iscte
TECNOLOGIAS
E ARQUITETURA

Desenvolvimento para a Internet e Aplicações Móveis

Django:

CSS e imagens

Herança entre templates

Otimização de *views*

Upload de imagens/ficheiros

Decoradores

Sumário

- Django
  - Ficheiros "estáticos": CSS e imagens
  - Herança entre *templates*
  - *Views* únicas para mostrar formulários e tratar os dados
  - Upload de imagens (e ficheiros)
  - Decoradores

Ponto de situação

- Na semana passada…
  - Continuaram o desenvolvimento de um projeto tutorial de Django
  - Acabámos a semana com as seguintes funcionalidade na app "votacao" :
    - *Login*, *logout* e registo de novos utilizadores
    - Possibilidade de apagar questões e opções
    - Acesso a funcionalidades consoante o tipo de utilizador e se está "logado" ou não
  - Nesta semana:
    - Definir CSS e incluir imagens no site
    - Redefinir os *templates* de forma a que tirem partido de herança
    - Possibilidade de realizar upload de imagens
    - Simplificar código de forma a ter menos *views* e usar "decoradores" para controlar o acesso

Exercício 7

- Considere o projeto **`sitepr`** desenvolvido nas últimas semanas. Pretende-se:

  1. **Desenvolver o *layout* do site** através de um ficheiro CSS, incluindo cores de background/fontes e posicionamento central dos *forms* na página. pretende-se que o site fique com um aspeto original e ao seu gosto. Pode reutilizar um CSS desenvolvido anteriormente, ou definir um novo ficheiro CSS
  2. **Aplicar herança de *templates* ao seu site,** em que index.html será o *template* base e os outros serão derivados.
  3. **O utilizador deverá poder fazer upload da sua foto no momento do registo.** A foto deverá depois aparecer na página que apresenta a informação sobre o utilizador.

Exercício 7

4. **Simplificar o código de algumas *views***, nomeadamente ter uma única *view* para o código de inserir/submeter questões, assim como para o código de inserir/submeter opções
5. **Retirar o formulário de login da página inicial e colocá-lo numa nova página** específica para o efeito.
6. **Realizar a gestão das permissões dos utilizadores e dos administradores com recurso aos "decoradores" do Django.** As *views* do aluno devem ser decoradas com `@login_required` e encaminhar para a página de login caso ainda não tenha sido feita a autenticação. Por outro lado, as funções específicas ao administrador devem ser programadas com permissões através de `@permission_required`.

iscte
TECNOLOGIAS
E ARQUITETURA

Django

Ficheiros "estáticos": CSS e imagens

Localização dos ficheiros estáticos

- No ficheiro `"settings.py"`, do package geral do seu site, poderá encontrar a variável `STATIC_URL`

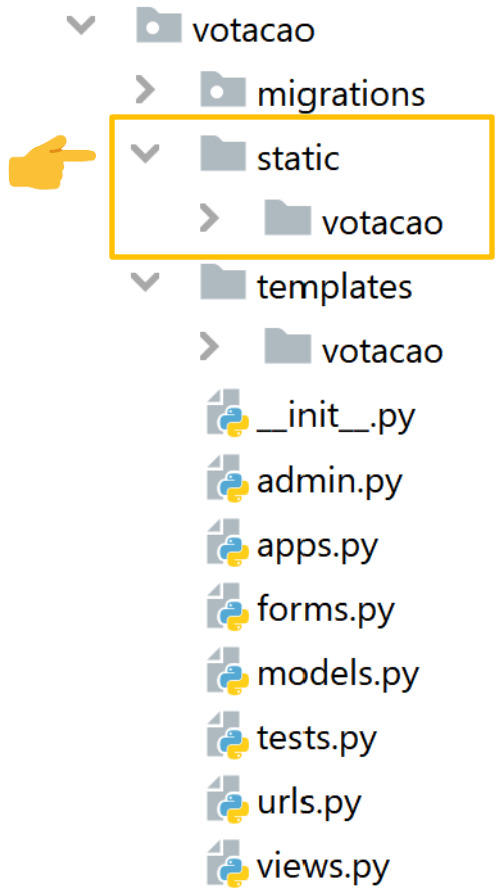
```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/

STATIC_URL = 'static/'
```

