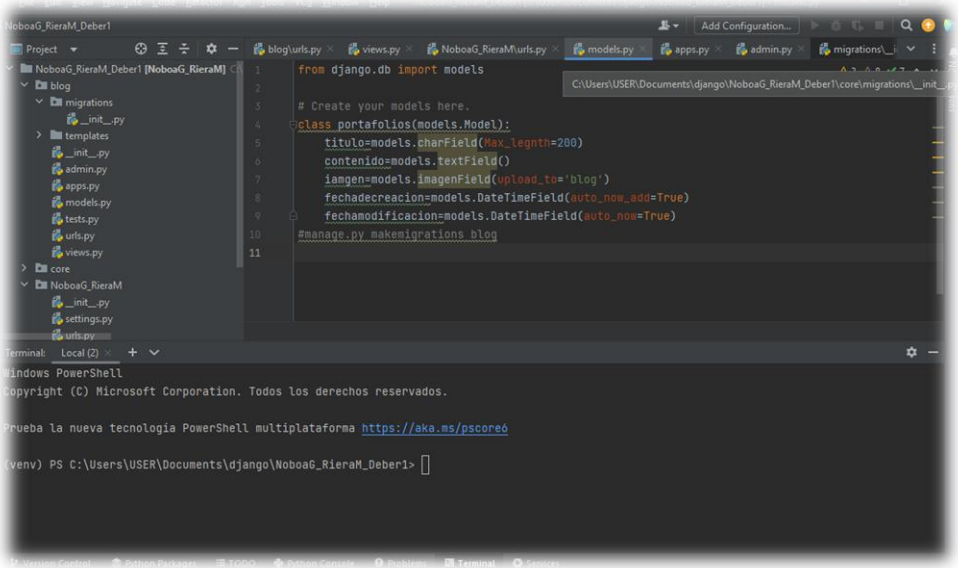
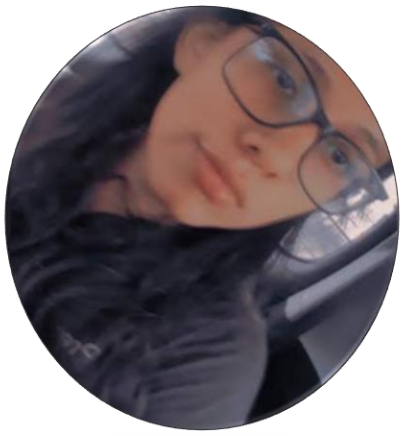
Instituto Superior Tecnológico
GUAYAQUIL

TALLER # 1

TEMA: GIT-GITHUB





Estudiante: Melany Paola Riera Andrade

Carrera: Desarrollo de software

Curso: Tercer nivel

Paralelo: E

Docente: Carlos Luis Pasmiño Palma

INDICE

Tabla de contenido

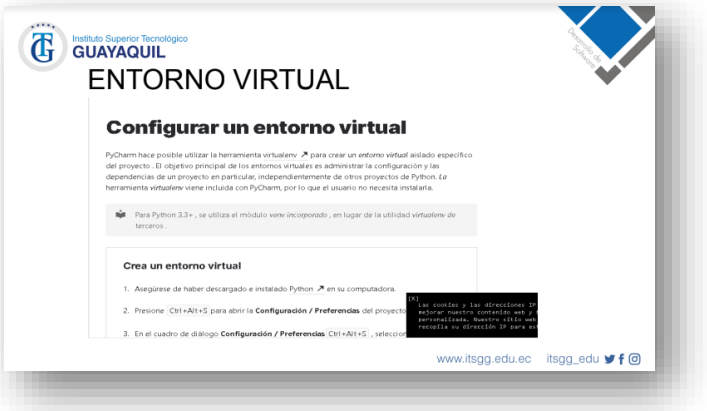
¿Qué es Django?

es un framework de aplicaciones web gratuito y de código abierto (open source) escrito en Python. Diseñado para realizar aplicaciones de cualquier complejidad en unos tiempos muy razonables. Respeta el patrón de diseño conocido como modelo–vista–controlador (MVC).

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio «DRY» (del inglés Don't Repeat Yourself,«No te repitas»). El lenguaje Python es usado en todos los componentes del framework, incluso en configuraciones, archivos,1 y en sus modelos de datos.

