Contenido

[Crear proyecto usando CMD 3](#_Toc81674185)

[Cómo correr el servidor de Django 3](#_Toc81674186)

[Configurando nuestra base de datos SQLite 3](#_Toc81674187)

[Usuario root de PostgreSQL y puerto 4](#_Toc81674188)

[Configurar base de datos PostgreSQL con Django 4](#_Toc81674189)

[Creando aplicaciones en Python 5](#_Toc81674190)

[Creando modelos en Django 5](#_Toc81674191)

[Panel de administrador Django 5](#_Toc81674192)

[Crear un usuario para el panel de administrador 5](#_Toc81674193)

[Django y URLS 6](#_Toc81674194)

[Vistas basadas en funciones 6](#_Toc81674195)

[Incluyendo sub urls 8](#_Toc81674196)

[Templates en Django 9](#_Toc81674197)

[Haciendo If y For en templates 11](#_Toc81674198)

[Herencia de templates en Django 11](#_Toc81674199)

[Rutas con nombre 14](#_Toc81674200)

[Configuración de archivos estáticos 15](#_Toc81674201)

[Usando archivos estáticos en templates 15](#_Toc81674202)

[Incluir fragmentos HTML dentro de otros con Include 16](#_Toc81674203)

[Reemplazo de parámetros en templates heredadas 16](#_Toc81674204)

[Conclusión herencia de templates en Django 18](#_Toc81674205)

[SELECT (listar registros) en Django 18](#_Toc81674206)

[Vistas basadas en clases 20](#_Toc81674207)

[Sobreescritura del método dispatch 21](#_Toc81674208)

[Create view Primer Formulario 21](#_Toc81674209)

[Personalizar inputs o elementos dentro del formulario 25](#_Toc81674210)

[Agregar validaciones a el formulario 28](#_Toc81674211)

[Variables de Django dentro de Javascript 32](#_Toc81674212)

[Creando template base para renderizar cualquier formulario 32](#_Toc81674213)

[Atajos Pycharm 33](#_Toc81674214)

[Solucionar errores 33](#_Toc81674215)

[django.core.exceptions.ImproperlyConfigured: Cannot import 'erp'. Check that 'core.erp.apps.ErpConfig.name' is correct. 33](#_Toc81674216)

# Crear proyecto usando CMD

Crear una carpeta para el proyecto e ingresar en ella a través de CMD.

Se crea un entorno virtual.

Python -m venv env

Se crea el entorno llamado env en la carpeta env.

Entrar en env/Scripts.

Ejecutar el comando actívate para activar el entorno virtual:

actívate

Dentro de la carpeta Scripts instalar Django usando pip:

pip install Django

Devolverse a la carpeta raíz del proyecto, al mismo nivel de env y ejecutamos:

django-admin startproject app

En este caso, app es el nombre de la aplicación web.

## Cómo correr el servidor de Django

Entrar en la carpeta app y ejecutar este comando en CMD:

python manage.py runserver

# Configurando nuestra base de datos SQLite

Los modelos en Python son la representación de las tablas de la base de datos.

Posteriormente se crean las migraciones.

Django ya contiene un grupo de migraciones pre establecidas para crear ciertas tablas en la bd, como Admin, auth, contenttypes, etc.

Desde pycharm, ir a tools > run manage.py task.

## Usuario root de PostgreSQL y puerto

Contraseña usuario root: root

Puerto en el que corre: 5432

Contraseña para pgAdmin: root

# Configurar base de datos PostgreSQL con Django

Ir a File > settings > Project: app > clkick en simbolo de + e instalar el paquete psycopg2 el cual es el dcriver de conexión entre Python y postques.

Para configurar una bd de postgresql con django, primero creamos dentro de la carpeta app un arhcivo llamado db.py. Este archivo contiene lo siguiente:

POSTGRESQL = {

'default' : {

'ENGINE' : 'django.db.backends.postgresql\_psycopg2',

'NAME' : 'db',

'USER' : 'postgres',

'PASSWORD' : 'root',

'HOST' : 'localhost',

'PORT' : '5432'

}

}

En Python, la sintaxis variable = { } significa que la variable es un diccionario.

Editamos el archivo settings.py en la línea DATABASES = y lo hamos igual al diccionario de db.POSTGRESQL.

Ya que están conectada la base de datos de portgresql, hacemos la migraciones nuevamente para que esta vez, se eguaden dentro de la base de datos de postgresql

(desde la carpeta app)

python manage.py migrate

# Creando aplicaciones en Python

Una aplicación en Python es una parte separada del proyecto. Se puede crear una apliacion creando una nueva carpeta (en este caso llamada core) y dentro de ella ejecutando el comando:

Python ../manage.py startapp erp

Este comando creará una app llamada erp dentro de la carpeta core.

# Creando modelos en Django

Para hacer los modelos y hacerlo de una manera mas ordenada, se puede crear una app solo para esta función.

Adentro de cada app de Django hay varios archivos .py, uno de ellos es models. Adentro de models se definen las clases que serán tratadas como modelos.

Posteriormente se debe ir al archivo settings.py y agregar esa nueva app creada.

# Panel de administrador Django

Perite interactuar con la base de datos de manera automática.

## Crear un usuario para el panel de administrador

Desde la línea de comandos, ejecutar:

\Horus\app>Python manage.py createsuperuser

Para este proyecto el super usuario es:

Nombre: root

Password: brunito1119981

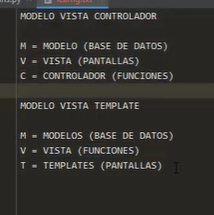
Una vez hecho esto, y ejecutando el servidor, podemos ir a la URL

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

Y acceder al panel

# Django y URLS

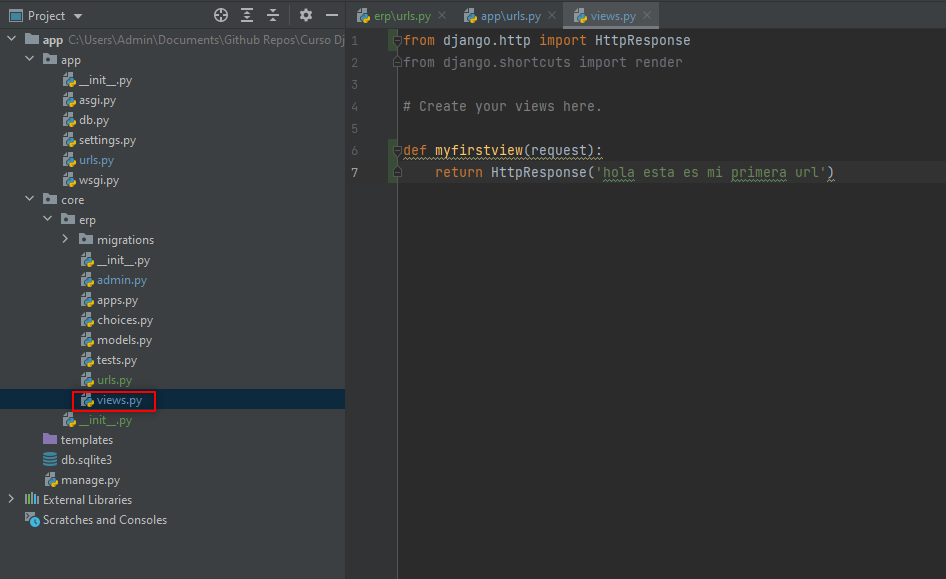
Esta es la manera como se manejan las URLs en Django:



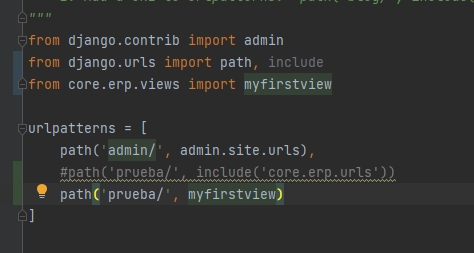
De esta manera, las vistas son los fragmentos del código Python que se ejecutan del lado del servidor para atender una petición hecha por el usuario desde un template.

