Será composto por duas partes:

* Um site público que permite que as pessoas vejam as enquetes e votem nelas.
* Um site de administração que permite adicionar, alterar e excluir enquetes.

Vamos supor que você já tenha o [Django instalado](https://docs.djangoproject.com/en/4.1/intro/install/) . Você pode dizer que o Django está instalado e qual versão executando o seguinte comando em um prompt de shell (indicado pelo prefixo $):

**python -m django --version**

Se o Django estiver instalado, você deverá ver a versão da sua instalação. Se não estiver, você receberá um erro dizendo “Nenhum módulo chamado django”.

**Pip install Django**

Este tutorial foi escrito para Django 4.1, que suporta Python 3.8 e posterior. Se a versão do Django não corresponder, você pode consultar o tutorial da sua versão do Django usando o seletor de versão no canto inferior direito desta página ou atualizar o Django para a versão mais recente.

Criando um projeto [**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial01/#creating-a-project)

Se esta for sua primeira vez usando o Django, você terá que cuidar de algumas configurações iniciais. Ou seja, você precisará gerar automaticamente algum código que estabeleça um projeto Django – uma coleção de configurações para uma instância do Django, incluindo configuração do banco de dados, opções específicas do Django e configurações específicas do aplicativo.

Na linha de comando, **cd** em um diretório onde você gostaria de armazenar seu código, execute o seguinte comando:

**$ django-admin startproject mysite .**

Isso criará umdiretório **mysite** em seu diretório atual.

**Observação**

Você precisará evitar nomear projetos com base em componentes Python ou Django integrados. Em particular, isso significa que você deve evitar o uso de nomes como **django**(que entrará em conflito com o próprio Django) ou **test**(que entrará em conflito com um pacote Python embutido).

**Onde esse código deve ficar?**

Se o seu plano de fundo estiver em PHP antigo simples (sem o uso de estruturas modernas), você provavelmente está acostumado a colocar o código na raiz do documento do servidor da Web (em um local como **/var/www**). Com Django, você não faz isso. Não é uma boa ideia colocar nenhum desses códigos Python na raiz de documentos do seu servidor da web, pois isso arrisca a possibilidade de que as pessoas possam visualizar seu código na web. Isso não é bom para a segurança.

Coloque seu código em algum diretório **fora** da raiz do documento, como **/home/mycode**.

Vejamos o que criou [**startproject**](https://docs.djangoproject.com/en/4.1/ref/django-admin/#django-admin-startproject):

mysite/

manage.py

mysite/

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

Esses arquivos são:

* **mysite/**O diretório raiz externo é um contêiner para seu projeto. Seu nome não importa para Django; você pode renomeá-lo para o que quiser.
* **manage.py**: Um utilitário de linha de comando que permite interagir com este projeto Django de várias maneiras. Você pode ler todos os detalhes sobre **manage.py**em [django-admin e manage.py](https://docs.djangoproject.com/en/4.1/ref/django-admin/) .
* O diretório interno **mysite/**é o pacote Python real para o seu projeto. Seu nome é o nome do pacote Python que você precisará usar para importar qualquer coisa dentro dele (por exemplo, **mysite.urls**).
* **mysite/\_\_init\_\_.py**: Um arquivo vazio que informa ao Python que esse diretório deve ser considerado um pacote Python. Se você é iniciante em Python, leia [mais sobre pacotes](https://docs.python.org/3/tutorial/modules.html#tut-packages) nos documentos oficiais do Python.
* **mysite/settings.py**: Definições/configuração para este projeto Django. [As configurações do Django](https://docs.djangoproject.com/en/4.1/topics/settings/) lhe dirão tudo sobre como as configurações funcionam.
* **mysite/urls.py**: As declarações de URL para este projeto Django; um “índice” do seu site desenvolvido com Django. Você pode ler mais sobre URLs em [URL dispatcher](https://docs.djangoproject.com/en/4.1/topics/http/urls/) .
* **mysite/asgi.py**: Um ponto de entrada para servidores da Web compatíveis com ASGI para servir ao seu projeto. Consulte [Como implantar com ASGI](https://docs.djangoproject.com/en/4.1/howto/deployment/asgi/) para obter mais detalhes.
* **mysite/wsgi.py**: um ponto de entrada para servidores da Web compatíveis com WSGI para servir ao seu projeto. Consulte [Como implantar com WSGI](https://docs.djangoproject.com/en/4.1/howto/deployment/wsgi/) para obter mais detalhes.

O servidor de desenvolvimento [**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial01/#the-development-server)

Vamos verificar se o seu projeto Django funciona. Mude para o diretório externo **mysite**, se ainda não o fez, e execute os seguintes comandos:

**$ py manage.py runserver**

Você verá a seguinte saída na linha de comando:

|  |
| --- |
| You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for ap  p(s): admin, auth, contenttypes, sessions.  Run 'python manage.py migrate' to apply them.  December 09, 2022 - 15:41:54  Django version 4.1.4, using settings 'mysite.settings'  Starting development server at http://127.0.0.1:8000/  Quit the server with CTRL-BREAK. |

Você iniciou o servidor de desenvolvimento Django, um servidor web leve escrito exclusivamente em Python. Incluímos isso no Django para que você possa desenvolver coisas rapidamente, sem ter que lidar com a configuração de um servidor de produção – como o Apache – até que esteja pronto para produção.

Agora é um bom momento para observar: **não** use este servidor em nada que se assemelhe a um ambiente de produção. Destina-se apenas para uso durante o desenvolvimento. (Nosso negócio é criar estruturas da web, não servidores da web.)

Agora que o servidor está em execução, visite <http://127.0.0.1:8000/> com seu navegador da web. Você verá um "Parabéns!" página, com um foguete decolando. Funcionou!

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

|  |
| --- |
| **Mudando a porta**  Por padrão, o comando [**runserver**](https://docs.djangoproject.com/en/4.1/ref/django-admin/#django-admin-runserver) inicia o servidor de desenvolvimento no IP interno na porta 8000.  Se você deseja alterar a porta do servidor, passe-a como um argumento de linha de comando. Por exemplo, este comando inicia o servidor na porta 8080:  py manage.py runserver 8080 |

## Criando o aplicativo de enquetes [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial01/#creating-the-polls-app)

Agora que seu ambiente – um “projeto” – está configurado, você está pronto para começar a trabalhar.

Cada aplicativo que você escreve em Django consiste em um pacote Python que segue uma certa convenção. O Django vem com um utilitário que gera automaticamente a estrutura básica de diretórios de um aplicativo, para que você possa se concentrar em escrever código em vez de criar diretórios.

Seus aplicativos podem residir em qualquer lugar em seu [caminho Python](https://docs.python.org/3/tutorial/modules.html#tut-searchpath) . Neste tutorial, criaremos nosso aplicativo de enquete no mesmo diretório do seu **manage.py**arquivo para que ele possa ser importado como seu próprio módulo de nível superior, em vez de um submódulo de **mysite**.

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

py manage.py startapp polls

Isso criará um diretório **polls**, que é definido assim:

Essa estrutura de diretório abrigará o aplicativo de votação.