Flamework.- Es un conjunto de código desarrollado por uno o muchos programadores para hacer más fácil la programación.

¿Qué es la máquina virtual en Django?

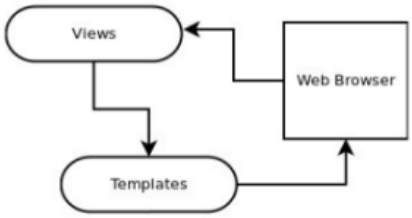


Un entorno virtual es un entorno aislado creado para un proyecto. Este entorno tiene su propio intérprete, bibliotecas y paquetes, lo que significa que el intérprete y las dependencias instaladas pertenecen solo a este proyecto. Podemos tener diferentes versiones y diversas bibliotecas para varios proyectos con entornos virtuales sin acumular las dependencias instaladas globalmente. El entorno de desarrollo es una instalación de Django en tu computadora local que puedes usar para desarrollar y probar apps Django antes de desplegarlas al entorno de producción.

Las principales herramientas que el mismo Django proporciona son un conjunto de scripts de Python para crear y trabajar con proyectos Django, junto

con un simple servidor web de desarrollo que puedes usar para probar de forma local (es decir en tu computadora, no en un servidor web externo) aplicaciones web Django con el explorador web de tu computadora.

¿Qué es MVT en Django?

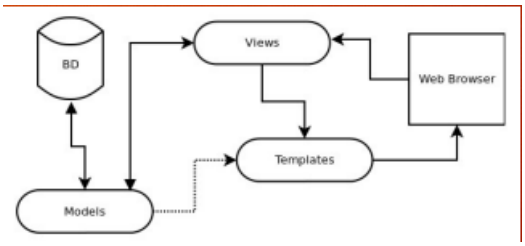


PATRON: MODELO VISTA TEMPLATE

Actualmente se podría indicar que se recibe una petición del navegador, se ejecuta la vista correspondiente y se renderiza el template para que el navegador muestre el HTML resultante

En el momento en que se involucra la base de datos y los modelos, este proceso se extiende:

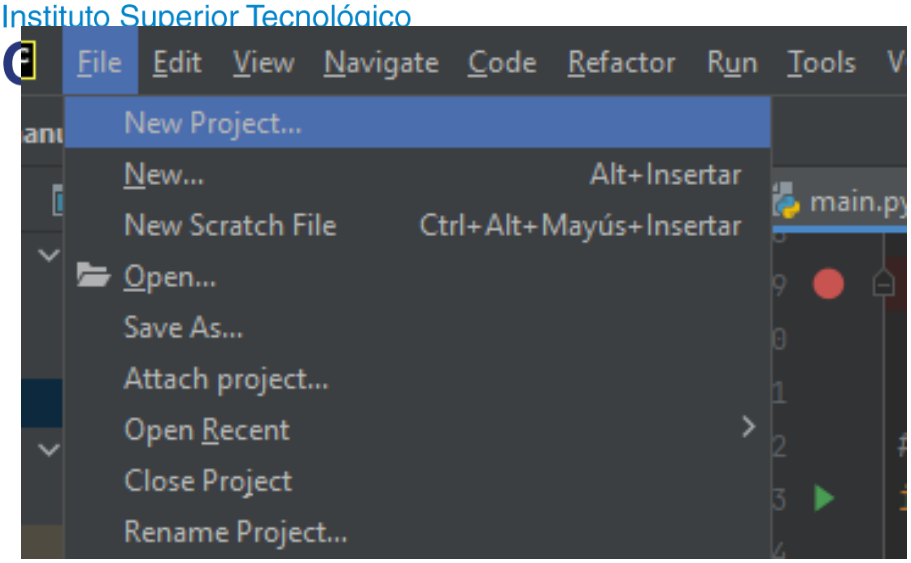
se recibe la petición, se pasará a la vista, en la vista recuperamos los datos del modelo correspondiente y finalmente renderizaremos el template pero esta vez integrando los datos dinámicos recuperados del modelo, antes de enviar al html final al navegador



