django dará um erro (NoReverseMatch) porque as URLs deletar_tarefa e concluir_tarefa ainda não existem. Vamos criá-las.

---

3. Módulo 2: O Roteamento (URLs Dinâmicas) (15 min)

Precisamos dizer ao Django o que fazer quando alguém acessar /tarefa/5/deletar/.

Abra core/urls.py e adicione os novos path:

Python

```
# core/urls.py

from django.urls import path
from . import views
```

```
urlpatterns = [
    # Nossas URLs antigas
    path('', views.home, name='home'),

    # NOSSAS NOVAS URLs DINÂMICAS

    # Ex: /tarefa/5/concluir/
    # <int:pk> captura um inteiro da URL e o passa para a view como um argumento chamado 'pk'
    path('tarefa/<int:pk>/concluir/', views.concluir_tarefa, name='concluir_tarefa'),

    # Ex: /tarefa/5/deletar/
    path('tarefa/<int:pk>/deletar/', views.deletar_tarefa, name='deletar_tarefa'),
]
```

- <int:pk>: Esta é a sintaxe do Django para capturar partes de uma URL. int significa que ele espera um número inteiro, e pk é o nome da variável que será passada para a nossa view.

Agora o site não quebra mais ao carregar, mas os botões ainda não funcionam, pois as views concluir_tarefa e deletar_tarefa não existem.

---

4. Módulo 3: A Lógica (Views de Update e Delete) (25 min)

Finalmente, vamos escrever a lógica em core/views.py.

Passo 1: Importações
Precisamos de mais uma ferramenta do Django: get_object_or_404.
Este atalho genial tenta buscar um objeto. Se não o encontrar (ex: ID não existe), ele automaticamente retorna uma página de "Erro 404 - Não Encontrado", em vez de quebrar seu site.
No topo do core/views.py, mude a linha do shortcuts:

Python

```
# core/views.py
...
# Adicione 'get_object_or_404'
```

```
from django.shortcuts import render, redirect, get_object_or_404
...
```

Passo 2: Adicione as novas views
Adicione estas duas novas funções ao final do seu core/views.py:

Python

```
# core/views.py
... (depois da sua view 'home') ...

def concluir_tarefa(request, pk):
    # 1. Busca a tarefa pela 'pk' (ID) vinda da URL.
    # Se não achar, retorna um erro 404.
    tarefa = get_object_or_404(Tarefa, pk=pk)

    # 2. Segurança: Apenas execute se o método for POST
    if request.method == 'POST':
        # 3. A Lógica de "Update"
        tarefa.concluida = True
        tarefa.save() # Não se esqueça de salvar!

        # 4. Redireciona de volta para a 'home' (Padrão PRG)
        return redirect('home')

def deletar_tarefa(request, pk):
    # 1. Busca a tarefa
    tarefa = get_object_or_404(Tarefa, pk=pk)

    # 2. Segurança: Apenas execute se o método for POST
    if request.method == 'POST':
        # 3. A Lógica de "Delete"
        tarefa.delete()

        # 4. Redireciona de volta para a 'home'
        return redirect('home')
```

---

5. Módulo 4: Teste Final (O CRUD Completo) (10 min)

Rode seu servidor (python manage.py runserver) e vamos testar o ciclo completo.

1. Acesse http://127.0.0.1:8000/.
2. **Create:** Use o formulário para criar uma nova tarefa, ex: "Testar botões". A tarefa deve aparecer na lista como "Pendente".
3. **Read:** Você está lendo (vendo) a lista de tarefas.
4. **Update:** Encontre a tarefa "Testar botões" e clique no botão "Marcar como Concluída".
   - A página deve recarregar.
   - A tarefa agora deve estar marcada como "(✓ Concluída)".
   - O botão "Marcar como Concluída" deve ter desaparecido para *aquela* tarefa.
5. **Delete:** Encontre qualquer tarefa e clique em "Deletar".
   - A página deve recarregar.
   - A tarefa deve ter desaparecido permanentemente da lista.

---

6. Revisão da Aula

Parabéns! Você construiu um aplicativo CRUD completo em Django!

O que aprendemos nesta apostila:

1. **CRUD:** Concluímos as operações de Update e Delete.
2. **URLs Dinâmicas:** Aprendemos a capturar partes da URL (como <int:pk>) no urls.py para agir sobre objetos específicos.
3. **Segurança (POST):** Entendemos por que *qualquer* alteração de dados (Update/Delete) DEVE ser feita via POST, e como fazer isso com um mini-formulário.
4. **get_object_or_404**: Um atalho essencial para buscar um objeto ou retornar um Erro 404 de forma limpa.
5. **ORM (Update/Delete):** Aprendemos a usar objeto.save() para atualizar um objeto e objeto.delete() para removê-lo.
6. **Tag {% url %} com Parâmetros:** Como gerar URLs dinâmicas no template ({% url 'nome_da_url' objeto.pk %}).
7. **Lógica no Template:** Usamos {% if %} para exibir ou ocultar elementos (como o botão "Concluir") de forma condicional.