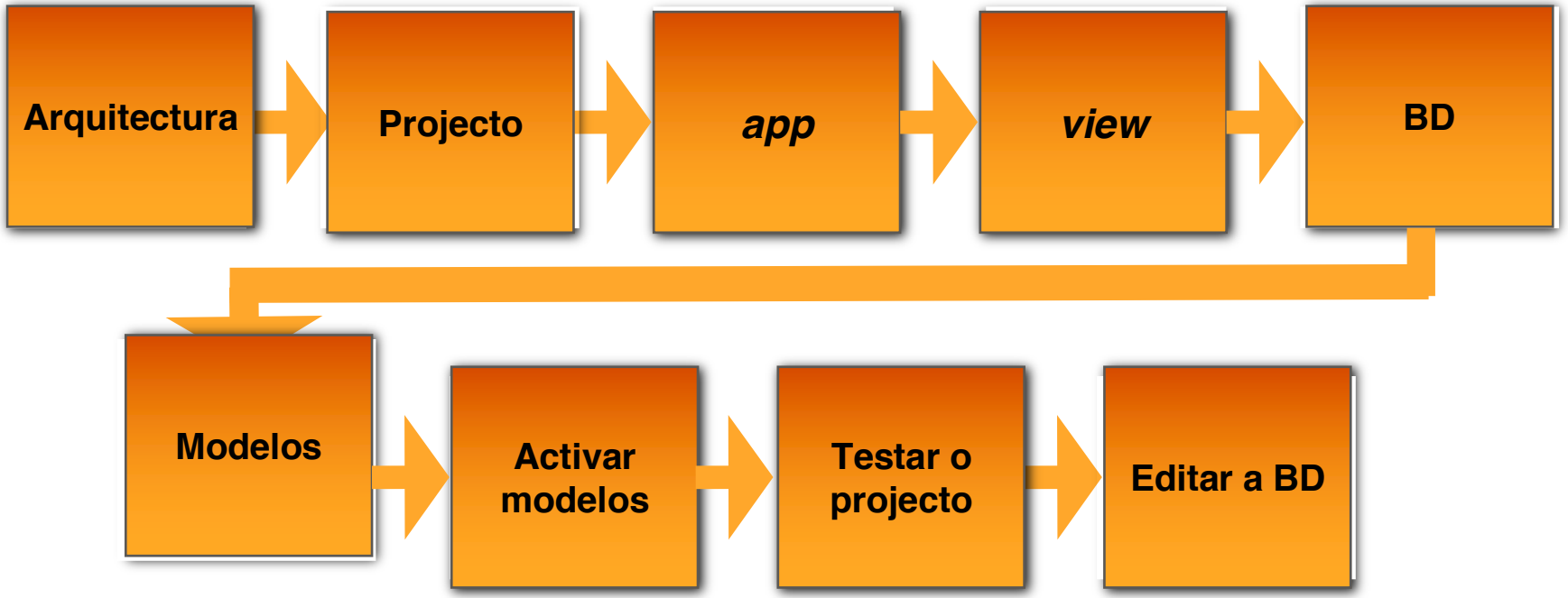
Desenvolvimento para a Internet e Aplicações Móveis

2023/24

Django - parte 1

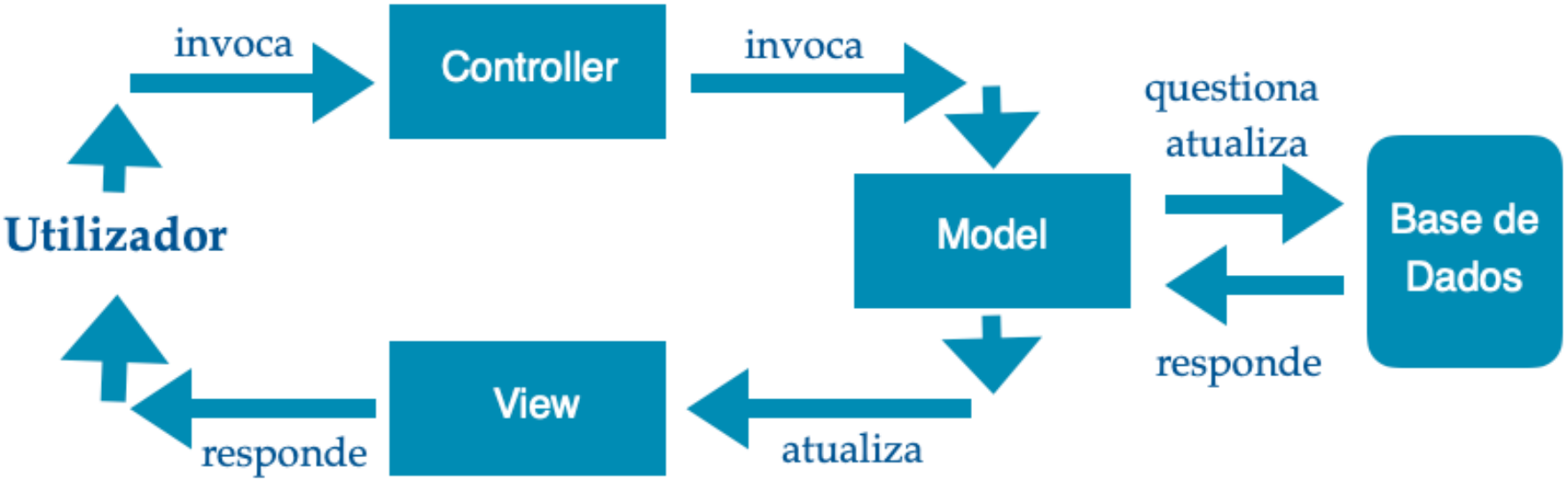
Objectivo:



Arquitectura

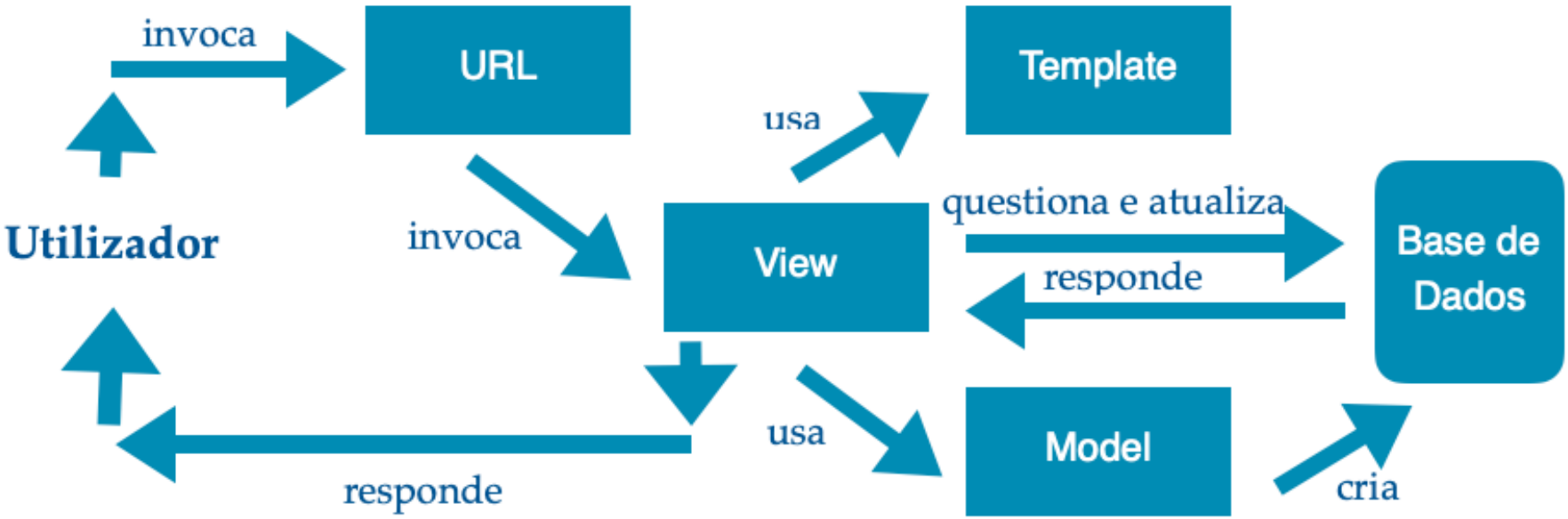
Arquitetura MVC

A arquitetura clássica de tipo MVC compreende o nível "Controller" responsável pelo fluxo de informação, o nível "Model" que atualiza dados e finalmente o nível "View" que apresenta a informação atualizada ao utilizador.



