**Manual de Desenvolvimento: Sistema de Advocacia em Django**

Este documento serve como um guia e "receita de bolo" para adicionar novas funcionalidades ao nosso sistema de gestão. O objetivo é manter um padrão de desenvolvimento consistente, facilitando a manutenção e a colaboração entre os desenvolvedores.

O nosso projeto segue o padrão **MVT (Model-View-Template)** do Django. Qualquer nova funcionalidade geralmente envolverá a modificação ou criação de arquivos nestas três camadas, além dos arquivos de rotas (URLs) e formulários (Forms).

**Estrutura dos Passos**

O desenvolvimento de uma nova funcionalidade quase sempre seguirá esta ordem:

1. **Model (Onde os dados vivem):** Definir a estrutura dos dados em gestao/models.py.
2. **Admin (Acesso rápido):** Tornar o novo modelo gerenciável em gestao/admin.py.
3. **Form (Como os dados são inseridos):** Criar formulários em gestao/forms.py.
4. **URL (O endereço da página):** Definir a rota em gestao/urls.py.
5. **View (A lógica da página):** Escrever a função em gestao/views.py.
6. **Template (O que o usuário vê):** Criar o arquivo HTML em gestao/templates/gestao/.

A seguir, detalhamos os cenários mais comuns.

**Cenário 1: Criar um Novo Cadastro (Ex: um modelo de "Documento")**

Este é o fluxo para adicionar uma nova entidade ao sistema.

**Passo 1: Definir o Modelo (models.py)**

1. Abra gestao/models.py.
2. Crie uma nova classe que herda de models.Model.
3. Defina os campos (colunas do banco de dados).
4. Adicione um \_\_str\_\_ para uma representação legível e uma class Meta para organização.

**Exemplo:**

# Em gestao/models.py

class Documento(models.Model):

processo = models.ForeignKey(Processo, on\_delete=models.CASCADE, related\_name="documentos")

nome = models.CharField(max\_length=200, verbose\_name="Nome do Documento")

arquivo = models.FileField(upload\_to='documentos\_processos/')

data\_upload = models.DateTimeField(auto\_now\_add=True)

history = HistoricalRecords()

def \_\_str\_\_(self):

return self.nome

class Meta:

verbose\_name = "Documento"

verbose\_name\_plural = "Documentos"

ordering = ['-data\_upload']

**Passo 2: Criar e Aplicar as Migrações**

Sempre que você altera o models.py, precisa sincronizar com o banco de dados.

1. No terminal, com o ambiente virtual ativo, rode:
2. python manage.py makemigrations gestao
3. Em seguida, aplique a migração:
4. python manage.py migrate

**Passo 3: Registrar no Admin (admin.py)**

Para poder gerenciar o novo modelo facilmente pelo painel /admin/.

1. Abra gestao/admin.py.
2. Importe o novo modelo no topo do arquivo.
3. Crie uma nova classe Admin e registre-a.

**Exemplo:**

# Em gestao/admin.py

# 1. Importe o novo modelo

from .models import Documento # Adicione junto aos outros

# 2. Registre-o

@admin.register(Documento)

class DocumentoAdmin(admin.ModelAdmin):

list\_display = ('nome', 'processo', 'data\_upload')

search\_fields = ('nome', 'processo\_\_numero\_processo')

autocomplete\_fields = ['processo']

**Cenário 2: Criar uma Nova Tela (Ex: um painel financeiro geral)**

Este é o fluxo para criar uma nova página que exibe informações para o usuário.

**Passo 1: Criar a Rota (urls.py)**

1. Abra gestao/urls.py.
2. Adicione um novo path à lista urlpatterns.

**Exemplo:**

# Em gestao/urls.py

urlpatterns = [

# ... outras urls

path('financeiro/', views.painel\_financeiro, name='painel\_financeiro'),

]

**Passo 2: Criar a Lógica (views.py)**

1. Abra gestao/views.py.
2. Crie a função que a rota aponta. Esta função deve buscar os dados no banco e passá-los para um template.

**Exemplo:**

# Em gestao/views.py

@login\_required

def painel\_financeiro(request):

receitas\_pendentes = LancamentoFinanceiro.objects.filter(tipo='RECEITA', status\_\_in=['A\_PAGAR', 'ATRASADO'])

despesas\_pendentes = LancamentoFinanceiro.objects.filter(tipo='DESPESA', status='A\_PAGAR')

context = {

'receitas': receitas\_pendentes,

'despesas': despesas\_pendentes,

}

return render(request, 'gestao/painel\_financeiro.html', context)

**Passo 3: Criar o Template (HTML)**

1. Na pasta gestao/templates/gestao/, crie um novo arquivo HTML com o nome que você usou na view.
2. Estruture o HTML para exibir os dados recebidos no context. Sempre use {% extends 'gestao/base.html' %} para manter a consistência visual.

**Exemplo (painel\_financeiro.html):**

{% extends 'gestao/base.html' %}

{% block title %}Painel Financeiro{% endblock %}

{% block content %}

<h1>Painel Financeiro</h1>

<div class="card">

<div class="card-header"><h2>Contas a Receber</h2></div>

<div class="card-body">

<ul>

{% for receita in receitas %}

<li>{{ receita.descricao }} - R$ {{ receita.valor }}</li>

{% empty %}

<li>Nenhuma conta a receber pendente.</li>

{% endfor %}

</ul>

</div>

</div>

{% endblock %}

**Cenário 3: Adicionar um Modal de Cadastro Rápido**

Este é um fluxo mais avançado, que usamos para "Cliente" e "Tipo de Serviço". Ele envolve comunicação via JavaScript (AJAX/Fetch).

1. **Criar o Formulário (forms.py):** Crie um ModelForm simples para o modelo que será cadastrado (Ex: TipoAcaoForm).
2. **Criar a Rota (urls.py):** Crie uma rota específica para o modal, que será chamada pelo JavaScript. *Ex: path('tipoacao/adicionar/modal/', views.adicionar\_tipoacao\_modal, name='adicionar\_tipoacao\_modal')*
3. **Criar a View (views.py):** Esta view é especial. Ela deve:
   * Receber uma requisição POST.
   * Validar o formulário.
   * Se for válido, salvar o objeto.
   * Retornar uma **JsonResponse** com o status de sucesso e os dados do novo objeto (pk e nome).
   * Se for inválido, retornar uma JsonResponse com o status de erro e a lista de erros.
4. **Adicionar o HTML do Modal:** No template onde o cadastro rápido é necessário (ex: dashboard.html), adicione o HTML do novo modal.
5. **Adicionar o JavaScript (base.html ou no template específico):**
   * Capture o evento de submit do formulário do modal.
   * Use e.preventDefault() para impedir o envio normal.
   * Use a API fetch() para enviar os dados do formulário para a URL que você criou.
   * No .then() da resposta, verifique o status.
   * Se for success, adicione a nova Option ao select desejado, limpe o formulário e feche o modal.
   * Se for error, exiba as mensagens de erro para o usuário.
   * Lembre-se de lidar com o problema do "backdrop" ao fechar um modal que foi aberto por outro.

Seguindo estes fluxos, qualquer desenvolvedor conseguirá adicionar novas funcionalidades ao sistema de forma organizada e mantendo o padrão que estabelecemos juntos.