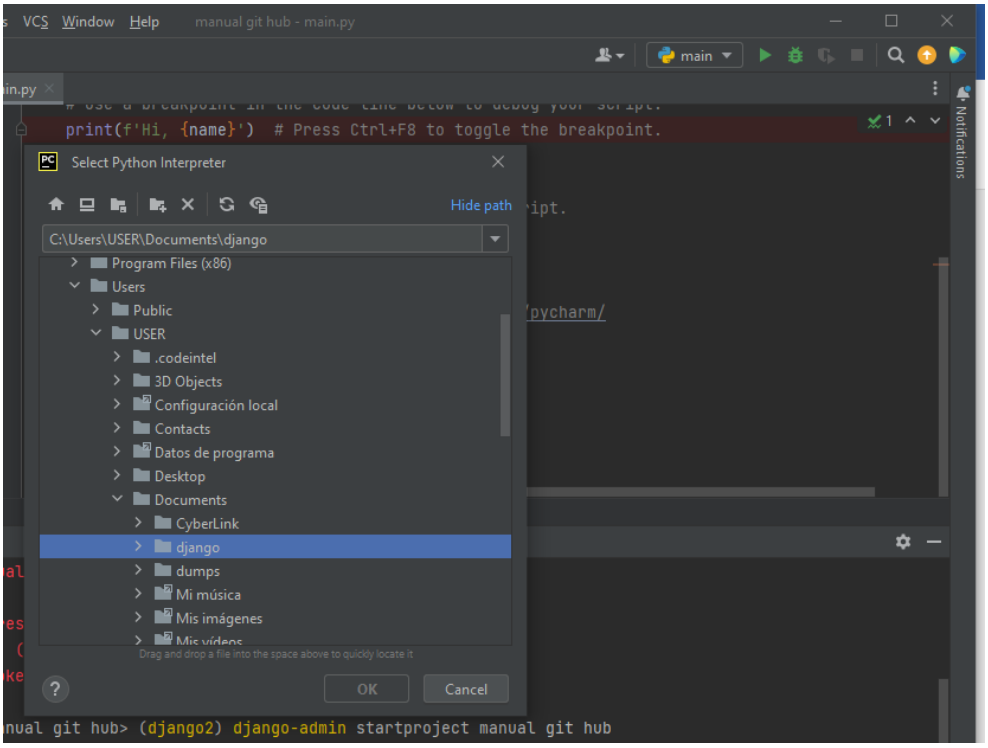
- **Modelo:** Maneja todo lo relacionado con la información, esto incluye como acceder a esta, la validación, relación entre los datos y su comportamiento.
- **Vista:** Es un enlace entre el modelo y el template. Decide que información sera mostrada y por cual template.
- **Template:** Decide como sera mostrada la información.

Crear un proyecto con la máquina virtual.