## Escreva sua primeira visualização [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial01/#write-your-first-view)

Vamos escrever a primeira visualização. Abra o arquivo **polls/views.py** e coloque o seguinte código Python nele:

|  |
| --- |
| **from** **django.http** **import** HttpResponse  **def** index(request):  **return** HttpResponse("Hello, world. You're at the polls index.") |

Esta é a visão mais simples possível no Django.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente Para chamar a view, precisamos mapeá-la para uma URL - e para isso precisamos de um URLconf.

Para criar um URLconf no diretório de polls, crie um arquivo chamado **urls.py**. O diretório do seu aplicativo agora deve se parecer com:

|  |
| --- |
| **from** **django.urls** **import** path  **from** **.** **import** views  urlpatterns = [  path('', views.index, name='index'),  ] |

A próxima etapa é apontar o **URLconf raiz** para módulo **polls.urls**.

Em **mysite/urls.py**, adicione um import for **django.urls.include** e insira um [**include()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.include) na lista **urlpatterns**, assim você terá:

|  |
| --- |
| **from** **django.contrib** **import** admin  **from** **django.urls** **import** include, path  urlpatterns = [  path('polls/', include('polls.urls')),  path('admin/', admin.site.urls),  ] |

A função [**include()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.include)permite referenciar outros URLconfs. Sempre que o Django encontra [**include()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.include), ele corta qualquer parte da URL correspondente até aquele ponto e envia a string restante para o URLconf incluído para processamento adicional.

A ideia por trás [**include()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.include)é facilitar o plug-and-play de URLs.

Como as enquetes estão em seu próprio URLconf ( **polls/urls.py**), elas podem ser colocadas em “/polls/”, ou em “/fun\_polls/”, ou em “/content/polls/”, ou qualquer outro caminho raiz, e o aplicativo ainda trabalhar.

**Quando usar**[**include()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.include)

Você sempre deve usar **include()**quando incluir outros padrões de URL. **admin.site.urls** é a única exceção a isso.

Agora você conectou uma visualização **index** ao URLconf. Verifique se está funcionando com o seguinte comando:

py manage.py runserver

Acesse <http://localhost:8000/polls/> em seu navegador e você deverá ver o texto “ Hello, world. Você está no índice de pesquisas. ”, que você definiu na  exibição **index**.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Página não encontrada?**

Se você receber uma página de erro aqui, verifique se está indo para <http://localhost:8000/polls/> e não para <http://localhost:8000/> .

Configuração do banco de dados [**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#database-setup)

Agora, abra **mysite/settings.py**. É um módulo Python normal com variáveis ​​de nível de módulo que representam as configurações do Django.

Por padrão, a configuração usa SQLite. Se você é novo em bancos de dados ou está apenas interessado em experimentar o Django, esta é a escolha mais fácil. O SQLite está incluído no Python, então você não precisará instalar mais nada para dar suporte ao seu banco de dados. Ao iniciar seu primeiro projeto real, no entanto, você pode querer usar um banco de dados mais escalável como o PostgreSQL, para evitar dores de cabeça com a troca de banco de dados no futuro.

Se você deseja usar outro banco de dados, instale as [vinculações de banco](https://docs.djangoproject.com/en/4.1/topics/install/#database-installation) de dados apropriadas e altere as seguintes chaves no item para corresponder às configurações de conexão do banco de dados: [**DATABASES**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DATABASES) **'default'**

* [**ENGINE**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DATABASE-ENGINE)– Ou **'django.db.backends.sqlite3'**, **'django.db.backends.postgresql'**, **'django.db.backends.mysql'**ou **'django.db.backends.oracle'**. Outros back-ends [também estão disponíveis](https://docs.djangoproject.com/en/4.1/ref/databases/#third-party-notes) .
* [**NAME**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-NAME)– O nome do seu banco de dados. Se você estiver usando SQLite, o banco de dados será um arquivo em seu computador; nesse caso, [**NAME**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-NAME) deve ser o caminho absoluto completo, incluindo o nome do arquivo, desse arquivo. O valor padrão, , armazenará o arquivo no diretório do seu projeto.**BASE\_DIR / 'db.sqlite3'**

Se você não estiver usando SQLite como seu banco de dados, configurações adicionais como [**USER**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-USER), [**PASSWORD**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-PASSWORD) e [**HOST**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-HOST) devem ser adicionadas. Para obter mais detalhes, consulte a documentação de referência do [**DATABASES**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DATABASES).

Exemplo de configuração de Db com Mysq, PostgreSQL, MariDB

|  |
| --- |
| DATABASES = {  ‘default’: {  ‘ENGINE’: ‘django.db.backends.postgresql’,  ‘NAME’: ‘mydatabase’,  ‘USER’: ‘mydatabaseuser’,  ‘PASSWORD’: ‘mypassword’,  ‘HOST’: ‘127.0.0.1’,  'PORT': '5432',  }  } |

Enquanto estiver editando **mysite/settings.py**, defina [**TIME\_ZONE**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-TIME_ZONE) para o seu fuso horário.

Além disso, observe a configuração [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS) na parte superior do arquivo. Isso contém os nomes de todos os aplicativos Django que são ativados nesta instância do Django. Os aplicativos podem ser usados ​​em vários projetos e você pode empacotá-los e distribuí-los para uso por outras pessoas em seus projetos.

Por padrão, contém [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS) contém os seguintes aplicativos, todos com Django:

* [**django.contrib.admin**](https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#module-django.contrib.admin)– O site de administração. Você vai usá-lo em breve.
* [**django.contrib.auth**](https://docs.djangoproject.com/en/4.1/topics/auth/#module-django.contrib.auth)– Um sistema de autenticação.
* [**django.contrib.contenttypes**](https://docs.djangoproject.com/en/4.1/ref/contrib/contenttypes/#module-django.contrib.contenttypes)– Uma estrutura para tipos de conteúdo.
* [**django.contrib.sessions**](https://docs.djangoproject.com/en/4.1/topics/http/sessions/#module-django.contrib.sessions)– Uma estrutura de sessão.
* [**django.contrib.messages**](https://docs.djangoproject.com/en/4.1/ref/contrib/messages/#module-django.contrib.messages)– Uma estrutura de mensagens.
* [**django.contrib.staticfiles**](https://docs.djangoproject.com/en/4.1/ref/contrib/staticfiles/#module-django.contrib.staticfiles)– Uma estrutura para gerenciar arquivos estáticos.

Esses aplicativos são incluídos por padrão como uma conveniência para o caso comum.

Alguns desses aplicativos usam pelo menos uma tabela de banco de dados, portanto, precisamos criar as tabelas no banco de dados antes de podermos usá-las. Para fazer isso, execute o seguinte comando:

py manage.py migrate

O comando **[migrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-migrate)** analisa a configuração [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS) e cria quaisquer tabelas de banco de dados necessárias de acordo com as configurações do banco de dados em seu arquivo **mysite/settings.py** e as migrações de banco de dados enviadas com o aplicativo (abordaremos isso mais tarde). Você verá uma mensagem para cada migração aplicada.