Arquitetura MVT

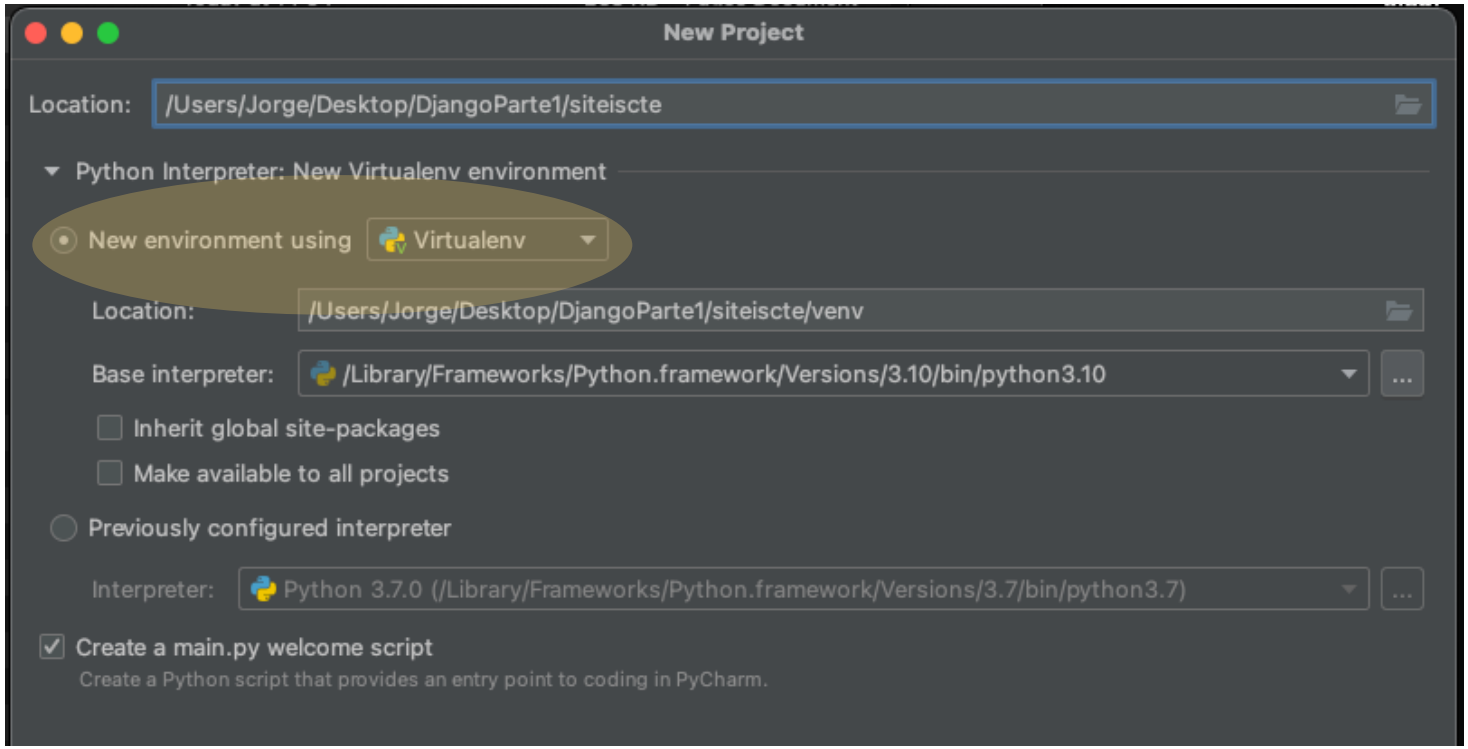
A arquitetura MVT, que significa "Model"-"View"-"Template", sobre a qual é desenhado o Django, é uma evolução da arquitetura MVC. Em Django, "Model" permite criar tabelas e campos numa base de dados relacional. "Views" contém funções que são invocadas por pedidos web, consultam e alteram a base de dados e respondem com HttpResponse. Finalmente, "Template" contém páginas web utilizadas na resposta para o utilizador.



Projecto

1 - Crie um novo projecto a partir do PyCharm.

Crie o projeto chamado siteiscte. Assegure-se de que o projeto é criado com um ambiente virtual.



Depois de criado o projeto, no terminal do PyCharm instale o Django:

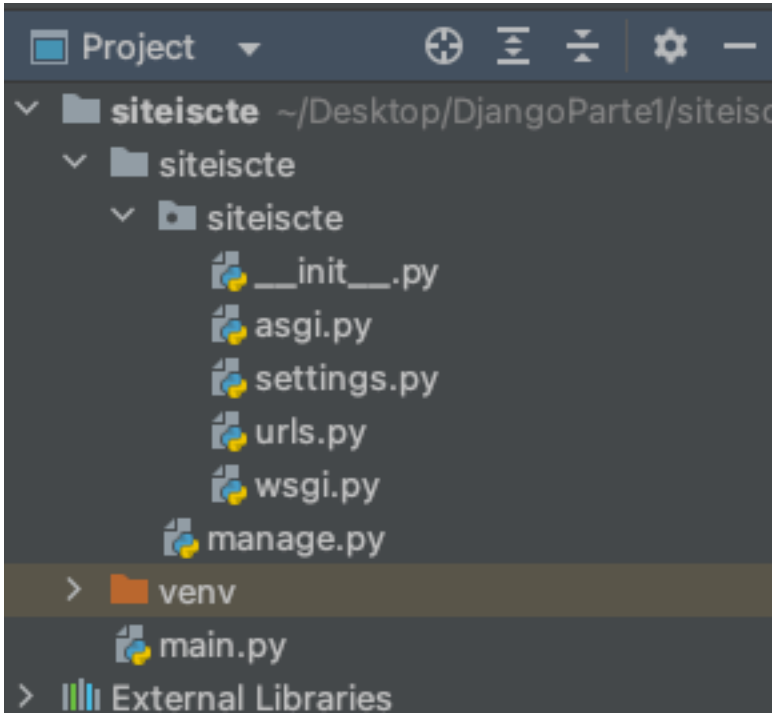
```
pip install django
```

Verifique que o Django existe na diretoria **`lib/python.../site-packages`** do ambiente virtual.

Finalmente crie a estrutura de ficheiros necessária a um projeto Django, através do comando

```
django-admin startproject siteiscte
```

Verifique que a estrutura de ficheiros Django está bem criada.