1. En el menú superior de PyCharm, accedemos a File —> New Project.

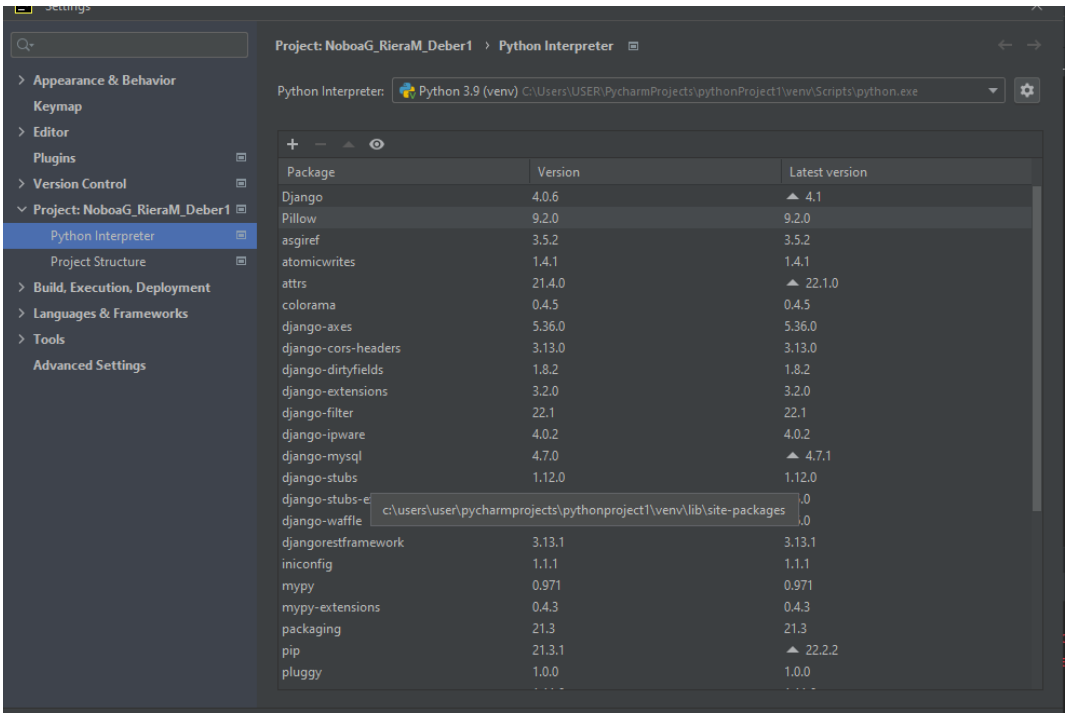


2. En el menú que aparece en la barra lateral, seleccionamos Django.

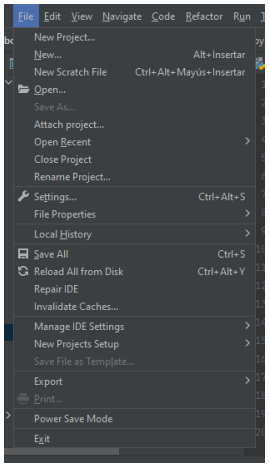


3. Definimos el directorio de destino del proyecto Django en el campo location. En nuestro caso, sustituimos untitled por ejemplo Pulsamos el botón Create.

Descargar los instaladores de Django al proyecto



Como podemos ver en la imagen tenemos lo que esta instalado el django y el pillow que es el encargado de dejar subir imágenes a nuestro proyecto



Para proceder a la instalación

1. Vamos a nuestro menú,
2. Buscamos la opción File
3. Una vez allí visualizamos la opción settings
4. Luego buscamos la opción Python interprete
5. Allí solo tenemos que buscar lo que vamos a descargar, ya sea django o su complemnto pillow
6. Y listo ya tenemos hecha nuestra instalacion

Crear un proyecto para programar en Django

Usar la herramienta django-admin para crear la carpeta del proyecto, los ficheros de plantillas básicos y el script de gestión del proyecto (manage.py).

Usar manage.py para crear una o más aplicaciones.

Registrar las nuevas aplicaciones para incluirlas en el proyecto.

En primer lugar abre una ventana de comandos/terminal, navega hasta donde quieres almacenar tus apps Django (hazlo en algún lugar que sea fácil de encontrar, como dentro de la carpeta de tus documentos), y crea una carpeta para tu nuevo sitio web

Crear el nuevo proyecto usando el comando django-admin startproject como se muestra, y navega luego dentro de la carpeta.

La herramienta django-admin crea una estructura de carpetas/ficheros como se muestra abajo:

manage.py

settings.py

urls.py

settings.py contiene todos los ajustes del sitio. Es donde registramos todas las aplicaciones que creamos, la localización de nuestros ficheros estáticos, los detalles de configuración de la base de datos, etc.

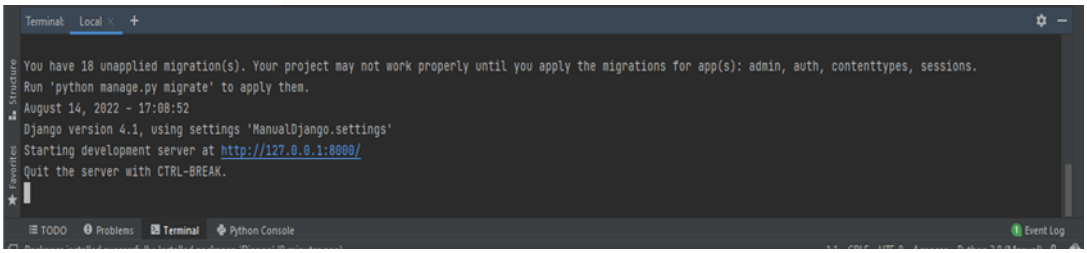
urls.py define los mapeos url-vistas. A pesar de que éste podría contener todo el código del mapeo url, es más común delegar algo del mapeo a las propias aplicaciones, como verás más tarde.

