Aula 01

1. Comandos do PIP

1.1. Exportando os Pacotes

Quando estamos trabalhando com um ambiente virtual e precisamos continuar de outra máquina ou então, um outro colega precisa também trabalhar no mesmo projeto, é muito importante que ambos esteja usando os mesmos pacotes e as mesmas versões. Para isso, o gerenciados de pacotes PIP do Python é muito importante para instalar e remover os que estamos usando.

Se digitarmos no terminal o comando pip freeze, ele vai nos listar todos os pacotes instalados até aquele momento.

Veja abaixo:

```
(.venv) C:\silver-enigma>pip freeze
asgiref==3.7.2
Django==4.2.3
Pillow==10.0.0
sqlparse==0.4.4
tzdata==2023.3
```

Isso é muito importante, já que assim é possível ver quais pacotes e suas verões que estão instalados naquele ambiente virtual.

Para exportar essa lista para um arquivo externo, basta executar o seguinte comando:

```
(.venv) C:\silver-enigma>pip freeze > requisitos.txt
```

O "requisitos.txt" será o nome do arquivo que será criado ao executarmos o comando. Com isso, uma lista de todos os pacotes e suas respectivas versões serão salvos no arquivo.

1.2. Instalando os Pacotes

Depois de criado o arquivo com a relação completa dos pacotes instalados, é possível instalá-los para um novo projeto, em um novo ambiente virtual.

Para isso, temos que executar o comando abaixo:

```
(.venv) C:\silver-enigma>pip install -r requisitos.txt
```

Com isso, todos os pacotes especificados no arquivo "requisitos.txt" serão instalados no ambiente virtual.

1.3. Atualizando um Pacote

Com o passar do tempo, é comum que alguns vários pacotes sejam atualizados, seja por motivos de performance ou de novas funcionalidades.

Para atualizar um pacote, basta digitarmos o seguinte comando:

```
(.venv) C:\silver-enigma>pip install pillow --upgrade
Requirement already satisfied: pillow in c:\silver-enigma\.venv\lib\site-packages (10.0.0)
```

ou então:

```
(.venv) C:\silver-enigma>pip install pillow -U
Requirement already satisfied: pillow in c:\silver-enigma\.venv\lib\site-packages (10.0.0)
```

Nesse caso, estamos tentando atualizando o pacote Pillow. Como ele já está atualizado, não será feito nada. Mas, numa eventualidade de um pacote desatualizado, ele será atualizado para a última versão disponível.

1.4. Atualizando o Arquivo

Uma vez que nossos pacotes estão atualizados, eles podem não estar com a referência das versões atualizadas no arquivo "requisitos.txt". Caso seja esse o caso, para atualizar o arquivo basta reexecutar o comando para exportar os pacotes. Isso irá reescrever todo o arquivo com as informações atualizadas dos pacotes. Se algum pacote foi atualizado, a nova versão será adicionada no local da antiga; se algum pacote foi instalado, ele será escrito no arquivo junto aos demais; se algum pacote foi removido, ele será removido da lista.

2. Formulário de Busca

É importante que nossa aplicação seja capaz de realizar buscas através de determinados itens usando algum campo de input e um formulário.

Para esse exemplo, iremos usar como exemplo um projeto (chamado Picasa). Ele possuirá um aplicativo (chamado galeria) que terá diversas imagens que, por enquanto, serão cadastradas pelo painel administrativo. Futuramente, será adicionado uma forma de cadastrar usuários que serão capazes de entrar no sistema e registrar as imagens eles mesmos.

2.1. Os Arquivos

O sistema de arquivos será o mesmo que já vimos até o momento. As configurações que não forem mostradas, é porque se está sendo usado o que foi visto até o momento.

2.2. Algumas Melhorias

Abaixo, temos algumas capturas de telas com melhorias do que já vimos:

Nas imagens abaixo, em vez de chamar o aplicativo pelo nome (no arquivo **/picasa/settings.py**), é usado o caminho da classe que configura o aplicativo (no arquivo **/galeria/apps.py**):

C:\silver-enigma\picasa\settings.py

```
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     # meus apps
41     'galeria.apps.GaleriaConfig',
42 ]
```

C:\silver-enigma\galeria\apps.py

```
1 from django.apps import AppConfig
2
3
4 class GaleriaConfig(AppConfig):
5     default_auto_field = 'django.db.models.BigAutoField'
6     name = 'galeria'
7
```

3.2. Arquivo models.py

Agora, veja como está o arquivo /galeria/models.py, com a classe que será usada pelo aplicativo:

C:\silver-enigma\galeria\models.py

```python
from django.db import models
from datetime import datetime

class Imagem(models.Model):
    OPCOES_CATEGORIA = [
        ('PESSOA', 'pessoa'),
        ('CONSOLE', 'console'),
    ]

    nome = models.CharField(max_length=100, null=False, blank=False)
    legenda = models.CharField(max_length=150, null=False, blank=False)
    categoria = models.CharField(max_length=100, choices=OPCOES_CATEGORIA, default='')
    descricao = models.TextField(null=False, blank=False)
    foto = models.ImageField(upload_to='imagem/%Y/%m/%d/', blank=True)
    eh_publicada = models.BooleanField(default=False)
    data_fotografia = models.DateTimeField(default=datetime.now, blank=False)

    def __str__(self):
        return self.nome

```

Repare nas diferenças agora dele, comparado com o que foi visto até o momento.

Dentro da classe, há uma constante com uma lista de opções (que são tuplas). Essa constante será usada no atributo **categoria** através do parâmetro **choices**.

Além desse parâmetro, há também outros parâmetros novos que estão sendo usados. O **null** vai ser usado para determinar se aquele campo da tabela aceitará valores nulos (True) ou não (False). Já o parâmetro **blank** é para indicar que aquele campo do formulário é obrigatório (False) ou não (True), isto é, se ele pode ficar em branco (True) ou não (False).

Também há o uso do pacote datetime para pegar a data e hora atual e, então, usar ele como preenchimento padrão para o campo **data_fotografia**.

3.3. Arquivo admin.py

O arquivo admin.py também possui algumas alterações. Veja abaixo:

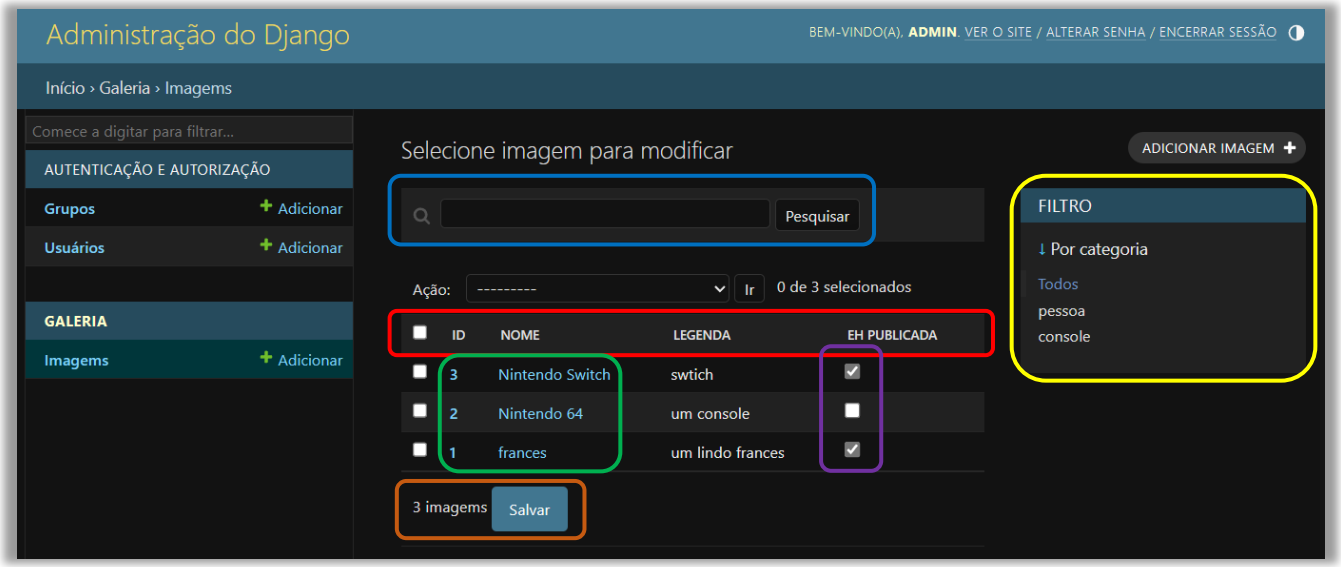
C:\silver-enigma\galeria\admin.py

```python
from django.contrib import admin
from galeria.models import Imagem

class ImagemAdmin(admin.ModelAdmin):
    list_display = ('id', 'nome', 'legenda', 'eh_publicada')
    list_display_links = ('id', 'nome')
    search_fields = ('nome',)
    list_filter = ('categoria',)
    list_editable = ('eh_publicada',)
    list_per_page = 10

admin.site.register(Imagem, ImagemAdmin)

```

Repare nas variáveis que estão sendo usadas dentro da classe. Essa classe tem a finalidade de gerenciar a nossa classe dentro do painel administrativo do projeto.

