

Criação e (Re)utilização de Código Próprio e de Terceiros Django Web Framework

1. Introdução

A programação de aplicações web pode constituir um conjunto de atividades e tarefas de complexa realização. A construção de raiz de aplicações web dinâmicas com ligação a bases de dados é um processo complexo dado o número de aspetos técnicos a considerar, tais como definir esquemas de bases de dados, efetuar e programar a ligação da base de dados à aplicação e ao servidor, definir a interface web do lado do cliente tendo em conta os aspetos de usabilidade, ter em atenção os aspetos de segurança, entre muitas outras tarefas de programação, integração e configuração associadas à arquitetura de software de uma aplicação web. As frameworks de software para o desenvolvimento de aplicações web têm como principal objetivo minimizar, gerir e integrar os aspetos de programação e configuração de software para a web, fundamentalmente permitir e facilitar uma integração coerente entre as diferentes camadas (layers) de uma aplicação web.

Normalmente, uma framework está associada a uma determinada linguagem de programação. A crescente popularidade e utilização da linguagem Python promoveu o desenvolvimento de diversos ambientes (frameworks) de software para a construção de aplicações web. Este material prático analisa um exemplo ilustrativo e globalmente reconhecido como ferramenta integradora (framework) para a construção de aplicações web em Python: o Django.

2. Django Web Framework

Django é um ambiente (framework) de desenvolvimento de software que simplifica significativamente o processo de construção de uma aplicação web. A framework Django foi concebida para facilitar o design, programação e integração de recursos de software para a web. A abordagem principal é minimizar os detalhes de implementação no âmbito da programação web, reduzindo os aspetos técnicos e os detalhes de integração com uma base de dados, assim como a gestão das diferentes sessões (e URL) de uma aplicação web.

A framework Django foi construída com base na linguagem de programação Python. O Python possui muitos recursos avançados, além de uma biblioteca padrão de software poderosa, o que faz com que esta linguagem, em conjunto com o ambiente de software Django, possam ser usados em aplicações complexas em servidores web, assim como em dispositivos móveis.

A conceção deste ambiente (framework) assim como o desenvolvimento subjacente de aplicações web rege-se pela metáfora e princípio designado por Don't Repeat Yourself (DRY), o qual enfatiza a redução de erros de implementação com base na eliminação de repetições (duplicações) de funções e procedimentos e respetivo código-fonte. No âmbito da programação e integração de componentes de software, ou seja, numa abordagem à engenharia de software, a metáfora DRY pretende representar que uma determinada função, procedimento, processo, entidade ou área de conhecimento do sistema em análise tem associado um artefacto (componente) de software reutilizável. Esta abordagem tem também por objetivo minimizar aspetos do processo de manutenção de software.

A framework Django assegura um ambiente de desenvolvimento de software de alto nível de abstração, o que permite construir aplicações web, de dimensão considerável, com um número reduzido de linhas de código-fonte. O Django é considerado uma ferramenta simples, robusta e flexível no sentido de conceber e desenvolver aplicações

web com o mínimo possível de tarefas de programação intensiva. O Django foi construído em Python e combina a capacidade das linguagens orientadas aos objetos com as linguagens de scripting (e.g., Ruby). Esta característica integradora permite aos programadores web a criação de aplicações para a resolução de um leque mais alargado de problemas do mundo real e das empresas.

O Django é uma framework escrita em Python para o desenvolvimento de aplicações web. Dois conceitos estão subjacentes à conceção da framework Django, o padrão (design pattern) Model-Template-View (MTV) e o princípio DRY (Don't Repeat Yourself) referido anteriormente. O padrão MTV tem as seguintes características principais que definem um padrão de desenvolvimento que separa o processo de desenvolvimento em três camadas:

- **Model** – Definição do modelo conceptual de base de dados no modelo (lógico) relacional. Um modelo em Django é uma classe Python que representa uma tabela de uma base de dados relacional. O processo de codificação SQL para a criação e manipulação da base de dados é automatizado;
- **View** – Componente (view) Django que contém o código-fonte referente à lógica da aplicação (business logic);
- **Template** – Define a interface do sistema através de um conjunto de páginas HTML que, por sua vez, definem a interação web com o utilizador. O Django tem um sistema de templates (Django templating system) com capacidades muito interessantes de interface (e interação) web.

3. Instalação da Framework Django

O processo de instalação da framework Django deverá ser efetuado com base nas instruções do website (<https://www.djangoproject.com/download/>). Existem diferentes formas de instalar a framework, assim como diferentes versões de instalação. Por um lado, temos a última versão em desenvolvimento (Django development version), que inclui as funcionalidades mais recentes da framework. Por outro lado, temos uma versão estável da plataforma (latest official version), mais indicada para produção. O primeiro passo é definir qual a versão do Django instalar, para então prosseguir com o download e instalação.

Neste material teórico, foi adotada a versão estável disponível no momento da redação (novembro de 2024, versão 5.1.3). Para simplificar o processo, utilizamos o gestor de pacotes pip, que é integrado ao ambiente Python e bastante utilizado para a instalação de bibliotecas e frameworks.

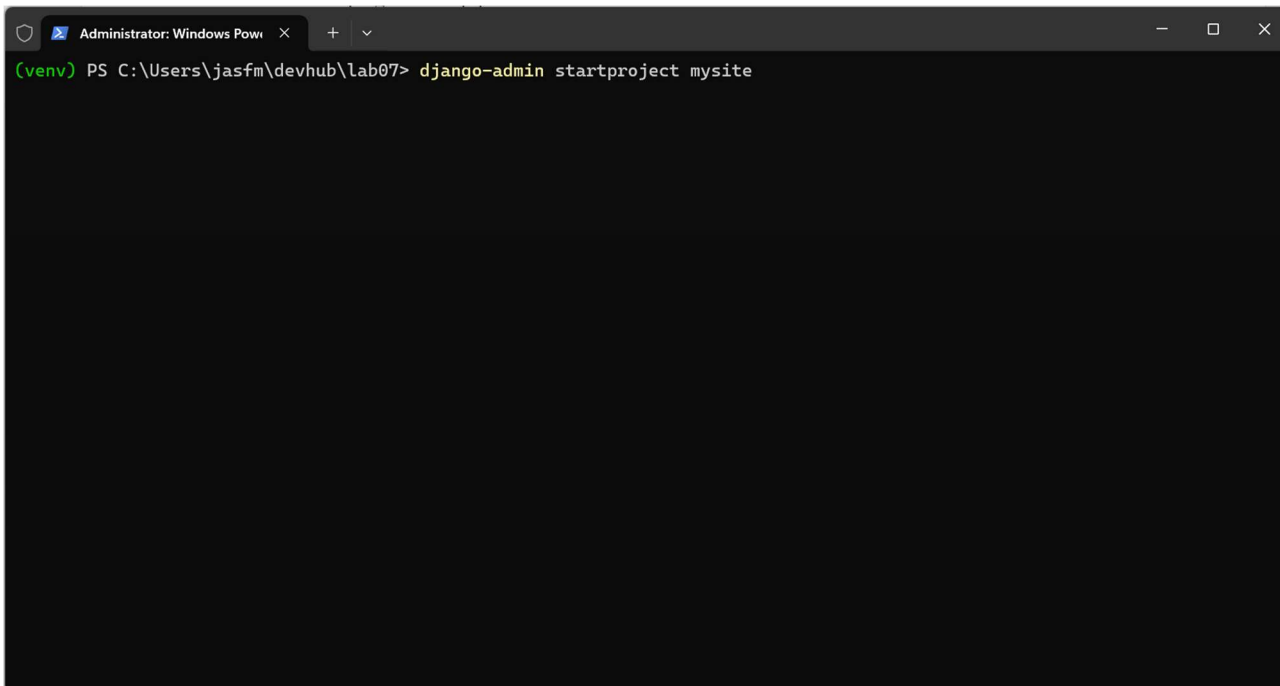
Para instalar o Django utilizando o pip, siga os seguintes passos:

1. Certifique-se de que o Python e o pip estão instalados:
 - 1.1. Execute o comando `python --version` (ou `python3 --version`, dependendo da configuração do seu sistema) para verificar se o Python está instalado.
 - 1.2. Em seguida, confirme a presença do pip com o comando `pip --version`.
2. Instale o Django com o pip:
 - 2.1. Para instalar a versão mais recente e estável, use o comando `pip install django`.
 - 2.2. Caso prefira instalar uma versão específica, como a versão 5.1.3., utilize `pip install django==5.1.3`.
3. Verifique a instalação:
 - 3.1. Após a instalação, confirme que o Django foi instalado corretamente executando `django-admin --version`.

4. Criar um Projeto Django

Os próximos procedimentos e respetivo código têm por base a documentação da framework Django. Para criar um novo projeto é necessário definir um conjunto de código para criar uma nova instância Django, incluindo as instruções para a configuração da base de dados de suporte da aplicação. Deste modo, na linha de comando do sistema operativo, devemos aceder ao diretório onde reside o ambiente (django-admin) de administração da framework. Podemos adicionar o caminho (path) às variáveis do sistema, de modo a ter a instrução disponível através de qualquer diretório.

Para criar um projeto Django, na linha de comando executamos a seguinte instrução: `django-admin startproject mysite <designacao-do-projeto>` conforme exemplo especificado na Figura 1 para um projeto Django designado por mysite.



```
Administrator: Windows PowerShell
(venv) PS C:\Users\jasfm\devhub\lab07> django-admin startproject mysite
```

Figura 1. Criação de um projeto Django.

Após executar a instrução `django-admin startproject <designacao-do-projeto>`, um conjunto de diretórios e ficheiros são criados a partir dessa localização (Figura 2):

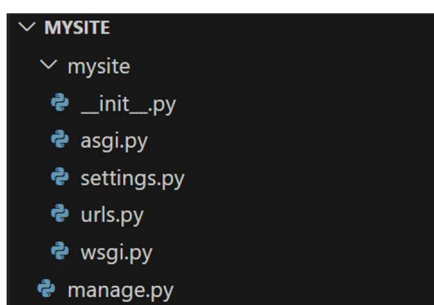


Figura 2. Diretórios e ficheiros criados para um projeto Django.

No contexto de um novo projeto e website Django, estes ficheiros têm o seguinte significado:

- `mysite/` – É o nome (exemplo) atribuído a este projeto e representa o diretório raiz (root directory), no qual serão inseridos os recursos (ficheiros) deste projeto Django referentes ao processo de desenvolvimento de software e respetiva aplicação web;
- `manage.py` – É uma ferramenta utilizada através de uma linha de comando do sistema (command line) que permite interagir e configurar o projeto Django em desenvolvimento. Os detalhes e funcionalidades desta ferramenta poderão ser consultados nos próprios ficheiros Python `manage.py` e `django-admin.py`;
- `mysite/mysite/` – É um diretório que representa o nome da package Python para o projeto Django. Esta designação será utilizada para referenciar a package Python e importar outros recursos para o projeto, por exemplo `mysite.urls` que refere os URL utilizados através do website;
- `__init__.py` – Representa inicialmente um ficheiro vazio (sem conteúdo) e indica o ambiente Python que este diretório (`mysite`) deverá ser considerado como uma package Python para a integração e manipulação dos recursos do projeto Django;
- `asgi.py` – É um ficheiro Python que configura a interface ASGI (Asynchronous Server Gateway Interface) para suporte a comunicações assíncronas, permitindo funcionalidades como web sockets no projeto Django. É essencial para a execução de aplicações Django em servidores compatíveis com ASGI.
- `settings.py` – É o ficheiro de configuração para o projeto Django de modo a definir e atualizar parâmetros de desenvolvimento do projeto, por exemplo a definição do motor de base de dados;
- `urls.py` – Referencia os URL do projeto Django. Este ficheiro incorpora a estrutura (mapa de navegação) do projeto e respetiva aplicação web;
- `wsgi.py` – É um ficheiro Python que define os servidores (Web Server Gateway Interface – WSGI) compatíveis para o desenvolvimento do projeto Django. WSGI é uma especificação da interface entre servidores web e aplicações ou frameworks para a linguagem Python.

De modo a organizar o código-fonte a especificar, deverá ser definido outro diretório para armazenar o código Python para um projeto Django. Neste exemplo foi criado um diretório `mycode` no qual serão colocados novos programas para o desenvolvimento do projeto Django.

5. Configuração de um Gestor de Base de Dados

O processo de instalação e configuração de um motor de bases de dados é necessário, no caso de pretendermos desenvolver uma aplicação web de dimensão significativa, com um número elevado de transações e acessos à base de dados. Para uma aplicação web simples temos o motor SQLite incorporado no ambiente Python e Django. Neste contexto, no caso de pretendermos desenvolver uma aplicação de grande dimensão, podemos utilizar um dos seguintes gestores de bases de dados: PostgreSQL, MySQL ou Oracle.

Nos próximos exemplos e aplicações web desenvolvidas no ambiente Django, vamos aplicar o motor de bases de dados SQLite. Como esta base de dados faz parte integrante do ambiente Python, não será necessário efetuar qualquer tipo de instalação ou configuração para a base de dados de um projeto Django. O SQLite é uma opção simples para o desenvolvimento da aplicação web, dado que não requer a implementação de um servidor de base de dados, separado do servidor web.

A variável `NAME` (Figura 3) define o nome da base de dados. No caso de o projeto Django utilizar o SQLite, a base de dados será refletida num ficheiro no disco do computador. O valor por defeito especificado por `BASE_DIR / 'db.sqlite3'` define que esse ficheiro, que é representativo da base de dados, será armazenado no diretório do projeto (neste exemplo, `c:/.../mysite/`).

```
72
73 # Database
74 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.sqlite3',
79         'NAME': BASE_DIR / 'db.sqlite3',
80     }
81 }
82
```

Figura 3. Ficheiro Python de configuração (parâmetros, settings.py) de um projeto.