## Vistas basadas en funciones

Dentro de nuestra app llamada erp (core.erp) podemos entrar al archivo de views y crear una función que reciba una petición



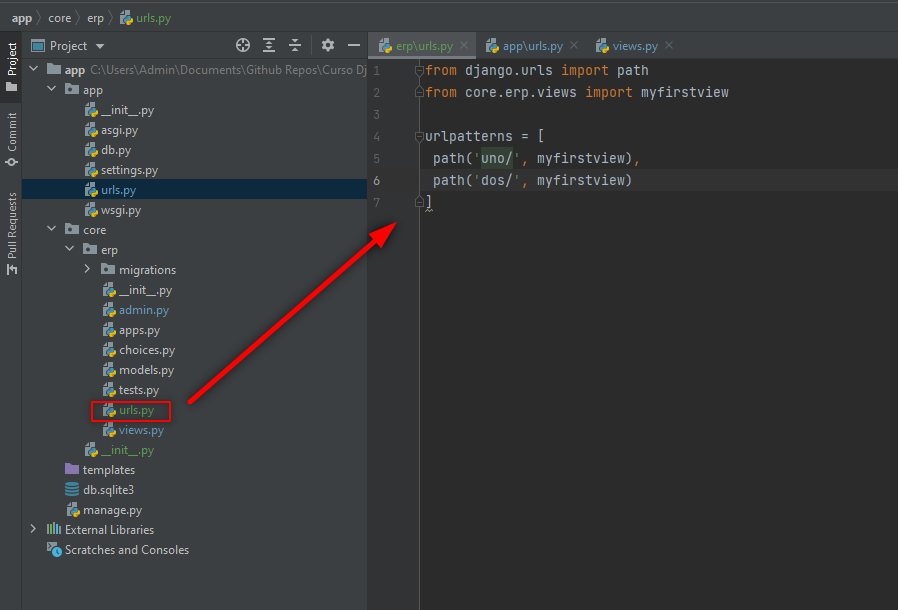
Luego, desde la carpeta principal app registramos esta view abriendo el archivo urls.py



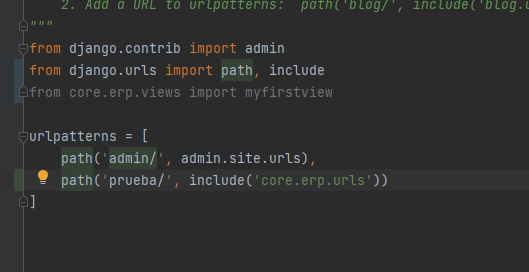
Se agrega una nueva ruta y de esta menra, al entrar en localhost:8000/prueba se ejecuta la función anterior y eso es lo que se ve en pantalla.

## Incluyendo sub urls

Tambien se puede crear un archivo urls.py dentro de la app core.erp e incluir sub urls



Luego, estas dos urls se incluyen en el archivo urls.py dentro de la carpeta app



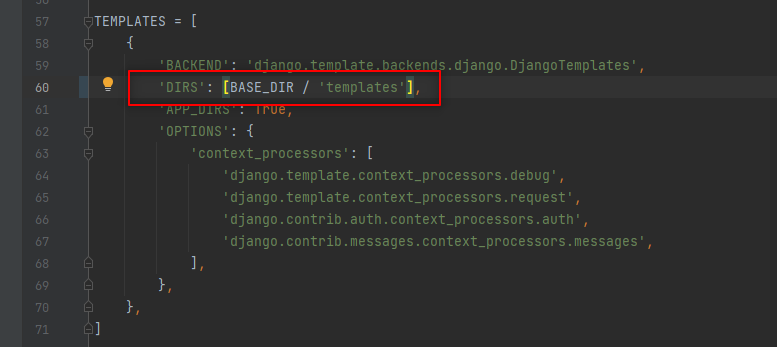
Por lo que ahora hay 2 vistas:

<http://127.0.0.1:8000/prueba/uno/>

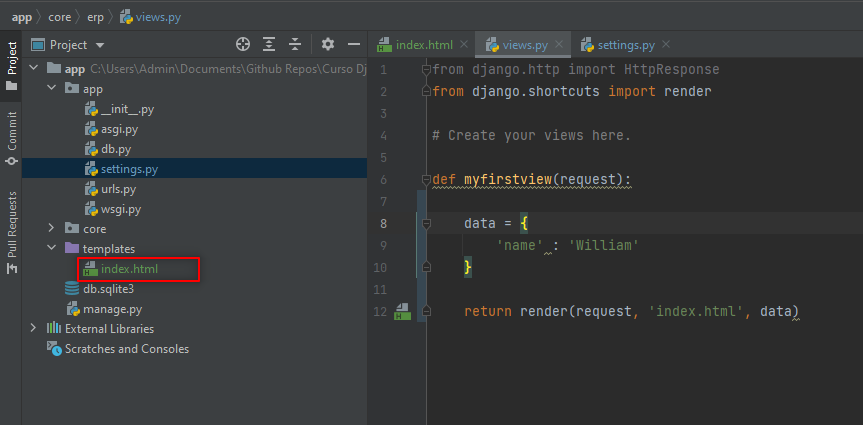
<http://127.0.0.1:8000/prueba/dos/>

# Templates en Django

Verificar que exista una carpeta template en la raíz del proyecto (al mismo nivel que app) y que dentro de settings.py exista esta línea para agregar os archivos de la carpeta template.

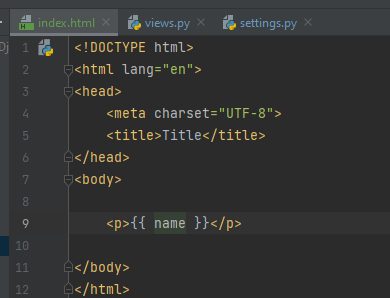


Una vez hecho esto, creamos un archivo html en la carpeta template, y en el archivo de views de nuestra aplicación erp de Django, modificamos la función para que ejecute la función render, y le decimos que renderice el archivo HTML que creamos.



Dentro de render() también podemos enviar un diccionario de llaves y valores, para luego mostrarlos en el template, en este caso el diccionario se llama data.

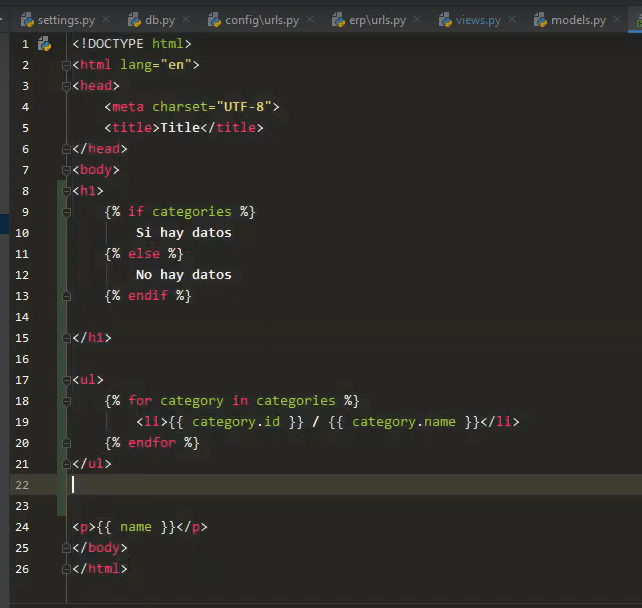
Dentro de nuestro template html podemos llamar los valores dentro del diccionario simplemente usando la notación de dobles llaves así: {{ name }}



Y al acceder a esta url, la cuel es prueba/dos vemos:

## Haciendo If y For en templates

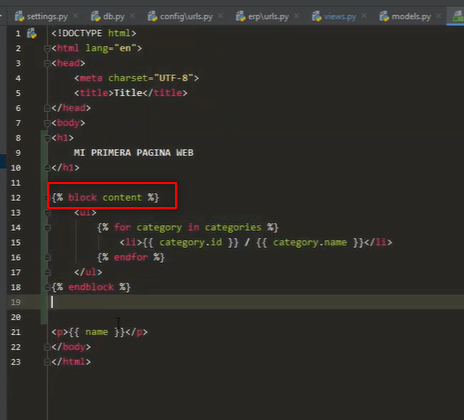
Si dentro de nuestro diccionario data nosotros pasaramos un arreglo de elementos, podríamos hacer un ciclo for de esta manera



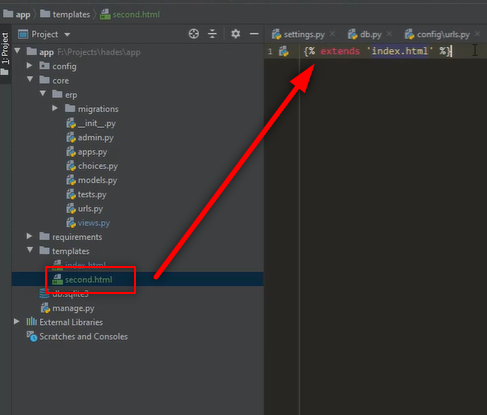
## Herencia de templates en Django

Dentro de un template “padre” se puedendefinir diferentes bloques de código HTML. Estos bloques de código se pueden sobreescribir en un template “hijo”que herede de el primer template.

