Creación dun proxecto con Django

Xurxo Freitas

Data	Motivo da modificación
13/11/25	**Versión inicial**

Índice

Índice de imaxes

1. Introdución

Este manual está enfocado para instalar e crear un pequeno proxecto en **Windows 11** con **Django 5.2.8.** Tamén ter en conta que fará falta ter unha base en **Python**

As rutas dos arquivos do proxecto (da nosa web) partirán do directorio **entorno_django**

2. Preparar o entorno

2.1. Instalar Python

Descarga **Python** dende a **web oficial** ou co comando

winget install Python.Python.[versión]

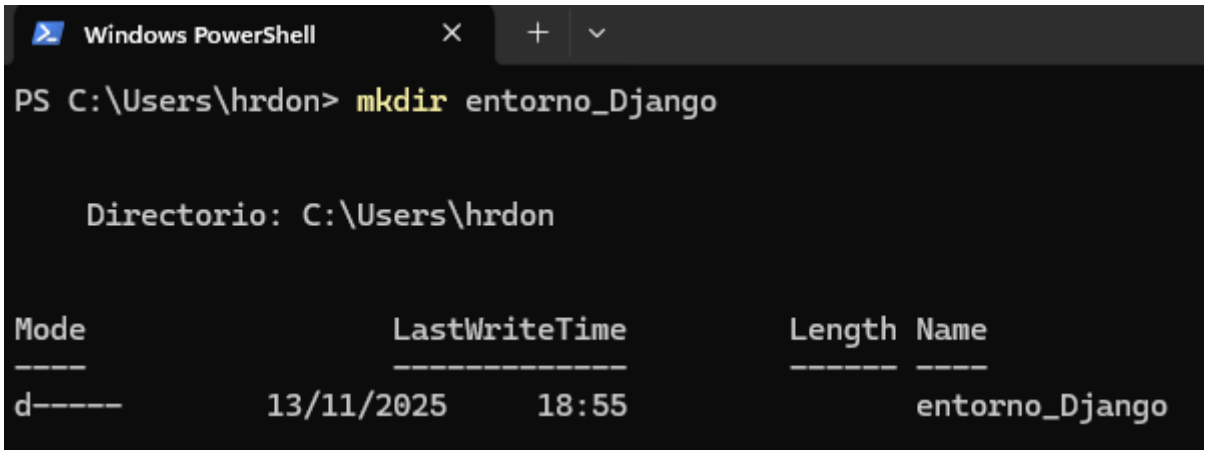
Utilizaremos a **versión 3.14.0** xa que é a ultima versión **compatible con Django**

Figura 1: Comprobar versión Python

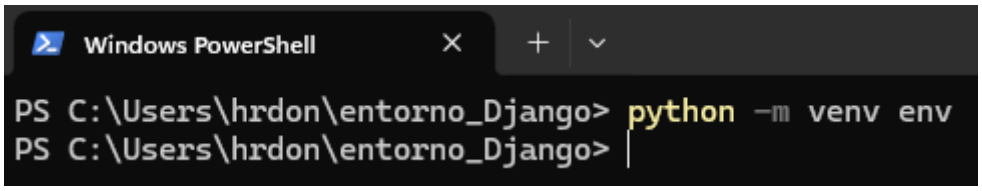
2.2. Instalar Django

2.2.A) Preparar entorno virtual

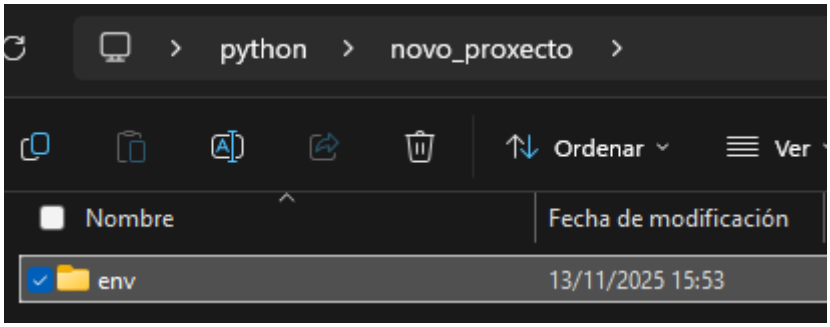
Antes de comenzar vamos a preparar o noso un **entorno virtual** para manter as dependencias de Python illadas para cada proxecto



A continuación creamos o entorno virtual **dentro do directorio** que acabamos de crear co comando **python -m venv env**



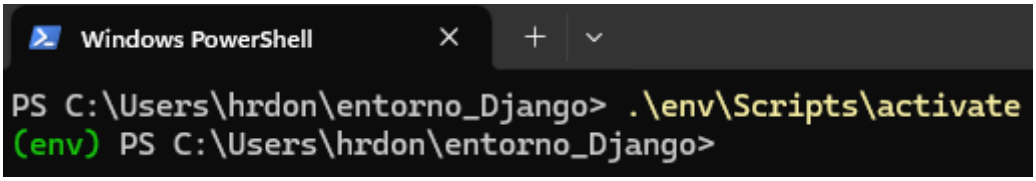
Agora xa temos unha nova carpeta chamada **env** con todo o necesario



Antes de activar o entorno **habilitamos** a **execución de scripts** como **administrador**

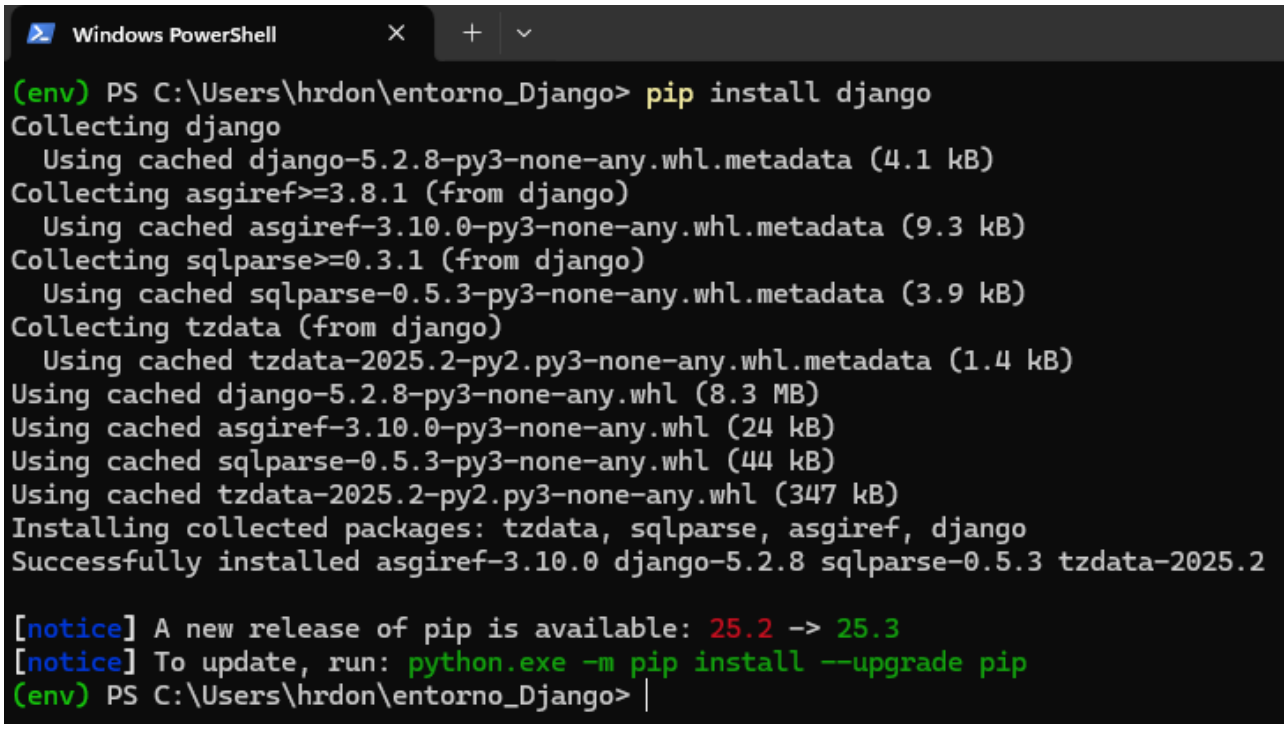
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

Por último vamos activalo co comando **env\Scripts\activate**

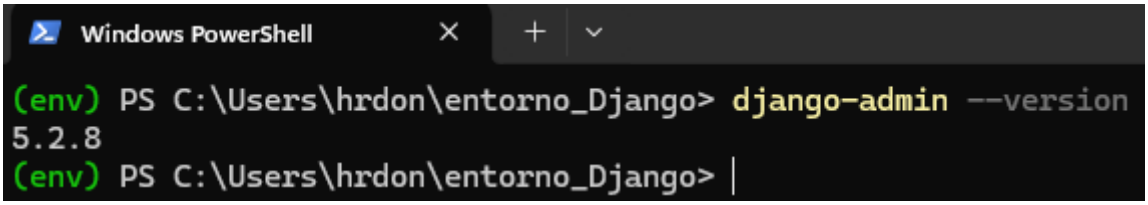


2.2.B) Instalación de Django

Tendo o **entorno virtual activo** executamos o seguinte comando



para asegurarnos comprobamos a instalación con **django-admin –versión**



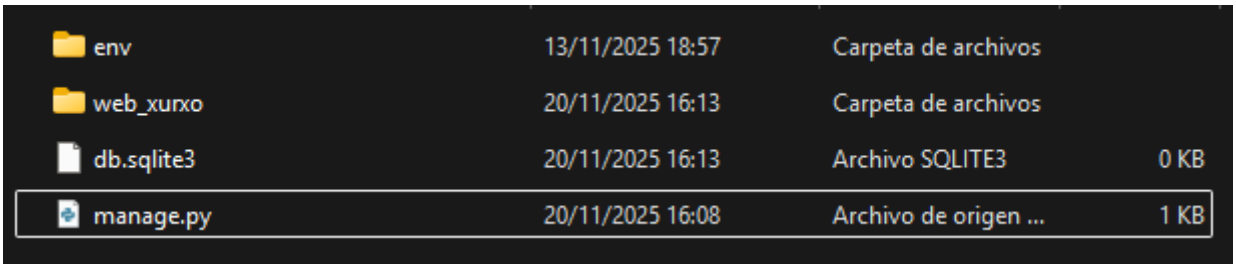
3. Crear un proxecto

Imos crear un novo proxecto, falémolo co comando **django-admin startproject [nome] .**

Ollo: o punto ao final refírese ao directorio actual

```
(env) PS C:\Users\hrdon\entorno_Django> django-admin.exe startproject web_xurxo
(env) PS C:\Users\hrdon\entorno_Django>
```

Os ficheiros que temos agora dentro de **entorno_django** son os seguintes:



Agora comprobamos dende **entorno_django** (donde temos gardado o ficheiro **manage.py**) executamos o comando **python manage.py runserver** para arrancar o servidor.

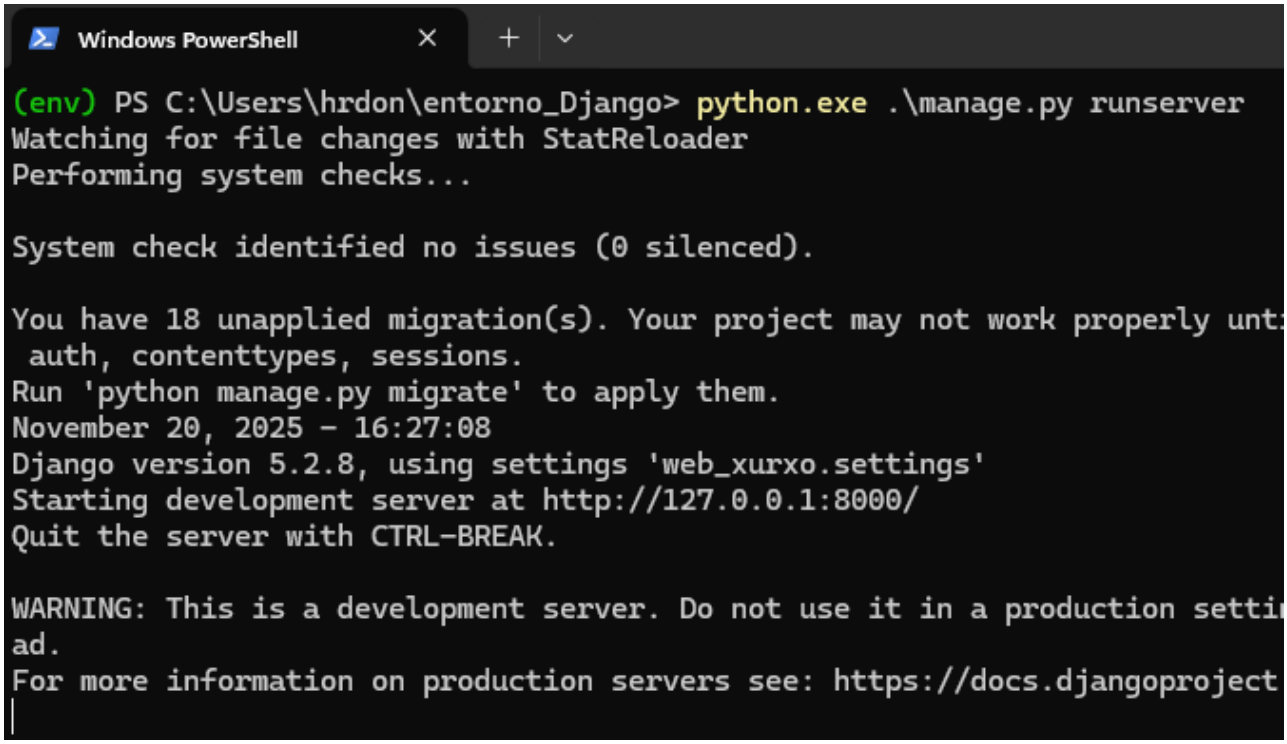
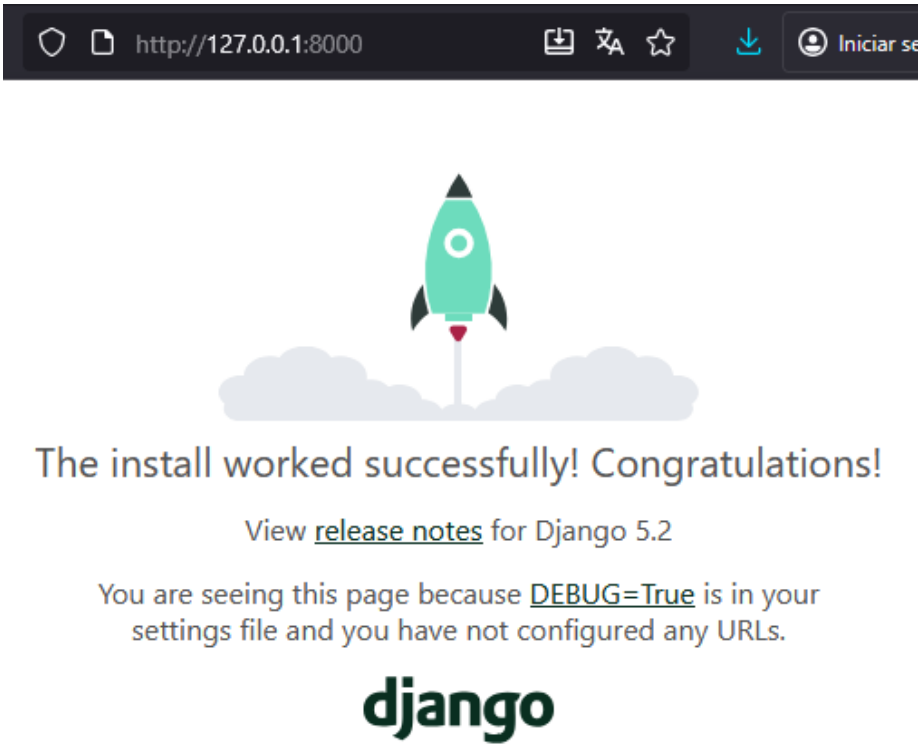


*Figura 2: Arrancar servidor con **python manage.py runserver***

comprobamos se arrancou correctamente buscando **https://127.0.0.1:8000** no **navegador**



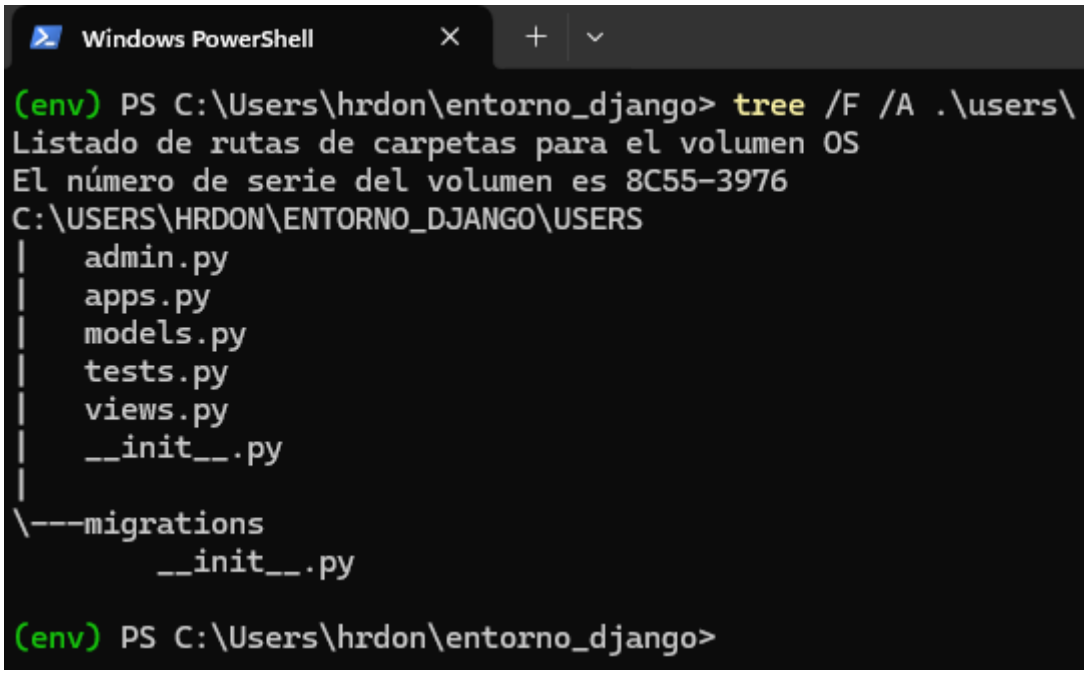
Logo para **deter o servidor** podemos facer un **Ctrl+C** ou simplemente **pechando a terminal**

4. Modelos

Agora que xa temos a web funcionando vamos crear un **modelo** para a nosa web. Django ten o seu propio sistema de **autenticación** e modelo para usuarios. Utilizarémolo para crear a nosa propia clase

Para crear o modelo de usuarios executaremos o seguinte comando:

python manage.py startup users



Aparte deses ficheiros en **entorno_django** tamén nos creou outro ficheiro chamado **db.sqlite3**

Preparar Vscode para Django

Si vscode non utiliza o **interprete correcto** de **Python** teremos que **indicarllo** nos.

