Django Framework do Zero

Como Criar um Aplicativo Web

*Canal **youtube.com/@VamosCodar** - Wanderson Reis*

05 - Introdução aos Models

Requisitos

1) Aula 01 - Introdução a Frameworks Web e Django: https://youtu.be/LpFNMn6Uw5s
2) Aula 02 - Configurações do Ambiente: https://youtu.be/7zxJi3_HRuw
3) Aula 03 - Primeira Aplicação - Olá Django: https://youtu.be/5MoZZttT6UA
4) Aula 04 - Introdução aos Templates: https://youtu.be/nYnZ-J8PrxA

Nesta aula:

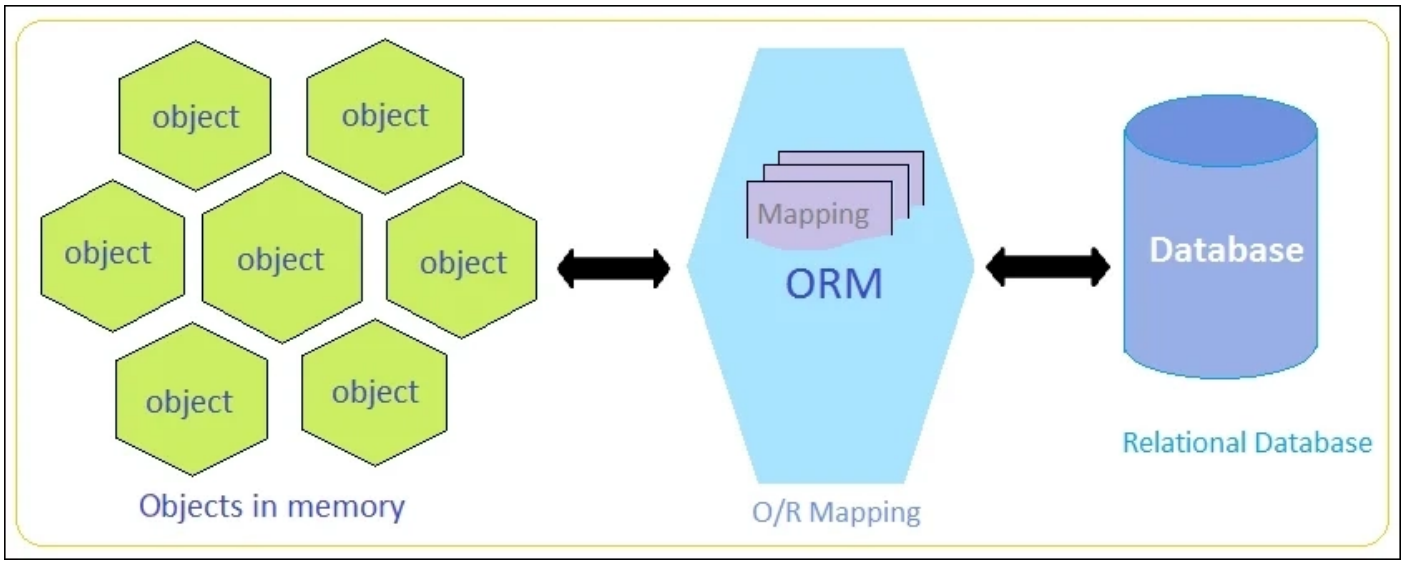
1) Introdução - Mapeamento objeto-relacional (ORM)
2) Migração dos Models básicos
3) Criar o Model Question e migrá-lo para o Banco de dados
4) Revertendo migrações de dados
5) Carregar as Questions em uma view

Referências: https://docs.djangoproject.com/en/4.2/topics/templates/

Introdução

Mapeamento objeto-relacional (ORM - Object–relational mapping)

ORM, sigla para Object Relational Mapper, é uma técnica utilizada na programação que tem como objetivo facilitar o processo de comunicação entre um banco de dados e uma aplicação. Ela cria uma camada entre a aplicação e o banco de dados.



No Django um Model é a representação do comportamento e estado das informações do que precisa ser armazenada do sistema. Em síntese são as regras de negócio traduzidas para a sintaxe do Django. Os dados são automaticamente gerenciados e mapeados pelo Django para um banco de dados correspondente. As configurações do banco de dados são definidas no arquivo **settings.py** dentro de django-do-zero/webapp (no nosso caso) e por padrão já vem configurado para utilizar uma base de dados SQLite com o nome **db.sqlite3**.

Preparação

Editar o arquivo **webapp/settings.py** e configurar as seguintes chaves (modifique o que está em azul):
LANGUAGE_CODE = 'pt-br'
TIME_ZONE = 'America/Sao_Paulo'

Passo 1: Migração dos Models básicos

Para que os Models sejam convertidos para o banco de dados é necessário realizar o processo de migração. Este recurso do Django gerencia e analisa quais Models devem ser gerados, atualizados ou revertidos (desfazer uma migração anterior).

