

# Tecnologias e Programação Web

## A Plataforma Django



# Plataforma Django

Models

Django's Database Layer

# Model



- MTV – *Model, Template, View*
- Model
  - Consiste na camada de acesso a dados - “*Data Access Layer*”
  - Esta camada permite definir, em relação aos dados:
    - o Acesso;
    - a Validação;
    - o Comportamento;
    - as Relações entre os dados.

# DB Configuration



- A definição do acesso aos dados, começa pela configuração do acesso à base de dados.
- No ficheiro “settings.py”, procurar pela variável DATABASES e definir os parâmetros ENGINE, NAME, USER, PASSWORD, HOST, PORT.
  - Alguns destes parâmetros podem ser omitidos, conforme a DB a aceder.
- Por defeito, na criação do projeto é configurado o acesso à base de dados local SQLite. Para outras DBs, consultar a documentação:
  - <https://docs.djangoproject.com/en/3.1/topics/db/multi-db/>

# Criação de um modelo (i)



- No ficheiro “models.py” da pasta “app”, definir as classes do modelo de dados a usar.

```
models.py x
1 from django.db import models
2
3 class Author(models.Model):
4     name = models.CharField(max_length=70)
5     email = models.EmailField()
6     def __str__(self):
7         return self.name
8
9
10 class Publisher(models.Model):
11     name = models.CharField(max_length=70)
12     city = models.CharField(max_length=50)
13     country = models.CharField(max_length=50)
14     website = models.URLField()
15     def __str__(self):
16         return self.name
17
18
19 class Book(models.Model):
20     title = models.CharField(max_length=100)
21     date = models.DateField()
22     authors = models.ManyToManyField(Author)
23     publisher = models.ForeignKey(Publisher,
24                                   on_delete=models.CASCADE)
25     def __str__(self):
26         return self.title
```

# Criação de um modelo (ii)



- Para cada atributo das classes é instanciado um objeto tipo “Field” e/ou subtipo, como: CharField, DataField, etc.
  - Ver a documentação em:  
<https://docs.djangoproject.com/en/3.1/ref/models/#field-types>
- Alguns atributos, representam a criação de relações entre as classes, tendo como efeito a criação de colunas com chaves estrangeiras (1:1, 1:M, M:1) ou de tabelas de associação (M:N)
  - Ver a documentação em:  
<https://docs.djangoproject.com/en/3.1/topics/db/models/#relationships>

# Criação de um modelo (iii)



- Relações entre as classes:
- 1:M e M:1
  - Conseguída com um atributo da classe “models.ForeignKey”
  - Exemplo de 1:M e M:1
    - Publisher (1) : (M) Book ou Book (M) : (1) Publisher
    - O atributo é colocado na classe Book (M), aquela que representa muitos objetos para um.
- 1:1 (único)
  - Conseguída com um atributo da classe “models.OneToOne”
    - Exemplo: Book (1) : (1) Author
    - O atributo “deve” ser colocado na classe que mais “necessita” da outra

# Criação de um modelo (iv)



- M:N
  - Conseguída com um atributo da classe “models.ManyToManyField”
- Exemplo
  - Book (M) : (N) Author
  - O atributo “deve” ser colocado na classe que mais “necessita” da outra
    - No presente exemplo, o autor pode existir, por si, sem necessidade de ter livros, mas o livro necessita de um autor



# Criação de um modelo (v)



- A classe base “Model”, donde são derivadas todas as classes do modelo, possui todos os mecanismos necessários para interagir com a base de dados.
- Cada classe derivada é implementada na BD na forma de uma tabela e os seus atributos são implementados na forma de colunas (campos) da tabela.
- Exemplo da classe “Author”, que corresponde a:

```
CREATE TABLE “app_author” (  
    “id” integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    “name” varchar (70) NOT NULL,  
    “email” varchar (254) NOT NULL );
```

# Criação de um modelo (vi)



- Com vista à ativação do modelo, a aplicação web, “app”, deve ser incluída na variável `INSTALLED_APPS` do ficheiro “`settings.py`”, caso ainda não esteja.
- De seguida deve-se proceder à validação do modelo (sintaxe e lógica) e para isso executar o seguinte comando na consola:
  - `python manage.py check`
- Dentro da pasta “`app/migrations`”, caso existam, apagam-se todos os ficheiros, com exceção do “`__init__.py`” e executa-se os seguintes comandos, na consola:
  - `python manage.py makemigrations app` (produz código de migração)
  - `python manage.py sqlmigrate app 0001` (opcional: mostra o código SQL)
  - `python manage.py migrate` (produz as tabelas na BD)

