ROTEIRO DESENVOLVIMENTO DE PROJETO

Para iniciar um novo aplicativo no Django, você pode seguir estas etapas:

1. Certifique-se de que o Django esteja instalado

Antes de criar um novo aplicativo, você precisa garantir que o Django esteja instalado em seu ambiente. Se ainda não estiver instalado, você pode fazê-lo usando pip:

bash
pip install django

2. CRIE UM PROJETO NOVO

comando:

bash

django-admin startproject meu_projeto ← NOME DO SEU PROJETO

3. Navegue até o diretório do seu projeto

Abra o terminal (ou prompt de comando) e navegue até o diretório do seu projeto Django.

Por exemplo:

bash
cd /caminho/para/seu/projeto ← PASTA DO SEU PROJETO

4. Crie um novo aplicativo

Use o comando startapp para criar um novo aplicativo. Substitua nome_do_app pelo nome que você deseja dar ao seu aplicativo:

bash
python manage.py startapp nome_do_app

Este comando criará uma nova pasta chamada nome_do_app dentro do diretório do seu projeto, contendo a seguinte estrutura básica:

markdown
Copiar código
nome_do_app/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py

5. Adicione o aplicativo às configurações do projeto

Após criar o aplicativo, você precisa registrá-lo no seu projeto. Abra o arquivo settings.py do seu projeto e adicione o nome do aplicativo à lista INSTALLED_APPS:

python

```
INSTALLED_APPS = [
   'django.contrib.admin',
     'django.contrib.auth',
     'django.contrib.contenttypes',
     'django.contrib.sessions',
     'django.contrib.messages',
    'django.contrib.staticfiles',
    'nome_do_app', ← NOME DO SEU APP
]
```

6. Definição dos Modelos no models.py

Defina a estrutura do banco de dados da seguinte forma:

python

```
from django.db import models

class Empresa(models.Model):
   id_empresa = models.AutoField(primary_key=True)
   nome = models.CharField(max_length=100)
   logo = models.ImageField(upload_to='logos/', null=True, blank=True)
   createdAt = models.DateTimeField(auto_now_add=True)
   updatedAt = models.DateTimeField(auto_now=True)

   def __str__(self):
      return self.nome

class Publicacao(models.Model):
   id_publicacao = models.AutoField(primary_key=True)
   foto = models.ImageField(upload_to='publicacoes/', null=True, blank=True)
   titulo_prato = models.CharField(max_length=100)
   local = models.CharField(max_length=100)
   cidade = models.CharField(max_length=100)
   empresa = models.ForeignKey(Empresa, on_delete=models.CASCADE, related_name='publicacoes')
   createdAt = models.DateTimeField(auto_now_add=True)
   updatedAt = models.DateTimeField(auto_now=True)

   def __str__(self):
      return self.titulo_prato

class Usuario(models.Model):
   name = models.CharField(max_length=100)
   email = models.EmailField(unique=True)
   nickname = models.CharField(max_length=50)
   senha = models.CharField(max_length=100)  # Armazene as senhas de forma segura
   foto = models.ImageField(upload_to='fotos/', null=True, blank=True)
   createdAt = models.DateTimeField(auto_now_add=True)
   updatedAt = models.DateTimeField(auto_now=True)

   def __str__(self):
      return self.name

class Comentario(models.Model):
   usuario = models.ForeignKey(Usuario, on_delete=models.CASCADE, related_name='comentarios')
   publicacao = models.ForeignKey(Publicacao, on_delete=models.CASCADE,
related_name='comentarios')
   conteudo = models.TextField()
   createdAt = models.DateTimeField(auto_now_add=True)
```

```
    updatedAt = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f'Comentário de {self.usuario} na publicação {self.publicacao}'

class Curtida(models.Model):
    usuario = models.ForeignKey(Usuario, on_delete=models.CASCADE, related_name='curtidas')
    publicacao = models.ForeignKey(Publicacao, on_delete=models.CASCADE, related_name='curtidas')
    createdAt = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.usuario} curtiu {self.publicacao}'
```

7. Criar as Migrações

Depois de definir seus modelos, você precisa criar e aplicar as migrações:

1. **Criar Migrações:**

   bash

   python manage.py makemigrations

2. **Aplicar Migrações:**

   bash
   python manage.py migrate

8. Importar Dados dos Arquivos CSV

A importação de dados será semelhante ao que discutimos anteriormente. Contudo, é importante garantir que os dados importados preservem as relações definidas.

Criando um Comando Personalizado

No Django, para criar comandos personalizados como o de importação de dados, você precisa usar o framework de comandos de gerenciamento (management commands).