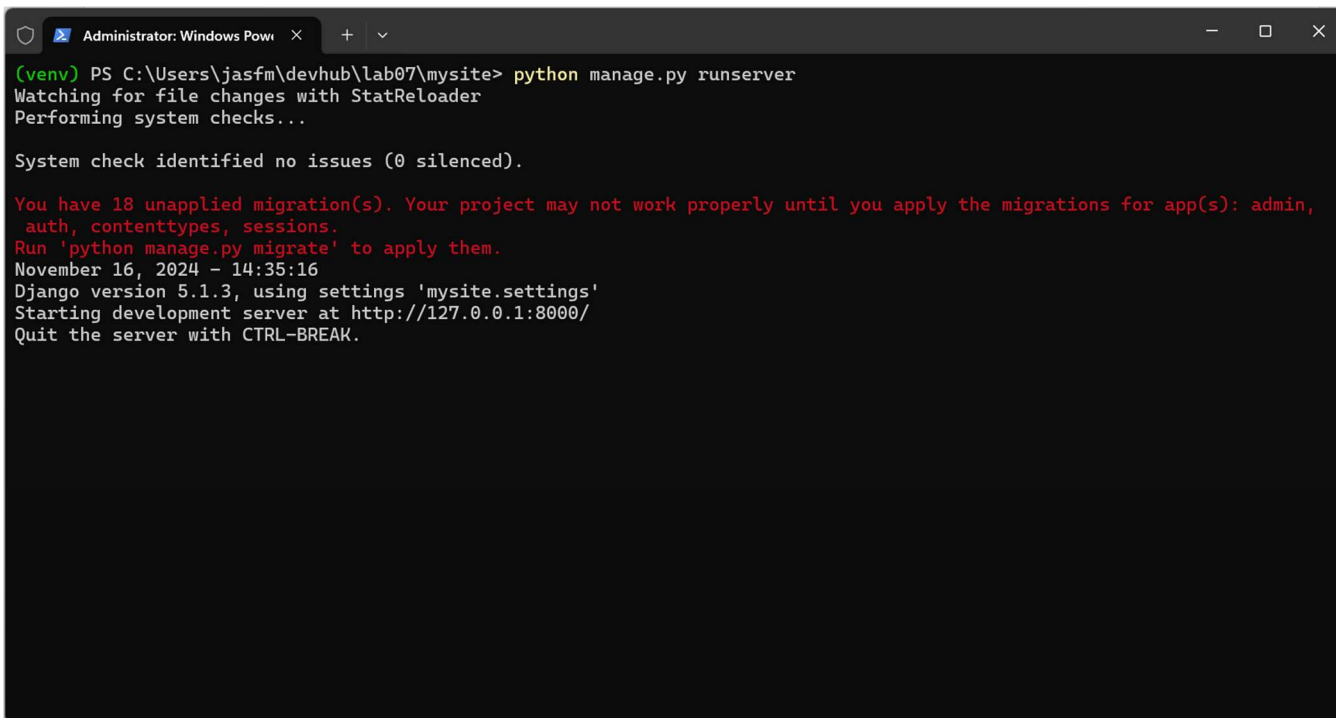
No caso de pretendermos desenvolver uma aplicação web com outro gestor de base de dados, será necessário instalar o respetivo software (processo de instalação do gestor de base de dados escolhido) e, posteriormente, alterar a variável (dicionário Python) DATABASES referenciadas no ficheiro settings.py. Por defeito, este ficheiro de configuração contém esta variável com a designação do motor SQLite. No caso de se utilizar outro motor de bases de dados (e.g., MySQL, PostgreSQL ou Oracle), será necessário alterar desta variável com a designação da outra base de dados escolhida, assim como os restantes parâmetros de conexão à base de dados. Com base no motor (ENGINE) selecionado (SQLite), temos as seguintes opções para a definição da conexão a uma base de dados:

- **SQLite** (valor por defeito) – django.db.backends.sqlite3
- **PostgreSQL** – django.db.backends.postgresql
- **MariaDB** – django.db.backends.mysql
- **MySQL** – django.db.backends.mysql
- **Oracle** – django.db.backends.oracle

6. Servidor de Desenvolvimento

A framework Django inclui um servidor web leve e simples, de modo a permitir o desenvolvimento da aplicação e website. Este servidor incorporado no ambiente Django permite a prototipagem e o desenvolvimento rápido de uma aplicação web. Contudo, este servidor web é somente aplicado como servidor de desenvolvimento e testes do software em construção e não deverá ser aplicado como servidor de produção. Para uma efetiva implementação e produção de uma aplicação web em ambiente Django, será necessário aplicar um servidor web robusto como é o caso do servidor Apache.

Neste contexto, os próximos exemplos e aplicações web têm um carácter ilustrativo (prototipagem web). Para implementar e colocar em produção (web deployment) uma aplicação web, teremos necessariamente de instalar e configurar outro servidor web, como o Apache ou o Microsoft IIS. O próximo exemplo descreve a forma simples como iniciamos o servidor web de desenvolvimento, incorporado por defeito no ambiente Django. Este servidor acompanha o código desenvolvido e de uma forma automática reinicia para facilitar as alterações efetuadas ao projeto, e decorrentes da construção do software. Para iniciar o servidor de desenvolvimento, executamos a instrução python manage.py runserver no diretório do projeto (Figura 4). Como resultado, temos as seguintes notificações:



```
Administrator: Windows PowerShell
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 16, 2024 - 14:35:16
Django version 5.1.3, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figura 4. Instrução para a criação do website para um projeto Django.

Após executar a instrução anterior, e se não ocorrerem erros, o servidor de desenvolvimento especificado em Python inicia a sua função. Enquanto o servidor está em execução, podemos abrir uma sessão no web browser com o endereço: `http://127.0.0.1:8000/`. Neste sentido, temos acesso à página de entrada “The install worked successfully! Congratulations!” (Figura 5).

Por defeito, a instrução para executar o servidor (`runserver`) inicia o servidor de desenvolvimento no IP (Internet Protocol) interno na porta 8000. No caso de pretender utilizar outra porta, por exemplo a 8080, será necessário executar a seguinte instrução no diretório do projeto, de modo a iniciar o servidor web na porta 8080:

```
python manage.py runserver 8080
```

No caso de pretender alterar o IP, procedemos de forma idêntica. Por exemplo, para receber qualquer IP público, de modo a disponibilizar o projeto e website Django noutras máquinas (network interfaces) no sentido de tornar acessível o projeto para outros utilizadores e programadores, podemos aplicar a seguinte instrução:

```
python manage.py runserver 0.0.0.0:8000
```

7. Modelos Django e Interação com Bases de Dados

A partir do momento em que um projeto Django está definido, temos o ambiente de software preparado para o desenvolvimento de aplicações (apps). Cada aplicação Django consiste numa package Python. O ambiente Django tem uma ferramenta para gerar o diretório principal representativo da aplicação (Django app), de modo a organizar uma estrutura (hierarquia) de ficheiros-fonte Python.

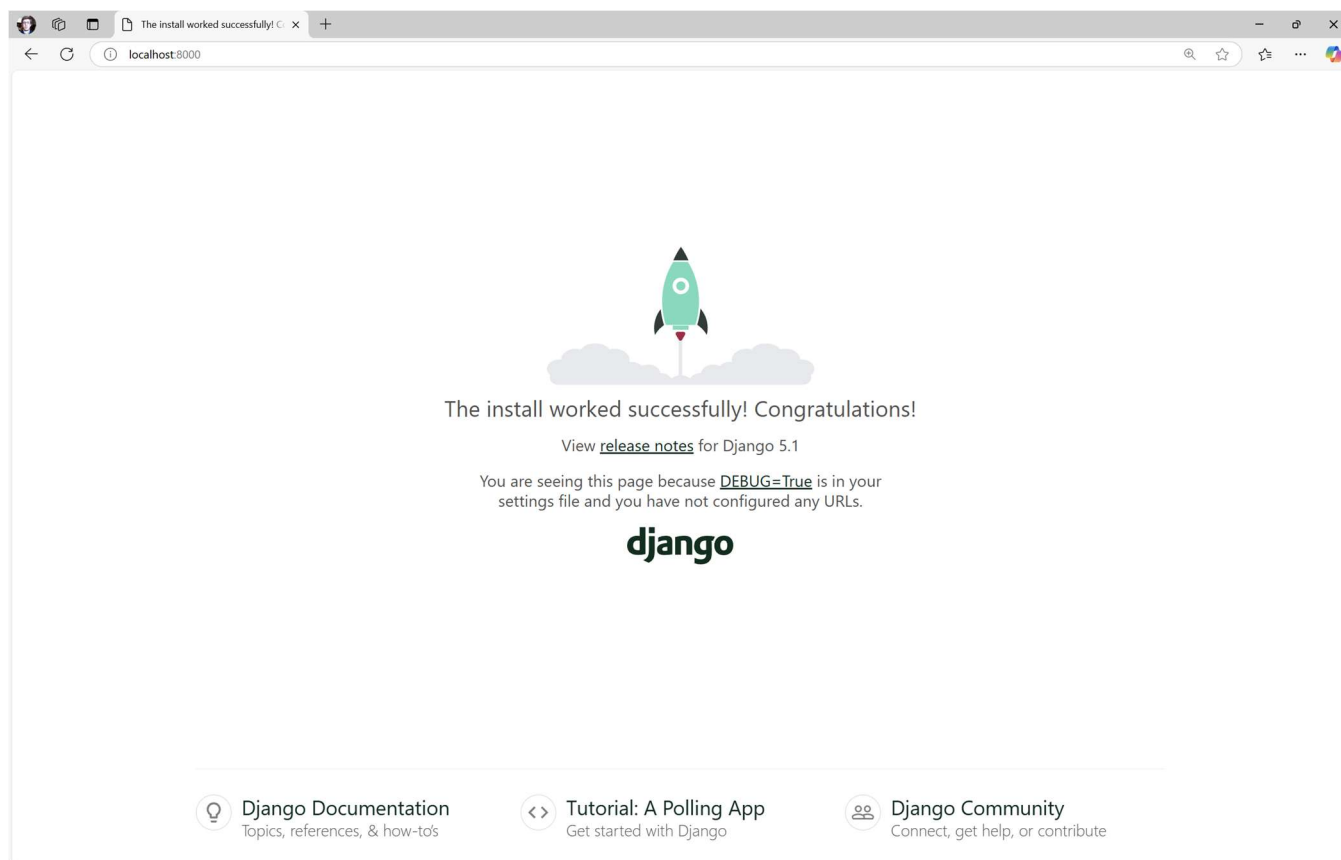


Figura 5. Primeiro projeto e website Django.

Os dados e a interação com base de dados são fundamentais para o desenvolvimento de aplicações web. A framework Django incorpora um conjunto de funcionalidades para a definição de estruturas de dados e para a implementação de mecanismos de persistência dos dados. Os modelos Django têm subjacentes os aspetos e as práticas atuais do design e desenvolvimento web através do modelo MTV (Model-Template-View). Um modelo permite a definição de uma estrutura geral de uma aplicação web, de modo a parametrizar os mecanismos de persistência de dados entre os pedidos e respostas do servidor web (request-response) e respetivas sessões.

Os modelos Django são definidos com classes (OO) Python. O ambiente Django contém um componente designado por mapeador objeto-relacional (Object Relational Mapper – ORM) que mecaniza a ligação do programa Python, e respetivas classes e instâncias, com as bases de dados que possuem representação (lógica) relacional. O mapeador ORM permite ligações à base de dados sem utilização (especificação de código-fonte) da linguagem SQL, considerada a forma tradicional de acesso ao conteúdo de uma base de dados.

Um modelo define uma estrutura de dados única para a representação e comportamento das entidades do domínio e respetivo sistema de software. Um modelo define os atributos (estrutura) e as operações que definem o comportamento dos dados do sistema a manipular e armazenar. Conforme referido anteriormente, o design e o desenvolvimento de software em Django têm subjacente o princípio DRY, o qual enfatiza a definição de modelos (estrutura dos dados e comportamento) e a sua reutilização para o desenvolvimento das aplicações web.

7.1. Projetos e Aplicações Django

Um projeto Django pode conter diferentes aplicações e uma determinada aplicação pode fazer parte de diferentes projetos. Uma aplicação Django está associada a um determinado domínio e efetua um conjunto de operações e transações, com possíveis ligações a bases de dados, por exemplo um sistema de weblog ou uma aplicação web para ler e atualizar registos de uma base de dados. Por sua vez, um projeto Django caracteriza a configuração de um conjunto de aplicações para um determinado website (Django website). Neste contexto de design de software, e com base no exemplo do projeto anterior, diferentes aplicações podem ser criadas a partir do caminho (path) `c:/.../mysite/` junto do ficheiro `manage.py`. Os próximos exemplos e respetivo código-fonte serão especificados neste diretório representativo do projeto Django previamente designado por `mysite`.

7.2. Criação de uma Aplicação (Django App)

Para criar uma nova aplicação (app) Django, acedemos ao diretório que contém o ficheiro `manage.py` e executamos a seguinte instrução numa janela de comando (window command line) do sistema operativo. A aplicação deverá ser criada no diretório que contém o ficheiro de configuração `manage.py`:

```
python manage.py startapp polls
```

Esta instrução cria um conjunto de ficheiros e um diretório (migrations), com a seguinte configuração apresentada a seguir. O diretório `polls` com a designação de aplicação Django contém os recursos para o desenvolvimento de software da aplicação web (Figura 6).

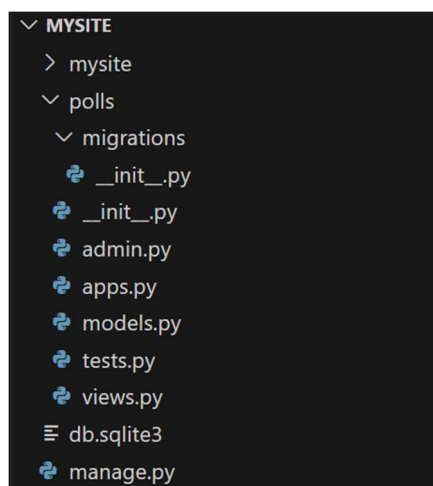


Figura 6. Estrutura do projeto após a criação da aplicação `polls`.

Após a criação da aplicação (app), o passo seguinte é a definição da estrutura da base de dados (database web app) através da criação dos respetivos modelos. Os modelos especificados em Python no ficheiro-fonte `models.py` (Figuras 7 e 8) definem o esquema da base de dados.