Antes de nada comproba **aquí** que as versións de Python e Django sexan **compatibles**, recorda que neste manual estamos utilizando **Python 3.14** e **Django 5.2.8**

Agora vamos buscar a ruta do interprete de Python para indicarllo a vscode, para iso en **powershell** utiliza o comando **python -c "import sys; print(sys.executable)"**

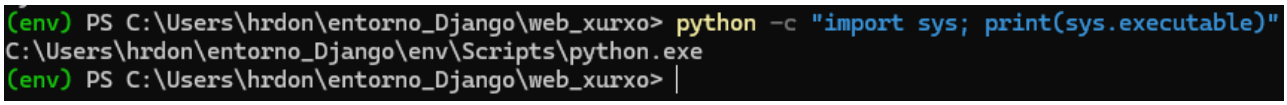
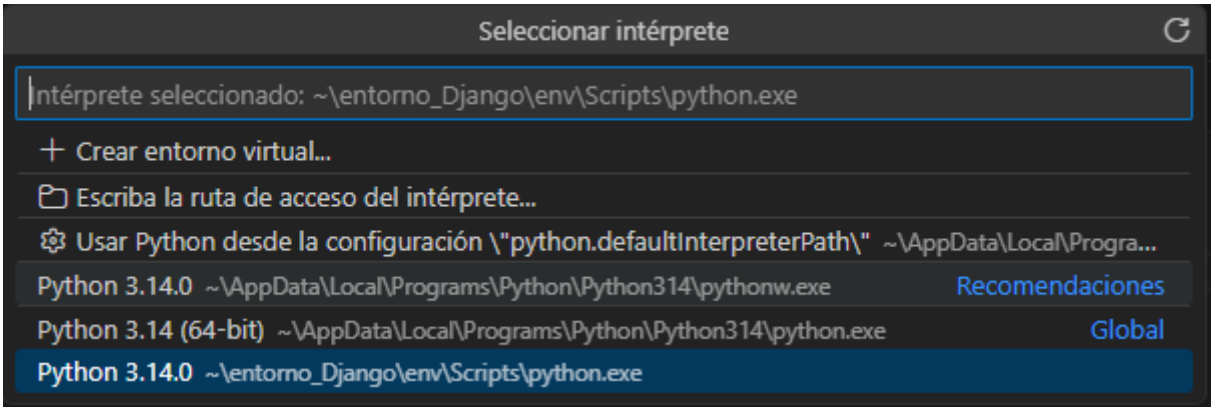


Figura 3: Obter ruta arquivo interprete de Python

A continuación dende **vscode** preme en **Ctrl+shift+P** e escribe **select interpreter** no panel de comandos e sairache un despregable coma o seguinte



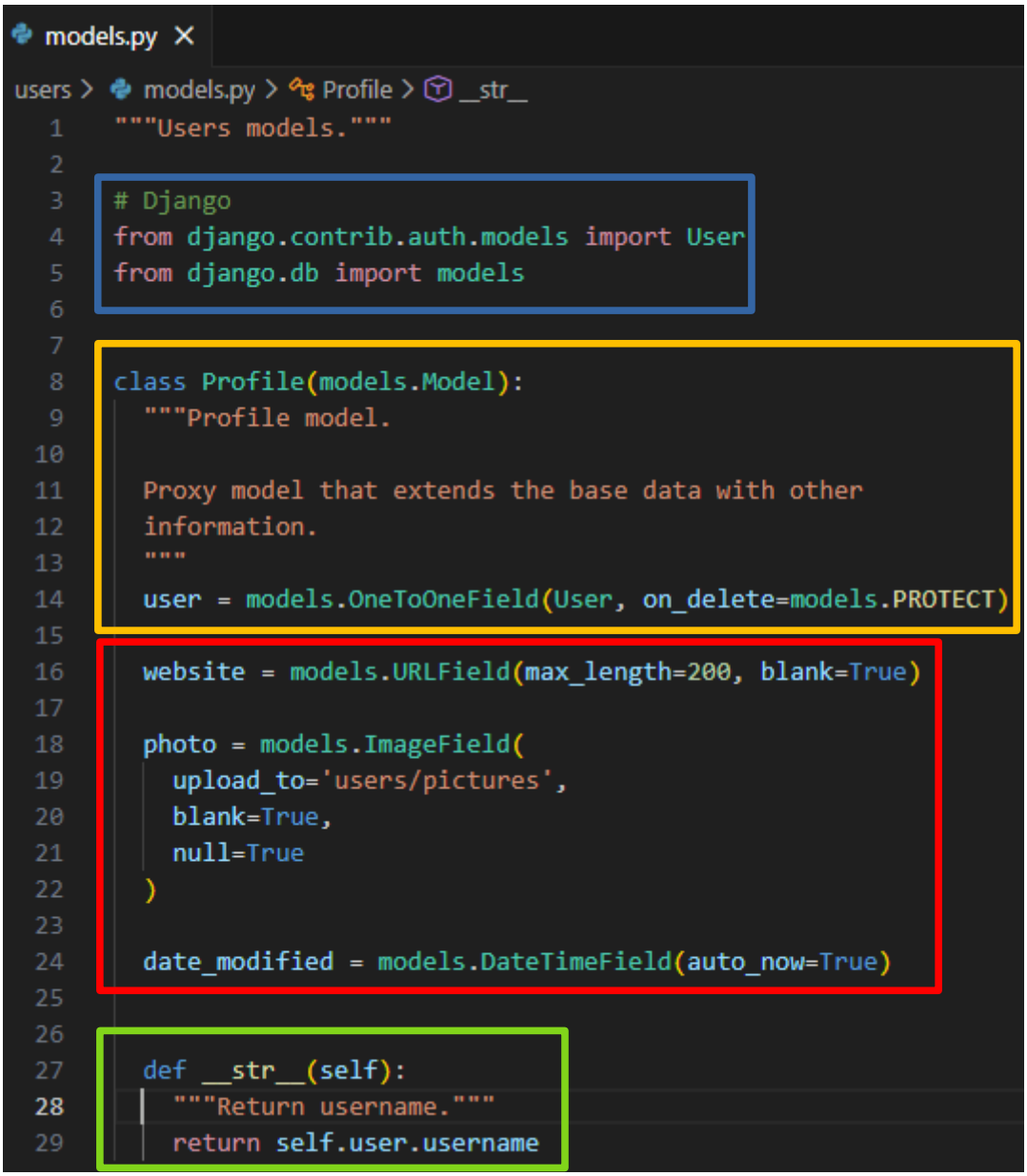
Preme en **Escriba la ruta de acceso del intérprete...** e pega a ruta do paso anterior

Por último **reinicia vscode** e debería funcionar correctamente

4.1. Usuarios

Unha vez estes pasos completados vamos co interesante, vamos modificar o ficheiro **users/models.py**

O ficheiro estará practicamente baleiro, engádelle o seguinte código:



O que fai este anaco de código é **importar a clase User** de Django, que xa nos prove de varios campos para a nosa táboa.

O seguinte que facemos na nosa clase **Profile** é relacionar a clase User coa nosa clase un a un e engadiremos a opción **on_delete=models.PROTECT** que impide que se eliminen usuarios, isto porque temos a opción de desactivas os usuarios, e xa que os usuarios están relacionados con varias táboas o eliminar usuarios poderíamos eliminar información importante.

A continuación personalizamos a clase e engadimos os **nosos propios campos**, para este proxecto vamos engadir a opción de que o usuario poida **gardar a súa website**, engadir unha **foto** e a **data de modificación** do usuario.

Por último engadimos o método **__str__** e retornamos o nome de usuario, isto utilizámolo para cando fagamos un print dun obxecto user nos amose o nome de usuario, o que nos dará máis información do obxecto a simple vista.

4.2. Crear perfiles para os usuarios

Agora vamos engadir o código para crear os perfiles de usuario cando creemos un usuario novo. Creamos o ficheiro **users/signals.py** e engade o seguinte código:

signals.py | apps.py | settings.py

users > signals.py > ...

```python
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from .models import Profile

@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()
```

A continuación dirixímonos a **users/apps.py** e comprobamos que temos o seguinte código:

```python
# apps.py
from django.apps import AppConfig


class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'

# Rexistrar sinais de users/signlas.py
from django.apps import AppConfig

class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'

    def ready(self):
        import users.signals
```

Por último vámonos a **web_xurxo/settings.py** e no array **INSTALLED_APPS** sustituimos **‘users’** por **‘ursers.apps.UserConfig’**

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Ritch text editor
    'ckeditor',
    # Modelos creados por nos
    'posts',
    # 'users',
    'users.apps.UsersConfig',
    'categories',
    'comments',
    'bootstrap4',
```

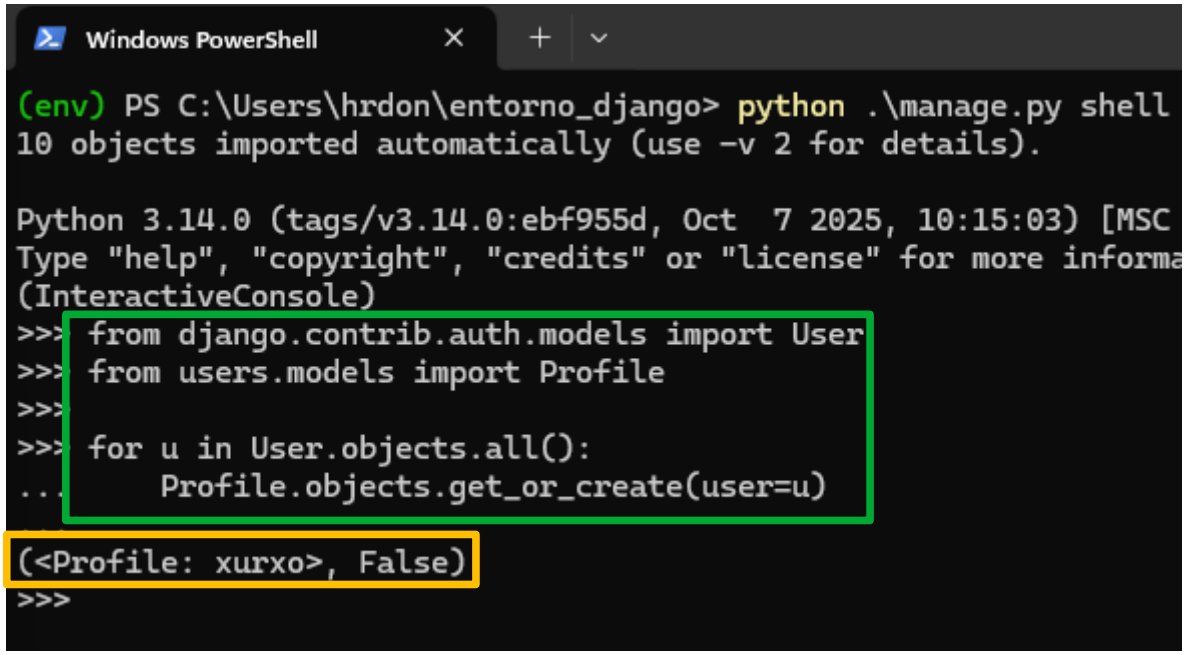
Co código que acabamos de engadir xa se crean os perfiles de usuario automaticamente cando creamos un usuario novo.

Para os usuarios existentes primeiro abrimos a **shell** de Django » **python manage.py shell**

E unha vez aberto a **shell** do noso entorno virtual engadimos o seguinte código:

```
from django.contrib.auth.models import User
from users.models import Profile

for u in User.objects.all():
    Profile.objects.get_or_create(user=u)
```

E prememos en **Enter** para executar o código e que cree os perfiles.



No **cadro amarelo** indícasenos que **non engadiu o perfil** a ese usuario, en principio porque **xa o tiña**, asi que vámolo comprobar:

Sen pechar o shell engade o seguinte código:

```
for u in User.objects.all():
    print(u.username, hasattr(u, 'profile'))
```

Ao executalo amosaranos a mensaxe do usuario **xurxo True**

```
>>> for u in User.objects.all():
...     print(u.username, hasattr(u, 'profile'))
...
xurxo True
>>>
```

E con isto xa temos solucionado o tema dos perfiles de usuario.

4.3. Post

Agora vamos co **modelo Post** e ao igual que con Users creamos un modelo para os **posts**, para os **comentarios** e outro para **categorías**

```
Windows PowerShell
(env) PS C:\Users\hrdon\entorno_django> python .\manage.py startapp posts
(env) PS C:\Users\hrdon\entorno_django> python .\manage.py startapp categories
(env) PS C:\Users\hrdon\entorno_django> python .\manage.py startapp comments
```

Logo instalamos o paquete **django-ckeditor** que necesitaremos máis adiante

```
Windows PowerShell
Windows PowerShell
(env) PS C:\Users\hrdon\entorno_Django\web_xurxo> pip install django-ckeditor
Collecting django-ckeditor
  Downloading django_ckeditor-6.7.3-py3-none-any.whl.metadata (33 kB)
Requirement already satisfied: Django>=3.2 in c:\users\hrdon\entorno_django\env\lib\site-packages (from django-ckeditor)
 (5.2.8)
Collecting django-js-asset>=2.0 (from django-ckeditor)
  Downloading django_js_asset-3.1.2-py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: asgiref>=3.8.1 in c:\users\hrdon\entorno_django\env\lib\site-packages (from Django>=3.2->
django-ckeditor) (3.10.0)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\hrdon\entorno_django\env\lib\site-packages (from Django>=3.2-
>django-ckeditor) (0.5.3)
Requirement already satisfied: tzdata in c:\users\hrdon\entorno_django\env\lib\site-packages (from Django>=3.2->django-c
keditor) (2025.2)
Downloading django_ckeditor-6.7.3-py3-none-any.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 27.4 MB/s  0:00:00
Downloading django_js_asset-3.1.2-py3-none-any.whl (5.9 kB)
Installing collected packages: django-js-asset, django-ckeditor
Successfully installed django-ckeditor-6.7.3 django-js-asset-3.1.2

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS C:\Users\hrdon\entorno_Django\web_xurxo>
```

Con todo instalado engadimos o seguinte código en **post/modesl.py**

models.py

posts > models.py > Post

```python
"""Posts models."""

# Django
from django.db import models

from django.utils.text import slugify
from django.contrib.auth.models import User
from ckeditor.fields import RichTextField
from categories.models import Category

class Post(models.Model):
    """Post model."""

    user = models.ForeignKey(User, on_delete=models.PROTECT)
    profile = models.ForeignKey('users.Profile', on_delete=models.PROTECT)

    title = models.CharField(max_length=255)
    image_header = models.ImageField(upload_to='posts/photos')
    post = RichTextField()

    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)
    is_draft = models.BooleanField(default=True)
    url = models.SlugField(max_length=255, unique=True)
    views = models.PositiveIntegerField(default=0)
    categories = models.ManyToManyField(Category)

    class Meta:
        ordering = ('title',)


    def __str__(self):
        """Return title and username."""
        return '{} by @{}'.format(self.title, self.user.username)


    def save(self, *args, **kwargs):
        self.url = slugify(self.title)
        super(Post, self).save(*args, **kwargs)
```

Para este modelo importamos:

- **slugify** que nos servirá para crear URL amigables en Django
- Clase **User** xa que un post será creado por un usuario
- Tipo de dato **RichTextField**, que é un campo que ven da librería que instalamos anteriormente

O seguinte que vamos facer será crear a clase **Post** e relacionar os usuarios cos posts coa función **ForeignKey**, será unha **relación 1 a N** xa que un usuario pode crear varios posts e un post pertence a un único usuario. Como no modelo du usuarios utilizaremos **PROTECT** xa que non queremos perder os posts.

Agora describiremos los demais campos:

- **title**: De tipo CharField e un tamaño máximo de 255 caracteres.
- **image_header**: Será a cabeceira do noso post gardarémolo en media/posts/photos.
- **post**: Será de tipo RichTextField que nos permitirá crear textos con distintos tamaños, tipos de letra, cores, etc.
- **date_created**: Data de creación. Gardarase a data actual por defecto polo atributo auto_now_add na creación.
- **date_modified**: Data de modificación. Gardarase a data actual por defecto cada vez que se actualice o post polo atributo auto_now.
- **is_draft**: Un campo de tipo booleano, guardaremos si es un borrador o lo queremos mostrar en la web.
- **url**: De tipo slugField, aquí guardaremos la url, se creará a partir del título.
- **views**: Número de visitas.
- **categories**: El listado de categorías asignadas a un post, es una relación de tipo N a N ya que un post puede tener varias categorías y una categoría puede pertenecer a varios posts.

Despois declaramos a clase **Meta** e engadimos o orde por título, isto fará que cando retornemos unha lista de obxectos tivo post, por defecto ordenará por post.

O método __str__ que xa o explicamos anteriormente

O método **save**. Aqui o que facemos é que cando se vaia gardar o **Post** pasamos a función **slugify** sobre o título e gardamolo no campo url.

4.4. Categorías

Este non ten ciencia ningunha, simplemente ten un campo **name**

models.py

categories > models.py > Category > __str__

```python
"""Categories"""

from django.db import models

# Models

# Create your models here.
class Category(models.Model):
    """Category model."""

    name = models.CharField(max_length=100,unique=True)

    class Meta:
        ordering = ('name',)

    def __str__(self):
        return self.name
```

4.5. Comentarios

Por último creamos o modelo para os **comentarios**:

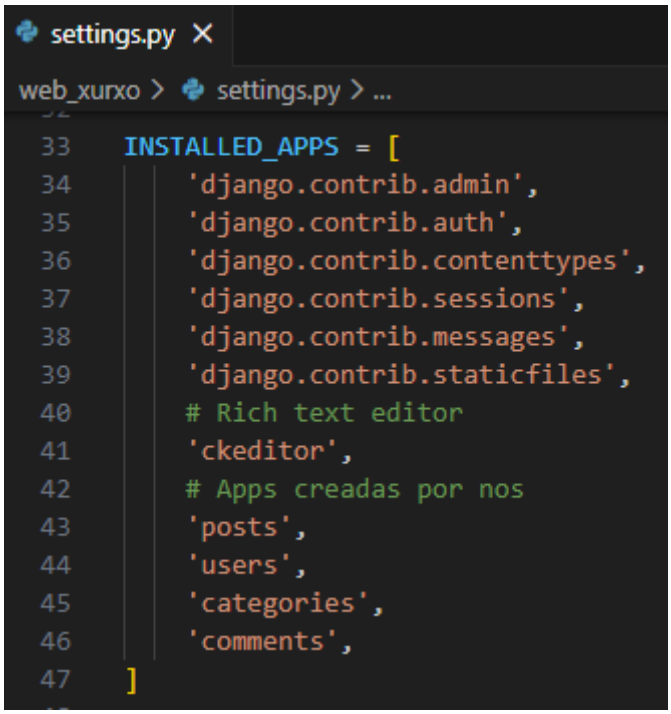
models.py

comments > models.py > Comment > __str__

```python
"""Comments"""

from django.db import models
from django.contrib.auth.models import User
from posts.models import Post

# Model

# Create your models here.
class Comment(models.Model):
    """Comment model."""

    user = models.ForeignKey(User, on_delete=models.PROTECT)
    profile = models.ForeignKey('users.Profile', on_delete=models.PROTECT)
    post = models.ForeignKey(Post, on_delete=models.PROTECT)
    comment = models.CharField(max_length=5000)

    def __str__(self):
        return self.comment
```

Con este código establecemos relacións 1-a-N entre comentarios, usuarios e publicacións, e evitamos cargar automaticamente tódolos comentarios dende o modelo de Post para mellorar o rendemento e obtelos só cando sexan necesarios.

A continuación modificamos o arquivo **web_xurxo/settings.py** e engadimos os modelos que acabamos de crear:



Unha vez creados os modelos executamos os comandos

- **python manage.py makemigrations**
- **python manage.py migrate**

```
(env) PS C:\Users\hrdon\entorno_Django\web_xurxo> python .\manage.py makemigrations
SystemCheckError: System check identified some issues:

ERRORS:
posts.Post.image_header: (fields.E210) Cannot use ImageField because Pillow is not installed.
        HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip install Pillow".
users.Profile.photo: (fields.E210) Cannot use ImageField because Pillow is not installed.
        HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip install Pillow".

