Apostila 7: Personalizando o Django Admin

Transformando o Admin em um Painel de Controle

1. Introdução

O Django Admin foi projetado para ser "hackeável" e personalizável. O comportamento padrão (apenas listar o __str__ do objeto) é só o ponto de partida.

Para personalizar o Admin, paramos de usar a forma "simples" de registro...

Python

```
# A forma antiga:
admin.site.register(Tarefa)
```

...e passamos a usar a forma "avançada", que envolve criar uma classe que herda de admin.ModelAdmin. Esta classe funciona como um painel de configuração para como o Model de Tarefa deve se comportar dentro do Admin.

Nosso Objetivo: Adicionar colunas, filtros, uma barra de busca e organizar melhor o formulário de edição das nossas tarefas.

---

2. Módulo 1: O ModelAdmin (A Base da Personalização)

O primeiro passo é criar uma classe de configuração para o nosso Model Tarefa.

Passo 1: Edite core/admin.py
Abra o arquivo core/admin.py e vamos modificá-lo para usar uma classe ModelAdmin.

Python

```
# core/admin.py

from django.contrib import admin
from .models import Tarefa

# 1. Crie uma classe que herda de admin.ModelAdmin
# A convenção é usar o nome do Model + "Admin"
class TarefaAdmin(admin.ModelAdmin):
    # Por enquanto, ela está vazia. Vamos configurar aqui dentro.
    pass

# 2. Modifique o registro
# Em vez de registrar 'Tarefa' sozinho,
# registre 'Tarefa' COM a sua classe de configuração 'TarefaAdmin'
admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Rode o servidor (python manage.py runserver), acesse o Admin (/admin) e entre na área "Tarefas".
Visualmente, **nada mudou**. Mas agora temos o "andaime" pronto para adicionar nossas personalizações.

---

3. Módulo 2: list_display (Adicionando Colunas)

Por padrão, o Admin só mostra uma coluna: o resultado do método __str__ (no nosso caso, o titulo). Vamos mudar isso.

O atributo list_display nos permite definir quais campos do nosso Model devem aparecer como colunas.

Passo 1: Edite core/admin.py
Adicione o list_display à sua classe TarefaAdmin.

Python

```
# core/admin.py

class TarefaAdmin(admin.ModelAdmin):
    # 'list_display' é uma tupla com os nomes dos campos
    # que queremos exibir como colunas na lista
    list_display = ('titulo', 'user', 'concluida', 'criada_em')

admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Recarregue a página de listagem de Tarefas no Admin.
Mágica! Agora, em vez de uma simples lista de títulos, você tem uma tabela organizada com colunas clicáveis para "TÍTULO", "USER", "CONCLUÍDA" e "CRIADA EM". O Django até formata o booleano concluida com ícones de "sim" (✓) ou "não" (✗).

---

4. Módulo 3: list_filter (Adicionando Filtros)

Nossa lista está bonita, mas e se tivermos 5.000 tarefas? Precisamos de filtros.

O atributo list_filter adiciona uma barra lateral de filtros rápidos.

Passo 1: Edite core/admin.py
Adicione list_filter à classe TarefaAdmin.

Python

```
# core/admin.py

class TarefaAdmin(admin.ModelAdmin):
    list_display = ('titulo', 'user', 'concluida', 'criada_em')

    # Adiciona uma barra lateral de filtros
    # O Django entende o tipo de campo e cria o filtro certo:
```

```
    # - 'concluida': Filtro de "Sim/Não"
    # - 'user': Filtro de lista (baseado em um ForeignKey)
    # - 'criada_em': Filtro de data (Hoje, Últimos 7 dias, Este mês, etc.)
    list_filter = ('concluida', 'user', 'criada_em')

admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Recarregue o Admin. Agora você tem uma barra lateral "Filtrar por". Tente clicar em "Concluída: Sim" ou filtrar por um usuário específico. O painel ficou instantaneamente mais útil.

---

5. Módulo 4: search_fields (Adicionando Busca)

Filtros são ótimos, mas às vezes você só quer encontrar *aquela* tarefa específica.

O atributo search_fields adiciona uma barra de busca no topo da lista.

Passo 1: Edite core/admin.py
Adicione search_fields à classe TarefaAdmin.

Python

```
# core/admin.py

class TarefaAdmin(admin.ModelAdmin):
    list_display = ('titulo', 'user', 'concluida', 'criada_em')
    list_filter = ('concluida', 'user', 'criada_em')

    # Adiciona uma barra de busca
    # 'titulo': Vai buscar no campo 'titulo'
    # 'user__username': Este é avançado! Usamos o lookup '__'
    # para dizer ao Django: "Busque também no campo 'username'
    # do Model 'User' que está relacionado".
    search_fields = ('titulo', 'user__username')

admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Recarregue o Admin. Agora você tem uma barra de busca. Tente buscar pelo título de uma tarefa ou pelo nome de usuário de quem a criou.

