----------1ER PASO ----------

CREAR UN ENTORNO DE TRABAJO CON EL NOMBRE DEL ESTABLECIMIENTO

py -3 -m venv .venvpapeleriamovi

----------2DO PASO----------

.venv\scripts\activate.bat

ACTIVAR EL ENTORNO

----------3ER PASO----------

SELECCIONAR INTERPRETE DE PREFERENCIA EL PYTHON.EXE (NO EL RECOMENDADO) CON LA TECLA F1

----------4TO PASO----------

python -m pip install --upgrade pip

INSTALAR EL PIP

----------5TO PASO----------

python -m pip install django

INSTALAR DJANGO

----------6TO PASO----------

django-admin startproject papeleriamovi .

CREAR EL PROYECTO PRINCIPAL

----------7MO PASO----------

CREAR APP

Django-admin startapp papeleria_app

----------8VO PASO----------

EN ESTE PASO VAMOS A CAMBIAR LAS LINEAS DEL PROYECTO

ARCHIVO SETTINGS.PY LINEAS:

39: (crear app) = ‘papeleria_app’,

```
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'papeleria_app',
41  ]
```

79:(cambiar el nombre a base de datos) = ‘papeleria.db’

```
77  DATABASES = {
78      'default': {
79          'ENGINE': 'django.db.backends.sqlite3',
80          'NAME': 'papeleria.db',
81      }
82  }
83
```

107: (cambiar a español): ‘es-mx’

```
106
107 LANGUAGE_CODE = 'es-mx'
108
```

----------9NO PASO----------

CREAR EL ARCHIVO URLS.PY EN LA APP

----------10MO PASO----------

LLAMAR EL URLS.PY DE LA APP PARA PODER UTILIZARLO

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('papeleria_app.urls')),
]

```

----------11V0 PASO----------

CREAR UNA FUNCION PARA LLAMAR EL ARCHIVO INDEX

```python
from django.shortcuts import render

# Create your views here.

def index_vista(request):
    return render(request,'index.html')

```

----------12VO PASO----------

AHORA UTILIZAMOS ESA FUNCION EN EL ARCHIVO URLS EN LA APP

RECUERDA IMPORTAR EL ARCHIVO VIEWS PARA UTILIZAR LA FUNCION

papeleria_app > urls.py > ...

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

urlpatterns = [
    path('',views.index_vista, name='index.html'),
]
```

----------13VO PASO----------

CREAR LAS CARPETAS “TEMPLATES” Y “STATIC” EN LA CARPETA DE LA APP

----------14VO PASO----------

CREAR EL ARCHIVO INDEX.HTML EN LA CARPETA TEMPLATES

----------15VO PASO----------

HASTA AQUÍ DEBERIA DE FUNCIONARTE TU INDEX.HTML]

----------16VO PASO----------

AHORA APARTIR DE AQUÍ ES LO MAS CONFUSO PERO NO TE PREOCUPES YO TE EXPLICARE TODO PASO A PASO :3

BIEN AHORA LO QUE VAMOS HACER ES CREAR LOS MODELOS

QUE ES UN MODELO? PUES PARA QUE ENTIENDAS ES UNA TABLA TUYA DONDE IRAN TUS ATRIBUTOS

ESTOS SON LOS TIPOS DE DATOS QUE PUEDEN LLEVAR TUS ATRIBUTOS:3

nombre = models.CharField(max_length=100) # VARCHAR

descripcion = models.TextField() # TEXT

precio = models.DecimalField(max_digits=10, decimal_places=2) # DECIMAL
cantidad = models.IntegerField() # INT

fecha_nacimiento = models.DateField() #DATE ES DECIR DIA MES Y AñO

hora_evento = models.TimeField() #TIME SOLO PARA REGISTRAR HORA

codigo = models.CharField(primary_key=True, max_length=6) #ID

ESTA SERIA MI PRIMERA TABLA

```
papeleria_app > models.py > Productos
1   from django.db import models
2
3   # Create your models here.
4
5
6   class Productos(models.Model):
7       id_producto=models.CharField(primary_key=True,max_length=1000)
8       precio=models.DecimalField(max_digits=10000,decimal_places=2)
9       marca=models.CharField(max_length=20)
10      nombre=models.CharField(max_length=40)
11      categoria=models.CharField(max_length=20)
12      diseño=models.CharField(max_length=40)
13      calidad=models.CharField(max_length=50)
```