```
polls > models.py
1  from django.db import models
2
3  # Create your models here.
4
```

Figura 7. Ficheiro Python `models.py` para a criação de modelos.

```
polls > models.py > ...
1  from django.db import models
2
3
4  class Question(models.Model):
5      question_text = models.CharField(max_length=200)
6      pub_date = models.DateTimeField("date published")
7
8
9  class Choice(models.Model):
10     question = models.ForeignKey(Question, on_delete=models.CASCADE)
11     choice_text = models.CharField(max_length=200)
12     votes = models.IntegerField(default=0)
13
```

Figura 8. Ficheiro `models.py` com dois modelos (classes Python).

Este exemplo simples de uma aplicação define um sistema de registo na web de questões e respetivos comentários. Para isso, são criados dois modelos: `Question` e `Choice`. O primeiro contém dois atributos: uma questão e a respetiva data de publicação. O segundo (`Choice`) contém também dois atributos: uma opção de escolha e um registo de contagem. Cada opção de escolha tem associada uma questão previamente definida. Estes conceitos são representados através de classes (object oriented) Python.

Os modelos são representados com classes Python e com uma sintaxe simples para a definição dos atributos e métodos. Cada modelo é representado através de uma classe, as quais, por sua vez, são subclasses da classe `Model` (`django.db.models.Model`). Cada modelo contém um conjunto de variáveis de classe representativas dos campos da base de dados. Neste sentido, cada campo (field) está representado pela instância da classe `Field`. Por exemplo, `CharField` para campos (atributos) do tipo alfanumérico (texto), e `DateTimeField` para campos do tipo data e hora (datetime). Desta forma, o ambiente Django define os tipos de dados dos atributos do sistema de software e respetiva base de dados.

Cada classe contém também a definição dos atributos (ou campos) de instância (field instances). No exemplo anterior, os atributos `question_text` e `pub_date` da classe `Question`, e os atributos `question`, `choice_text`, e `votes` da classe `Choice`, são usados na implementação Python, assim como a designação dos campos (colunas) das tabelas criadas na base de dados de suporte da aplicação.

De modo opcional, é possível definir outro nome para uma determinado campo, de modo a refletir uma leitura mais compreensiva (human-readable) para o utilizador. Por exemplo, o atributo `codfunc` de uma classe `Funcionario` poderá ser definido 'Cod. Funcionário' através do seguinte código:

```
class Funcionario(models.Model):  
    codfunc = models.IntegerField(max_length=5, verbose_name='Cod. Funcionario')  
    nome = models.CharField(max_length=50)  
    ...
```

Alguns atributos de classe requerem determinados argumentos. Por exemplo, o atributo `CharField` tem associado o valor máximo de caracteres (`max_length`) para esse atributo de instância. Com base no exemplo anterior, temos o código do funcionário no máximo com 5 dígitos, e o nome no máximo com 50 caracteres. Um campo também pode ter um valor por defeito (`default value`) definido no modelo (classe) através do argumento `default = value`.

Outro aspeto fundamental na definição de modelos Django e respetivas classes é a especificação de relacionamentos através do argumento `ForeignKey` que representa o conceito (base de dados) de chave importada ou chave estrangeira. Django suporta diferentes formas de multiplicidade entre entidades (classes), tais como:

- **1:N** – Um para muitos (`many-to-one`);
- **N:M** – Muitos para muitos (`many-to-many`);
- **1:1** – Um para um (`one-to-one`).

7.3. Ativação de Modelos Django

Após a definição dos modelos (classes) associados ao domínio da aplicação, é necessário implementar os modelos através da criação do respetivo esquema de base de dados. Com base no exemplo anterior (`polls` app), o código Python do ficheiro `models.py` permite à framework Django efetuar as seguintes operações para a criação das tabelas da base de dados e uma API para aceder aos objetos (instâncias) `Question` e `Choice` definidos em `models.py`.

O primeiro passo para ativar a aplicação e os respetivos modelos é a indicação ao projeto Django que a aplicação (`polls`) está instalada. Este passo é efetuado através da edição do ficheiro `settings.py` no diretório do projeto. Neste sentido, é necessário inserir a nova aplicação e a alterar a variável `INSTALLED_APPS`, incluindo a referência à aplicação (string '`polls`') em desenvolvimento (Figura 9).



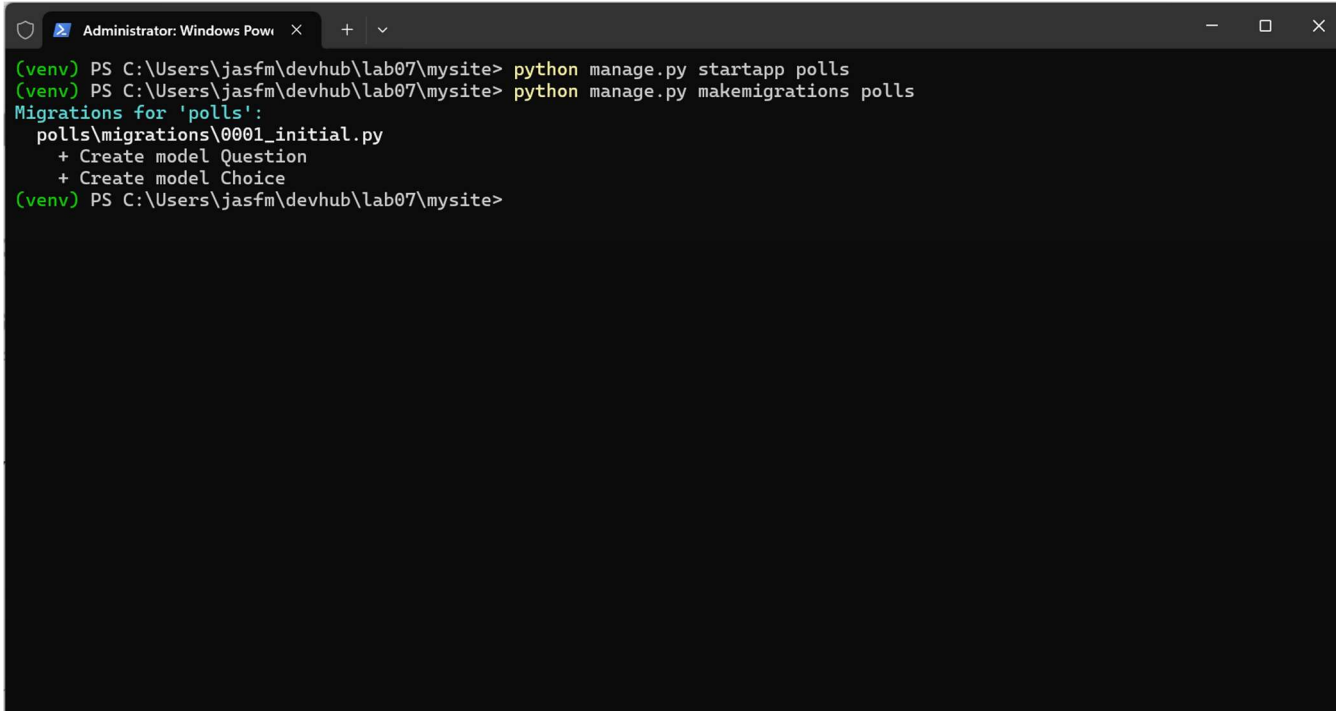
```
mysite > settings.py > ...  
30  
31 # Application definition  
32  
33 INSTALLED_APPS = [  
34     'django.contrib.admin',  
35     'django.contrib.auth',  
36     'django.contrib.contenttypes',  
37     'django.contrib.sessions',  
38     'django.contrib.messages',  
39     'django.contrib.staticfiles',  
40     'polls'  
41 ]  
42
```

Figura 9. Inserção de uma nova app num projeto Django.

Após incluir a aplicação na parametrização do projeto, será necessário executar a seguinte instrução para a ativação efetiva da aplicação:

```
python manage.py makemigrations polls
```

Esta instrução cria os modelos Question e Choice previamente definidos e define um campo de ligação (relacionamento) entre as classes. A próxima figura (Figura 10) ilustra o resultado desta operação.



```
Administrator: Windows PowerShell
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py startapp polls
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    + Create model Question
    + Create model Choice
(venv) PS C:\Users\jasfm\devhub\lab07\mysite>
```

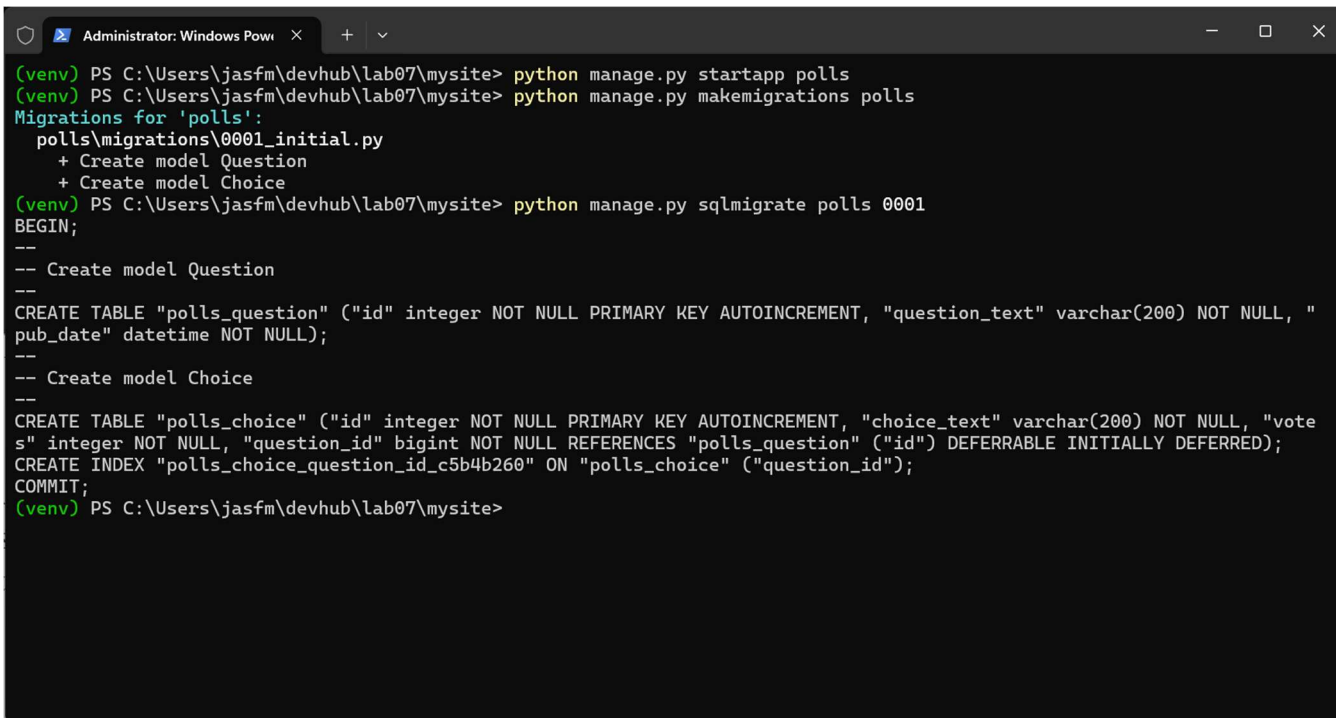
Figura 10. Processo de criação dos modelos de uma app num projeto Django.

A execução da instrução `makemigrations` permite o registo em memória secundária dos modelos previamente definidos. `makemigrations` cria um conjunto de ficheiros representativos do esquema conceptual da base de dados de suporte da aplicação Django. A partir deste momento é possível definir o esquema de base de dados através da instrução `sqlmigrate` para a criação do código SQL (Figura 11) que define a estrutura da base de dados.

A instrução `sqlmigrate` define automaticamente o esquema da base de dados. Para isso, executamos na linha de comandos (Figura 11) do sistema operativo no diretório que contém o ficheiro do projeto `manage.py`, o seguinte comando:

```
python manage.py sqlmigrate polls 0001
```

Com base nos modelos Question e Choice definidos em `models.py`, o resultado completo da execução do comando anterior reflete um conjunto de instruções SQL que definem a base de dados de suporte da aplicação Django.