- Esta variável define a localização dos ficheiros estáticos – por exemplo CSS, imagens e javascript
- Por defeito essa localização será o diretório `static` de cada *app*

Criação do diretório `static`

- Assim, se quisermos adicionar estilos CSS às páginas do nosso site, deveremos

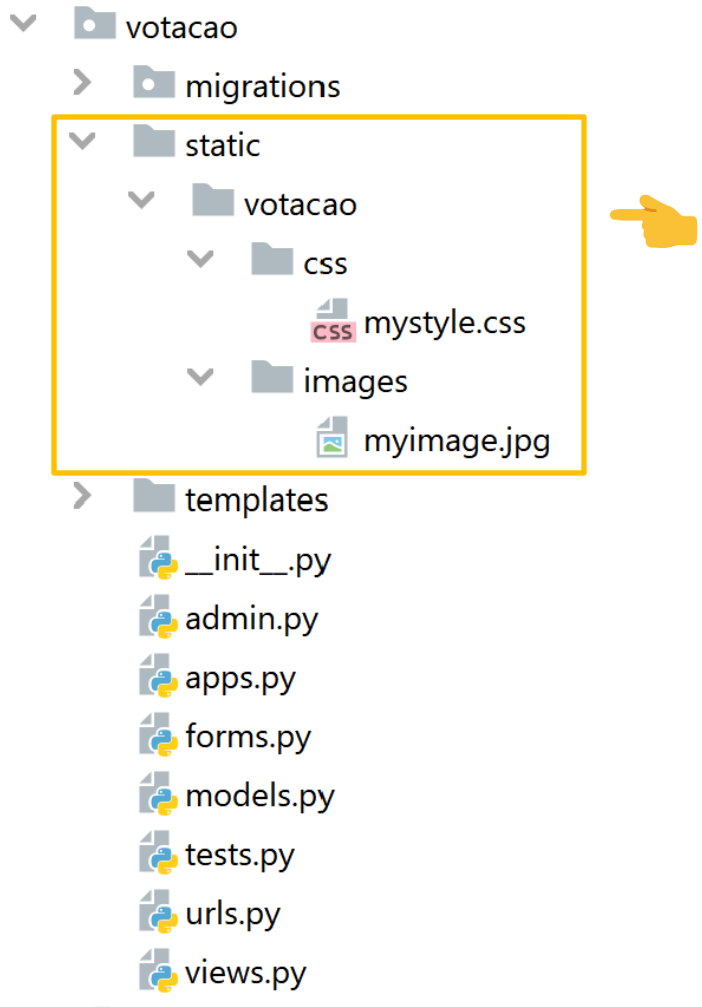
  1. criar um diretório `static` dentro da *app* (e.g. `votacao/static`)
  2. criar um subdiretório com o nome da *app*, dentro do `static` (e.g. `votacao/static/votacao`)
  3. Poderemos posteriormente subdividir mais, para criar diretórios em separados para estilos, imagens, scripts JS, etc. (e.g. `votacao/static/votacao/css`, `votacao/static/votacao/imagens`, …)



Criar CSS simples com ligação a uma imagem

- Vamos assumir que existem duas pastas – `css` e `images` – dentro da pasta `votacao/static/votacao`
  - pasta `css` para colocar o(s) ficheiro(s) `.css`
  - pasta `images` para colocar as imagens usadas no site
- Como exemplo vamos colocar o seguinte código num ficheiro chamado `mystyle.css`:

```
body {
  background-image: url('../images/myimage.jpg');
  background-size: cover;
  color: darkslategrey;
  font-family: Arial, sans-serif;
}
```