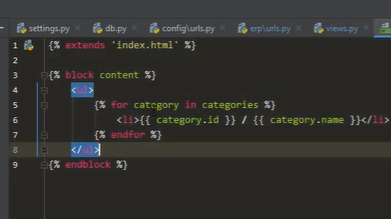
Como ejemplo, en el template padre llamado index, definimos el siguiente bloque de código llamado “content” el cual contiene un ciclo que muestra las categorías obtenidas desde la base de datos



Posteriormente se crea otro template el cual hereda de index usando esta línea:

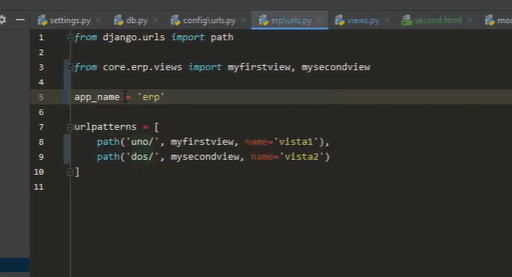


Por lo cual, la vista second.html mostrará el mismo contenido de index.html, pero se puede sobreescribir lo que va a mostrar dentro del bloque content



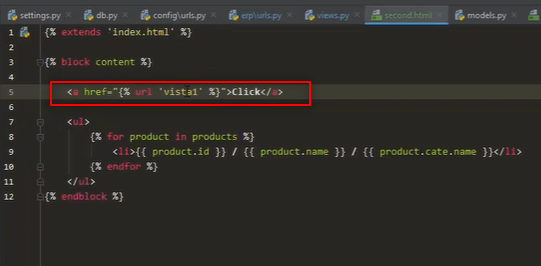
## Rutas con nombre

Las rutas también pueden tener un nombre para usarlas de manera mas fácil dentro del framework



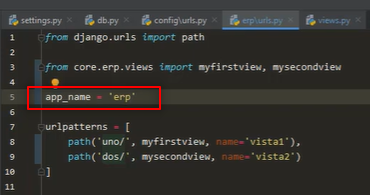
En este caso hay dos rutas con nombres vista1 y vista2

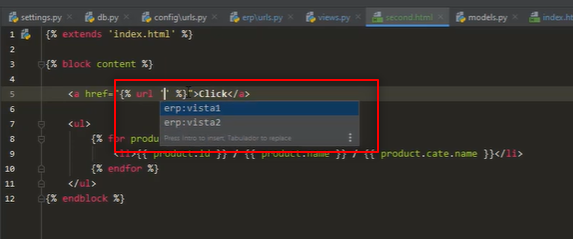
Dentro del html de un template se puede escribir lo siguiente:



Al usar url ‘vista1’ automáticamente se completa por el url que conduce a esa ruta.

También dentro de el archivo urls.py se puede especificar un app\_name, el cual sirve como un namespace para identificar las rutas de manera mas especifica. Por ejemplo si hay 2 rutas llamadas vista1, pero de namespaces diferentes, entonces no habrá confusión





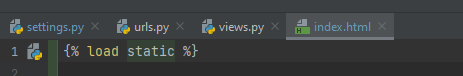
## Configuración de archivos estáticos

Los archivos estáticos son imágenes, multimedia, css, js, y cualquier otra clase de archivos que necesitemos agregar a nuestra solución.

Para configurar el lugar en donde se van a encontrar todos estos archivos, debemos ir a el archivo settings.py y agregar lo siguiente

## Usando archivos estáticos en templates

Para cargar archivos estáticos, en las primeras líneas de cada template se debe incuir lo siguiente

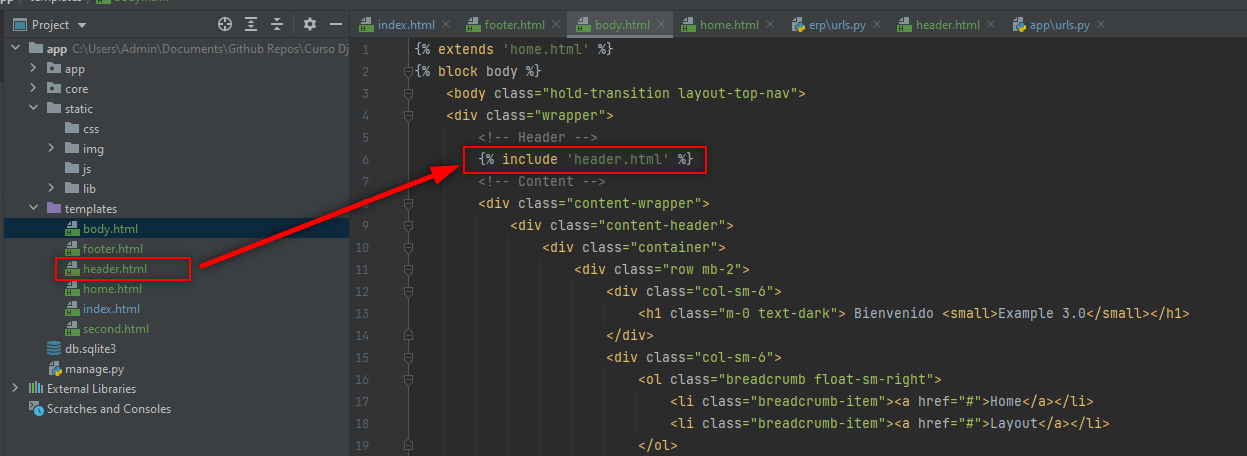


Puesto que la carpeta contenedora de todos los archivos tiene nombre static, y es la que se configuro dentro del archivo settings.py

## Incluir fragmentos HTML dentro de otros con Include

De manera similar se puede incluir todo el contenido HTML de un template dentro de otro, en el lugar en donde se desee, usando la función include de Django.

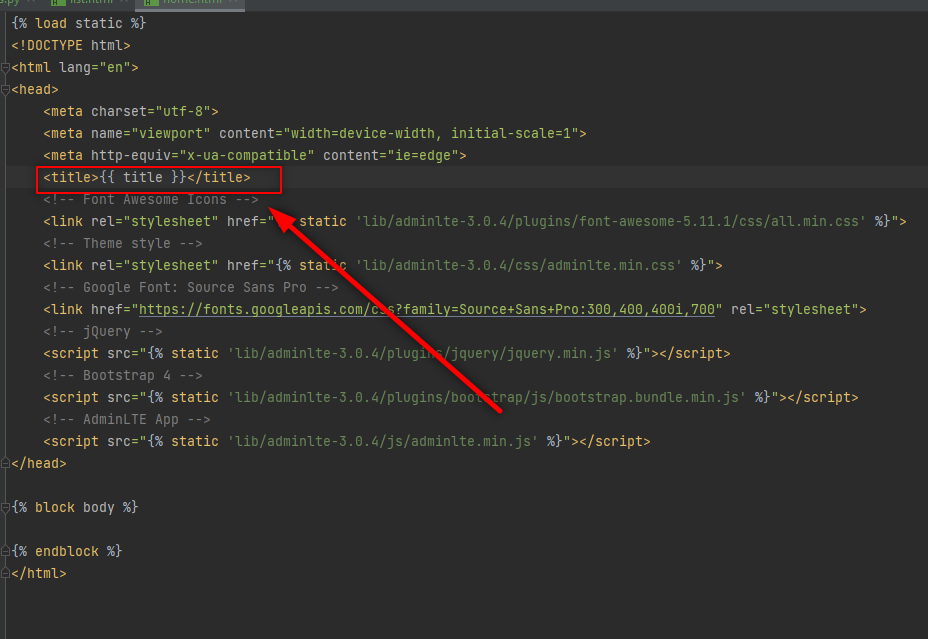
En este ejemplo, el template body.html incluye dentro de su html todo el contenido del template header.html, por lo que el header aparece en esa sección del html.