## Criando modelos [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#creating-models)

Agora definiremos seus modelos – essencialmente, o layout do banco de dados, com metadados adicionais.

|  |
| --- |
| **Filosofia**  Um modelo é a fonte única e definitiva de informações sobre seus dados. Ele contém os campos e comportamentos essenciais dos dados que você está armazenando. Django segue o [princípio DRY](https://docs.djangoproject.com/en/4.1/misc/design-philosophies/#dry) . O objetivo é definir seu modelo de dados em um só lugar e derivar coisas dele automaticamente.  Isso inclui as migrações - ao contrário do Ruby On Rails, por exemplo, as migrações são totalmente derivadas de seu arquivo de modelos e são essencialmente um histórico que o Django pode percorrer para atualizar seu esquema de banco de dados para corresponder aos seus modelos atuais. |

Em nosso aplicativo de enquete, criaremos dois modelos: **Question** e **Choice**. A **Question** tem uma pergunta e uma data de publicação. A **Choice**tem dois campos: o texto da escolha e uma contagem de votos. Cada um **Choice** está associado a um arquivo **Question**.

Esses conceitos são representados por classes Python. Edite o arquivo **polls/models.py** para que fique assim:

|  |
| --- |
| **from** **django.db** **import** models  **class** **Question**(models.Model):  question\_text = models.CharField(max\_length=200)  pub\_date = models.DateTimeField('date published')  **class** **Choice**(models.Model):  question = models.ForeignKey(Question, on\_delete=models.CASCADE)  choice\_text = models.CharField(max\_length=200)  votes = models.IntegerField(default=0) |

Aqui, cada modelo é representado por uma classe que subclasse **[django.db.models.Model](https://docs.djangoproject.com/en/4.1/ref/models/instances/" \l "django.db.models.Model" \o "django.db.models.Model)**. Cada modelo tem um número de variáveis ​​de classe, cada uma representando um campo de banco de dados no modelo.

Cada campo é representado por uma instância de uma [**Field**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field) classe – por exemplo, **[CharField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.CharField" \o "django.db.models.CharField)** para campos de caracteres e **[DateTimeField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.DateTimeField" \o "django.db.models.DateTimeField)** para datas e horas. Isso diz ao Django que tipo de dados cada campo contém.

O nome de cada instância [**Field**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field) (por exemplo , **question\_text** ou **pub\_date**) é o nome do campo, em formato amigável à máquina. Você usará esse valor em seu código Python e seu banco de dados o usará como o nome da coluna.

Você pode usar um primeiro argumento posicional opcional para  [**Field**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field) designar um nome legível por humanos. Isso é usado em algumas partes introspectivas do Django e funciona como documentação. Se este campo não for fornecido, o Django usará o nome legível por máquina. Neste exemplo, definimos apenas um nome legível para **Question.pub\_date**. Para todos os outros campos neste modelo, o nome legível por máquina do campo será suficiente como seu nome legível por humanos.

Algumas classes  [**Field**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field) exigem argumentos. **[CharField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.CharField" \o "django.db.models.CharField)**, por exemplo, requer que você dê a ele um **[max\_length](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.CharField.max_length" \o "django.db.models.CharField.max_length)**. Isso é usado não apenas no esquema do banco de dados, mas também na validação, como veremos em breve.

A [**Field**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field) também pode ter vários argumentos opcionais; neste caso, definimos o [**default**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.Field.default) valor de **votes** como 0.

Por fim, observe que um relacionamento é definido, usando **[ForeignKey](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)**. Isso diz ao Django que cada um **Choice** está relacionado a um único arquivo **Question**. O Django suporta todos os relacionamentos comuns de banco de dados: muitos para um, muitos para muitos e um para um.

Ativando modelos [**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#activating-models)

Esse pequeno pedaço de código de modelo fornece ao Django muitas informações. Com ele, o Django é capaz de:

* Crie um esquema de banco de dados ( instruções) para este aplicativo.**CREATE TABLE**
* Crie uma API de acesso ao banco de dados Python para acessar **Question**e **Choice**objetos.

Mas primeiro precisamos informar ao nosso projeto que o aplicativo  **polls** está instalado.

|  |
| --- |
| **Filosofia**  Os aplicativos Django são “conectáveis”: você pode usar um aplicativo em vários projetos e pode distribuir aplicativos, porque eles não precisam estar vinculados a uma determinada instalação do Django. |

Para incluir o aplicativo em nosso projeto, precisamos adicionar uma referência à sua classe de configuração no arquivo settings [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS). A classe **PollsConfig** está no arquivo **polls/apps.py**, então seu caminho pontilhado é **'polls.apps.PollsConfig'**. Edite o **mysite/settings.py**arquivo e adicione esse caminho pontilhado à configuração [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS). Vai ficar assim:

|  |
| --- |
| INSTALLED\_APPS = [  'polls.apps.PollsConfig',  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  ] |

Agora o Django sabe incluir o app  **polls**. Vamos executar outro comando:

py manage.py makemigrations polls

|  |
| --- |
| Migrations for 'polls':  polls/migrations/0001\_initial.py  - Create model Question  - Create model Choice |

Ao executar **makemigrations**, você está dizendo ao Django que fez algumas alterações em seus models (neste caso, você fez novos) e que gostaria que as alterações fossem armazenadas como migration .

As migrações são como o Django armazena as alterações em seus modelos (e, portanto, em seu esquema de banco de dados) - são arquivos em disco. Você pode ler a migração para seu novo modelo, se quiser; é o arquivo **polls/migrations/0001\_initial.py**. Não se preocupe, não se espera que você os leia toda vez que o Django fizer um, mas eles foram projetados para serem editáveis ​​por humanos, caso você queira ajustar manualmente como o Django muda as coisas.

Há um comando que executará as migrações para você e gerenciará seu esquema de banco de dados automaticamente - chamado **[migrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-migrate)**, e falaremos sobre ele em um momento - mas primeiro, vamos ver qual SQL essa migração executaria. O **[sqlmigrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-sqlmigrate)** comando pega nomes de migração e retorna seu SQL:

py manage.py sqlmigrate polls 0001

|  |
| --- |
| **BEGIN**;  *--*  *-- Create model Question*  *--*  **CREATE** **TABLE** "polls\_question" (  "id" bigint **NOT** **NULL** **PRIMARY** **KEY** **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY**,  "question\_text" varchar(200) **NOT** **NULL**,  "pub\_date" **timestamp** **with** **time** **zone** **NOT** **NULL**  );  *--*  *-- Create model Choice*  *--*  **CREATE** **TABLE** "polls\_choice" (  "id" bigint **NOT** **NULL** **PRIMARY** **KEY** **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY**,  "choice\_text" varchar(200) **NOT** **NULL**,  "votes" integer **NOT** **NULL**,  "question\_id" bigint **NOT** **NULL**  );  **ALTER** **TABLE** "polls\_choice"  **ADD** **CONSTRAINT** "polls\_choice\_question\_id\_c5b4b260\_fk\_polls\_question\_id"  **FOREIGN** **KEY** ("question\_id")  **REFERENCES** "polls\_question" ("id")  **DEFERRABLE** **INITIALLY** **DEFERRED**;  **CREATE** **INDEX** "polls\_choice\_question\_id\_c5b4b260" **ON** "polls\_choice" ("question\_id");  **COMMIT**; |

Observe o seguinte:

* A saída exata irá variar dependendo do banco de dados que você está usando. O exemplo acima é gerado para PostgreSQL.
* Os nomes das tabelas são gerados automaticamente combinando o nome do aplicativo ( **polls**) e o nome minúsculo do modelo – **question**e **choice**. (Você pode substituir esse comportamento.)
* Chaves primárias (IDs) são adicionadas automaticamente. (Você também pode substituir isso.)
* Por convenção, o Django anexa **"\_id"**ao nome do campo de chave estrangeira. (Sim, você também pode substituir isso.)
* A relação de chave estrangeira é explicitada por uma restrição. Não se preocupe com as peças; está dizendo ao PostgreSQL para não impor a chave estrangeira até o final da transação.**FOREIGN KEYDEFERRABLE**
* Ele é adaptado ao banco de dados que você está usando, portanto, tipos de campos específicos do banco de dados, como **auto\_increment**(MySQL), (PostgreSQL) ou (SQLite), são tratados automaticamente para você. O mesmo vale para aspas de nomes de campo – por exemplo, usando aspas duplas ou aspas simples.**bigint PRIMARY KEY GENERATED BY DEFAULT AS IDENTITYinteger primary key autoincrement**
* Na verdade **[sqlmigrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-sqlmigrate)**, o comando não executa a migração em seu banco de dados - em vez disso, ele a imprime na tela para que você possa ver o que o SQL Django considera necessário. É útil para verificar o que o Django fará ou se você tiver administradores de banco de dados que requerem scripts SQL para alterações.

Se você estiver interessado, também pode correr ; isso verifica se há algum problema em seu projeto sem fazer migrações ou mexer no banco de dados. [**py manage.py check**](https://docs.djangoproject.com/en/4.1/ref/django-admin/#django-admin-check)

Agora, execute novamente **[migrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-migrate)** para criar essas tabelas de modelo em seu banco de dados:

py manage.py migrate

O comando **[migrate](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-migrate)** pega todas as migrações que não foram aplicadas (o Django rastreia quais são aplicadas usando uma tabela especial em seu banco de dados chamada **django\_migrations**) e as executa em seu banco de dados - essencialmente, sincronizando as alterações feitas em seus modelos com o esquema no base de dados.

As migrações são muito poderosas e permitem que você altere seus modelos ao longo do tempo, à medida que desenvolve seu projeto, sem a necessidade de excluir seu banco de dados ou tabelas e criar novos - é especializado em atualizar seu banco de dados ao vivo, sem perder dados. Vamos abordá-los com mais profundidade em uma parte posterior do tutorial, mas, por enquanto, lembre-se do guia de três etapas para fazer alterações no modelo:

* Mude seus modelos (em **models.py**).
* Execute para criar migrações para essas alterações:

[**py manage.py makemigrations**](https://docs.djangoproject.com/en/4.1/ref/django-admin/#django-admin-makemigrations)

* Execute para aplicar essas alterações ao banco de dados:

[**py manage.py migrate**](https://docs.djangoproject.com/en/4.1/ref/django-admin/#django-admin-migrate)

A razão pela qual existem comandos separados para fazer e aplicar migrações é porque você confirmará migrações para seu sistema de controle de versão e as enviará com seu aplicativo; eles não apenas facilitam seu desenvolvimento, mas também podem ser usados ​​por outros desenvolvedores e na produção.

Altere o polls/models.py acrescentando o comando abaixo da classe Question as linhas marcadas em destaque

|  |
| --- |
| **import** **datetime**  **from** **django.db** **import** models  from django.utils.datetime\_safe import datetime  **class** **Question**(models.Model):  *# ...*  **def** was\_published\_recently(self):  **return** self.pub\_date >= timezone.now() - datetime.timedelta(days=1) |

Observe a adição de e , para referenciar o módulo padrão do Python e os utilitários relacionados ao fuso horário do Django em , respectivamente.

## Apresentando o Django Admin [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#introducing-the-django-admin)

**Filosofia**

Gerar sites de administração para sua equipe ou clientes adicionar, alterar e excluir conteúdo é um trabalho tedioso que não requer muita criatividade. Por esse motivo, o Django automatiza totalmente a criação de interfaces administrativas para modelos.

Django foi escrito em um ambiente de redação, com uma separação muito clara entre “editores de conteúdo” e o site “público”. Os gerentes do site usam o sistema para adicionar notícias, eventos, placares esportivos etc., e esse conteúdo é exibido no site público. O Django resolve o problema de criar uma interface unificada para os administradores do site editarem o conteúdo.

O admin não se destina a ser usado pelos visitantes do site. É para gerentes de site.

## Criando um usuário administrador [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#creating-an-admin-user)

Primeiro, precisamos criar um usuário que possa fazer login no site de administração. Execute o seguinte comando:

py manage.py createsuperuser

Digite o nome de usuário desejado e pressione enter.

Username: admin

Em seguida, você será solicitado a fornecer o endereço de e-mail desejado:

Email address: admin@example.com

A etapa final é inserir sua senha. Você será solicitado a digitar sua senha duas vezes, a segunda vez como confirmação da primeira.

Password: \*\*\*\*\*\*\*\*\*\*

Password (again): \*\*\*\*\*\*\*\*\*

Superuser created successfully.

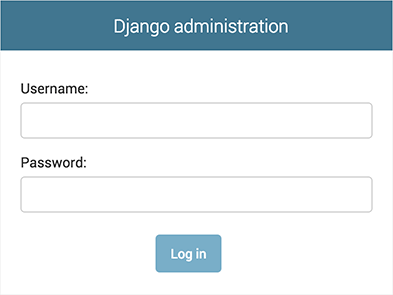
### Inicie o servidor de desenvolvimento [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#start-the-development-server)

O site de administração do Django é ativado por padrão. Vamos iniciar o servidor de desenvolvimento e explorá-lo.

Se o servidor não estiver em execução, inicie-o assim:

py manage.py runserver

Agora, abra um navegador da Web e vá para “/admin/” em seu domínio local – por exemplo, <http://127.0.0.1:8000/admin/> . Você deve ver a tela de login do administrador:



Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Você deve ver alguns tipos de conteúdo editável: grupos e usuários. Eles são fornecidos por **[django.contrib.auth](https://docs.djangoproject.com/en/4.1/topics/auth/" \l "module-django.contrib.auth" \o "django.contrib.auth: Estrutura de autenticação do Django.)**, o framework de autenticação fornecido pelo Django.

### Torne o aplicativo de enquete modificável no admin [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#make-the-poll-app-modifiable-in-the-admin)

Mas onde está nosso aplicativo de votação? Não é exibido na página de índice do administrador.