Abra um novo terminal pelo menu **"Terminal" -> "New Terminal",** com o terminal aberto e o projeto webapp já criado ("Primeira Aplicação - Olá Django") execute:

python manage.py migrate **(ENTER)**

Se o comando acima for executado sem erros, a migração dos models **admin**, **auth**, **contenttypes** e **sessions** será aplicada (executada) conforme a imagem abaixo. Estes models são a base que toda aplicação mínima Django terá: funcionalidades para administração, autenticação, autorização, e controle de sessão.

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, 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 sessions.0001_initial... OK
```

As migrações dos models / dados é versionada automaticamente pelo Django que cuida de todo processo e em geral não exige intervenção do desenvolvedor. As migrações são organizadas de forma cronológica / sequencial e mantém uma dependência entre cada versão. Desta forma o Django consegue reverter e/ou reaplicar uma migração de dados quando necessário. Os arquivos são armazenados na pasta **migrations** dentro de cada módulo do projeto. Nunca modifique ou apague os arquivos dentro desta pasta se não souber o que está fazendo, pois do contrário não o Django não saberá qual o estado atual dos dados.

Todo o processo gira em torno de três comandos administrativos: **makemigrations**, **migrate** e **showmigrations**. O **migrate** como já vimos aplica uma migração de dados que já foi anteriormente gerada / preparada pelo makemigrations e o **showmigrations** mostra o estado atual.

Para visualizar o estado atual das migrações execute comando no terminal:

python manage.py showmigrations **(ENTER)**

O comando acima listará os módulos e as migrações realizadas ou não. O [X] indica o que foi migrado e por consequência o estado atual do banco de dados. Veja o exemplo, observe o App **polls** que não possui nenhuma “migration” disponível:

```
└─ wanderson >> python manage.py showmigrations
admin
 [X] 0001_initial
 [X] 0002_logentry_remove_auto_add
 [X] 0003_logentry_add_action_flag_choices
auth
 [X] 0001_initial
 [X] 0002_alter_permission_name_max_length
 [X] 0003_alter_user_email_max_length
 [X] 0004_alter_user_username_opts
 [X] 0005_alter_user_last_login_null
 [X] 0006_require_contenttypes_0002
 [X] 0007_alter_validators_add_error_messages
 [X] 0008_alter_user_username_max_length
 [X] 0009_alter_user_last_name_max_length
 [X] 0010_alter_group_name_max_length
 [X] 0011_update_proxy_permissions
 [X] 0012_alter_user_first_name_max_length
contenttypes
 [X] 0001_initial
 [X] 0002_remove_content_type_name
polls
 (no migrations)
sessions
 [X] 0001_initial
```

Passo 2: Criar o Model Question e migrá-lo para o Banco de dados

Como exemplo, vamos criar um Model simples para o App polls que representa uma “Question” (pergunta de enquente) com apenas o texto e a data de publicação. Edite o arquivo **polls/models.py e** acrescente as linhas em azul:

```
# Create your models here.
class Question(models.Model):
    question_text = models.CharField("Pergunta", max_length=200)
    pub_date = models.DateTimeField("Data de publicação")
```

(**salve o arquivo**).
Observe que o Model apenas descreve/especifica as regras que precisamos, mas precisamos conhecer a API de Models do Django que é bem completa com diversos tipos de dados/campos, restrições e regras que podem ser definidas para cada campo (atributo do modelo). A referência para os tipos de campo pode ser consultada na documentação oficial: https://docs.djangoproject.com/en/4.2/ref/models/fields/

Agora que temos o model precisamos fazer o Django gerar a migração com o comando **makemigrations** conforme segue:

python manage.py makemigrations **(ENTER)**

```
└─ wanderson >> python manage.py makemigrations
Migrations for 'polls':
  polls/migrations/0001_initial.py
    - Create model Question
```

Uma vez que a migration inicial foi gerada podemos visualizar as migrações disponíveis e aplicá-las na sequência:

python manage.py showmigrations **(ENTER)**

python manage.py migrate **(ENTER)**