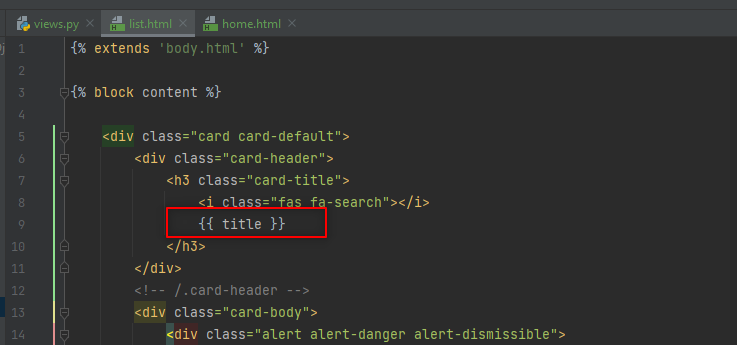
## Reemplazo de parámetros en templates heredadas

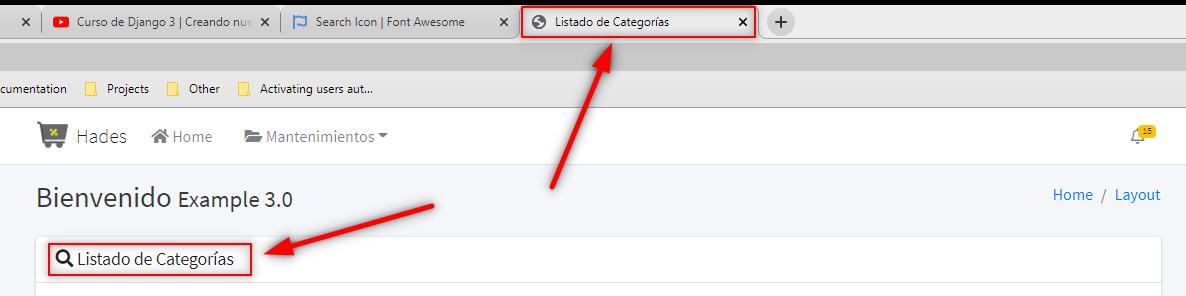
Si un template hereda de otro, y el primero (el template padre) tiene declarada una variable dentro de su html, esa variable obtiene el valor que el template hijo le ponga.

En este ejemplo, Home.html declara o usa la variable ‘title’ que se le pasa desde la view.



El archivo list.html hereda de home.html, y la view le pasa un valor de title (category list), por lo que ese valor se reeemplaza tanto en el template padre como en el hijo, ambas muestran el mismo valor.

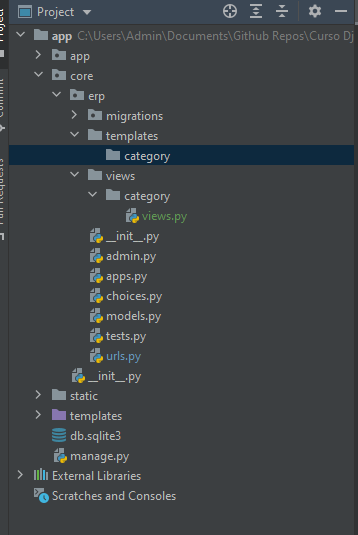




## Conclusión herencia de templates en Django

* La instrucción {% block %} funciona cuando se hace un {%extends%} en otra plantilla, para que plantilla hija pueda sobreescribir contenido dentro del {% block %} que el padre tiene declarado.
* La instrucción {%include%} sirve para insertar todo el contenido HTML de un template dentro de otro, sin poder sobreescribir nada.

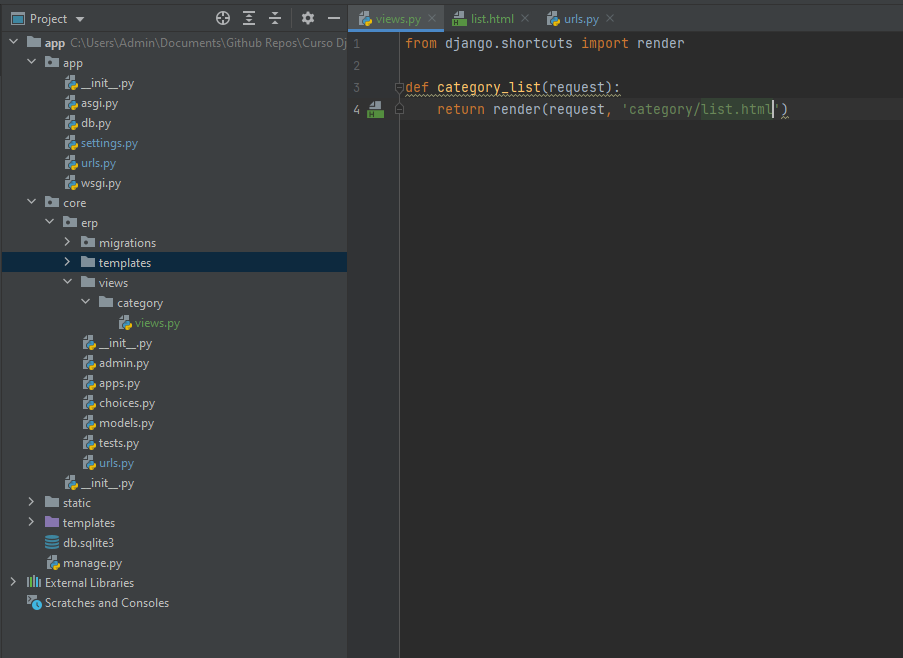
# SELECT (listar registros) en Django



Para organizar mejor el proyecto, es posible crear una carpeta de templates y views independiente dentro de la app erp, **para que haya una carpeta para almacenar vistas y otra para templates, separadas por cada entidad (Category en este caso)**

La carpeta Templates que se encuentra en la raíz del proyecto solo se usará para las templates básicas, pero los templates y vistas específicas de cada entidad tendrán carpetas independientes.

Dentro de el archivo views.py para la entidad category se crea una función que retorne un template html

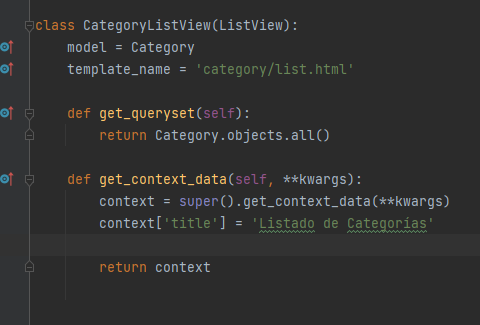


‘category/list.html’ es la ruta a el archivo html, aunque la carpeta category está dos niveles mas arribaía ser ../../templates/category/list.html) solo basta con poner category/list.html ya que Django va a buscar en todas las carpetas aquella llamada category que contenga adentro archivos html, entre ellos list.html.

## Vistas basadas en clases

Para crear una vista creada en clase, dentro del archivo views.py se declara una clase que herede de ListView a la que se le especifica el modelo, el nombre del template que se va a usar, y se pueden sobreescribir algunos métodos de la clase ListView

Get\_queryset es el método que define cual será la consulta a ejecutar. Get\_context\_data es el método que permite agregar mas parámetros a la respuesta que se enviará al template, como el atrinbuto title.