1. **Estrutura do Comando Personalizado**

   Crie a seguinte estrutura de diretórios dentro da sua aplicação Django (supondo que sua aplicação seja chamada seu_app):

   markdown
   Copiar código

```
seu_app/
├── management/
│   ├── __init__.py
│   ├── commands/
│       ├── __init__.py
│       ├── import_csv.py
```

Comando para Importação em import_csv.py:

Aqui está o código atualizado para import_csv.py,

Crie um arquivo de nome import_csv.py conforme acima informado. Primeiro crie dentro do seu app uma pasta de nome management e de pois dentro desta pasta crie outra pasta com o nome commands e dentro desta pasta crie o arquivo import.csv.py:

python

```
import csv
import os
from django.conf import settings # Para acessar BASE_DIR
from django.core.management.base import BaseCommand
from app.models import Usuario, Empresa, Publicacao


class Command(BaseCommand):
help = 'Importa dados de arquivos CSV para as tabelas do banco de dados'

def handle(self, *args, **kwargs):
# Caminho para os arquivos CSV
caminho_usuarios = os.path.join(settings.BASE_DIR, 'usuario.csv')
caminho_empresas = os.path.join(settings.BASE_DIR, 'empresa.csv')
caminho_publicacoes = os.path.join(settings.BASE_DIR, 'publicacao.csv')

# Importar dados de usuarios.csv
self.stdout.write(self.style.NOTICE('Importando dados de usuarios.csv...'))
with open(caminho_usuarios, newline='', encoding='utf-8') as csvfile:
reader = csv.DictReader(csvfile)
for row in reader:
Usuario.objects.create(
name=row['nome'],
email=row['email'],
nickname=row['nickname'],
senha=row['senha'],
foto=row['foto'], # Certifique-se de que as fotos estão acessíveis
)
self.stdout.write(self.style.SUCCESS('Dados de usuários importados com sucesso!'))

# Importar dados de empresa.csv
self.stdout.write(self.style.NOTICE('Importando dados de empresa.csv...'))
with open(caminho_empresas, newline='', encoding='utf-8') as csvfile:
reader = csv.DictReader(csvfile)
for row in reader:
Empresa.objects.create(
nome=row['nome'],
logo=row['logo'], # Certifique-se de que os logos estão acessíveis
)
self.stdout.write(self.style.SUCCESS('Dados de empresas importados com sucesso!'))

# Importar dados de publicacao.csv
self.stdout.write(self.style.NOTICE('Importando dados de publicacao.csv...'))
with open(caminho_publicacoes, newline='', encoding='utf-8') as csvfile:
reader = csv.DictReader(csvfile)
```

```
for row in reader:
try:
empresa = Empresa.objects.get(id_empresa=row['empresa_id']) # Buscando a empresa relacionada
Publicacao.objects.create(
foto=row['foto'],
titulo_prato=row['titulo_prato'],
local=row['local'],
cidade=row['cidade'],
empresa=empresa, # Relacionando a empresa
)
except Empresa.DoesNotExist:
self.stdout.write(self.style.WARNING(f'Empresa com id {row["empresa_id"]} não existe. Pulando...'))
continue
self.stdout.write(self.style.SUCCESS('Dados de publicações importados com sucesso!'))
```

realize todas as indentações necessárias

Importante: Substitua 'caminho/para/...' pelos caminhos reais dos arquivos CSV.

Executar o Comando

Após criar o comando, você pode executá-lo com o seguinte comando no terminal:

bash

```
python manage.py import_csv
```

9. Exportar a Estrutura e Dados para um Arquivo .SQL

Para exportar a estrutura e os dados do banco de dados para um arquivo .SQL, você pode usar o seguinte comando:

bash

```
python manage.py dumpdata --output=backup.json
```

Se precisar de um arquivo SQL especificamente, você pode usar ferramentas como pg_dump para PostgreSQL ou mysqldump para MySQL:

Para PostgreSQL:

bash

```
pg_dump -U usuario -W -F p nome_do_banco > backup.sql
```

Para MySQL:

bash

```
mysqldump -u db.sqlite3 > backup.sql
```

10. Crie as migrações (se necessário)

Se você planeja adicionar modelos ao seu aplicativo, você deve criar migrações. Para isso, você pode definir seus modelos no arquivo models.py do aplicativo. Depois de definir os modelos, você pode criar as migrações com:

bash
Copiar código
python manage.py makemigrations nome_do_app

E aplicar as migrações com:

bash
Copiar código
python manage.py migrate

11. CRIE O STATICFILES

Dentro da pasta raiz crie uma pasta com o nome STATIC

No arquivo settings.py configura a rota para a pasta static

Obs.: importe o pacote OS no settings.py conforme abaixo e instale as configurações do static

import os

python

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = [
os.path.join(BASE_DIR, "static")
]
```

isso permite que suas imagens sejam renderizadas na paginas chamando o {% load static %}

12. Crie suas views, models e templates

Agora que seu aplicativo está criado e registrado, você pode começar a desenvolver suas views, modelos e templates conforme necessário.

Dentro do settings.py configure o TEMPLATES para direcionar para a pasta templates

