DJANGO

- Django é um framework web Python de código aberto.

- Simplifica a criação de soluções web escaláveis, promove o desenvolvimento rápido e um design limpo.

- Amplamente adotado para o desenvolvimento de modelos de machine learning, inteligência artificial e análise de dados.

- O Django surgiu com a ideia de fornecer uma estrutura robusta com recursos para facilitar o processo de desenvolvimento para web em Python.

**O que é um framework?**

- São ferramentas que fornecem estruturas prontas para poupar o trabalho da pessoa desenvolvedora de ter que desenvolver algo comum do zero.

- Oferecem soluções prontas para a maior parte das tarefas repetitivas que costumavam exigir execução manual.

Para entender melhor o que é um framework: <https://www.alura.com.br/artigos/framework-o-que-e-pra-que-serve-essa-ferramenta>

**Para que serve o Django?**

- Simplifica e acelera a criação de aplicações web robustas, escaláveis e seguras.

- Pode ser usado em criação de sites e aplicações web, plataformas de redes sociais, sistemas de gerenciamento interno, aplicações e-commerce, aplicações backend para aplicativos mobile, construção de APIs etc.

- O Django simplifica o processo de desenvolvimento web, permitindo que as pessoas desenvolvedoras foquem na criação de funcionalidades, de acordo com seu contexto específico.

**O que é o Django REST Framework e qual sua relação com APIs?**

- O Django REST Framework foi especialmente concebido para simplificar o desenvolvimento de APIs RESTful, oferecendo recursos e ferramentas dedicadas a esse propósito.

- É comum a recomendação de sua utilização ao construir APIs em projetos Django.

- Alguns motivos para essas indicações: autenticação e autorização, serialização simplificada, documentação automática, views e endpoints predefinidos, validação de entrada.

Para saber mais sobre a diferença entre Django e Django REST: <https://www.alura.com.br/artigos/django-django-rest-diferencas>

**Características do Django**

Arquitetura MVT: Model, View e Template, representando, respectivamente, a lógica de negócios, a lógica de exibição e a renderização da interface do usuário. Essa estrutura facilita a criação de aplicações, promovendo a separação das atividades e reutilização de partes do código.

Segurança: É um framework que se preocupa bastante com segurança e fornece recursos para prevenir ameaças e incluir proteções como validação automática de formulários, gerenciamento de autenticação etc.

Mapeamento ORM: permite que as pessoas desenvolvedoras interajam com o banco de dados usando classes e objetos em Python, sem precisar escrever diretamente para o SQL. Isso simplifica o acesso de dados e ajuda a evitar problemas de segurança ao mesmo tempo que melhora a legibilidade do código.

Interface de Administração: disponibiliza um painel administrativo pronto para uso, o que facilita o gerenciamento de dados da aplicação. Esse painel é automaticamente gerado com base nos modelos definidos no projeto, e permite que as pessoas administradoras realizem operações CRUD, sem a necessidade de desenvolver uma interface personalizada do zero. Isso facilita a manutenção e torna a experiência de quem administra mais agradável, além de diminuir o tempo de trabalho.

Mapeamento de URL e URLs amigáveis: O Django permite associar URLs específicas às views correspondentes nas aplicações, através do mapeamento de URL. Isso fornece uma maneira mais organizada de direcionar as solicitações do usuário para as partes apropriadas do código. Além disso, o Django facilita a criação de URLs limpas, estruturadas de forma semântica e amigável ao usuário, melhorando a experiência e otimizando os mecanismos de busca.

Sistemas de template: O Django fornece uma maneira poderosa e flexível de separar a lógica de apresentação do código backend, através de arquivos que definem a estrutura e o visual das páginas web. Isso permite que os desenvolvedores e desenvolvedoras criem interfaces dinâmicas sem ter que misturar código de backend com código de frontend.

O Template Engine é o mecanismo que realiza o processamento dos arquivos de template, tornando o conteúdo dinâmico. Ele permite a inclusão de lógica simples, como loops e condicionais, diretamente nos templates, sem exigir linguagens de programação complexas, simplificando a criação de layouts mais elaborados.

Sistemas de cache e caching: É possível armazenar temporariamente dados acessados com frequência em uma área de memória de alta velocidade para acelerar o acesso subsequente. Isso melhora o desempenho da aplicação, reduzindo a carga sobre o banco de dados e os recursos do servidor.

O caching pode ser aplicado em vários níveis, desde o cache de página inteira até o cache de fragmentos de página ou de consultas de banco de dados específicas.

Internacionalização: As aplicações criadas podem ser facilmente adaptadas para suportar múltiplos idiomas e regiões. Com a internacionalização, mensagens e textos exibidos na interface do usuário podem ser traduzidos e ajustados de acordo com a preferência das pessoas usuárias.

O Django oferece ferramentas para identificar, extrair e traduzir strings de texto, bem como para ajustar formatos de data, hora e números para diferentes localizações. Isso possibilita a criação de aplicações verdadeiramente globais e acessíveis a públicos diversos.

Apresentamos apenas algumas das principais características que tornam o Django um framework mais atrativo para pessoas desenvolvedoras. Agora que já compreendemos isso, vem a pergunta: como é que esse framework funciona?

**Componentes da arquitetura MVT**

**Model**

É responsável por representar a estrutura e a lógica de dados da aplicação. Ele define como os dados serão acessados, manipulados e armazenados, assim como faz a interação com o banco de dados.

O Django utiliza o [ORM (Object-Relational Mapping)](https://www.alura.com.br/conteudo/orm-nodejs-api-sequelize-mysql), que permite que você utilize classes e objetos no Python para mapear as tabelas do banco de dados, fazendo a criação, consulta, atualização e exclusão de dados com mais segurança e facilidade.

Isso permite que você trabalhe com banco de dados relacionais sem ter a necessidade de lidar com SQL diretamente. Além disso, você pode trabalhar com vários sistemas de gerenciamento de banco de dados relacionais utilizando a mesma sintaxe e API.