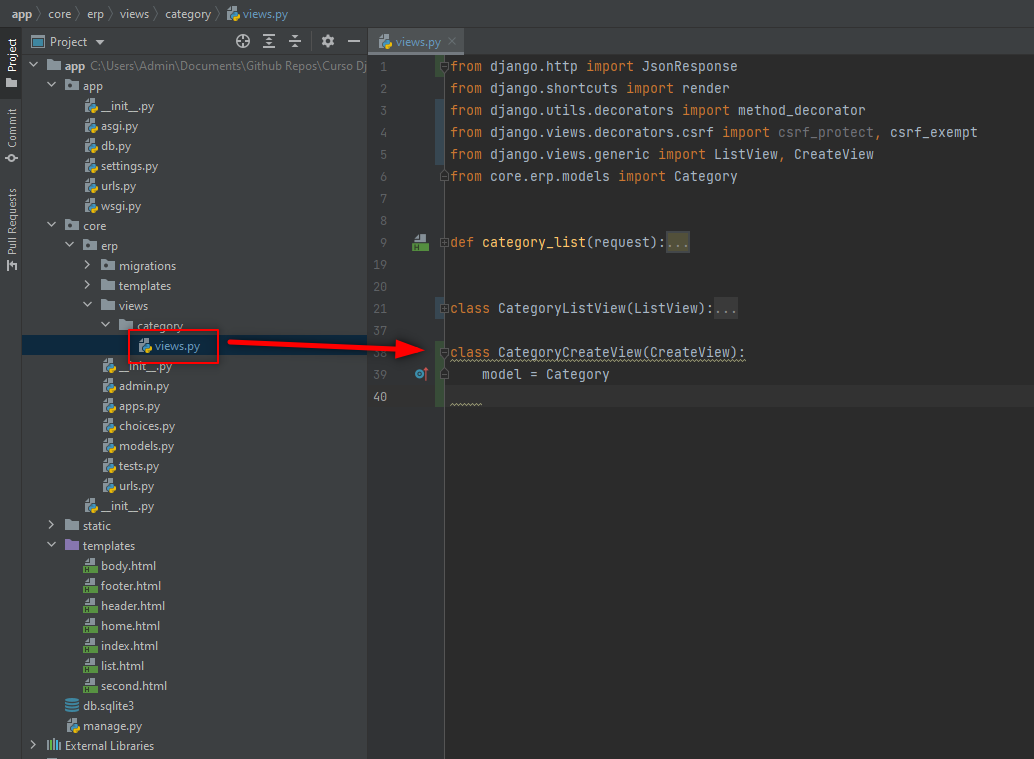
Texto

# Sobreescritura del método dispatch

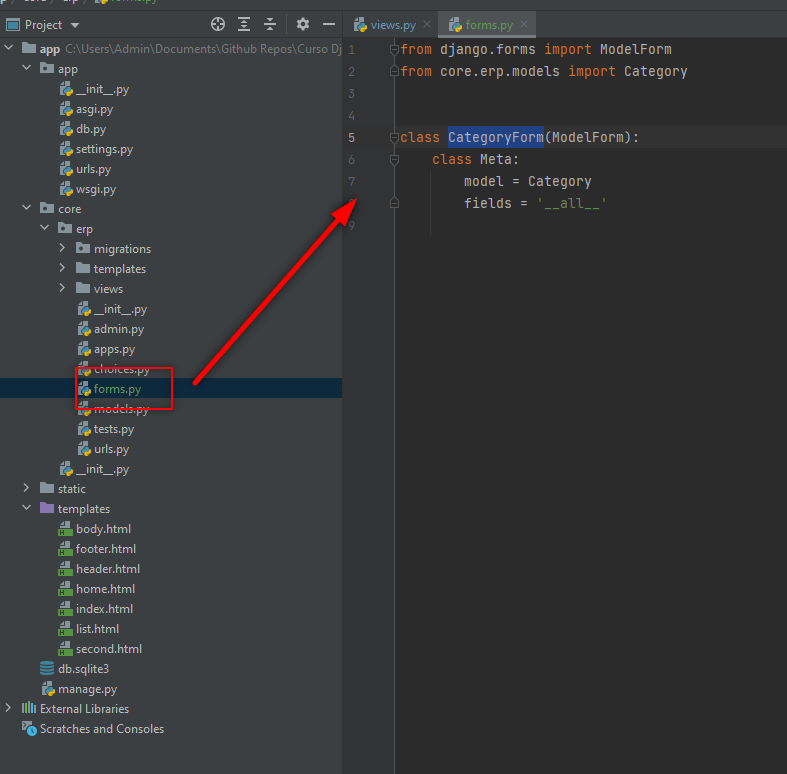
El método dispatch redicerriona a el método POST o GET dependiendo de la petición que recibe

# Create view Primer Formulario

Para añadir la funcionalidad de insertar registros sobre una tabla de la base de datos, añadimos una nueva vista que herede de la clase CreateView. Utilizando la clase ModelForm vamos a crear el formulario.



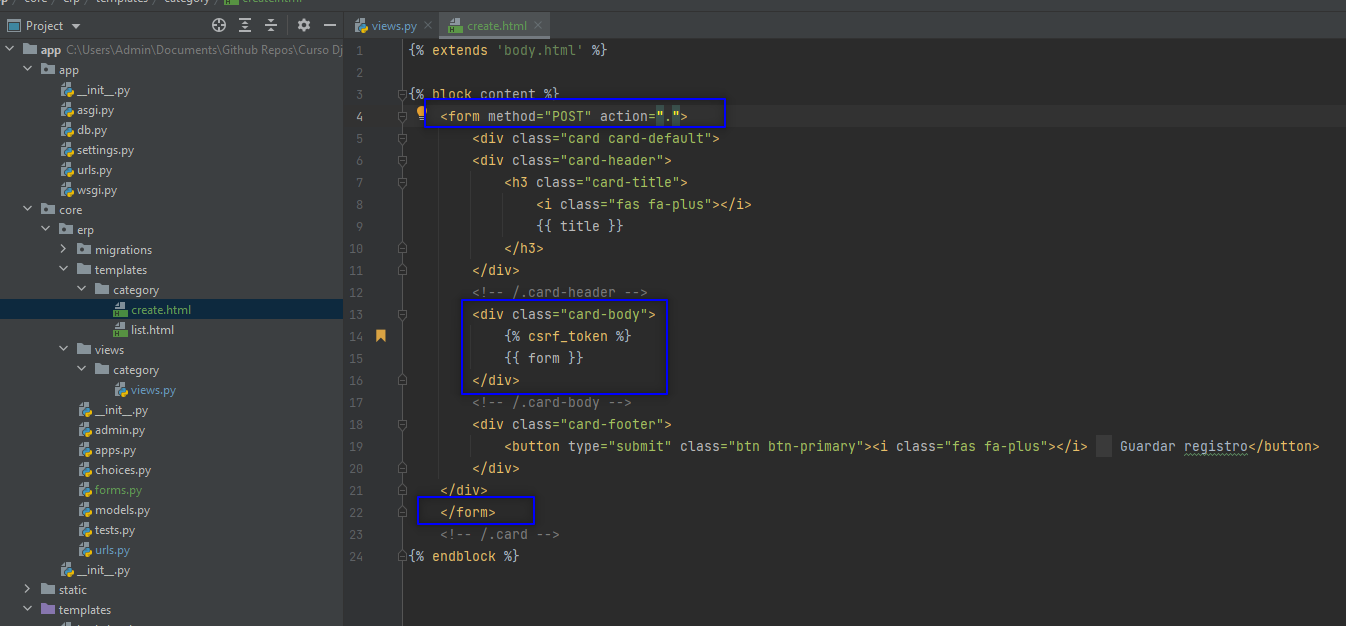
Para crear un formulario, creamos un archivo de Python que va a contener todos los formularios de la app, en este caso se llamará forms.py dentro de la app erp.

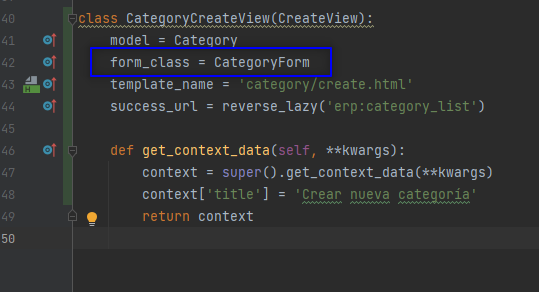


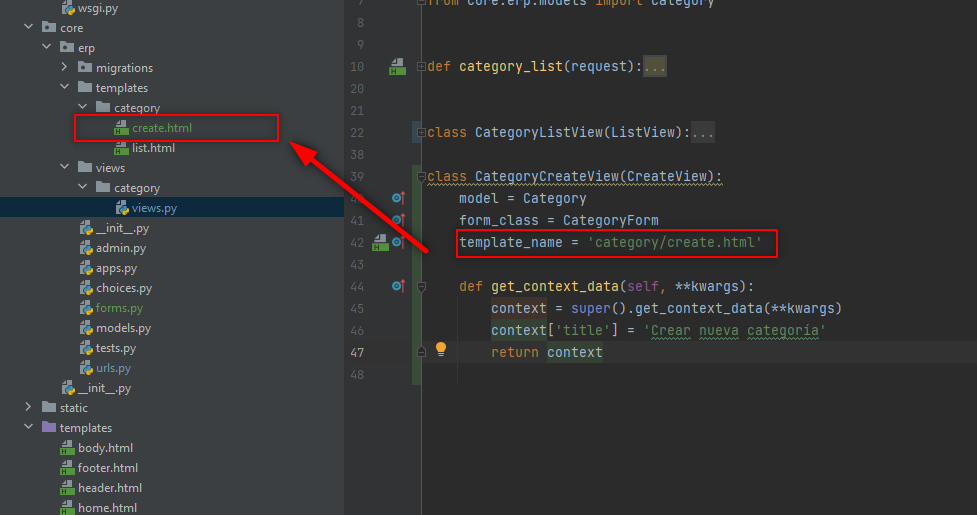
Y posteriormente, esta clase-formulario se referencia dentro de la clase CreateView de esta entidad (en este caso la entidad es categoría)

Las propiedades del formulario se definen dentro de la inner-class Meta.