```
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py startapp polls
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    + Create model Question
    + Create model Choice
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py sqlmigrate polls 0001
BEGIN;
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "question_text" varchar(200) NOT NULL, "pub_date" datetime NOT NULL);
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" bigint NOT NULL REFERENCES "polls_question" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");
COMMIT;
(venv) PS C:\Users\jasfm\devhub\lab07\mysite>
```

Figura 11. Processo de criação dos modelos de uma app num projeto Django.

O resultado da execução deste comando depende do motor de base de dados que estamos a utilizar. Conforme vimos anteriormente, para aplicações web simples, podemos aplicar o motor SQLite que está incorporado por defeito no ambiente Python e Django. Para aplicações de maior dimensão, poderemos aplicar os gestores de bases de dados MySQL ou PostgreSQL.

A designação das tabelas é automaticamente gerada através da combinação da designação da aplicação (neste exemplo, `polls`) com a designação do modelo (classe Python), por exemplo `polls_question` (Figura 11). Esta abordagem (terminologia) pode ser reescrita pelo programador se assim o pretender. No contexto da definição da estrutura da base de dados em SQL, existem outros aspetos de sintaxe a ter em conta. Qualquer um destes aspetos (código gerado) pode também ser reescrito:

- As chaves primárias (`id`) são também adicionadas automaticamente;
- Por convenção, o ambiente Django adiciona `_id` aos campos que representam chaves importadas (foreign key);
- O relacionamento subjacente à definição de uma chave importada é explicitado através da restrição (instrução) `FOREIGN KEY`;
- Dependendo do gestor de base de dados, são aplicadas os tipos de dados numéricos (inteiros) de incremento automático, com a seguinte sintaxe: `auto_increment` (MySQL), `serial` (PostgreSQL), ou `integer primary key autoincrement` (SQLite).

A instrução `sqlmigrate` antecipa e especifica o código (scripts) SQL para a criação da estrutura da base de dados. Contudo, nesta fase ainda não foi criada fisicamente a base de dados. Para isso, é necessário executar a instrução `migrate` a seguir especificada (Figura 12) para a efetiva criação da base de dados e respetivas tabelas:

```
python manage.py migrate
```

```
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" bigint NOT NULL REFERENCES "polls_question" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");
COMMIT;
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
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 polls.0001_initial... OK
  Applying sessions.0001_initial... OK
(venv) PS C:\Users\jasfm\devhub\lab07\mysite>
```

Figura 12. Código SQL associado aos modelos de uma app num projeto Django.

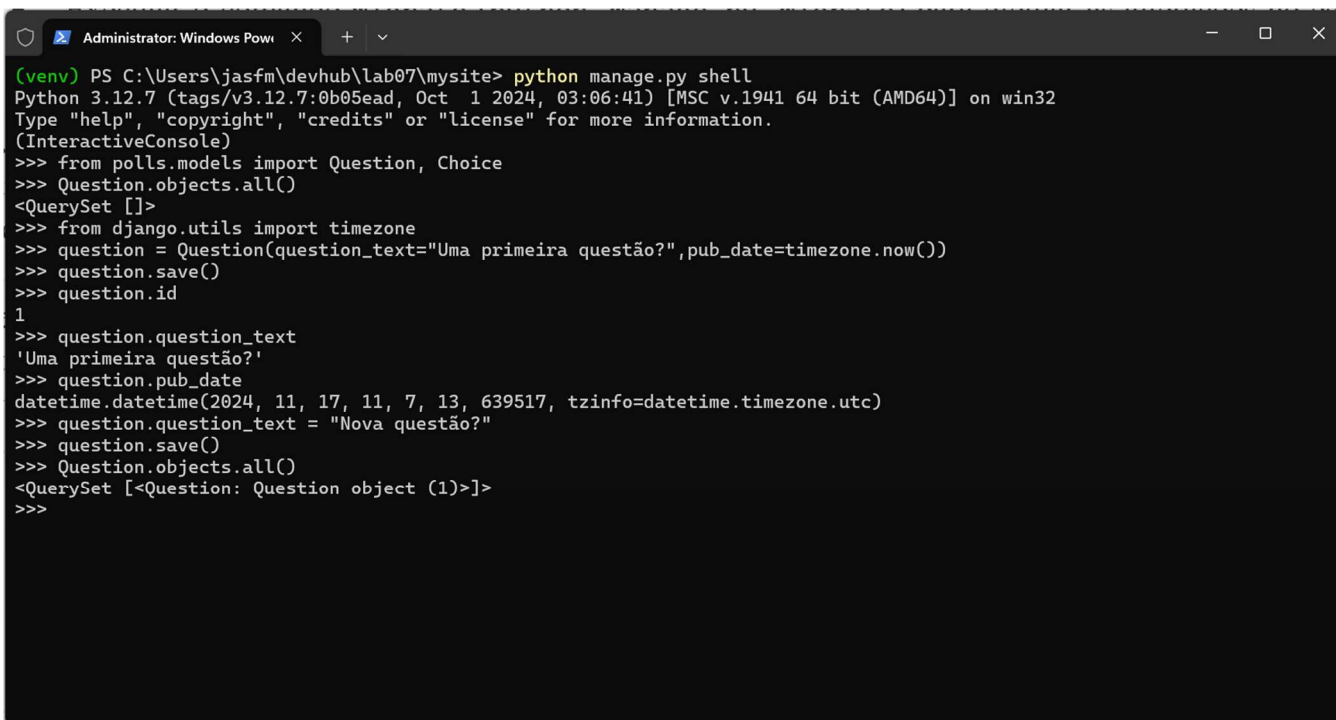
A execução da instrução migrate permite sincronizar as alterações efetuadas nos modelos com o esquema da base de dados. Desta forma, é possível alterar de um modo dinâmico os modelos de um determinado projeto Django, sem necessitar de eliminar e criar novas tabelas para essa tarefa de atualização de modelos. Em resumo, existem três passos essenciais para um processo de atualização de modelos de um projeto Django:

- Alterar (atualizar) os modelos (classes Python) no ficheiro-fonte (Python) models.py;
- Executar o comando makemigrations (python manage.py makemigrations) para criar as alterações (migrations);
- Executar o comando migrate (python manage.py migrate) para efetuar as alterações no esquema da base de dados.

7.4. Django API

O ambiente Django incorpora uma API na forma de uma shell Python interativa (Figura 13). Para executar a shell especificamos python manage.py shell no diretório do projeto Django. A próxima figura (Figura 13) ilustra um conjunto de comandos simples para o acesso à estrutura dos modelos criados, assim como para a inserção e manipulação de dados. No início, uma aplicação Django não contém dados associados a objetos ou instâncias das classes Python. Por exemplo, a instrução seguinte especificada na shell (Question.objects.all()) devolve uma lista vazia <QuerySet []> (Figura 13), a qual representa que a classe (e referente tabela de base de dados) Question não contém dados associados.

O próximo exemplo (Figura 13) apresenta um conjunto simples de instruções para definir (e gravar através do método save()) novas instâncias de dados. Naturalmente que o acesso e atualização dos dados será posteriormente efetuado através do navegador web. Contudo, esta shell é útil para efetuar testes ao software e ao respetivo desenvolvimento da aplicação web.



```
Administrator: Windows PowerShell
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py shell
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct 1 2024, 03:06:41) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Question, Choice
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> question = Question(question_text="Uma primeira questão?", pub_date=timezone.now())
>>> question.save()
>>> question.id
1
>>> question.question_text
'Uma primeira questão?'
>>> question.pub_date
datetime.datetime(2024, 11, 17, 11, 7, 13, 639517, tzinfo=datetime.timezone.utc)
>>> question.question_text = "Nova questão?"
>>> question.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
>>>
```

Figura 13. Django API e inserção de dados através da shell.

O programa seguinte (Figura 14) efetua um conjunto de alterações aos modelos da aplicação poll anteriormente definida. Neste sentido, vamos especificar as alterações no ficheiro Python `models.py` que representam a estrutura de dados da aplicação web em desenvolvimento. As alterações efetuadas incluem dois métodos `__str__(self)` para devolver o conteúdo (valor) dos atributos `question_text` da classe `Question` e `choice_text` da classe `Choice`. Adicionalmente, foi também incluído um novo método na classe `Choice`: `was_published_recently(self)`. Este método simplesmente indica se a data de publicação do comentário é (true) ou não (false) recente, tendo em conta um intervalo de tempo de 1 dia (`date.time.timedelta(days=1)`, Figura 14). Este método devolve um valor booleano (true ou false).

Após gravar as alterações ao modelo, podemos iniciar a shell para importar os modelos e verificar as alterações efetuadas. As alterações anteriores aos modelos (classes Python) `Question` e `Choice` podem ser posteriormente testadas através da shell API Django (Figura 15).

7.5. Administração e Desenvolvimento Web

O ambiente Django disponibiliza uma interface web para a administração e gestão de conteúdos. O perfil de administração é suposto ser gerido pelo responsável (administrador do ambiente Django) do design e desenvolvimento dos projetos e aplicações web. Por outras palavras, o componente de administração Django não será gerido pelos utilizadores (visitantes) das aplicações web.


```
polls > models.py > ...
1  import datetime
2  from django.db import models
3  from django.utils import timezone
4
5
6  class Question(models.Model):
7      question_text = models.CharField(max_length=200)
8      pub_date = models.DateTimeField("date published")
9
10     def __str__(self):
11         return self.question_text
12
13     def was_published_recently(self):
14         return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
15
16
17     class Choice(models.Model):
18         question = models.ForeignKey(Question, on_delete=models.CASCADE)
19         choice_text = models.CharField(max_length=200)
20         votes = models.IntegerField(default=0)
21
22     def __str__(self):
23         return self.choice_text
24
```

Figura 14. Ficheiro Python `models.py` da aplicação poll (alteração das classes).

```
Administrator: Windows Powr
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py shell
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct 1 2024, 03:06:41) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Question, Choice
>>> Question.objects.all()
<QuerySet [<Question: Nova questão?>]>
>>> Question.objects.filter(id=1)
<QuerySet [<Question: Nova questão?>]>
>>> q=Question.objects.get(pk=1)
>>> q.was_published_recently()
True
>>> q.choice_set.all()
<QuerySet []>
>>> q.choice_set.create(choice_text='Primeira escolha', votes=0)
<Choice: Primeira escolha>
>>> q.choice_set.create(choice_text='Segunda escolha', votes=0)
<Choice: Segunda escolha>
>>> c = q.choice_set.create(choice_text='Terceira escolha', votes=0)
>>> c.question
<Question: Nova questão?>
>>>
```

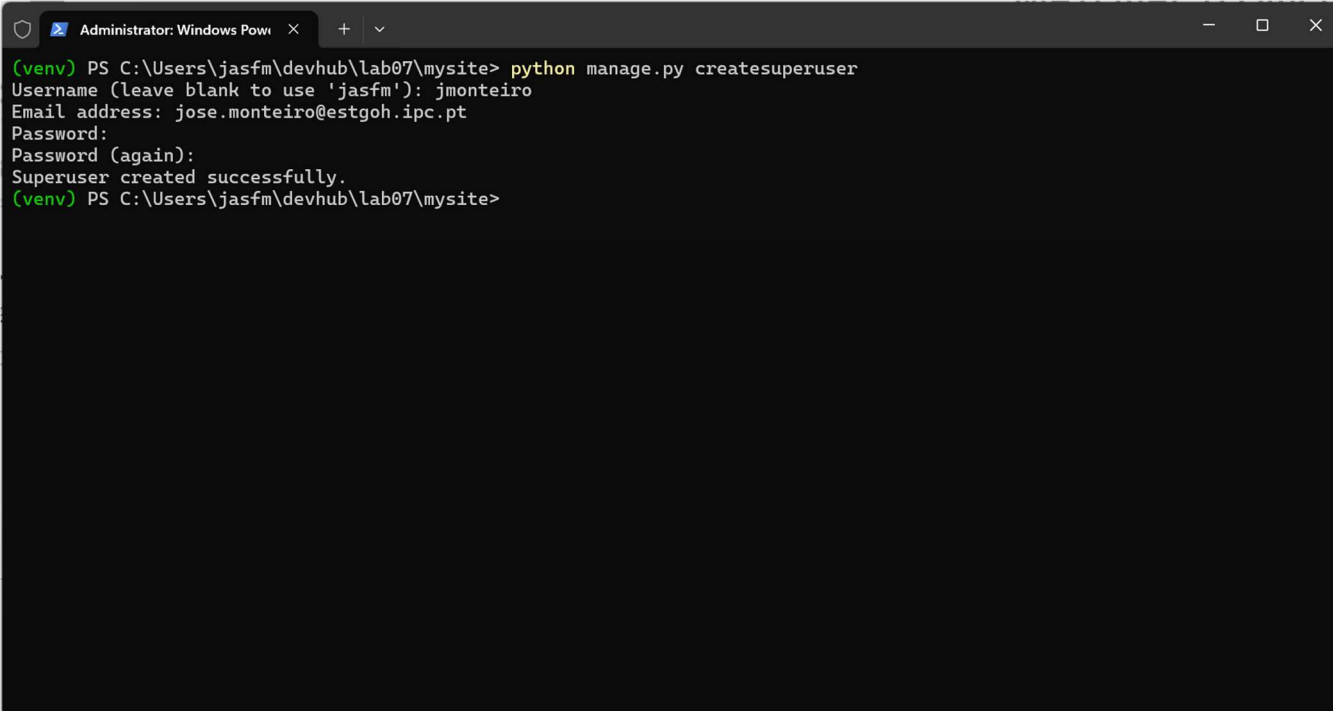
Figura 15. Inserção de dados através da shell.

7.5.1. Criação da Conta de Administrador

Criar uma conta de administrador (admin) é um processo simples. Para efetuar esta operação, devemos executar a instrução `createsuperuser` através da seguinte expressão de comando no diretório do projeto Django (Figura 16):

```
python manage.py createsuperuser
```

Neste exemplo (projeto Django) foi criado um utilizador com perfil de administrador que possui os seguintes dados (nome do utilizador, endereço de correio eletrónico, palavra-passe e confirmação).