No terminal, entre na diretoria siteiscte que contém **`manage.py`**

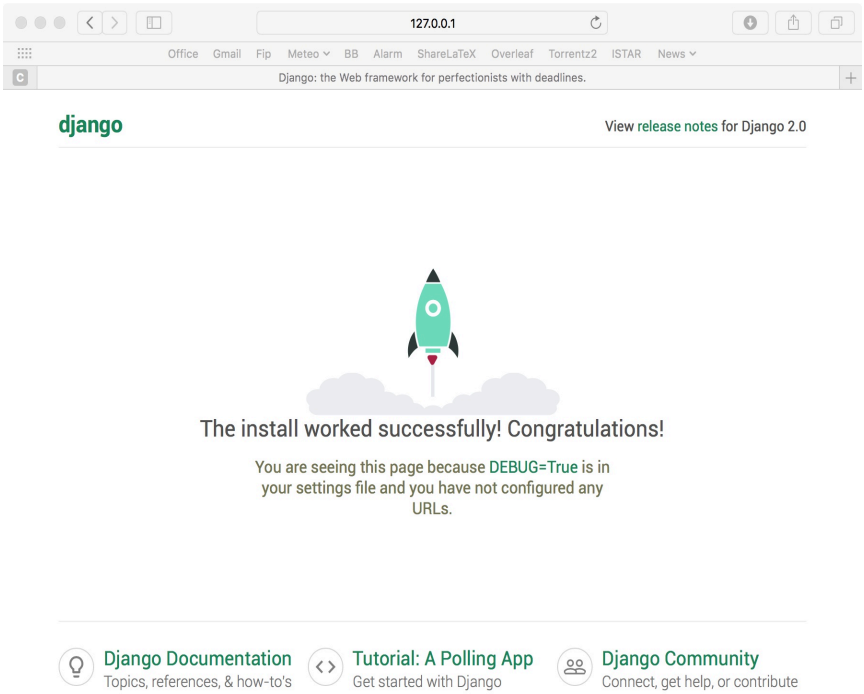
Verifique que o projecto Django funciona. Para isso corra o servidor com o comando seguinte:

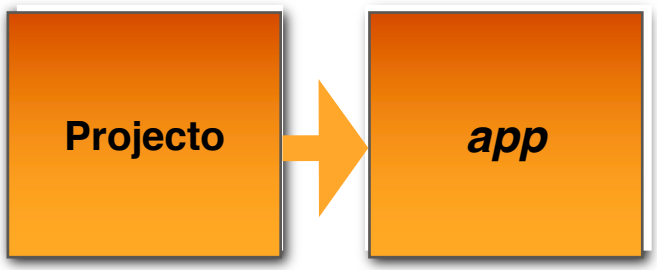
```
python manage.py runserver
```

e depois abra o URL seguinte num browser:

```
http://127.0.0.1:8000/
```

Para sair de Django, na linha de comando: CTRL-C





2 - Criar uma *app*

Diferença entre uma *app* (aplicação) e um projeto

App - aplicação web que executa determinada operação.

Um projeto pode conter várias apps.

Uma app é reutilizável em vários projetos.

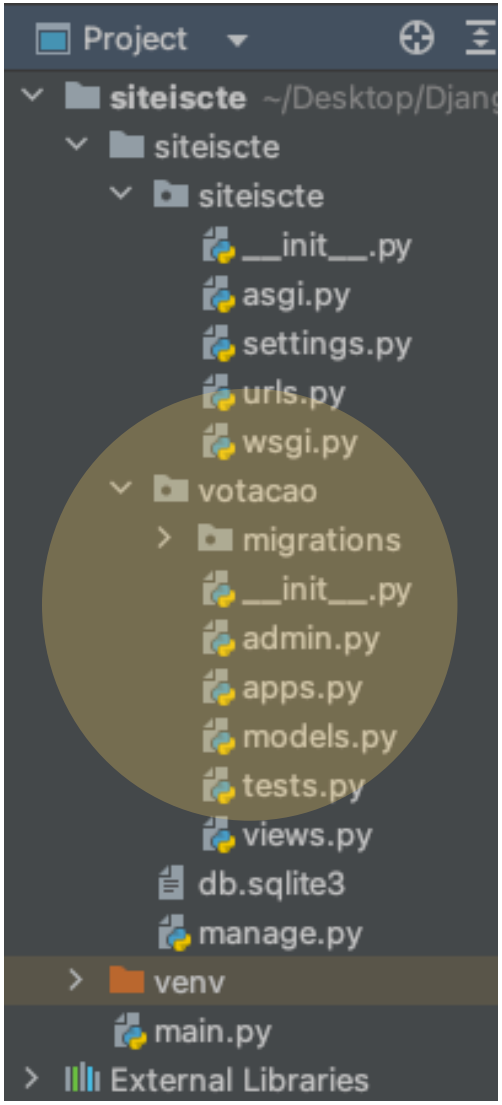
Exemplo: uma *app* que propõe uma questão para ser decidida pelos alunos e que depois recolhe a votação dos alunos.

No terminal, deve estar posicionado na diretoria que contém **`manage.py`**

Execute:

```
python manage.py startapp votacao
```

Verifique que criou a seguinte sub-diretoria para alojar a sua nova *app*:



3 - Criação uma *view*

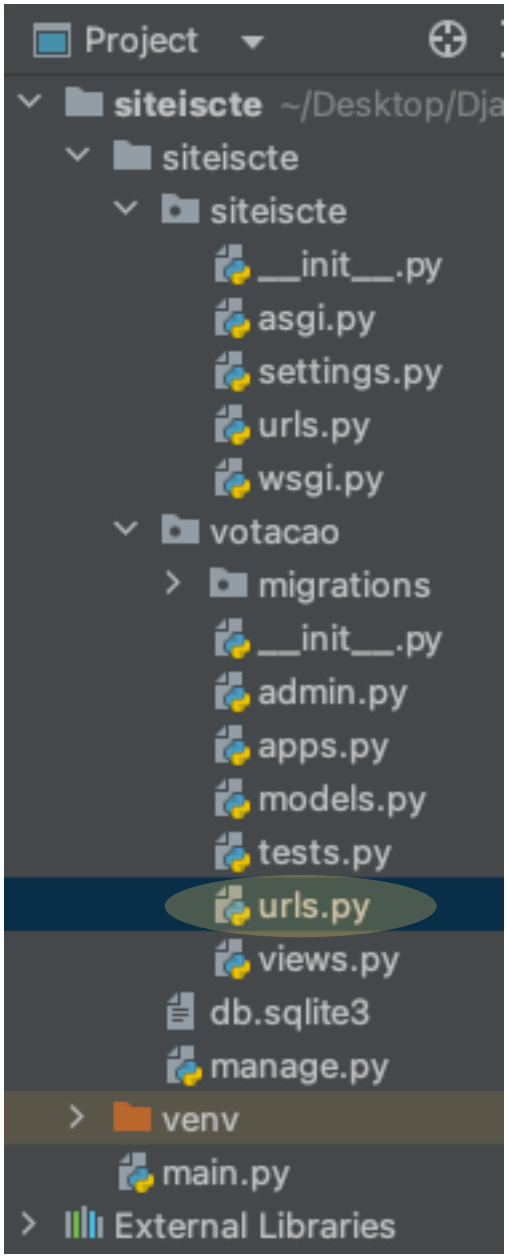
3.1 - Edite o ficheiro **`votacao/views.py`** e adicione o código seguinte:

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("Pagina de entrada da app
    votacao.")
```

3.2 - A nova view deve ser associada a um URL.

Para criar uma URLconf (*URL configuration*): crie um novo ficheiro **`votacao/urls.py`**

A diretoria da app ficou:



Insira o seguinte código em **votacao/urls.py** :