AHORA DESPUES DE ESTO LE DAMOS PERMISO DE QUE PUEDAS ADMINISTRARLA CON DJANGO ES DECIR EL /ADMIN :3

ESTO SE HACE EN EL ARCHIVO ADMIN.PY DE LA APP

```
papeleria_app > admin.py
1   from django.contrib import admin
2   from .models import Productos
3
4   admin.site.register(Productos)
5
6   # Register your models here.
7
```

RECUERDA SIEMPRE IMPORTAR TU MODELO PARA PODER DARLE PERMISO DE ADMIN

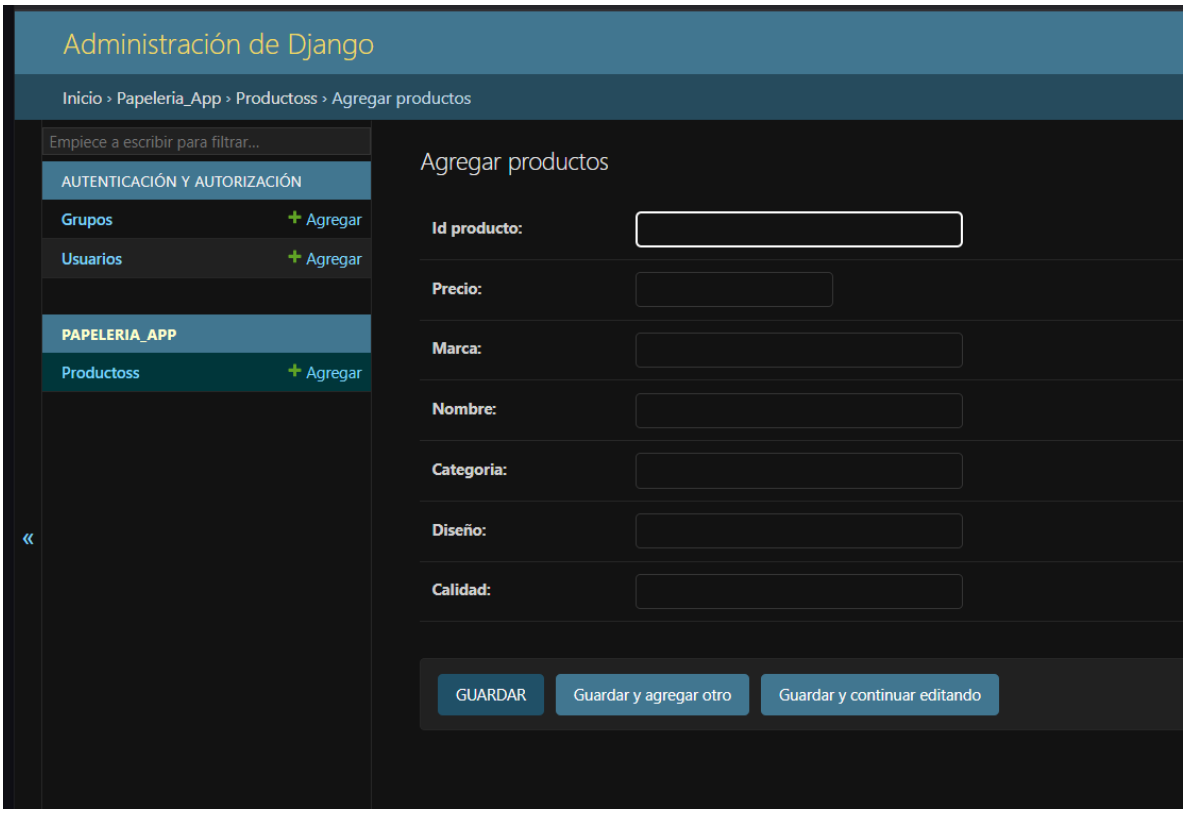
----------17VO PASO----------

CREAMOS UN ADMINISTRADOR

PARA ESO USAMOS LA LINEA

Python manage.py createsuperuser

AGREGAMOS LOS DATOS Q NOS PIDEN Y LISTO

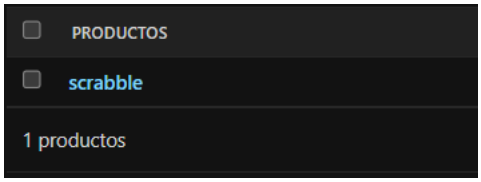


AQUÍ VEMOS COMO SI FUNCIONA NUESTRA TABLITA (❁´◡`❁)

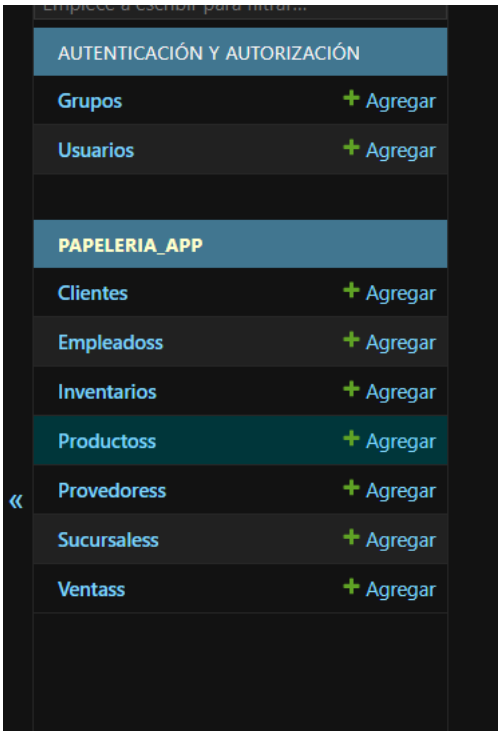
DESPUES PARA MOSTRAR EL NOMBRE DE LOS OBJETOS CREADOS (REGISTROS) VAMOS A COLOCAR ESTAS DOS LINEAS DE CODIGO

```python
class Productos(models.Model):
    id_producto=models.CharField(primary_key=True,max_length=1000)
    precio=models.DecimalField(max_digits=10000,decimal_places=2)
    marca=models.CharField(max_length=20)
    nombre=models.CharField(max_length=40)