El script manage.py se usa para crear aplicaciones, trabajar con bases de datos y empezar el desarrollo del servidor web.

Ejecutar el proyecto y el mensaje de felicitaciones.

Terminando de crear nuestro proyecto , para verificar que en nuestro proyecto este instalado django, en nuestro terminal escribimos el commando Manage.py runserver .

Una vez ejercutado nos deberá mostrar una direccion ,copiamos esta direccion en nuestro navegador .

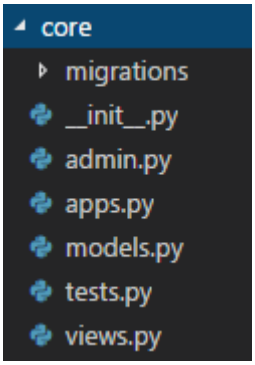


Crear una Apps core.

Las Apps activas en un proyecto de Django, las encontramos definidas en el fichero de configuración **settings.py**, en la lista *INSTALLED_APPS*:

```
webpersonal/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Un proyecto puede contener múltiples apps, y una app puede ser incluida en múltiples proyectos.

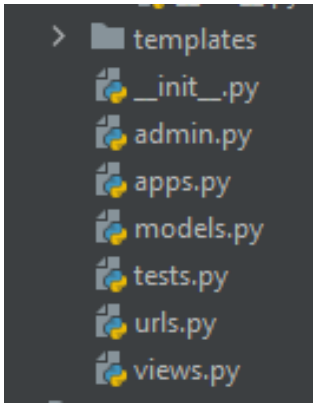


Así que vamos a la terminal, presionamos *Control + C* para cancelar la ejecución del servidor y escribimos:

```
python manage.py startapp core
```

Al hacerlo podréis observar como se ha creado un nuevo directorio core en nuestro proyecto:

¿Qué es la Carpeta Templates?

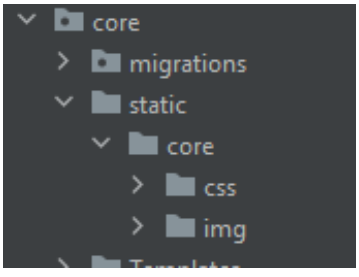


Una plantilla es un archivo de texto que determina la estructura o diseño de un archivo (como una página HTML), con marcadores usados para representar el contenido real. Django automáticamente buscará plantillas en un directorio llamado 'templates' de su aplicación.

Una plantilla Django es un documento de texto o una cadena de Python marcada con el lenguaje de plantillas Django. Algunas construcciones son reconocidas e interpretadas por el motor de plantillas. Las principales son las variables y las etiquetas.

¿Qué es la Carpeta stactic?

Los archivos estáticos, como su nombre indica, son aquellos que no cambian, y su hoja de estilos (CSS/SCS) no cambia para cada solicitud del cliente, aunque la plantilla puede ser dinámica. Además, su logotipo, la imagen en el diseño no



cambiará a menos que rediseñe, por lo que estos son archivos estáticos que necesitan ser renderizados con plantillas.

Básicamente tenemos tres tipos de archivos estáticos, css, archivos javascript y archivos multimedia/plantillas estáticas, etc. todos se presentan de la misma manera, pero de acuerdo a sus convenciones y usos. Cómo configurar un archivo estático

En primer lugar, puede crear una carpeta para todos los archivos estáticos en la carpeta raíz. El nombre de la carpeta comúnmente acordada es static. Por lo tanto, si ya ha creado una carpeta de plantillas en el directorio raíz, puede crear carpetas como carpetas estáticas en esta ruta.

Después de crear una carpeta estática en la carpeta raíz del proyecto, necesitamos configurar el archivo settings.py para decirle al servidor web Django que busque todos los archivos estáticos en esa carpeta. Para ello, vaya al archivo settings.py, donde ya conoce la ubicación del archivo settings.py (en el proyecto). Añadir lo siguiente al final del archivo settings.py.

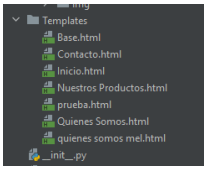
```
# import os
# STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static/"),
)