```
from django.urls import include, path
from . import views
# (. significa que importa views da mesma directoria)

urlpatterns = [
    path("", views.index, name="index"),
]
```

Isto é, quando for invocado o url "" o Django procura e executa a função **index** em **views.py**

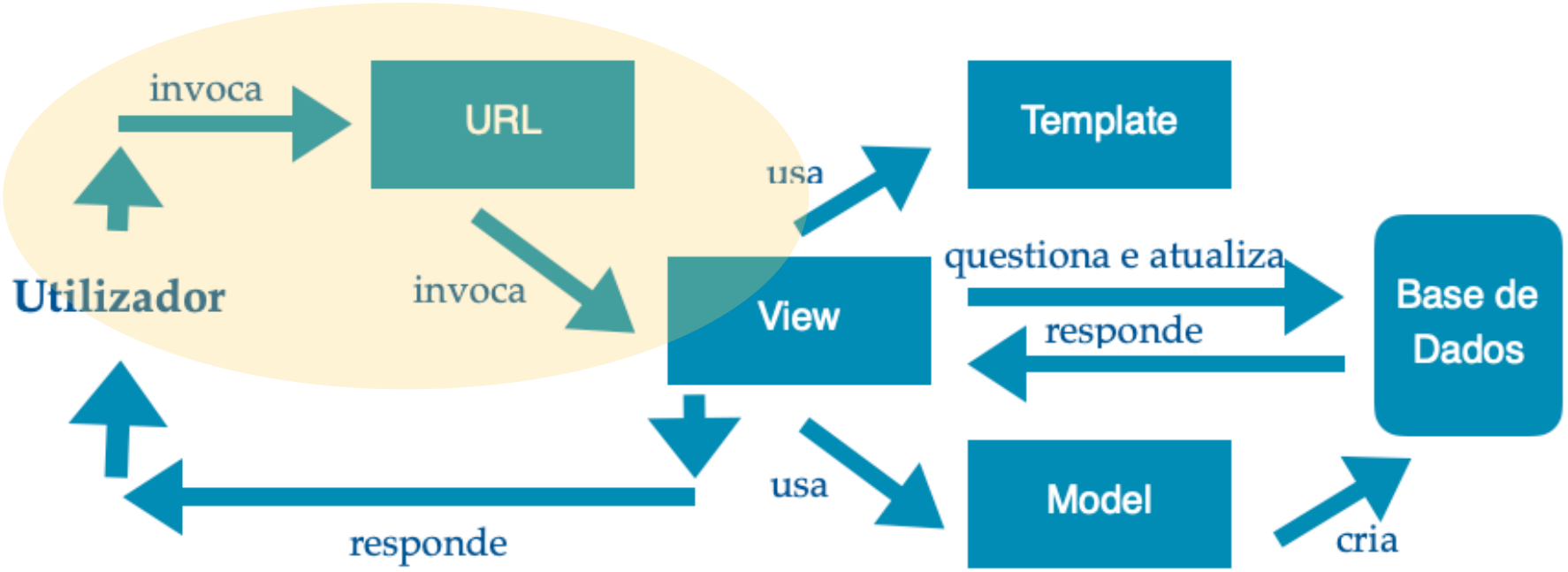
3.3 - Seguidamente associe **votacao.urls** ao projeto. Para isso edite **siteiscte/urls.py** , importe **django.conf.urls.include** e insira **include()** na lista urlpatterns.

siteiscte/urls.py deve ficar com o código seguinte:

```
from django.urls import include, path
from django.contrib import admin

urlpatterns = [
    path('votacao/', include('votacao.urls')),
    path('admin/', admin.site.urls),
]
```

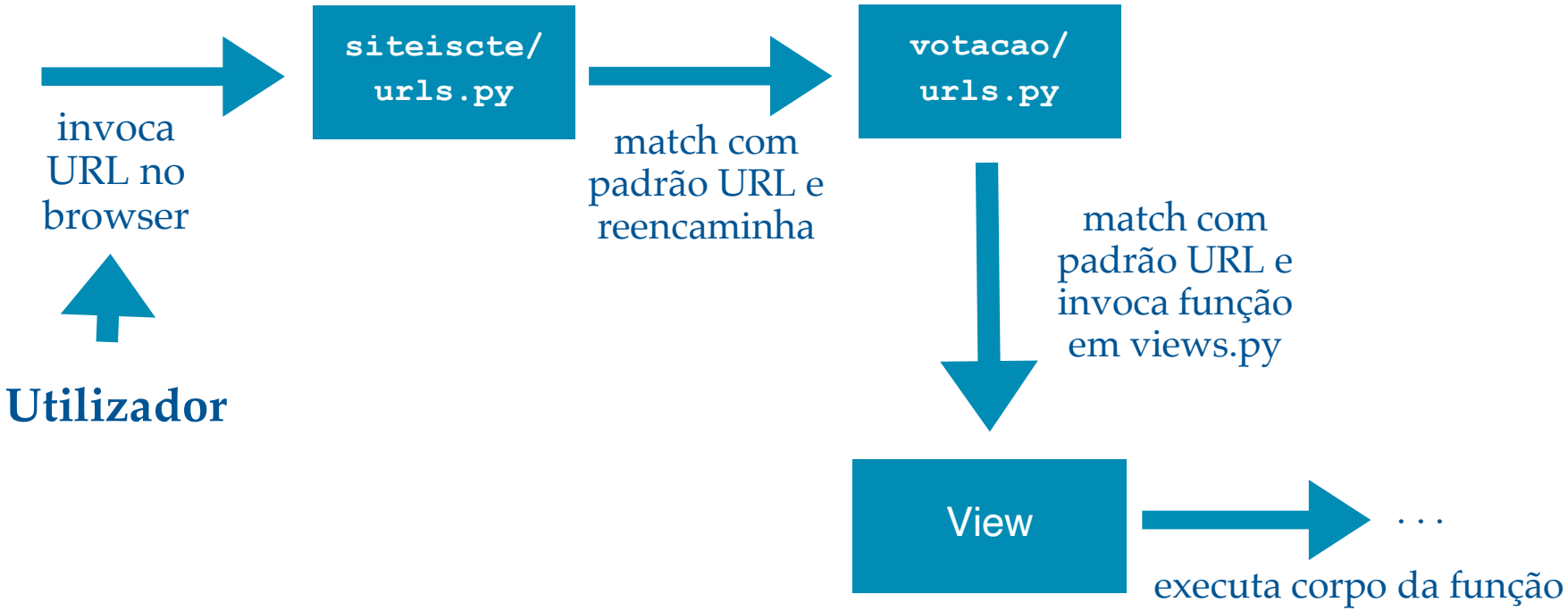
Assim, quando o projeto for invocado num browser, reencaminha de **siteiscte/urls.py** para **votacao/urls.py** e inclui os urls que lá encontrar. A inclusão de URLs é feita através de **include()**.

Para entender este reencaminhamento, voltemos ao esquema geral de invocações do Django:



O utilizador introduz um endereço URL no browser, que inicialmente faz match com um padrão URL existente em **`siteiscte/urls.py`** . Este reencaminha para **`votacao/urls.py`** , onde por sua vez é feito match com um padrão URL aí existente e que finalmente executa uma função existente

em **`votacao/views.py`** . Representado graficamente, o reencaminhamento é o seguinte:



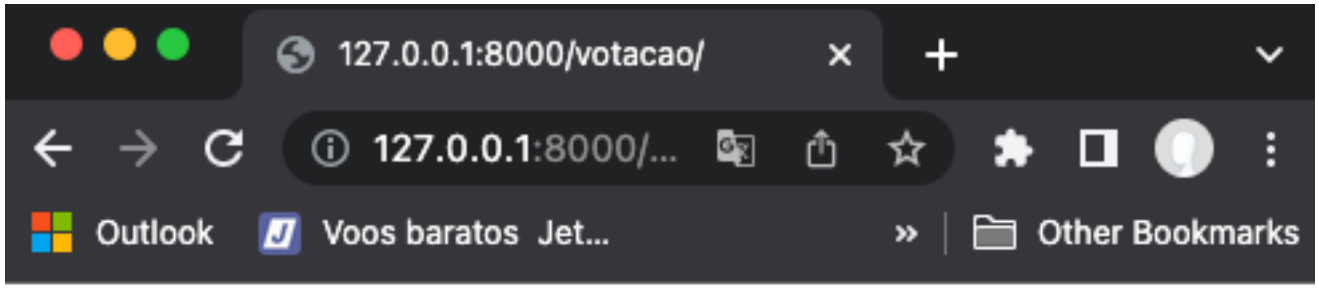
Para verificar o funcionamento correto, execute:

```
python manage.py runserver
```

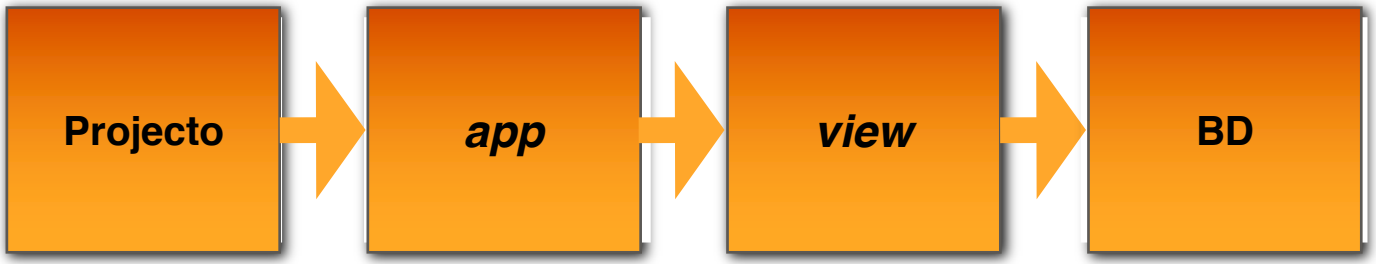
Seguidamente abra o URL

http://localhost:8000/votacao/

Deve obter :



Pagina de entrada da app votacao.



4 - Parametrização da Base de Dados

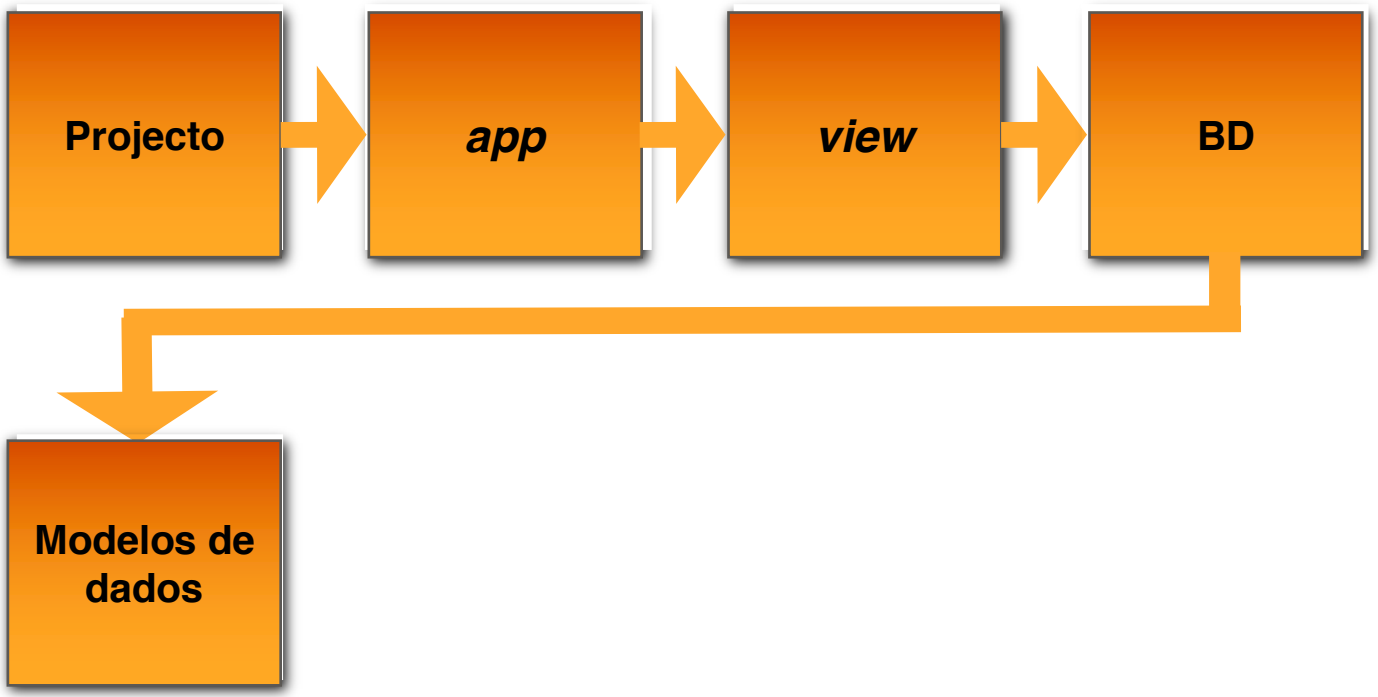
Edite **`siteiscte/settings.py`**.

Django já inclui a base de dados relacional SQLite (http://sqlitebrowser.org), que não precisa de instalar.

Verifique que a base de dados SQLite está devidamente parametrizada em **`siteiscte/settings.py`** através do código seguinte:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Não precisa de alterar o código de parametrização da base de dados, pois Django gera a parametrização correta.



5 - Criação de modelos de dados

Um modelo corresponde a um conceito nos seus dados, representado por uma classe Python.

Lembre-se que a nossa app votação propõe uma questão para ser decidida pelos alunos presentes e depois recolhe a votação dos alunos.

Vamos modelar os conceitos de Questão e de Opção. Por exemplo, a questão "Vamos ao cinema" pode ter duas opções: "sim" ou "não".

Então uma **`Questao`** pode ter várias instâncias de **`Opcao`**. Cada **`Opcao`** tem uma **`Questao`**.

Uma **`Questao`** tem um texto e uma data de publicação.

Uma **`Opcao`** tem um texto e o número de votos que já recolheu. Cada **`Opcao`** é associada a uma **`Questao`**.

Para criar os modelos **`Questao`** e **`Opcao`** edite o ficheiro **`votacao/models.py`** e altere o código para que fique :

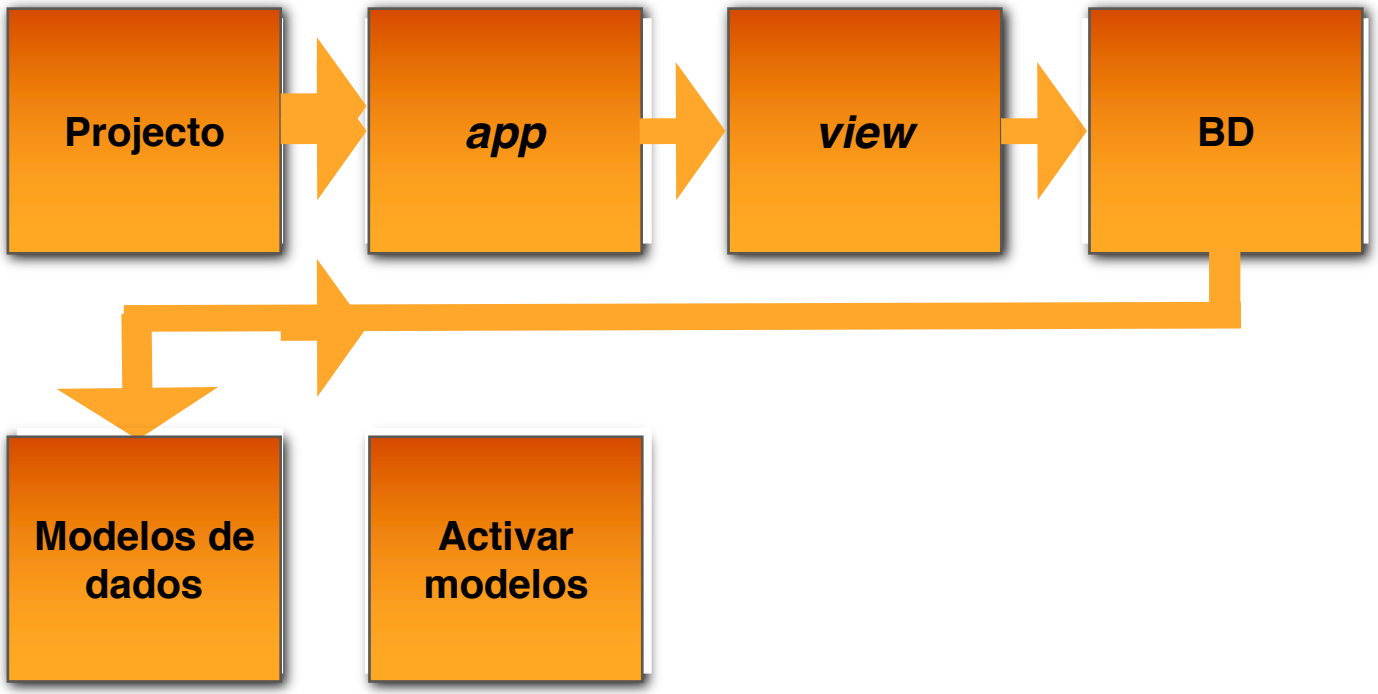
```
from django.db import models
from django.utils import timezone