    categoria=models.CharField(max_length=20)
    diseño=models.CharField(max_length=40)
    calidad=models.CharField(max_length=50)
    def __str__(self) -> str:
        return self.nombre
```

RESULTADO



TODAS LAS TABLAS HECHAS



AHORA PARA IMPRIMIR LOS DATOS PRIMERO DEBEMOS OBTENER LOS REGISTRO DE ESTOS Y GUARDARLOS, ESTO SE HACE EN EL ARCHIVO VIEWS.PY

PRIMERO OBTENEMOS TODOS LOS REGISTOS CON LA LINEA

RECUERDA QUE GUARDAMOS LOS REGISTROS COMO DICCIONARIOS ENTONCES YA SABES QUE PRIMERO IRA EL NOMBRE DONDE GUARDAREMOS LOS DATOS EN ESTE EJEMPLO QUEDARIA

RECUERDA IMPORTAR LOS MODELOS

RECUERDA TAMBIEN QUE POR CADA TABLA NUEVA QUE LE METAS AL CODIGO TIENES QUE BORRAR LA BASE DE DATOS Y EN LA APP EN LA CARPETA MIGRATIONS BORRAR EL PYCACHE Y LOS 001,002 ETC DESPUES

COLOCAR python manage.py makemigrations, python manage.py migrate y crear un super user python manage.py createsuperuser

LINEAS DE CODIGO PARA PODER MOSTRAR DATOS EN PANTALLA

empleadosNombres y donde están los datos es el nombre que le colocaremos a la variable donde guardamos todos los registros

(NOMBRE) = NonbreModelo.objects.all()

return
render(request,'index.html',{'nombreDiccionario:"variableConRegistros"})

CON MI TABLA QUEDARIA DE LA FORMA

ListadoEmpleados = empleados.objects.all()

return render(request,'index.html','{MisEmpleados':ListadoEmpleados})

LINEAS DE CODIGO EN EL HTML PARA PODER IMPRIMIR EN PANTALLA ESOS DATOS

{% for (nombre donde guardaremos los registros en el html) in (nombre que colocamos para guardar nuetros registros)%}

 {{(nombre donde guardaremos los registros en el html).variable del modelo}}

CON MI TABLA

{% for m in / %}

 {{m.codigo}}.-.-.-{{m.nombre}}.-.-.-{{m.creditos}}

{% endfor %}

NOTA

CUANDO SON VARIOS CODIGOS PARA IMPRIMIR EN PANTALLA LOS DATOS

ListadoEmpleados = empleados.objects.all()

ListadoCliente=cliente.object.all()

return render(request,'index.html','{

'MisEmpleados':ListadoEmpleados,

'MisClientes:ListadoCliente'})

COMO SE VE SOLAMENTE SE LA AGREGA UNA COMA AL FINAL DEL PRIMER DICCIONARIO SE CREA OTRO DICCIONARIO DENTRO DE LAS MISMAS LLAVES

Y REPITES EL HTML

----------RELACIONES----------

ESTE SERIA EL PASO MAS DIFICIL PUES ES LO MAS COMPLEJO Y RARO DESCONOZCO SI VENDRA EN EL EXAMEN PERO IGUAL LO COLOCARE Y LO EXPLICARE PASO A PASO Y EL COMO FUNCIONA

BUENO PARA EMPEZAR ME GUSTARIA ACLARAR COMO FUNCIONAN CIERTAS COSAS

ESTOS EJEMPLOS LOS APLICARE EN MI TABLA VENTAS PUES HAY UNA RELACION ENTRE VENTAS Y ID PRODUCTO Y ID CLIENTE, LA SINTAXIS VIENE EN EL ARCHIVO ADMIN.PY DEL REPOSITORIO

Lunes_21_octubre_alumnoComentario

----------ADMIN----------

PRIMERO SE IMPORTAN TODOS LOS MODELOS QUE VAMOS A ULTIZAR

UN CONSEJO PARA IMPORTAR DE MANERA FACIL Y SENCILLA SIN TANTA LINEA DE CODIGO SERIA

```
from .models import (
    Productos,
    Provedores,
    Sucursales,
    Cliente,
    Inventario,
    Empleados,
    Ventas,
)

for model in [Productos, Provedores, Sucursales, Cliente, Inventario,
Empleados, Ventas]:
    admin.site.register(model)
```

FUNCIONA IGUAL QUE SI LO HICIERAMOS UNO A UNO PERO ESTE LO HACE MÁS LEGIBLE

LO IMPORTANTE A ENTENDER ES EL FOR PUES VEMOS QUE EN MODEL GUARDAMOS TODOS NUESTROS MODELOS A IMPORTAR RECUERDA PONERLO ENTRE [] DESPUES REALIZAMOS LA ACCION admin.site.register(model) LO QUE HACE ES QUE EL FOR SE REPITE POR CADA MODELO ESTO HACE QUE NO TENGAMOS QUE ESCRIBIR UN admin.site.model para cada modelo que vamos a importar

BIEN DEJANDO ESTO CLARO VAMOS A CONTINUAR

VEMOS QUE HAY UN

class ComentarioIntLine(admin.TabularInline):

    model=Frase

    extra=1

como funciona? Pues primero hay que aclarar que el ComentarioIntLine esta definiendo una clase tipo TabularInLine este se utiliza en el panel de django de admin para representar las relaciones entre modelos de manera intuitiva

después vemos que tenemos el model=Frase

aquí indicamos que esta clase (ComentarioIntLine) se relaciona al modelo Frase

El extra = 1 indica que se debe mostrar un formulario adicional vacio cuando se edita un alumno, esto porque es una relación de muchos a uno

(1 alumno puede tener muchas frases)

----------2da parte----------

Despues de la clase anterior el código sigue con

class AlumnoAdmin(admin.ModelAdmin):

    fields=
['apaterno','amaterno','nombre','fecha_nacimiento','fecha_ingreso']

    list_display=["apaterno","amaterno","nombre"]

    inlines=[ComentarioIntLine]

Para que funciona?

Pues funciona para mostrar lo que queremos en el panel de administración de django esto no es obligatorio pero le da una mejor vista y tiene más opciones

Como funciona

Pues empezamos determinando una clase en este caso es

Class AlumnoAdmin(admin.model):

El nombre de la clase puede cambiar ese no importa, pero en cambio lo que va dentro de los paréntesis es obligatorio si o si

Después le continua con el

fields= ['apaterno','amaterno','nombre','fecha_nacimiento','fecha_ingreso']

aquí determinamos los campos que queremos que se vea en el panel de administración de django

Despues sigue

List_display=["apaterno","amaterno","nombre"]

Lo que hace esto es que en el panel de django se agreguen los nombres por columas en donde visualizamos los objetos (registros) esto es más para algo visual

Y por ultimo sigue la línea

inlines=[ComentarioIntLine]

Usas el atributo inlines en la clase de administración del modelo principal para agregar la clase inline.

Cuando editas un objeto del modelo principal en el panel de administración, verás una sección donde puedes agregar o editar los objetos relacionados.

Y obvio para poder visualizar esa tabla en el panel adin utilizamos el

admin.site.register(Alumno,AlumnoAdmin)

OJO

Recuerda que debes colocar primero tu modelo (Alumno) y luego el modelo que modificamos visualmente es decir el AlumnoAdmin

----------Parte 3----------

```
@admin.register(Frase)
class FraseAdmin(admin.ModelAdmin):
    fields=  ['comentario','alumno_fk']
    list_display=["comentario"]
```

@admin.register(Frase)

Esta linea es un decorador (función que permite modificar o extender el comportamiento de otra función o método.) que registra el modelo Frase en el panel de administración de Django, al usar esto estamos vinculando nuestro modelo con el modelo a del admin es decir FraseAdmin que hereda admin.model. esto permite personalizar como se muestra y gestiona la interfaz del admin en frase

En esta clase FraseAdmin especificamos que campos debe visualizar el admin es decir los fields y la lista de objetos (list_display)

BIEN ESTE SERIA EL 1ER PASO DE 4 PARA LAS RELACIONES

----------PASO 2----------

ARCHIVO: MODELS.PY

```
from django.db import models


# Create your models here.


class Alumno(models.Model):
    apaterno =  models.CharField(max_length=50,verbose_name="Apellido Paterno")
    amaterno=  models.CharField(max_length=50,verbose_name="Apellido Materno")
    nombre = models.CharField(max_length=100,verbose_name="Nombre (s)")
    fecha_nacimiento=  models.DateField(verbose_name="Fecha de Nacimiento",null=False,blank=False)
    fecha_ingreso=   models.DateField(verbose_name="Fecha de Ingreso",null=False,blank=False)
    class Meta:
        db_table="Alumnos"
        verbose_name="Alumno"
        verbose_name_plural="Alumnos"


    def __str__(self) -> str:
        return self.nombre


class Frase(models.Model):
    comentario = models.TextField(verbose_name="Comentario", max_length=400)
    alumno_fk = models.ForeignKey(Alumno, on_delete=models.CASCADE)
    class Meta:
        db_table = "Frase"
```

verbose_name = "Frase"

Bien como vemos todo sigue igual hasta la parte de

Class Meta:

db_table="Alumnos"

verbose_name="Alumno"

verbose_name_plural="Alumnos"

db_table: Especificar el nombre de la tabla en la base de datos que se utilizará para este modelo, en lugar de usar el nombre por defecto

verbose_name: Definir un nombre legible para el modelo en singular, que se usará en el panel de administración y otras interfaces.

verbose_name_plural: : Establecer el nombre legible en plural para el modelo, facilitando la lectura en contextos donde se listan múltiples objetos del modelo.

Este paso no es obligatorio pero es algo recomendable para ser legible

----------2da parte----------

En la parte de crear la clase Frase vemos que no conocemos la función de la línea

alumno_fk = models.ForeignKey(Alumno, on_delete=models.CASCADE)

Este crea una relación de clave foránea entre el modelo Frase y el modelo Alumno. Esto significa que cada objeto de Frase está asociado a un objeto de Alumno.

alumno_fk: Es el nombre del campo que representa la relación.

on_delete=models.CASCADE: Si se elimina un Alumno, también se eliminarán todas las Frase asociadas a él. Esto mantiene la integridad referencial en la base de datos. Es decir que no se quedaran frases solas en caso de borrar un alumno

cabe aclarar que cuando es solamente la elección de un objeto de la otra tabla solo coloca el foreingkey pero si también agregaras texto haz todo lo anterior

ESTE SERIA EL PASO 2 DE 4

---------- 3er paso----------

LINEA 6 DEL ARCHIVO URLS.PY

path('Alumno/<int:id>',views.Alumno_vista,name='Alumno_vista')

La línea path('Alumno/<int:id>', views.Alumno_vista, name='Alumno_vista') en urls.py define una URL en tu aplicación Django. Cuando un usuario accede a una URL que coincide con este patrón (por ejemplo, Alumno/1), Django llama a la vista Alumno_vista y le pasa el id del alumno como argumento. El nombre Alumno_vista permite referenciar esta URL en otras partes del código, facilitando la gestión de enlaces y redirecciones.

Es decir que esto hace que el 'link' coloque una / y el id del objeto es decir registro lo cual nos redirecciona a otro archivo el cual puede contener mas información de este

----------4to Paso----------

AHORA NOS IREMOS AL ARCHIVO VIEWS.PY

from django.shortcuts import render,get_object_or_404

from .models import Alumno

```
def index_vista(request):
    misalumnos=Alumno.objects.all().order_by('id')
    return render(request, 'index.html',{"misalumnos":misalumnos})
def Alumno_vista(request,id):
    alumno = get_object_or_404(Alumno, id=id)
    return render(request,'alumno.html',{'objeto':alumno})
```

ahora vemos que no conocemos algunas líneas o una parte de ellas

vamos a empezar por la mas fácil la cual es .order_by('id') como lo dice este ordena los objetos/registros por medio del id

Obtiene un objeto Alumno: Usa get_object_or_404() para buscar el objeto Alumno en la base de datos usando su id. Si el objeto no existe, envía automáticamente una respuesta de error 404 (no encontrado).

Renderiza una plantilla: Devuelve una respuesta HTML renderizada, usando la plantilla alumno.html.

Pasa el contexto: Envía el objeto alumno a la plantilla bajo el nombre 'objeto', lo que permite acceder a los datos del alumno específico dentro del HTML.

SI TE PREGUNTAS COMO SABE DJANGO CUAL ES EL ID DEL OBJETO ES PORQUE EN TU MODELO ASI TENGAS ID_COMIDA, LE COLOCAS EL PRIMARY KEY DJANGO LO GESTIONA POR SI SOLO ASI QUE NO TE PREOCUPES SI LE COLOCAS OTRO NOMBRE APARTE DE ID SOLO RECUERDA COLOCARLE PRIMARY KEY

----------Paso final----------

ARCHIVO INDEX.HTML

```
    {% for i in misalumnos %}
```

```
    <li> <a href="Alumno/{{i.id}}">{{i.nombre}}</a></li>
  {% endfor %}
```

Colocamos un ‘a’ y en el href colocamos el nombre donde lo queremos redireccionar por ejemplo empleado/{{e.id}} y dentro de los a colocamos {{e.nombre}} este es un ejemplo recuerda siempre el for para traer los datos al html

AHORA EL TU HTML DONDE LO VAS A REDIRECCIONAR

En este ejemplo alumnos tenemos estas líneas

```
<h1>Un alumno y sus mensas (comentarios)</h1>
<h1>{{objeto.nombre}}</h1>
<hr>
<ul>
  {% for i in objeto.frase_set.all %}
    <li> {{i.comentario}}</li>
  {% endfor %}
```

Ahora utilizamos el diccionario que creamos en la función Alumno_vista

El cual es ‘objeto’:alumno

Recordemos que el alumno guarda todos los registros por medio del id y los que no encuentra tira el error 404 despues de eso crea un diccionario el cual es objeto y guardamos todos los alumnos estos contienen su información

---------- ULTIMO PASO “BLOCK”----------

Recuerda que podemos usar un html como base y simplemente en otros html colocar el código bueno pues esto se hace con los “block”

No tenemos que hacer mucho pero cuando hagamos nuestra base con el nombre base.html colocaremos en los otros html donde agregaremos la información, esta línea

{% extends "base.html" %}

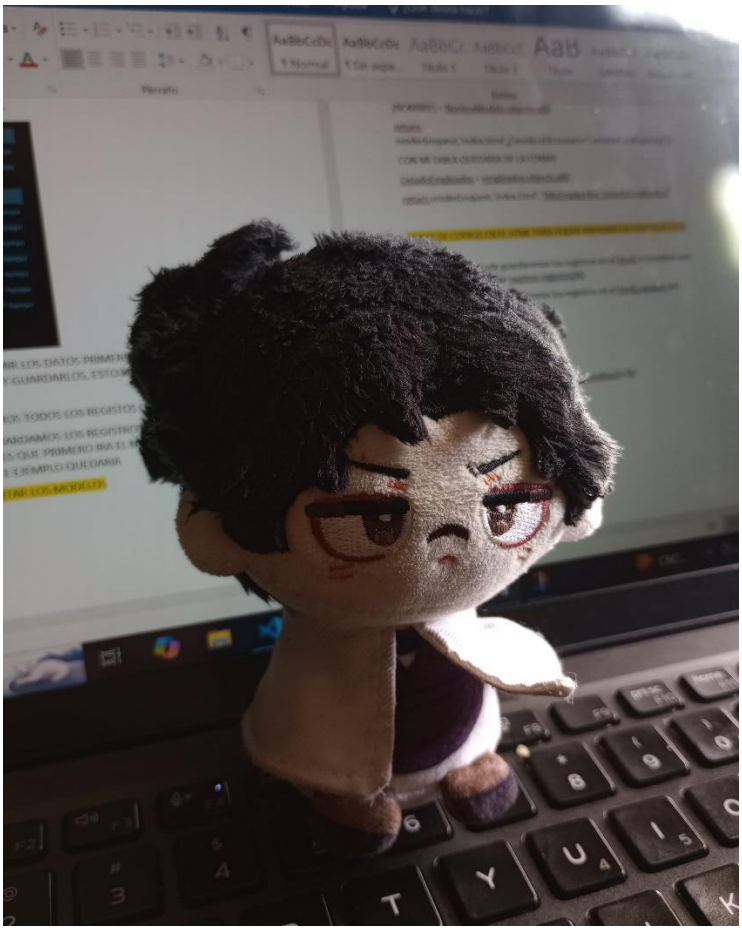
Para que nuestra base pueda utilizar los block que nombraremos

Los block pueden utilizar la info que tienen de los diccionarios creados, por lo cual ya solamente escribiremos

{% block nombre del block %}

{% end block %}

Y el en html base colocaremos las mismas líneas



PUES HASTA AQUÍ QUEDARIA LA GUIA SERIA LO MAS IMPORTANTE Y NECESARIO

DUDO QUE VENGAN ALGUNAS COSAS COMO LA REDIRECCION DE LINKS Y ESO PERO MÁS VALE PREVENIR QUE LAMENTAR CUALQUIER DUDA PUES MANDEN MENSAJE ESPERO Y LES SIRVA :3

GUIA HECHA POR ANGEL Y CHOSITO (HIJO LEGITIMO DE ANGEL) JIJIJI