```
(venv) PS C:\Users\jasfm\devhub\lab07\mysite> python manage.py createsuperuser
Username (leave blank to use 'jasfm'): jmonteiro
Email address: jose.monteiro@estgoh.ipc.pt
Password:
Password (again):
Superuser created successfully.
(venv) PS C:\Users\jasfm\devhub\lab07\mysite>
```

Figura 16. Criação de uma conta de administração.

Após a criação de uma conta de administrador, estamos em condições de iniciar o servidor web de desenvolvimento e explorar o website Django para gestão e administração de aplicações web.

7.5.2. Servidor Web de Desenvolvimento

O website Django de administração pode ser utilizado após o início do servidor de desenvolvimento. Conforme vimos anteriormente, de modo a iniciar e utilizar o servidor web de desenvolvimento incorporado na framework web, somente necessitamos de executar a instrução (expressão): `python manage.py runserver`. O website Django de administração (Django admin website) está ativado por defeito. As restantes aplicações (Django apps) necessitam de ser ativadas, de modo a tornarem-se visíveis e funcionais do lado do cliente (navegador web).

Através do servidor de desenvolvimento, é possível abrir o navegador web na página do domínio (hostname) local e especificar no final do endereço `/admin/`, de forma a aceder à página principal do website de administração, conforme o próximo endereço: `http://127.0.0.1:8000/admin/`

A página de login de administração é então apresentada (Figura 17). Para aceder ao website de administração, somente necessitamos de inserir as credenciais anteriormente definidas (nome do utilizador e palavra-passe). Conforme vamos ver posteriormente, podemos criar diferentes contas de utilizador Django e associar diferentes perfis de utilização, as quais poderão corresponder a diferentes permissões de leitura e atualização de dados das aplicações web desenvolvidas.

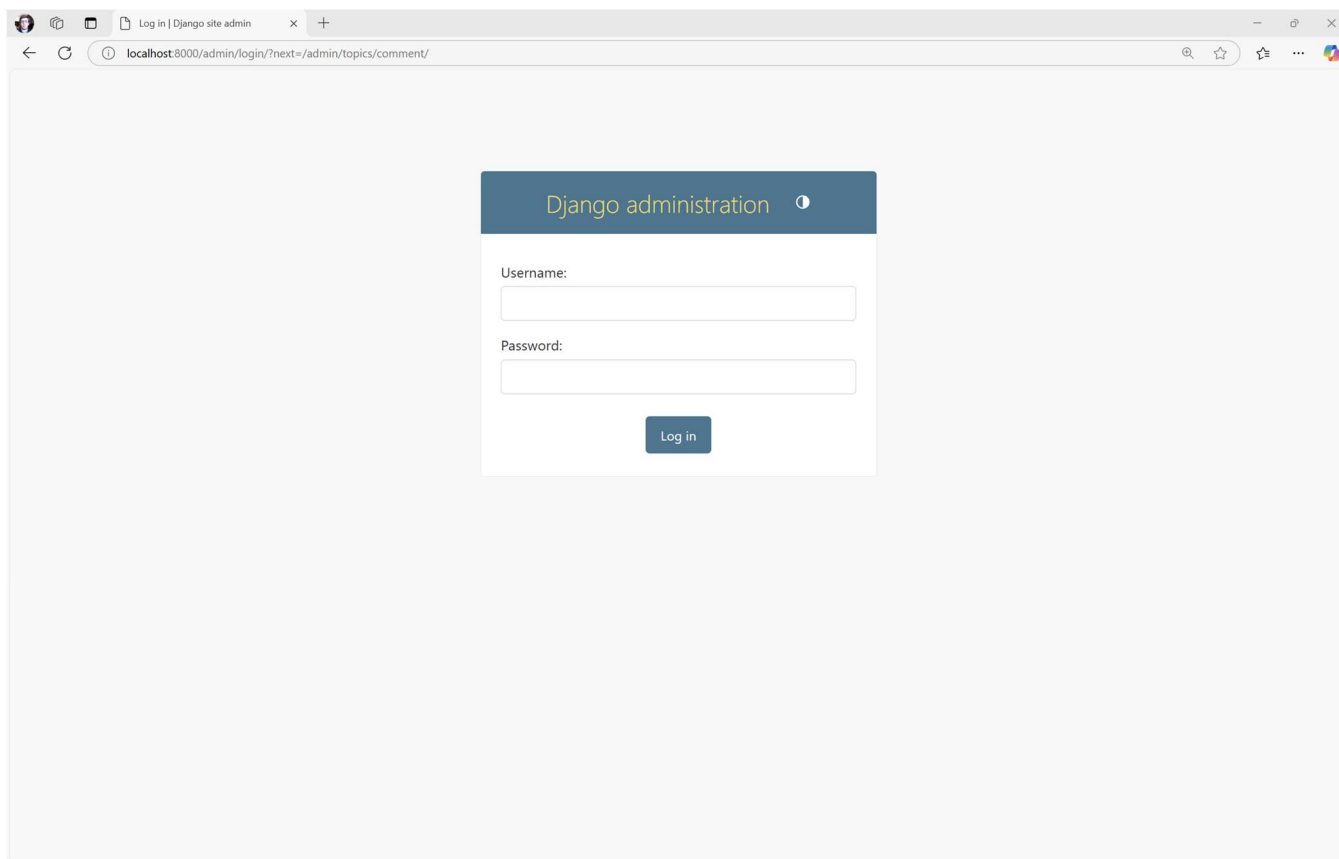


Figura 17. Criação de uma conta de administração.

Depois de inserir as credenciais de administrador previamente definidas, temos acesso à página principal web Django de administração (Figura 18), a qual permite efetuar um conjunto de operações e parametrizações para o desenvolvimento de aplicações web. Inicialmente estão visíveis e editáveis as secções Groups e Users.

7.5.3. Gestão de Aplicações (Site Admin)

Aplicações Django previamente criadas podem ser ativadas no ambiente de administração. Para isso, é necessário editar o ficheiro Python `admin.py` no diretório da respetiva aplicação (Figura 19). Desta forma simples é possível criar e desenvolver diferentes modelos de aplicações e efetuar testes através do navegador web. A ativação de uma determinada aplicação web é realizada através da importação (import codification) dos respetivos modelos previamente definidos.

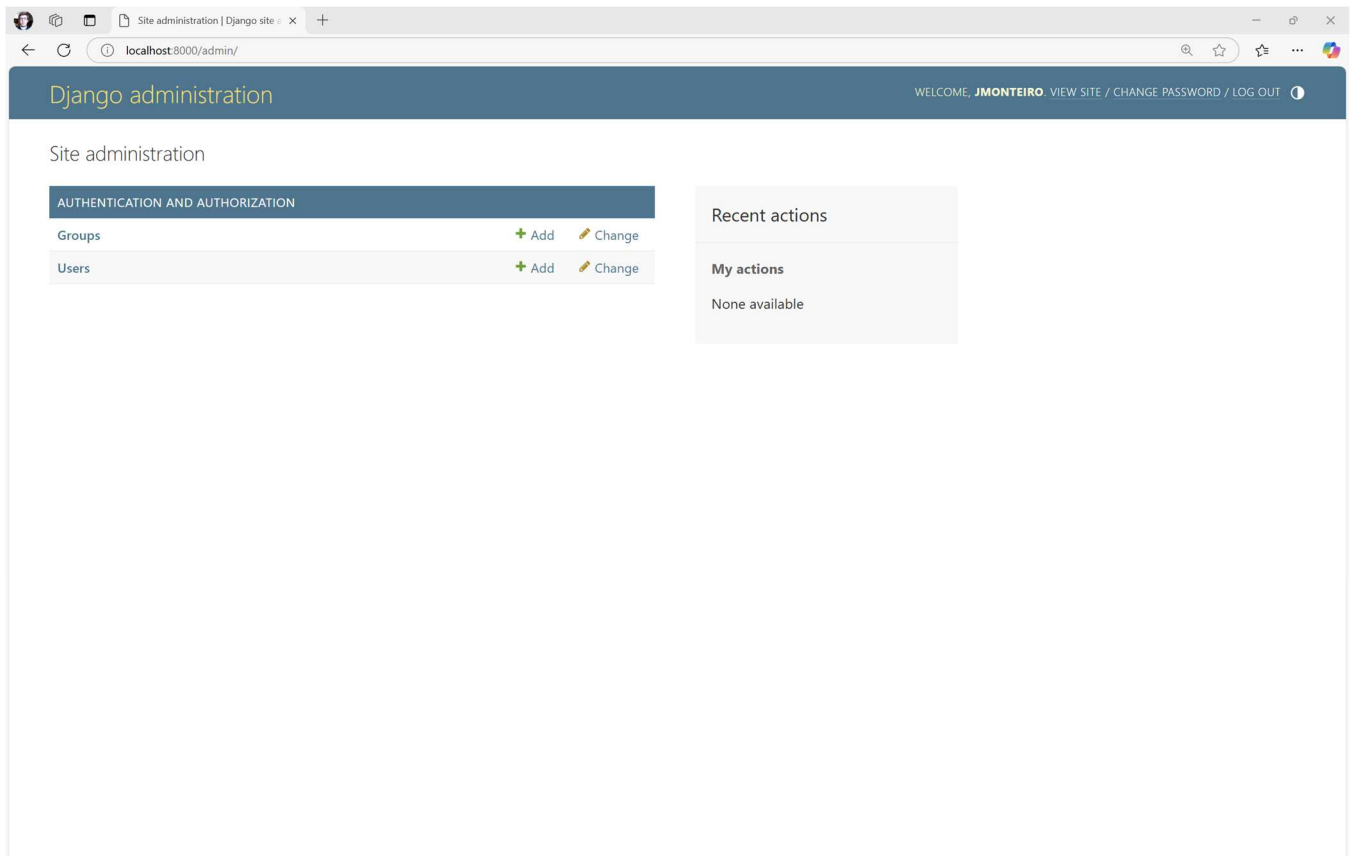


Figura 18. Django website administration.