STATIC_URL = '/static/'
```

import os se ignora si se ha importado o STATIC_URL si ya existe en el archivo. STATICFILES_DIRS es la configuración que le decimos al entorno Django que busque todos los archivos estáticos en el directorio base/raíz del proyecto donde se encuentra la carpeta static/. os.path.join() realmente obtiene la ruta del directorio de nuestro sistema operativo a la carpeta especificada en el proyecto BASE_DIR es la ruta del proyecto, y agregamos la carpeta estática a la ruta del proyecto. La última y clave es la ruta "static/", que puede ser otra ubicación para crear carpetas estáticas en su proyecto.

¡Eso es todo! Sí, es tan simple. Ahora podemos crear archivos estáticos y renderizarlos en plantillas.

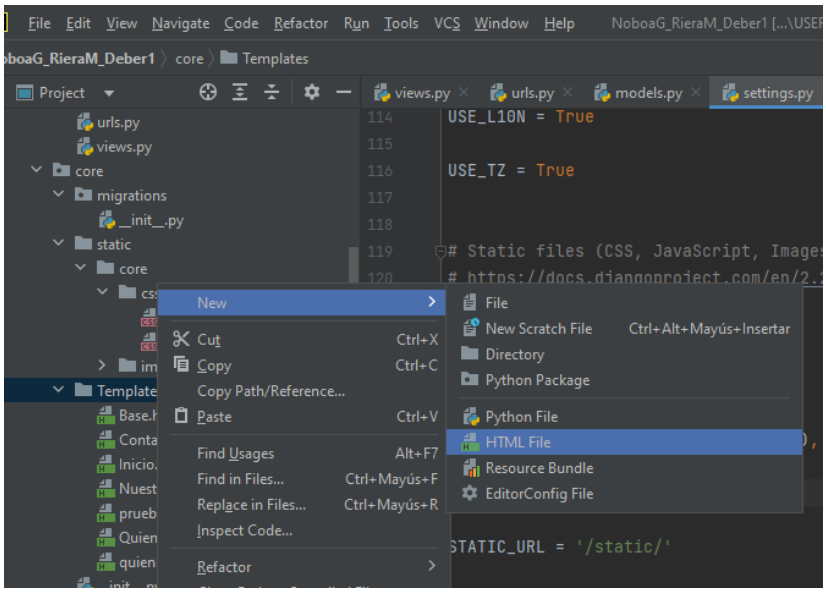
Crear un archivo base html en la APPS core

Lo primero que hay que hacer es localizar nuestra carpeta de nuestro proyecto

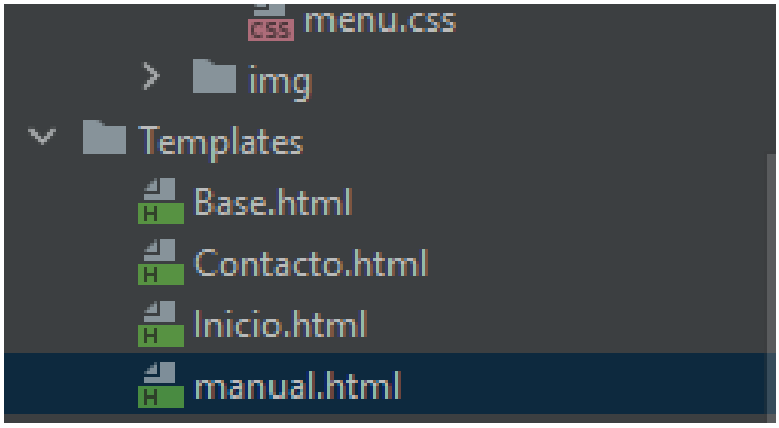


tempate

una vez allí damo clic derecho en la carpeta tempate y visualizaremos otro menú seleccionamos el que dice new , luego el tipo de archivo a crear en este caso seria un html



Luego seleccionemos y le ponemos nombre y listo ya se creo el archivo html



Como se llaman a los CSS desde el archivo base html.

Lo primero que debemos hacer es tener nuestro archivo css listo en la carpeta static

