**Resumo de Transição do Projeto Albino Marks**

**07/07/2024**



**Descrição Resumida do Projeto**

**Projeto:** Aplicativo Django para gerenciamento de artigos, autores, áreas e mídias

**Funcionalidades Implementadas:**

1. **Modelos:**
   * Autor
   * Area
   * Midia
   * Artigo
2. **Formulários:**
   * ArtigoForm
   * AutorForm
   * AreaForm
   * MidiaForm
3. **Views:**
   * Listagem, criação, atualização e exclusão para Artigo
   * Listagem, criação, atualização e exclusão para Autor
   * Listagem, criação, atualização e exclusão para Area
   * Listagem, criação, atualização e exclusão para Midia
4. **URLs:**
   * Configuração de URLs para todas as views acima.
   * Inclusão do aplicativo no projeto principal via include('A\_Lei\_no\_NT.urls')
5. **Admin:**
   * Registro dos modelos no admin do Django.
   * Acesso e manipulação dos dados via interface de administração do Django.
6. **Templates:**
   * Templates básicos para listagem, formulários e confirmações de exclusão para todos os modelos mencionados.

**Código para Compartilhar no Novo Chat:**

1. **Models (models.py):** Todos os modelos para Autor, Area, Midia e Artigo.
2. **Forms (forms.py):** Todos os formulários para os modelos acima.
3. **Views (views.py):** Todas as views para listagem, criação, atualização e exclusão de Artigo, Autor, Area e Midia.
4. **URLs (urls.py):** Configuração de URLs tanto do projeto principal quanto do aplicativo.
5. **Templates:** Pelo menos um exemplo de cada tipo de template usado (formulário, listagem, confirmação de exclusão).

**Resumo dos Arquivos Importantes**

1. **models.py:**

python

Copiar código

from django.db import models

class Autor(models.Model):

nome\_autor = models.CharField(max\_length=200)

biografia = models.TextField()

midia = models.FileField(upload\_to='A\_Lei\_no\_NT/Imagens/Autores/', null=True, blank=True)

foto = models.ImageField(upload\_to='A\_Lei\_no\_NT/Imagens/Autores/', null=True, blank=True)

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

return self.nome\_autor

class Area(models.Model):

nome\_area = models.CharField(max\_length=200)

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

return self.nome\_area

class Midia(models.Model):

tipo = models.CharField(max\_length=50)

caminho = models.CharField(max\_length=200)

descricao = models.CharField(max\_length=200, null=True, blank=True)

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

return f"{self.tipo}: {self.descricao}"

class Artigo(models.Model):

titulo = models.CharField(max\_length=200)

texto = models.TextField()

area = models.ForeignKey(Area, on\_delete=models.CASCADE)

autor = models.ForeignKey(Autor, on\_delete=models.CASCADE)

midia = models.ForeignKey(Midia, on\_delete=models.SET\_NULL, null=True, blank=True)

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

return self.titulo

1. **forms.py:**

python

Copiar código

from django import forms

from .models import Artigo, Autor, Area, Midia

class ArtigoForm(forms.ModelForm):

class Meta:

model = Artigo

fields = ['titulo', 'texto', 'area', 'autor', 'midia']

class AutorForm(forms.ModelForm):

class Meta:

model = Autor

fields = ['nome\_autor', 'biografia', 'midia', 'foto']

class AreaForm(forms.ModelForm):

class Meta:

model = Area

fields = ['nome\_area']

class MidiaForm(forms.ModelForm):

class Meta:

model = Midia

fields = ['tipo', 'caminho', 'descricao']

1. **views.py:**

python

Copiar código

from django.shortcuts import render, get\_object\_or\_404, redirect

from .models import Artigo, Autor, Area, Midia

from .forms import ArtigoForm, AutorForm, AreaForm, MidiaForm

import logging

logger = logging.getLogger(\_\_name\_\_)

def index(request):

artigos = Artigo.objects.all()

return render(request, 'A\_Lei\_no\_NT/index.html', {'artigos': artigos})

def artigo\_list(request):

artigos = Artigo.objects.all()

return render(request, 'A\_Lei\_no\_NT/artigo\_list.html', {'artigos': artigos})

def artigo\_detalhe(request, id):

artigo = get\_object\_or\_404(Artigo, id=id)

return render(request, 'A\_Lei\_no\_NT/artigo\_detalhe.html', {'artigo': artigo})

def artigo\_create(request):

if request.method == 'POST':

form = ArtigoForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('artigo\_list')

else:

form = ArtigoForm()

return render(request, 'A\_Lei\_no\_NT/artigo\_form.html', {'form': form})

def artigo\_update(request, id):

artigo = get\_object\_or\_404(Artigo, id=id)