A seguir podemos ver um exemplo de um modelo de um Livro para ser feito no banco de dados:

from django.db import models

class Livro(models.Model):

nome = models.CharField(max\_length=100)

preco = models.DecimalField(max\_digits=8, decimal\_places=2)

descricao = models.TextField()

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

return self.nome

O modelo “Livro” define três instâncias da classe, denominadas de: nome, preço e descrição. Cada campo tem sua definição como “CharField” para texto curto, “DecimalField” para números decimais e “TextField” para textos longos. É possível ainda definir métodos dentro desse modelo.

Como podemos verificar no código acima, cada classe de modelo representa uma tabela no banco de dados e define os campos e métodos relacionados a essa tabela. Os campos da classe de modelo correspondem às colunas da tabela no banco de dados, enquanto as instâncias da classe representam linhas da tabela no banco de dados,

**View**

É responsável por processar as solicitações [HTTP](https://www.alura.com.br/artigos/desmistificando-o-protocolo-http-parte-1) do usuário e retornar as respostas desse processamento. As views são funções e classes que lidam com a lógica de negócios da aplicação, sendo projetadas para serem reutilizáveis e utilizadas em diferentes aplicações de um mesmo projeto.

As views recebem as solicitações do usuário e consultam ou manipulam os dados nos modelos para retornar uma resposta HTTP adequada à solicitação, como uma página [HTML](https://www.alura.com.br/artigos/html), um arquivo JSON, um arquivo de imagem, dentre outros diversos formatos.

Dando prosseguimento ao projeto dos Livros, o código referente às views teria o seguinte formato:

from django.shortcuts import render

from .models import Livro

def lista\_de\_livros(request):

livros = Livro.objects.all()

return render(request, 'loja/lista\_de\_livros.html', {'livros': livros})

O arquivo de views.py contém então, as funções de view que controlam o processamento da lógica de apresentação. Nesse exemplo, a função lista\_de\_livro() obtém todos os livros do banco de dados usando Livro.objects.all(). Ela então usa a função render() para renderizar um template HTML e passa os livros como contexto para serem usados no template.

**Template**

Como o próprio nome indica, é responsável pela apresentação e pela renderização dos dados para o usuário. Eles são arquivos que contêm um código HTML com marcações especiais do Django, que permitem que você insira dinamicamente os dados dos modelos e crie páginas dinâmicas e interativas.

Para o exemplo anterior, podemos verificar um template do estilo abaixo:

<!DOCTYPE html>

<html>

<head>

<title>Lista de Livros</title>

</head>

<body>

<h1>Lista de Livros</h1>

<ul>

{% for livro in livros %}

<li>

<h2>{{ livros.nome }}</h2>

<p>Preço: R$ {{ livros.preco }}</p>

<p>{{ livros.descricao }}</p>

</li>

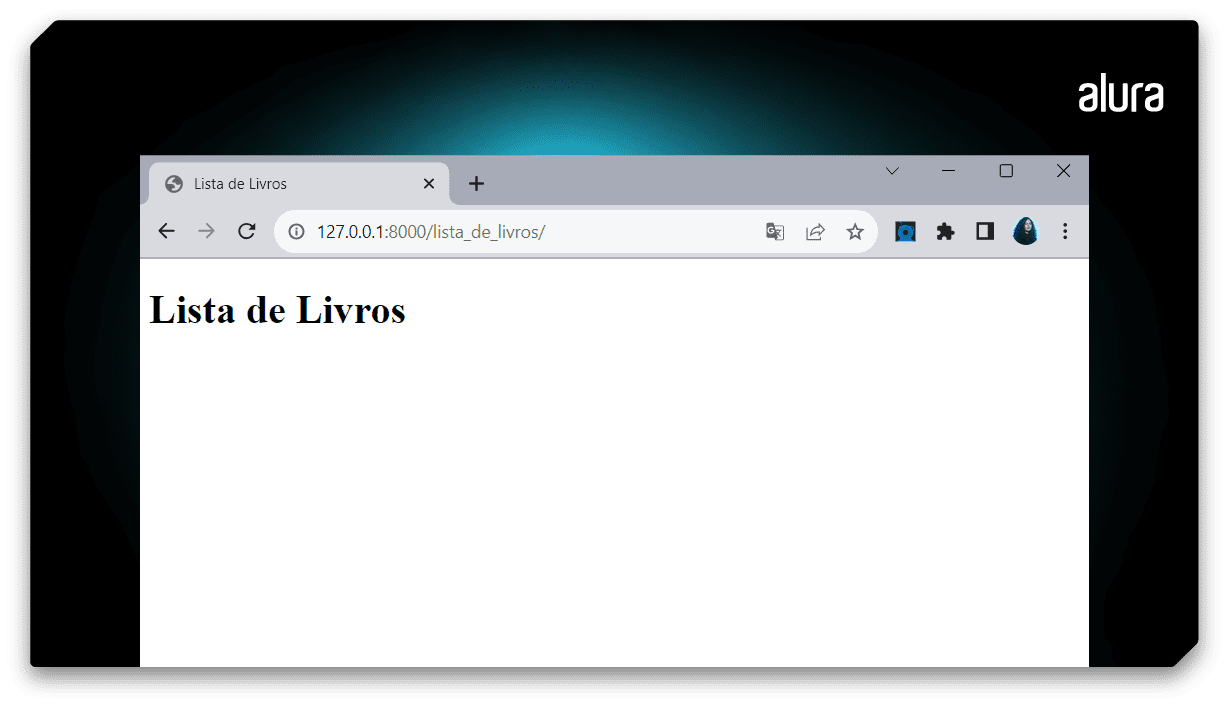
{% endfor %}

</ul>

</body>

</html>

Usamos a linguagem de template do Django, delimitada por {% ... %} para expressões de lógica e {{ ... }} para variáveis, para inserir a lógica de apresentação. Esse template base teria o seguinte formato no nosso navegador:



Agora que sabemos quais são as funcionalidades desses três componentes no Django, podemos analisar como funciona a interação entre eles.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Como usar**

**O que é preciso para aprender Django?**

Para utilizar o framework Django você precisará ter os seguintes itens instalados em sua máquina:

* **Python:** como já vimos, o Django é um framework de desenvolvimento web escrito em Python. Você pode conferir como fazer a instalação no próximo tópico;
* **PIP:** é o gerenciador de pacotes padrão do Python, sendo fundamental para instalação de pacotes e bibliotecas adicionais. É por meio do PIP que realizamos a instalação do Django;
* **Django:** para que você consiga trabalhar com esse framework, será necessário que ele esteja instalado em sua máquina;

Pode parecer muita coisa à primeira vista, mas não se preocupe. Vamos indicar aqui como realizar cada uma dessas instalações.