En nuestro Html donde tenemos la base de nuestro proyecto donde esta nuestro body debajo de title escribimos lo siguiente: en la static entre comillas, tenemos una dirección esta nos indica donde se encuentra el documento dentro de carpeta static, tenemos la sub carpeta llamada core dentro de ella otra sub carpeta llamada css y por ultimo el css que usaremos que se llama estilo.css.

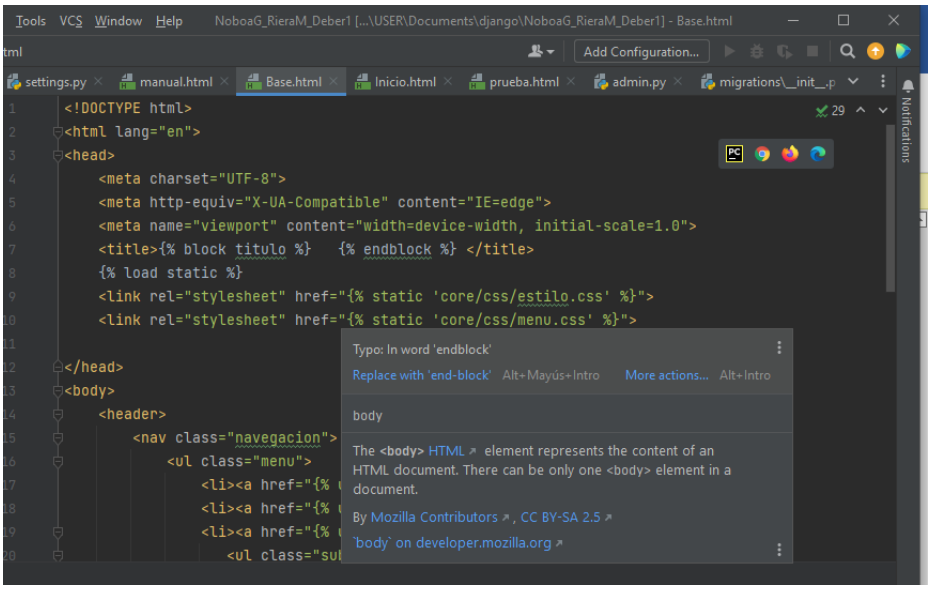
```
10      <link rel="stylesheet" href="{% static 'core/css/menu.css' %}">
11
```

Si nuestro proyecto fue creado con exito debería mostrarnos, nuestro proyecto según como lo hayas hecho

Como consume un archivo hijo html al utilizar la herencia del archivo base html

Vamos a empezar creando una plantilla base. Creamos el fichero **base.html** dentro de **templates/core**. podemos escribir **html:5** y presionar tabulador para generar una plantilla HTML bien estructurada (podemos poner lang=es y un título).

`core/templates/core/base.html`



Ahora la parte que nos interesa. ¿Qué es lo que se repetirá en todas nuestras páginas? ¿La cabecera y el menú no? Pues vamos a ponerlo dentro del body:

`core/templates/core/base.html`

Justo debajo viene la parte que cambia en cada página.

Poned lo siguiente:

En este caso el template tag **block** sirve para definir un bloque de contenido con un nombre.