WARNINGS:
```

Sáltanos **erro** porque **non** temos **instalado Pillow**

Simplemente executando o comando que nos indica (**python -m pip install Pillow**) solucionamos o problema

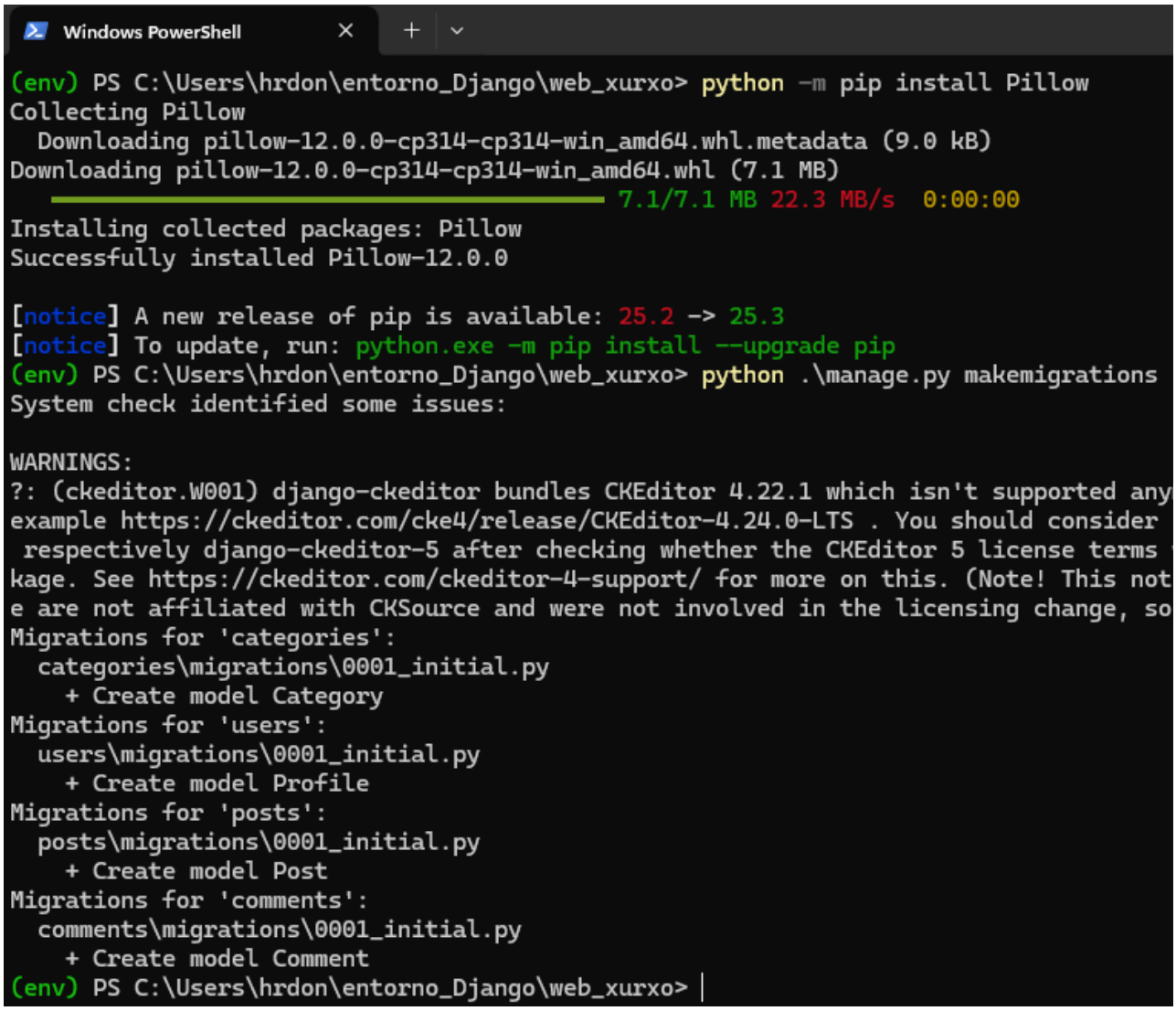


Figura 4: ***Instalación de pillow*** *e execución do comando* ***python manage.py makemigrations***

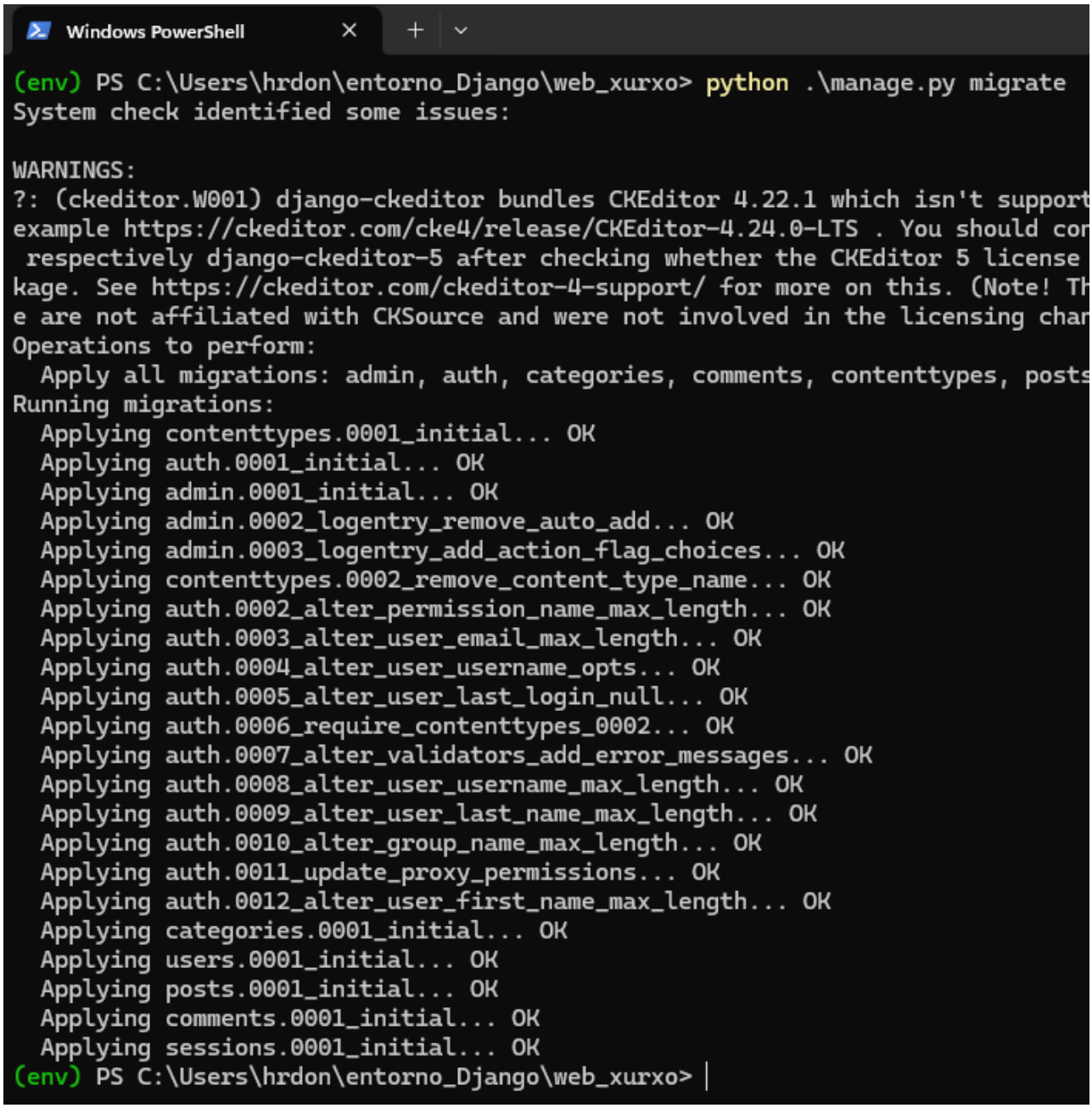


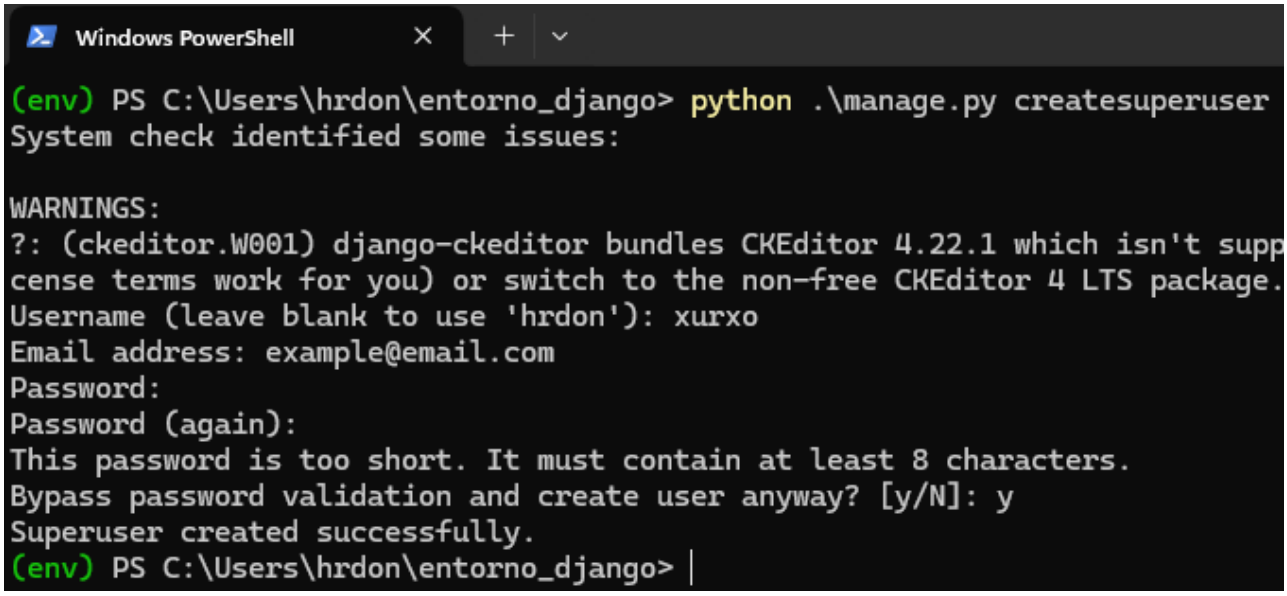
Figura 5: Execución do comando ***python manage.py migrate***

5. Panel de administración

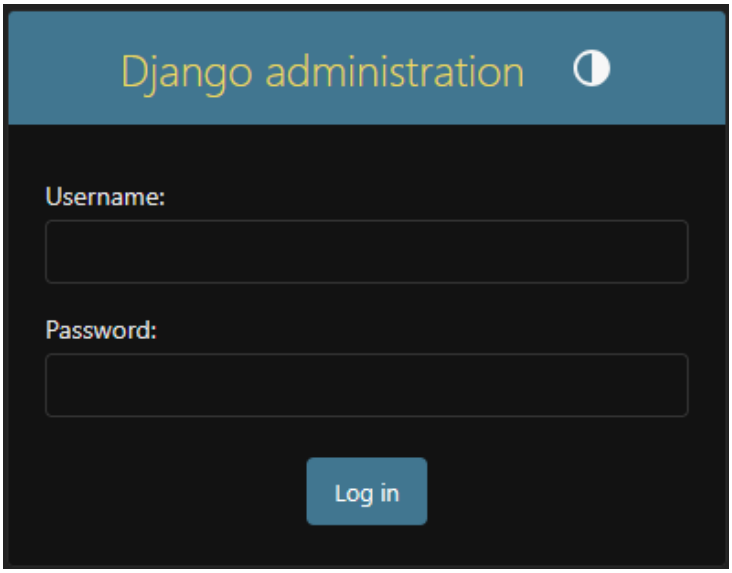
A interface de administración permítenos ler os metadatos dos modelos creados no noso proxecto e administrar o contido noso sitio web.

5.1. Super usuario

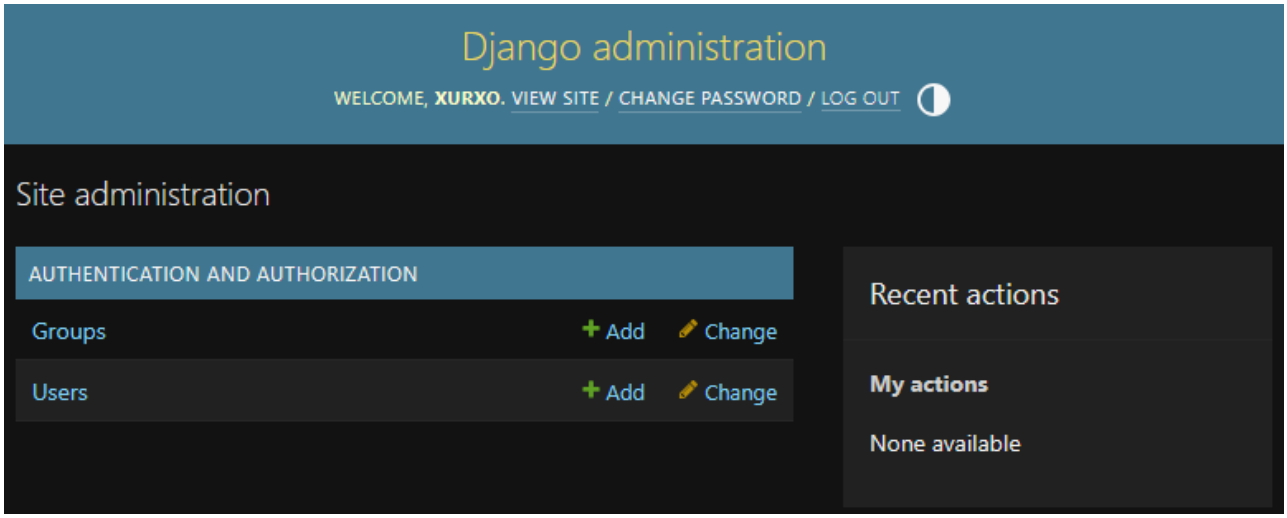
Para poder acceder ao panel primeiro teremos que crear un **súper usuario**



Unha vez creado o usuario accedemos a url http://127.0.0.1:8000/admin (**asegúrate de ter o servidor correndo**)



Ingresamos o **usuario** e **contrasinal** que acabamos de crear e listo



Dende aquí podemos engadir grupos e usuarios. Só se ven estes dous porque son os que veñen por defecto en Django

5.2. Administración – Usuarios

Abrimos o ficheiro **users/admin.py** e pegamos o seguinte código:

admin.py

users > admin.py > ProfileAdmin

```python
"""User admin classes."""