Criar CSS simples com ligação a uma imagem

- Para um *template* utilizar o CSS definido, é necessário fazer a ligação ao ficheiro .css correspondente
  - Na secção <**head**> de um *template* acrescenta-se:

👉

```
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'votacao/css/mystyle.css' %}">
```

- Para inserir imagens nas páginas (*templates*), o processo é semelhante:

```
<img src="{% static 'votacao/images/mypageimage.jpg' %}">
```

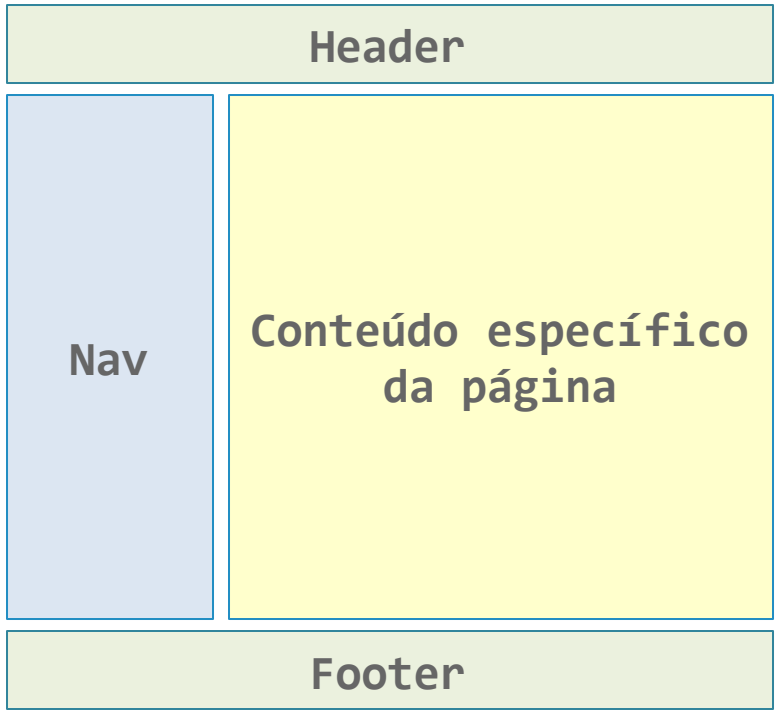
iscte
TECNOLOGIAS
E ARQUITETURA

Django

Herança entre templates

Herança entre *templates* – conceito

- Vamos supor que as paginas do seu site seguiam uma estrutura semelhante ao seguinte:



Como se pode fazer para definir outras páginas com estrutura semelhante, mas sem ter que reescrever o html das partes que são comuns (Header, Footer e Nav)?