Esta clase-formulario se referencia luego desde la view

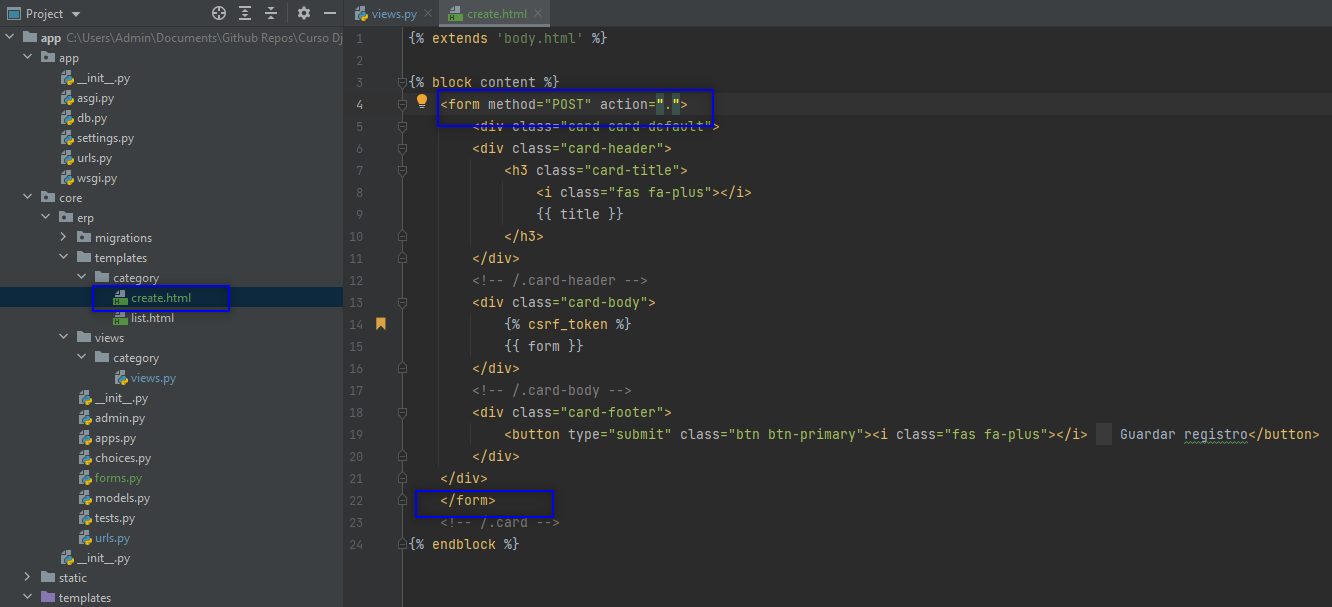




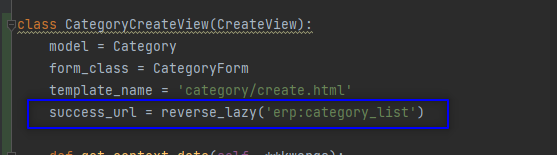


El template\_name apunta a el archivo html que va a mostrarse.

En la vista para crear categoria (la clase que implementa CreateView dentro del archivo de vistas de category), existe n varios elementos pero el mas importante es el {{ form }}, esta variable va a renderizar todo el formulario

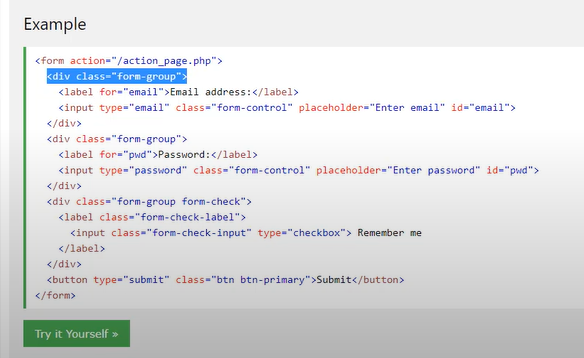


Dentro DE LA clase formulario con este atriobuto se define el nombre de la ruta a la que se va a redireccionar una vez se tenga éxito en la insecion de la categoría

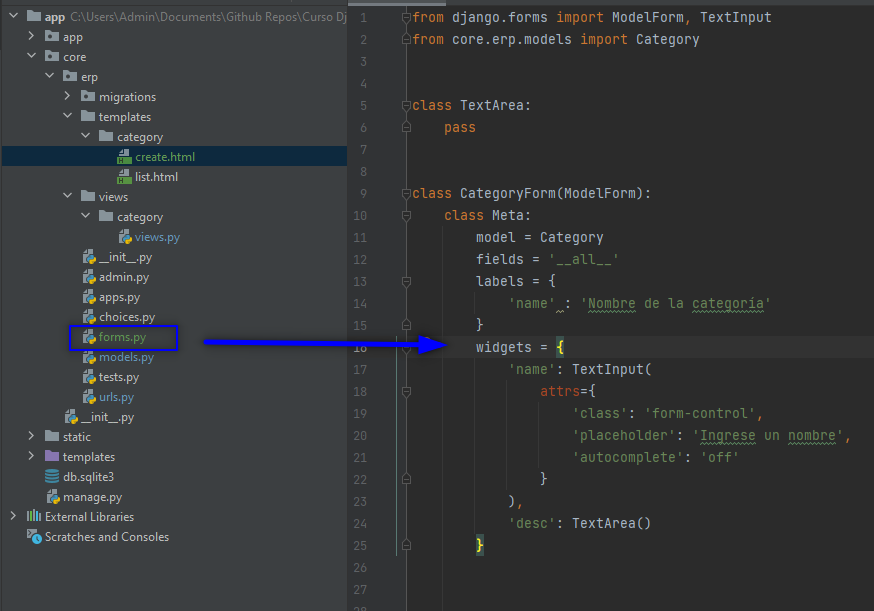


## Personalizar inputs o elementos dentro del formulario

Hasta ahora los inpus se renderizan sólo como inputs por defecto de HTML.



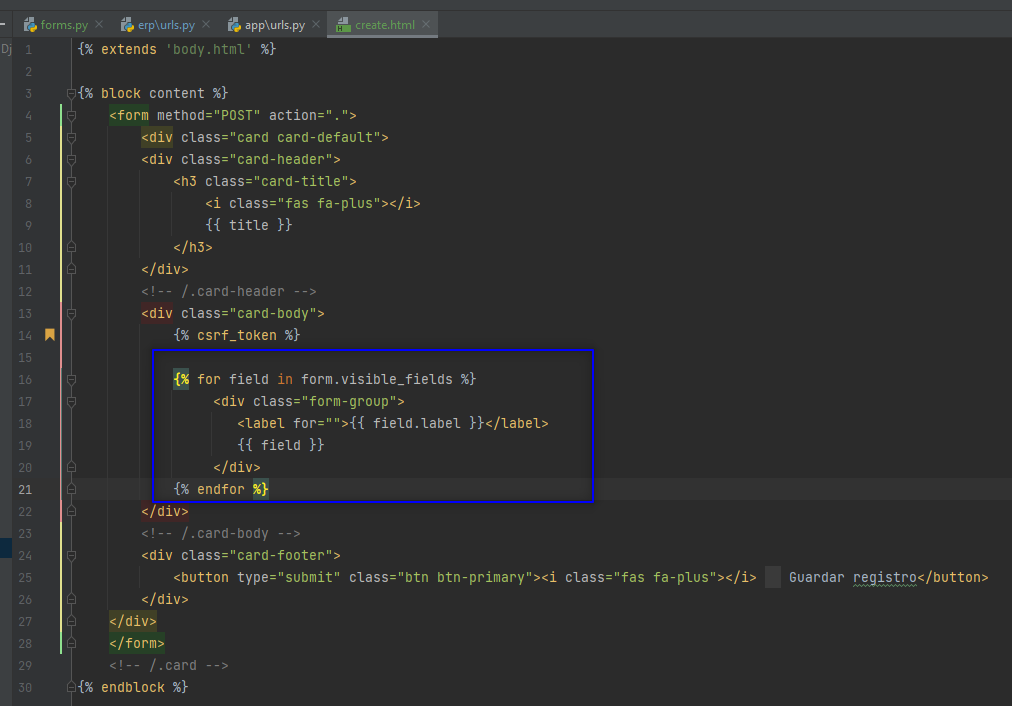
Para poder aplicar clases, ids, placeholders entre otros atributos seguimos los siguientes pasos:

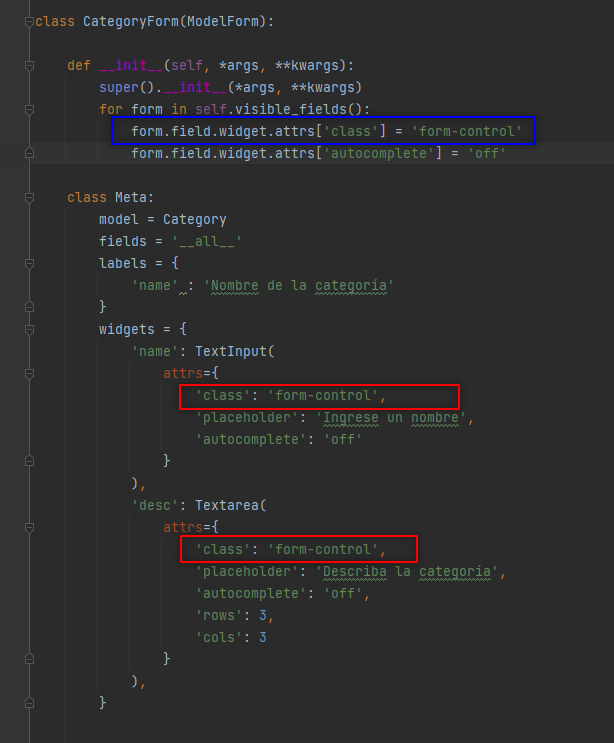


Dentro de la clase que define el formulario se definen dos diccionarios, labels para cambiar el label de los fields cuando se renderizen, y widgets.

En este diccionario definimos que tipo de input se renderizara en cada campo, y adentro en el atributo attrs se pone la clase y otras cosas como el id que va a llevar el input.

Ahora vamos a iterar sobre todos los elementos del form para ponerlos dentro de un form group:

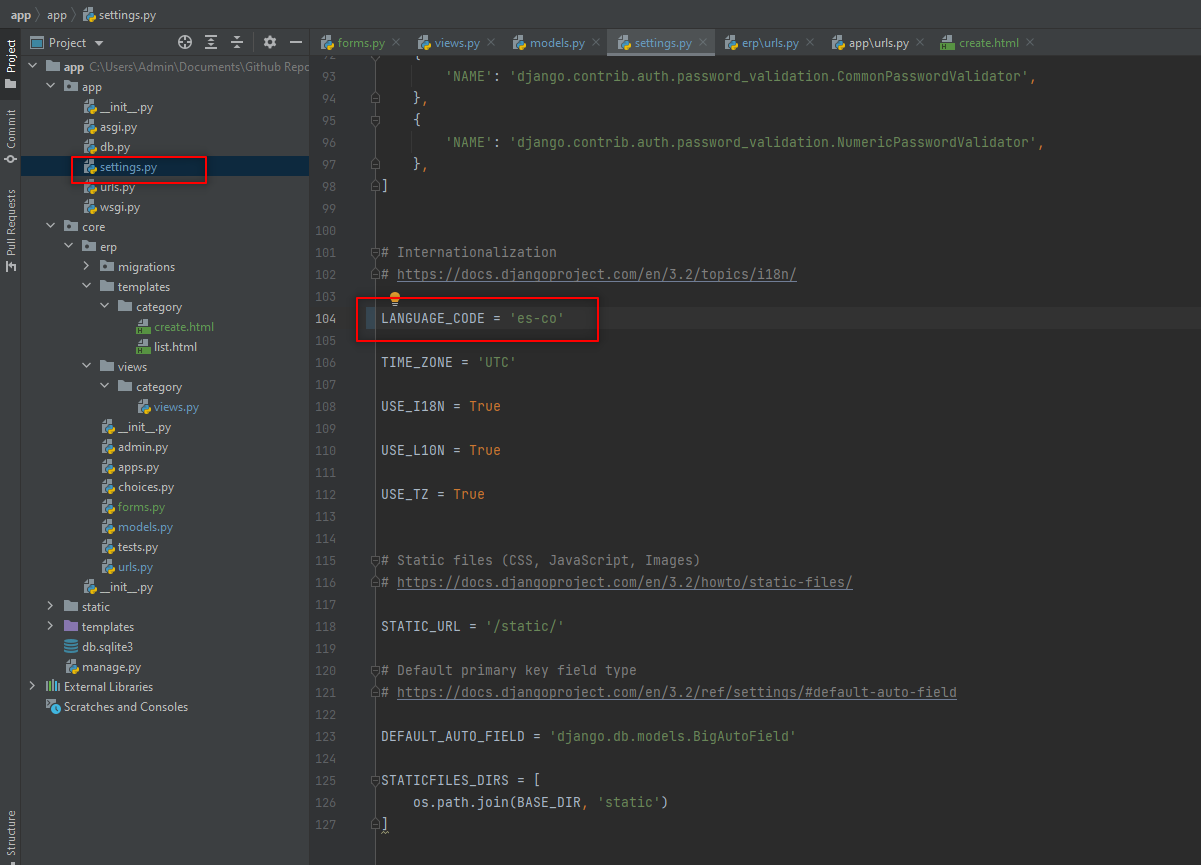


Para no repetir varias veces el mismo atributo dentro del parámetro attrs se puede agregar el mismo atributo a todos los fields de esta manera sobreescribiendo el método init.

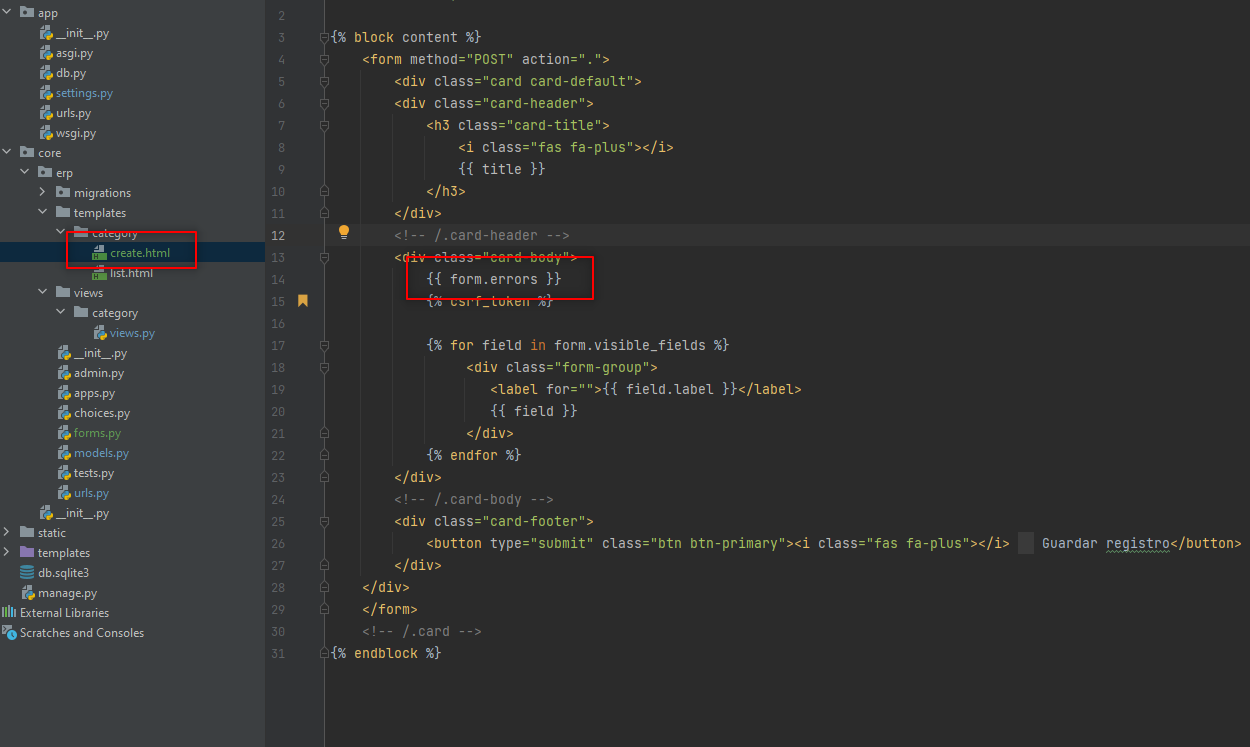
## Agregar validaciones a el formulario

Las validaciones se definen cuando se crea el modelo (cuales campos son únicos, etc). Para mostar los errores de validación, primero cambiar el lenguaje de Django a español