class Questao(models.Model):
    questao_texto =
            models.CharField(max_length=200)
    pub_data =
            models.DateTimeField('data de
                                  publicacao')

class Opcao(models.Model):
    questao = models.ForeignKey(Questao,
                          on_delete=models.CASCADE)
    opcao_texto = models.CharField(max_length=200)
    votos = models.IntegerField(default=0)
```

De modo geral dizemos que um modelo é representado por uma classe que herda de **`django.db.models.Model`**.

Cada atributo da classe representa um campo numa tabela da base de dados. Cada atributo é representado por uma instância da classe **`Field`**, por exemplo **`CharField`** para textos (exige o argumento max_length) e **`DateTimeField`** para datas. Os nomes dos atributos devem corresponder aos nomes dos campos na base de dados.

Django admite chaves muitos-para-um, muitos-para-muitos, e um-para-um.

No caso deste exemplo, uma chave estrangeira associa cada **`Opcao`** a uma única **`Questao`**. Uma **`Questao`** pode ter muitas **`Opcao`**. Logo, é necessária uma chave estrangeira muitos-para-um.



6 - Activação dos modelos de dados

As apps instaladas no projeto são listadas em **`siteiscte/settings.py`**.

Edite este ficheiro e actualize a lista **`INSTALLED_APPS`** para passar a incluir **`'votacao.apps.VotacaoConfig'`**.

Deve ficar:

```
INSTALLED_APPS = [
    'votacao.apps.VotacaoConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Seguidamente execute o comando que actualiza o projeto de acordo com as mudanças (*migrations*) entretanto realizadas nos modelos:

```
(venv) Jorges-MacBook:sitepr Jorge$ ls
db.sqlite3      manage.py       sitepr          votacao
(venv) Jorges-MacBook:sitepr Jorge$ python manage.py makemigrations votacao
Migrations for 'votacao':
  votacao/migrations/0001_initial.py
    - Create model Questao
    - Create model Opcao
(venv) Jorges-MacBook:sitepr Jorge$
```

Execute também o comando **`sqlmigrate`**, que gera o código SQL:

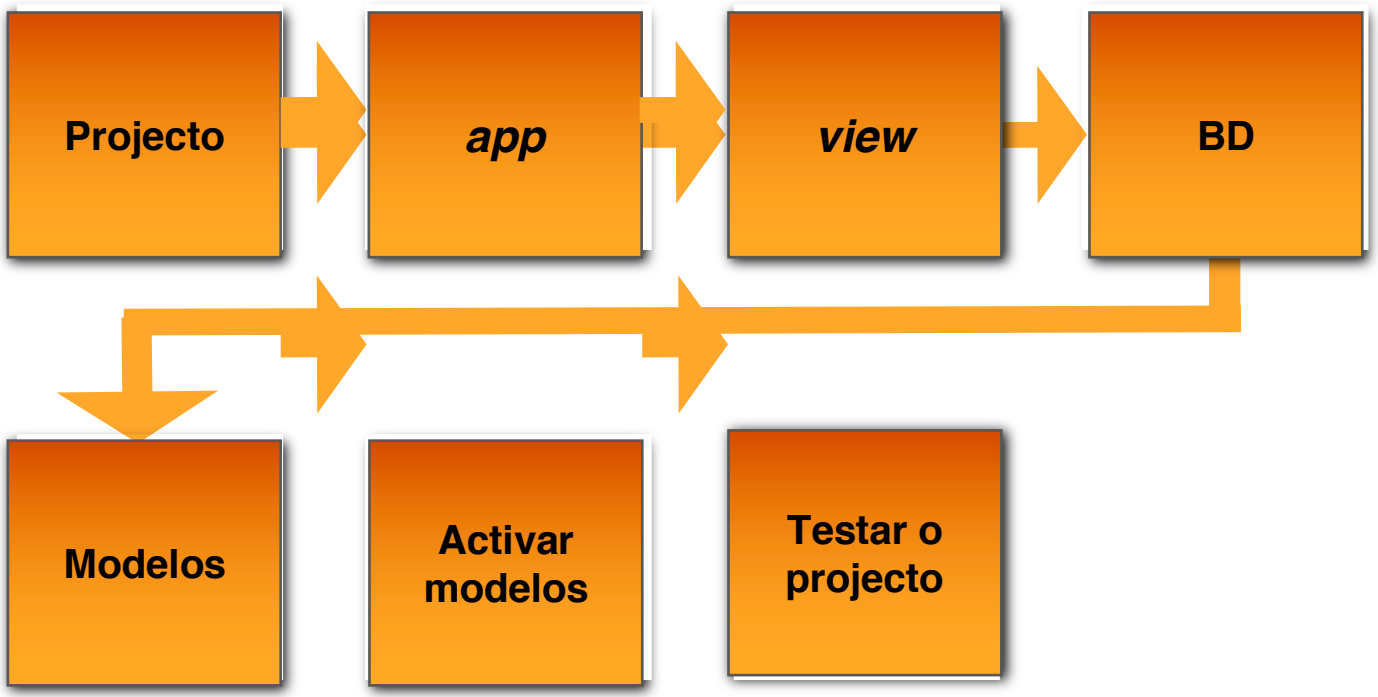
```
(venv) Jorges-MacBook:sitepr Jorge$ python manage.py sqlmigrate votacao 0001
BEGIN;
--
-- Create model Questao
--
CREATE TABLE "votacao_questao" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "questao_t
exto" varchar(200) NOT NULL, "pub_data" datetime NOT NULL);
--
-- Create model Opcao
--
CREATE TABLE "votacao_opcao" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "opcao_texto
" varchar(200) NOT NULL, "votos" integer NOT NULL, "questao_id" integer NOT NULL REFERENCES
 "votacao_questao" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "votacao_opcao_questao_id_e65d63a9" ON "votacao_opcao" ("questao_id");
COMMIT;
(venv) Jorges-MacBook:sitepr Jorge$
```

Finalmente crie as tabelas na base de dados, de acordo com o código SQL gerado:

```
(venv) Jorges-MacBook:sitepr Jorge$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, votacao
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying votacao.0001_initial... OK
(venv) Jorges-MacBook:sitepr Jorge$
```

O comando **`migrate`** atualiza a base de dados de acordo com as atualizações (*migrations*) dos modelos anteriormente realizadas.

A base de dados está agora atualizada de acordo com os modelos de dados.


Projecto
app
view
BD
Modelos
Activar modelos
Testar o projecto

7 - Testar o projecto na shell

Abra a shell:

```
(venv) Jorges-MacBook:sitepr Jorge$ python manage.py shell
Python 3.9.2 (v3.9.2:1a79785e3e, Feb 19 2021, 09:06:10)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Execute os seguintes comandos e verifique os resultados. **In** significa o input de uma instrução e **Out** significa o output respetivo.

In [1]: **`from votacao.models import Questao, Opcao`**

In [2]: **`Questao.objects.all()`**
Out[2]: **`<QuerySet []>`** # (ainda não existem instancias de Questao)

Crie uma nova Questao.
In [3]: **`from django.utils import timezone`**
In [4]: **`q = Questao(questao_texto="Vamos fazer uma festa no fim do ano? ", pub_data=timezone.now())`**

Grave a nova Questao na Base de Dados.
In [5]: **`q.save()`**

In [6]: **`q.id`**
Out[6]: **`1`**

In [7]: **`q.questao_texto`**
Out[7]: **`'Vamos fazer uma festa no fim do ano? '`**

In [8]: **`q.pub_data`**
Out[8]: **`datetime.datetime(2016, 3, 7, 11, 57, 37, 649060, tzinfo=<UTC>)`**

Altere o texto da questão q:

In [9]: **`q.questao_texto = 'Entao sempre nos decidimos a fazer uma festa no fim do ano? '`**

Nunca se esqueça de gravar após uma instanciação ou uma alteração:
In[10]: **`q.save()`**

In [11]: **`Questao.objects.all()`**
Out[11]: **`<QuerySet [<Questao: Questao object (1)>]>`**

A questão q já está na base de dados. No entanto ainda não conseguimos ler o seu texto.

Como descrever uma instância de Questao ou de Opcao quando esta é impressa na consola?

Adicione o metodo **`__str__()`** às classes **`Questao`** e **`Opcao`** (que já programou no ficheiro **`votacao/models.py`**): [1]

---

[1] Para poder importar a biblioteca **`six`** esta deverá estar préviamente instalada. Para isso use o comando

```
pip install six
```

```
from django.db import models
from django.utils import timezone
from six import string_types
import datetime

class Questao(models.Model):
    # ... (manter aqui os atributos)

    def __str__(self):
        return self.questao_texto


class Opcao(models.Model):
    # ... (manter aqui os atributos)

    def __str__(self):
        return self.opcao_texto
```

Aproveite para adicionar outro método que retorna True se a Questao foi publicada recentemente:

```
from django.db import models
from django.utils import timezone
from six import string_types
import datetime

class Questao(models.Model):
    # ...
    def foi_publicada_recentemente(self):
        return self.pub_data >= timezone.now() -
datetime.timedelta(days=1)
```

Grave **votacao/models.py**

Feche a shell para fazer as migrações e assim gravar as suas alterações às classes:

In [12]: **exit()**

Depois de alterar o modelo de dados (classes) deve sempre fazer as migrações (sequência de 3 instruções indicadas anteriormente).

Reabra a shell:

```
$ python manage.py shell
```

In [1]: `from votacao.models import Questao, Opcao`

In [2]: `Questao.objects.all()`
Out [2]: `[<Questao:  Entao sempre nos decidimos a fazer uma festa no fim do ano? >]`

In [3]: `Questao.objects.filter(id=1)`
Out[3]: `[<Questao:  Entao sempre nos decidimos a fazer uma festa no fim do ano? >]`

>>> In [4]:

```
Questao.objects.filter(questao_texto__startswith='Entao')
# nota: entre texto e startswith há 2 underscores
```

Out[4]: **[<Questao: Entao sempre nos decidimos a fazer uma festa no fim do ano? >]**

In [5]: **from django.utils import timezone**
In [6]: **current_year = timezone.now().year**
In [7]: **Questao.objects.get(pub_data__year=current_year)**
 # nota: entre pub_data e year há 2 underscores
Out[7]: **<Questao: Entao sempre nos decidimos a fazer uma festa no fim do ano? >**

In [8]: **Questao.objects.get(id=2)**
Out[8]: **...**
DoesNotExist: Questao matching query does not exist.

Atalho para usar uma chave primária:
In [9]: **Questao.objects.get(pk=1)**
Out[9]: **<Questao: Entao sempre nos decidimos a fazer uma festa no fim do ano? >**

In [10]: **q = Questao.objects.get(pk=1)**

In [11]: **`q.foi_publicada_recentemente()`**
Out[11]: **`True`**

O método create chama o construtor de instâncias
Opcao e insere-as num conjunto de uma Questao.

In [12]: **`q = Questao.objects.get(pk=1)`**

por enquanto q ainda não tem nenhuma Opcao
In [13]: **`q.opcao_set.all()`**
Out[13]: **`<QuerySet []>`**

Cria 3 instancias de Opcao associadas à Questao q:

In [14]: **`q.opcao_set.create(opcao_texto= 'Nao me apetece', votos=0)`**
Out[14]: **`<Opcao: Nao me apetece>`**

In [15]: **q.opcao_set.create(opcao_texto='Sim claro', votos=0)**
Out[15]: **<Opcao: Sim claro>**

In [16]: **c = q.opcao_set.create(opcao_texto='Nao sei', votos=0)**

A instância de Questao associada a uma Opcao pode ser acedida desta forma:
In [17]: **c.questao**
Out[17]: **<Questao: Entao sempre nos decidimos a fazer uma festa no fim do ano? >**

As instâncias de Opcao associadas a uma Questao podem ser acedidas desta forma:
In [18]: **q.opcao_set.all()**
Out[18]: **[<Opcao: Nao me apetece>, <Opcao: Sim claro>, <Opcao: Nao sei>]**

In [19]: **q.opcao_set.count()**
Out[19]: **3**

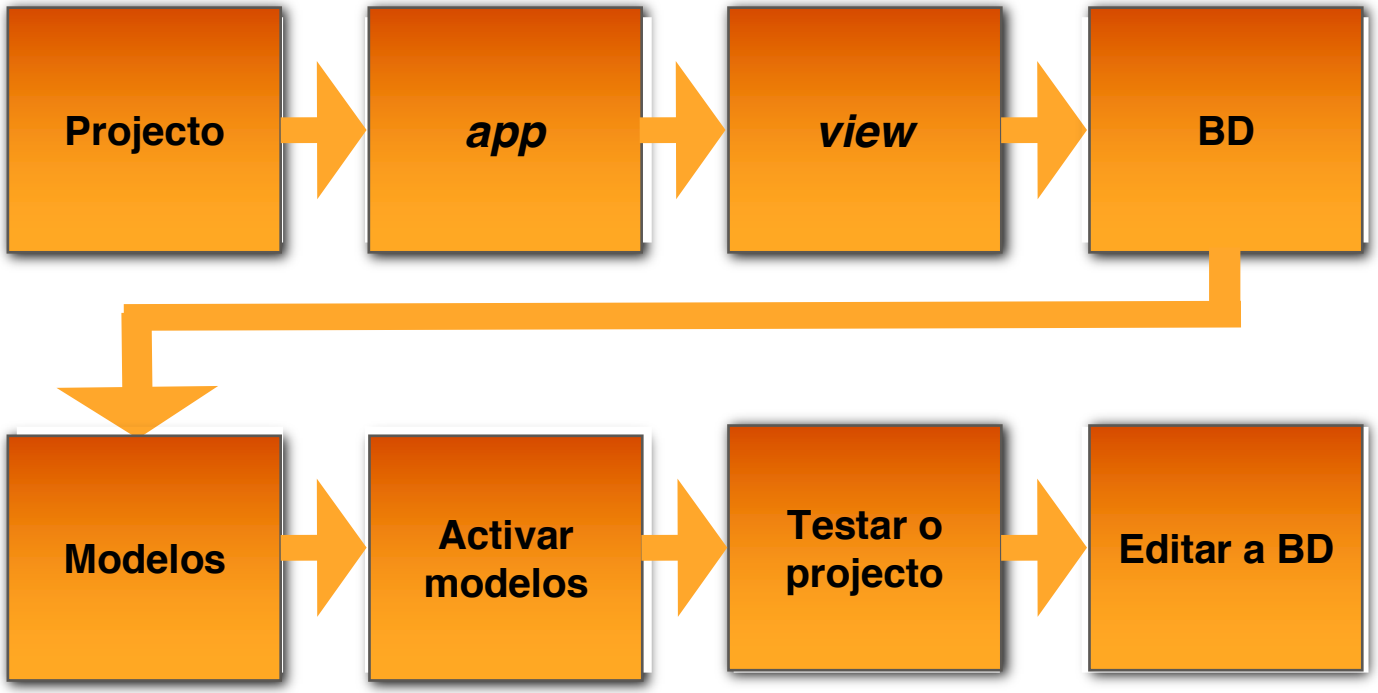
Procura de todas as Opcoes de uma Questao que tenha sido publicada este ano

In [20]:
```
Opcao.objects.filter(questao__pub_data__year=current_year)
```
Out[20]: `[<Opcao: Nao me apetece>, <Opcao: Sim claro>, <Opcao: Nao sei>]`

Apagar uma das Opcoes

In [21]:
```
c = q.opcao_set.filter(opcao_texto__startswith='Nao sei')
```
In [22]:
```
c.delete()
```
Out[22]: `(1, {'votacao.Opcao': 1})`

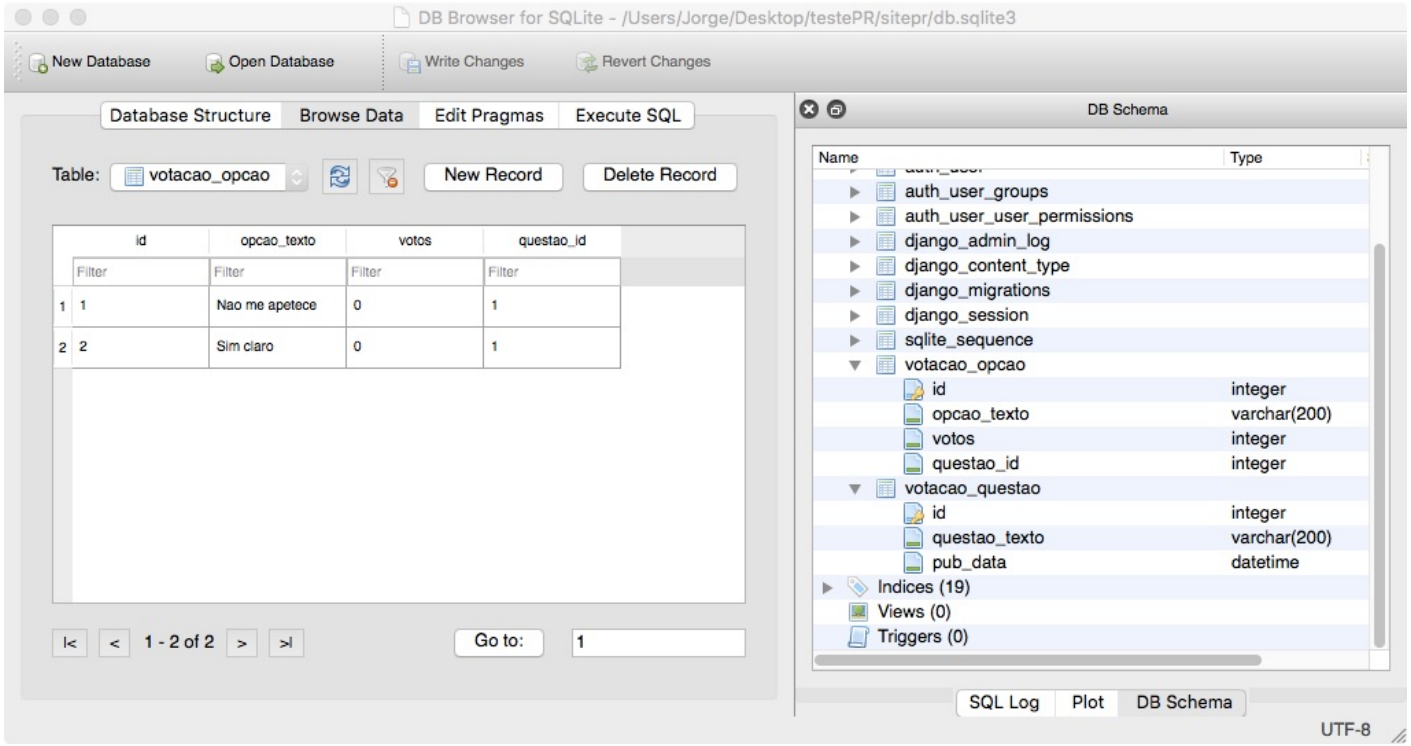
In [23]:
```
q.opcao_set.all()
```
Out[23]: `[<Opcao: Nao me apetece>, <Opcao: Sim claro>]`


Projecto
app
view
BD
Modelos
Activar modelos
Testar o projecto
Editar a BD

8 - Editar a Base de Dados

Faça o download e instale o “DB Browser for SQLite” (http://sqlitebrowser.org).

Edite a Base de Dados SQLite (ficheiro db.sqlite3) através do "DB Browser for SQLite". Verifique a estrutura da base de dados (tabelas **`votacao_questao`** e **`votacao_opcao`**) e os dados já existentes nestas tabelas.



Resumo

- Para criar um novo projecto:

```
django-admin startproject nomedoproj
```

- Para executar o servidor do projecto em localhost:

```
python manage.py runserver
```

- Para criar uma nova app:

```
python manage.py startapp nomedaapp
```

- Para criar uma nova view:

Edite o ficheiro **nomedaapp/views.py** e adicione o código seguinte:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello,world. Esta é a
    página de entrada da app votação.")
```

A nova view deve ser associada a um URL através de uma URLconf nos ficheiros **nomedoproj/urls.py** e **nomedaapp/urls.py**

- A parametrização da base de dados é feita em **nomedoproj/settings.py**. Django parametriza a base de dados Sqlite por defeito.

- Os modelos de dados são criados no ficheiro **nomedaapp/models.py**

- Os modelos de dados são activados em **nomedoproj/settings.py**

Seguidamente actualize as mudanças (*migrations*) realizadas nos modelos:

```
$ python manage.py makemigrations nomedaapp
```

Gere o código SQL:

```
$ python manage.py sqlmigrate nomedaapp 0001
```

e crie as tabelas na base de dados:

```
$ python manage.py migrate
```

- Teste o projecto na shell:

```
$ python manage.py shell
```

- Edite a sua base de dados com "DB Browser for SQLite".