# Gestão dos Dados (i)



- A plataforma Django possui um mecanismo que possibilita uma gestão muito facilitada de todos os dados pertencentes ao modelo de dados: o Django Admin Site
- A URL = <http://localhost:{port}/admin> dá acesso à área administrativa a qual permite, por defeito, gerir os utilizadores do site
- Nesta área, também é possível aceder e gerir os dados definidos no modelo

# Gestão dos Dados (ii)



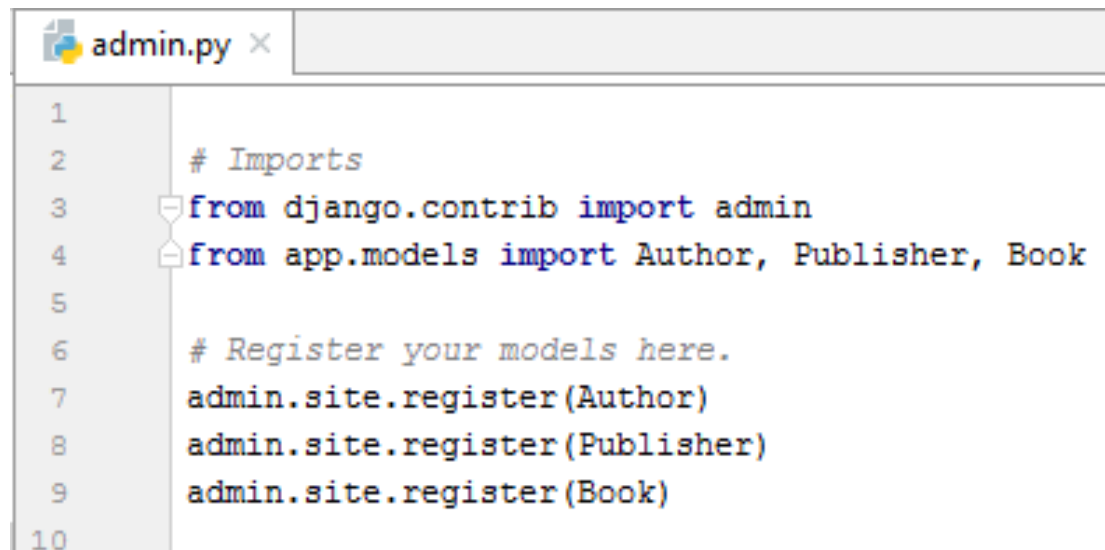
- Configuração
  - No ficheiro “urls.py”, adicionar o seguinte:

```
17
18 from django.urls import path
19 from django.contrib import admin
20
21 from app import views
22
23 urlpatterns = [
24     path('admin/', admin.site.urls),
25
```

# Gestão dos Dados (iii)



- Adicionar o modelo de dados ao Admin Site
  - No ficheiro “admin.py” na pasta “app”, registar as classes que se pretendem gerir



```
1
2  # Imports
3  from django.contrib import admin
4  from app.models import Author, Publisher, Book
5
6  # Register your models here.
7  admin.site.register(Author)
8  admin.site.register(Publisher)
9  admin.site.register(Book)
10
```

- Na página de administração, o modelo de dados torna-se acessível

# Gestão dos Dados (iv)



- Conta para aceder ao Django Admin Site:
  - Criar a conta de administração, com o comando:
    - `python manage.py createsuperuser`
- Testar o Django Admin Site
  - Executar o projeto e aceder ao link:
    - <http://localhost:{porto}/admin>

# Gestão dos Dados (v)



- Programando:

- Inserir um objeto

```
a = Author(name='Antonio Pedro',  
            email='apedro@email.com')  
a.save()
```

- Modificar um objeto

```
a.email = 'antonio.pedro@email.com'  
a.save()
```

- Selecionar todos os objetos

```
Autor.objects.all()
```

- Filtrar objetos (por nome)

```
Autor.objects.filter(name='Autor1')
```

# Gestão dos Dados (vi)



- Filtrar por nome e por email

```
Author.objects.filter(name='Autor1', email='...')
```

- Filtrar por nome parecido

```
Author.objects.filter(name_contains='Autor')
```

- Aceder a um único objeto

```
Author.objects.get(email='autor1@email.com')
```

- Ordenação

```
Publisher.objects.order_by("city", "country")
```

- Filtragem e Ordenação

```
Publisher.objects.filter(country='Portugal').order_by("-city")
```



# Gestão dos Dados (vii)



- Selecionar os primeiros resultados

```
Publisher.objects.order_by("city", "country")[0]
```

```
Publisher.objects.order_by("city", "country")[0:4]
```

- Não são permitidos índices negativos
- Remover um objeto  

```
Author.objects.get(email='autor1@email.com').delete()
```
- São possíveis muitas formas alternadas. Ver a documentação em:  
<https://docs.djangoproject.com/en/3.1/topics/db/queries/>