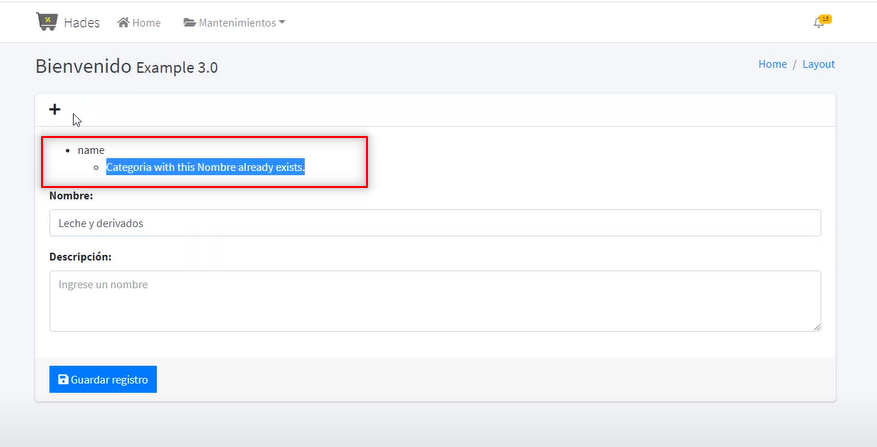
En App (la carpeta de app principal) ir a settings y poner el código de lenguaje correspondiente:



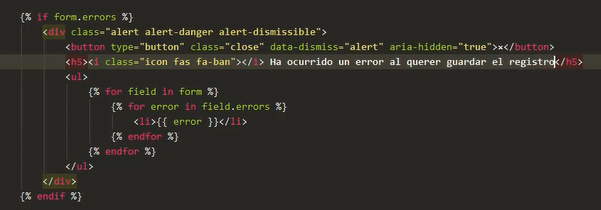
Para que se muestre de forma básica las validaciones, solo hay que agregar esta línea de código en el formulario:



De esta forma aparece este html con los mensajes de error:

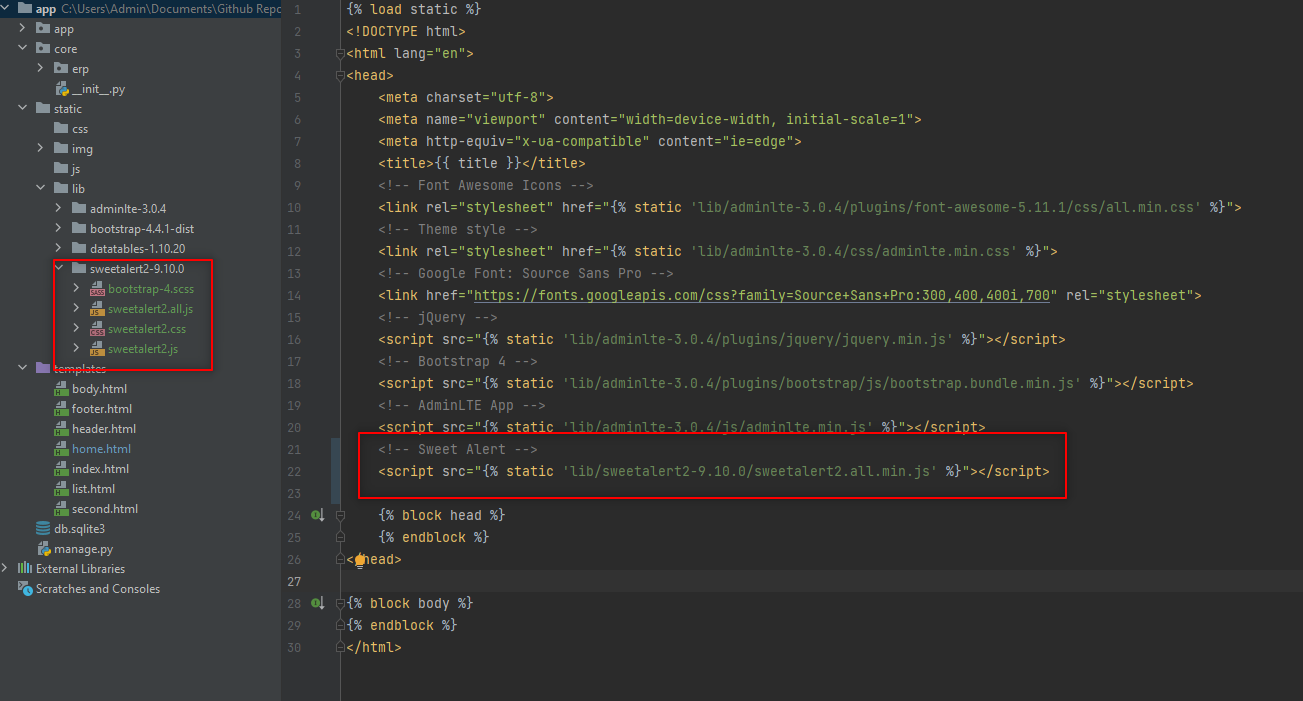


De manera similar, es posible iterar sobre los errores usando el sistema de templates de Django para mostrar los errores separadamente:

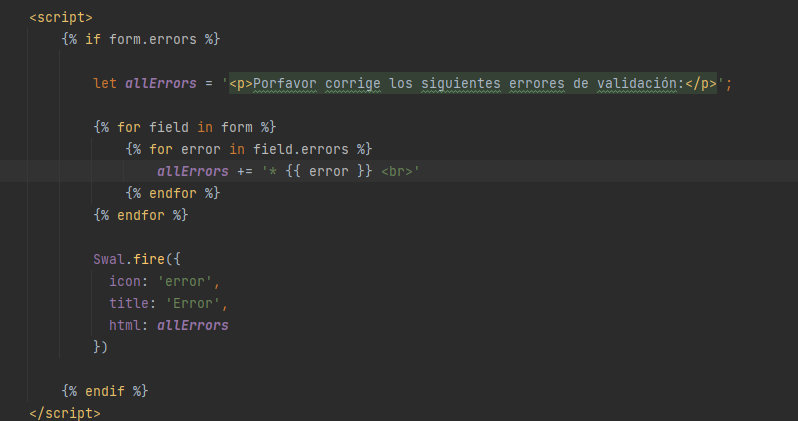


Para esta implementación en especial se usará un plugin que muestra alertas de forma mas estética: SweetAlerts.

<https://sweetalert2.github.io/#download>

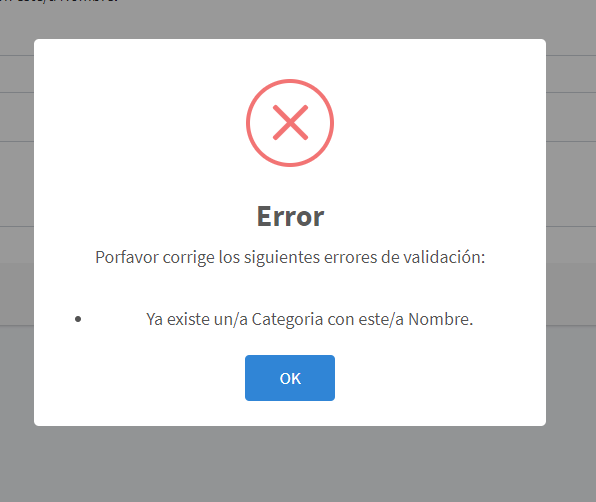


Dentro de el template créate de category:



### Variables de Django dentro de Javascript

Se agrega este script en el cual se puede ver que las variables como form se pueden usar dentro de javascript de esta manera.



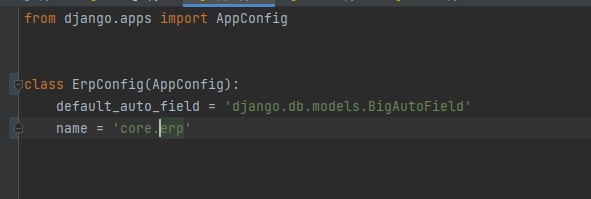
# Atajos Pycharm

Ctrl + Alt + L: Auto-identar el código

# Solucionar errores

## django.core.exceptions.ImproperlyConfigured: Cannot import 'erp'. Check that 'core.erp.apps.ErpConfig.name' is correct.

Para solucionar este error, ir a la app y abrir el archivo apps.py. El nombre de la app (name) debe tener el nombre de la carpeta contenedora primero. En este caso es core.erp, y no solo erp



Ya que el directorio del proyecto es:

