

Agregar la vista visit-us

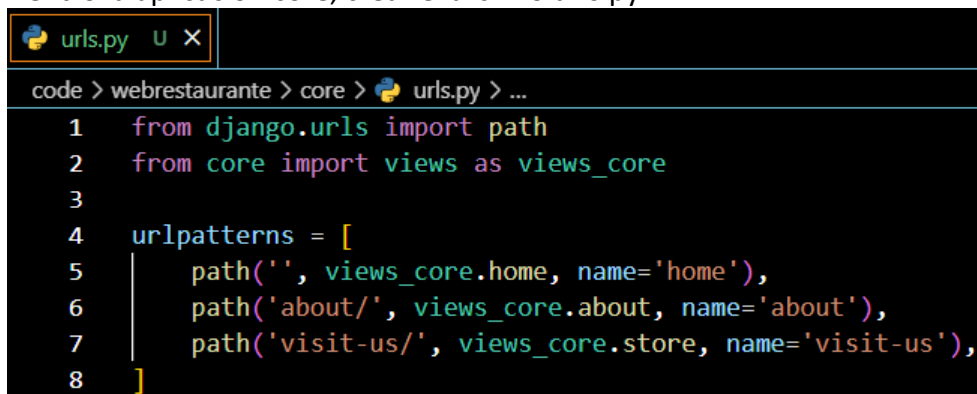
Práctica:

Crear el template, crear el método correspondiente en el view y finalmente configurar el archivo urls.py

Configuración de URLs

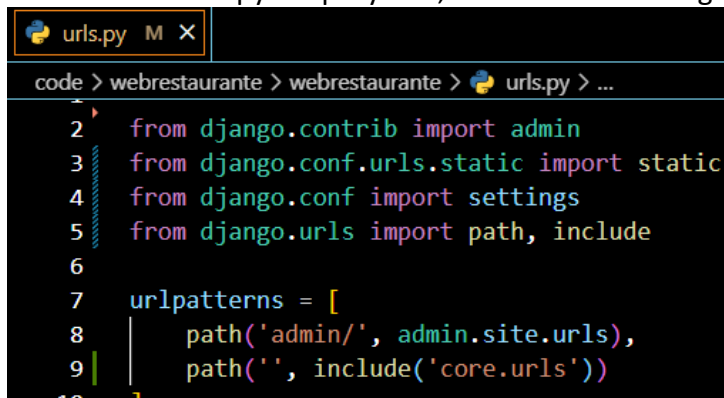
El archivo urls, puede ir creciendo y el proyecto volverse poco escalable.
Por lo que es recomendable que cada aplicación, tenga su propio archivo urls.

Dentro la aplicación core, crear el archivo urls.py



```
code > webrestaurante > core > urls.py > ...  
1 from django.urls import path  
2 from core import views as views_core  
3  
4 urlpatterns = [  
5     path('', views_core.home, name='home'),  
6     path('about/', views_core.about, name='about'),  
7     path('visit-us/', views_core.store, name='visit-us'),  
8 ]
```

En el archivo urls.py del proyecto, modificarlo de la siguiente manera:



```
code > webrestaurante > webrestaurante > urls.py > ...  
1  
2 from django.contrib import admin  
3 from django.conf.urls.static import static  
4 from django.conf import settings  
5 from django.urls import path, include  
6  
7 urlpatterns = [  
8     path('admin/', admin.site.urls),  
9     path('', include('core.urls'))  
10 ]
```

¿Qué pasaría si en otra aplicación se tiene el nombre “visit-us”?

Existiría un problema, porque django no sabría a cual renderizar, por lo que se recomienda colocar un nombre de aplicación en el urlpatterns.

```
urls.py U X
code > webrestaurante > core > urls.py > ...
1 from django.urls import path
2 from core import views as views_core
3
4 core_urlpatterns = ([
5     path('', views_core.home, name='home'),
6     path('about/', views_core.about, name='about'),
7     path('visit-us/', views_core.store, name='visit-us'),
8 ], 'core')
```

En el archivo urls.py del proyecto, modificarlo de la siguiente manera.

```
urls.py M X
code > webrestaurante > webrestaurante > urls.py > ...
1
2 from django.contrib import admin
3 from django.conf.urls.static import static
4 from django.conf import settings
5 from django.urls import path, include
6 from core.urls import core_urlpatterns
7
8 urlpatterns = [
9     path('admin/', admin.site.urls),
10    path('', include(core_urlpatterns)),
11 ]
```

Es necesario cambiar la referencia desde el archivo base.html

```
ive{% endif %} px-lg-4">
href="{% url 'core:home' %}">Ir
```

Aplicación Blog

1. Crear la aplicación “blog”
2. Crear los modelos “Category” y “Post”

```
models.py U X
code > webrestaurante > blog > models.py > Category > __str__
1 from django.db import models
2 from django.utils.timezone import now
3 from django.contrib.auth.models import User
```

```

5 class Category(models.Model):
6     name = models.CharField(max_length=100, verbose_name="Nombre")
7     created = models.DateTimeField(auto_now_add=True, verbose_name="Fecha de creación")
8     updated = models.DateTimeField(auto_now=True, verbose_name="Fecha de actualización")
9
10    class Meta:
11        verbose_name = 'Categoría'
12        verbose_name_plural = 'Categorías'
13        ordering = ['name']
14
15    def __str__(self):
16        return self.name

```

```

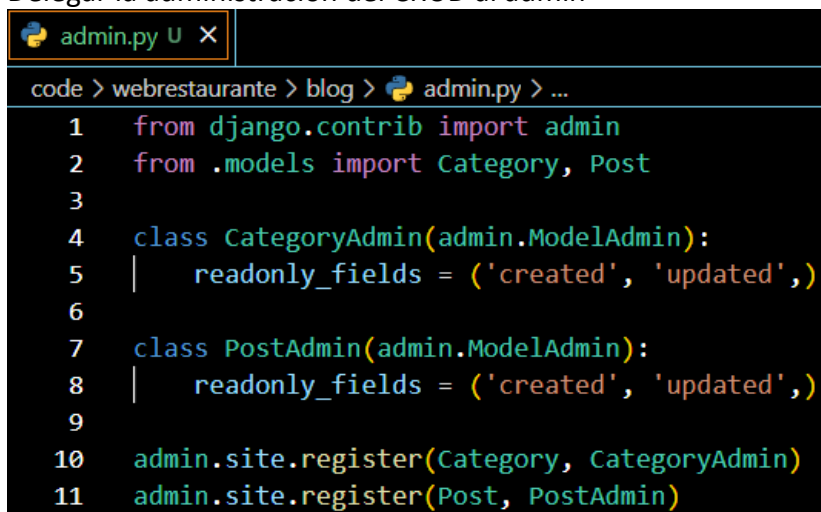
18 class Post(models.Model):
19     title = models.CharField(max_length=100, verbose_name="Título")
20     content = models.TextField(verbose_name='Contenido')
21     published = models.DateTimeField(default=now, verbose_name='Fecha de publicación')
22     image = models.ImageField(upload_to='blog', null=True, blank=True, verbose_name='Imagen')
23     author = models.ForeignKey(User, verbose_name='Autor', on_delete=models.CASCADE)
24     categories = models.ManyToManyField(Category, verbose_name='Categorías', related_name='get_posts')
25     created = models.DateTimeField(auto_now_add=True, verbose_name="Fecha de creación")
26     updated = models.DateTimeField(auto_now=True, verbose_name="Fecha de actualización")
27
28    class Meta:
29        verbose_name = 'Post'
30        verbose_name_plural = 'Posts'
31        ordering = ['-created']
32
33    def __str__(self):
34        return self.title

```

3. Ver documentación

- a. <https://docs.djangoproject.com/en/5.0/search/?q=relationships>

4. Delegar la administración del CRUD al admin



```

code > webrestaurant > blog > admin.py > ...
1  from django.contrib import admin
2  from .models import Category, Post
3
4  class CategoryAdmin(admin.ModelAdmin):
5      readonly_fields = ('created', 'updated',)
6
7  class PostAdmin(admin.ModelAdmin):
8      readonly_fields = ('created', 'updated',)
9
10 admin.site.register(Category, CategoryAdmin)
11 admin.site.register(Post, PostAdmin)

```

5. Agregar la aplicación

6. Agregar también la aplicación django_cleanup

7. Realizar las migraciones correspondientes

8. Verificar su funcionamiento
 - a. Crear 3 usuarios
 - i. juanito1 Zow#29Pv.
 - b. Colocar nombre y apellido al usuario Admin
 - i. (Puede agregar otro usuario si lo desea)
 - c. Agregar 2 o 3 categorías
 - d. Agregar 2 o 3 Post's
9. Realizar:
 - a. Creación de la estructura de directorios para el template
 - b. Crear el archivo post_list.html

```
<div class="container">
  {% for post in posts %}
  <div class="row">
    <div class="col-xl-9 mx-a
    <div class="cta-inner
```

- i.
- ii. Colocar el siguiente filtro

```
>{{post.published|date:"d/m/Y"}}</s
>{{post.title}}</span>
```

```
<p class="mb-0 mbt">
  <span class="section-heading-under">Publicado por <em><b>{{post.author.first_name}}</b></em> en <em>
  {% for category in post.categories.all %}
  | | <a href="#" class="link">{{category.name}}</a>{% if not forloop.last %},{% endif %}
  {% endfor %}
  </em></span>
```

iii.

- c. Crear la vista

```
views.py U X
code > webrestaurante > blog > views.py > ...
1 from django.shortcuts import render
2 from .models import Post, Category
3
4 def blog(request):
5     posts = Post.objects.all()
6     return render(request, 'blog/post_list.html', {'posts':posts})
7
```

- d. Crear el archivo urls.py

```
urls.py U X
code > webrestaurante > blog > urls.py > ...
1 from django.urls import path
2 from blog import views
3
4 blog_urlpatterns = ([
5     path('', views.blog, name='post'),
6 ], 'blog')
```

i.

- e. Modificar el archivo urls.py del proyecto
- f. Modificar el archivo base.html para incluir ésta opción

10. Verificar su funcionamiento

<https://docs.djangoproject.com/en/5.0/ref/templates/builtins/#date>

11. Crear la vista de category

```
11 def category(request, category_id):
12     category = Category.objects.get(id=category_id)
13
14     return render(request, 'blog/category.html', {'category': category})
```

12. Agregar el url correspondiente

```
4 blog_urlpatterns = (
5     path('', views.blog, name='post_list'),
6     path('category/<int:category_id>', views.category, name='category'),
7 ], 'blog')
```

13. Modificar el archivo post_list.html para mostrar las categorías y el autor

```
<p class="mb-0 mbt">
  <span class="section-heading-under">Publicado por <em><b>{{post.author.first_name}}</b></em> en <em>
    {% for category in post.categories.all %}
    | <a href="{% url 'blog:category' category.id %}" class="link">{{category.name}}</a> {% if not forloop.last %},{% endif %}
    {% endfor %}
  </em></span>
</p>
```

14. La página category.html es una copia de todo el contenido de la página post_list.html, por lo que es conveniente crear la página blog.html (con todo que se encuentra dentro del for)

15. Modificar la página post_list.html para incluir la página blog.html

```
1 {% extends "core/base.html" %}
2 {% block content %}
3 <section class="page-section cta">
4     <div class="container">
5         {% for post in posts %}
6         |     {% include "blog/blog.html" %}
7         |     {% endfor %}
8     </div>
9 </section>
10 {% endblock %}
```

16. Crear la página category.html, pero ahora se realiza ingeniería en reversa para la consulta. Se puede personalizar el “related_name”, en el modelo, realizar el siguiente cambio.

```
categories = models.ManyToManyField(
    Category, verbose_name="Categorías", related_name='get_posts')
created = models.DateTimeField()
```

17. Ahora se cambia el nombre en la página category.html

```
3 {% block content %}
4 {% for post in category.get_posts.all %}
5     {% include 'blog/blog.html' %}
```

18. Cambiar el menú de opciones, utilizar el filtro “slice”

```
{% if request.path|slice:':6' == '/blog/'%}active{%  
innercase text-expanded"
```

Cambiar a TemplateView (opcional)

1. Modificar el archivo views.py

```
views.py ×  
webRestaurante > blog > views.py > ...  
1 from django.shortcuts import render  
2 from django.views.generic.list import ListView  
3 from django.views.generic import TemplateView  
4 from .models import Post, Category  
5  
6  
7 class CategoryPageView(TemplateView):  
8     template_name = "blog/category.html"  
9  
10    def get_context_data(self, **kwargs):  
11        context = super().get_context_data(**kwargs)  
12        category_id = self.kwargs['category_id']  
13        category = Category.objects.get(id=category_id)  
14        context['category'] = category  
15    return context
```

2. Modificar el archivo urls.py

```
urls.py ×  
webRestaurante > blog > urls.py > ...  
1 from django.urls import path  
2 from blog.views import PostListView  
3 from blog import views  
4 from blog.views import CategoryPageView  
5 post_urlpatterns = ([  
6     path('', PostListView.as_view(), name='post_list'),  
7     path('category/<int:category_id>',  
8         CategoryPageView.as_view(), name='category'),  
9 ], 'posts')
```