Só mais uma coisa a fazer: precisamos informar ao administrador que os objetos **Question** possuem uma interface administrativa. Para fazer isso, abra o arquivo **polls/admin.py** e edite-o para ficar assim:

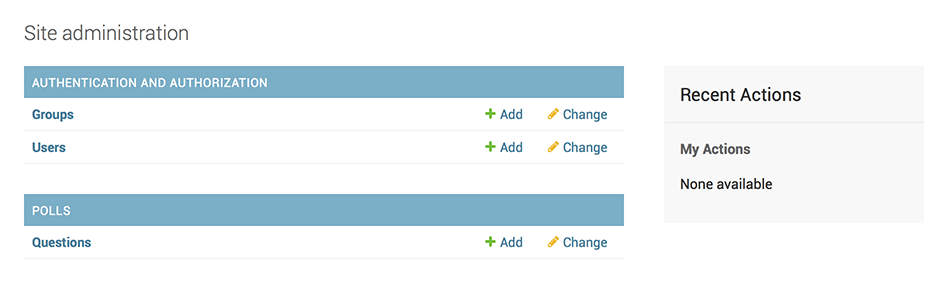
|  |
| --- |
| **from** **django.contrib** **import** admin  **from** **.models** **import** Question  admin.site.register(Question) |

Interface gráfica do usuário, Aplicativo, Site

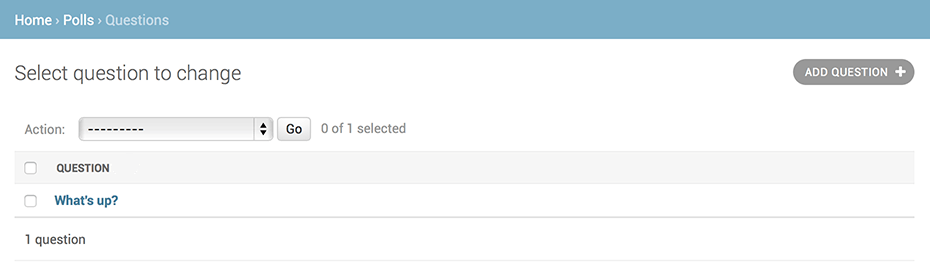
Descrição gerada automaticamente

### Explore a funcionalidade administrativa gratuita [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial02/#explore-the-free-admin-functionality)

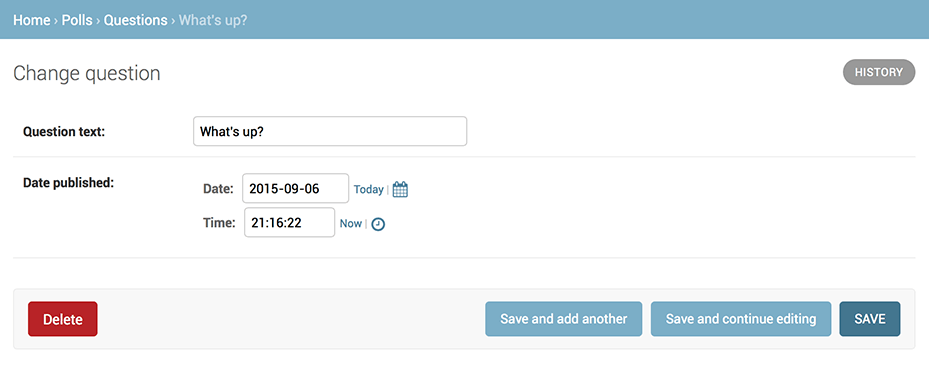
Agora que registramos **Question**, o Django sabe que deve ser exibido na página de índice do administrador:



Clique em “Perguntas”. Agora você está na página "alterar lista" para perguntas. Esta página exibe todas as perguntas no banco de dados e permite que você escolha uma para alterá-la. Há o "E aí?" pergunta que criamos anteriormente:



Clique no botão "E aí?" pergunta para editá-lo:

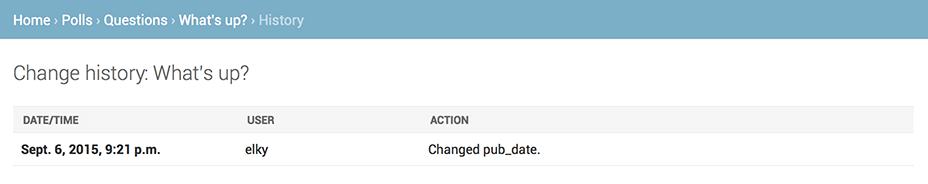


Coisas a serem observadas aqui:

* O formulário é gerado automaticamente a partir do **Question**modelo.
* Os diferentes tipos de campo de modelo ( **[DateTimeField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.DateTimeField" \o "django.db.models.DateTimeField)**, **[CharField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.CharField" \o "django.db.models.CharField)**) correspondem ao widget de entrada HTML apropriado. Cada tipo de campo sabe como se exibir no admin do Django.
* Cada um **[DateTimeField](https://docs.djangoproject.com/en/4.1/ref/models/fields/" \l "django.db.models.DateTimeField" \o "django.db.models.DateTimeField)**recebe atalhos JavaScript gratuitos. As datas recebem um atalho “Hoje” e um pop-up de calendário, e os horários recebem um atalho “Agora” e um pop-up conveniente que lista os horários comumente inseridos.

A parte inferior da página oferece algumas opções:

* Salvar – salva as alterações e retorna à página da lista de alterações para esse tipo de objeto.
* Salvar e continuar editando – Salva as alterações e recarrega a página de administração para este objeto.
* Salvar e adicionar outro – Salva as alterações e carrega um novo formulário em branco para esse tipo de objeto.
* Excluir – Exibe uma página de confirmação de exclusão.



Note que ao criar questões o resultado da listagem é Question ObjectInterface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Para alterar modifie a model para

def \_\_str\_\_(self):  
 return self.question\_text

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Visualizações – Views

Visão geral [**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#overview)

Uma view é um “tipo” de página da web em seu aplicativo Django que geralmente atende a uma função específica e possui um modelo específico. Por exemplo, em um aplicativo de blog, você pode ter as seguintes exibições:

* Página inicial do blog – exibe as entradas mais recentes.
* Página de “detalhes” da entrada – página de link permanente para uma única entrada.
* Página de arquivo baseada no ano – exibe todos os meses com entradas no ano determinado.
* Página de arquivo com base no mês – exibe todos os dias com entradas no mês determinado.
* Página de arquivo com base no dia – exibe todas as entradas no dia determinado.
* Ação de comentário – lida com a postagem de comentários em uma determinada entrada.

Em nosso aplicativo de enquete, teremos as quatro visualizações a seguir:

* Página de “índice” da pergunta – exibe as perguntas mais recentes.
* Página de “detalhes” da pergunta – exibe o texto da pergunta, sem resultados, mas com um formulário para votar.
* Página de “resultados” da pergunta – exibe os resultados de uma pergunta específica.
* Ação de votação – lida com a votação para uma escolha específica em uma questão específica.

No Django, as páginas da web e outros conteúdos são entregues por visualizações. Cada visão é representada por uma função Python (ou método, no caso de visões baseadas em classes). O Django escolherá uma visualização examinando a URL solicitada (para ser preciso, a parte da URL após o nome do domínio).

Agora, em seu tempo na web, você pode ter encontrado belezas como **ME2/Sites/dirmod.htm?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B**. Você ficará satisfeito em saber que o Django nos permite *padrões de URL* muito mais elegantes do que isso.

Um padrão de URL é a forma geral de uma URL - por exemplo: **/newsarchive/<year>/<month>/**.

Para ir de uma URL para uma view, o Django usa o que é conhecido como 'URLconfs'. Um URLconf mapeia padrões de URL para visualizações.

