Models

Yuri Kaszubowski Lopes

Especialização em Tecnologia Python para Negócios

makemigrations e migrate

- Após alterar os modelos execute:

```
$ ./manage.py makemigrations
$ ./manage.py migrate
```

- O diretório `migrations` de cada app é atualizado automaticamente com `makemigrations`.
  - Ou seja, `makemigrations` converte os modelos de `models.py` para as migrações no diretório `migrations`
- `migrate` aplica as migrações no BD
  - Ou seja, aplica as migrações do diretório `migrations` no banco de dados
    - Comandos `CREATE TABLE`, `ALTER TABLE`, ...
- Assim, se você compartilha (inclui migrations no seu repositório, e.g., git), cada colaborador necessita executar apenas:

```
$ ./manage.py migrate
```

- Lembre-se que executamos apenas `migrate` após criar nosso projeto para iniciar o BD com as tabelas das aplicações padrões (e.g. contrib.auth)

Modelos

- Por enquanto temos (+ as alterações da atividade):

server/seriados/model.py

```
1  from django.db import models
2
3
4  class Serie(models.Model):
5      nome = models.CharField(max_length=70)
6
7      def __str__(self):
8          return self.nome
9
10
11 class Temporada(models.Model):
12     numero = models.IntegerField()
13     serie = models.ForeignKey(Serie, on_delete=models.CASCADE)
14
15
16 class Episodio(models.Model):
17     data = models.DateField()
18     titulo = models.CharField(max_length=200)
19     temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
20
21     def __str__(self):
22         return self.titulo
```

Campos (fields)

- O único requisito de um model é que ele tenha fields
- Tipos:
  - Dados (não relacionais):
    - ⋆ BooleanField, CharField, DateField, DateTimeField, DecimalField, DurationField, EmailField, FileField, FloatField, ImageField, IntegerField, ...
  - Relacionais (relação entre tabelas):
    - ⋆ ForeignKey: muitos para um
    - ⋆ ManyToManyField: muitos para muitos (pode ter tabela intermediária)
    - ⋆ OneToOneField: um para um
- Opções: Modificam o comportamento do campo
  - null, blank, choices, default, editable, primary_key, unique, verbose_name, ...
- Atributos: Descreve funcionalidades dos campos
  - concrete, hidden, is_relation, ...

Consulte

https://docs.djangoproject.com/en/4.0/ref/models/fields/

Opções dos campos

- `null`: Se `True`, guarda `NULL` no BD para campos **vazios**
  - Padrão: `False`
- `blank`: Se `True`, o campo pode ser **vazio**
  - Padrão: `False`. Ou seja, por padrão o campo é obrigatório
- `db_column`: Permite sobrescrever o nome do campo no BD, por padrão Django usa o nome do campo
- `db_index`: Se `True`, um indíce é criado no BD
- `default`: O valor padrão para o campo se ele não for informado explicitamente em uma inserção
- `editable`: Se `False`, o campo não é mostrado no admin e nem em qualquer ModelForm (formulário para modelos)
  - Padrão: `True`
- `primary_key`: Se `True`, este campo será a chave primária para o modelo.
  - Se nenhum campo do modelo for `primary_key` o Django automaticamente adicionará um campo para isso com nome `id`
  - `primary_key=True` implica em `null=False` e `unique=True`
  - Apenas uma chave primária é permitida por modelo
- `unique`: Se `True`, o valor deste campo deverá ser único para cada registro da tabela

ForeignKey

server/seriados/model.py

```
1 class Temporada(models.Model):
2     numero = models.IntegerField()
3     serie = models.ForeignKey(Serie, on_delete=models.CASCADE)
4
5
6 class Episodio(models.Model):
7     data = models.DateField()
8     titulo = models.CharField(max_length=200)
9     temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
10
11    def __str__(self):
12        return self.titulo
```

- `on_delete`: Quando um objeto referenciado for uma ForeignKey for removido o que deve ser feito
  - Neste caso ao deletar uma temporada o que fazer com seus episódios
  - `CASCADE` deletar os episódios
  - `PROTECT` proíbe a deleção da temporada
  - `SET_NULL` altera o campo `temporada` dos episódios para `NULL`
  - `SET_DEFAULT` altera o campo `temporada` para o valor padrão