# Django
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from django.contrib import admin

# Models
from django.contrib.auth.models import User
from users.models import Profile


@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    """Profile admin."""

    list_display = ('pk', 'user', 'photo')
    list_display_links = ('pk', 'user',)
    list_editable = ('photo',)

    search_fields = (
        'user__email',
        'user__username',
        'user__first_name',
        'user__last_name',
    )

    list_filter = (
        'user__is_active',
        'user__is_staff',
        'date_modified',
    )

    fieldsets = (
        ('Profile', {
            'fields': (('user', 'photo', 'website'),),
        }),
        ('Extra info', {
            'fields': (('date_modified'),),
        })
    )

    readonly_fields = ('date_modified',)
```

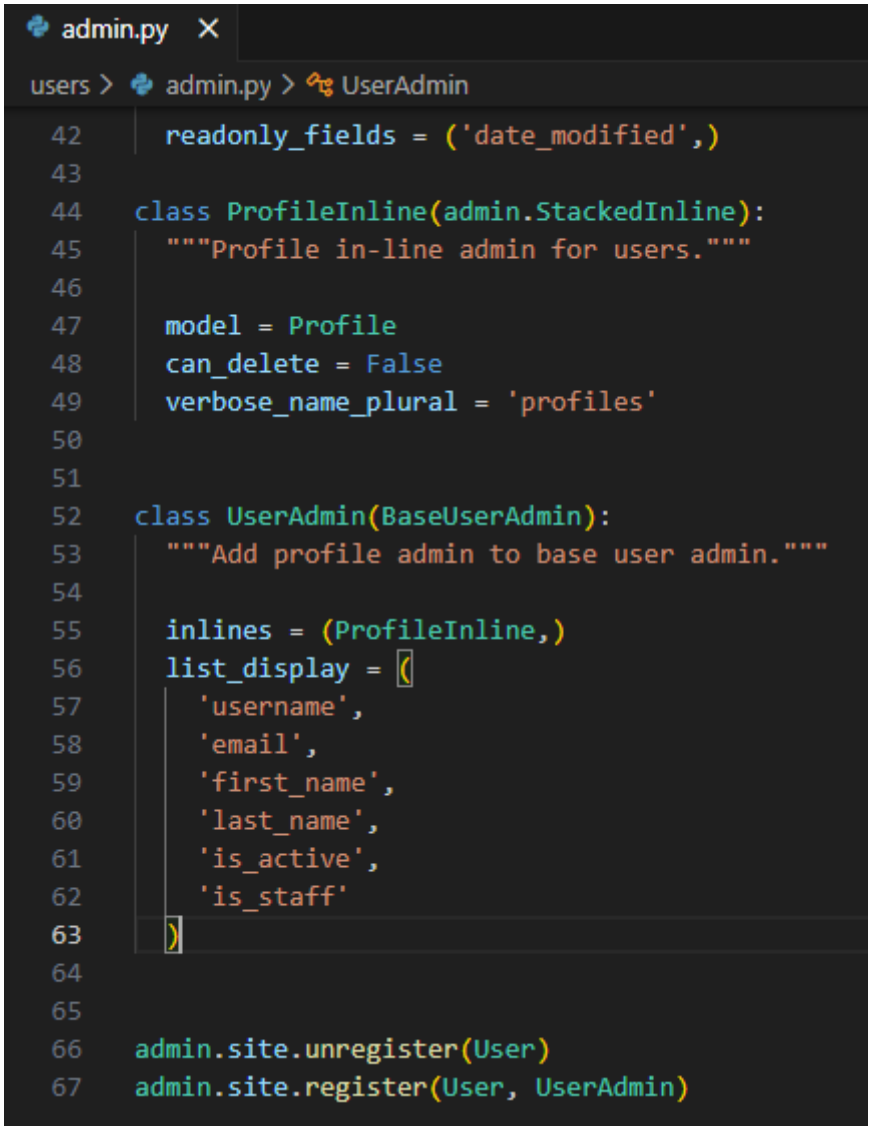
Con este código importamos a librería admin e os nosos modelos de **User** e **Profile**.

Con **admin.register(Profile)** rexistramos o noso modelo, o que fará que nos apareza no panel.

Despois creamos a nosa clase **ProfileAdmin**, por convección engádese ao final do nome a palabra **Admin** e aquí é onde vamos a meter configuracións personalizas:

- **list_display**: Amosaranos a lista de usuarios creados cando esteamos no apartado de profile, a información que se amosará é a que engadimos na lista, neste caso a primary key, o usuario e a foto.
- **list_display_links**: Engadirá un link para a edición nos campos que especifiquemos, neste caso na primary key e no usuario.
- **list_editable**: Esta funcionalidade permitiranos editar a foto dun usuario dende o panel de listados.
- **search_fields**: Permite a procura polos campos que engadas.
- **list_filter**: Engade un panel de filtros cos campos que engadimos.
- **field_sets**: Permite personalizar como se visualizarán os datos no panel, no noso caso, profile e metadata serían as cabeceiras dos campos e dentro de fields se os datos están dentro dunha tupla aparecerán horizontalmente (noso caso), senon verticalmente.
- **readonly_fields**: Os datos que engadas aquí amosaranse só como lectura.

Por último vamos crear a opción de crear usuarios dende o mesmo panel de control. Para iso utilizaremos a clase **StackedInline**. Engadimos o seguinte código ao final do arquivo **users/admin.py**



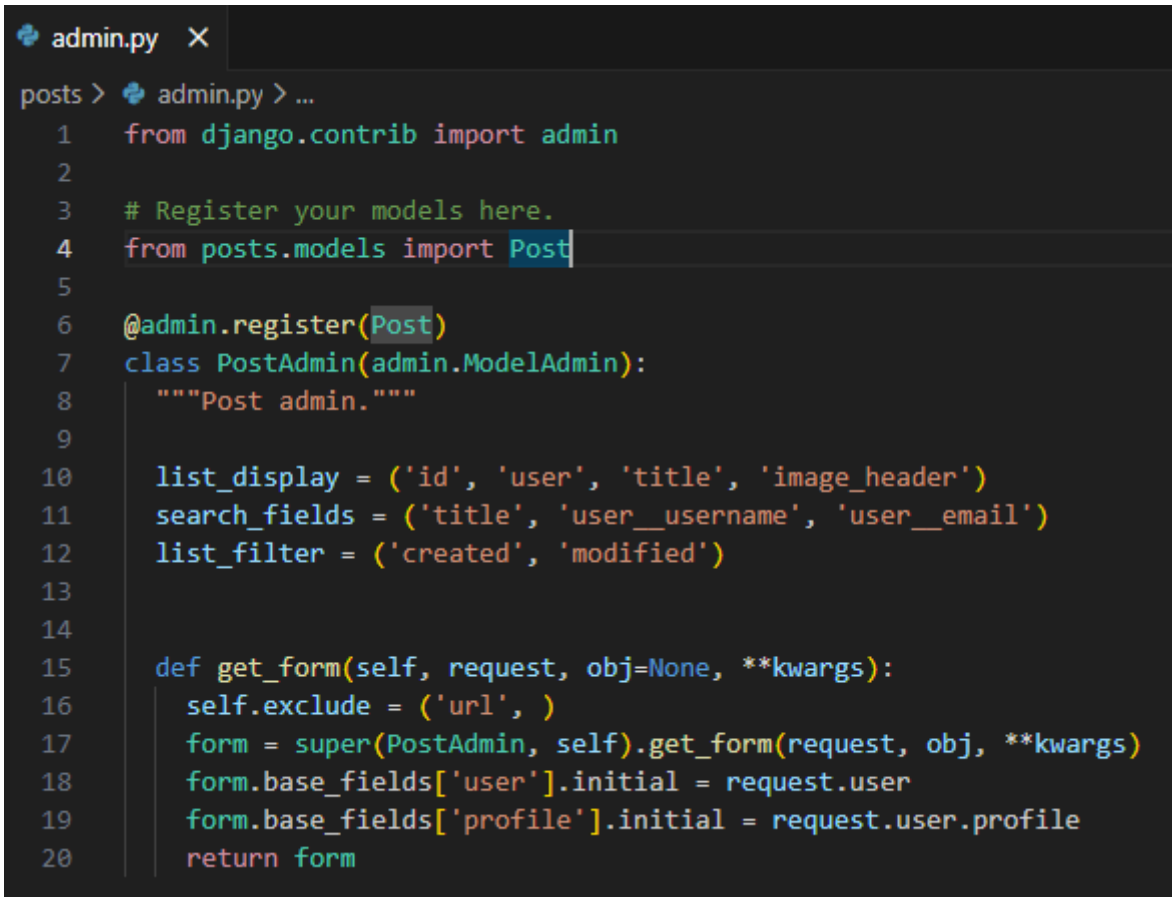
Neste código, primeiro creamos a clase **ProfileInline**, nesta gardamos o modelo que queremos engadir ao modelo pai.

Logo o que temos que facer é **quitar** a **clase User** que é a que está por defecto e rexistrar a nosa clase nova. Logo de declarala engadimos a clase de tipo **StackedInline** e no **list_display** metemos os datos que queremos que se amosen ao listar os usuarios.

Por último utilizamos o método **unregister** para quitar a nosa clase **User** e rexistramos a nova clase pasando o modelo base e a nova clase.

5.3. Administración – Post

Para crear os paneis de administración abrimos o ficheiro **web_xurxo/posts/admin.py** e pegamos o seguinte código

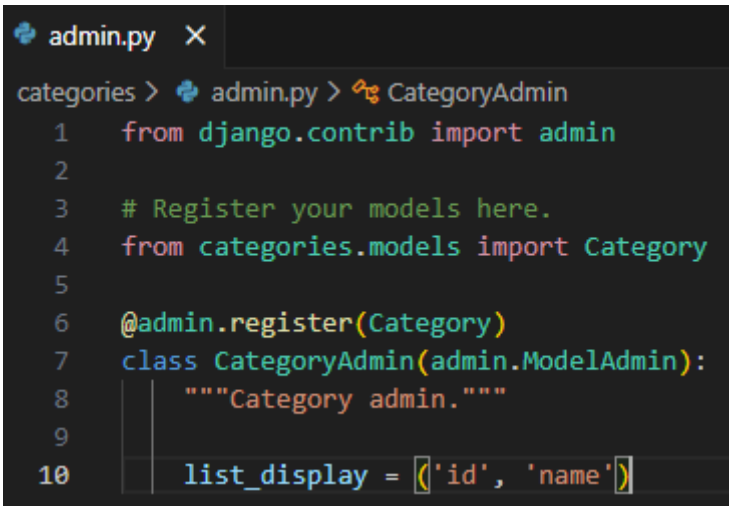


Neste caso engadimos o método **get_form** que nos permite realizar cambios no formulario antes de amosalo.

Os cambios que fixemos son: **ocultar a URL** co método **self.exclude** e que por defecto o creador do post sexa o usuario co que estás logueado.

5.4. Administración – Categorías

No modelo categories do noso proxecto abrimos o ficheiro de administración **web_xurxo/categories/admin.py**



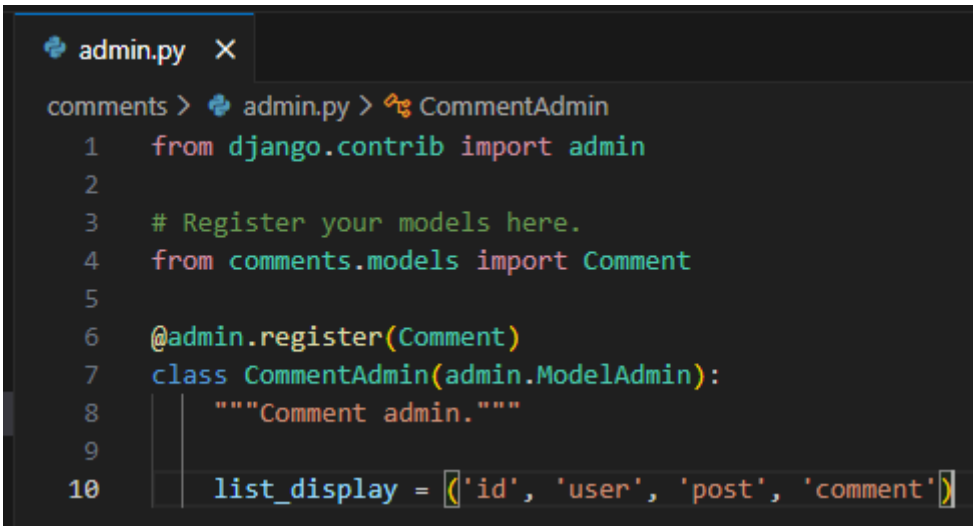
Usamos a clase **CategoryAdmin** para personalizar a administración e con **list_display** amosamos as columnas **id** e **name**.

Con isto podemos ver e modificar as categorías dende a interface de administración de Django.

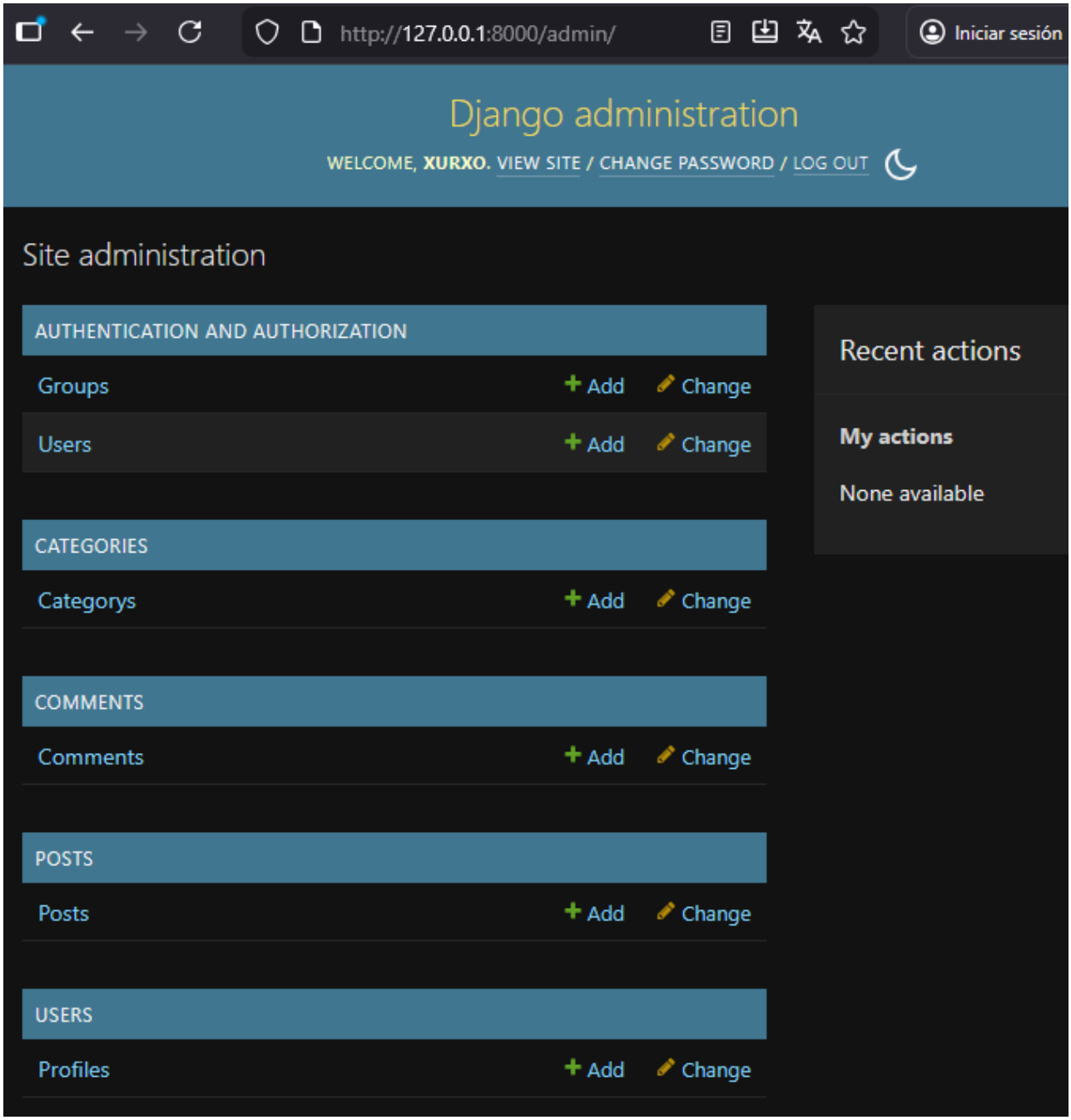
5.5. Administración – Comentarios

Neste caso utilizaremos o modelo **comments** para listar os comentarios de forma que podamos buscar comentarios que non cumpran a nosa política e así poder eliminalos.

Abrimos o ficheiro **web_xurxo/comments/admin.py** e pegamos o seguinte código:



Se agora (co servidor arrancado) nos diriximos ao panel de administración do noso Django deberíamos ver algo como o da páxina seguinte:


http://127.0.0.1:8000/admin/
Iniciar sesión
Django administration
WELCOME, XURXO. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
CATEGORIES
Categorys + Add Change
COMMENTS
Comments + Add Change
POSTS
Posts + Add Change
USERS
Profiles + Add Change
Recent actions
My actions
None available

6. Deseño

Utilizaremos **Bootstrap** xa que faremos énfase no framework de **Django** e non na parte frontend. Para traballar con templates vamos utilizar o **motor de plantillas de Django.**

O primeiro que temos que facer é **engadir a navegación** ao noso blog.

A nosa web terá unha **páxina principal** onde se amosarán todos os posts, unha páxina de **detalle** por **cada post**, unha de **login**, outra para o **rexistro** e a paxina "**sobre min**".

Para poder ver as páxinas que vamos crear necesitamos crealas e apuntar onde están para que cando se acceda mediante a url aparezan, para iso abrimos o arquivo **web_xurxo/urls.py** e pegamos o seguinte código:

urls.py

web_xurxo > urls.py > ...

```
1  """ ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.views.generic import TemplateView
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path(
24         route='',
25         view=TemplateView.as_view(template_name='posts/index.html'),
26         name='index'
27     ),
28     path(
29         route='post/my-post.html',
30         view=TemplateView.as_view(template_name='posts/detail.html'),
31         name='detail'
32     ),
33     path(
34         route='login',
35         view=TemplateView.as_view(template_name='users/login.html'),
36         name='login'
37     ),
```

```
38      path(
39          route='registro',
40          view=TemplateView.as_view(template_name='users/register.html'),
41          name='register'
42      ),
43      path(
44          route='sobre-mi',
45          view=TemplateView.as_view(template_name='about.html'),
46          name='about'
47      )
48  ]
```

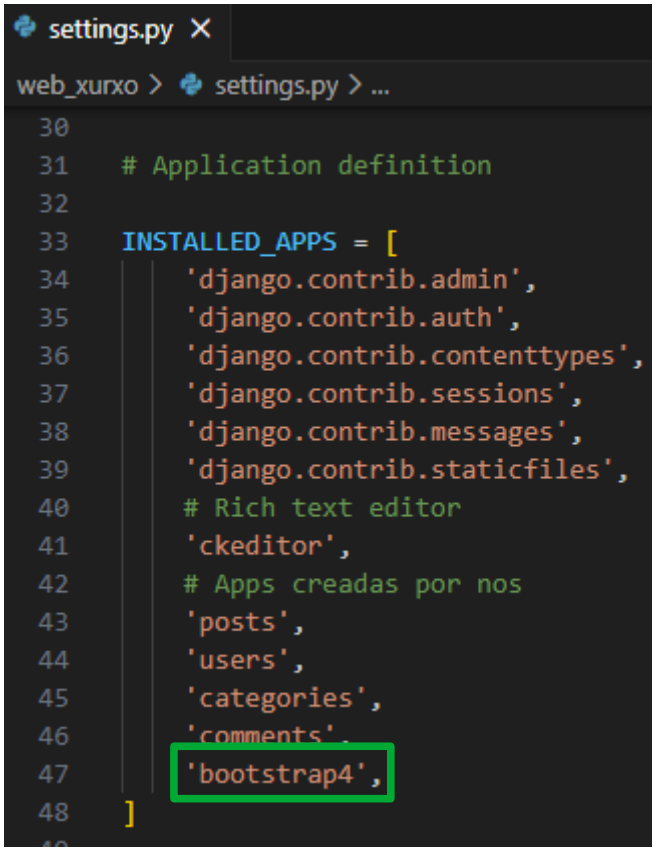
Engadimos unha **función** de tipo **path** na que gardamos a ruta que se debe escribir para acceder a nosa vista, o nome da plantilla e un nome que se utilizará posteriormente cando teñamos que engadir unha url nas nosas plantillas.

A continuación vamos descargar **bootstrap**, para iso executamos o seguinte comando:

```
Windows PowerShell
(env) PS C:\Users\hrdon\entorno_Django\web_xurxo> pip install django-bootstrap4
Collecting django-bootstrap4
  Downloading django_bootstrap4-25.3-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: django>=4.2 in c:\users\hrdon\entorno_django\env\lib\site-packages (from djang
4) (5.2.8)
Collecting beautifulsoup4>=4.10.0 (from django-bootstrap4)
  Downloading beautifulsoup4-4.14.2-py3-none-any.whl.metadata (3.8 kB)
Collecting soupsieve>1.2 (from beautifulsoup4>=4.10.0->django-bootstrap4)
  Downloading soupsieve-2.8-py3-none-any.whl.metadata (4.6 kB)
Collecting typing-extensions>=4.0.0 (from beautifulsoup4>=4.10.0->django-bootstrap4)
  Downloading typing_extensions-4.15.0-py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: asgiref>=3.8.1 in c:\users\hrdon\entorno_django\env\lib\site-packages (from dj
django-bootstrap4) (3.10.0)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\hrdon\entorno_django\env\lib\site-packages (from d
>django-bootstrap4) (0.5.3)
Requirement already satisfied: tzdata in c:\users\hrdon\entorno_django\env\lib\site-packages (from django>=4.
ootstrap4) (2025.2)
Downloading django_bootstrap4-25.3-py3-none-any.whl (25 kB)
Downloading beautifulsoup4-4.14.2-py3-none-any.whl (106 kB)
Downloading soupsieve-2.8-py3-none-any.whl (36 kB)
Downloading typing_extensions-4.15.0-py3-none-any.whl (44 kB)
Installing collected packages: typing-extensions, soupsieve, beautifulsoup4, django-bootstrap4
Successfully installed beautifulsoup4-4.14.2 django-bootstrap4-25.3 soupsieve-2.8 typing-extensions-4.15.0

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS C:\Users\hrdon\entorno_Django\web_xurxo>
```

Figura 6: Instalación de Bootstrap 4 en Django

E engadimos a app ao noso arquivo **web_xurxo/settings.py**



No mesmo arquivo engadimos a ruta onde se gardarán as plantillas, para isto temos que editar a constante **TEMPLATES**:

- importamos **os**: módulo estándar de Python que permite interactuar co sistema operativo
- Definir **BASE_DIR**: variable que apunta ó cartafol raíz do noso proxecto Django

settings.py

web_xurxo > settings.py > ...