## Escrevendo mais visualizações [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#writing-more-views)

Agora vamos adicionar mais algumas visualizações a **polls/views.py**. Essas visões são um pouco diferentes, porque aceitam um argumento:

|  |
| --- |
| **def** detail(request, question\_id):  **return** HttpResponse("You're looking at question **%s**." % question\_id)  **def** results(request, question\_id):  response = "You're looking at the results of question **%s**."  **return** HttpResponse(response % question\_id)  **def** vote(request, question\_id):  **return** HttpResponse("You're voting on question **%s**." % question\_id) |

Conecte essas novas visualizações ao **polls.urls**módulo adicionando as seguintes [**path()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.path)chamadas:

|  |
| --- |
| **from** **django.urls** **import** path  **from** **.** **import** views  urlpatterns = [  *# ex: /polls/*  path('', views.index, name='index'),  *# ex: /polls/5/*  path('<int:question\_id>/', views.detail, name='detail'),  *# ex: /polls/5/results/*  path('<int:question\_id>/results/', views.results, name='results'),  *# ex: /polls/5/vote/*  path('<int:question\_id>/vote/', views.vote, name='vote'),  ] |

Dê uma olhada no seu navegador, em “/polls/34/”. Ele executará o **detail()** método e exibirá qualquer ID que você fornecer na URL.

Tente

/polls/34/results/

/polls/34/vote/

eles exibirão os resultados do espaço reservado e as páginas de votação.

Quando alguém solicita uma página do seu site – digamos, “/polls/34/”, o Django carregará o módulo Python  **mysite.urls** porque ele é apontado pela configuração [**ROOT\_URLCONF**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-ROOT_URLCONF). Ele encontra a variável nomeada **urlpatterns** e percorre os padrões em ordem. Depois de encontrar a correspondência em **'polls/'**, ele remove o texto correspondente ( **"polls/"**) e envia o texto restante – **"34/"**– para o URLconf 'polls.urls' para processamento adicional. Lá ele corresponde a **'<int:question\_id>/'**, resultando em uma chamada para a **detail()**exibição da seguinte forma:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## Escreva visualizações que realmente fazem algo [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#write-views-that-actually-do-something)

Cada visualização é responsável por fazer uma de duas coisas: retornar um **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)**objeto contendo o conteúdo da página solicitada ou gerar uma exceção como [**Http404**](https://docs.djangoproject.com/en/4.1/topics/http/views/#django.http.Http404). O resto é com você.

Sua visão pode ler registros de um banco de dados ou não. Ele pode usar um sistema de templates como o do Django – ou um sistema de templates Python de terceiros – ou não. Ele pode gerar um arquivo PDF, gerar XML, criar um arquivo ZIP em tempo real, qualquer coisa que você quiser, usando as bibliotecas Python que desejar.

Tudo o que Django quer é isso **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)**. Ou uma exceção.

Aqui está uma nova **index()** visão, que exibe as últimas 5 perguntas da enquete no sistema, separadas por vírgulas, de acordo com a data de publicação:

|  |
| --- |
| **from** **django.http** **import** HttpResponse  **from** **.models** **import** Question  **def** index(request):  latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]  output = ', '.join([q.question\_text **for** q **in** latest\_question\_list])  **return** HttpResponse(output)  ... não altere as funções abaixo ok |

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Porém, há um problema aqui: o design da página é codificado na exibição. Se você quiser mudar a aparência da página, terá que editar este código Python. Então, vamos usar o sistema de templates do Django para separar o design do Python criando um template que a view possa usar.

Primeiro, crie um diretório chamado **templates** em seu diretório **polls**. O Django irá procurar por templates lá.

A configuração do seu projeto descreve [**TEMPLATES**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-TEMPLATES) como o Django carregará e renderizará os modelos. O arquivo de configurações padrão configura um **DjangoTemplates** back-end cuja opção [**APP\_DIRS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-TEMPLATES-APP_DIRS) é definida como **True**. Por convenção,  **DjangoTemplates** procura um subdiretório “templates” em cada um dos arquivos [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-INSTALLED_APPS).

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente Dentro do diretório **templates** que você acabou de criar, crie outro diretório chamado **polls**, e dentro dele crie um arquivo chamado **index.html**. Em outras palavras, seu modelo deve estar em **polls/templates/polls/index.html**. Por causa de como o **app\_directories** carregador de modelo funciona conforme descrito acima, você pode se referir a esse modelo no Django como **polls/index.html**.

|  |
| --- |
| **Namespace do modelo**  Agora podemos conseguir colocar nossos modelos diretamente **polls/templates**(em vez de criar outro subdiretório  **polls**), mas na verdade seria uma má ideia. O Django escolherá o primeiro modelo que encontrar cujo nome corresponda e, se você tiver um modelo com o mesmo nome em um aplicativo diferente , o Django não conseguirá distinguir entre eles. Precisamos ser capazes de apontar o Django para o caminho certo, e a melhor maneira de garantir isso é dando namespace a eles. Ou seja, colocando esses templates dentro de outro diretório com o nome do próprio aplicativo. |

Coloque o seguinte código nesse modelo: dentro da página html entre as tags <body> e </body>

|  |
| --- |
| {% **if** latest\_question\_list %}  <**ul**>  {% **for** question **in** latest\_question\_list %}  <**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**>  {% **endfor** %}  </**ul**>  {% **else** %}  <**p**>Sem questões.</**p**>  {% **endif** %} |

Agora vamos atualizar nossa view **index em polls/views.py** para usar o template:

|  |
| --- |
| **from** **django.http** **import** HttpResponse  **from** **django.template** **import** loader  **from** **.models** **import** Question  **def** index(request):  latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]  template = loader.get\_template('polls/index.html')  context = {  'latest\_question\_list': latest\_question\_list,  }  **return** HttpResponse(template.render(context, request))  ... não altere o código abaixo ok |

Esse código carrega o modelo chamado **polls/index.html**e passa um contexto para ele. O contexto é um dicionário de nomes de variáveis ​​de modelo de mapeamento para objetos Python.

Carregue a página apontando seu navegador para “/polls/”

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

### Um atalho:  [render()](https://docs.djangoproject.com/en/4.1/topics/http/shortcuts/#django.shortcuts.render)

É uma linguagem muito comum carregar um template, preencher um contexto e retornar um objeto [**HttpResponse**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpResponse) com o resultado do template renderizado. Django fornece um atalho. Aqui está a **index()**visão completa, reescrita:

|  |
| --- |
| **from** **django.shortcuts** **import** render  **from** **.models** **import** Question  **def** index(request):  latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]  context = {'latest\_question\_list': latest\_question\_list}  **return** render(request, 'polls/index.html', context)  ... não altere o código para baixo ok |

Observe que, depois de fazer isso em todas essas exibições, não precisamos mais importar **[loader](https://docs.djangoproject.com/en/4.1/topics/templates/" \l "module-django.template.loader" \o "django.template.loader)** e **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)**(você desejará manter **HttpResponse** se ainda tiver os métodos de stub para **detail**, **results** e **vote**).

A função [**render()**](https://docs.djangoproject.com/en/4.1/topics/http/shortcuts/#django.shortcuts.render)usa o objeto de solicitação como primeiro argumento, um nome de modelo como segundo argumento e um dicionário como terceiro argumento opcional. Ele retorna um **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** objeto do modelo fornecido renderizado com o contexto fornecido.

## Gerando um erro 404 [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#raising-a-404-error)

Agora, vamos abordar a visualização de detalhes da pergunta – a página que exibe o texto da pergunta para uma determinada enquete. Aqui está a visão:

|  |
| --- |
| **from** **django.http** **import** Http404  **from** **django.shortcuts** **import** render  **from** **.models** **import** Question  *# ...*  **def** detail(request, question\_id):  **try**:  question = Question.objects.get(pk=question\_id)  **except** Question.DoesNotExist:  **raise** Http404("Question does not exist")  **return** render(request, 'polls/detail.html', {'question': question}) |

O novo conceito aqui: A exibição levanta a [**Http404**](https://docs.djangoproject.com/en/4.1/topics/http/views/#django.http.Http404)exceção se uma pergunta com o ID solicitado não existir.