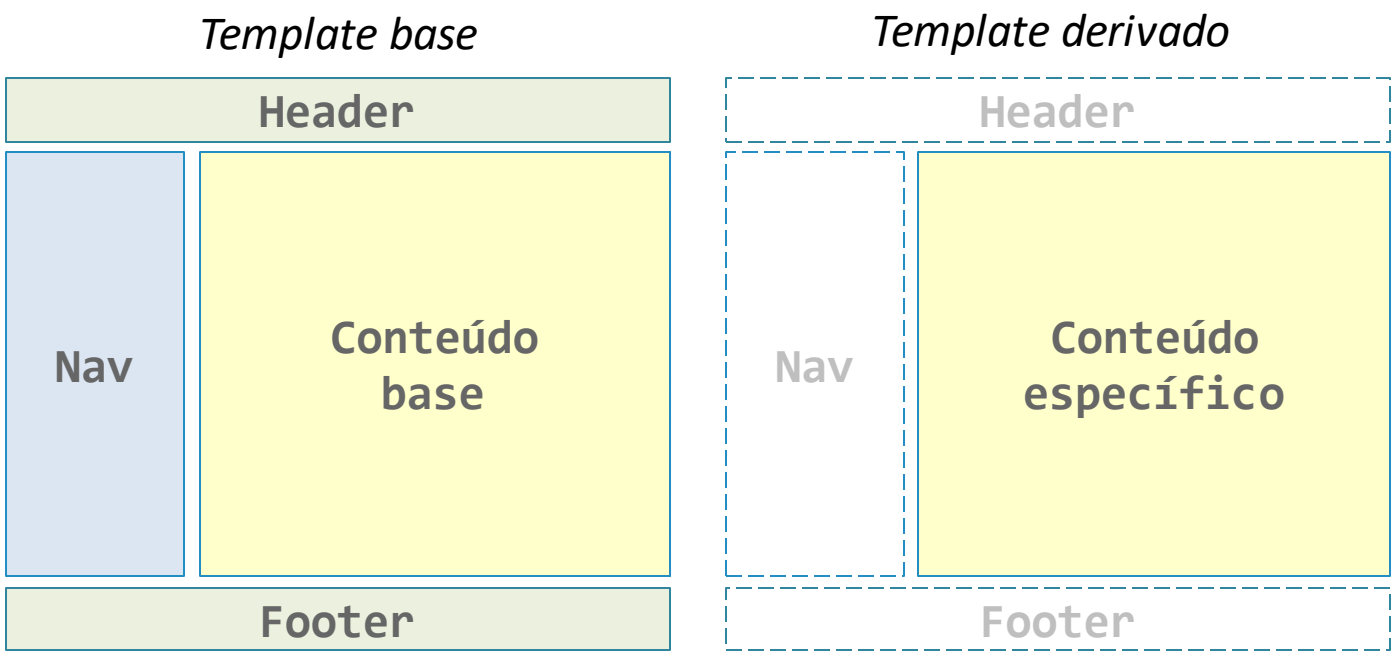
Solução – utilizar herança entre *templates*!

Herança entre *templates* – conceito

- Pode-se define-se um *template base* onde se programam os elementos comuns
- E *templates derivados*, onde se implementam apenas as partes específicas desse *template*

No *template base* poderíamos definir o Header, Footer, Nav e o conteúdo base da página – por exemplo o conteúdo da página de entrada no site.

Nos *templates derivados* apenas será necessário redefinir o conteúdo específico das páginas correspondentes



Herança entre *templates* – exemplo (base)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
    {% load static %}
   <link rel="stylesheet" type="text/css" href="{% static 'votacao/css/mystyle.css'%}">
   <title> {% block titulo %} Template base {% endblock %} </title>
</head>
<body>
   <header><h1> Este é o header (definido no template base)</h1></header>
   <nav>
      <a href={% url 'votacao:base' %}>Link para a base</a>
      {% block outros_links %}
         <a href={% url 'votacao:derivado' %}>Link para a derivada</a>
      {% endblock %}
   </nav>
   <main>
      {% block conteudo %}
         <h1>Base</h1>
         <p> Isto aqui seriam conteúdos da página base.</p>
      {% endblock %}
   </main>
</body>
</html>
```

👈 **block titulo**

👈 **block outros_links**

👈 **block conteudo**

Herança entre *templates* – exemplo (derivado)