if request.method == 'POST':

form = ArtigoForm(request.POST, request.FILES, instance=artigo)

if form.is\_valid():

form.save()

return redirect('artigo\_list')

else:

form = ArtigoForm(instance=artigo)

return render(request, 'A\_Lei\_no\_NT/artigo\_form.html', {'form': form})

def artigo\_delete(request, id):

artigo = get\_object\_or\_404(Artigo, id=id)

if request.method == 'POST':

artigo.delete()

return redirect('artigo\_list')

return render(request, 'A\_Lei\_no\_NT/artigo\_confirm\_delete.html', {'artigo': artigo})

def autor\_detalhe(request, id):

autor = get\_object\_or\_404(Autor, id=id)

return render(request, 'A\_Lei\_no\_NT/autor\_detalhe.html', {'autor': autor})

def autor\_list(request):

autores = Autor.objects.all()

return render(request, 'A\_Lei\_no\_NT/autor\_list.html', {'autores': autores})

def autor\_create(request):

if request.method == 'POST':

form = AutorForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('autor\_list')

else:

form = AutorForm()

return render(request, 'A\_Lei\_no\_NT/autor\_form.html', {'form': form})

def autor\_update(request, id):

autor = get\_object\_or\_404(Autor, id=id)

if request.method == 'POST':

form = AutorForm(request.POST, request.FILES, instance=autor)

if form.is\_valid():

form.save()

return redirect('autor\_list')

else:

form = AutorForm(instance=autor)

return render(request, 'A\_Lei\_no\_NT/autor\_form.html', {'form': form})

def autor\_delete(request, id):

autor = get\_object\_or\_404(Autor, id=id)

if request.method == 'POST':

autor.delete()

return redirect('autor\_list')

return render(request, 'A\_Lei\_no\_NT/autor\_confirm\_delete.html', {'autor': autor})

def area\_detalhe(request, id):

area = get\_object\_or\_404(Area, id=id)

return render(request, 'A\_Lei\_no\_NT/area\_detalhe.html', {'area': area})

def area\_list(request):

areas = Area.objects.all()

return render(request, 'A\_Lei\_no\_NT/area\_list.html', {'areas': areas})

def area\_create(request):

if request.method == 'POST':

form = AreaForm(request.POST)

if form.is\_valid():

form.save()

return redirect('area\_list')

else:

form = AreaForm()

return render(request, 'A\_Lei\_no\_NT/area\_form.html', {'form': form})

def area\_update(request, id):

area = get\_object\_or\_404(Area, id=id)

if request.method == 'POST':

form = AreaForm(request.POST, instance=area)

if form.is\_valid():

form.save()

return redirect('area\_list')

else:

form = AreaForm(instance=area)

return render(request, 'A\_Lei\_no\_NT/area\_form.html', {'form': form})

def area\_delete(request, id):

area = get\_object\_or\_404(Area, id=id)

if request.method == 'POST':

area.delete()

return redirect('area\_list')

return render(request, 'A\_Lei\_no\_NT/area\_confirm\_delete.html', {'area': area})

def midia\_list(request):

midias = Midia.objects.all()

return render(request, 'A\_Lei\_no\_NT/midia\_list.html', {'midias': midias})

def midia\_create(request):

if request.method == 'POST':

form = MidiaForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('midia\_list')

else:

form = MidiaForm()

return render(request, 'A\_Lei\_no\_NT/midia\_form.html', {'form': form})

def midia\_update(request, id):

midia = get\_object\_or\_404(Midia, id=id)

if request.method == 'POST':

form = MidiaForm(request.POST, request.FILES, instance=midia)

if form.is\_valid():

form.save()

return redirect('midia\_list')

else:

form = MidiaForm(instance=midia)

return render(request, 'A\_Lei\_no\_NT/midia\_form.html', {'form': form})

def midia\_delete(request, id):

midia = get\_object\_or\_404(Midia, id=id)

if request.method == 'POST':

midia.delete()

return redirect('midia\_list')

return render(request, 'A\_Lei\_no\_NT/midia\_confirm\_delete.html', {'midia': midia})

def midia\_detalhe(request, id):

midia = get\_object\_or\_404(Midia, id=id)

return render(request, 'A\_Lei\_no\_NT/midia\_detalhe.html', {'midia': midia})