ManyToManyField

server/seriados/model.py (versão 1)

```
1  from django.conf import settings
2
3  # ...
4
5  class Episodio(models.Model):
6      data = models.DateField()
7      titulo = models.CharField(max_length=200)
8      temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
9
10     def __str__(self):
11         return self.titulo
12
13
14 class Revisor(models.Model):
15     user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
16     reviews_episodios = models.ManyToManyField(Episodio)
```

- Apenas indicamos que existe a relação
  - Não damos nenhum dado sobre ela (e.g., nota)

ManyToManyField (versão 2)

server/seriados/model.py

```
1  from django.conf import settings
2
3  # ...
4
5  class Episodio(models.Model):
6      data = models.DateField()
7      titulo = models.CharField(max_length=200)
8      temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
9
10     def __str__(self):
11         return self.titulo
12
13
14 class Revisor(models.Model):
15     user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
16     reviews_episodios = models.ManyToManyField(Episodio, through='ReviewEpisodio')
17
18
19 class ReviewEpisodio(models.Model):
20     episodio = models.ForeignKey(Episodio, on_delete=models.CASCADE)
21     revisor = models.ForeignKey(Revisor, on_delete=models.CASCADE)
22     nota = models.IntegerField(validators=[
23         MinValueValidator(0),
24         MaxValueValidator(5)
25     ])
```

Opção `validators`

- O Django possui uma série de validadores
- A opção `validators` aceita uma lista de validadores
- No exemplo anterior adicionamos dois validadores: valor máximo e mínimo

Consulte

https://docs.djangoproject.com/en/4.0/ref/validators/

Choices

- Vamos focar no campo `nota` do modelo `ReviewEpisodio`
- Podemos criar um campo de inteiro ou char que mapeia um valor no BD para uma versão que será mostrada (display)

server/seriados/model.py

```
1 class ReviewEpisodio(models.Model):
2     episodio = models.ForeignKey(Episodio, on_delete=models.CASCADE)
3     revisor = models.ForeignKey(Revisor, on_delete=models.CASCADE)
4     nota = models.CharField(max_length=1, choices=(
5         ('A', "Excelente"),
6         ('B', "Bom"),
7         ('D', "Ruim"),
8     ))
```

Choices

server/seriados/model.py

```
1  class ReviewEpisodio(models.Model):
2      NOTA_A = 'A'
3      NOTA_B = 'B'
4      NOTA_C = 'C'
5      NOTAS_CHOICES = [
6          (NOTA_A, "Excelente"),
7          (NOTA_B, "Bom"),
8          (NOTA_C, "Ruim"),
9      ]
10     episodio = models.ForeignKey(Episodio, on_delete=models.CASCADE)
11     revisor = models.ForeignKey(Revisor, on_delete=models.CASCADE)
12     nota = models.CharField(
13         max_length=1,
14         choices=NOTAS_CHOICES,
15         default = NOTA_B
16     )
```

Tradução no Choices

server/seriados/model.py

```
1  from django.db import models
2  from django.conf import settings
3  from django.utils.translation import gettext_lazy as _
4
5  # ...
6
7  class ReviewEpisodio(models.Model):
8      NOTA_A = 'A'
9      NOTA_B = 'B'
10     NOTA_C = 'C'
11     NOTAS_CHOICES = [
12         (NOTA_A, _("Excelente")),
13         (NOTA_B, _("Bom")),
14         (NOTA_C, _("Ruim")),
15     ]
16     episodio = models.ForeignKey(Episodio, on_delete=models.CASCADE)
17     revisor = models.ForeignKey(Revisor, on_delete=models.CASCADE)
18     nota = models.CharField(
19         max_length=1,
20         choices=NOTAS_CHOICES,
21         default = NOTA_B
22     )
```

Acessando o valor de display de um choice

- Execute com `./manage.py shell`

```
1 from seriados import models
2
3 s = models.Serie(nome="Fawlty Towers")
4 s.save()
5
6 t = models.Temporada(numero=1, serie=s)
7 t.save()
8
9 import datetime
10 e = models.Episodio(data=datetime.date(1975,9,19),
11     titulo="A Touch of Class", temporada=t
12 )
13 e.save()
14
15 from django.contrib.auth import models as auth_models
16 u = auth_models.User.objects.get(pk=1)
17
18 r = models.Revisor(user=u)
19 r.save()
20
21 re = models.ReviewEpisodio(episodio=e, revisor=r, nota = 'A')
22 re.save()
23
24 re.get_nota_display()  # Imprime 'Excelente'
25 re.nota  # Imprime 'A'
```