```
{% extends "./base.html" %}

{% block titulo %}
   Template derivado
{% endblock %}

{% block outros_links %}
{% endblock %}

{% block conteudo %}
   <h1>Derivado</h1>
   <p>Aqui seria o conteúdo da pag derivada.</p>
{% endblock %}
```

👈 **extends "./base.html"** – indica que é derivado do *template* base.html, localizado no mesmo diretório

👈 **block titulo**
mudou-se o título da página

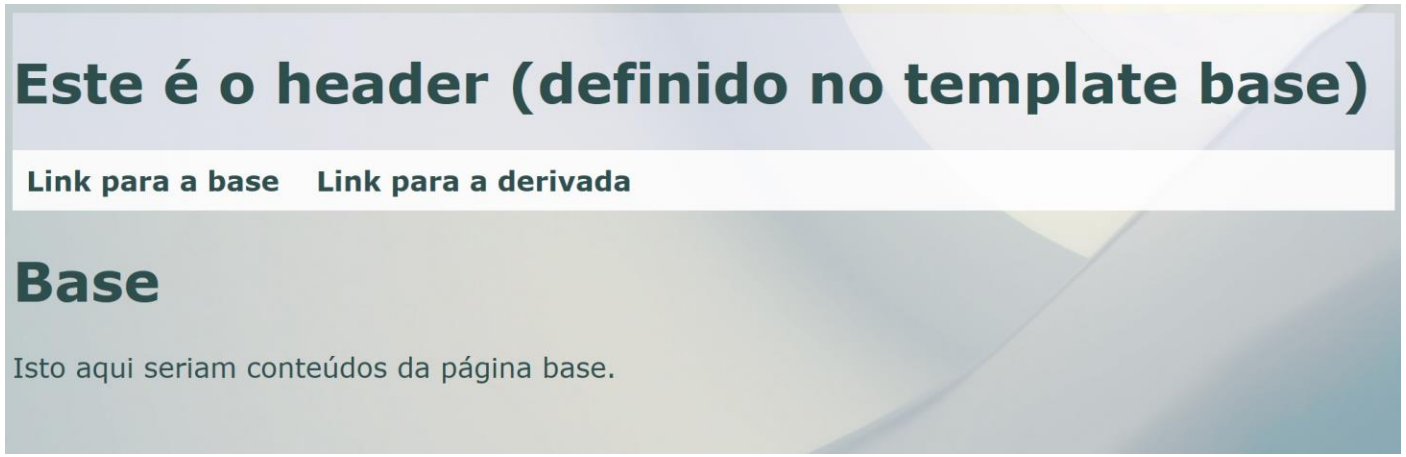
👈 **block outros_links**
tirou-se o link para a derivada

👈 **block conteúdo**
substituiu-se o conteúdo

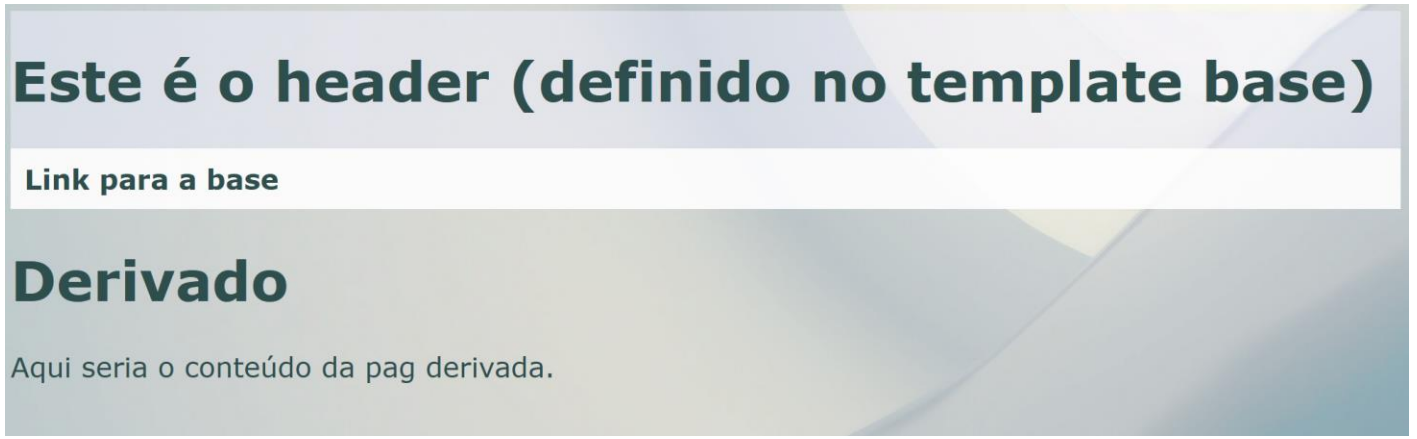
Nota! {% **extends …** deverá ser a primeira *tag* a ser encontrada no *template* derivado

Herança entre *templates* – exemplo (resultado)

Rendering do *template* base



Rendering do *template* derivado



Herança entre *templates* – super

base.html

