

# End-to-End Proyecto Completo que Cubre las Bases del Framework de Django

Este proyecto es una aplicación web desarrollada en Django, diseñada para gestionar tareas de distintos usuarios de forma modular. La aplicación proporciona una interfaz intuitiva para los usuarios y administradores, facilitando la inserción y gestión de items.

Cualquier fallo, mejora o sugerencia es absolutamente bienvenida.

Irene 😊

## ENTORNO

### 1. Activar un entorno

#### >>Terminal

Crear entorno

```
virtualenv entorno1  
.\entorno1\Scripts\activate
```

#### **Desactivar:**

deactivate

#### **Comprobar bibliotecas instaladas:**

pip freeze

#### **En la creación de proyecto:**

**Creamos carpeta mi-web**

**Creamos venv web (virtualenv web), dentro de mi-web**

**Activamos → .\web\Scripts\activate**

```
PS C:\Users\Irene\desktop\entornos\mi_web> .\web\Scripts\activate  
(web) PS C:\Users\Irene\desktop\entornos\mi_web> |
```

## 2. Instalar Django

### Instalación

*pip install django*

Creación de subcarpeta como fuente dentro de mi-web

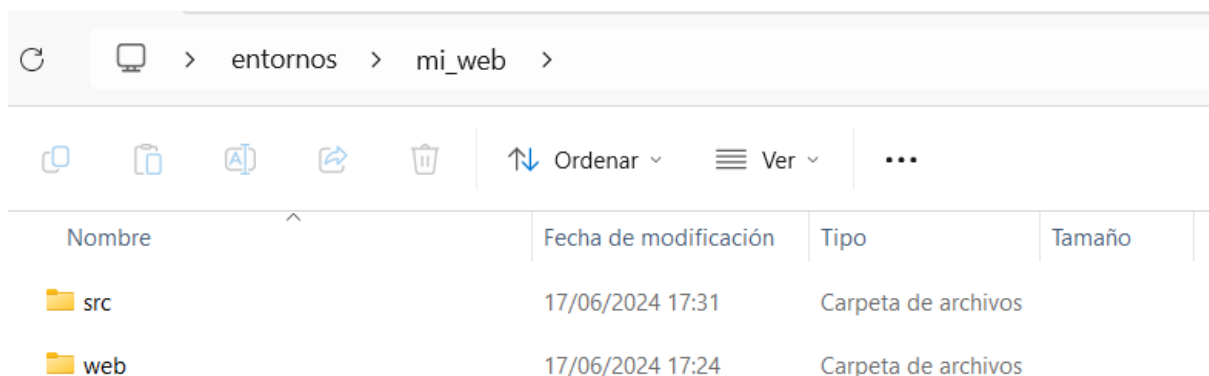
Aquí almacenaremos nuestro proyecto, es una buena práctica llamarlo **src**

```
(web) PS C:\Users\Irene\desktop\entornos\mi_web> mkdir src

Directorio: C:\Users\Irene\desktop\entornos\mi_web

Mode                LastWriteTime         Length Name
----                -
d-----         17/06/2024   17:31             src

(web) PS C:\Users\Irene\desktop\entornos\mi_web>
```



The screenshot shows a Windows File Explorer window with the address bar set to 'entornos > mi\_web >'. The main pane displays a table of files and folders. A new folder named 'src' has been created, with a modification date of 17/06/2024 at 17:31. Below it, the 'web' folder is also visible, modified at 17:06/2024 17:24. The table has columns for 'Nombre', 'Fecha de modificación', 'Tipo', and 'Tamaño'.

Nombre	Fecha de modificación	Tipo	Tamaño
src	17/06/2024 17:31	Carpeta de archivos	
web	17/06/2024 17:24	Carpeta de archivos	

### Cambiar a src

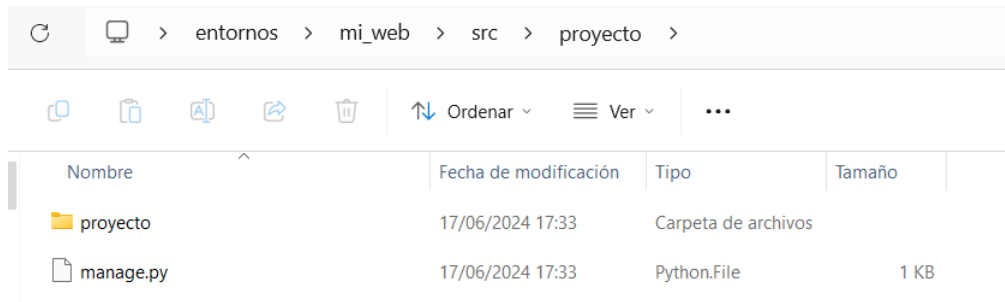
*cd src*

### Iniciar proyecto

```
(web) PS C:\Users\Irene\desktop\entornos\mi_web\src> django-admin startproject proyecto
(web) PS C:\Users\Irene\desktop\entornos\mi_web\src>
```

> Al hacerlo, verás que se crea automáticamente el archivo manage.py

## End-to-end task manager con Django



### Crear servidor

> Primero cambiamos a la carpeta proyecto  
cd proyecto

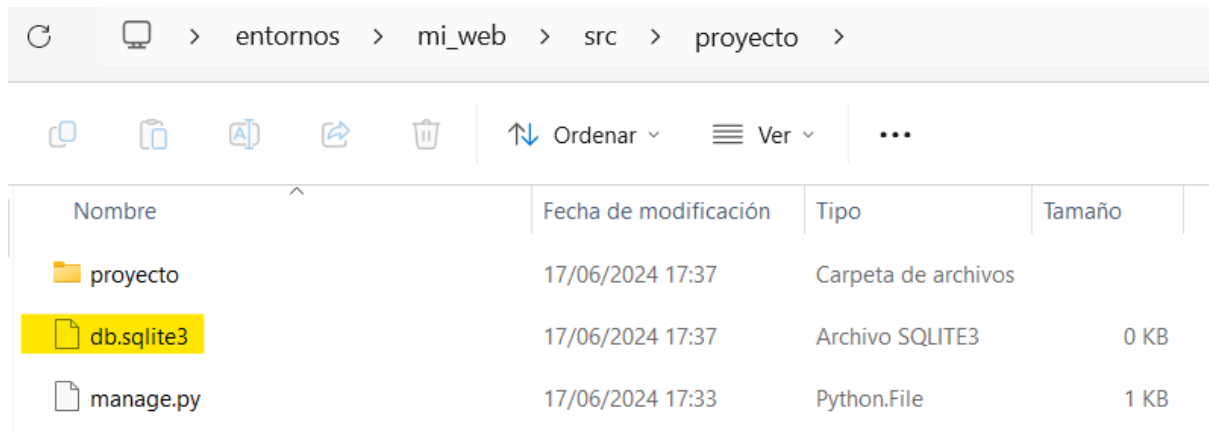
> Después ejecutamos **python manage.py runserver**

```
(web) PS C:\Users\Irene\desktop\entornos\mi_web\src\proyecto> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 17, 2024 - 17:37:19
Django version 5.0.6, using settings 'proyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Al hacerlo verás que en la carpeta se crea el archivo db.sqlite3



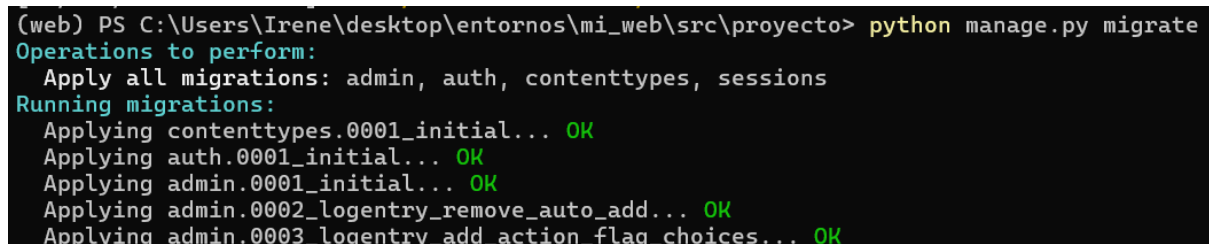
### Aplicar migraciones

En el anterior paso vemos en rojo que no tenemos las migraciones aplicadas.  
queremos ejecutar <http://127.0.0.1:8000/admin>, pero sin migraciones dará error.

## End-to-end task manager con Django



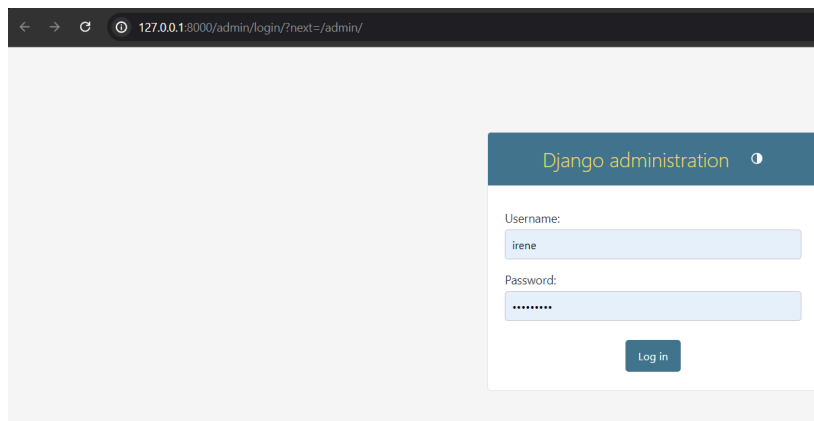
Para solucionarlo, escribimos control c para salir de la ejecución, y después  
*python manage.py migrate*



Volvemos a ejecutar el servidor en terminal

*python manage.py runserver*

Volvemos <http://127.0.0.1:8000/admin>



### 3. Generar usuario

ctr c

*python manage.py createsuperuser*

## End-to-end task manager con Django

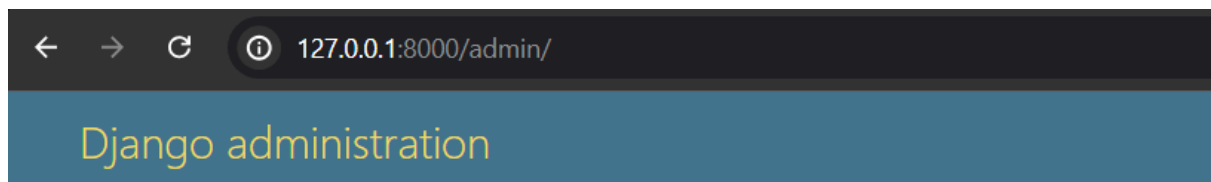
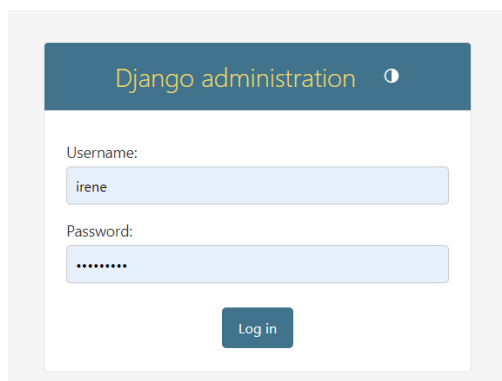
```
(web) PS C:\Users\Irene\desktop\entornos\mi_web\src\proyecto> python manage.py createsuperuser  
Username (leave blank to use 'irene'):
```

