Programação de Sistemas para Internet

Prof. Diego Cirilo

Aula 18: Customizando o Django Admin

Django Admin

- Interface administrativa gerada automaticamente;
- Permite CRUD completo dos models;
- Útil para gerenciamento interno;
- Altamente customizável através do `ModelAdmin`.

Registro Básico

```
from django.contrib import admin
from .models import Produto

# Registro simples
admin.site.register(Produto)
```

- Usa configurações padrão;
- Exibe apenas o `__str__` do objeto na listagem.

ModelAdmin

- Classe que define como o model aparece no Admin;
- Permite customizar listagem, formulários, filtros, etc.

```
from django.contrib import admin
from .models import Produto

class ProdutoAdmin(admin.ModelAdmin):
    pass  # configurações padrão

admin.site.register(Produto, ProdutoAdmin)
```

Decorator @admin.register

- Forma alternativa e mais limpa de registrar:

```
from django.contrib import admin
from .models import Produto

@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'estoque']
```

- Equivalente ao `admin.site.register(Produto, ProdutoAdmin)`

list_display

- Define as colunas exibidas na listagem;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'estoque', 'ativo']
```

- Aceita campos do model e métodos;
- Torna a listagem mais informativa.

list_display com Métodos

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'estoque_formatado']

    def estoque_formatado(self, obj):
        if obj.estoque > 10:
            return f"{obj.estoque} unidades"
        return f"{obj.estoque} unidades"

    estoque_formatado.short_description = "Estoque"
```

list_filter

- Adiciona filtros na barra lateral;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'categoria', 'preco', 'ativo']
    list_filter = ['categoria', 'ativo', 'data_criacao']
```

- Útil para filtrar por categoria, status, data, etc.

search_fields

- Adiciona caixa de busca;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'categoria', 'preco']
    search_fields = ['nome', 'descricao', 'categoria__nome']
```

- Busca em múltiplos campos;
- Use `__` para campos de relacionamentos.

ordering

- Define ordenação padrão da listagem;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'data_criacao']
    ordering = ['-data_criacao']  # mais recentes primeiro
```

- Use `-` para ordem decrescente.

list_editable

- Permite editar campos direto na listagem;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'estoque', 'ativo']
    list_editable = ['preco', 'estoque', 'ativo']
```

- O campo deve estar em `list_display`;
- O primeiro campo de `list_display` não pode ser editável.

date_hierarchy

- Adiciona navegação por data no topo;

```
@admin.register(Pedido)
class PedidoAdmin(admin.ModelAdmin):
    list_display = ['id', 'cliente', 'total', 'data']
    date_hierarchy = 'data'
```

- Permite navegar por ano/mês/dia.

list_per_page

- Define quantos itens por página;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco']
    list_per_page = 25  # padrão é 100
```

Exemplo Completo - Listagem

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'categoria', 'preco', 'estoque', 'ativo']
    list_filter = ['categoria', 'ativo']
    search_fields = ['nome', 'descricao']
    ordering = ['nome']
    list_editable = ['preco', 'ativo']
    list_per_page = 20
    date_hierarchy = 'data_criacao'
```

Customizando o Formulário

- Por padrão, o Admin exibe todos os campos editáveis;
- Podemos customizar quais campos aparecem e como.

fields

- Define quais campos aparecem e em qual ordem;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    fields = ['nome', 'categoria', 'preco', 'descricao']
```

- Campos não listados não aparecem no formulário.

exclude

- Exclui campos específicos;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    exclude = ['slug', 'data_atualizacao']
```

- Todos os outros campos aparecem.

readonly_fields

- Campos exibidos mas não editáveis;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    readonly_fields = ['data_criacao', 'data_atualizacao', 'slug']
```

- Útil para campos automáticos.

fieldsets

- Agrupa campos em seções;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    fieldsets = [
        ('Informações Básicas', {
            'fields': ['nome', 'categoria', 'descricao']
        }),
        ('Preço e Estoque', {
            'fields': ['preco', 'estoque']
        }),
        ('Configurações', {
            'fields': ['ativo', 'destaque'],
            'classes': ['collapse']  # seção recolhível
        }),
    ]
```

fieldsets - Opções

```
fieldsets = [
    ('Seção', {
        'fields': ['campo1', 'campo2'],
        'classes': ['collapse'],  # recolhível
        'description': 'Texto explicativo'
    }),
    ('Campos na mesma linha', {
        'fields': [('campo1', 'campo2')]  # tuple = mesma linha
    }),
]
```

Inlines

- Edita objetos relacionados na mesma página;
- Útil para relações ForeignKey;
- Ex: editar itens do pedido na página do pedido.