---

6. Módulo 5: Personalizando o Formulário (Edição e Criação)

A página de *lista* está ótima. Mas e a página de *edição* (quando você clica em uma tarefa)?

Ela está meio bagunçada. E, pior, ela permite que o admin *edite* o campo criada_em, o que não faz sentido (esse campo é automático).

Vamos organizar isso usando readonly_fields e fieldsets.

Passo 1: Edite core/admin.py
Adicione readonly_fields e fieldsets à classe TarefaAdmin.

Python

```
# core/admin.py

class TarefaAdmin(admin.ModelAdmin):
    list_display = ('titulo', 'user', 'concluida', 'criada_em')
    list_filter = ('concluida', 'user', 'criada_em')
    search_fields = ('titulo', 'user__username')

    # ----------------------------------------------------
    # NOVAS CONFIGURAÇÕES (para o formulário de edição)
    # ----------------------------------------------------

    # 'fieldsets' permite agrupar os campos em seções
    # A estrutura é: ( ('Nome da Seção', {'fields': ('campo1', 'campo2')}) , ... )
    fieldsets = (
        ('Informações Principais', {
            'fields': ('user', 'titulo')
        }),
        ('Status da Tarefa', {
            'fields': ('concluida', 'criada_em')
        }),
    )
```

```
    # 'readonly_fields' define campos que podem ser vistos, mas não editados
    # Perfeito para campos automáticos como 'criada_em'
    readonly_fields = ('criada_em',)

admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Volte para a lista de tarefas no Admin e clique em qualquer tarefa para editá-la.
Você verá duas mudanças drásticas:

1. O formulário agora está limpo e organizado nas seções "Informações Principais" e "Status da Tarefa".
2. O campo "Criada em" é exibido, mas está "acinzentado" (grayed out) e não pode ser alterado.

---

7. Módulo 6 (Bônus): Adicionando Colunas "Customizadas"

E se quisermos exibir uma coluna que *não* é um campo direto do Model? Por exemplo, o **email** do usuário. O list_display pode chamar funções!

Passo 1: Edite core/admin.py
Vamos definir um método dentro da nossa classe TarefaAdmin e adicioná-lo ao list_display.

Python

```
# core/admin.py

class TarefaAdmin(admin.ModelAdmin):
    # 1. Adicione 'get_user_email' ao list_display
    list_display = ('titulo', 'user', 'get_user_email', 'concluida', 'criada_em')
    list_filter = ('concluida', 'user', 'criada_em')
    search_fields = ('titulo', 'user__username')
    fieldsets = (
        ('Informações Principais', {
            'fields': ('user', 'titulo')
        }),
        ('Status da Tarefa', {
            'fields': ('concluida', 'criada_em')
```

```
        }),
    )
    readonly_fields = ('criada_em',)

    # 2. Defina o método
    # O 'obj' é a instância da Tarefa que está sendo exibida na linha
    @admin.display(description='Email do Usuário') # Define o título da coluna
    def get_user_email(self, obj):
        return obj.user.email

admin.site.register(Tarefa, TarefaAdmin)
```

Passo 2: Teste!
Recarregue a página de lista de tarefas. Agora você tem uma nova coluna, "EMAIL DO USUÁRIO", que busca o email do modelo User relacionado!

---

8. Revisão da Aula

O que fizemos:

1. **admin.ModelAdmin**: Aprendemos a forma correta de registrar um Model para personalização.
2. **list_display**: Controlamos as colunas da nossa tabela de dados.
3. **list_filter**: Adicionamos filtros poderosos com uma única linha de código.
4. **search_fields**: Adicionamos uma barra de busca (inclusive em models relacionados com __).
5. **fieldsets**: Organizamos nosso formulário de edição em seções lógicas.
6. **readonly_fields**: Protegemos campos automáticos contra edição.
7. **Métodos Customizados**: Criamos uma coluna nova (get_user_email) que busca dados de um Model relacionado.

Nosso Admin agora é um painel de gerenciamento de dados completo, robusto e muito mais profissional.