**Instalação do Python**

A linguagem Python é distribuída sob a Licença Python de código aberto, permitindo que você utilize o Python gratuitamente. Para fazer a instalação, você pode acessar diretamente o [site oficial](https://www.python.org/downloads/) ou seguir o passo a passo disponibilizado em nosso artigo sobre a linguagem [Python](https://www.alura.com.br/artigos/python#primeiros-passos-e-recursos-basicos). Lá você também encontrará o tutorial de instalação para diferentes sistemas operacionais, como Windows, Linux e macOS.

É importante mencionar que o Django requer o Python 3.6 ou posterior, então, certifique-se de instalar uma versão mais atualizada do Python ou verificar a [FAQ: Instalação](https://docs.djangoproject.com/en/4.2/faq/install/#what-python-version-can-i-use-with-django) do framework para garantir que a versão selecionada do Python é suportada para a versão do Django que você pretende instalar do Django.

**Instalação do PIP**

Geralmente, o PIP já vem instalado junto com o Python. Contudo, é possível que ocorra algum problema em sua instalação, por isso, é sempre importante verificar se você tem o PIP instalado usando o seguinte comando no terminal ou prompt de comando:

pip *--version*

Caso o PIP não esteja instalado em sua máquina, você pode realizar sua instalação seguindo o [tutorial da página oficial](https://pip.pypa.io/en/stable/installation/) desse gerenciador ou utilizando o passo a passo a seguir:

1. **Usando o ensurepip**

Ao instalar o Python, ele vem com um módulo chamado [ensurepip](https://docs.python.org/pt-br/3/library/ensurepip.html" \l "module-ensurepip" \t "_blank), que oferece suporte para inicializar o instalador do PIP em uma instalação existente do Python ou em um ambiente virtual. Dentro de sua home, utilize o comando:

**Para Windows:**

C:> py -m ensurepip --upgrade

**Para Linux:**

$ python -m ensurepip *--upgrade*

ou

sudo apt install python3-pip

**Para macOs:**

$ python -m ensurepip *--upgrade*

2.) **Utilizando o get-pip.py**

Esse é um script que usa uma lógica de inicialização para instalar o PIP. Para executar esse comando, é preciso que você baixe o [script](https://bootstrap.pypa.io/get-pip.py) e abra um terminal ou uma prompt de comando na pasta que contém o arquivo get-pip.py e, na sequência, execute o seguinte comando:

**Para Windows:**

C:> py get-pip.py

**Para Linux:**

$ python get-pip.py

**Para macOs:**

$ python get-pip.py

**Instalação do Django**

O Django é um framework de código aberto distribuído sob a Licença BSD (Berkeley Software Distribution), o que significa que você pode usá-lo sem nenhum custo.

Para garantir que o processo de instalação seja organizado e seguro, é altamente recomendado utilizar uma venv (Virtual Environment, ou Ambiente Virtual, em português). A venv é uma ferramenta Python que cria ambientes virtuais isolados, permitindo que você gerencie as dependências do seu projeto de forma independente do ambiente global do Python em sua máquina.

Usar uma venv (Virtual Environment) ao programar em Django é uma boa prática recomendada por várias razões, tais como:

* **Isolamento do ambiente:** uma venv permite isolar seu ambiente de desenvolvimento do sistema operacional e de outros projetos. Isso significa que as dependências, pacotes e versões de bibliotecas instaladas em uma venv específica não afetarão outros projetos ou o sistema globalmente.

Isso é bastante útil quando você está trabalhando em vários projetos em paralelo ou quando precisa lidar com diferentes versões de bibliotecas ou pacotes, pois evita conflitos de dependências e facilita o gerenciamento dos pacotes do seu projeto.

* **Reprodutibilidade e portabilidade:** ao criar uma venv, você pode documentar as dependências do seu projeto em um arquivo requirements.txt, o que permite que outras pessoas, incluindo você, reproduzam exatamente o mesmo ambiente de desenvolvimento em suas máquinas.

Isso facilita bastante o processo de colaboração em um projeto e a implantação de uma aplicação em diferentes ambientes, como servidores de produção, por exemplo. Além disso, você pode facilmente mover seu projeto para outro ambiente ou máquina. Basta copiar a venv para a nova máquina e ativar o ambiente para ter acesso a todas as dependências e configurações necessárias para o projeto.

* \*Facilidade de limpeza:\*\* se você decidir encerrar ou excluir um projeto Django, basta excluir a venv associada a ele para remover completamente todas as dependências e arquivos relacionados ao projeto. Isso simplifica a limpeza e evita deixar resíduos indesejados no sistema.

Sendo assim, o uso de um ambiente virtual é uma prática altamente recomendada para garantir a consistência e a organização do seu ambiente de desenvolvimento. Desse modo, antes de mostrar como fazer a instalação do Django, iremos realizar a criação de uma venv.

**Como criar uma venv**

Para realizar essa criação, você pode seguir o passo a passo mostrado no artigo [ambiente virtual em python](https://www.alura.com.br/artigos/ambientes-virtuais-em-python). Lembre-se de criar uma nova pasta para o seu projeto Django e utilizar essa pasta na criação de seu ambiente, podendo colocar como nome da venv algo que esteja relacionado ao seu projeto.

**Instalando o Django**

Com sua venv já ativada, verifique isso observando o nome do ambiente entre parênteses na linha de comando, você pode instalar o Django utilizando o pip, como mostra o comando a seguir:

pip install django

O pip baixará e instalará a versão mais recente do Django no ambiente virtual. Para baixar uma versão específica do Django, use o comando pip seguido do nome do pacote django e da versão desejada. O formato para especificar a versão é: <nome\_do\_pacote>==<versão>.