```python
import os

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR / 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Seguimos modificando o mesmo arquivo. Agora engadimos o seguinte código:

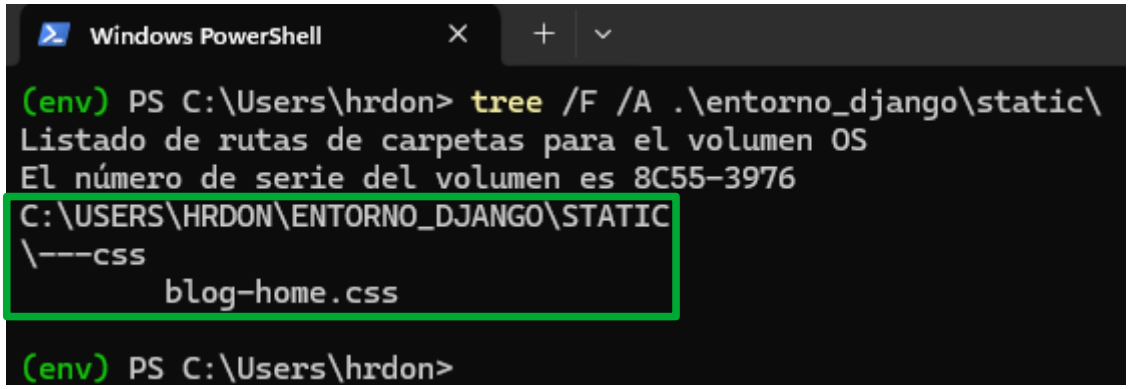
settings.py

web_xurxo > settings.py > ...

```python
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = (
  os.path.join(BASE_DIR / 'static'),
)
STATICFILES_FINDERS = [
  'django.contrib.staticfiles.finders.FileSystemFinder',
  'django.contrib.staticfiles.finders.AppDirectoriesFinder'
]
```

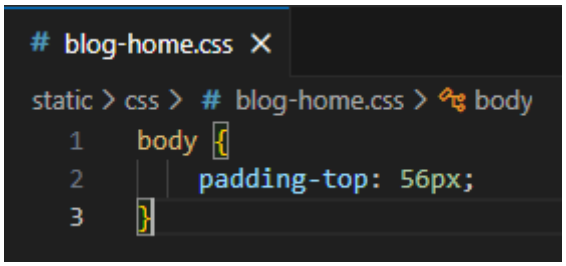
Para indicarlle a Django onde gardaremos os arquivos estáticos (css, js, imaxes, etc.)

Unha vez gardados os cambios na raíz do noso proxecto creamos o cartafol **static** e dentro desta o cartafol **css** que conterá todalas follas de estilo que creemos, nesta carpeta creamos un arquivo **blog-home.css**. A estrutura debería verse asi:



Para este proxecto utilizaremos esta plantilla creada por startbootstrap, utilizaremolo no seguinte paso (Login)

Por último engadimos o seguinte código no ficheiro css que acabamos de crear

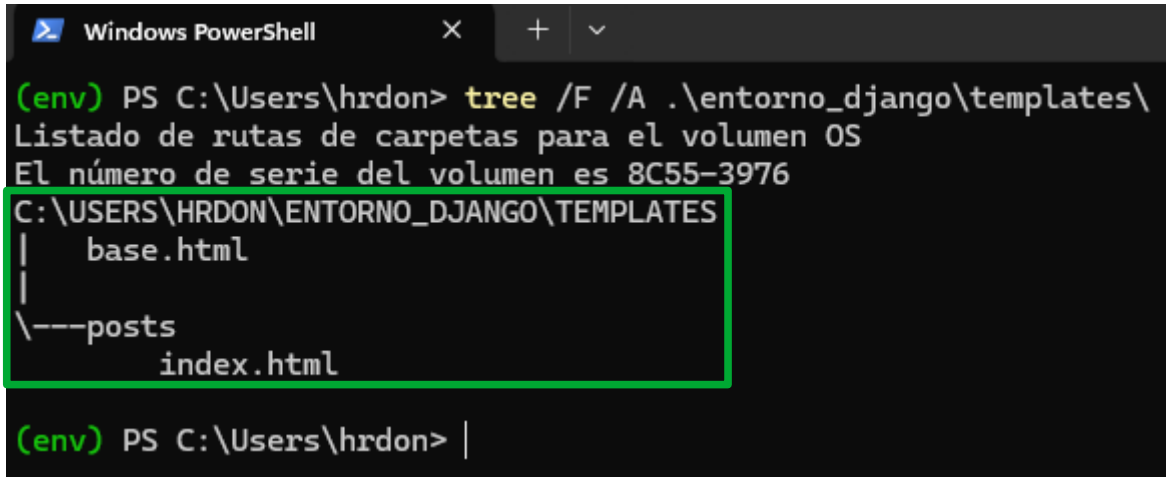


Despois, no directorio raíz, creamos a carpeta **templates** que será na que gardemos tódalas plantillas que utilicemos.

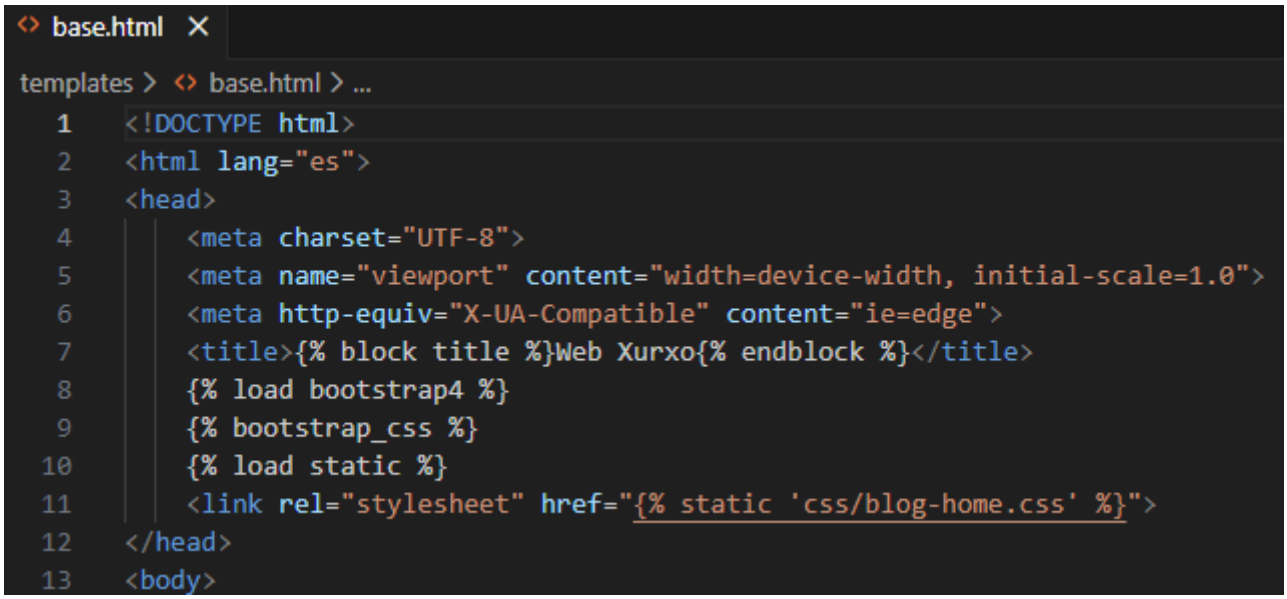
6.1. Index

Dentro da carpeta **templates** que acabamos de crear, creamos un arquivo chamado **base.html** que será o que conteña a estrutura básica da web.

Crearemos tamén unha carpeta chamada **posts** e dentro dela un arquivo chamado **index.html** que será a páxina principal do blog. A estrutura será a seguinte:



Unha vez creada a estrutura de carpetas vamos crear as plantillas. Abrimos o arquivo **base.html** e engadimos o seguinte código:



```
14
15    <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
16      <div class="container">
17        <a class="navbar-brand" href="#">Web xurxo</a>
18        <button class="navbar-toggler" type="button" data-toggle="collapse"
          data-target="#navbarResponsive" aria-controls="navbarResponsive"
          aria-expanded="false" aria-label="Toggle navigation">
19          <span class="navbar-toggler-icon"></span>
20        </button>
21        <div class="collapse navbar-collapse" id="navbarResponsive">
22          <ul class="navbar-nav ml-auto">
23            <li class="nav-item active">
24              <a class="nav-link" href="#">Blog
25                <span class="sr-only">(current)</span>
26              </a>
27            </li>
28            <li class="nav-item">
29              <a class="nav-link" href="#">Sobre min</a>
30            </li>
31            <li class="nav-item">
32              <a class="nav-link" href="#">Login</a>
33            </li>
34            <li class="nav-item">
35              <a class="nav-link" href="#">Rexistro</a>
36            </li>
37          </ul>
38        </div>
39      </div>
40    </nav>
41
42
43    <div class="container">
44
45  {% block content %}{% endblock %}
46
47        <!-- Sidebar Widgets Column -->
48        <div class="col-md-4">
49          <!-- Categories Widget -->
50          <div class="card my-4">
51            <h5 class="card-header">Categorias</h5>
52            <div class="card-body">
53              <div class="row">
54                <div class="col-lg-6">
55                  <ul class="list-unstyled mb-0">
56                    <li>
```

```
57                      <a href="#">Deseño web</a>
58                    </li>
59                    <li>
60                      <a href="#">HTML</a>
61                    </li>
62                    <li>
63                      <a href="#">Cortesía</a>
64                    </li>
65                  </ul>
66                </div>
67                <div class="col-lg-6">
68                  <ul class="list-unstyled mb-0">
69                    <li>
70                      <a href="#">JavaScript</a>
71                    </li>
72                    <li>
73                      <a href="#">CSS</a>
74                    </li>
75                    <li>
76                      <a href="#">Tutoriales</a>
77                    </li>
78                  </ul>
79                </div>
80              </div>
81            </div>
82          </div>
83
84
85          <div class="card my-4">
86            <h5 class="card-header">Widget lateral</h5>
87            <div class="card-body">
88              Puedes colocar lo que quieras dentro de estos widgets
                laterales. Son fáciles de usar e incluyen los nuevos
                contenedores de tarjetas de Bootstrap 4!
89            </div>
90          </div>
91
92        </div>
93
94      </div>
95      <footer class="py-5 bg-dark">
96          <div class="container">
97          <p class="m-0 text-center text-white">Copyright &copy; Simple Blog
            {% now "Y" %}</p>
98      </div>
99      </footer>
100     {% bootstrap_javascript jquery='full' %}
101 </body>
102 </html>
```

Agora que temos a nosa plantilla base vamos ver para que sirve cada **elemento** de **Django template** que usamos.

```
<title>{% block title %}Web Xurxo{% endblock %}</title>
```

Na anterior imaxe definimos un bloque de plantilla para o **<title>** que as páxinas fillas poden sobrescribir ao estender **base.html**

```
{% load bootstrap4 %}
{% bootstrap_css %}
```

O código de aquí arriba **carga** a librería **bootstrap4** que instalamos (figura 6) anteriormente en Django e engade o CSS de Bootstrap a nosa plantilla

- {% load bootstrap4 %} » activa as etiquetas da librería django-bootstrap4
- {% bootstrap_css %} » inserta automaticamente o enlace do arquivo CSS de Bootstrap

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/blog-home.css' %}">
```

Este código **carga** o sistema de arquivos estáticos de Django e logo inserta no **<link>** a url correcta do arquivo **css/blog-home.css** dentro da nosa carpeta **static**

```
{% bootstrap_javascript jquery='full' %}
```

Este último bloque **inserta** automaticamente o **JavaScript de Bootstrap** na nosa plantilla e ademais carga **jQuery completo** como dependencia.

Agora que xa temos a nosa plantilla base configurada engadiremos o seguinte código a nosa plantilla **index.html**

index.html × base.html

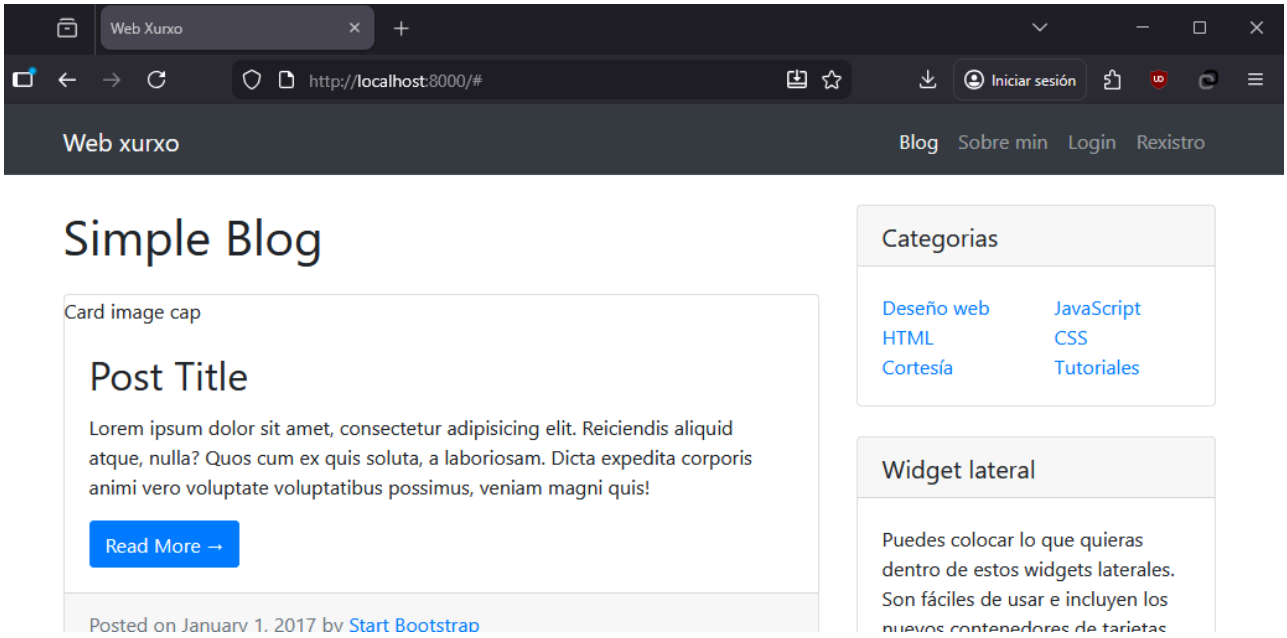
templates > posts > index.html > div.row > div.col-md-8

```
{% extends "base.html" %}
{% block content %}
    <div class="row">
      <div class="col-md-8">

        <h1 class="my-4">Simple Blog
          {% comment %} <small>Other text</small> {% endcomment %}
        </h1>

        <div class="card mb-4">
          <img class="card-img-top" src="http://placehold.it/750x300"
          alt="Card image cap">
          <div class="card-body">
            <h2 class="card-title">Post Title</h2>
            <p class="card-text">Lorem ipsum dolor sit amet, consectetur
            adipisicing elit. Reiciendis aliquid atque, nulla? Quos cum ex
            quis soluta, a laboriosam. Dicta expedita corporis animi vero
            voluptate voluptatibus possimus, veniam magni quis!</p>
            <a href="#" class="btn btn-primary">Read More &rarr;</a>
          </div>
          <div class="card-footer text-muted">
            Posted on January 1, 2017 by
            <a href="#">Start Bootstrap</a>
          </div>
        </div>

        <div class="card mb-4">
          <img class="card-img-top" src="http://placehold.it/750x300"
          alt="Card image cap">
          <div class="card-body">
            <h2 class="card-title">Post Title</h2>
            <p class="card-text">Lorem ipsum dolor sit amet, consectetur
            adipisicing elit. Reiciendis aliquid atque, nulla? Quos cum ex
            quis soluta, a laboriosam. Dicta expedita corporis animi vero
            voluptate voluptatibus possimus, veniam magni quis!</p>
            <a href="#" class="btn btn-primary">Read More &rarr;</a>
          </div>
```