```
└─ wanderson >> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK
```

Para saber como a migration foi gerada abra o arquivo **polls/migrations/0001_initial.py** e veja o código / instruções geradas. E não modifique o arquivo pois irá "quebrar" o controle automático de migração de dados do sistema.

Passo 3: Revertendo migrações de dados

O mecanismo de migration de dados do Django é muito robusto e facilita muito a manutenção da camada de dados. É comum precisarmos desfazer alguma migration, mas este processo é destrutivo, isto é, os dados já armazenados são perdidos. Por isso tal operação é mais indicada durante o desenvolvimento. Existem rotinas mais especializadas para trabalhar com as migrações. Mas basicamente para reverter uma migração basta executar o comando migrate seguindo do **nome do módulo** e do **número da migração** que deseja desfazer. Será uma operação em cascata que reverterá na forma cronológica inversa todas as migrações que dependam da qual esteja desfazendo. Exemplo (execute e observe):

python manage.py migrate auth 0001 **(ENTER)**

python manage.py showmigrations **(ENTER)**

Se for necessário reverter até a migration inicial (0001), use o parâmetro especial "**zero**", assim:

python manage.py migrate admin zero **(ENTER)**

Em geral, estes casos de reversão estarão relacionados a alguma modificação no Model que deseja aplicar. Nesta situação deve-se excluir os arquivos de migração correspondentes dentro da pasta **<módulo>/migrations** antes de gerar novamente as novas migrations com o comando **makemigrations**. Mas uma vez que o modelo foi migrado para um ambiente de produção (com o sistema já funcionando) as modificações nos Models devem seguir outra estratégia para não destruir os dados já existentes.

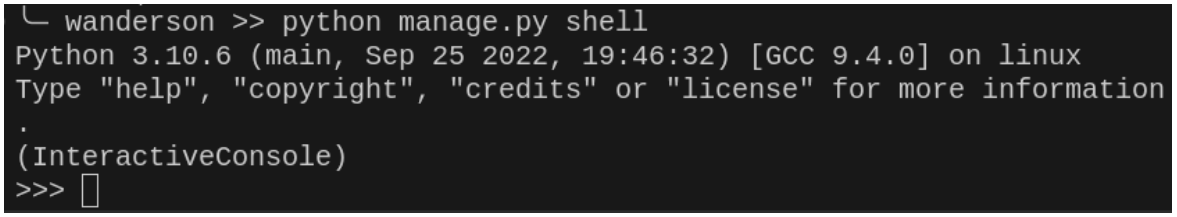
Para aplicar todas as migrações novamente, execute:

python manage.py migrate **(ENTER)**

Passo 4: Criando uma Question via Model API

Agora é a hora de testarmos se o Model Question está funcional e armazenado os dados conforme definido pelo modelo. Usaremos o modo interativo do Django, execute:

python manage.py shell **(ENTER)**



No terminal interativo do Django execute a sequência de comandos abaixo e observe os resultados.

>>> from polls.models import Question **(ENTER)**
>>> Question.objects.all() **(ENTER)**
>>> from datetime import datetime **(ENTER)**
>>> agora = datetime.now() **(ENTER)**
>>> question1 = Question(question_text="Qual sua cor favorita ?", pub_date=agora) **(ENTER)**
>>> question1.save() **(ENTER)**
>>> question1.id **(ENTER)**
>>> question1.pub_date **(ENTER)**
>>> Question.objects.all() **(ENTER)**
>>> quit() **(ENTER)**

Com estas poucas linhas já é possível consultar e exibir dados nas views. A referência completa para acesso e consulta usando a Model API está disponível em https://docs.djangoproject.com/en/4.2/topics/db/queries/

Passo 5: Carregar as Questions em uma view

Vamos criar uma view que processa / acessa e exibe todas as Enquetes (questions) em template. Para isso crie ou modifique os seguintes arquivos (incluindo ou alterando **as linhas em azul**) a seguir:

polls/views.py:

```
from polls.models import Question # Acrescentar

def ola(request): # Modificar
    # return HttpResponse('Olá django')
    questions = Question.objects.all() # recupera todos questions do banco de dados
    context = {'all_questions': questions } # cria um dicionário com todas as perguntas
    return render(request,
                  'polls/questions.html',
                   context) # renderiza o template e passa o contexto
```

(**salve o arquivo**).

Na pasta **django-do-zero/templates** (já existente na raiz do projeto) crie o arquivo **questions.html** (use o modelo abaixo como referência, pois contém a lógica para acessar os dados passados para a view).

django-do-zero/templates/polls/questions.html:

```
{% extends '_layout1.html' %}

{% block head_title %}
   Minhas Enquetes
{% endblock %}

{% block content %}
   <h1>Enquetes</h1>
   {% for question in all_questions %}
      <article>
      <p>{{ question.pub_date|date:"d / m / Y" }} | {{ question.question_text }}</p>
      </article>
   {% endfor %}
{% endblock %}
```

(**salve o arquivo**).

Para testar carregue novamente a respectiva rota para a view ola():

python manage.py runserver **(ENTER)**

No navegador web acesse: **127.0.0.1:8000/ola**

O servidor do ambiente de desenvolvimento deixa o terminal ocupado (preso) e para liberar é necessário cancelar a execução do servidor pressionando CTRL + C

Conclusão

Nesta introdução sobre models tivemos uma visão geral do processo e elementos envolvidos na criação e manutenção de regras de negócio do sistema. Nas próximas aulas vamos aprofundar ainda mais nos conceitos e estruturas do framework Django.