1) A mismo nivel manage.py

- Crear carpeta apps

- Crear carpeta templates

- Crear carpeta static

- Crear carpeta media

2) Ingresar a la carpeta principal del proyecto:

- Crear carpeta settings

- Dentro de settings creamos: base.py, local.py, production.py

3) Dentro de static

- Crear carpeta css, img y js

4) pasamos el contenido del settings.py a base.py y eliminamos settings.py.-

5) En el archivo base.py modificamos la configuración de la siguiente manera:

import os

from django.urls import reverse\_lazy

# AUTH

LOGIN\_REDIRECT\_URL = reverse\_lazy('home')

LOGOUT\_REDIRECT\_URL = reverse\_lazy('home')

LOGIN\_URL = reverse\_lazy('login')

# En TEMPLATES > DIRS :

'DIRS': [os.path.join(os.path.dirname(BASE\_DIR),'templates')],

LANGUAGE\_CODE = 'es-ar'

TIME\_ZONE = 'America/Argentina/Buenos\_Aires'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

STATIC\_URL = '/static/'

STATICFILES\_DIRS = (

os.path.join(os.path.dirname(BASE\_DIR),'static'),

)

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(os.path.dirname(BASE\_DIR),'media')

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

- Base Dejamos la configuración general que no cambia si está en local o en producción.

- Local (en desarrollo) Desarrollamos en localhost, con conexiones y contraseñas de prueba.

- Production Ya está subido a internet, con datos reales en DB.

6) de base.py a local.py:

from .base import \*

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Database

# <https://docs.djangoproject.com/en/4.2/ref/settings/#databases>

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

7) Modificar en manage.py la ruta en donde está ahora la configuración (local.py).

Linea 9 -> os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'prueba.settings.**local**')

Django está compuesto por aplicaciones. La idea es hacer las aplicaciones mas independientes posibles y luego implementarlas a todas.

La carpeta principal donde se encuentra el settings es nuestra aplicación principal.

Django funciona con el modelo MVT (Model View Template)

Navegador -> Activa urls.py manipula y checkea que vista (view) debe ejecutar -> Va al models para conectar y consultar con la DB -> y con esa información levanta un template -> lo manda al navegador.

Creamos en la app principal un archivo views y lo importamos en el urls.py:

from . import views

urlpatterns = [

path('admin/', admin.site.urls),

path('', views.home, name='index')

]

Herencia de HTMLs.

{% extends ‘base.html’ %}

{% load static %}

{% block content %}

{% endblock content %}

Creación de aplicaciones:

django-admin startapp noticias

Normalmente los nombres van en plural

Cuando se crea una aplicación, hay que agregarla a INSTALLED\_APPS en el settings/base.py

Luego agregar “apps.” al nombre de la aplicación ubicado en el archivo apps.py de la aplicación:

*class* NoticiasConfig(AppConfig):

    default\_auto\_field = '*django.db.models.BigAutoField*'

    name = '*apps.noticias*'

Y luego creamos un archivo urls.py en la aplicación con su plantilla default al no traer por defecto.

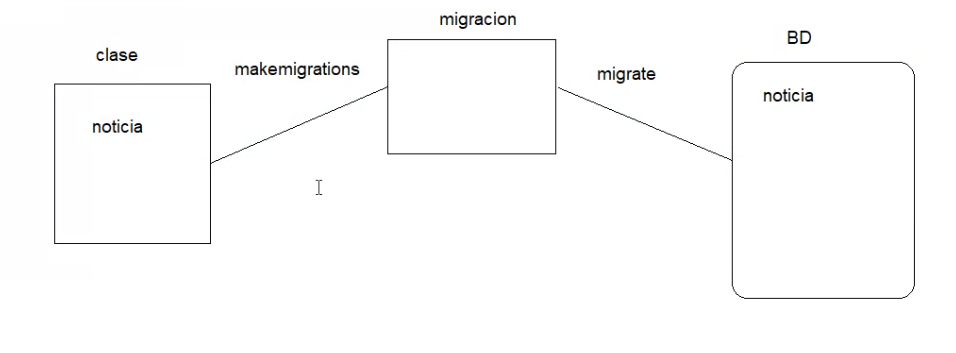
En urls.py pero de la aplicación principal hacemos referencia al archivo urls.py de la app en cuestión de la siguiente manera:

    path('*noticias/*', include('*apps.noticias.urls*')),

importando include de django.urls

En views.py de la aplicación hacemos referencia a una carpeta aplicacion/archivo.html y la creamos también en la carpeta templates para hacer referencia.

