Conseguimos inserir alguns dados diferentes nas nossas imagens, analisando na aplicação temos dois cartões, com nomes e legendas diferentes. Entretanto, será que o ideal é usarmos dicionários para isso? Por exemplo, no caso da estrutura de "dados" para que possamos exibir determinados dados.

Esse método não é muito interessante, o ideal é utilizarmos banco de dados. **Banco de dados são estruturas que geram tabelas onde podemos inserir determinados valores** e, portanto, podemos usá-lo em conjunto com o *framework* Django.

O Django nos fornece a possibilidade de **manipular banco de dados sem a necessidade de usar a linguagem SQL**. Acessando a pasta "galeria" e depois "models.py", neste arquivo criaremos estruturas de classes de orientação a objetos. O Django "traduz" uma classe para uma tabela no banco de dados, chamamos essa ferramenta de **Django ORM** ("*Object Relational Mapper*").

models.py:

**from** django.db **import** modelsCopiar código

[Para saber mais: Object-Relational-Mapper (ORM)](https://cursos.alura.com.br/course/django-persistencia-dados-admin/task/118862)

Em suma, o ORM do Django realiza essa tradução entre o banco de dados e o Python orientado a objetos, a classe que vamos criar a partir de agora.

Vamos criar uma classe nomeada "Fotografia" (representa cada fotografia que temos) que receberá como parâmetro a biblioteca *models* que estamos importando do Django.db. Assim, ele herdará a biblioteca.

models.py:

**from** django.db **import** models

**class** **Fotografia**(models.Model):Copiar código

Após a classe, incluímos as colunas que desejamos. Por exemplo, a coluna "nome" que importa do *models.CharField()* (usamos charField por ser uma string). Dentro dos parênteses, colocamos o parâmetro *max length*, o *null* e o *blank*.

models.py:

**from** django.db **import** models

**class** **Fotografia**(models.Model):

nome = models.CharField(max\_length=100, null=**False**, blank=**False**)Copiar código

O max length é o **número limite de caracteres aceitos em um nome**, o null inserimos igual a falso **porque não pode ser um campo vazio**, e o blank é falso também.

Há uma diferença entre *null* e *blank*, o primeiro é um campo que **não contém nenhuma informação**, já o segundo podemos atribuir algo semelhante a x = "". Ou seja, está vazio mas contém algo (string vazia).

Na linha seguinte, inseriremos o campo "legenda", com as mesmas informações do campo de "nome", sendo alterado somente o número de caracteres para 150.

models.py:

**from** django.db **import** models

**class** **Fotografia**(models.Model):

nome = models.CharField(max\_length=100, null=**False**, blank=**False**)

legenda = models.CharField(max\_length=150, null=**False**, blank=**False**)Copiar código

Logo após, vamos inserir a "descricao" que será um TextField. A descrição será um texto mais longo comentando o que há na imagem. Depois inserimos o campo "foto" (para exibirmos o caminho da foto), que será um CharField (porque desejamos o nome que colocaremos para cada foto), com as mesmas informações no parênteses.

models.py:

**from** django.db import models

class Fotografia(models.Model):

nome = models.CharField(max\_length=100, **null**=**False**, blank=**False**)

legenda = models.CharField(max\_length=150, **null**=**False**, blank=**False**)

descricao = models.TextField(**null**=**False**, blank=**False**)

foto = models.CharField(max\_length=150, **null**=**False**, blank=**False**)Copiar código

Construímos a classe que se tornou um *model*, **isso acontece porque a classe representa uma tabela no banco de dados**. Para aplicarmos uma boa prática, após a classe criaremos uma função que nos devolverá o nome de cada item.

models.py:

**from** django.db **import** models

**class** **Fotografia**(models.Model):

nome = models.CharField(max\_length=100, null=**False**, blank=**False**)

legenda = models.CharField(max\_length=150, null=**False**, blank=**False**)

descricao = models.TextField(null=**False**, blank=**False**)

foto = models.CharField(max\_length=150, null=**False**, blank=**False**)

**def** **\_\_str\_\_**(self):

**return** f"Fotografia [nome={self.nome}]"Copiar código

Esse *model* precisa ser considerado pelo Django. Para isso, vamos ao terminal e, primeiro, acionar o ambiente virtual (.venv\Scripts\activate) e depois rodarmos o comando python manage.py help para analisar as possibilidades sobre as *migrations*.

.venv\Scripts\activateCopiar código

Ao teclarmos "Enter", obtemos:

(.venv) PS C:\Users\PCESTUDIO\Desktop\projetos\alura\_space

python manage.py helpCopiar código

Parte do retorno selecionado pelo instrutor:

makemigrations

Usamos o comando makemigrations para mostrar para o Django que há uma nova tabela do banco de dados que desejamos traduzir para, definitivamente, uma tabela no banco de dados. Por isso, rodamos python manage.py makemigrations.

python manage.py makemigrationsCopiar código

Como retorno, obtemos:

Migrations for 'galeria':

galeria\migrations\0001\_initial.py

* Create model Fotografia

Ele gera um documento nomeado "0001\_initial.py", em "galeria > migrations".

0001\_initial.py:

# Generated by Django 4.1 on 2022-12-02 00:48

**from** django.db **import** migrations, models

**class** **Migration**(migrations.Migration):

initial = **True**

dependencies = [

]

operations = [

migrations.CreateModel(

name='Fotografia',

fields=[

('id', models.BigAutoField(auto\_created=**True**, primary\_key=**True**, serialize=**False**, verbose\_name='ID')),

('nome', models.CharField(max\_length=100)),

('legenda', models.CharField(max\_length=150)),

('descricao', models.TextField()),

('foto', models.CharField(max\_length=100)),

],

),

]Copiar código

Esse documento é um relatório completo do que desejamos fazer: criar uma nova tabela dentro do banco de dados, com todos os campos que inserimos no *model*. Inclusive, ele gerou um id que usaremos como chave primária.

Agora, para criarmos a tabela efetivamente, voltaremos ao terminal e rodaremos python manage.py migrate.

python manage.py migrateCopiar código

Assim, ele aplica todas as migrações, inclusive, as que estavam pendentes. Como retorno, ele nos informa tudo o que está aplicando:

Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK

Applying galeria.0001\_initial... OK

Applying sessions.0001\_initial... OK

Feita todas as aplicações, temos um banco de dados. Estamos usando um banco de dados embutido no Django, o SQLite. Agora, abriremos as extensões do VSCode clicando nos três pontos do lado esquerdo da IDE e clicando em "Extensões".

Observe que agora ao invés de "Explorador" consta como "Extensões". No campo de busca, pesquisaremos por "sqlite" e selecionamos o botão verde "Instalar" do "SQlite Viewer" criado por Florian Klampfer.

Logo após, voltamos aos arquivos das pastas do projeto e observe que abaixo de ".gitignore" temos o arquivo "db.sqlite3", o nosso banco de dados. Clicando nele, temos a mensagem "O arquivo não é exibido no editor porque é um binário ou usa uma codificação de texto sem suporte".

Por isso, clicamos com o botão direito do mouse em "db.sqlite3", depois escolhemos as opções "Abrir com... > SQlite Viewer". Desse modo, podemos visualizar todos os bancos criados, como a "galeria\_fotografia".

Ao clicarmos em "galeria\_fotografia", temos uma tabela com os campos nome, legenda, descrição e foto. Podemos até criar mais itens, mas isso faremos no próximo vídeo.

Até mais!