Por exemplo, se você deseja instalar o Django na versão 3.2.5, execute o seguinte comando:

pip install django==3.2.5

E então, o PIP baixará e instalará a versão 3.2.5 do Django no seu ambiente Python. Uma vez concluída a instalação, você terá o Django pronto para uso. Você pode verificar se o Django foi instalado corretamente executando o seguinte comando:

python -m django *--version*

Isso exibirá a versão do Django instalada em sua máquina.

**Como usar o Django**

Antes de nos aprofundarmos nas complexidades do Django, é importante entender sua lógica e estrutura. Esta poderosa estrutura de desenvolvimento web, baseada em Python, oferece flexibilidade e eficiência. A seguir, veremos como usar o Django, desde a configuração inicial até funções mais complexas.

**Criando um projeto no Django**

Agora que a instalação foi concluída, você pode criar um novo projeto Django. Ainda no terminal e com a venv ativada, execute o seguinte comando:

django-admin startproject <nome\_do\_projeto>.

Certifique-se de substituir nome\_do\_projeto pelo nome que você deseja dar ao seu projeto. Esse comando criará uma pasta com o nome escolhido, com uma estrutura de diretório inicial e alguns arquivos Python dentro dela, seguindo o seguinte formato:

.

├── <nome\_do\_projeto>/

├── \_\_init\_\_.py

├── asgi.py

├── settings.py

├── urls.py

└── wsgi.py

Para finalizar a configuração de seu projeto, você pode alterar no arquivo settings.py, o idioma e o horário que deseja utilizar em sua aplicação, substituindo as seguintes linhas de código:

LANGUAGE\_CODE = 'pt-br'

TIME\_ZONE = 'America/Sao\_Paulo'

Após isso, navegue para o diretório onde está o arquivo manage.py do seu projeto, e execute o seguinte comando:

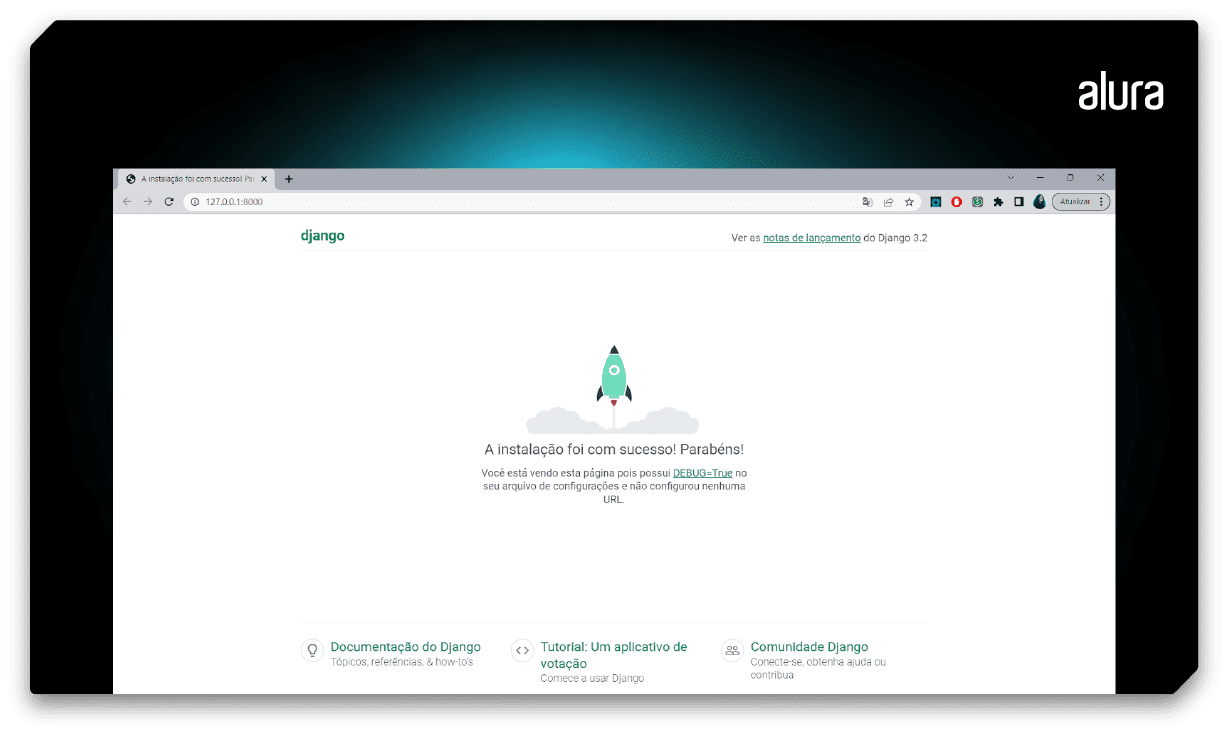
python manage.py runserver

Isso iniciará o servidor de desenvolvimento do Django e você poderá acessar o seu aplicativo web em: http://localhost:8000/ no seu navegador, ou na URL disponibilizada no seu terminal, como mostra a imagem abaixo:

Tela de computador com letras brancas em fundo preto

Descrição gerada automaticamente

Ao acessar essa URL sem realizar nenhuma alteração em sua aplicação, você terá acesso à tela padrão do Django com uma mensagem de: “A instalação foi feita com sucesso”. Além disso, serão exibidas algumas informações de notas de lançamento da versão que você está utilizando do framework e links para documentação, tutoriais e comunidade do Django, como mostrado na imagem a seguir:



Para encerrar esse servidor Django e voltar para seu terminal, basta utilizar o atalho ctrl + C. Dessa maneira, você concluiu o primeiro passo para construção de um projeto utilizando o Django. Agora você pode realizar as alterações que desejar e iniciar os trabalhos em sua aplicação!

**Criando sua aplicação**

Com seu projeto em Django criado com sucesso, o próximo passo fundamental é a criação da aplicação. A estrutura modular do Django permite que você divida seu projeto em pequenas partes funcionais chamadas de “aplicativos”. Essa abordagem organizada facilita o desenvolvimento e a manutenção do projeto, tornando-o altamente escalável.