```
30              <div class="card-footer text-muted">
31                Posted on January 1, 2017 by
32                <a href="#">Start Bootstrap</a>
33              </div>
34            </div>
35
36            <ul class="pagination justify-content-center mb-4">
37              <li class="page-item">
38                <a class="page-link" href="#">&larr; Older</a>
39              </li>
40              <li class="page-item disabled">
41                <a class="page-link" href="#">Newer &rarr;</a>
42              </li>
43            </ul>
44
45          </div>
46    {% endblock %}
```

Aquí o que facemos é **estender** da plantilla **base.html** e no bloque **content** engadimos o corpo da nosa páxina.

Agora se arrancamos o servidor e buscamos **localhost:8000** no navegador verase a nosa web



Agora que temos a páxina visible, vamos crear o detalle dun post, para iso dentro de **templates/post** creamos o arquivo **detial.html** e engadimos o seguinte código

detail.html

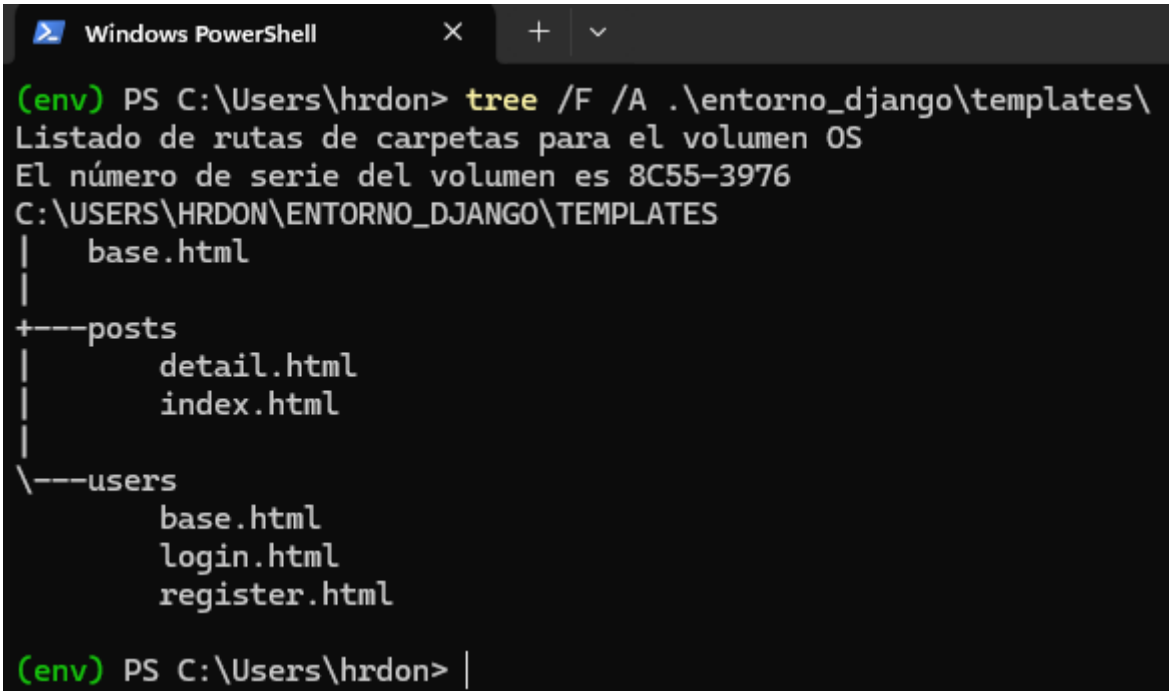
templates > posts > detail.html > div.container > div.row

```
{% extends "base.html" %}
{% block content %}
  <div class="container">
    <div class="row">
      <div class="col-lg-8">
        <h1 class="mt-4">Post Title</h1>
        <p class="lead">
          by
          <a href="#">Start Bootstrap</a>
        </p>
        <hr>
        <p>Posted on January 1, 2019 at 12:00 PM</p>
        <hr>
        <img class="img-fluid rounded" src="http://placehold.it/900x300"
        alt="">
        <hr>
        <p class="lead">Lorem ipsum dolor sit amet, consectetur adipisicing
        elit. Ducimus, vero, obcaecati, aut, error quam sapiente nemo saepe
        quibusdam sit excepturi nam quia corporis eligendi eos magni
        recusandae laborum minus inventore?</p>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut,
        tenetur natus doloremque laborum quos iste ipsum rerum obcaecati
        impedit odit illo dolorum ab tempora nihil dicta earum fugiat.
        Temporibus, voluptatibus.</p>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos,
        doloribus, dolorem iusto blanditiis unde eius illum consequuntur
        neque dicta incidunt ullam ea hic porro optio ratione repellat
        perspiciatis. Enim, iure!</p>
        <blockquote class="blockquote">
          <p class="mb-0">Lorem ipsum dolor sit amet, consectetur
          adipiscing elit. Integer posuere erat a ante.</p>
          <footer class="blockquote-footer">Someone famous in
            <cite title="Source Title">Source Title</cite>
          </footer>
        </blockquote>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error,
        nostrum, aliquid, animi, ut quas placeat totam sunt tempora commodi
        nihil ullam alias modi dicta saepe minima ab quo voluptatem
        obcaecati?</p>
```

```
26          <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Harum,
            dolor quis. Sunt, ut, explicabo, aliquam tenetur ratione tempore
            quidem voluptates cupiditate voluptas illo saepe quaerat numquam
            recusandae? Qui, necessitatibus, est!</p>
27          <hr>
28          <div class="card my-4">
29            <h5 class="card-header">Leave a Comment:</h5>
30            <div class="card-body">
31              <form>
32                <div class="form-group">
33                  <textarea class="form-control" rows="3"></textarea>
34                </div>
35                <button type="submit" class="btn btn-primary">Submit</button>
36              </form>
37            </div>
38          </div>
39          <div class="media mb-4">
40            <img class="d-flex mr-3 rounded-circle" src="http://placehold.it/
              50x50" alt="">
41            <div class="media-body">
42              <h5 class="mt-0">Commenter Name</h5>
43              Cras sit amet nibh libero, in gravida nulla. Nulla vel metus
                scelerisque ante sollicitudin. Cras purus odio, vestibulum in
                vulputate at, tempus viverra turpis. Fusce condimentum nunc ac
                nisi vulputate fringilla. Donec lacinia congue felis in
                faucibus.
44            </div>
45          </div>
46
47        </div>
48    {% endblock %}
```

Logo de pegar o código vamos encargarnos da parte de login. O primeiro que vamos facer é: no noso directorio **templates** engadir o cartafol **users**, dentro desta crearemos 3 arquivos que se chamarán **base.html, login.html** e **register.html**.

A estrutura de cartafoles debe quedar algo asi:



6.2. Post » base.html, login.html y register.html

base.html

templates > users > base.html > html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %}Simple blog - Login{% endblock %}</title>
    {% load bootstrap4 %}
    {% bootstrap_css %}
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/signin.css' %}">
</head>
<body class="text-center">
    {% block content %}{% endblock %}
    {% bootstrap_javascript jquery='full' %}
</body>
</html>
```

Figura 7: base.html

login.html

templates > users > login.html > ...

```
{% extends "users/base.html" %}
{% block content %}
<form class="form-signin">
    <h1><a href="#" class="title-link">Simple Blog</a></h1>
    <h1 class="h3 mb-3 font-weight-normal">Login</h1>
    <label for="inputEmail" class="sr-only">Email</label>
    <input type="email" id="inputEmail" class="form-control"
    placeholder="Email" required autofocus>
    <label for="inputPassword" class="sr-only">Contraseña</label>
    <input type="password" id="inputPassword" class="form-control"
    placeholder="Contraseña" required>
    <div class="checkbox mb-3"></div>
    <button class="btn btn-lg btn-primary btn-block" type="submit">Login</
    button>
    <p class="mt-5 mb-3 text-muted">&copy; Simple Blog {% now "Y" %}</p>
    </form>
{% endblock %}
```

Figura 8: login.html

register.html

templates > users > register.html > ...

```
1   {% extends "users/base.html" %}
2   {% block title %}Simple blog - Registro{% endblock %}
3   {% block content %}
4   <form class="form-signin">
5       <h1><a href="#" class="title-link">Simple Blog</a></h1>
6       <h1 class="h3 mb-3 font-weight-normal">Registro</h1>
7       <label for="inputEmail" class="sr-only">Email</label>
8       <input type="email" id="inputEmail" class="form-control"
        placeholder="Email address" required autofocus>
9       <label for="inputUsername" class="sr-only">Nombre de usuario</label>
10      <input type="text" id="inputUsername" class="form-control"
        placeholder="Nombre de usuario" required>
11      <label for="inputPassword" class="sr-only">Contraseña</label>
12      <input type="password" id="inputPassword" class="form-control"
        placeholder="Contraseña" required>
13      <label for="inputPassword" class="sr-only">Repetir contraseña</label>
14      <input type="password" id="inputPasswordRepeat" class="form-control"
        placeholder="Repetir contraseña" required>
15      <div class="checkbox mb-3"></div>
16
17      <button class="btn btn-lg btn-primary btn-block"
        type="submit">Registrarse</button>
18      <p class="mt-5 mb-3 text-muted">&copy; Simple Blog {% now "Y" %}</p>
19      </form>
20  {% endblock %}
```

Figura 9: register.html

Dentro de **static/css** creamos o arquivo **signin.css** e engadimos o seguinte código:

signin.css

static > css > # signin.css > ...

```
html,
body {
    height: 100%;
}

body {
    display: -ms-flexbox;
    display: flex;
    -ms-flex-align: center;
    align-items: center;
    padding-top: 40px;
    padding-bottom: 40px;
    background-color: #f5f5f5;
}

.form-signin {
    width: 100%;
    max-width: 330px;
    padding: 15px;
    margin: auto;
}

.form-signin .checkbox {
    font-weight: 400;
}

.form-signin .form-control {
    position: relative;
    box-sizing: border-box;
    height: auto;
    padding: 10px;
    font-size: 16px;
}

.form-signin .form-control:focus {
    z-index: 2;
}

.form-signin input[type="email"] {
    border-bottom-right-radius: 0;
    border-bottom-left-radius: 0;
}

.form-signin input[type="text"] {
    border-radius: 0;
}

.form-signin input[type="password"] {
    border-radius: 0;
}

#inputPasswordRepeat {
    border-top-left-radius: 0;
    border-top-right-radius: 0;
}

.title-link {
    color: #000;
    cursor: pointer;
}

.title-link:hover {
    color: #000;
    text-decoration: none;
}
```

Agora só nos falta a paxina **sobre min**, para iso dentro da carpeta **templates** creamos o arquivo **about.html** e engadimos o seguinte código:

about.html

templates > about.html > div.container > div.row

```
{% extends "base.html" %}
{% block title %}Simple blog - Sobre mi{% endblock %}
{% block content %}
  <div class="container">
    <div class="row">
      <div class="col-lg-8">
        <h1 class="mt-4">Sobre mi</h1>
        <hr>
        <img class="img-fluid rounded" src="http://placehold.it/150x200" alt="">
        <hr>
        <p class="lead">Lorem ipsum dolor sit amet, consectetur adipisicing elit.
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos, doloribus
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos, doloribus
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos, doloribus
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error, nostrum
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Harum, dolorem
        <hr>
      </div>
{% endblock %}
```

Agora que xa temos as páxinas listas vamonos ao ficheiro **templates/base.html** e engadimos as urls das páxinas na cabeceira co bloque **{% url 'home' %}**

```
<div class="collapse navbar-collapse" id="navbarResponsive">
  <ul class="navbar-nav ml-auto">
    <li class="nav-item active">
      <a class="nav-link" href="{% url 'index' %}">Blog
        <span class="sr-only">(current)</span>
      </a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'about' %}">Sobre min</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'users:login' %}">Login</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'users:register' %}">Rexistro</a>
    </li>
  </ul>
</div>
```

A sintaxe dos bloques seria a seguinte:

url -> refírese ao ficheiro **urls.py** do directorio **web_xurxo**

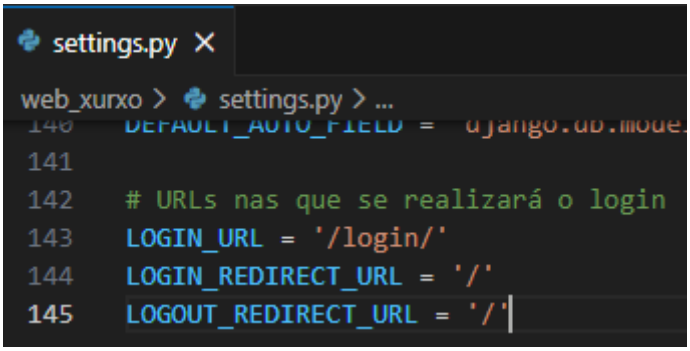
'index'-> nome que asignamos no **path** do array **urlpatterns**

users:register -> chamamos aos enlaces de **users/urls.py** que incluiremos no noso **web_xurxo/urls.py** máis adiante

7. Login

Para esta parte utilizaremos o sistema de autenticación e autorización que nos prove Django.

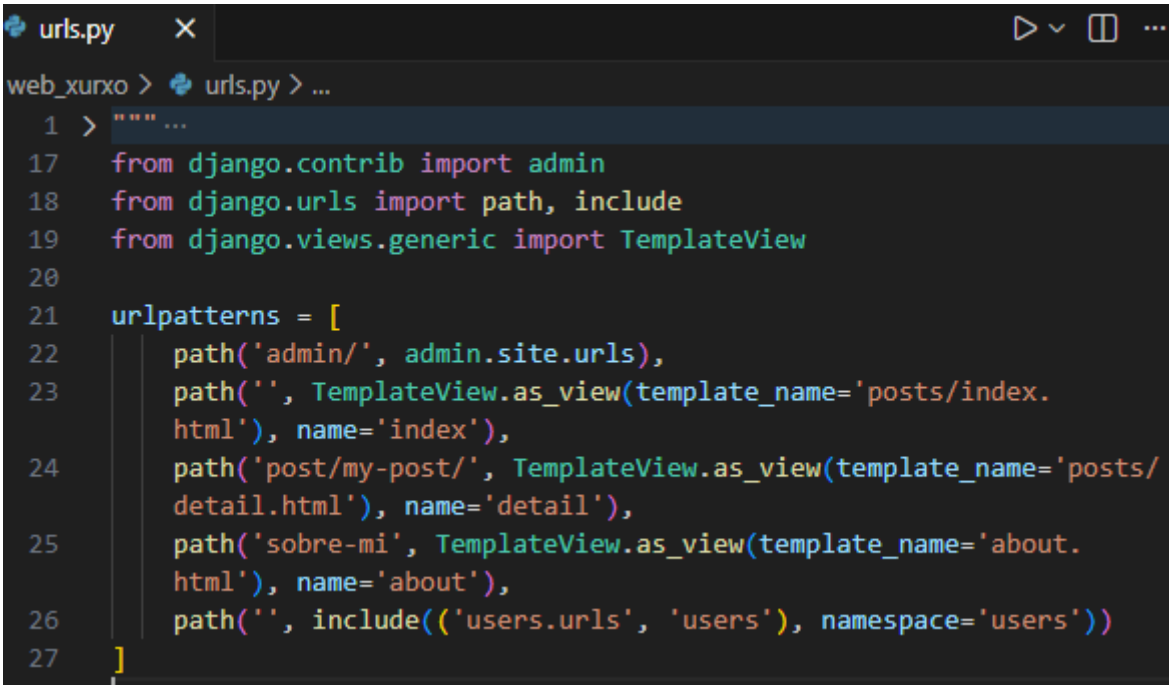
Primeiro teremos que indicar as urls nas que se realizrá o login, a redirección ao facelo e a redirección ao facer logout, para iso modificaremos o ficheiro **web_xurxo/settings.py** e ao final deste engadimos o seguinte código:



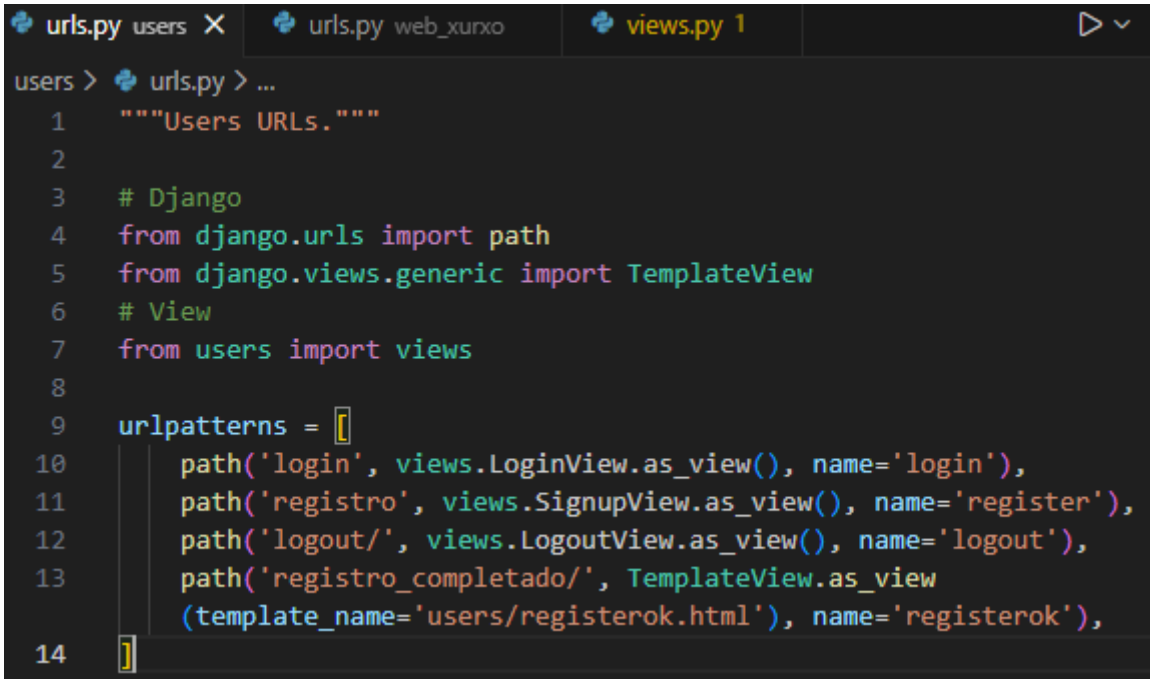
O seguinte paso que temos que facer será crear o ficheiro **users/url.py** e mover as urls de login e de rexistro que estaban no ficheiro **web_xurxo/urls.py**

Aproveitando que temos que modificar as rutas vamos mellorar a lexibilidade do código e vamos cambiar os **path** pola súa forma compacta.

Asi quedaría o noso ficheiro **web_xurxo/urls.py**



E asi quedaría **users/urls.py**



O que acabamos de facer foi modificar as urls de **login** e **rexistro** para que apunten aos ficheiros **user/views.py** e ás clases que lles darán a funcionalidade.

Tamén creamos a ruta para o rexistro completo que apunta a unha plantilla que crearemos no seguinte paso e o logout.

7.1. Formulario de rexistro

Creamos o ficheiro **users/forms.py** e pegamos o seguinte código:

forms.py

users > forms.py > SignupForm

```python
"""User forms."""

# Django
from django import forms

# Models
from django.contrib.auth.models import User
from users.models import Profile


class SignupForm(forms.Form):
    """Sign up form."""

    email = forms.CharField(
        min_length=6,
        max_length=70,
        widget=forms.EmailInput()
    )
    username = forms.CharField(
        min_length=6,
        max_length=70,
        widget=forms.TextInput()
    )
    password = forms.CharField(
        max_length=70,
        widget=forms.PasswordInput()
    )
    password_confirmation = forms.CharField(
        max_length=70,
        widget=forms.PasswordInput()
    )
```

```
32
33
34      def clean(self):
35          """Verify password confirmation match."""
36          data = super().clean()
37
38          password = data['password']
39          password_confirmation = data['password_confirmation']
40
41          if password != password_confirmation:
42              raise forms.ValidationError('Las contraseñas no coinciden.')
43
44          return data
45
46      def save(self):
47          """Create user and profile."""
48          data = self.cleaned_data
49          data.pop('password_confirmation')
50
51          user = User.objects.create_user(**data)
52          profile = Profile(user=user)
53          profile.save()
```

Neste código o que facemos é declarar a clase **SignupForm** que conterá todos os campos do noso formulario. Co campo **widget** declaramos o **tipo de input**, neste caso todos os campos son de tipo texto, excepto os campos email e password.

A continuación declaramos a función **clean** que se executará despois de enviar o formulario e validará se o contrasinal é igual ao contrasinal repetido.