Tipos de Inline

```
from django.contrib import admin
from .models import Pedido, ItemPedido

# Exibe como tabela
class ItemInline(admin.TabularInline):
    model = ItemPedido
    extra = 1  # linhas vazias extras

# Exibe como formulários empilhados
class ItemStackedInline(admin.StackedInline):
    model = ItemPedido
    extra = 1
```

Usando Inlines

```
class ItemInline(admin.TabularInline):
    model = ItemPedido
    extra = 1
    fields = ['produto', 'quantidade', 'preco']

@admin.register(Pedido)
class PedidoAdmin(admin.ModelAdmin):
    list_display = ['id', 'cliente', 'data', 'total']
    inlines = [ItemInline]
```

Inline - Opções

```
class ItemInline(admin.TabularInline):
    model = ItemPedido
    extra = 1                # linhas extras vazias
    min_num = 1              # mínimo de itens
    max_num = 10             # máximo de itens
    can_delete = True        # permitir deletar
    show_change_link = True  # link para editar item
    readonly_fields = ['subtotal']
```

Actions (Ações em Lote)

- Ações aplicadas a múltiplos objetos selecionados;
- Por padrão: apenas "Deletar selecionados".

Criando Actions

```python
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ['nome', 'preco', 'ativo']
    actions = ['ativar_produtos', 'desativar_produtos']

    @admin.action(description="Ativar produtos selecionados")
    def ativar_produtos(self, request, queryset):
        queryset.update(ativo=True)

    @admin.action(description="Desativar produtos selecionados")
    def desativar_produtos(self, request, queryset):
        queryset.update(ativo=False)
```

Actions com Feedback

```
@admin.action(description="Ativar produtos selecionados")
def ativar_produtos(self, request, queryset):
    count = queryset.update(ativo=True)
    self.message_user(
        request,
        f"{count} produto(s) ativado(s) com sucesso."
    )
```

Customizando o Cabeçalho

```
# admin.py ou urls.py
admin.site.site_header = "Minha Loja - Administração"
admin.site.site_title = "Admin Minha Loja"
admin.site.index_title = "Painel de Controle"
```

- `site_header`: texto no topo da página
- `site_title`: título da aba do navegador
- `index_title`: título da página inicial

prepopulated_fields

- Preenche campos automaticamente baseado em outros;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    prepopulated_fields = {'slug': ('nome',)}
```

- Muito usado para gerar slugs automaticamente.

autocomplete_fields

- Busca com autocomplete para ForeignKey;

```
@admin.register(Categoria)
class CategoriaAdmin(admin.ModelAdmin):
    search_fields = ['nome']  # necessário!

@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    autocomplete_fields = ['categoria']
```

- O model relacionado precisa ter `search_fields`.

raw_id_fields

- Alternativa ao select para ForeignKey com muitos itens;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    raw_id_fields = ['categoria']
```

- Exibe campo com ID e botão de busca.

filter_horizontal / filter_vertical

- Widget melhorado para ManyToMany;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    filter_horizontal = ['tags']
    # ou
    filter_vertical = ['tags']
```

save_model

- Customiza o que acontece ao salvar;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        if not change:  # novo objeto
            obj.criado_por = request.user
        super().save_model(request, obj, form, change)
```

get_queryset

- Customiza a query da listagem;

```
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    def get_queryset(self, request):
        qs = super().get_queryset(request)
        # Apenas produtos da loja do usuário
        if not request.user.is_superuser:
            qs = qs.filter(loja=request.user.loja)
        return qs
```

Exemplo Completo

```python
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    # Listagem
    list_display = ['nome', 'categoria', 'preco', 'ativo']
    list_filter = ['categoria', 'ativo']
    search_fields = ['nome', 'descricao']
    ordering = ['nome']

    # Formulário
    fieldsets = [
        (None, {'fields': ['nome', 'slug', 'categoria']}),
        ('Detalhes', {'fields': ['descricao', 'preco', 'estoque']}),
        ('Status', {'fields': ['ativo'], 'classes': ['collapse']}),
    ]
    prepopulated_fields = {'slug': ('nome',)}
    readonly_fields = ['data_criacao']
```

Referências

- https://docs.djangoproject.com/en/5.1/ref/contrib/admin/
- https://docs.djangoproject.com/en/5.1/ref/contrib/admin/#modeladmin-options
- https://docs.djangoproject.com/en/5.1/ref/contrib/admin/#inlinemodeladmin-objects

Dúvidas?