Discutiremos o que você poderia colocar nesse modelo **polls/detail.html** um pouco mais tarde, mas se você quiser fazer o exemplo acima funcionar rapidamente, um arquivo contendo apenas:

|  |
| --- |
| {{ question }} |

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Vai começar por agora

### Um atalho: [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#a-shortcut-get-object-or-404)[get\_object\_or\_404()](https://docs.djangoproject.com/en/4.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)

É um idioma muito comum usar **[get()](https://docs.djangoproject.com/en/4.1/ref/models/querysets/" \l "django.db.models.query.QuerySet.get" \o "django.db.models.query.QuerySet.get)** e aumentar [**Http404**](https://docs.djangoproject.com/en/4.1/topics/http/views/#django.http.Http404)se o objeto não existir. Django fornece um atalho. Aqui está a **detail()**visão, reescrita:

|  |
| --- |
| **def** detail(request, question\_id):  question = get\_object\_or\_404(Question, pk=question\_id)  **return** render(request, 'polls/detail.html', {'question': question}) |

A função [**get\_object\_or\_404()**](https://docs.djangoproject.com/en/4.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)recebe um modelo Django como seu primeiro argumento e um número arbitrário de argumentos de palavras-chave, que passa para a função **[get()](https://docs.djangoproject.com/en/4.1/ref/models/querysets/" \l "django.db.models.query.QuerySet.get" \o "django.db.models.query.QuerySet.get)**do gerenciador do modelo. Aciona [**Http404**](https://docs.djangoproject.com/en/4.1/topics/http/views/#django.http.Http404) se o objeto não existe.

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

## Use o sistema de modelo [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#use-the-template-system)

Voltar para a **detail()**exibição de nosso aplicativo de votação. Dada a variável de contexto **question**, veja como o **polls/detail.html** modelo pode parecer:

|  |
| --- |
| <**h1**>{{ question.question\_text }}</**h1**>  <**ul**>  {% **for** choice **in** question.choice\_set.all %}  <**li**>{{ choice.choice\_text }}</**li**>  {% **endfor** %}  </**ul**> |

O sistema de modelo usa a sintaxe de pesquisa de ponto para acessar atributos de variáveis. No exemplo de , primeiro o Django faz uma pesquisa no dicionário do objeto . Caso contrário, ele tenta uma pesquisa de atributo – o que funciona, neste caso. Se a pesquisa de atributo tivesse falhado, ele tentaria uma pesquisa de índice de lista.**{{ question.question\_text }}question**

A chamada de método acontece no loop: é interpretada como o código Python , que retorna um iterável de objetos e é adequado para uso na tag.[**{% for %}**](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#std-templatetag-for)**question.choice\_set.allquestion.choice\_set.all()Choice**[**{% for %}**](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#std-templatetag-for)

## Removendo URLs codificadas em modelos [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#removing-hardcoded-urls-in-templates)

Lembre-se, quando escrevemos o link para uma pergunta no **polls/index.html** modelo, o link foi parcialmente codificado assim:

|  |
| --- |
| <**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**> |

O problema com essa abordagem codificada e fortemente acoplada é que se torna um desafio alterar URLs em projetos com muitos modelos. No entanto, como você definiu o argumento name nas funções do módulo [**path()**](https://docs.djangoproject.com/en/4.1/ref/urls/#django.urls.path) **polls.urls**, pode remover a dependência de caminhos de URL específicos definidos em suas configurações de URL usando a tag de modelo:**{% url %}**

|  |
| --- |
| <**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**> |

A maneira como isso funciona é procurando a definição de URL conforme especificado no módulo **polls.urls**. Você pode ver exatamente onde o nome da URL de 'detail' está definido abaixo:

|  |
| --- |
| ...  *# the 'name' value as called by the {% url %} template tag*  path('<int:question\_id>/', views.detail, name='detail'),  ... |

Se você quiser alterar a URL da exibição de detalhes das enquetes para outra coisa, talvez para algo como **polls/specifics/12/**, em vez de fazê-lo no modelo (ou modelos), altere-o em **polls/urls.py**:

|  |
| --- |
| ...  *# added the word 'specifics'*  path('specifics/<int:question\_id>/', views.detail, name='detail'),  ... |

## Nomes de URL de namespace [¶](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#namespacing-url-names)

O projeto tutorial tem apenas um aplicativo, **polls**. Em projetos Django reais, pode haver cinco, dez, vinte aplicativos ou mais. Como o Django diferencia os nomes de URL entre eles? Por exemplo, o **polls**aplicativo tem uma **detail** exibição e, portanto, um aplicativo no mesmo projeto que é para um blog. Como fazer para que o Django saiba qual visualização de aplicativo criar para uma url ao usar a tag de modelo?**{% url %}**

A resposta é adicionar namespaces ao seu URLconf. No **polls/urls.py** arquivo, vá em frente e adicione um **app\_name** para definir o namespace do aplicativo:

|  |
| --- |
| polls/urls.py[**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial03/#id12)  **from** **django.urls** **import** path  **from** **.** **import** views  app\_name = 'polls'  urlpatterns = [  path('', views.index, name='index'),  path('<int:question\_id>/', views.detail, name='detail'),  path('<int:question\_id>/results/', views.results, name='results'),  path('<int:question\_id>/vote/', views.vote, name='vote'),  ] |

Agora altere seu modelo **polls/index.html de**:

|  |
| --- |
| <**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**> |

para apontar para a visualização de detalhes com namespace:

|  |
| --- |
| <**li**><**a** href="{% **url** 'polls:detail' question.id %}">{{ question.question\_text }}</**a**></**li**> |

Minimal Form

Vamos atualizar nosso modelo de detalhes da enquete (“polls/detail.html”) para que o modelo contenha um **<form>**elemento HTML:

|  |
| --- |
| <**form** action="{% **url** 'polls:vote' question.id %}" method="post">  {% **csrf\_token** %}  <**fieldset**>  <**legend**><**h1**>{{ question.question\_text }}</**h1**></**legend**>  {% **if** error\_message %}<**p**><**strong**>{{ error\_message }}</**strong**></**p**>{% **endif** %}  {% **for** choice **in** question.choice\_set.all %}  <**input** type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">  <**label** for="choice{{ forloop.counter }}">{{ choice.choice\_text }}</**label**><**br**>  {% **endfor** %}  </**fieldset**>  <**input** type="submit" value="Vote">  </**form**> |

Um rápido resumo:

* O modelo acima exibe um botão de opção para cada opção de pergunta. O **value**de cada botão de opção é o ID da opção de pergunta associada. O **name**de cada botão de opção é **"choice"**. Isso significa que, quando alguém selecionar um dos botões de opção e enviar o formulário, ele enviará os dados do POST **choice=#**onde # é o ID da opção selecionada. Este é o conceito básico de formulários HTML.
* Definimos o formulário **action**como , e definimos . Usar (ao contrário de ) é muito importante, porque o ato de enviar este formulário alterará os dados do lado do servidor. Sempre que você criar um formulário que altere os dados do lado do servidor, use . Esta dica não é específica para Django; é uma boa prática de desenvolvimento web em geral.**{% url 'polls:vote' question.id %}method="post"method="post"method="get"method="post"**
* **forloop.counter**indica quantas vezes a **[for](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/" \l "std-templatetag-for)**tag passou por seu loop
* Como estamos criando um formulário POST (que pode ter o efeito de modificar os dados), precisamos nos preocupar com as falsificações de solicitação entre sites. Felizmente, você não precisa se preocupar muito, porque o Django vem com um sistema útil para proteção contra isso. Resumindo, todos os formulários POST direcionados a URLs internas devem usar a tag de modelo.[**{% csrf\_token %}**](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#std-templatetag-csrf_token)

Agora, vamos criar uma visualização Django que manipule os dados enviados e faça algo com eles

criamos um URLconf para a aplicação de enquetes que inclui esta linha:

|  |
| --- |
| path('<int:question\_id>/vote/', views.vote, name='vote'), |

Também criamos uma implementação fictícia da função **vote()**. Vamos criar uma versão real. Adicione o seguinte a **polls/views.py**:

Colei todo o arquivo aqui ok

|  |
| --- |
| from django.http import HttpResponse, HttpResponseRedirect from django.shortcuts import get\_object\_or\_404, render, loader from django.urls import reverse,  from .models import Choice, Question   def index(request):  latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]  template = loader.get\_template('polls/index.html')  context = {  'latest\_question\_list': latest\_question\_list,  }  return HttpResponse(template.render(context, request))   def detail(request, question\_id):  question = get\_object\_or\_404(Question, pk=question\_id)  return render(request, 'polls/detail.html', {'question': question})   def results(request, question\_id):  response = "Você está olhando o resultado questão %s."  return HttpResponse(response % question\_id)   def vote(request, question\_id):  question = get\_object\_or\_404(Question, pk=question\_id)  try:  selected\_choice = question.choice\_set.get(pk=request.POST['choice'])  except (KeyError, Choice.DoesNotExist):  # Redisplay the question voting form.  return render(request, 'polls/detail.html', {  'question': question,  'error\_message': "You didn't select a choice.",  })  else:  selected\_choice.votes += 1  selected\_choice.save()  # Always return an HttpResponseRedirect after successfully dealing  # with POST data. This prevents data from being posted twice if a  # user hits the Back button.  return HttpResponseRedirect(reverse('polls:results', args=(question.id,))) |

ste código inclui algumas coisas que ainda não abordamos neste tutorial:

* [**request.POST**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.POST)é um objeto semelhante a um dicionário que permite acessar dados enviados por nome de chave. Nesse caso, **request.POST['choice']**retorna o ID da opção selecionada, como uma string. **[request.POST](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "django.http.HttpRequest.POST)**os valores são sempre strings.

Observe que o Django também fornece **[request.GET](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.GET" \o "django.http.HttpRequest.GET)**acesso aos dados GET da mesma maneira - mas estamos usando explicitamente **[request.POST](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "django.http.HttpRequest.POST)**em nosso código, para garantir que os dados sejam alterados apenas por meio de uma chamada POST.

* **request.POST['choice']**aumentará **[KeyError](https://docs.python.org/3/library/exceptions.html" \l "KeyError" \o "(em Python v3.11))**se **choice**não for fornecido nos dados POST. O código acima verifica **[KeyError](https://docs.python.org/3/library/exceptions.html" \l "KeyError" \o "(em Python v3.11))**e exibe novamente o formulário de pergunta com uma mensagem de erro se **choice**não for fornecido.
* Depois de incrementar a contagem de opções, o código retorna um **[HttpResponseRedirect](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)**em vez de um **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)**. **[HttpResponseRedirect](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)**recebe um único argumento: a URL para a qual o usuário será redirecionado (consulte o ponto a seguir para saber como construímos a URL nesse caso).

Como o comentário do Python acima indica, você sempre deve retornar um **[HttpResponseRedirect](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)**depois de lidar com dados POST com sucesso. Esta dica não é específica para Django; é uma boa prática de desenvolvimento web em geral.sjang

* Estamos usando a [**reverse()**](https://docs.djangoproject.com/en/4.1/ref/urlresolvers/#django.urls.reverse)função no **[HttpResponseRedirect](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)**construtor neste exemplo. Esta função ajuda a evitar ter que codificar uma URL na função de exibição. É dado o nome da visualização para a qual queremos passar o controle e a parte variável do padrão de URL que aponta para essa visualização. Neste caso, usando o URLconf que configuramos no [Tutorial 3](https://docs.djangoproject.com/en/4.1/intro/tutorial03/) , esta [**reverse()**](https://docs.djangoproject.com/en/4.1/ref/urlresolvers/#django.urls.reverse)chamada retornará uma string como

|  |
| --- |
| '/polls/3/results/' |

* onde o **3** é o valor de **question.id**. Esse URL redirecionado chamará a **'results'**exibição para exibir a página final.

Conforme mencionado **request** é um objeto [**HttpRequest**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest).

Depois que alguém vota em uma pergunta, a **vote()**visualização é redirecionada para a página de resultados da pergunta. Vamos escrever essa visão:

polls/views.py[**¶**](https://docs.djangoproject.com/en/4.1/intro/tutorial04/#id4)

|  |
| --- |
| **def** results(request, question\_id):  question = get\_object\_or\_404(Question, pk=question\_id)  **return** render(request, 'polls/results.html', {'question': question}) |

so é quase exatamente igual à exibição **detail()** . A única diferença é o nome do modelo. Corrigiremos essa redundância mais tarde.

Agora, crie um modelo **polls/results.html**

|  |
| --- |
| <**h1**>{{ question.question\_text }}</**h1**>  <**ul**>  {% **for** choice **in** question.choice\_set.all %}  <**li**>{{ choice.choice\_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</**li**>  {% **endfor** %}  </**ul**>  <**a** href="{% **url** 'polls:detail' question.id %}">Vote again?</**a**> |

Agora, acesse **/polls/1/**no seu navegador e vote na questão. Você deve ver uma página de resultados que é atualizada cada vez que você vota. Se você enviar o formulário sem ter escolhido uma opção, deverá ver a mensagem de erro.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Acrescente o CHoice para ser administrado

|  |
| --- |
| from django.contrib import admin  from .models import Question from .models import Choice  admin.site.register(Question) admin.site.register(Choice) |

Altere o model para exibir opções amigáveis

|  |
| --- |
| class Choice(models.Model):  question = models.ForeignKey(Question, on\_delete=models.CASCADE)  choice\_text = models.CharField(max\_length=200)  votes = models.IntegerField(default=0)   def \_\_str\_\_(self):  return self.choice\_text |

Insira uma questão e suas opções

**Observação**

O código para nossa **vote()**visão tem um pequeno problema. Ele primeiro obtém o **selected\_choice** objeto do banco de dados, depois calcula o novo valor de **votes**, e então o salva de volta no banco de dados. Se dois usuários do seu site tentarem votar exatamente ao mesmo tempo , isso pode dar errado: o mesmo valor, digamos 42, será recuperado para **votes**. Então, para ambos os usuários, o novo valor de 43 é calculado e salvo, mas 44 seria o valor esperado.