Por último cramos a función **save** que se encarga de eliminar o campo **password_confirmation** e gardar a información enviada na táboa **User** e **Profile**.

Agora vamos encargarnos da parte visual, abrimos o ficheiro **users/views.py** e pegamos o seguinte código:

views.py | signals.py | forms.py

users > views.py > LogoutView

```python
from django.shortcuts import redirect
from django.views.generic import FormView
from django.urls import reverse_lazy
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth import views as auth_views

# Forms
from users.forms import SignupForm

class SignupView(FormView):
    """Users sign up view."""

    template_name = 'users/register.html'
    form_class = SignupForm
    success_url = reverse_lazy('users:registerok')

    def form_valid(self, form):
        """Save form data."""
        form.save()
        return redirect(self.success_url)

class LoginView(auth_views.LoginView):
    """Login view."""
    template_name = 'users/login.html'


class LogoutView(LoginRequiredMixin, auth_views.LogoutView):
    """Logout view."""
    template_name = 'users/logged_out.html'
```

No **template_name** gardamos a plantilla que vamos utilizar.

No **form_class** gardamos o formulario que creamos anteriormente.

success_url será a url á que redireccionaremos ó usuario se todo foi ben.

Logo usamos **form_valid()** para gardar o usuario se todo vai ben.

Á clase **LoginView** asignámoslle unha plantilla e toda a lóxica encárgase de facela Django.

No caso do **logout** é exactamente igual aínda que como neste caso non necesita plantilla engadimos unha (aínda que non exista).

Agora toca modificar a plantilla que creamos para o rexistro no punto 6. Deseño.

Abre o ficheiro **templates/users/register.html** e pega o seguinte código:

register.html × forms.py views.py

templates > users > register.html > ...

```
{% extends "users/base.html" %}
{% block title %}Web Xurxo - Rexistro{% endblock %}
{% block content %}
<form method="post" class="form-signin">
    {% csrf_token %}
    <h1><a href="{% url 'index' %}" class="title-link">Web Xurxo</
    a></h1>
    <h1 class="h3 mb-3 font-weight-normal">Rexistro</h1>

    {{ form.email }}
    {{ form.username }}
    {{ form.password }}
    {{ form.password_confirmation }}

    <br>

    <button class="btn btn-lg btn-primary btn-block"
    type="submit">Rexistrarse</button>

    {% if form.errors %}
        {% for field in form %}
            {% for error in field.errors %}
                <div class="alert alert-danger">
                    <strong>{{ error|escape }}</strong>
                </div>
            {% endfor %}
        {% endfor %}
```

```
26          {% for error in form.non_field_errors %}
27              <div class="alert alert-danger">
28                  <strong>{{ error|escape }}</strong>
29              </div>
30          {% endfor %}
31      {% endif %}
32
33      <p class="mt-5 mb-3 text-muted">&copy; Web Xurxo {% now "Y" %}</
        p>
34      </form>
35  {% endblock %}
```

Engadimos o **csrf_roken** que necesitaremos para validar o formulario e se hay erros ao crear usuarios amosará unha alerta.

Agora os campos do formularios creamolos co obxecto **form.campo** que é unha clase que creamos no **views.py** donde instanciamos o obxecto da clase **SignupForm** e **LoginForm**

Se todo sae ben á hora de crear un usuario enviarémolo a unha plantilla avisando de que o usuario foi creado correctamente, pera iso creamos o ficheiro **templates/users/registerok.html** e engadimos o seguinte código:

registerok.html | register.html | forms.py | views.py

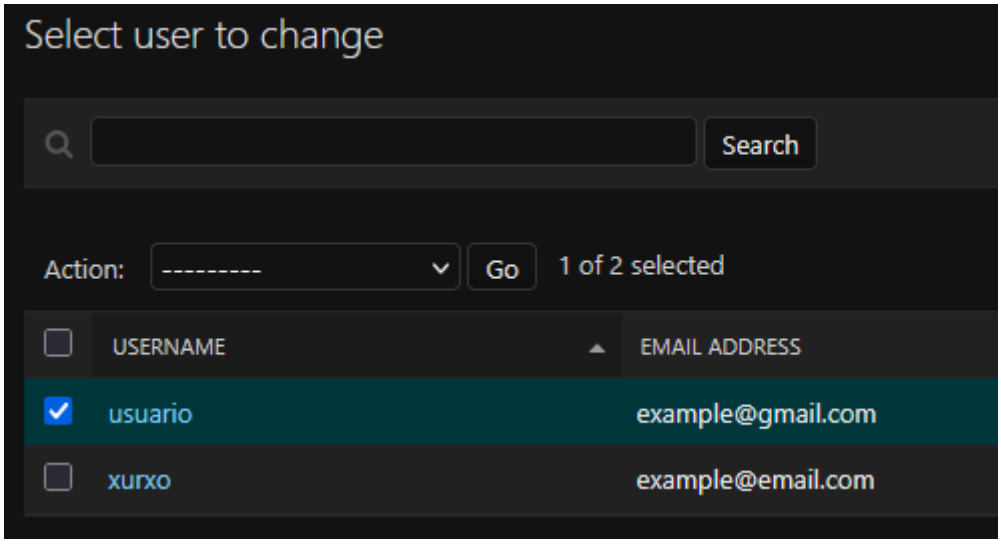
templates > users > registerok.html > ...

```
1   {% extends "users/base.html" %}
2   {% block title %}Web Xurxo - Rexistro completado{% endblock %}
3   {% block content %}
4   <main role="main" class="container">
5       <div class="starter-template">
6           <h1>O teu usuario foi creado con éxito</h1>
7           <p class="lead">Se a páxina non é redirixida en 5 segundos
            pulsa <a href="{% url 'users:login' %}">aquí</a></p>
8       </div>
9   </main>
10  <script>
11      setTimeout(function(){
12          window.location.replace("{% url 'users:login' %}");
13      }, 5000);
14  </script>
15  {% endblock %}
```

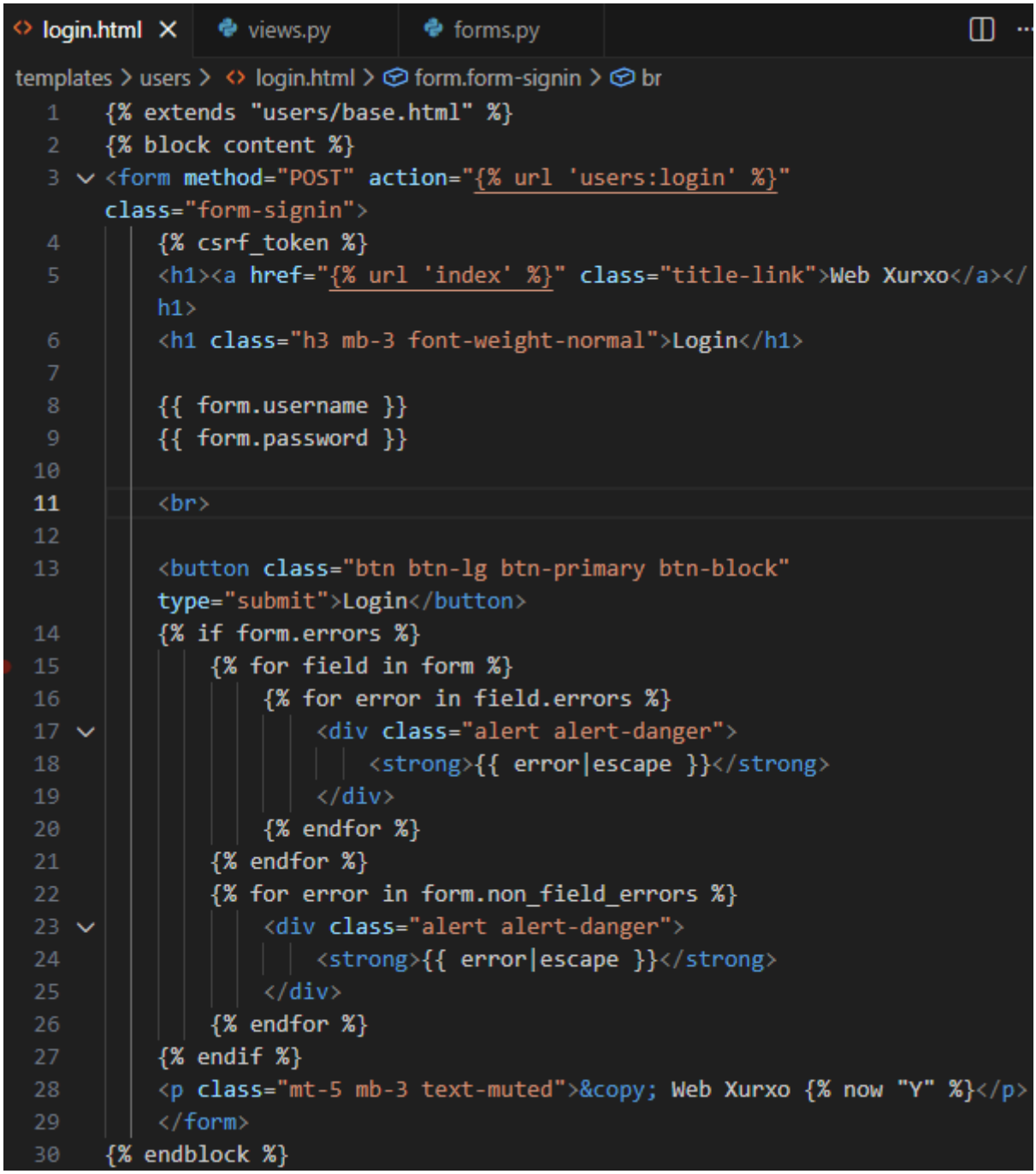
Amosamos unha mensaxe e redireccionamos á paxina de login utilizando **{% url 'user:login' %}**, isto mira nos nosos arquivos **urls.py** e xera unha url en base ao **namespace** e **name** que engadimos as nosas urls.

Para acceder ao rexistro entramos **http://127.0.0.1:8000/register** e probamos a crear un usuario para crear probas posteriormente

Comprobamos que estea creado na parte de administración e listo:



Para rematar abrimos o ficheiro **templates/users/login.html** e modificamos o código polo seguinte:



Ao igual que na plantilla de rexistro engadimos o **csrf_token** e se hay erros capturámolos.

Ademais tamén chamamos os campos do formulario dende o **views.py**, que falando del debería quedar asi o noso **users/views.py**

views.py | forms.py

users > views.py > LoginView

```python
from django.shortcuts import redirect
from django.views.generic import FormView
from django.urls import reverse_lazy
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth import views as auth_views

# Forms
from users.forms import SignupForm
from users.forms import LoginForm

class SignupView(FormView):
    """Users sign up view."""

    template_name = 'users/register.html'
    form_class = SignupForm
    success_url = reverse_lazy('users:registerok')

    def form_valid(self, form):
        """Save form data."""
        form.save()
        return redirect(self.success_url)

class LoginView(auth_views.LoginView):
    """Login view."""

    template_name = 'users/login.html'
    form_class = LoginForm


class LogoutView(LoginRequiredMixin, auth_views.LogoutView):
    """Logout view."""
    template_name = 'users/logged_out.html'
```

E o noso **users/forms.py** terá que quedar asi:

forms.py

users > forms.py > LoginForm

```python
"""User forms."""

# Django
from django import forms
from django.contrib.auth.forms import AuthenticationForm

# Models
from django.contrib.auth.models import User
# from users.models import Profile

# Clase cos campos do formulario de rexistro
class SignupForm(forms.Form):
    """Sign up form."""

    email = forms.CharField(
        min_length=6,
        max_length=70,
        widget=forms.EmailInput(attrs={
            'id': 'inputEmail',
            'class': 'form-control',
            'placeholder': 'Correo electrónico'
        })
    )
    username = forms.CharField(
        min_length=6,
        max_length=70,
        widget=forms.TextInput(attrs={
            'id': 'inputUsername',
            'class': 'form-control',
            'placeholder': 'Nome de usuario',
        }),
    )
    password = forms.CharField(
        max_length=70,
        widget=forms.PasswordInput(attrs={
            'id': 'inputPassword',
            'class': 'form-control',
            'placeholder': 'Contrasinal',
            'auto-complete': 'new-password'
        })
    )
```

```
41      )
42      password_confirmation = forms.CharField(
43          max_length=70,
44          widget=forms.PasswordInput(attrs={
45              'id': 'inputPasswordRepeat',
46              'class': 'form-control',
47              'placeholder': 'Repetir contrasinal'
48          })
49      )
50
51
52      def clean(self):
53          """Verify password confirmation match."""
54          data = super().clean()
55
56          password = data['password']
57          password_confirmation = data['password_confirmation']
58
59          if password != password_confirmation:
60              raise forms.ValidationError('Os contrasinais non
                coinciden.')
61
62          return data
63
64      def save(self):
65          """Create user and profile."""
66          data = self.cleaned_data
67          data.pop('password_confirmation')
68          user = User.objects.create_user(**data)
69          return user
70
71  # Clase cos campos do formulario de Login
72  class LoginForm(AuthenticationForm):
73      """Login Form"""
74
75      username = forms.CharField(
76          min_length=6,
77          max_length=70,
78          widget=forms.TextInput(attrs={
79              'id': 'id_username',
80              'class': 'form-control',
81              'placeholder': 'Usuario',
82          }),
83      )
```

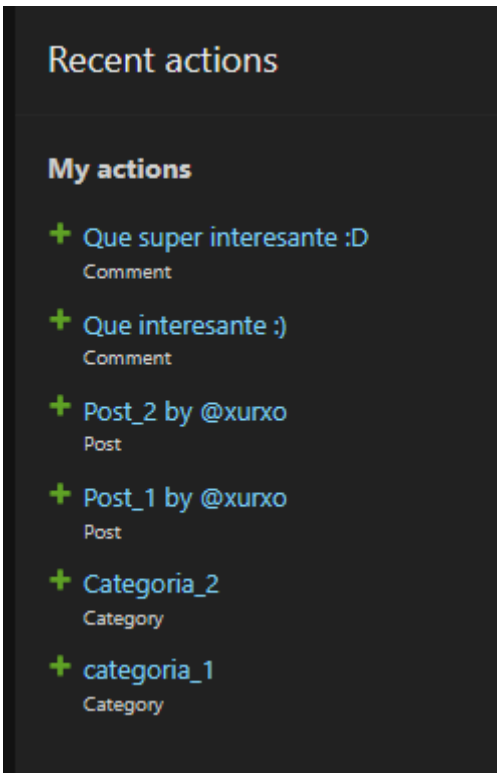
```
84      password = forms.CharField(
85          max_length=70,
86          widget=forms.PasswordInput(attrs={
87              'id': 'id_password',
88              'class': 'form-control',
89              'placeholder': 'Contrasinal',
90          })
91      )
```

8. Vistas (Views)

Neste último paso vamos conectar tódalas partes da nosa aplicación. O primeiro que faremos será encher con datos a nosa **base de datos**.

Antes de crear os **posts** teremos que crear o directorio **media** dentro do noso entorno e logo dentro engadimos **posts/photos** que será o cartafol no que gardaremos as fotos para amosalas na nosa web.

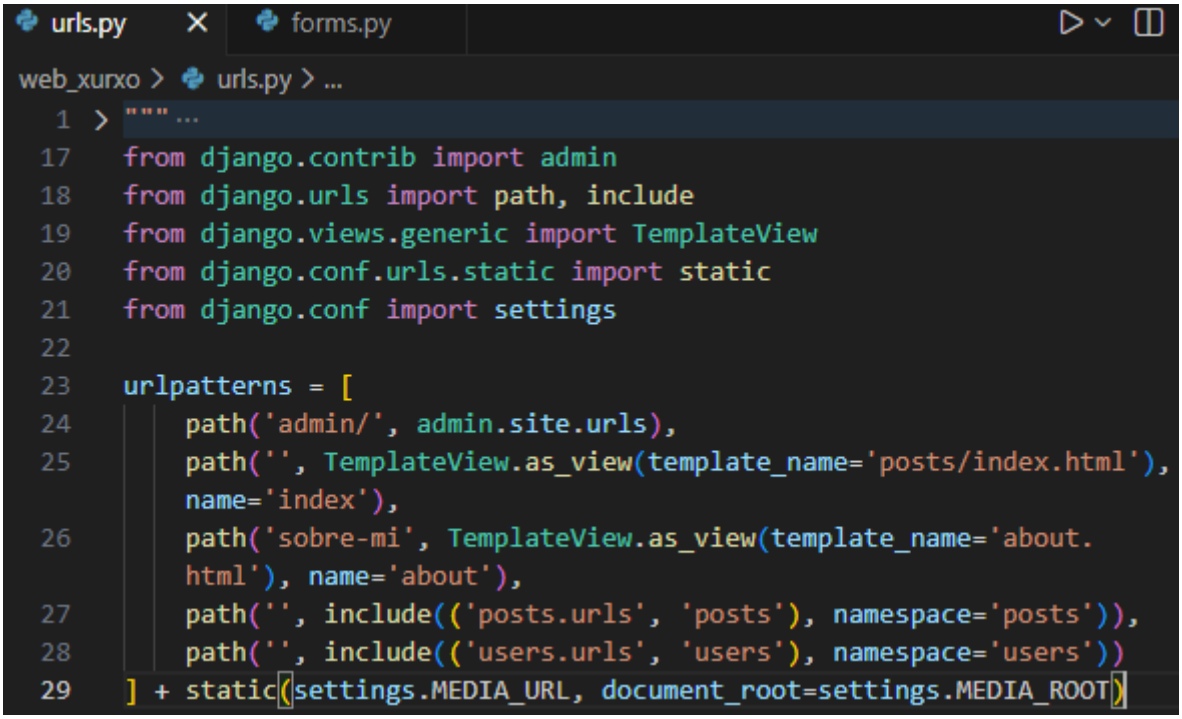
Para iso vámonos a **localhost:8000/admin** e agora engadimos algúns post, categorías, comentarios...



Recorda **desmarca o cadro is_draft** para que os **posts** se amosen no blog, senón Django entende que son borradores e non se amosarán.

Agora vamos cambiar as urls para que apunten á vista que crearemos no seguinte paso.

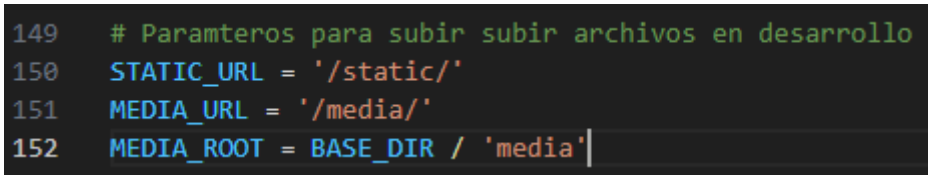
Abrimos o ficheiro **web_xurxo/urls.py** e modificamos o código que temos polo seguinte:



Como se pode ver fixemos para os enlaces de **posts** o mesmo que cos enlaces de **users**. Agora vamos gardar tódalas urls de posts dentro de **posts/urls.py**.