Chave primária

- Relembrando: por padrão, se você não adicionar um campo de chave primária, o Django sempre adiciona o campo `id` para você
- O Django permite apenas um campo como chave primária
  - Chave primária composta não é suportado

Nome dos campos (verbose name)

- Todos os campos, exceto os relacionais, podem receber como primeiro argumento posicional o nome “verboso” do campo.
  - Se não informado, o Django usa o nome do campo (nome do atributo), trocando _ por espaços
  - Este nome é o mostrado para o usuário, por exemplo, em formulários.
- Para campos relacionais, passar o argumento por palavra chave `verbose_name`

```
1 first_name = models.CharField("person's first name", max_length=30)
2
3 first_name = models.CharField(max_length=30)  # Aqui o Django automaticamente
4                                               # cria "first name"
5
6 poll = models.ForeignKey(Poll, on_delete=models.CASCADE,
7     verbose_name="the related poll"
8 )
```

Modelos em outros arquivos

- Pode importar o modelo

```
1 from django.db import models
2 from geography.models import ZipCode
3
4 class Restaurant(models.Model):
5     # ...
6     zip_code = models.ForeignKey(ZipCode, on_delete=models.SET_NULL,
7         blank=True, null=True)
```

- Ou usar o nome exato do modelo como string
  - Se for de outra app adicionar o prefixo + ponto + nome do modelo
- Útil para referenciar modelo do mesmo arquivo que será definido abaixo

```
1 from django.db import models
2
3 class Restaurant(models.Model):
4     # ...
5     zip_code = models.ForeignKey('geography.ZipCode', on_delete=models.SET_NULL,
6         blank=True, null=True)
```

Restrições de nomes dos campos

- Os nomes dos campos devem ser identificadores válidos do Python (como o nome de qualquer variável
- Porém o Django adiciona algumas restrições extras

O nome **não** pode ...

- Ser uma palavra reservada do Python (e.g. `pass, is`)
  - Coloque um _ na frente se for realmente necessário
- Conter dois _ em sequencia
  - O Django usa como uma notação especial para a sintaxe de consulta
- Terminar com um _
  - O Django usa como uma notação especial para a sintaxe de consulta

Meta

- Podemos definir dentro do modelo uma sub classe `Meta` que controla alguns aspectos

```
1 from django.db import models
2
3 class Ox(models.Model):
4     horn_length = models.IntegerField()
5
6     class Meta:
7         ordering = ["horn_length"]
8         verbose_name_plural = "oxen"
```

Consulte

https://docs.djangoproject.com/en/4.0/ref/models/options/

Métodos do Modelo e Gerenciadores (managers)

Métodos do Modelo

Funcionalidades a nível de registro (row/record), linha da tabela

Manager

Funcionalidades a nível da tabela. Usado para filtrar vários objetos, retornando um `QuerySet`

- Acessado pelo campo `objects`
- Exemplos:
  - `models.ReviewEpisodio.objects.all()`
  - `models.ReviewEpisodio.objects.filter(nota='A')`
  - `models.ReviewEpisodio.objects.filter(nota='B')`
- Ver: https://docs.djangoproject.com/en/4.0/topics/db/queries/

Métodos do Modelo

```
1 class Episodio(models.Model):
2     data = models.DateField()
3     titulo = models.CharField(max_length=200)
4     temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
5
6     # ...
7
8     def eh_antigo(self):
9         import datetime
10        if self.data < datetime.date(2000, 1, 1):
11            return True
12        return False
```

Métodos do Modelo

Você provavelmente vai querer sempre implementar os métodos

- `__str__`: Método que diz como imprimir um objeto, um dos métodos mágicos do Python
- `get_absolute_url`: Retorna uma URL que aponta para este objeto
  - Em geral para uma visualização dos detalhes do objeto
    - E.g. página de perfil de um usuário
    - É comum temos as URLS para: listar, inserir, remover, atualizar/editar e **detalhes**

Métodos do Modelo

```
1 class Episodio(models.Model):
2     data = models.DateField()
3     titulo = models.CharField(max_length=200)
4     temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
5
6     def __str__(self):
7         return self.titulo
8
9     def get_absolute_url(self):
10        return "/seriados/episodios/%i/" % self.id
11
12    # ...
```

Métodos do Modelo

```
1 class Episodio(models.Model):
2     data = models.DateField()
3     titulo = models.CharField(max_length=200)
4     temporada = models.ForeignKey(Temporada, on_delete=models.CASCADE)
5
6     def __str__(self):
7         return self.titulo
8
9     def get_absolute_url(self):
10        from django.urls import reverse
11        return reverse('seriados:episodio_detalhes', kwargs={'pk' : self.pk})
```

- Usar o nome da URL (arquivo `urls.py` da aplicação)

Usando `get_absolute_url` no template

```
<!-- Não fazer assim -->
<a href="/seriados/episodios/{{ object.id }}/">{{ object.name }}</a>

<a href="{{ object.get_absolute_url }}">{{ object.name }}</a>
```

Atualize urls.py e views.py

server/seriados/urls.py

```
1 from django.urls import path, re_path, include, register_converter
2
3 from . import views
4
5 app_name = 'seriados'
6
7 urlpatterns = [
8     path('series/', views.series_lista, name='series_lista'),
9
10    path('episodio/<int:pk>/', views.episodio_detalhes, name='episodio_detalhes'),
11 ]
```

Atualize urls.py e views.py

server/seriados/views.py

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 from .models import Serie, Episodio
5
6 def series_lista(request):
7     lista_series = Serie.objects.all()
8     context = {'lista_series': lista_series}
9     return render(request, 'series_lista.html', context)
10
11
12 def episodio_detalhes(request, pk):
13     e = Episodio.objects.get(pk=pk)
14     return HttpResponse("Titulo: {} <br/> Data: {} <br/> Temporada: {}".format(
15         e.titulo, e.data, e.temporada
16     ))
```

Acessando o valor de display de um choice

- Execute com `./manage.py shell`

```
1 from seriados import models
2 e = models.Episodio.objects.get(pk=1)
3 e.get_absolute_url()  #  Retorna '/seriados/episodio/1/'
```

Páginas da documentação recomendadas

- https://docs.djangoproject.com/en/4.0/topics/db/models/
- https://docs.djangoproject.com/en/4.0/ref/models/fields/
- https://docs.djangoproject.com/en/4.0/ref/models/indexes/
- https://docs.djangoproject.com/en/4.0/ref/models/options/
- https://docs.djangoproject.com/en/4.0/topics/db/queries/

Atividades

1. Estudar a documentação sobre consultas em:
   - https://docs.djangoproject.com/en/4.0/topics/db/queries/
   - Tenha certeza que você entende como utilizar:
     - `Modelo.objects.get(...)`
     - `Modelo.objects.all()`
     - `Modelo.objects.filter(...)`
     - `Modelo.objects.exclude(...)`
     - Como encadear filtros (Chaining filters)
2. Crie uma view (e sua respectiva url e template) para listar todos os episódios com uma determinada nota
   - Exemplo de URL: `'/seriados/episodios/nota/A/'`
   - Solução nos próximos slides
3. Complete os modelos e crie outras views para:
   - Listar, incluir link em cada item para a página de detalhes
   - Ver detalhes

Solução Atividade 2

urls.py

```
path('episodios/nota/<str:nota>/', views.episodio_lista_nota,
    name='episodio_lista_nota'),
```

views.py

```
def episodio_lista_nota(request, nota):
    objects = Episodio.objects.filter(reviewepisodio__nota=nota)
    context = {'objects': objects, 'nota':nota}
    return render(request, 'episodio_lista_nota.html', context)
```

Solução Atividade 2

Template

```
1 {% extends "base.html" %}
2
3 {% block content %}
4 <h1>Lista de Episódios com nota {{ nota }}</h1>
5
6 <table>
7     <tr><th>titulo</th><th>data</th><th>temporada</th></tr>
8 {% for obj in objects %}
9     <tr>
10         <td>{{obj.titulo}}</td>
11         <td>{{obj.data}}</td>
12         <td>{{obj.temporada.numero}}</td>
13     </tr>
14 {% endfor %}
15 </table>
16 {% endblock %}
```

Models

Yuri Kaszubowski Lopes

Especialização em Tecnologia Python para Negócios