```
Superuser created successfully.  
(web) PS C:\Users\Irene\desktop\entornos\mi_web\src\proyecto>
```

<<< Hay que volver a ejecutar el server después de crear USER >>>

*python manage.py runserver*

<http://127.0.0.1:8000/admin/login/?next=/admin/>



### Site administration

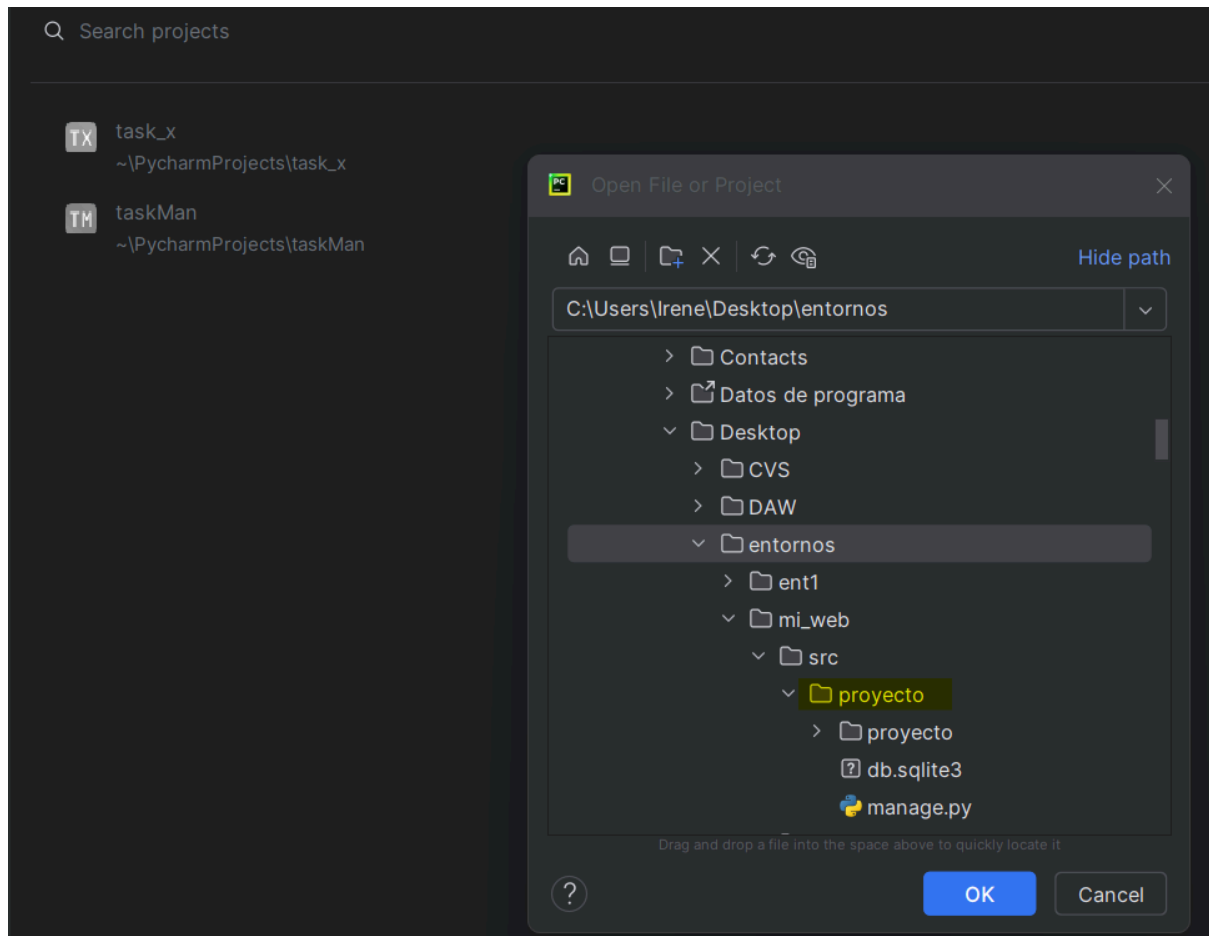
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

## I ESTRUCTURA Y PROYECTO BÁSICO

### 4. Configurar las URLs

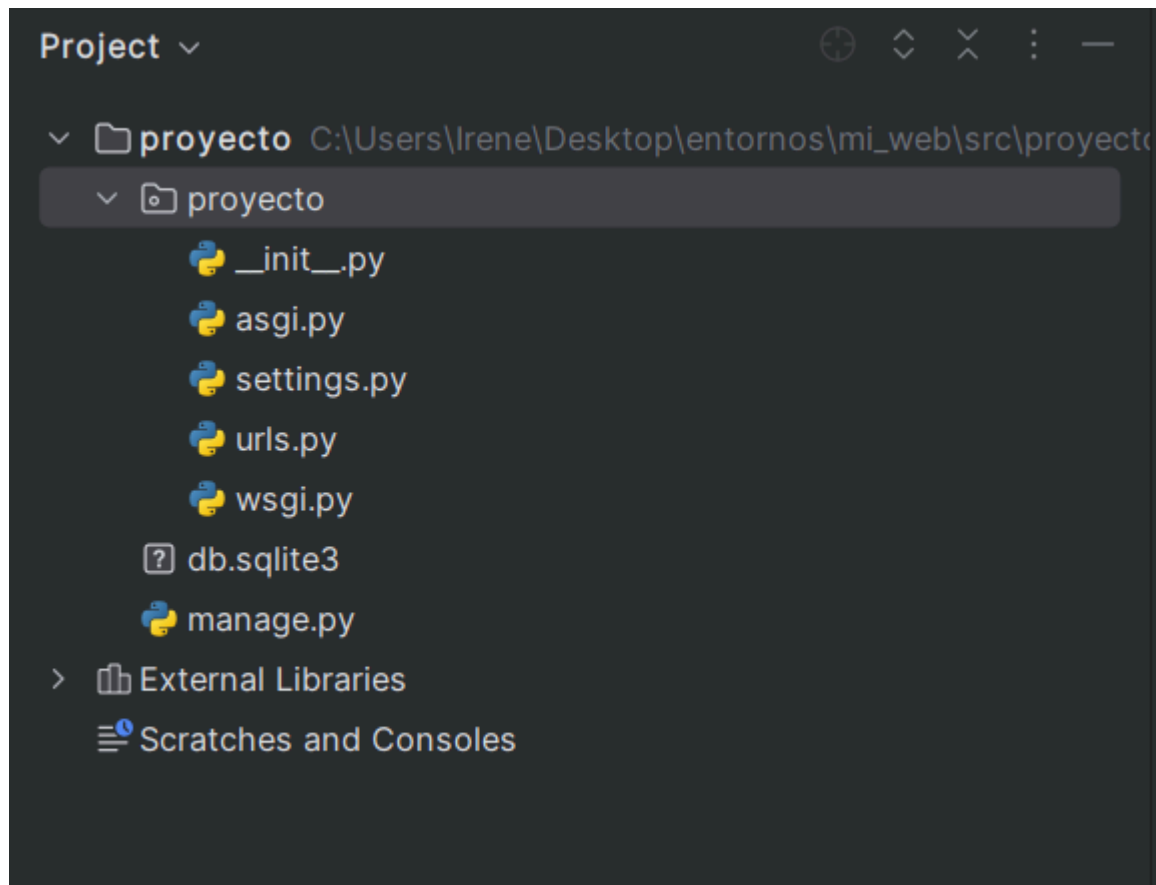
Vamos a ir a PyCharm para empezar con los archivos python y configurar las URLs para verlas en navegador

## Abrimos proyecto directamente



-----

Al hacerlo tendremos la siguiente vista:

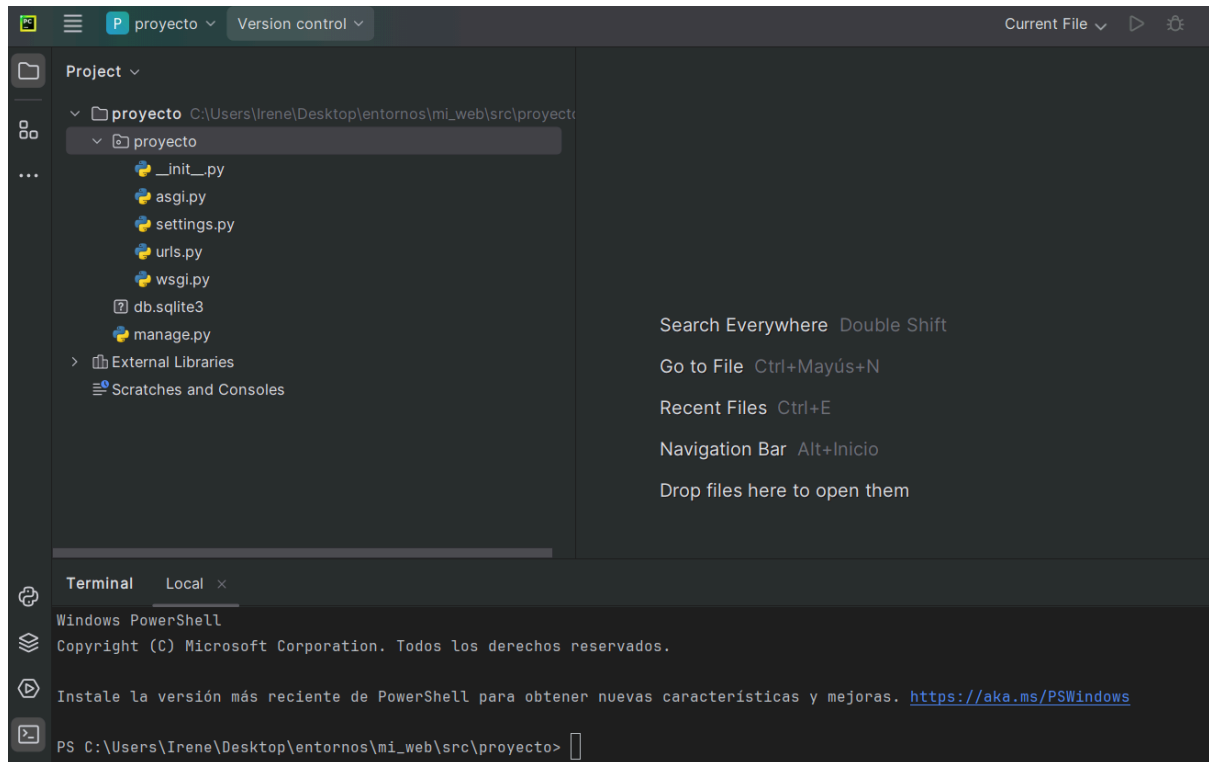


### Crear una app

Las apps normalmente son pequeños componentes para un sitio web. En nuestro caso vamos a crear este administrador de tareas pendientes. Crearemos el archivo de nuestra app y allí pondremos el código, estructura, ajustes y lógica principal.

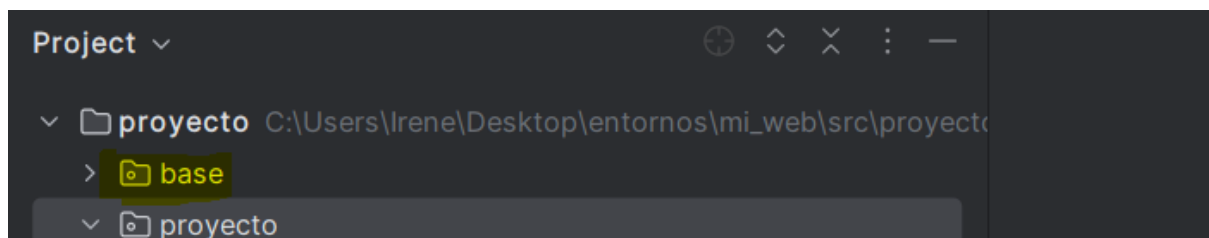
> Abrimos terminal

## End-to-end task manager con Django



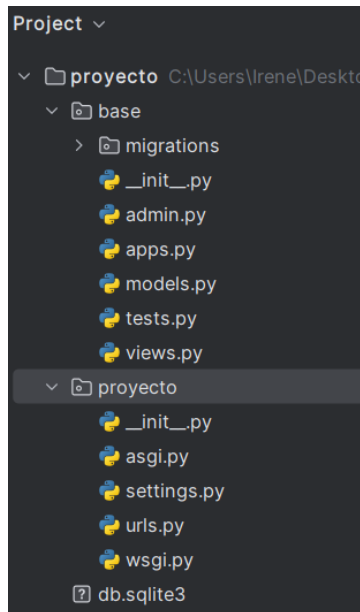
*PS C:\Users\Irene\Desktop\entornos\mi\_web\src\proyecto> python manage.py startapp base*

Vemos que se ha creado





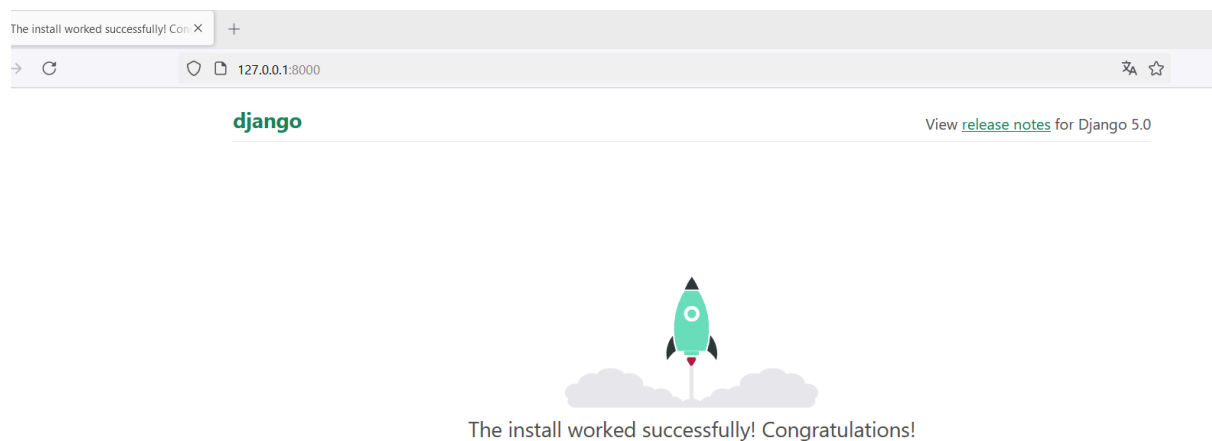
## End-to-end task manager con Django



—

Ejecutamos servidor para ver la web que tenemos hasta ahora  
python manage.py runserver

Seguimos viendo la página inicial, lo cual quiere decir que aún no tenemos nada de nuestra web



[The install worked successfully! Congratulations!](#)

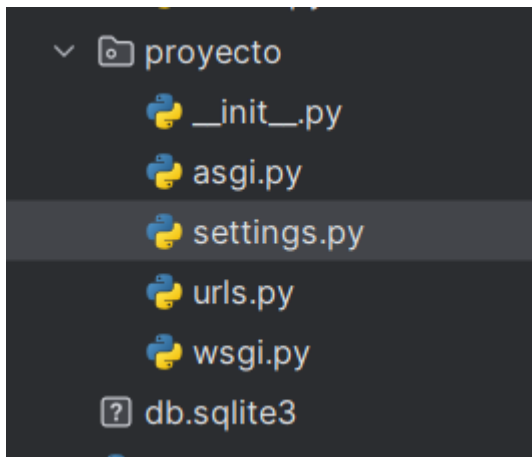
—

## Conectar base y proyecto

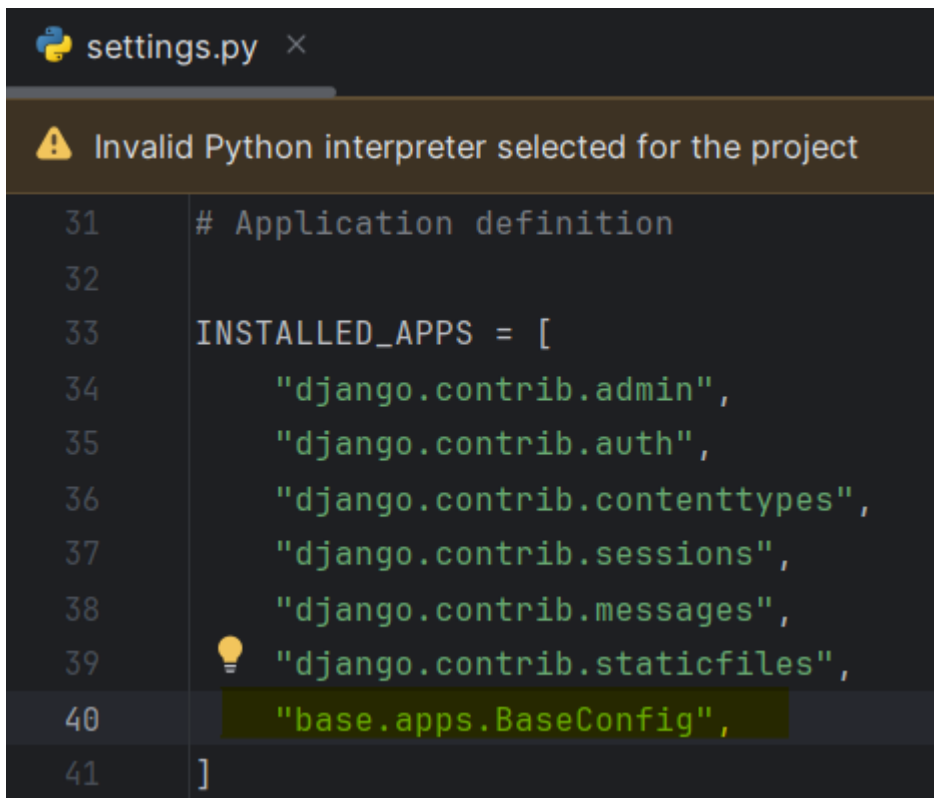
El primer paso es conectar ambas partes de la pp que hemos creado

## End-to-end task manager con Django

> Vamos a **settings** de proyecto



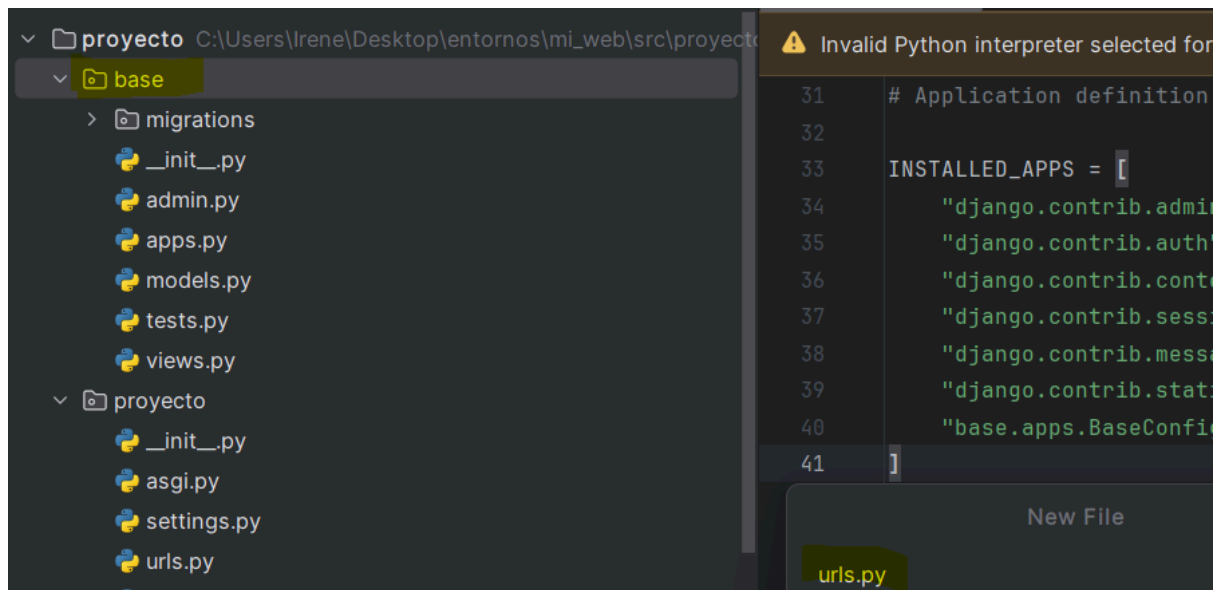
Añadimos en apps instaladas base.apps.BaseConfig (apuntando a la clase BaseConfig)



## Conectar URSS

Crear un archivo urls en base (new file urls.py)

## End-to-end task manager con Django



### Importar mis vistas

Vamos a nuestro nuevo archivo urls de base y escribimos:

```
from django.urls import path
from . import views
```

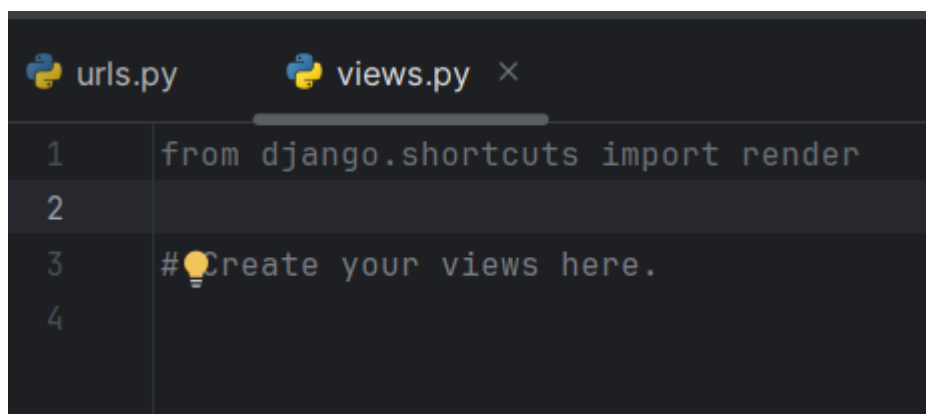
Añadimos los patrones de URL en el mismo archivo

```
urlpatterns = []
```

El archivo urls se ve así:

```
from django.urls import path
from . import views
urlpatterns = []
```

En base, vamos a views



## End-to-end task manager con Django

Importamos http response:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
```

### Creamos primera vista

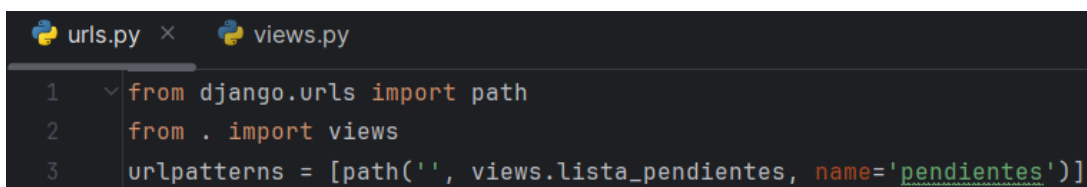
Seguimos en Views de base, creamos función que devuelve respuesta http:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def lista_pendientes(pedido):
    return HttpResponse('Lista de cosas pendientes')
```

Volvemos a urls donde estaban nuestros urls patterns (en base)

Añadimos nuestro primer patrón, con una cadena vacía para que sea la dirección de llegada, la función y un nombre

```
urlpatterns = [path('', views.lista_pendientes, name='pendientes')]
```



```
1 from django.urls import path
2 from . import views
3 urlpatterns = [path('', views.lista_pendientes, name='pendientes')]
```

Vamos a URLs de proyecto (queremos que nuestro proyecto conozca las URLs que acabamos de generar)

Añadimos función include en proyecto - urls

```
from django.contrib import admin
from django.urls import path, include
```

Seguimos en URLs de proyecto, y ahora que podemos incluir otros paths, agregamos otro. Lo que hacemos es decirle que maneje la petición con el archivo de base.urls

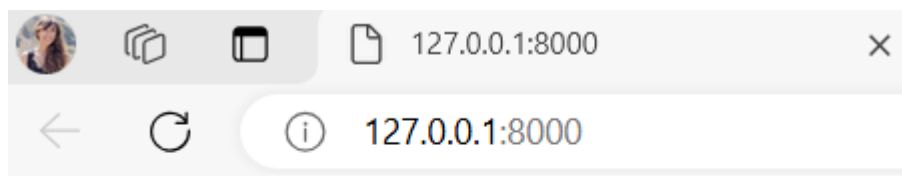
```
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include('base.urls')),
]
```

```
base\urls.py  proyecto\urls.py x  views.py
7          2. Add a URL to urlpatterns: path('', views.no
10      Class-based views
11          1. Add an import: from other_app.views import
12          2. Add a URL to urlpatterns: path('', Home.as_
13      Including another URLconf
14          1. Import the include() function: from django.u
15          2. Add a URL to urlpatterns: path('blog/', inc
16      """
17      from django.contrib import admin
18      from django.urls import path, include
19
20      urlpatterns = [
21          path("admin/", admin.site.urls),
22          path('', include('base.urls')),
23      ]
```

## Comprobación

Ahora volvemos a nuestra dirección oficial y refrescamos, para comprobar si se ejecuta la respuesta http que hemos configurado:

[127.0.0.1:8000](http://127.0.0.1:8000)



Lista de cosas pendientes

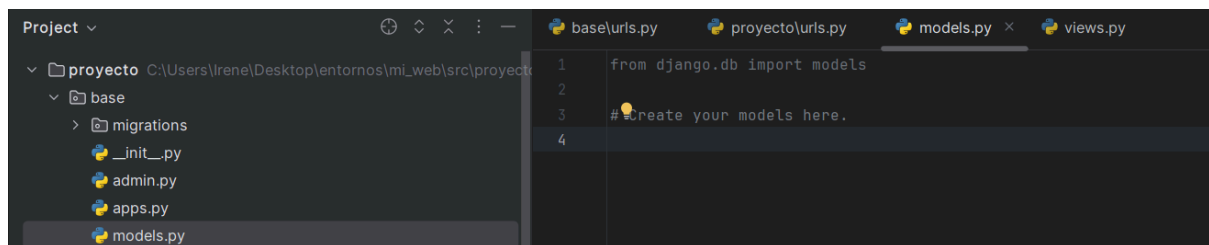
Proceso: al hacer la petición en urls de proyecto, se ha ido a urls de base, allí teníamos que se devolviera lo que hubiera en vistas, y en vistas estaba la función que devolvía una respuesta http con el String 'lista de cosas pendientes'.

### 5. Crear tabla de tareas

Vamos a necesitar disponer de una base de datos que vaya almacenando todas las tareas creadas por el usuario. Esta base de datos debe estar estructurada de acuerdo con las necesidades de registros.

Las tablas deben ser creadas dentro del archivo models.py que se encuentra en nuestra base.

Para cada tabla debe de crearse una clase. Sus atributos serán columnas//campos.



En models.py de base, creamos nuestra tabla con los campos correspondientes. Para el usuario, nos aprovechamos de la funcionalidad ya existente de Python user, que importamos.

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Tarea(models.Model):
    usuario = models.ForeignKey(User,
                                on_delete=models.CASCADE,
                                null=True,
                                blank=True)

    titulo = models.CharField(max_length=200)

    descripcion = models.TextField(null=True,
                                   blank=True)

    completo = models.BooleanField(default=False)
    creado = models.DateTimeField(auto_now_add=True)

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

```
class Meta:
    ordering = ['completo']
```

## Migración

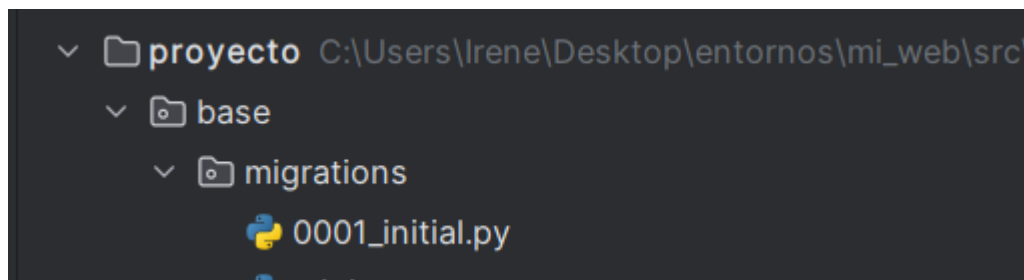
Hay dos cosas que tendremos que hacer para migrar nuestra base de datos:

> La primera es crear el archivo

Sobre la terminal:

```
PS C:\Users\Irene\Desktop\entornos\mi_web\src\proyecto> python manage.py
makemigrations
```

Esto ha generado una carpeta dentro de nuestra aplicación llamada migrations:



Dentro tenemos todos los elementos de nuestra tabla tareas que ya ha sido migrada y existe.

### > Migrar a nuestro sitio

llamamos de nuevo a python manage.py

```
PS C:\Users\Irene\Desktop\entornos\mi_web\src\proyecto> python manage.py
migrate
```

```
PS C:\Users\Irene\Desktop\entornos\mi_web\src\proyecto> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, base, contenttypes, sessions
Running migrations:
  Applying base.0001_initial... OK
PS C:\Users\Irene\Desktop\entornos\mi_web\src\proyecto> 
```

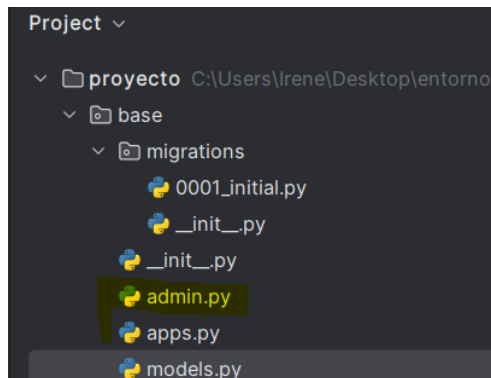
Ahora se migró esta tabla nueva en nuestro sitio. Para poder ver como se refleja esto en nuestro sitio, vamos a ejecutar nuevamente nuestro server.

```
PS C:\Users\Irene\Desktop\entornos\mi_web\src\proyecto> python manage.py
runserver
```

(Aún no se ve)

## Último paso: registrar la tabla

En base, buscamos el archivo admin.py, donde se registran los modelos



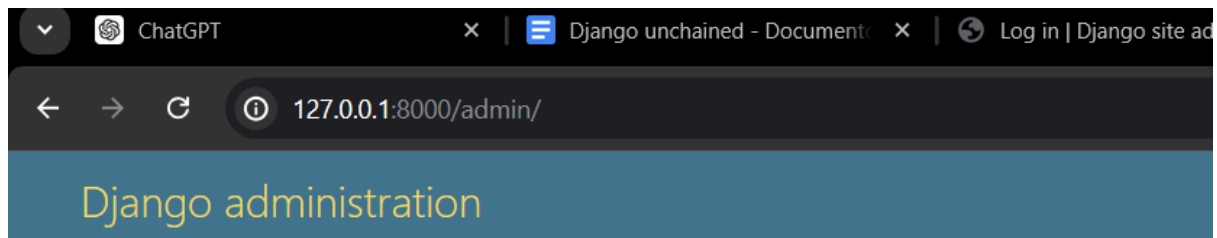
>Dentro de admin:

```
from django.contrib import admin

# Register your models here.

from .models import Tarea
admin.site.register(Tarea)
```

Ahora si volvemos a nuestro sitio y lo refrescamos, tenemos tareas. Esto todavía no se ve en el sitio, pero sí dentro de nuestra página de administración.



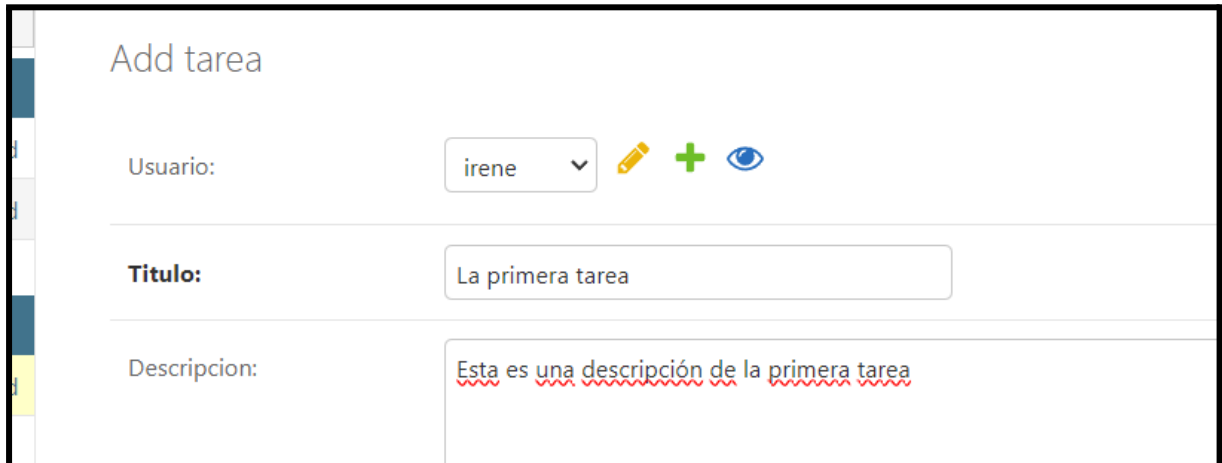
### Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change
BASE		
Tareas	+ Add	Change



## Agregar una tarea

Desde nuestra página de admin, ya podemos agregar tareas:

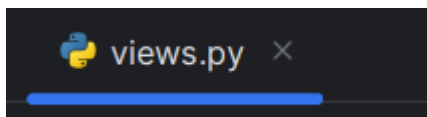


The screenshot shows the 'Add tarea' form in the Django Admin interface. It has a sidebar on the left with a blue and yellow header. The form fields are: 'Usuario:' with a dropdown menu showing 'irene' and icons for edit, add, and view; 'Titulo:' with a text input field containing 'La primera tarea'; and 'Descripción:' with a text area containing 'Esta es una descripción de la primera tarea'.

## 6. Configurar la vista

Ya tenemos nuestras primeras tareas, y las queremos listar en nuestra web.

En views.py, cambiamos algunas cosas (anteriormente tenemos la función de retornar HttpResponse)



Borramos todo e importamos listView, usado para listas y listas filtradas por criterio

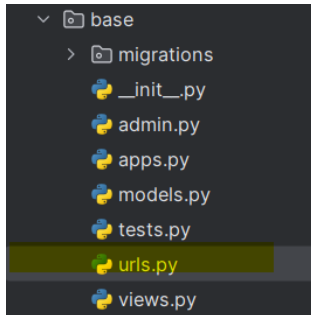
```
from django.views.generic.list import ListView
```

Creamos clase listaPendientes, a la que pasaremos la tabla(modelo) tarea y una query que definimos más adelante

```
from django.shortcuts import render
# Create your views here.
from django.views.generic.list import ListView
from .models import Tarea
class ListaPendientes(ListView):
```

```
model = Tarea
```

Modificar URLs también



Conectamos a nuestra clase como vista e importamos view de otra manera

```
from django.urls import path
from .views import ListaPendientes
urlpatterns = [path('', ListaPendientes.as_view(), name='pendientes')]
```

Configurar plantilla

Al meter la url <http://127.0.0.1:8000/>

Vemos un error “plantilla no existe”, junto con una dirección de plantilla proporcionada.

base/tarea\_list.html

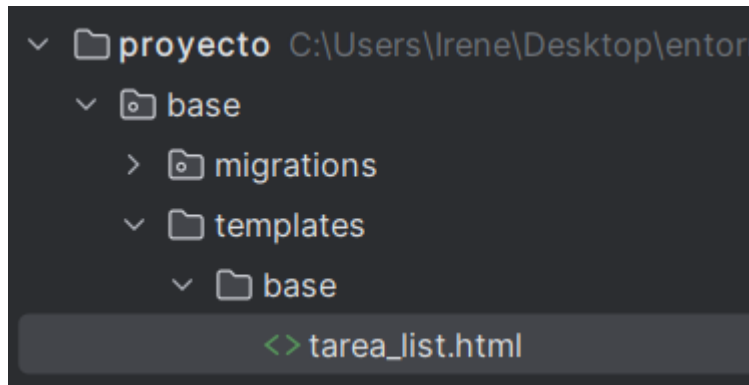
TemplateDoesNotExist at /  
base/tarea\_list.html

Request Method: GET

Request URL: http://127.0.0.1:8000/

Django Version: 5.0.6

Dentro de base, creamos directorio templates, dentro del mismo el directorio “base de nuevo”, y finalmente el archivo tarea\_list.html

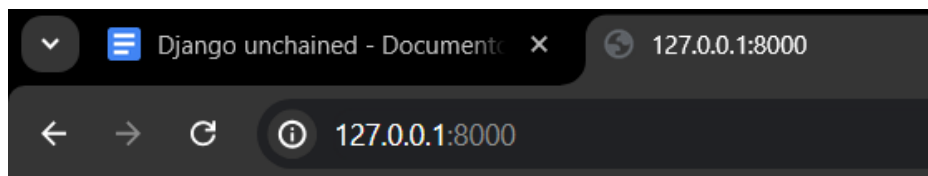


## Configurar settings en proyecto

En el archivo settings del proyecto, en Templates → dirs actualizamos a la ubicación del archivo (templates). Copiamos la ruta absoluta de “templates” y la pegamos con comilla y doble barra

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS":
        ['C:\\Users\\Irene\\Desktop\\entornos\\mi_web\\src\\proyecto\\base\\templates'],
        "APP_DIRS": True,
```

Con esta conexión, lo que escribamos en html ya nos debe aparecer en el navegador



## Lista de tareas pendientes

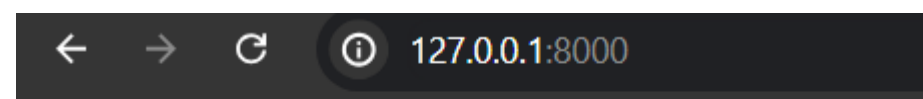
## Plantilla inicial

La plantilla itera con un bucle for y comprueba si la lista está vacía. Hemos escrito simplemente “ object list”, en el paso siguiente lo modificamos.

```
<h1>Lista de tareas pendientes</h1>
<table>
  <tr>
    <th>
      ELEMENTOS
    </th>
  </tr>
  {% for tarea in object_list%}
  <tr> <td> {{tarea.titulo}}</td></tr>

  {% empty %}
  <h3> No hay elementos en esta lista</h3>
  {% endfor %}

</table>
```



# Lista de tareas pendientes

## ELEMENTOS

La segunda tarea

La tercera tarea

La primera tarea

## Pasar lista

En views de base:

cambiamos object name pasándolo como atributo a la clase:

```
from django.shortcuts import render
# Create your views here.
from django.views.generic.list import ListView
from .models import Tarea
class ListaPendientes(ListView):
    model = Tarea
    context_object_name = 'tareas'
```

En HTML:

modificamos object\_list por tareas:

```
<h1>Lista de tareas pendientes</h1>
<table>
  <tr>
    <th>
      ELEMENTOS
    </th>
  </tr>
  {% for tarea in tareas%}
  <tr> <td> {{tarea.titulo}}</td></tr>

  {% empty %}
  <h3> No hay elementos en esta lista</h3>
  {% endfor %}

</table>
```

## 7. Configurar la vista de detalle

Queremos la información concreta para cada elemento de la lista. Nuestro primer paso es crear una vista de detalles.

### Importar vista detalle y clase

En views, importamos detalle:

```
from django.views.generic.detail import DetailView
```

Creamos la clase DetalleTarea:

```
class DetalleTarea(DetailView):  
    model = Tarea
```

Creamos otro archivo HTML

```
▼ base  
  <> tarea_detail.html  
  <> tarea_list.html
```

## Editar URLs

Después vamos a URLs de nuestra app base y añadimos el path para tarea con un int dependiente de pk

```
from django.urls import path  
from .views import ListaPendientes, DetalleTarea  
urlpatterns = [path("", ListaPendientes.as_view(), name='ListaPendientes'),  
               path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea')]
```



← → ↻ ⓘ 127.0.0.1:8000/tarea/1

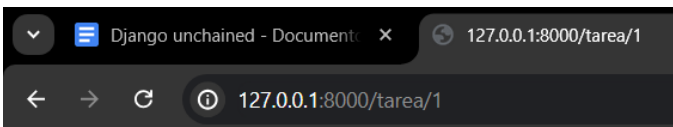
# TAREA:

## Editar vista detalle

En tarea\_detail:

```
<h1>TAREA: {{object}} </h1>
```

Al escribir esto ya nos devuelve el título de nuestro objeto tarea:



▼ Django unchained - Document... x 127.0.0.1:8000/tarea/1  
← → ↻ ⓘ 127.0.0.1:8000/tarea/1

## TAREA: La primera tarea

Personalizar 'object' →

En Views, vamos a nuestra clase y agregamos un object name:

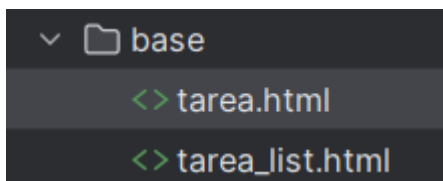
```
class DetalleTarea(DetailView):  
    model = Tarea  
    context_object_name = 'tarea'
```

→ Después en HTML cambiamos object por tarea:

## Cambiar nombre plantilla

Ahora, por último, sobre escribimos la plantilla para que la busque por un nombre que elijamos, details se había generado por defecto.

>> Cambiamos el nombre al html con refactor

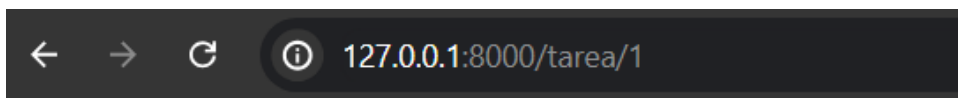


```
▼ base  
  <> tarea.html  
  <> tarea_list.html
```

>> Volvemos a views e incluimos como argumento el nombre de la plantilla

```
class DetalleTarea(DetailView):  
    model = Tarea  
    context_object_name = 'tarea'  
    template_name = 'base/tarea.html'
```

>> Comprobamos que todo sigue funcionando

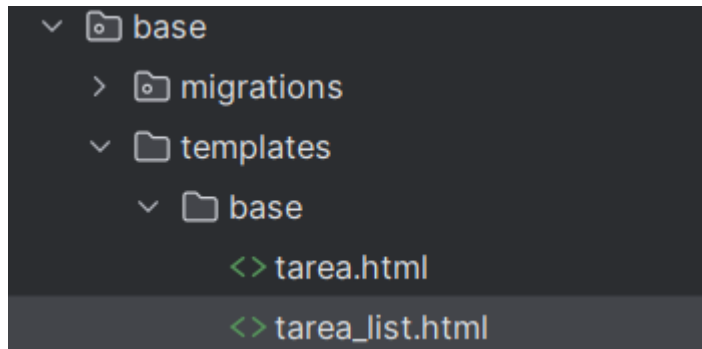


```
← → ↻ ⓘ 127.0.0.1:8000/tarea/1
```

# TAREA: La primera tarea

## 8. Crear links al detalle

En nuestro home tareas\_list



Añadimos encabezado vacío y un campo de enlace que vaya a tarea + int de pk

```
<h1>Lista de tareas pendientes</h1>
<table>
  <tr>
    <th>
      ELEMENTOS
    </th>
    <th> </th>
  </tr>
  {% for tarea in tareas%}
  <tr> <td> {{tarea.titulo}}</td>
    <td> <a href="{% url 'tarea' tarea.id %}">VER</a></td>
  </tr>

  {% empty %}
  <h3> No hay elementos en esta lista</h3>
  {% endfor %}

</table>
```

Al refrescar nuestra home ya podemos clicar





## 9. Agregar una nueva tarea

En esta sección vamos a generar un método para que el usuario pueda generar nuevos items desde la página que hemos creado.

### Clase crear vista

En views base agregamos una importación más

```
from django.views.generic.edit import CreateView
```

Creamos la clase crear tarea, también en views

```
class CrearTarea(CreateView):  
    model= 'tarea'
```

Creamos archivo tarea\_form.html

–

Volvemos a views y en nuestra clase especificamos los campos que queremos, en este caso all

```
class CrearTarea(CreateView):  
    model= 'tarea'  
    fields = '__all__'
```

### Lazy reverse

Ahora agregamos la página a la que dirigiremos al usuario tras haber completado el form, también sobre nuestra clase tarea.

>> Importamos django.urls reverse\_lazy

```
from django.urls import reverse_lazy
```

>> Lo usamos como atributo en la clase

```
class CrearTarea(CreateView):  
    model= 'tarea'  
    fields = '__all__'  
    success_url = reverse_lazy('tareass')
```

## Agregar path

En base URLs, importamos la clase y creamos el nuevo path

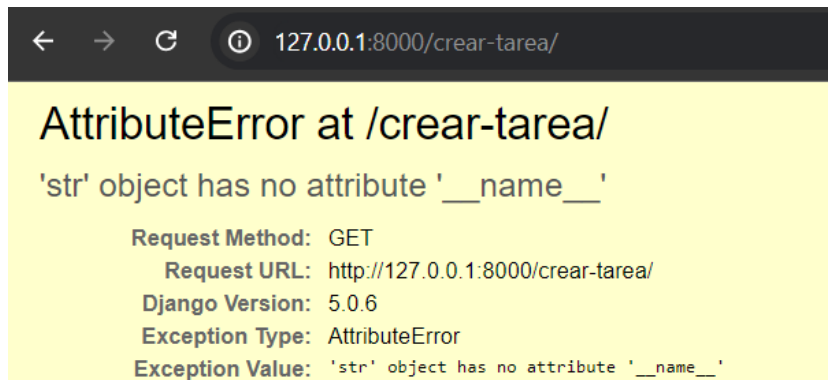
```
from django.urls import path
from .views import ListaPendientes, DetalleTarea, CrearTarea
urlpatterns = [path("", ListaPendientes.as_view(), name='ListaPendientes'),
               path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea'),
               path('crear-tarea/', CrearTarea.as_view(), name='crear-tarea')]
```

\*\*no olvidad la barra en las urls

Probamos la url

<http://127.0.0.1:8000/crear-tarea/>

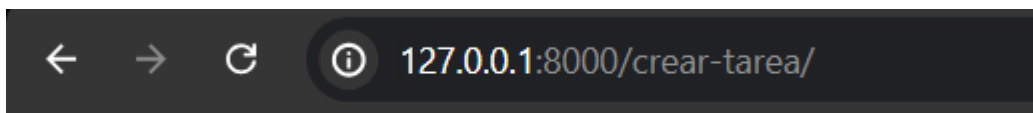
vemos error



corregimos el modelo a tarea sin comillas en la clase CrearTarea

```
2 usages
class CrearTarea(CreateView):
    model = Tarea
    fields = '__all__'
```

Actualizamos y ya aparece nuestro form



# Formulario de tareas

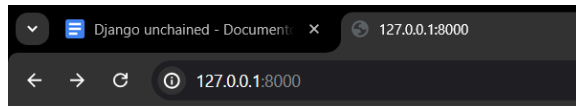
Agregar enlace a form

Vamos al html tarea\_list

>> Agregamos etiqueta de enlace después del encabezado y voilà

```
<h1>Lista de tareas pendientes</h1>
<br>
<a href="{% url 'crear-tarea' %}">CREAR NUEVA TAREA</a>
<br>

<table>
```



### Lista de tareas pendientes

[CREAR NUEVA TAREA](#)

#### ELEMENTOS

La segunda tarea [VER](#)

La tercera tarea [VER](#)

La primera tarea [VER](#)

## 10. Formulario para nueva tarea

### Reflejar el form

Ahora que ya tenemos un enlace, y una página para el form, nos queda crear el formulario.

Comenzaremos con la etiqueta form en nuestro html de formulario.

En realidad, gracias a la clase de la que heredamos el form “ya está creado”, solo hay que reflejarlo:

```
<h1>Formulario de tareas</h1>
<form method="post" action="" >
{{form.as_p}}

    <input type="submit" value=" ENVIAR " >
</form>
```

## End-to-end task manager con Django



← → ↻ ⓘ 127.0.0.1:8000/crear-tarea/

### Formulario de tareas

Usuario:

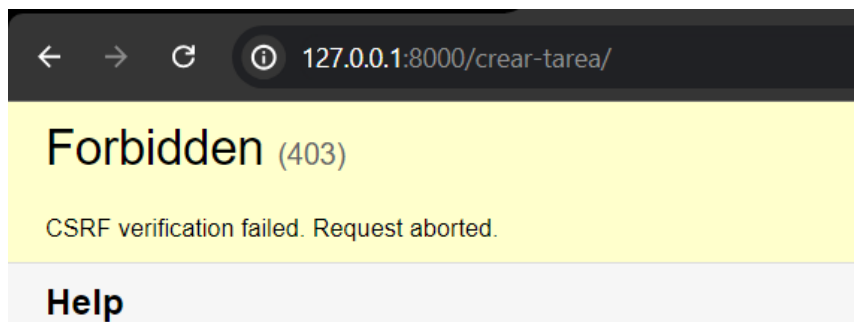
Titulo:

Descripcion:

Completo: ☐

Crear nueva tarea (error)

Al completar tarea y enviar tendremos un error



## Corregir la verificación

En tarea\_form.html

Añadimos token para que nos permita realizar el proceso.

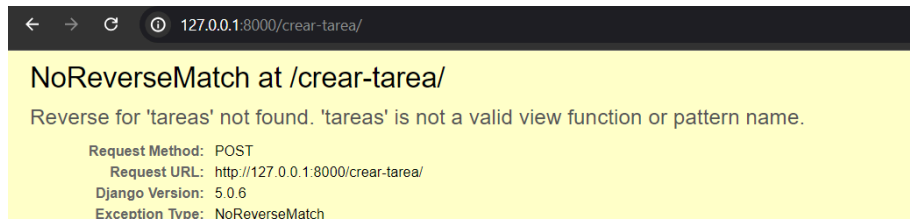
```
<h1>Formulario de tareas</h1>
<form method="post" action="" >

{% csrf_token %}

{{form.as_p}}
  <input type="submit" value=" ENVIAR ">
</form>
```

Probamos de nuevo a crear una tarea y vuelve a dar error. Esto es porque después de enviar deberíamos volver a 'tareas', pero la app no encuentra que le hayamos indicado esto.

## End-to-end task manager con Django



En nuestro `reverse_lazy` de views, modificamos el nombre a pendientes para que coincida con el path

```
class CrearTarea(CreateView):
    model = Tarea
    fields = '__all__'
    success_url = reverse_lazy('ListaPendientes')
```

Comprobamos:

### Lista de tareas pendientes

[CREAR NUEVA TAREA](#)

#### ELEMENTOS

La segunda tarea [VER](#)

La tercera tarea [VER](#)

ññ [VER](#)

ññ [VER](#)

kkkk [VER](#)

La primera tarea [VER](#)

(Finalmente como era mejor nombre hemos puesto 'tareas' tanto en el path como en el lazy reverse.)

----

## 11. Editar una tarea

Editar botón de volver en el formulario

```
<h1>Formulario de tareas</h1>
```

```
<a href="{% url 'tareas' %}"> VOLVER </a>
```

```
<form method="post" action="" >

{% csrf_token %}

{{form.as_p}}
    <input type="submit" value=" ENVIAR " >

</form>
```

En views, importamos update y creamos clase

```
from django.views.generic.edit import CreateView, UpdateView
```

```
class EditarTarea(UpdateView):
    model = Tarea
    fields = '__all__'
    success_url = reverse_lazy('tarefas')
```

Editamos urls

```
path('editartarea/<int:pk>', EditarTarea.as_view(), name='editar-tarea']
```

Añadimos la opción editar en el HTML de la lista

```
<td> <a href="{% url 'tarea' tarea.id %}">VER</a></td>
<td> <a href="{% url 'editar-tarea' tarea.id %}">EDITAR</a></td>
```

## 12. Eliminar tarea

Enlace de eliminación en html

```
<td> <a href="{% url 'editar-tarea' tarea.id %}">EDITAR</a></td>
<td> <a href="">ELIMINAR</a></td>
```

Importamos clase que verifica la eliminación y elimina (views)

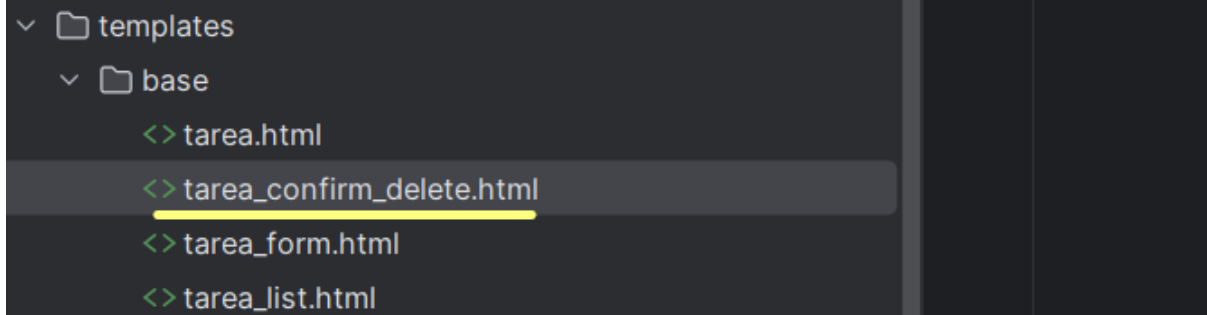
```
from django.views.generic.edit import CreateView, UpdateView, DeleteView
```

```
class BorrarTarea(DeleteView):
    model = Tarea
```

## End-to-end task manager con Django

```
context_object_name = 'tareas'  
success_url = reverse_lazy('tareas')
```

### Creamos html



### Edición html borrado

```
<a href="{% 'tareas' %}">VOLVER</a>  
<form method="post">  
  {% csrf_token %}  
  <p>Vas a eliminar esta tarea: "{{tarea}}"</p>  
  <input type="submit" value="ELIMINAR">  
</form>
```

### Editar URLs

```
from .views import ListaPendientes, DetalleTarea, CrearTarea,  
EditarTarea, BorrarTarea  
urlpatterns = [path('', ListaPendientes.as_view(), name='tareas'),  
                path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea'),  
                path('crear-tarea/', CrearTarea.as_view(), name='crear-tarea'),  
                path('editartarea/<int:pk>', EditarTarea.as_view(), name='editar-tarea'),  
                path('borrar-tarea/<int:pk>', BorrarTarea.as_view(), name='borrar-tarea')]
```

### Editar enlace en tarea-list

```
{% for tarea in tareas%}  
<tr> <td> {{tarea.titulo}}</td>  
  <td> <a href="{% url 'tarea' tarea.id %}">VER</a></td>  
  <td> <a href="{% url 'editar-tarea' tarea.id %}">EDITAR</a></td>  
  <td> <a href="{% url 'borrar-tarea' tarea.id %}">ELIMINAR</a></td>
```

### Corregir nombre de la tarea en views

```
context_object_name = 'tarea'
```

## II REGISTRO Y USUARIOS

### 13. Lógica de logueo/deslogueo

El primer paso es comprobar si el usuario está logueado y, si no lo está, proporcionarle un enlace para que pueda registrarse. Esto lo podemos hacer con un condicional de la autenticación.

Editar la home, en tarea-list

```
{% if request.user.is_authenticated %}

<p>{{request.user}}</p>

<a href="">SALIR</a>

{% else %}
<a href="">INGRESAR</a>

{% endif %}

<hr>
```

irene

[SALIR](#)

## Lista de tareas pendientes

### 14. Formulario logueo/deslogueo

Configurar logueo

Importar vista en views y crear la clase de logueo.

Además, sobrescribimos el método necesario para redirigir después del logueo:

```
from django.contrib.auth.views import LoginView
```



## End-to-end task manager con Django

```
class Logueo(LoginView):
    template_name = "base/login.html"
    fields = '__all__'
    redirect_authenticated_user = True

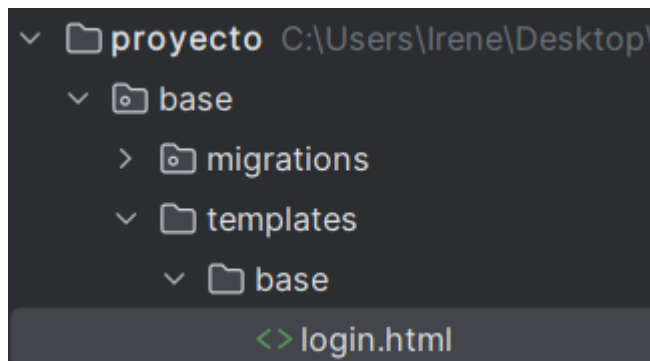
    def get_success_url(self):
        return reverse_lazy('tareas')
```

### En URLs

```
from .views import ListaPendientes, DetalleTarea, CrearTarea,
EditarTarea, BorrarTarea, Logueo
urlpatterns = [path("", ListaPendientes.as_view(), name='tareas'),
               path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea'),
               path('crear-tarea/', CrearTarea.as_view(), name='crear-tarea'),
               path('editartarea/<int:pk>', EditarTarea.as_view(), name='editar-tarea'),
               path('borrar-tarea/<int:pk>', BorrarTarea.as_view(), name='borrar-tarea'),
               path('login/', Logueo.as_view(), name='login')]
```

### En Templates

Creamos plantilla de login



```
<h1>INGRESAR</h1>
<form METHOD="post">

{% csrf_token %}
{{form.as_p}}

    <input type="submit" value="INGRESAR">
</form>
```

### En tarea\_list.html

Volvemos para completar los enlaces que habíamos dejado pendientes.

```
<a href="{% url 'login' %}">INGRESAR</a>
```

## End-to-end task manager con Django

Comprobamos

(\*nota: para probar a simular deslogueo, clic derecho sobre nuestra home en Navegador: inspect→applications→cookies→ borrar sessionId)



← → 🔍 127.0.0.1:8000/login/

# INGRESAR

Username:

Password:

## Configurar deslogueo

En URLs

```
from django.contrib.auth.views import LogoutView
path('logout/', LogoutView.as_view(next_page='login'), name='logout')
```

EN tarea\_list, en SALIR

Referenciamos a nuestro logout

```
<a href="{% url 'logout' %}">SALIR</a>
```

\*\*\* Corrección, el método anterior de redirección no funcionaba, hemos cambiado el método para garantizar que es post:

```
{% if request.user.is_authenticated %}

<p>{{request.user}}</p>

<form method="post" action="{% url 'logout' %}">

    {% csrf_token %}
    <input type="submit" value="SALIR">
</form>
```

Comprobamos:

irene

SALIR

---

## 15. Restringir acceso

Vamos a editar nuestro código para que la mayoría de las vistas no sean accesibles sin estar logueado.

### Importar restricciones

En views

```
from django.contrib.auth.mixins import LoginRequiredMixin
```

Hacemos que nuestra lista de tareas herede esta clase también, y vemos lo que ocurre al intentar acceder a ella sin estar registrado:

```
class ListaPendientes(ListView, LoginRequiredMixin):  
    model = Tarea  
    context_object_name = 'tareas'
```

### Page not found (404)

Request Method: GET

Request URL: http://127.0.0.1:8000/accounts/login/?next=/  

---

Using the URLconf defined in proyecto.urls, Django tried these URL patterns, in this order:

### Editar página de error

EN SETTINGS, se añade log de registro, así desde la home se nos lanza a logueo

```
USE_TZ = True
```

```
LOGIN_URL = 'login'
```

```
# Static files (CSS, JavaScript, Images)
```

```
# https://docs.djangoproject.com/en/5.0/howto/static-files/
```

## Más restricciones

```
class DetalleTarea(LoginRequiredMixin, DetailView):
```

```
class CrearTarea(LoginRequiredMixin, CreateView):
```

## 15. Establecer información específica del usuario

Crear nuevo usuario desde el panel administrador

usuario

SALIR

Para lograr que cada usuario vea sus tareas nada más, vamos a emplear un método que se llama get context data. Vamos a sobrescribir este método para que se comporte como nosotros queremos dentro de la clase de tareas pendientes.

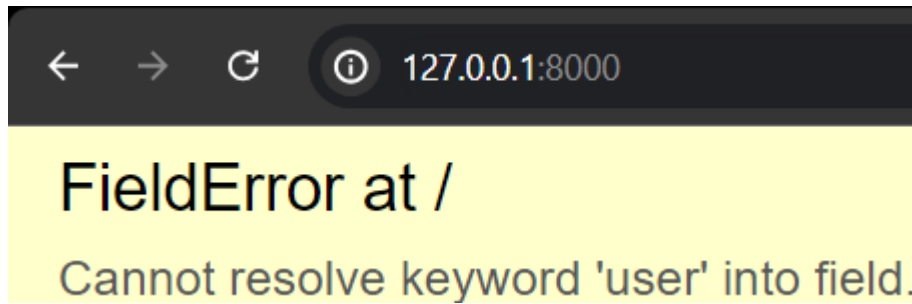
## Context filtro

Dentro de la clase ListaPendientes, generamos una función.

```
class ListaPendientes(LoginRequiredMixin, ListView):
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['tareas'] = context['tareas'].filter(user=self.request.user)
        context['count'] = context['tareas'].filter(completo=False).count()

        return context
```

Refrescamos como usuario nuevo y vemos un error, no resuelve la palabra user dentro del campo  
¿Por qué? → hemos empleado 'user', pero el campo creado para identificar a los users es usuario



Corrección:

```
context['tareass'] = context['tareass'].filter(usuario=self.request.user)
```

usuario

SALIR

---

## Lista de tareas pendientes

[CREAR NUEVA TAREA](#)

No hay elementos en esta lista

—

### Asignar autor automáticamente

Nuestro campo de selección de usuario en el formulario de creación de tareas ya no nos resulta útil, vamos a modificar asignar de forma automática la tarea al usuario registrado.

En views, crear-tarea:

```
class CrearTarea(LoginRequiredMixin, CreateView):
    model = Tarea
    fields = '__all__'
    success_url = reverse_lazy('tareass')
    def form_valid(self, form):
        form.instance.usuario = self.request.user
        return super(CrearTarea, self).form_valid(form)
```

Y modificamos los campos de crear tarea, que ya no son todos (\_\_all\_\_)

```
fields = ['titulo', 'descripcion', 'completo']
```

(también lo modificamos en editar\_tarea).

## Formulario de tareas

[VOLVER](#)

Título:

Descripción:

Completo: ☐

## Crear un nuevo usuario

*Pasos: enlace, formulario, vista, URL*

### Enlace

En login.html

```
<p>no tienes una cuenta? <a href="">REGÍSTRATE</a></p>
```

### Formulario

Nuevo HTML (registro.html)

```
<h1>REGISTRARSE</h1>
<form method="post">

{% csrf_token %}
{{ form.as_p }}

    <input type="submit" value="REGISTRARSE">
</form>
<p>YA tienes una cuenta? <a href="{% url 'login' %}">INGRESA</a></p>
```

## Vista

En views:

Imports

```
from django.views.generic.edit import CreateView, UpdateView, DeleteView,
FormView
```

```
from django.contrib.auth.forms import UserCreationForm
```

```
from django.contrib.auth import login
```

Clase

```
class PaginaRegistro(FormView):

    template_name = 'base/registro.html'

    form_class = UserCreationForm

    redirect_authenticated_user= True

    success_url = reverse_lazy('tareass')
```

## URL

```
from .views import ListaPendientes, DetalleTarea, CrearTarea,
EditarTarea, BorrarTarea, Logueo, PaginaRegistro
```

```
path('registro/', PaginaRegistro.as_view(), name='registro'),
```

—

Volvemos al enlace del form para completarlo

```
<p>no tienes una cuenta? <a href="{% url 'registro' %}">REGÍSTRATE</a></p>
```

## End-to-end task manager con Django

### Comprobación

← → ↻ 127.0.0.1:8000/registro/

# REGISTRARSE

Username:  Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:  Enter the same password as before, for verification.

REGISTRARSE

YA tienes una cuenta? [INGRESA](#)

### Edición del form por defecto

En Django>>Proyecto>>Settings (cambiamos en por el español)

```
LANGUAGE_CODE = "es-es"
```

---

# REGISTRARSE

Nombre de usuario:  Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/-/\_

### Logueo automático tras el registro

Queremos que, una vez registrado un nuevo usuario, se le ingrese (loguee) automáticamente para evitar que tenga que hacer una tarea redundante.

En views

Editamos nuestra clase registro:

```
class PaginaRegistro(FormView):
    template_name = 'base/registro.html'
    form_class = UserCreationForm
    redirect_authenticated_user= True
    success_url = reverse_lazy('tarefas')

    def form_valid(self, form):
        usuario = form.save()
        if usuario is not None:
            login(self.request, usuario)
        return super(PaginaRegistro, self).form_valid(form)
```



Sobre escribimos form valid para que no nos mande a la página de registro si estamos ya logueados (ahora mismo nos deja hacerlo).

En views,

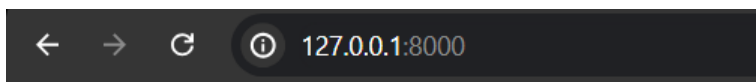
```
from django.shortcuts import render, redirect
```

> PaginaRegistro

```
class PaginaRegistro(FormView):
    template_name = 'base/registro.html'
    form_class = UserCreationForm
    redirect_authenticated_user= True
    success_url = reverse_lazy('tareass')

    def form_valid(self, form):
        usuario = form.save()
        if usuario is not None:
            login(self.request, usuario)
            return super(PaginaRegistro, self).form_valid(form)
    def get(self, *args, **kwargs):
        if self.request.user.is_authenticated:
            return redirect('tareass')
        return super(PaginaRegistro, self).get(*args, **kwargs)
```

Ahora al meter url registro logueados nos redirige a la principal



usuario

SALIR

## Lista de tareas pendientes

### III Completar la aplicación

#### 16. Barra de búsqueda

Con esta funcionalidad permitimos que el usuario encuentre por palabra o fragmento de palabra. Primero creamos tareas con palabra clave para probar.

### ELEMENTOS

La tarea 1 de pedro [VER](#) [EDITAR](#) [ELIMINAR](#)

Tarea importante numero 1 [VER](#) [EDITAR](#) [ELIMINAR](#)

Tarea importante numero 2 [VER](#) [EDITAR](#) [ELIMINAR](#)

## Edición formulario

En tarea\_list(html)

```
<h1>Lista de tareas pendientes</h1>
<br>
<a href="{% url 'crear-tarea' %}">CREAR NUEVA TAREA</a>
<br>
<form method="GET">
<input type="text" name="area-buscar">
<input type="submit" value="Buscar">
```

## Vista

En views:

Recordemos que en context data, ya teníamos algunos filtros que habíamos creado para que se nos mostrasen solo las tareas creadas por el usuario registrado. Ahora queremos agregar que solo muestre las de la caja de texto

>> En lista de pendientes, en la función get context data

Creamos una variable de valor que asociamos a lo que haya en el área de texto que hemos creado. Después filtramos el contexto en función de si su título contiene el valor que hemos guardado en la variable

```
def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['tareas'] = context['tareas'].filter(usuario=self.request.user)
    context['count'] = context['tareas'].filter(completo=False).count()

    valor_buscado = self.request.GET.get('area-buscar', '')
    if valor_buscado:
        context['tareas'] = context['tareas'].filter(titulo__icontains=valor_buscado)

    return context
```

### CREAR NUEVA TAREA

#### **ELEMENTOS**

Tarea importante numero 1 [VER](#) [EDITAR](#) [ELIMINAR](#)

Tarea importante numero 2 [VER](#) [EDITAR](#) [ELIMINAR](#)

Finalmente añadimos un detalle para que la palabra no desaparezca de la caja de texto al clicar.

En views

```
valor_buscado = self.request.GET.get('area-buscar', '')
if valor_buscado:
    context['tareas'] = context['tareas'].filter(titulo__icontains=valor_buscado)
    context['valor_buscado'] = valor_buscado
```

En tarea-list.html

```
<form method="GET">
<input type="text" name="area-buscar" value="{{valor_buscado}}">
  <input type="submit" value="Buscar">
</form>
```

## Dar estilo a vistas

Primero hemos eliminado la opción de “VER” directamente sobre el html de tarea-list para tener un diseño más claro.

#### **ELEMENTOS**

Tarea importante numero 1 [EDITAR](#) [ELIMINAR](#)

Ahora, vamos a darle estilo. Seleccionamos un estilo que puedan heredarlas vistas para no aplicar individualmente:

>> Creamos archivo principal.html, del que heredarán el resto de plantillas, con una estructura html y etiqueta de estilo para probar:

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
<meta charset="UTF-8">
<title>Lista de tareas pendientes</title>
<style>
  body {
    background-color: aqua;
  }
</style>
</head>
<body>

<div class = "container">
  {% block content %}
  {% endblock content %}
</div>

</body>
</html>
```

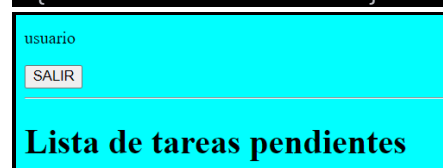
En tareas\_list:

en la parte superior del html, escribimos

```
{% extends 'base/principal.html' %}
```

y metemos todo el resto de pagina entre las etiquetas de block content

```
{% block content %}
{% endblock content %}
```



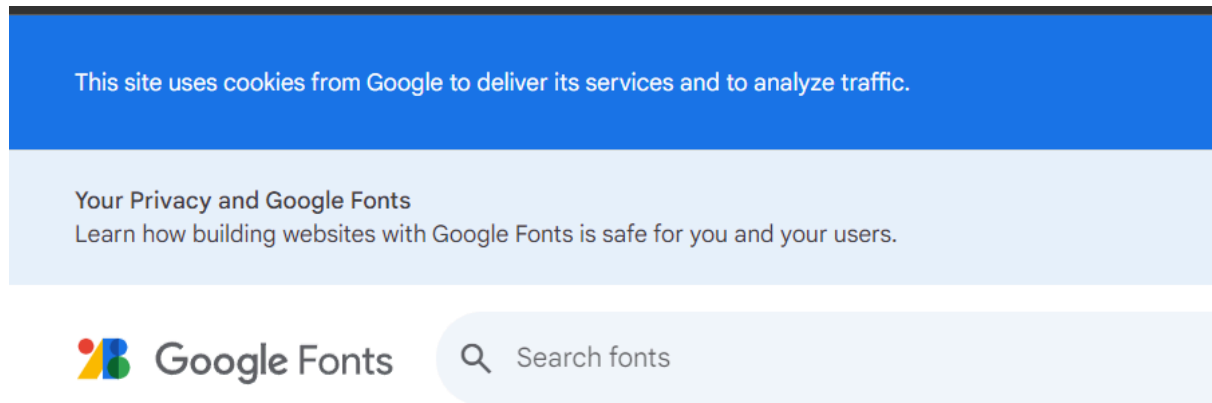
Hacemos esto con todas las plantillas

## 16. Estilo general

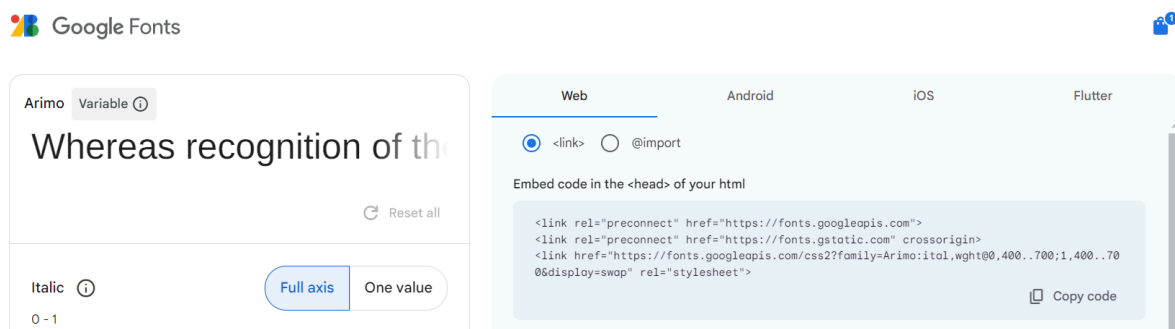
Fuente

Elegir una de google fonts

## End-to-end task manager con Django



Copiamos todo el link



```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Lista de tareas pendientes</title>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Arimo:ital,wght@0,400..700;1,400..700&display=swap" rel="stylesheet">
```

En style, continuamos dando estilo y fuente:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Lista de tareas pendientes</title>
  <!-- Preconexión a Google Fonts para mejorar la carga de las fuentes -->
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <!-- Enlace a las fuentes personalizadas desde Google Fonts -->
```

```
<link
href="https://fonts.googleapis.com/css2?family=Anek+Gujarati:wght@700&family=
Dosis:wght@200..800&family=Mukta:wght@200;300;400;500;600;700;800&displa
y=swap" rel="stylesheet">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Anek+Gujarati:wght@700&family=
Mukta:wght@200;300;400;500;600;700;800&display=swap" rel="stylesheet">
<style>
  body {
    background-color: #EFF1E2; /* Color de fondo del cuerpo de la página */
    font-family: "Dosis", sans-serif; /* Fuente personalizada para el cuerpo */
    font-optical-sizing: auto;
    font-weight: <weight>;
    font-style: normal;
    padding-top: 60px; /* Espacio superior en el cuerpo */
  }

  .container {
    max-width: 550px; /* Ancho máximo del contenedor */
    margin: auto; /* Centrar el contenedor horizontalmente */
    background-color: #F4FDC0; /* Color de fondo del contenedor */
    -webkit-box-shadow: 2px 2px 13px -4px rgba(0, 0, 0, 0.21); /* Sombra del
contenedor en WebKit */
    box-shadow: 2px 2px 13px -4px rgba(0, 0, 0, 0.21); /* Sombra del contenedor */
  }

  h1,
  h2,
  h3,
  h4,
  h5,
  h6 {
    font-family: "Mukta", sans-serif; /* Fuente personalizada para los
encabezados */
    font-weight: 400; /* Peso de la fuente para los encabezados */
    font-style: normal; /* Estilo de la fuente para los encabezados */
  }

  a,
  p {
    color: #7B755B; /* Color de texto para enlaces y párrafos */
  }
```

## End-to-end task manager con Django

```
</style>
</head>
<body>

<div class="container">
  {% block content %}
  {% endblock content %}
</div>
</body>
</html>
```



A partir de este paso, únicamente he dado estilo al CSS y la distribución de algunos elementos.

Muchas gracias,  
Irene