Para criar um aplicativo, basta você navegar pela pasta modular de seu projeto:

cd <nome\_do\_projeto>

E execute o seguinte comando para criar um novo aplicativo:

python manage.py startapp <nome\_do\_aplicativo>

Lembre-se de substituir o <nome\_do\_aplicativo> pelo nome desejado de seu app. Esse processo criará uma pasta com o nome de seu aplicativo, uma estrutura de diretório inicial e alguns arquivos Python dentro dela, no mesmo diretório que a pasta de seu projeto, como mostrado a seguir:

.

├── <nome\_do\_aplicativo>/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── migrations/

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

│ └── views.py

└── <nome\_do\_projeto>/

├── \_\_init\_\_.py

├── asgi.py

├── settings.py

├── urls.py

└── wsgi.py

É importante destacar que a estrutura de diretório inicial pode variar de acordo com os aplicativos e módulos adicionais que você criar em seu projeto. Essa é a estrutura básica de um projeto Django. Você pode adicionar outros diretórios e arquivos conforme necessário para o seu projeto específico.

Agora basta você adicionar o seu aplicativo à lista INSTALLED\_APPS no arquivo settings.py, dentro da pasta do seu projeto, como mostrado a seguir:

INSTALLED\_APPS = [

*# ...*

'<nome\_do\_aplicativo>',

*# ...*

]

Pronto, sua aplicação já está pronta para ser desenvolvida! Agora, podemos fazer as migrações e avançar na criação de nosso projeto.

**Fazendo migrações**

As migrações são um recurso do Django utilizado para gerenciar alterações no esquema de banco de dados. Ao criar, modificar ou excluir modelos, são realizadas as migrações para que essas alterações sejam refletidas no banco de dados, mantendo-o sincronizado com seu projeto. Isso garante que você faça modificações e evolua sua aplicação de maneira ordenada e reversível.

As migrações são armazenadas como arquivos Python no diretório migrations, dentro de cada aplicação do projeto, com nomes sequenciais, indicando a ordem em que foram criadas. Para realizar esse processo, você deve criar uma migração usando o seguinte comando em seu terminal:

python manage.py makemigrations

Esse comando fará uma análise das alterações feitas nos modelos e criará um arquivo de migração que contém as instruções necessárias para atualizar o esquema do banco de dados. Uma vez criada, você poderá aplicar essa migração, ou qualquer outra migração pendente, no banco de dados através do seguinte comando:

python manage.py migrate

Além disso, se por alguma razão for necessário desfazer uma migração realizada no banco de dados, você poderá utilizar o comando:

python manage.py migrate <nome\_do\_aplicativo> <nome\_da\_migracao>

Assim, suas alterações aplicadas pela migração realizada serão revertidas. O rastreio de histórico de alterações feitas no banco de dados e a praticidade na realização das migrações, facilitam muito a manutenção e evolução de suas aplicações em Django.

**Escolhendo o banco de dados**

Uma das primeiras decisões a serem tomadas ao iniciar um projeto em Django, é escolher o banco de dados que você utilizará. A escolha do banco de dados dependerá das necessidades específicas do seu projeto, como: escalabilidade, desempenho, recursos e requisitos do sistema.

Por padrão, o banco de dados utilizado pelo Django ao criar um projeto é o [SQLite](https://www.alura.com.br/artigos/sqlite-da-instalacao-ate-primeira-tabela). Dessa forma, o Django já vem configurado para utilizar esse banco para testes e desenvolvimento, o que é uma opção adequada para projetos pequenos, pois permite criar aplicações sem a necessidade de configurar um servidor de banco de dados.

Em caso de projetos que necessitam de opções mais robustas e em ambientes de produção, você pode escolher sistemas de gerenciamento de banco de dados relacionais, como o PostgreSQL, por exemplo, para obter alguns recursos mais avançados ou o [MySQL](https://www.alura.com.br/conteudo/mysql-manipule-dados-com-sql), conhecido por ser rápido e escalável.

Algumas outras opções também estão disponíveis para serem utilizadas, como:

* [Oracle Database](https://www.alura.com.br/curso-online-manipulacao-dados-oracle-dml);
* [Microsoft SQL Server](https://www.alura.com.br/curso-online-microsoft-sql-server-conhecendo-sql);
* MariaDB.

Além disso, existem backends de terceiros para outros sistemas de banco de dados, como Firebird, CockroachDB, TiDB, entre outros. Caso esteja em dúvida sobre o suporte, escolha e configuração do banco, é sempre recomendado acessar a [documentação](https://docs.djangoproject.com/en/4.2/ref/databases/) do Django sobre esse tópico.

Se você pretende usar um banco de dados diferente do SQLite, para configurar o Django na adoção do banco de dados de sua escolha, você pode atualizar as configurações no arquivo settings.py do projeto, adequando-o ao banco desejado.

O primeiro passo que você precisa fazer é garantir que o driver adequado esteja instalado no seu ambiente Python. Por exemplo, se você deseja usar o PostgreSQL, instale o pacote psycopg2:

pip install psycopg2

Concluída a instalação do driver desejado, o próximo passo é abrir o arquivo settings.py e procurar a configuração do banco de dados, que está definida na variável DATABASES. Por padrão, a configuração do SQLite é a seguinte:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

Para alterar o banco de dados, substitua a configuração ENGINE pelo driver adequado e forneça as informações de conexão, como nome do banco de dados, usuário, senha, host e porta (se necessário). Por exemplo, para usar o PostgreSQL:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'mydatabase',

'USER': 'myuser',

'PASSWORD': 'mypassword',

'HOST': 'localhost',

'PORT': '',

}

}

Após configurar o novo banco de dados, é necessário criar as migrações e aplicá-las para que as tabelas e esquemas do banco de dados sejam criados ou atualizados de acordo. Para isso, você pode utilizar os seguintes comandos em seu terminal:

python manage.py makemigrations

python manage.py migrate

Depois de realizar a configuração e as migrações, execute o servidor de desenvolvimento do Django e teste a conexão para garantir que tudo esteja funcionando corretamente. Lembre-se de que ao alterar o banco de dados padrão do Django, pode ser necessário que você instale e configure o servidor de banco de dados correspondente em sua máquina ou servidor de hospedagem.