```
polls > admin.py
1  from django.contrib import admin
2  from .models import Question
3
4
5  admin.site.register(Question)
6
```

Figura 19. Ativação de aplicações Django.

O comando `import` referencia e incorpora os modelos criados para a aplicação web. A expressão `admin.site.register(Model, ...)` efetua o registo e a ativação efetiva dos modelos e, consequentemente, a visibilidade e disponibilização das funcionalidades web (Figuras 20 e 21).

Depois de gravar as alterações referentes à ativação dos modelos, neste caso o modelo `Question`, uma nova secção surge no website, e as páginas web subjacentes com a representação das estruturas de dados anteriormente especificadas nos modelos da aplicação (Figura 21).

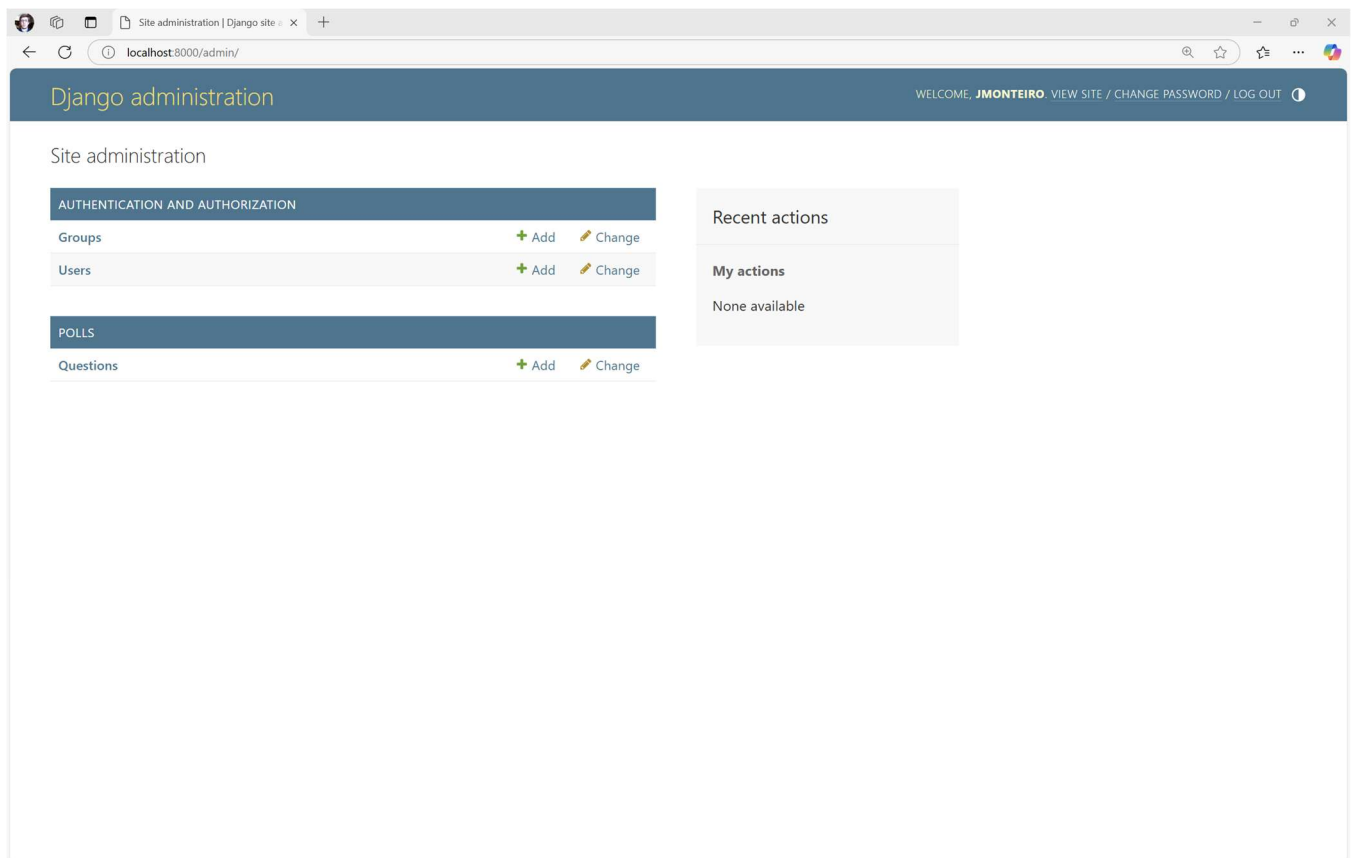


Figura 20. Ativação da aplicação Django polls.

Conforme foi especificado nos modelos (`models.py`) da aplicação polls, ao selecionar Questions temos acesso à página para editar as questões e os respetivos comentários (choices). Os conteúdos foram inicialmente inseridos através da shell Django anteriormente (Secção 7.4) apresentada. Depois de ativarmos os modelos e a aplicação, o ambiente Django disponibiliza um conjunto de funcionalidades base para a consulta e edição (inserção, alteração e eliminação) de dados. O ambiente incorpora um conjunto significativo e coerente de funcionalidades web para facilitar a manipulação dos dados e a respetiva interação com o utilizador, incluindo os tradicionais controlos de interface web.

Com base no modelo (`models.py`) criado e representado através de um conjunto de classes Python, o ambiente Django assegura as seguintes funcionalidades e automatismos para o desenvolvimento de aplicações web:

- Um formulário (form) web automaticamente gerado com base num modelo definido. Exemplo: Question model (Figura 21);
- Para cada web form, os diferentes tipos de dados especificados no modelo, tais como `DateTimeField`, `CharField`, e `IntegerField`, são formatados (e validados) do lado do cliente (navegador), em HTML de acordo com os tipos anteriormente definidos (`models.py`);
- O tipo de dados `DateTimeField` tem associadas uma formatação e validação com base na linguagem JavaScript. Deste modo, os conteúdos do tipo data (date) têm associada uma interface simples com o utilizador, incluindo a possibilidade de inserir a data (atual) do sistema (Now, Figura 22);

- No cabeçalho de um formulário web temos um conjunto de indicadores (Home / App name / Object, neste exemplo Home > Polls > Questions, Figura 22) de navegação, de modo a facilitar a leitura, a compreensão e a interface web. Também na zona do cabeçalho encontramos uma opção (History) para verificar o histórico das operações efetuadas;
- O rodapé do formulário (Figura 22) apresenta um conjunto útil de opções de gravação: “Save and add another” para gravar na base de dados as alterações efetuadas e, posteriormente, abrir outra página web para inserir novos dados sobre o mesmo objeto; “Save and continue editing” para gravar e continuar a editar os dados do mesmo objeto; e a opção mais simples “Save”, simplesmente para gravar e voltar à página do objeto referente aos dados em análise;
- Opção “Delete” para eliminar dados.

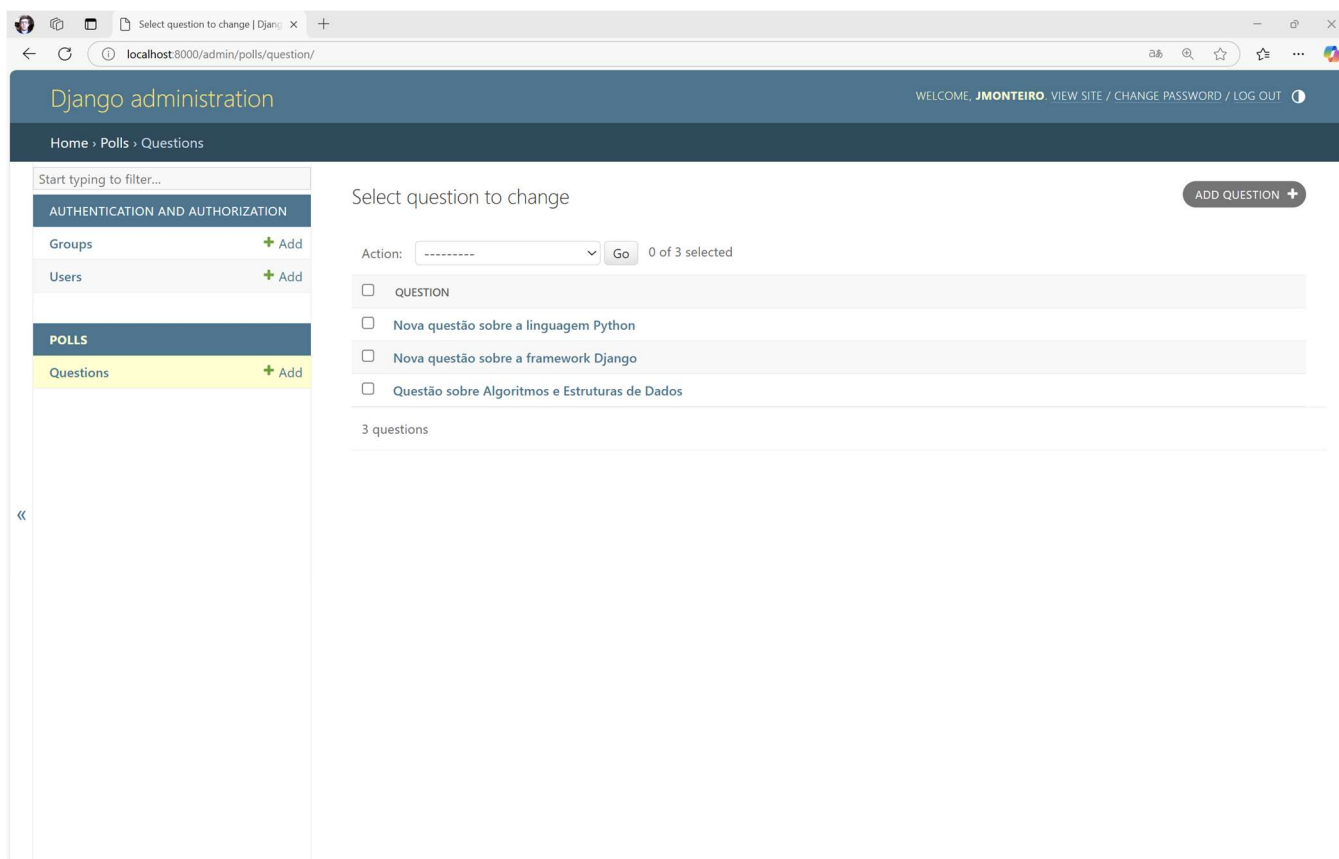


Figura 21. Aplicação Django polls.

No âmbito do processo de edição de dados, ainda no rodapé de um formulário web, temos a opção “Delete” para eliminar os dados do objeto atual e itens (dados) relacionados. Esta opção incorpora uma página web (ver exemplo na Figura 23) de confirmação da respetiva eliminação.

Conforme aferimos com os exemplos de páginas e formulários associados à aplicação polls, o ambiente Django incorpora um conjunto significativo de interfaces web que facilitam o processo de desenvolvimento de aplicações web.

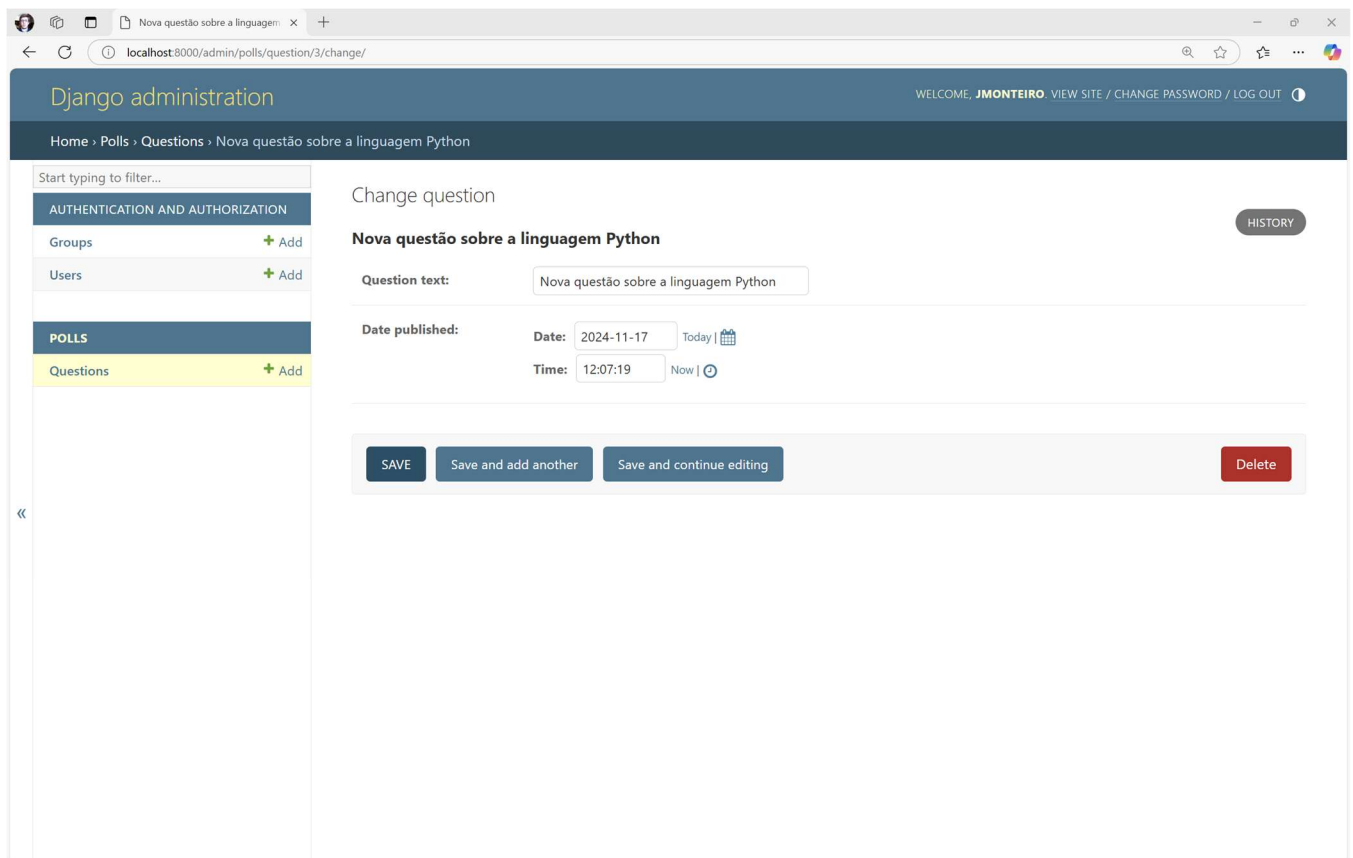


Figura 22. Interface web.

7.5.4. Desenvolvimento de Aplicação Web

O ambiente Django através de website de administração (site admin) permite, de uma forma simples, a alteração e atualização (personalização) de uma aplicação web, e o consequente desenvolvimento de novas funcionalidades. O processo de atualização e personalização do modelo e respetiva aplicação anteriormente definida tem por base a documentação principal deste ambiente de desenvolvimento de software para a web.

No momento em que se efetua o registo (por exemplo, `admin.site.register(Question)`) de um modelo, o Django automatiza um conjunto de procedimento para a definição web das respetivas páginas e formulários para a manipulação (edição) de dados. Posteriormente, é possível alterar e configurar a interface web das páginas e formulários. O código (exemplo) seguinte (Figura 24) ilustra a forma simples para reordenar a apresentação dos campos de um formulário web. No ficheiro Python `admin.py` da aplicação `polls`, deve efetuar as seguintes alterações de modo a alterar o registo do modelo `Question`.

Esta simples atualização do código-fonte permitiu criar um novo objetivo (`QuestionAdmin`) que após passar por argumento em `admin.site.register()`, define que o campo “Date Published” (`pub_date`) surge primeiro que o campo “Question” (`question_text`). As restantes alterações (Figura 25) refletem determinadas opções e utilização de classes Django, como é o caso da classe `collapse`, que simplesmente permite a visualização ou ocultação dos campos e respetivos dados de uma página web. Esta forma simples de interface é interessante quando temos formulários web com um conjunto de campos opcionais que normalmente são pouco utilizados.

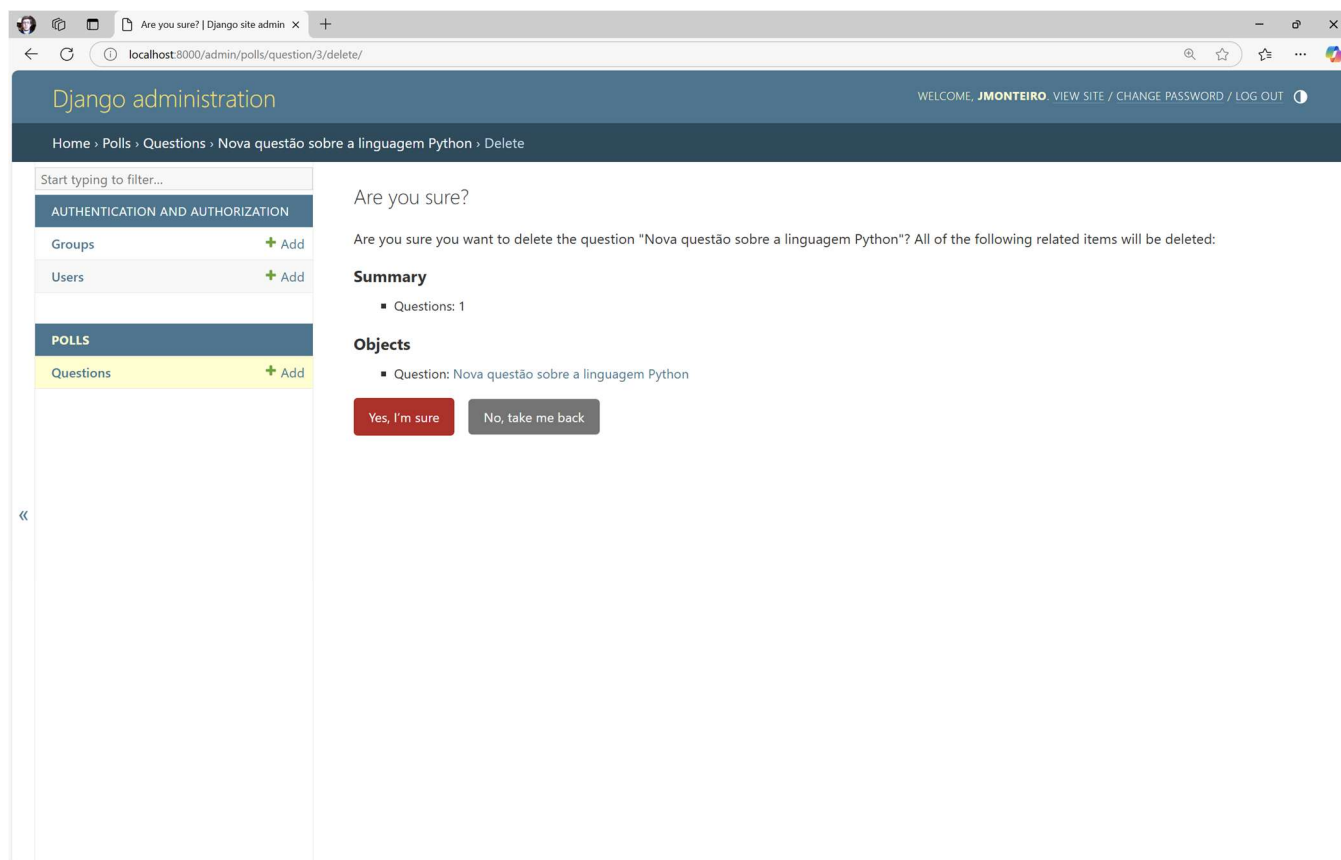


Figura 23. Confirmação da eliminação dos dados de um objeto.