Los modelos es el modo de conectar nuestro proyecto a la base de datos.



Creamos el modelo  
python manage.py makemigrations  
Se pasó el modelo al stage de migrations.   
python manage.py migrate  
Impactó en la base de datos.

Para el admin:   
python manage.py createsuperuser  
user: scz  
pss: fscz

Para traer la información de la base de datos trabajamos en el archivo views.py

Existen las vistas basadas en funciones y en clases.

**Vista basada en funciones:**

*def* home\_noticias(*request*):

    # *ORM*

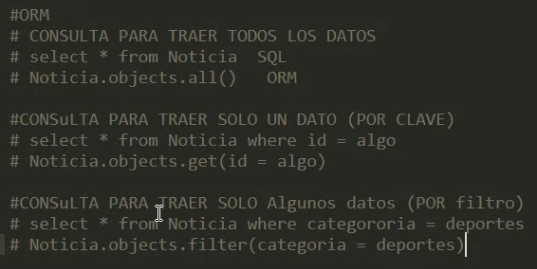
    noticias = Noticia.objects.all()

    # *contexto*

    ctx = {}

    ctx['*noticias*'] = noticias

*return* render(request, '*noticias/home.html*', ctx)

ORM:

Y en el HTML mostramos la información de la siguiente manera:

{% block contenido %}

    <h1>HOME NOTICIAS </h1>

    {% for n in noticias %}

        <div>

            <h2>{{n.titulo}}</h2>

            <h3>{{n.contenido}}</h3>

        </div>

    {% endfor %}

{% endblock contenido %}

Y en urls.py se muestran así:

# *url para una view basada en funciones*

path('', views.home\_noticias, *name*="*h\_noticias*")

**View basada en clases:**

Así como tenemos el ORM para la vista basada en funciones, también lo tenemos para las basadas en clases:

Debemos importar primero dichas interfaces:

*from* django.views.generic *import* ListView

y luego podemos aplicarlas:

# *ListView -> findAll*

# *DetailView -> findOne*

# *CreateView -> createOne*

# *DeleteView -> deleteOne*

*class* home\_noticias\_clase(ListView):

    model = Noticia

    template\_name = '*noticias/home.html*'

    context\_object\_name = '*noticias*'

ListView por ejemplo trae automaticamente del model Noticia todas las noticias en un object y se lo pasa al template y con context\_object\_name le damos un nombre a ese objeto ya creado por ListView.

En HTML se muestra la información de igual manera que en las vistas basadas en funciones.

Pero existe una diferencia al momento de crear la URL y se hace de la siguiente manera:

    # *url para una view basada en clases*

    path('', views.home\_noticias\_clase.as\_view(), *name*="*h\_noticias*")

pillow ayuda a manipular las imágenes (ImageField)

pip install pillow

Y actualizamos el model de la siguiente manera:

*class* Categoria(models.Model):

    nombre = models.CharField(*max\_length*=100)

*def* \_\_str\_\_(*self*):

*return* *self*.nombre

# *Esta herencia es la que va a ocupar el ORM*

*class* Noticia(models.Model):

    publicado = models.DateTimeField('*creado*', *auto\_now\_add*=*True*)

    modificado = models.DateTimeField('*modificado*', *auto\_now*= *True*)

    titulo = models.CharField(*max\_length*= 250)

    contenido = models.TextField()

    imagen = models.ImageField(*upload\_to*= '*noticias*')

    categoria = models.ForeignKey(Categoria, *on\_delete*=models.CASCADE)

    # *models.CASCADE cuando se borra la categoría, se borra todas las publicaciones relacionadas a esa categoría.*

*def* \_\_str\_\_(*self*):

*return* *self*.titulo