**Trabalhando com o Django Admin**

Como falamos anteriormente, o Django possui um sistema de administração incorporado que facilita a criação de uma interface de administração para gerenciar os dados do seu aplicativo, sem a necessidade de escrever um código personalizado.

Para utilizar essa interface, o primeiro passo necessário é realizar o registro dos modelos que você deseja gerenciar no Admin. Você pode fazer isso acessando o arquivo admin.py do seu aplicativo e configurando, por exemplo, da seguinte forma:

from django.contrib import admin

from .models import <modelo>

admin.site.register(<modelo>)

Feito isso, antes de acessar a interface de administração, é necessário criar um superusuário. Você pode usar o seguinte comando para isso:

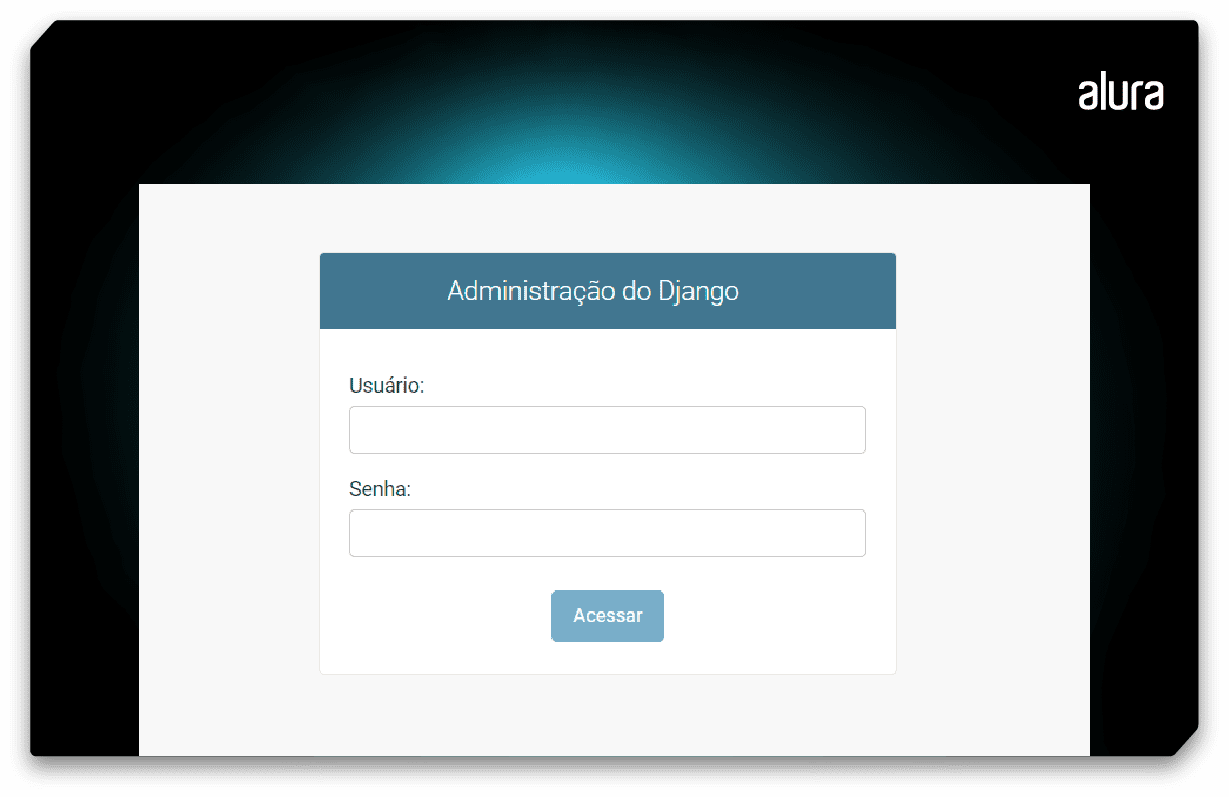
python manage.py createsuperuser

Siga as instruções para definir um nome de usuário ou usuária, endereço de e-mail e senha. Durante a digitação, por questões de segurança, sua senha não será exibida no visor do terminal. Portanto, tenha cuidado ao digitá-la. Além disso, há uma verificação de senha, então, é recomendado criar uma senha forte. Após essa etapa, você receberá a mensagem de “Superuser created successfully” (em português, “Superusuário criado com sucesso”), como mostrado na imagem a seguir:

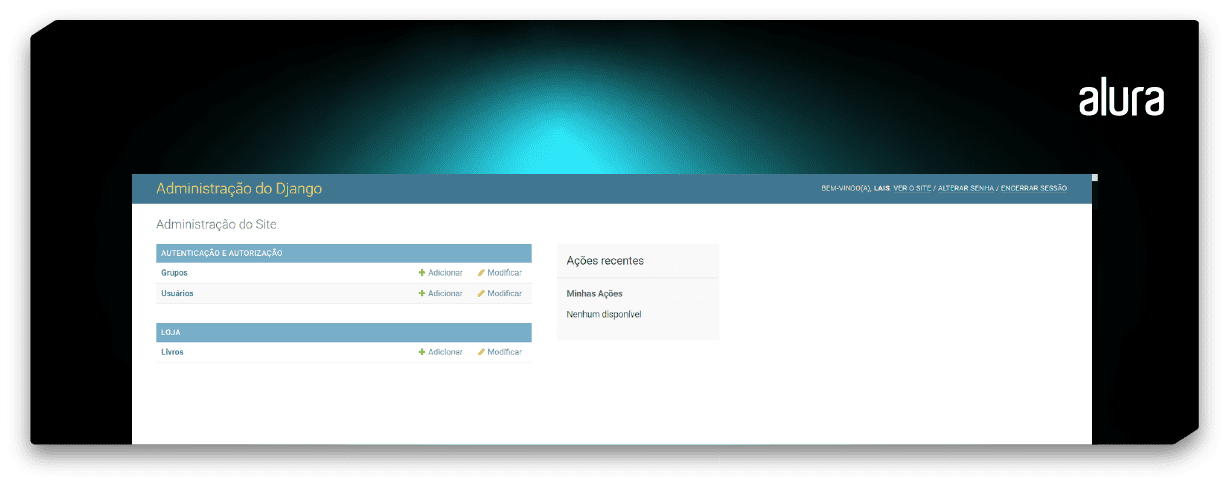
Texto

Descrição gerada automaticamente

Finalizado esse processo, você poderá executar o servidor do Django e acessar a interface de administração no navegador, normalmente em: http://localhost:8000/admin/. Ela terá o seguinte formato:



Faça o login com as credenciais do superusuário criado anteriormente e teste as funcionalidades de seu projeto. Para o projeto de Livros que apresentamos no tópico ‘Como funciona o Django’, teríamos a seguinte interface:



Repare que o admin.py foi configurado de modo a identificar os dados do modelo da seguinte maneira:

from django.contrib import admin

from .models import Livro

class LivroAdmin(admin.ModelAdmin):

lista = ('nome', 'preco', 'descricao') *# Define os campos a serem exibidos na lista*

admin.site.register(Livro, LivroAdmin)

**Criando URLs**

Para criar URLs em um projeto Django, você precisa definir as rotas (URLs) que correspondem às funções de visualização (views) em seu aplicativo. As rotas são definidas em um arquivo chamado urls.py em cada aplicativo, e também podem ser configuradas no arquivo urls.py do projeto principal.

No arquivo urls.py em seu aplicativo, você deve importar os módulos necessários para criação das rotas. Se ainda não tiver um, crie um arquivo urls.py dentro da pasta do aplicativo. O código ficará no seguinte formato:

from django.urls import path

from . import views

urlpatterns = [

path('', views.<nome\_da\_funcao\_de\_visualizacao>, name='<nome\_da\_url>'),

*# Outras URLs podem ser definidas aqui*

]

Analisando o seu funcionamento, o sinal '' no primeiro atributo da função path(), corresponde à URL raiz do aplicativo. Você pode alterá-lo para um caminho específico se desejar que a visualização seja acessada em uma URL específica.

Além disso, substitua <nome\_da\_funcao\_de\_visualizacao> pelo nome da função de visualização que você deseja chamar quando a URL for acessada, e também o <'nome\_da\_url'> para atribuir um nome à URL, o que é útil para referenciá-la em modelos e em outras partes do código Django.

Para o exemplo do projeto de Livros, teríamos:

from django.urls import path

from . import views

urlpatterns = [

path('lista\_de\_livros/', views.lista\_de\_livros, name='lista\_de\_livros'),

]

Ou seja, quando a URL de rota http://127.0.0.1:8000/lista\_de\_livros for acessada, a função de visualização lista\_de\_livros será chamada em seu código. Adiante, você pode incluir as URLs do aplicativo no arquivo urls.py do projeto principal. Para fazer isso, adicione um caminho que inclua o nome do aplicativo:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

path('<nome\_do\_aplicativo>/', include('<nome\_do\_aplicativo>.urls')),

]

Neste exemplo, todas as URLs definidas em <nome\_do\_aplicativo>.urls serão acessíveis através do caminho http://seusite.com/<nome\_do\_aplicativo>/. Para o exemplo de Livros, você teria o seguinte código:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

path('', include('loja.urls')),

]

Observe que o nome do aplicativo é loja. Depois de definir suas URLs, você pode acessá-las em seu navegador ou em seu código, usando o nome atribuído à URL (por exemplo, {% url 'nome\_da\_url' %} em um template). Certifique-se de ajustar os nomes dos aplicativos, funções de visualização e URLs de acordo com a estrutura do seu projeto.

**Como fazer deploy**

Para realizar o deploy de uma aplicação em Django, ou seja, colocar seu projeto em um ambiente de produção para que ele esteja disponível na internet e acessível aos usuários e usuárias, você precisa realizar uma escolha baseada em duas condições: conhecimentos e recursos disponíveis.

Existem várias opções para realizar o deploy, como indicamos a seguir:

* [**AWS**](https://aws.amazon.com/pt/codedeploy/): a Amazon Web Services, em especial a AWS Elastic Beanstalk, oferece um plano gratuito limitado a 12 meses para uso de suas funcionalidades. Depois desse período, os custos variam de acordo com os recursos a serem utilizados e o tráfego. Acesse a [documentação](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html) para aprender a fazer o deploy;
* [**PythonAnywhere**](https://www.pythonanywhere.com/)**:** oferece planos gratuitos e pagos para hospedar aplicações Django. O plano gratuito tem algumas restrições, como uso limitado de recursos e domínios personalizados. Os planos pagos oferecem mais recursos e funcionalidades adicionais. Acesse a [documentação](https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/) para aprender a fazer o deploy;
* [**Microsoft Azure App Service**](https://azure.microsoft.com/en-us/free/apps/)**:** a Microsoft fornece um plano gratuito com recursos limitados. Você pode consultar os planos pagos para obter mais recursos de acordo com sua necessidade. Acesse a [documentação](https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/tutorial-django-app-service-postgres?tabs=clone) para aprender a fazer o deploy;
* [**Render**](https://render.com/)**:** o Render é uma plataforma de serviço em nuvem para fazer o deploy e executar suas aplicações de forma gratuita. Acesse a [documentação](https://render.com/docs/deploy-django) para aprender a fazer o deploy.

Claro, a lista apresentada aqui inclui apenas algumas opções que você pode utilizar para fazer o deploy de suas aplicações. Existem diversas outras plataformas, como a Fly.io, Heroku, Railway, Google App Engine, entre outros provedores de serviços privados.

É muito importante lembrar que os planos gratuitos geralmente possuem limitações de recursos, tais como largura de banda, capacidade de armazenamento, horas de atividade e suporte técnico. Para projetos maiores ou com necessidades específicas, pode ser necessário investir em um plano pago para obter melhor desempenho e recursos mais robustos.

Antes de escolher um provedor de hospedagem, é importante revisar seus recursos, valores e suporte para garantir que atendam às necessidades do seu projeto em termos de escala, desempenho e suporte técnico. Além disso, lembre-se que as opções e preços podem mudar com o tempo, então, sempre verifique os sites oficiais dos provedores para obter as informações mais atualizadas.

**Escolhendo uma IDE**

Para tirar o máximo proveito do processo de desenvolvimento e torná-lo mais produtivo e eficiente, é altamente recomendável utilizar uma IDE (Integrated Development Environment).