1. **urls.py:**

python

Copiar código

from django.contrib import admin

from django.urls import path

from . import views

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

path('', views.index, name='index'),

path('artigos/', views.artigo\_list, name='artigo\_list'),

path('artigos/novo/', views.artigo\_create, name='artigo\_create'),

path('artigos/<int:id>/', views.artigo\_detalhe, name='artigo\_detalhe'),

path('artigos/<int:id>/editar/', views.artigo\_update, name='artigo\_update'),

path('artigos/<int:id>/deletar/', views.artigo\_delete, name='artigo\_delete'),

path('autores/<int:id>/', views.autor\_detalhe, name='autor\_detalhe'),

path('autores/', views.autor\_list, name='autor\_list'),

path('autores/novo/', views.autor\_create, name='autor\_create'),

path('autores/<int:id>/editar/', views.autor\_update, name='autor\_update'),

path('autores/<int:id>/deletar/', views.autor\_delete, name='autor\_delete'),

path('areas/<int:id>/', views.area\_detalhe, name='area\_detalhe'),

path('areas/', views.area\_list, name='area\_list'),

path('areas/novo/', views.area\_create, name='area\_create'),

path('areas/<int:id>/editar/', views.area\_update, name='area\_update'),

path('areas/<int:id>/deletar/', views.area\_delete, name='area\_delete'),

path('midias/<int:id>/', views.midia\_detalhe, name='midia\_detalhe'),

path('midias/', views.midia\_list, name='midia\_list'),

path('midias/novo/', views.midia\_create, name='midia\_create'),

path('midias/<int:id>/editar/', views.midia\_update, name='midia\_update'),

path('midias/<int:id>/deletar/', views.midia\_delete, name='midia\_delete'),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

1. **Templates:** (Exemplo de artigo\_form.html)

html

Copiar código

{% load static %}

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% if form.instance.pk %}Editar Artigo{% else %}Novo Artigo{% endif %}</title>

<link rel="stylesheet" href="{% static 'A\_Lei\_no\_NT/style.css' %}">

</head>

<body>

<div class="container">

<header>

<h1>{% if form.instance.pk %}Editar Artigo{% else %}Novo Artigo{% endif %}</h1>

</header>

<section id="artigo-form">

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Salvar</button>

</form>

</section>

</div>

</body>

</html>

**Resumo de Transição do Projeto Albino Marks**

**06/07/2024**



**Objetivo**: O projeto Albino Marks é um aplicativo web desenvolvido em Django. Ele envolve a criação de páginas HTML que exibem diversos conteúdos textuais e biográficos.

**Estrutura do Projeto**

**Diretórios e Arquivos Principais**:

* A\_Lei\_no\_NT/ - Contém a aplicação Django.
  + migrations/ - Arquivos de migração do banco de dados.
  + static/ - Arquivos estáticos.
  + templates/
    - A\_ Lei\_no\_NT/ -
      * Templates HTML.
        + views.py - Arquivo de views.
        + urls.py - Configuração de URLs.
        + models.py - Modelos do banco de dados.
        + tests.py - Testes da aplicação.
* Albino\_Marks/ - Contém as configurações principais do projeto.
  + settings.py - Configurações do Django.
  + urls.py - Configuração de URLs principais.
  + wsgi.py - Configuração do servidor WSGI.

**Configurações e Implementações**

1. **Configuração do Logging**:
   * Para ajudar na depuração, configuramos o logging no settings.py:

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'handlers': {

'console': {

'class': 'logging.StreamHandler',

},

},

'loggers': {

'django': {

'handlers': ['console'],

'level': 'DEBUG',

},

},

}

1. **Problemas de Renderização de Páginas**:
   * Embora as migrations tenham sido aplicadas corretamente, houve um problema na renderização das páginas HTML.
   * Examinamos os arquivos views.py e os templates HTML, mas o problema persistiu.
2. **Uso de Debugging**:
   * Discutimos a importância de utilizar ferramentas de debugging para identificar o problema de renderização.
   * Propusemos a configuração de breakpoints e uso do debugger integrado do VS Code para inspecionar o fluxo de execução e variáveis.

**Próximos Passos Sugeridos**

1. **Debugging**:
   * Configurar o ambiente de debugging no VS Code.
   * Colocar breakpoints no arquivo views.py e seguir a execução do código para identificar onde está o problema.
2. **Verificação de Templates**:
   * Garantir que os templates HTML estão configurados corretamente.
   * Verificar se os caminhos dos templates estão corretos no settings.py.
3. **Inspeção das Views**:
   * Revisar as views para assegurar que estão retornando os templates corretos e passando os contextos necessários.
4. **Testes**:
   * Escrever e executar testes unitários e de integração para verificar o funcionamento das views e templates.

**Como Iniciar um Novo Chat**

1. **Inicie um novo chat na plataforma.**
2. **Prefixe suas mensagens com "[python]" para que eu possa identificar suas solicitações diretamente.**
3. **Compartilhe este resumo para que possamos continuar de onde paramos.**

Dessa forma, poderemos continuar o trabalho no novo chat sem perder o contexto do que já foi feito.