Veja o uso de cada um:

- list_display (vermelho): especifica quais campos estarão visíveis na lista de objetos;
- list_display_links (verde): indica quais campos são clicáveis para exibir o objeto;
- search_fields (azul): indica quais campos podem ser usados para buscar algum objeto;
- list_filter (amarelo): indica qual campo pode ser filtrado por uma caixa auxiliar à direita da página;
- list_editable (roxo): indica quais campos são editáveis a partir da lista de objetos;
- list_per_page (laranja): indica quantos objetos serão exibidos por página;



3.4. Arquivo urls.py

Como a busca será enviada através de um formulário HTML, não será necessário incluir uma variável para a url. Então ele não terá nenhuma diferença do que já foi visto:

C:/silver-enigma/galeria/urls.py

```python
from django.urls import path
from galeria.views import view_index, view_busca, view_imagens

app_name = 'galeria'

urlpatterns = [
    path('', view_index, name='index'),
    path('imagens/', view_imagens, name='imagens'),
    path('busca/', view_busca, name='busca'),
]

```

3.5. Arquivos HTMLs

Os arquivos HTMLs também não terão nada de especial, com exceção do formulário que será usado para realizar a busca. Veja como ele ficará:

C:\silver-enigma\galeria\templates\galeria\parciais\menu.html

```html
<nav>
    <ul>
        <li><a href="{% url 'admin:index' %}">ADMIN</a></li>
        <li><a href="{% url 'galeria:index' %}">GALERIA</a></li>
        <li><a href="{% url 'galeria:imagens' %}">IMAGENS</a></li>
    </ul>
</nav>
<div>
    <form action="{% url 'galeria:busca' %}">
        <input type="text" name="buscando" placeholder="o que quer buscar?">
        <button type="submit">buscar</button>
    </form>
</div>

```

A maior diferença está no parâmetro action da tag HTML form. O que usamos ali, é parecido com o que usamos para criar as urls. Usamos a tag do Django url para especificar o arquivo que irá receber a requisição do formulário. Nesse caso, será o arquivo **/galeria/templates/galeria/paginas/busca.html**.

Como ele servirá para exibir os resultados da busca, não será diferente de outros arquivos HTMLs usados para o mesmo fim. A diferença ficará na função que a chama, no arquivo **/galeria/views.py**.

3.6. Arquivo views.py

O valor que está sendo enviado agora é através de um formulário HTML, que é enviado juntamente com a requisição da página. Se verificar o valor da variável request recebida pela função será um tanto diferente.

Veja um exemplo:

```
<WSGIRequest: GET '/galeria/busca/'>
[17/Jul/2023 13:39:11] "GET /galeria/busca/ HTTP/1.1" 200 932
<WSGIRequest: GET '/galeria/busca/?buscando=ninte'>
[17/Jul/2023 13:39:43] "GET /galeria/busca/?buscando=ninte HTTP/1.1" 200 891
```

Na primeira linha, o valor do GET é apenas a página a ser carregada. Já na terceira linha, é enviado junto um parâmetro buscando (definido no formulário HTML) com um valor digitado no campo (nesse caso 'ninte').

Veja como ficará o arquivo /galeria/views.py:

C:\silver-enigma\galeria\views.py

```python
from django.shortcuts import render
from galeria.models import Imagem

def view_index(request):
    return render(request, 'galeria/paginas/index.html', context={})

def view_imagens(request):
    imagens = Imagem.objects.all()
    return render(request, 'galeria/paginas/imagens.html', context={'imagens':imagens})

def view_busca(request):
    imagens = Imagem.objects.all()
    print(request)

    if 'buscando' in request.GET:
        nome = request.GET['buscando']
        if nome:
            imagens = imagens.filter(nome__icontains=nome)

    return render(request, 'galeria/paginas/busca.html', context={'imagens':imagens})

```

Veja que, a função sempre carrega todos os objetos do banco de dados, com a diferença de que, se foi encontrado a palavra "buscando" na string **request.GET**, será realizado um filtro dos dados buscados. Para isso, se chama o método **filter** e passa para o parâmetro **nome__icontains** o valor enviado pelo GET.

Repare no nome do parâmetro. Ele tem como prefixo o nome do campo a ser verificada a string. Se o campo a ser buscado fosse **legenda**, o parâmetro seria **legenda__icontais**.