Uma [IDE](https://www.alura.com.br/artigos/o-que-e-uma-ide) oferece um ambiente mais robusto, completo e otimizado para desenvolvimento de software. Ela conta com recursos avançados de edição, depuração, gerenciamento de projetos, integração com sistemas de controle de versão entre outras diversas vantagens, que facilitam e aprimoram o processo de criação de aplicações em Django.

Dessa forma, é importante escolher qual IDE você utilizará no desenvolvimento de seus projetos, a depender de suas necessidades e escolhas pessoais. Existem diversas IDEs que você pode optar para desenvolver um aplicativo utilizando esse framework, como mostrado a seguir:

1. **Visual Studio Code (VSCode ou VSC):** é uma IDE gratuita e altamente extensível desenvolvida pela Microsoft. O [VSCode](https://www.alura.com.br/artigos/visualstudio-code-instalacao-teclas-de-atalho-plugins-e-integracoes) possui uma extensa comunidade e suporte para várias extensões que facilitam o desenvolvimento com Django. Você pode fazer o download gratuito na [página oficial](https://code.visualstudio.com/download) desse editor.
2. **Visual Studio:** é uma IDE também desenvolvida pela Microsoft que oferece suporte ao desenvolvimento de aplicativos Python e Django, com recursos avançados de depuração e edição. Você pode fazer o download na [página oficial](https://visualstudio.microsoft.com/pt-br/downloads/) em que você pode optar pelo plano gratuito Community, que já oferece recursos satisfatórios para desenvolvimento, ou utilizar os planos pagos profissionais como o Professional e o Enterprise que apresentam recursos adicionais.
3. **PyCharm:** trata-se de uma IDE focada para o desenvolvimento em Python criada pela JetBrains. Ela oferece muitos recursos avançados específicos para o Django, como suporte a modelos, autocompletar, navegação entre arquivos e muito mais. Você pode fazer o download na [página oficial](https://www.jetbrains.com/pt-br/pycharm/download/?section=windows), optando pelo plano gratuito PyCharm Community Edition, que já oferece recursos satisfatórios para desenvolvimento, ou selecionando o plano pago PyCharm Professional, que disponibiliza recursos adicionais.
4. **Sublime Text:** é um editor de texto que pode ser estendido para desempenhar o papel de uma IDE Python com plugins e pacotes de extensão específicos. Você pode fazer o download gratuito do [Sublime Text](https://www.alura.com.br/artigos/sublime-text-um-dos-melhores-editores-de-codigo) na [página oficial](https://www.sublimetext.com/3) desse editor.

É claro que há diversas outras opções que não foram citadas, pois priorizamos as mais conhecidas e utilizadas. A escolha da IDE depende das suas preferências pessoais, dos recursos que você precisa e de sua familiaridade com a interface da IDE.

Todas as opções mencionadas acima, são ótimas escolhas e bem conceituadas pela comunidade de pessoas desenvolvedoras em Python. Experimente algumas delas para selecionar aquela que melhor atende às suas necessidades e fluxo de trabalho.

**Suporte Django: documentação, repositório e comunidade**

O Django, assim como o Python, possui um suporte sólido e ativo, o que proporciona às pessoas desenvolvedoras uma variedade de recursos para obter ajuda, suporte técnico e uma documentação bem estruturada.

Você pode acessar diretamente o [Repositório do Django](https://github.com/django/django) ou acessar sua [documentação oficial](https://docs.djangoproject.com/pt-br/4.2/) para acessar tutoriais, guias, referências, exemplos e informações gerais.

Além disso, a comunidade Django é acolhedora e bastante ativa. Você pode se unir à [Comunidade do Django no Discord](https://discord.gg/xcRH6mN4fa) ou no [Fórum do Django](https://forum.djangoproject.com/), para compartilhar suas dúvidas, fazer questionamentos, e buscar ajuda para resolver possíveis problemas.

**Onde posso aprender mais sobre Django**

Caso você queira mergulhar nessa tecnologia, mas não saiba por onde começar a estudar, não deixe de acessar o [Tech Guide Python - Django](https://techguide.sh/pt-BR/path/python/django/). Nele, você vai ter acesso aos materiais necessários para te guiar nessa jornada com o Django com estudos indo do iniciante ao avançado.

Confira também o episódio da série Guia de Carreiras do #HipstersPontoTube sobre o Python e aprenda mais sobre essa tecnologia, suas vantagens e o que é necessário para iniciar sua profissão nesta área de desenvolvimento de software.

**Aprenda mais sobre Django gratuitamente**

Acesse gratuitamente as primeiras aulas dos cursos feitos pela [Escola de Programação](https://www.alura.com.br/escola-programacao) da Alura e continue aprendendo sobre temas como:

* [Python: começando com a linguagem](https://www.alura.com.br/conteudo/python-introducao-a-linguagem)
* [Python: avançando na linguagem](https://www.alura.com.br/conteudo/python-3-avancando-na-linguagem)
* [Python: entendendo a Orientação a Objetos](https://www.alura.com.br/conteudo/python-entendendo-orientacao-objetos)
* [Python: avançando na orientação a objetos](https://www.alura.com.br/conteudo/python-3-avancando-orientacao-objetos)
* [Django: templates e boas práticas](https://www.alura.com.br/conteudo/django-templates-boas-praticas)
* [Django: persistência de dados e Admin](https://www.alura.com.br/conteudo/django-persistencia-dados-admin)
* [Django: autenticação de formulários e alerta](https://www.alura.com.br/conteudo/django-autenticacao-formularios-alerta)
* [Django: CRUD e persistência no S3](https://www.alura.com.br/conteudo/django-crud-persistencia-s3)
* [API com Django 3: Django Rest Framework](https://www.alura.com.br/conteudo/api-django-3-rest-framework)
* [API com Django 3: Validações, buscas, filtros e deploy](https://www.alura.com.br/conteudo/api-django-3-validacoes-buscas-filtros-deploy)
* [API com Django 3: Versionamento, cabeçalhos e CORS](https://www.alura.com.br/conteudo/api-django-3-versionamento-cabecalhos-cors)