```
<title>{% block title %} MY SITE NAME {% endblock %}</title>
```

sectionX.html

```
{% extends "base.html" %}
{% block title %} {{ block.super }} - SECTION X {% endblock %}
```

iscte
TECNOLOGIAS
E ARQUITETURA

Django

Upload de imagens

Configurações

- Definir as variáveis `MEDIA_URL` e `MEDIA_ROOT`, em `settings.py`

```
import os
...
MEDIA_URL = '/votacao/static/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'votacao/static/media')
```

👈 O diretório é criado pelo django, quando se usar pela primeira vez

- Definir um URL adicional em `urls.py` (do package do seu site)

```
from django.conf import settings
from django.conf.urls.static import static
...
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

☝ Funciona para desenvolvimento (é o nosso caso).
Para *deployment* num servidor seriam necessárias configurações diferentes.

URL, *view* e *template* para realizar uploads

- URL para uma *view* de upload (acrescentar aos URLs da *app*)

```
path('fazer_upload', views.fazer_upload, name='fazer_upload')
```

- Exemplo de código da *view* para mostrar e tratar o formulário de upload

```
from django.core.files.storage import FileSystemStorage
...
def fazer_upload(request):

    if request.method == 'POST' and request.FILES['myfile']:
        myfile = request.FILES['myfile']
        fs = FileSystemStorage()
        filename = fs.save(myfile.name, myfile)
        uploaded_file_url = fs.url(filename)
        return render(request, 'votacao/fazer_upload.html', {'uploaded_file_url': uploaded_file_url})

    return render(request, 'votacao/fazer_upload.html')
```

URL, *view* e *template* para realizar uploads

- Exemplo de código de *template* para o formulário de upload

```
<form method="post" enctype="multipart/form-data"  action="{% url 'votacao:fazer_upload' %}">
   {% csrf_token %}
   <input type="file" name="myfile">
   <input type="submit" value="Upload">
</form>

{% if uploaded_file_url %}
   <p>Feito o upload para:
      <a href="{{ uploaded_file_url }}">{{ uploaded_file_url }}</a>
   </p>
{% endif %}

<p><a href="{% url 'votacao:index' %}"> Voltar ao inicio</a></p>
```

Quando há carregamento de ficheiros, não esquecer de incluir em <form …> o atributo enctype="multipart/form-data"

iscte
TECNOLOGIAS
E ARQUITETURA

Django

Decoradores

Decoradores – conceito

- Muitas vezes queremos garantir que um utilizador "não logado" não consegue correr o código de uma *view* através da barra de navegação de um browser. Poderíamos fazer, por exemplo:

```
def my_view(request):
    if not request.user.is_authenticated:
        return render(request, 'my_app/login.html')

    # código que só seria suposto correr caso o utilizador estivesse "logado"
```

- O Django oferece uma forma mais simples de conseguir este objetivo, usando "decoradores"

```
@login_required(login_url='/my_app/fazer_login')
def my_view (request):
    # desta maneira a view só é corrida caso o utilizador esteja "logado"
    # se o utilizador não estiver "logado", é encaminhado automaticamente
    # para a view que corresponder ao URL '/my_app/fazer_login'
```

`@login_required` é um dos vários "decoradores" oferecidos pelo Django para lidar com situações deste género (i.e., correr o código de uma função apenas no caso se serem satisfeitas determinadas condições

Decorador @login_required

@login_required – restringir o acesso a utilizadores "logados"

- Omissão do parâmetro login_url:

```
from django.contrib.auth.decorators import login_required
...
@login_required
def my_view (request):
    ...
```