```
TEMPLATES = [
{
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
        'context_processors': [
                        'django.template.context_processors.debug',
                        'django.template.context_processors.request',
                        'django.contrib.auth.context_processors.auth',
                        'django.contrib.messages.context_processors.messages',
                        ],
        },
```

},
]

crie um pasta com o nome templates dentro da pasta rais do projeto.

Insira todos os seus htmls dentro desta pasta de forma organizada. Se necessário crie subpastas para tornar mais facil a localização do componente que está sendo trabalhado.

13. CRIE O AMBIENTE DE LOGIN E SENHA CONFORME ABAIXO:

Crie o aplicativo de autenticação

python manage.py startapp autentication

uma vez que foi criado um nome aplicativo insira ele no installed_apps no settings.py

```
INSTALLED_APPS = [
        .....
        'autentication',
}
```

PASSO 1: criar views de login e logout

Django fornece views prontas para login e logout, que você pode personalizar e utilizar.

1. Configurar a Rota de Login:

No arquivo urls.py da raíz do projeto , adicione as rotas para login e logout.

python

```
from django.urls import path
from django.contrib.auth import views as auth_views
from autentication.views import register
from django.views.generic import TemplateView

urlpatterns = [
        path('dashboard/', TemplateView.as_view(template_name='dashboard.html'),
name='dashboard'),
        path('register/', register, name='register'),
        path('login/', auth_views.LoginView.as_view(template_name='login.html'),
name='login'),
```

```
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

• LoginView: Responsável por exibir e processar o login.

• LogoutView: Faz o logout do usuário, encerrando a sessão.

2. Criar o Template de Login:

Crie o arquivo templates/login.html para renderizar o formulário de login.

No Django, você pode usar a template language para exibir o formulário.

html

Copiar código

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Login</title>
</head>
<body>
<h2>Login</h2>
<form method="POST">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Entrar</button>
</form>
</body>
</html>
```

O formulário {{ form.as_p }} é gerado automaticamente com os campos username e password.

PASSO 2: criar view de registro de usuário

Agora vamos criar uma view para permitir que novos usuários se registrem no sistema.

1. View de Registro: Crie uma view para o registro de novos usuários.

No arquivo views.py do app autenticacao, adicione a lógica para processar o formulário de registro.

python

Copiar código

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm

def register(request):
if request.method == 'POST':
form = UserCreationForm(request.POST)
if form.is_valid():
form.save()return redirect('login') # Redireciona para a página de login após o registro
else:
form = UserCreationForm()
return render(request, 'register.html', {'form': form})
```

2. Criar o Template de Registro: Crie o template templates/register.html para exibir o formulário de registro.

html

Copiar código

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Registrar-se</title>
</head>
<body>
<h2>Registrar-se</h2>
<form method="POST">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Registrar</button>
</form>
</body>
</html>
```

3. Template de Dashboard: Crie o template templates/dashboard.html para renderizar essa página.

html

Copiar código

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Painel</title>
</head>
<body>
<h2>Bem-vindo ao Painel, {{ user.username }}</h2>
<p>Esta página está protegida e só pode ser acessada por usuários
logados.</p>
</body>
</html>
```

4. Redirecionar Usuários para o Login: Se um usuário não estiver logado e tentar acessar a página dashboard, ele será redirecionado para a página de login. Para definir a página de redirecionamento após o login ou logout, configure as URLs no settings.py.

Python

```
LOGIN_REDIRECT_URL = 'dashboard'
LOGOUT_REDIRECT_URL = 'login'
LOGIN_URL = 'login' # Se o usuário tentar acessar uma página protegida sem estar logado
```

7. Executar o servidor de desenvolvimento

Para verificar se tudo está funcionando, você pode executar o servidor de desenvolvimento do Django:

bash
Copiar código

```
python manage.py runserver
```

Depois, você pode acessar seu projeto em http://127.0.0.1:8000/.

Conclusão

Essas são as etapas básicas para iniciar um novo aplicativo no Django. A partir daqui, você pode personalizar e expandir seu aplicativo conforme necessário. Se tiver dúvidas ou precisar de mais informações, sinta-se à vontade para perguntar!