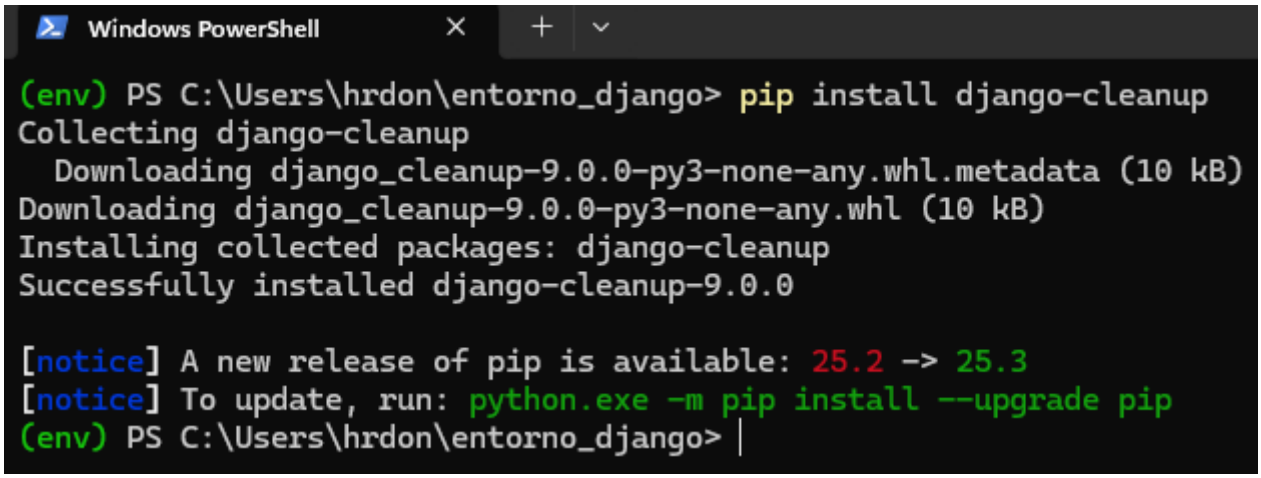
A liña **static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)** que xa estaba en settings e se utiliza para **acceder ós arquivos** aloxados na carpeta **media**.

Logo dirixímonos ao ficheiro **web_xurxo/settings.py** e no fondo do arquivo engadimos este código:

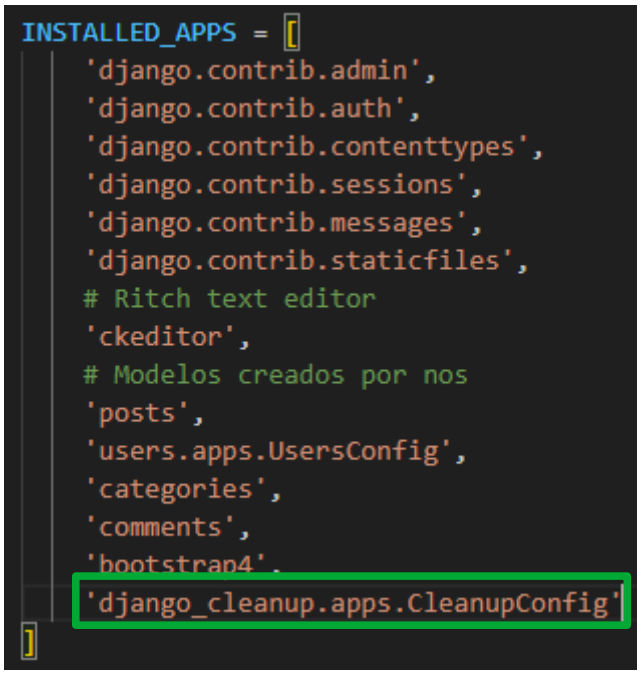


Por último para non deixar lixo no servidor instalaremos **django-cleanup**.

Abrimos a terminal co noso contorno activo e executa o comando **pip install django-cleanup**



Unha vez instalado engadímolo ao **INSTALLED_APPS**



E listo, agora se **eliminamos** un post tamén se eliminan as imaxes do cartafol **media**

Despois creamos o ficheiro **posts/urls.py** e engadimos o seguinte código para que as nosas urls apunten ás funcións que vamos crear nas nosas vistas:

urls.py

```
posts > urls.py > ...
1   """Posts URLs."""
2
3   # Django
4   from django.urls import path
5
6   # Views
7   from posts import views
8
9   urlpatterns = [
10      path('', views.PostsFeedView.as_view(), name='blog'),
11      path('posts/<slug:url>/', views.PostDetailView.as_view(),
        name='detail'),
12      path('posts/save_comment', views.save_comment,
        name='save_comment'),
13  ]
```

Acabamos de crear 3 urls:

1. Para a páxina principal.
2. Será o detalle do post e no que usaremos o método **slug** para recuperar a última parte do path da url.
3. Encargarase de gardar os comentarios.

Antes de poñernos coa vista necesitamos crear un formulario para posteriormen gardar os comentarios que os usuarios escriban nos nosos posts, así que crearemos o arquivo **comments/forms.py** e insertamos o código que aparece na seguinte páxina:

forms.py

comments > forms.py > CreateCommentForm > Meta

```python
"""User forms."""

# Django
from django import forms

# Models
from comments.models import Comment
from django.contrib.auth.models import User

class CreateCommentForm(forms.ModelForm):
    """Post model form."""

    comment = forms.CharField(widget=forms.Textarea)

    class Meta:
        """Form settings."""

        model = Comment
        fields = ('user', 'profile', 'post', 'comment')
```

Con este código estamos creando un campo de tipo **textarea** que será o que garde o comentario do usuario. Na clase **Meta** indicamos o modelo de referencia e os campos que ten que gardar, neste caso o usuario que o escribiu, o post ao que vai referenciado e o comentario.

Agora que temos algo de información que amosar vamos recuperala na vista para que se amoso lo na nosa web, para iso editamos **post/views.py** e engadimos este código:

views.py

posts > views.py > save_comment

```python
from django.shortcuts import render, redirect
from django.views.generic import DetailView, ListView
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

# Models
from posts.models import Post
from categories.models import Category
from comments.models import Comment

```

```python
# Forms
from comments.forms import CreateCommentForm


class PostsFeedView(ListView):
    """Index."""
    template_name = 'posts/index.html'
    model = Post
    ordering = ('-created',)
    paginate_by = 10
    context_object_name = 'posts'
    queryset = Post.objects.filter(is_draft=False)


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['categories'] = Category.objects.all()
        return context


class PostDetailView(DetailView):
    """Detail post."""
    template_name = 'posts/detail.html'
    model = Post
    context_object_name = 'post'
    slug_field = 'url'
    slug_url_kwarg = 'url'


    def get_queryset(self):
        return Post.objects.filter(is_draft=False)


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['categories'] = Category.objects.all()
        context['comments'] = Comment.objects.filter(post=self.
        get_object()).all()
        context['form_comments'] = CreateCommentForm()
        return context


```

```python
@login_required
def save_comment(request):
    if request.method == 'POST':
        url = request.POST['url']
        post = {
            'user': request.user.id,
            'profile': request.user.id,
            'comment': request.POST['comment'],
            'post': request.POST['post']
        }
        form = CreateCommentForm(post)
        if form.is_valid():
            form.save()
            return redirect('posts:detail', url=url)
    else:
        return HttpResponse(status=405)
    return HttpResponse(status=500)
```

Neste arquivo temos as tres vistas ás que se chaman dende as urls, vamos explicar que fan:

O que estamos a facer na clase **PostsFeedView** é utlizar unha das vistas predeterminadas que ten Django, neste caso **ListView** que permite retornar unha listas de obxectos:

- **template_name**: A plantilla que será a que reciba toda a información.
- **model**: O modelo que vamos utilizar, neste caso **Post**.
- **Ordering**: A orde polo que queremosque aparezan os posts, neste caso queremos que aparezan por orde de creación de forma descendente.
- **paginate_by**: O número de posts que se amosarán por páxina.
- **queryset**: Para engadir condicions adicionais, non queremos que se amosen os borradores así que descartamolos.

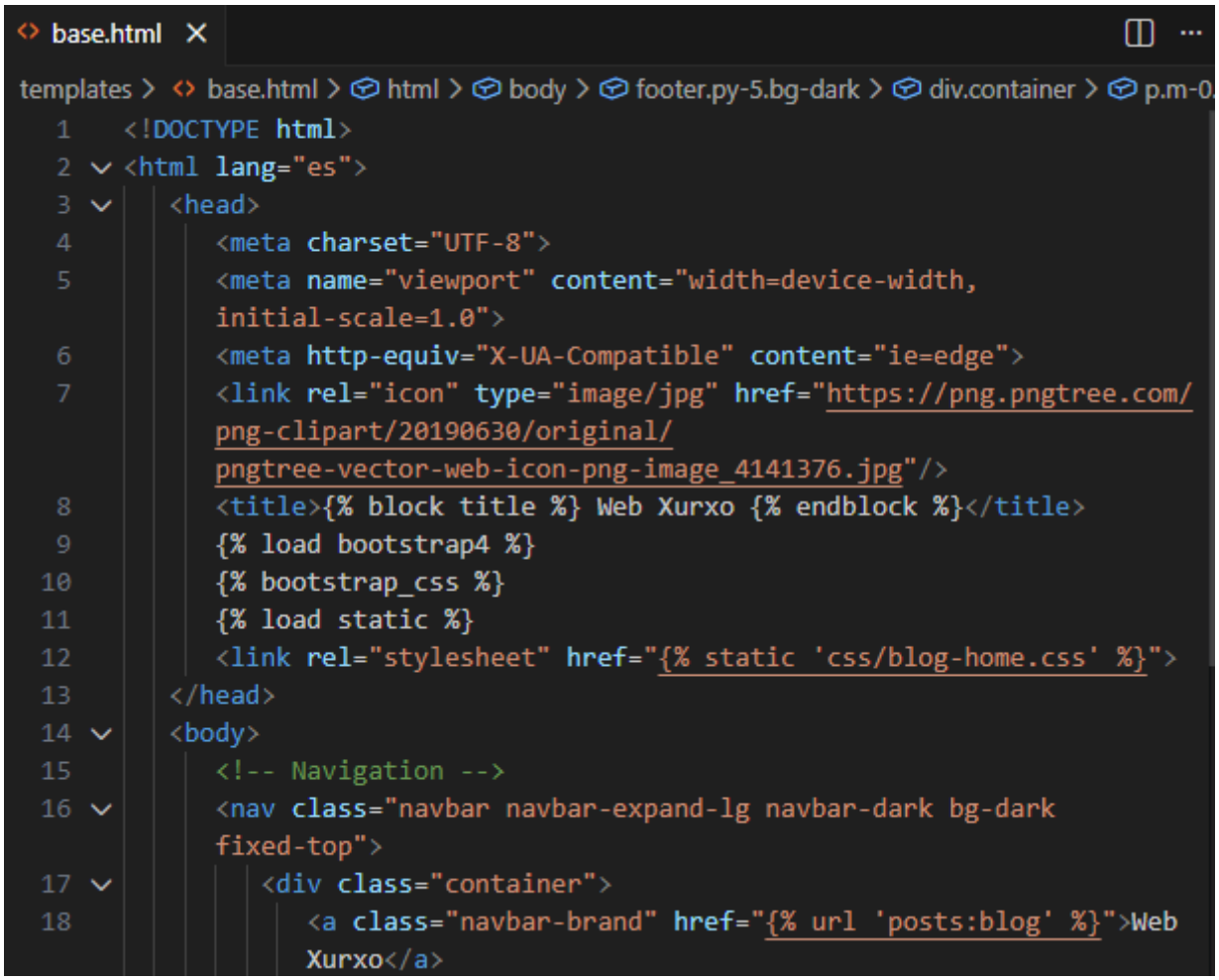
A parte utilizamos a función **get_context_data** para engadir información extra a nosa vista, neste caso as categorías para amosalas no menú lateral.

A clase chamada **PostDetailView** é de tipo **DetailView** e encargase de amosar un post concreto, para iso recupera o título na url e utilizao a modo de consulta. Para recuperalo utiliza **slug_field = 'url'** e **slug_url_kwarg = 'url'**

Neste código tamén utilizamos a función **get_context_data** para recoller as categorías e os comentarios que poida ter o post ademais do formulario que creamos anteriormente para gardar comentarios e que só se amosará en caso de que o usuario esté logeado.

Por último temos a función **saveComment** co decorado **@login_required** que nos obriga a estar logeados para poder gardar un comentario. Tamén revisamos que a chamada a faga mediante o método **POST** e senon devolvemos un erro. Se todo foi ben gardamos e recargamos a páxina na que estamos.

Agora que temos a vista completada vamos darlle un pouco de funcionalidade a todolos enlaces da web e amosar os posts e categorías, para iso abrimos o noso arquivo **templates/base.html** e substituímos o código polo seguinte:



```
19          <button class="navbar-toggler" type="button"
            data-toggle="collapse" data-target="#navbarResponsive"
            aria-controls="navbarResponsive" aria-expanded="false"
            aria-label="Toggle navigation">
20          <span class="navbar-toggler-icon"></span>
21          </button>
22          <div class="collapse navbar-collapse"
            id="navbarResponsive">
23              <ul class="navbar-nav ml-auto">
24                  <li class="nav-item active">
25                      <a class="nav-link" href="{% url 'posts:blog' %}
                        ">Blog
26                      <span class="sr-only">(current)</span>
27                      </a>
28                  </li>
29                  <li class="nav-item">
30                      <a class="nav-link" href="{% url 'about' %}
                        ">Sobre min</a>
31                  </li>
32                  {% if user.is_authenticated %}
33                  <li class="nav-item dropdown">
34                      <a class="nav-link dropdown-toggle"
                        data-toggle="dropdown" href="#" role="button"
                        aria-haspopup="true" aria-expanded="false">
                        {{request.user.username}}</a>
35                      <div class="dropdown-menu">
36                          <a class="dropdown-item" href="{% url
                            'users:logout' %}">Logout</a>
37                      </div>
38                  </li>
39                  {% else %}
40                  <li class="nav-item">
41                      <a class="nav-link" href="{% url 'users:login' %}
                        ">Login</a>
42                  </li>
43                  <li class="nav-item">
44                      <a class="nav-link" href="{% url
                        'users:register' %}">Rexistro</a>
45                  </li>
46                  {% endif %}
47              </ul>
48          </div>
49      </div>
50  </nav>
```

```
51          <!-- Page Content -->
52          <div class="container">
53              {% block content %}{% endblock %}
54              <!-- Sidebar Widgets Column -->
55              <div class="col-md-4">
56                  <!-- Categories Widget -->
57                  <div class="card my-4">
58                      <h5 class="card-header">Categorias</h5>
59                      <div class="card-body">
60                          <div class="row">
61                              <div class="col-lg-12">
62                                  <ul class="list-unstyled mb-0">
63                                      {% for category in categories %}
64                                      <li>
65                                          <a href="#">{{category.name}}</a>
66                                      </li>
67                                      {% endfor %}
68                                  </ul>
69                              </div>
70                          </div>
71                      </div>
72                  </div>
73                  <!-- Side Widget -->
74                  <div class="card my-4">
75                      <h5 class="card-header">Widget lateral</h5>
76                      <div class="card-body">
77                          Podes colocar o que quieras dentro destos widgets
                            laterais. Son fáciles de usar e incluen os novos
                            contedores de tarxetas de Bootstrap 4!
78                      </div>
79                  </div>
80              </div>
81          </div>
82          <!-- Footer -->
83          <footer class="py-5 bg-dark">
84              <div class="container">
85                  <p class="m-0 text-center text-white">Copyright &copy;
                    Web Xurxo {% now "Y" %}</p>
86              </div>
87              <!-- /.container -->
88          </footer>
89          {% bootstrap_javascript jquery='full' %}
90      </body>
91  </html>
```

Os **cambios** máis **importantes** que fixemos son as **urls**, unha condición para **comprobar se o usuario está logeado** con **{% if user.is_authenticated %}** e se está así pintamos o botón coa opción de **logout**, senon a opción de ir á pantalla de rexistro e de login. Tamén listamos as categorías.

Agora que xa temos a base completa abrimos o arquivo **templates/posts/index.html** e modificamos o contido polo seguinte:

index.html

templates > posts > index.html > div.row

```
{% block content %}
    <div class="row">
      <div class="col-md-8">
        <h1 class="my-4">Web Xurxo
        </h1>
        {% for post in posts %}
        <div class="card mb-4">
          <img class="card-img-top" src="{{ post.image_header.url }}"
          alt="Card image cap">
          <div class="card-body">
            <h2 class="card-title">{{post.title}}</h2>
            <p class="card-text">{{post.post|safe|truncatechars:300}}
            </p>
            <a href="{% url 'posts:detail' post.url%}" class="btn
            btn-primary">Leer máis &rarr;</a>
          </div>
          <div class="card-footer text-muted">
            Escrito o {{post.created}}
            <a href="#">{{post.user.username}}</a>
          </div>
        </div>
        {% endfor %}
        <ul class="pagination justify-content-center mb-4">
          <li class="page-item">
            <a class="page-link" href="#">&larr; Anterior</a>
          </li>
          <li class="page-item disabled">
            <a class="page-link" href="#">Seguinte &rarr;</a>
          </li>
        </ul>
      </div>
{% endblock %}
```

Acabamos de engadir os datos dos posts e substituilos polos que tíñamos de proba. O máis destacable sería o enlace ao detalle que é este **{% url 'posts:detail' post.url %}**. O que estamos facendo é coller a url do detalle e concatenarlle a url que xeramos con slug para crear o enlace ao detalle do post.

Por último editamos o arquivo **templates/posts/details.html** e substituímos o código existente por este:

```
33          {% endif %}
34          {% for comment in comments %}
35          <div class="media mb-4">
36            {% if comment.user.profile.photo %}
37            <img class="d-flex mr-3 rounded-circle" width="50px"
              height="50px" src="/media/{{ comment.user.profile.photo }}"
              alt="">
38            {% else %}
39            <img class="d-flex mr-3 rounded-circle" src="http://
              placehold.it/50x50" alt="">
40            {% endif %}
41            <div class="media-body">
42              <h5 class="mt-0">{{comment.user.username}}</h5>
43              {{comment.comment}}
44            </div>
45          </div>
46          {% endfor %}
47        </div>
48    {% endblock %}
```

Ó igual que no index engadimos os datos do post e modificamos o formulario para poder gardar os comentarios que escriba o usuario. Tamén se amosa a lista de tódolos comentarios se os ten o post.

Con isto xa teríamos todo o básico, aínda faltarían algunhas cousas como a paxinación, o login co email e a validación do email cando un novo usuario se rexistra.

Fontes

- Aprende Django desde 0 -> cosasdedevs.com
- Primera app con Django -> djangoproject.com
- Solución problema interprete Python en vscode -> you tube – Andrés Cruz
- Como crearl URLs amigables en Django -> cosasdedevs.com