Caso não se indique um URL, por defeito será encaminhado para o que estiver definido na variável **`settings.LOGIN_URL`**

- Se quisermos redirecionar para outro lado:

```
@login_required(login_url='/my_app/fazer_login')
...
```

ou

```
@login_required(login_url=reverse_lazy('my_app:fazer_login'))
...
```

Neste caso não esquecer de acrescentar o URL da *view* `fazer_login` à lista presente em `urls.py` da nossa *app*:

```
path('fazer_login', views.fazer_login, name='fazer_login'),
```

Decorador @user_passes_test

@user_passes_test – restringe o acesso a utilizadores que satisfaçam determinada condição

- A condição é dada pela função indicada no argumento deste decorador – essa função recebe um objeto User no seu argumento
- Exemplos:

```
def check_email(user):
    return user.email.endswith('@iscte-iul.pt')

@user_passes_test(check_email, login_url=reverse_lazy('my_app:fazer_login'))
def my_view (request):
    ...
```

Neste exemplo iria verificar se o email do utilizador terminava em `@iscte-iul.pt`. Caso isso não acontecesse, o utilizar seria encaminhado para a *view* `fazer_login`

```
def check_superuser(user):
    return user.is_superuser

@user_passes_test(check_superuser, login_url=reverse_lazy('my_app:fazer_login'))
def my_view (request):
    ...
```

Neste outro exemplo iria verificar se o utilizador é um superuser

Decorador @permission_required

@permission_required – restringir o acesso a utilizadores com determinadas permissões

- O Django cria automaticamente permissões associadas aos modelos de dados utilizados no projeto:
  - **add** – permissão para criar instâncias do modelo de dados
  - **delete** – permissão para apagar instâncias do modelo de dados
  - **change** – permissão para atualizar instâncias do modelo de dados
  - **view** – permissão para consultar instâncias do modelo de dados
- As permissões podem ser identificadas através de *strings* que seguem a seguinte regra:

<nome_da_app>.<nome_da_acao>_<nome_do_modelo>

- Exemplos:

'votacao.add_questao' **'votacao.view_questao'** **'votacao.delete_opcao'** **'votacao.change_opcao'**

Decorador @permission_required

- Exemplo – Restringir acesso para inserir questões

```
@permission_required('votacao.add_questao', login_url=reverse_lazy('votacao:fazer_login'))
def inserir_questao(request):
    ...
```

ou, caso não se usasse um decorador:

```
def inserir_questao(request):
    if not request.user.has_perm('votacao.add_questao'):
        return HttpResponseRedirect(reverse('votacao:fazer_login'))
    ...
```

- Num *template* também é possível verificar as permissões para aparecerem (ou não) alguns elementos de uma página:

```
{% if perms.votacao.add_questao %}
    <p><a href="{% url 'votacao:inserir_questao' %}">Inserir uma nova questão</a></p>
{% endif %}
```

Atribuir permissões a um utilizador

- Permissões e utilizadores
  - *superuser* – tem permissões para tudo
  - utilizador normal – por defeito, ao ser criado começa com um conjunto de permissões vazio, mas podem-lhe ser atribuídas permissões
  - utilizador anónimo – não tem permissões para nada (e não pode ter!)
- Exemplo de código para atribuir permissões a um utilizador

```
from django.contrib.auth.models import Permission
...
p1 = Permission.objects.get(codename="delete_questao")
p2 = Permission.objects.get(codename="add_questao")

user = User.objects.get(username="Rui")
user.user_permissions.add(p1,p2)

print(user.get_user_permissions())  ⟶  {'votacao.add_questao', 'votacao.delete_questao'}
```

Primeiro obtêm-se os objetos **Permission** pretendidos, com **Permission.objects.get(...)**, e depois atribuem-se as permissões ao utilizador, com **user_permissions.add(...)**.

Note que ao usar **Permission.objects.get(...)**, os "codename" das permissões não incluem o nome da *app*.