```
polls > admin.py > ...
1  from django.contrib import admin
2  from .models import Question
3
4
5  class QuestionAdmin(admin.ModelAdmin):
6      fields = ['pub_date', 'question_text']
7
8
9  admin.site.register(Question, QuestionAdmin)
10
```

Figura 24. Alterações aos modelos da aplicação polls.

As alterações efetuadas implementam uma funcionalidade simples que permite definir opções múltiplas (multiple choices) para cada questão (question). Primeiro é necessário referenciar e, posteriormente, registar o modelo Choice, da mesma forma que foi registado o modelo Question (`from .models import Choice, Question`). A classe `QuestionAdmin` permite reconfigurar o modelo Question e a classe `ChoiceInline` permite definir múltiplas opções para cada questão, conforme podemos verificar na Figura 26.


```
polls > admin.py > ...
1  from django.contrib import admin
2  from .models import Choice, Question
3
4
5  class ChoiceInline(admin.StackedInline):
6      model = Choice
7      extra = 3
8
9
10 class QuestionAdmin(admin.ModelAdmin):
11     fieldsets = [
12         (None, {'fields': ['question_text']}),
13         ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14     ]
15     inlines = [ChoiceInline]
16
17
18 admin.site.register(Question, QuestionAdmin)
19
```

Figura 25. Alterações aos modelos da aplicação polls.

The screenshot shows the Django administration interface for the 'polls' application. The browser address bar indicates the URL is `localhost:8000/admin/polls/question/3/change/`. The page title is 'Django administration' and the user is logged in as 'JMONTEIRO'. The breadcrumb trail is 'Home > Polls > Questions > Nova questão sobre a linguagem Python'. The left sidebar shows the 'POLL'S' section with 'Questions' highlighted. The main content area is titled 'Change question' and 'Nova questão sobre a linguagem Python'. It features a 'Question text' field with the value 'Linguagem Python e Web Frameworks'. Below this is a 'Date Information' section. The 'CHOICES' section lists three choices: 'Choice: #1' with text 'Django' and 1500 votes, 'Choice: #2' with text 'Flask' and 475 votes, and 'Choice: #3' with text 'Web2Py' and 130 votes. There is an 'Add another Choice' button at the bottom.

Figura 26. Aplicação polls após as alterações no ficheiro `admin.py`.

As alterações aos modelos (`models.py`) representam e refletem as alterações ao modelo de dados Django e às respetivas tabelas da base de dados relacional. O Django contém uma ferramenta que traduz (mapeia) as classes Python nas correspondentes tabelas de uma base de dados relacional. Este componente Django (ferramenta) de software é designado por Django Object Relational Mapper (ORM).

No caso de guardar (através do botão “Save”) as alterações efetuadas, a próxima figura (Figura 27) ilustra a página web subsequente, a qual apresenta no início (cabeçalho) uma referência à operação de alteração, anteriormente realizada no formulário. O botão “History”, quando clicado, apresenta uma lista ordenada cronologicamente com as últimas alterações efetuadas aos dados da questão selecionada.

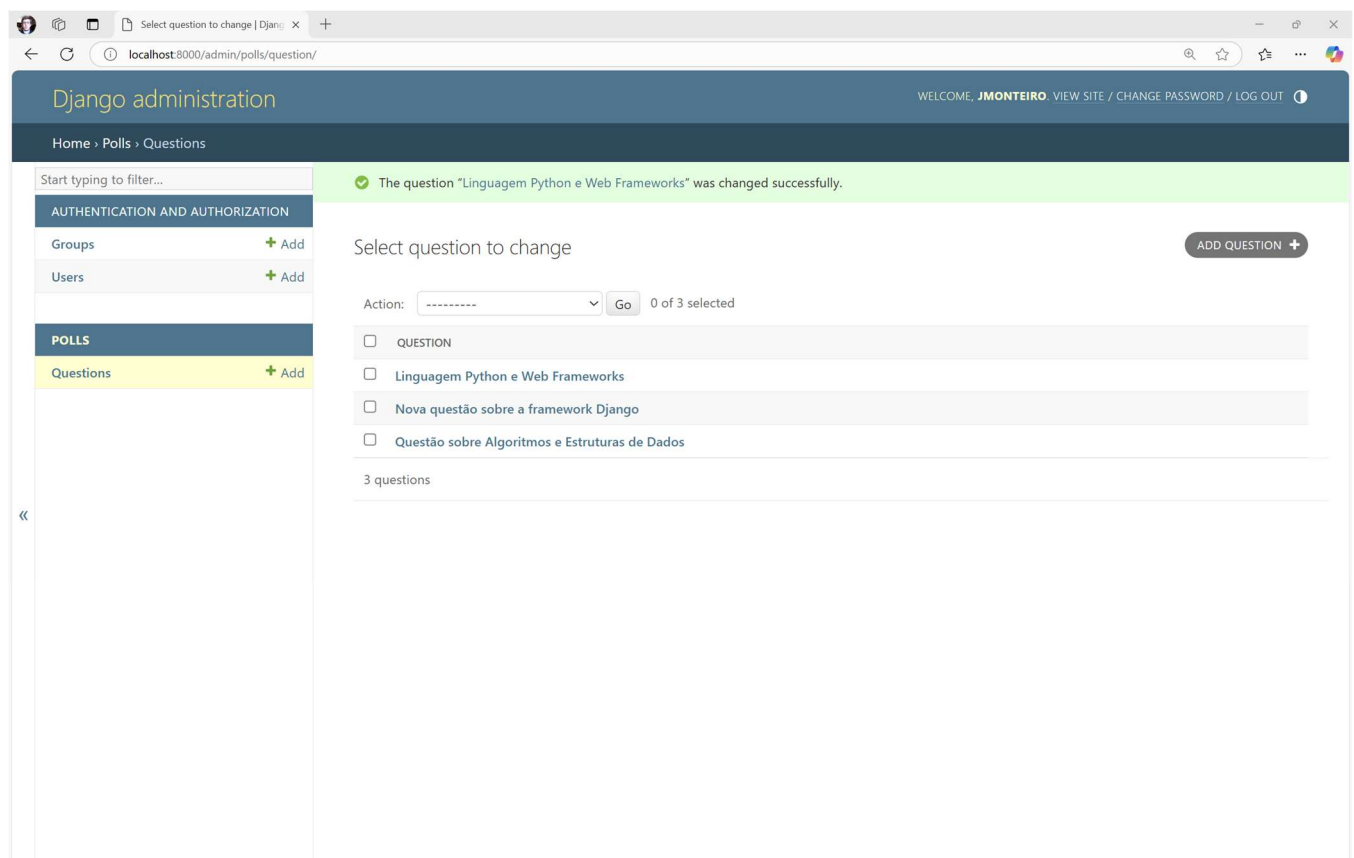


Figura 27. Registo de uma alteração (question).

7.5.5. Django Views

As visões (views) podem representar uma parte substancial de uma aplicação Django. Uma visão especificada através de uma função (código) Python para associar um ou mais URL e obter como resposta objetos HTTP (Figura 28). Cada visão é representada por uma função Python, ou método, no caso das visões baseadas em classes (class-based views). O ambiente Django aplica uma visão através de um URL e respetivo pedido (request) HTTP. Os URL especificam o ponto de entrada no website e as visões Django representam o código-fonte para responder (response) aos eventos solicitados (requests).

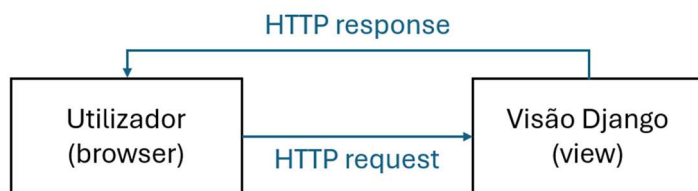


Figura 28. Visão (view) Django.

O ambiente Django analisa um URL após o nome do domínio (domain name). Neste sentido, é uma boa prática definir padrões para a especificação de URL, por exemplo para uma aplicação do tipo blog: `http://<dominio>/<ano>/<mes>/`. Para converter um URL numa visão, o Django aplica `URLconfs` para mapear os padrões URL em visões e respetivas páginas web.

Uma visão (view) em Django representa uma página web para implementar uma função específica do domínio da aplicação. Por exemplo, uma aplicação web do tipo blog normalmente tem associadas as seguintes visões:

- Página inicial do blog com as entradas mais recentes;
- Página web com o detalhe (interface) para o registo de novas entradas;
- Páginas de resumo (anual, mensal e diário) das entradas;
- Área de discussão e comentários a entradas anteriores.

Para definir (especificar) visões temos o ficheiro Python `views.py` incluído no diretório da aplicação (Figura 29). O próximo exemplo especifica uma visão referente a uma página inicial (index) da aplicação anterior polls. Em síntese, para definir uma visão (view) em Django, é necessário efetuar os seguintes procedimentos:

- Abrir o ficheiro Python `views.py` e especificar uma função (Figura 30) que define uma nova visão (view) e consequente página web;
- Definir `URLconf` no diretório da aplicação (polls) através da criação de um novo ficheiro Python `urls.py` (Figura 31);
- Abrir o ficheiro Python `urls.py` do diretório de projeto e atualizar a variável `urlpatterns` para a definição dos padrões de URL e incluir (include) um novo padrão, neste caso para definir a página de raiz (root).

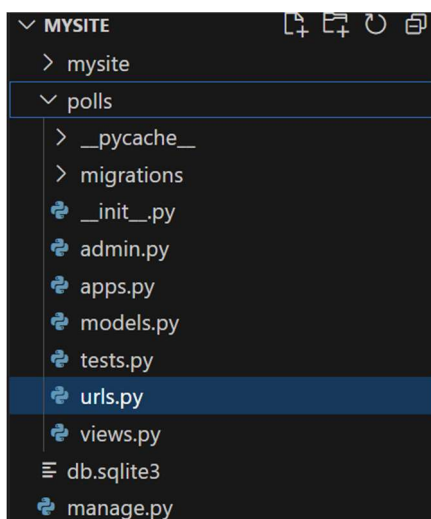


Figura 29. Ficheiros Python do diretório de uma aplicação Django.

```
polls > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4
5  def index(request):
6      return HttpResponse("Polls Django app Index: -- <b>Programação Web em Python</b> --")
7
```

Figura 30. Definição de uma visão (view) Django no ficheiro views.py.

```
polls > urls.py > ...
1  from django.urls import path
2  from . import views
3
4
5  urlpatterns = [
6      path("", views.index, name="index"),
7  ]
8
```

Figura 31. Ficheiro Python urls.py para a definição de padrões URL.

O passo seguinte permite definir a raiz (root) de URLconf no módulo polls.urls. Neste ficheiro urls.py referente ao projeto Django inicial (mysite), é necessário especificar os padrões dos URL (variável urlpatterns) das aplicações (Figura 32).

```
17  from django.contrib import admin
18  from django.urls import path, include
19
20  urlpatterns = [
21      path('admin/', admin.site.urls),
22      path('polls/', include('polls.urls'))
23  ]
24
```

Figura 32. Ficheiro Python urls.py no diretório do projeto.

Para verificar a visão definida e respetiva página web, abrimos o navegador na raiz da aplicação polls (Figura 33). Este resultado (texto) foi especificado anteriormente na função index(request) no ficheiro views.py (ver Figura 30).

O próximo exemplo apresenta uma visão Django para a criação de uma lista de questões registadas na aplicação polls. Esta visão apresenta uma lista de questões, separadas com vírgulas, e ordenada por data de publicação. A visão está especificada (Figura 34) através da função Python index() para a raiz (do URL) da aplicação, alterando desta forma a versão anterior já descrita (Figura 30).

O resultado desta visão (função) no navegador (Figura 35) apresenta cinco [:5] questões ordenadas por data de publicação, ou seja, as últimas cinco questões inseridas na base de dados da aplicação Django polls.

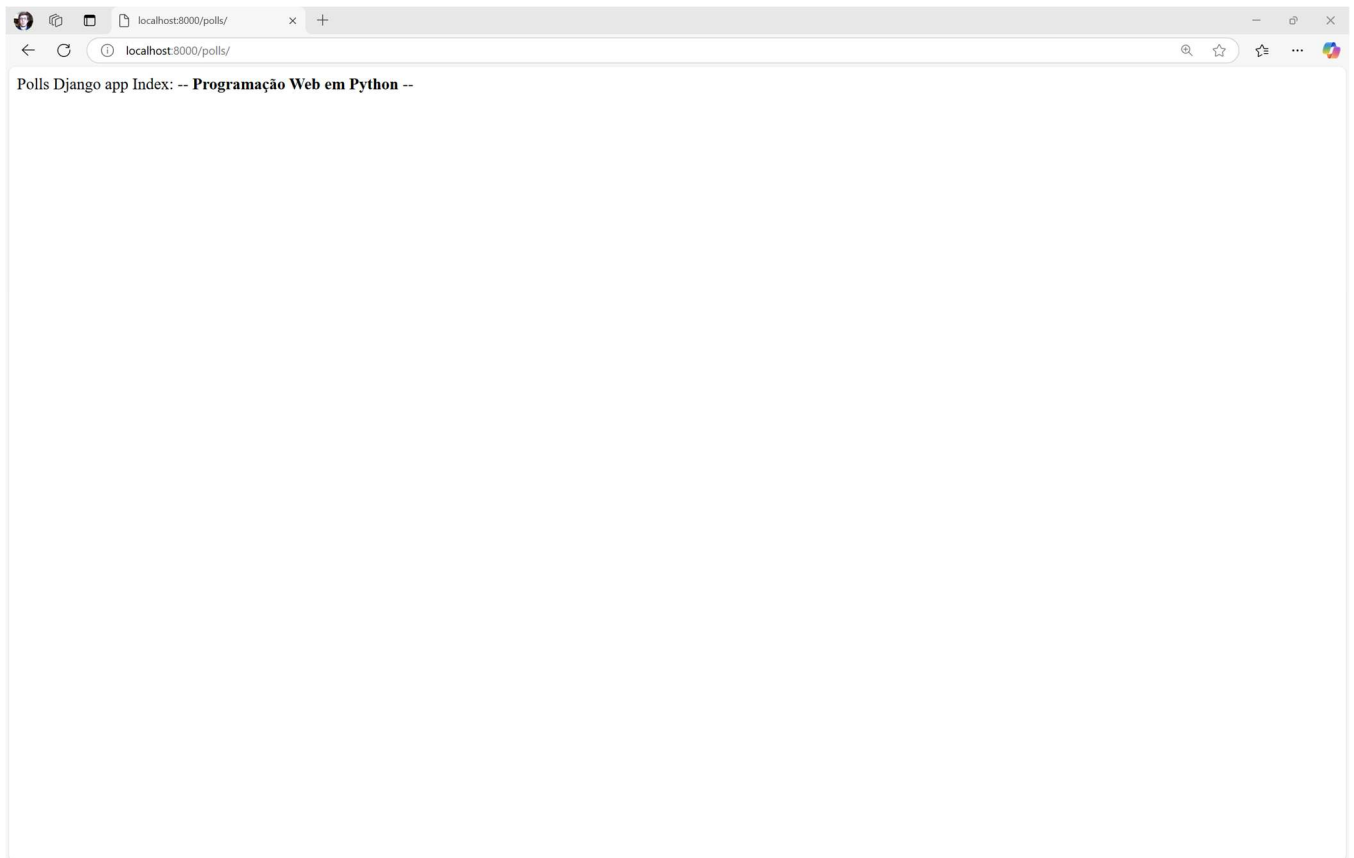


Figura 33. Resultado da visão Django no navegador.

```
polls > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponse
3  from .models import Question
4
5
6  def index(request):
7      latest_question_list = Question.objects.order_by('-pub_date')[:5]
8      output = ', '.join([p.question_text for p in latest_question_list])
9      return HttpResponse(output)
10
```

Figura 34. Visão Django especificada na função `index()`.

7.5.6. Django Templates

Um template Django é representado através de um simples ficheiro de texto, o qual poderá gerar um formato de uma linguagem de marcação, tais como HTML e XML. Um template contém um conjunto de variáveis subjacentes a um sistema de representação (Django template system). Este sistema permite separar o componente de design da aplicação do componente lógico de programação em Python. Os templates são normalmente associados às visões (views), de modo a definir o design e a apresentação dos dados de uma determinada visão.

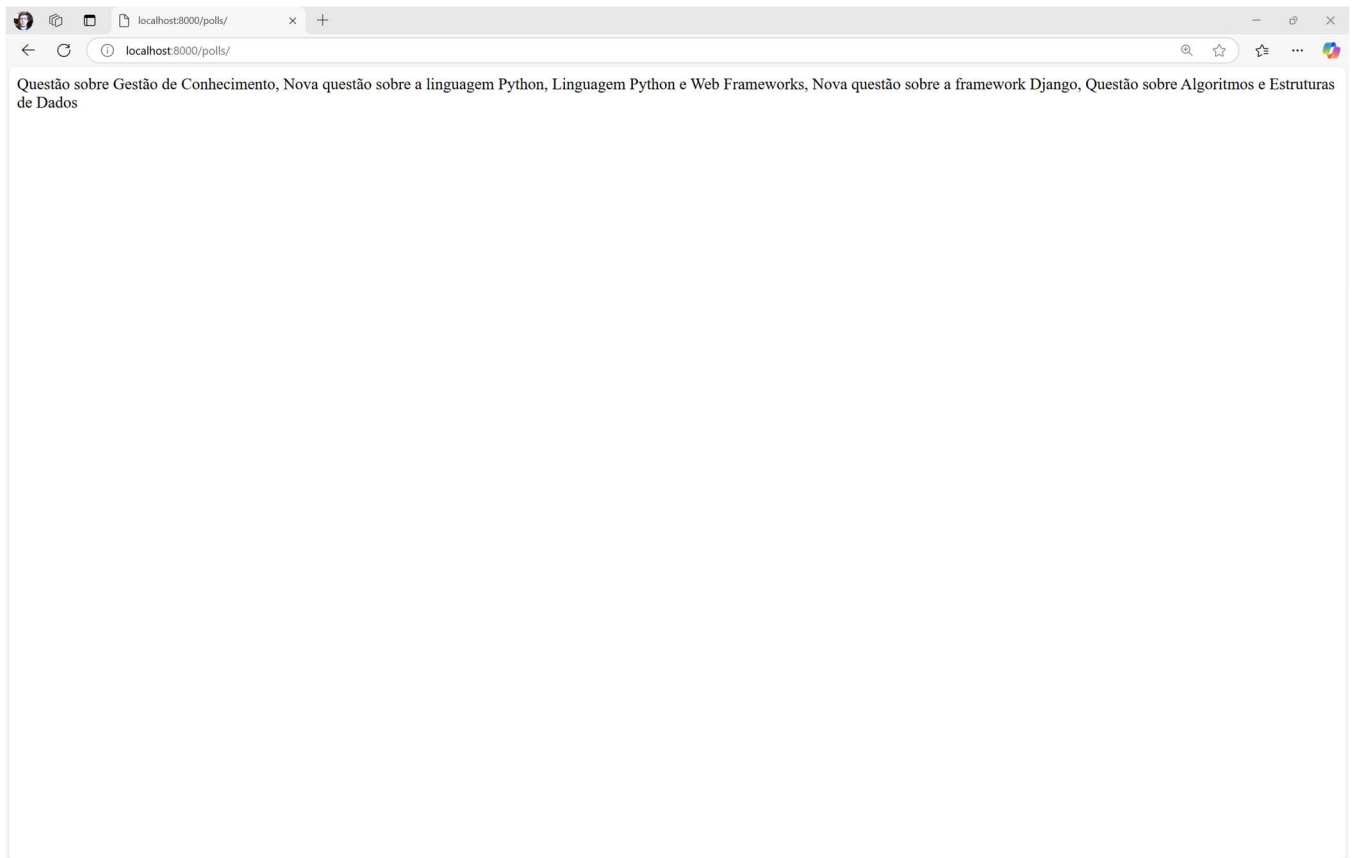


Figura 35. Resultado da visão no navegador.

As variáveis são representadas com a seguinte sintaxe (com um bloco de uma chavetas consecutivas) e os atributos das variáveis são especificados com um ponto (.):

```
{{ variable }}  
{{ variable.attribute }}  
Exemplo (Figura 36): {{ question.question.text }}
```

Quando um template (engine) encontra uma variável, efetua uma validação e substitui-a com um valor. Os nomes das variáveis consistem numa combinação de caracteres alfanuméricos, incluindo o underscore (_). O ponto (dot .) tem um significado especial, de modo a aceder aos atributos da respetiva variável. Não é possível definir variáveis com espaços ou outros sinais de pontuação.

Uma aplicação Django poderá ter um diretório para organizar templates. Desta forma, o ambiente irá procurar e utilizar os templates a partir deste diretório. É possível colocar todos os templates para as diferentes aplicações web juntos através de um único diretório. Contudo, é uma boa prática criar um diretório de templates para cada aplicação Django. Neste sentido, vamos criar um diretório `\templates` associado à aplicação `polls` previamente definida (`\polls\templates`). De modo a utilizar o ficheiro `index.html` na raiz da aplicação `polls`, vamos criar mais um diretório da aplicação `polls`, neste caso no interior do diretório `templates`: `\polls\templates\polls\index.html`.

O código (exemplo) seguinte apresenta (Figura 36) um template Django na forma de um ficheiro HTML. Conforme acontece com outras linguagens do lado do cliente, podemos integrar tags HTML, de modo a formatar a páginas web e visão (Django view) subjacente.

```
polls > templates > polls > <> index.html
1  {% if latest_question_list %}
2      <ul>
3          {% for question in latest_question_list %}
4              <li><a href="/polls/{{ question.id }}">{{ question.question_text }}</li>
5          {% endfor %}
6      </ul>
7  {% else %}
8      <p>No polls are available.</p>
9  {% endif %}
10
```

Figura 36. Template (exemplo) index.html.

As instruções da linguagem de templates são referenciadas com os símbolos {% e %}. Esta sintaxe permite separar e segmentar as instruções da linguagem Django de template das instruções (tags) HTML. Este template cria uma lista (tag HTML) de questões na forma de ligações (links) através da tag HTML <a href ... (Figura 36). Esta abordagem de segmentação é comum noutras linguagens de programação para a web, como é o caso da linguagem Java Server Pages (JSP).

O código seguinte (Figura 37) tem por base a aplicação polls e referencia no ficheiro Python views.py o template anterior (index.html) e visão subjacente index. A visão index especificada através da função Python index(request) pode ser chamada (browser, Figura 38) na raiz da aplicação (http://.../polls/) e a visão detail representada na função detail(request, question_id) no endereço (exemplo): http://127.0.0.1:8000/polls/6/, no qual está indicado o código (número) de uma questão, neste caso a questão com código 5 (question_id=5).

```
polls > views.py > ...
1  from django.http import HttpResponse
2  from django.template import loader
3  from .models import Question
4
5
6  def index(request):
7      latest_question_list = Question.objects.order_by("-pub_date")[:5]
8      template = loader.get_template("polls/index.html")
9      context = {
10         "latest_question_list": latest_question_list
11     }
12     return HttpResponse(template.render(context, request))
13
14  def detail(request, question_id):
15     return HttpResponse("You're looking at question %s." % question_id)
16
```

Figura 37. Código Python (ficheiro views.py) para aplicar o template index.html.

As variáveis especificadas num template estão definidas nas funções Python que representam as visões (Django views). Neste exemplo temos a variável `latest_question_list` que define as últimas cinco `[:5]` questões (ver código da Figura 37) e a variável `question` e respetivos atributos `id` e `question_text`.

```
polls > urls.py > ...
1  from django.urls import path
2  from . import views
3
4
5  urlpatterns = [
6      path("", views.index, name="index"),
7      path("<int:question_id>/", views.detail, name="detail"),
8      path("<int:question_id>/results/", views.results, name="results"),
9      path("<int:question_id>/vote/", views.vote, name="vote"),
10 ]
11
```

Figura 38. Ficheiro Python `urls.py` com referência às visões `index` e `detail`.

O código anterior contém as alterações efetuadas ao ficheiro `urls.py` para a especificação dos padrões de URL (URL patterns) e respetiva associação das visões previamente definidas, incluindo as visões (views) `index` e `detail`, esta última ilustrada na Figura 40, depois de clicar na primeira questão (Figura 39), e última questão inserida na base de dados, dado que a lista de questões está ordenada por data de publicação.

A aplicação deste template (`index.html`) associado à visão `index` simplesmente alterou o formato de visualização dos dados, neste caso criando uma lista de valores, e definindo cada um dos valores com uma ligação (link) web. Anteriormente, sem template associado, a visão especificada através da função Python (Figura 34), apresentava os dados (questões) em modo de texto separados por vírgulas.

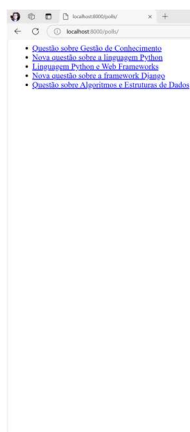


Figura 39. Resultado no navegador da aplicação do template (exemplo) `index.html`.

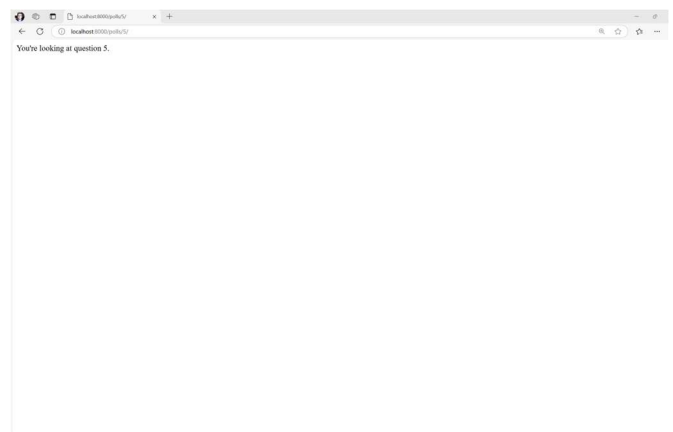


Figura 40. Resultado no navegador da visão `detail`.