```
{% block cotenido %}  {% endblock %}
    <!--<img src="{% static 'core/img/fondoriot.jpg' %}">-->
<footer class="footer">
    Derechos Reservados © 2022 <br>
    diseñado por: George Noboa , Melany Riera <br>
    www.PcXtreme.com
</footer>
</body>
</html>
<link rel="stylesheet" href="{% static 'core/css/estilo.css' %}">
```

Así es omo se utiliza la herencia con plantillas, con los template tags **block** y **extends**. Hemos usado la etiqueta de plantilla {% block %} para crear un área en la que se insertará HTML. Ese HTML vendrá de otra plantilla que extiende esta (base.html).

Dentro del hijo deberá ir lo que nos permitira utilizar la herencia llamando todo lo que tiene el padre

```
{% extends 'Base.html' %}
```

```
{% block titulo % }   Inicio{% endblock %}
```

Crear un view que llame al html hijo

Para crear un view que llame al hijo nos dirigimos al views.py en nuestro proyecto coreç Esto nos servirá para llamar al contacto.html en nuestro menú.

```
def datos(request):
  return render(request,'manual.html')
```

Podríamos fácilmente escribir la vista como una función regular (tal como nuestra vista de índice anterior), la cual consultaría a la base de datos, y luego llamar a render() para pasar dicha lista a una plantilla específica

Crear la urls que llame al views.

Ahora tenemos que asociar la vista que acabamos de definir con una dirección , para hacerlo vamos a manejar dos ficheros, uno lo crea Django al iniciar el proyecto y está en ./myapps/urls.py , el otro lo tendremos que crear nosotros dentro del directorio de nuestra proyecto.

Vamos a editar el primero ./myapps/urls.py

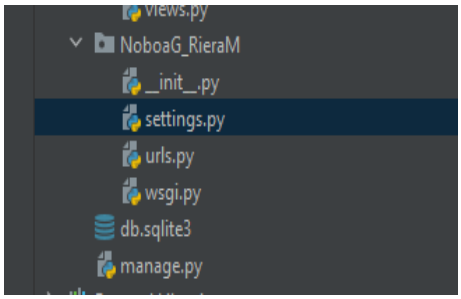
```
urlpatterns = [
    path('',views.Inicio, name='Inicio'),
    path('Quienes Somos',views.QuienesSomos, name='QuienesSomos'),
    path('Nuestros Productos',views.NuestrosProductos, name='NuestrosProductos'),
    path('Contacto',views.Contacto, name='Contacto'),
    path('quienes somos mel',views.datos, name='datos')
    path('manual',views.datos, name='datos')

```

La función url() necesita dos argumentos , el primero será una expresión regular (regex) que utiliza para identificar la dirección que escribimos en el navegador, el segundo es la función a la que llamará, first_view() para nosotros, pasándole como primer argumento el objeto HttpRequest que mencionábamos antes. Además se pueden pasar tres argumentos opcionales, name lo usaremos para identificar nuestra url de forma única y nos servirá para poder cambiar la url sin tener que modificar las partes del proyecto donde hacemos referencia a ella. Los otros dos argumentos son kwargs y prefix que no usaremos en nuestra aplicación.

Integrar la aplicación APPS core al proyecto principal

as aplicaciones incluyen alguna combinación de modelos, vistas, plantillas, etiquetas de plantilla, archivos estáticos, URL, middleware, etc. Por lo general, están conectadas a proyectos con la INSTALLED_APPS configuración y, opcionalmente, con otros mecanismos como URLconfs, MIDDLEWARE configuración o herencia de plantilla

Después de crear una aplicación, también necesitamos decirle a Django que debe utilizarla. Eso se hace en el fichero mysite/settings.py -- ábrelo en el editor. Tenemos que encontrar INSTALLED_APPS y agregar una línea que contienenuestra nueva app, justo por encima de]. El producto final debe tener este aspecto:




```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
]
```

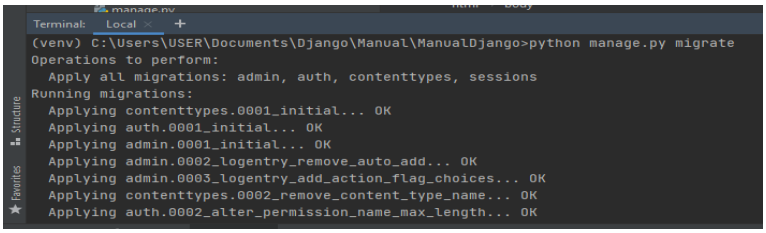
Crear las tablas del sistema de usuarios para utilizar el panel de administración.

- **`migrate`** que se encarga de aplicar y desaplicar migraciones.

Django realizará migraciones para cualquier cambio en sus modelos o campos, incluso las opciones que no afectan la base de datos, ya que la única forma en que puede reconstruir un campo correctamente es tener todos los cambios en el

historial, y es posible que necesite esas opciones en algunas migraciones de datos más adelante (por ejemplo, si ha configurado validadores personalizados).

Pero nosotros vamos a migrar todas nuestras tablas a nuestro panel de administración y por ello necesitamos el comando Python manage.py migrate



Y el sistema automáticamente subirá todas las tablas

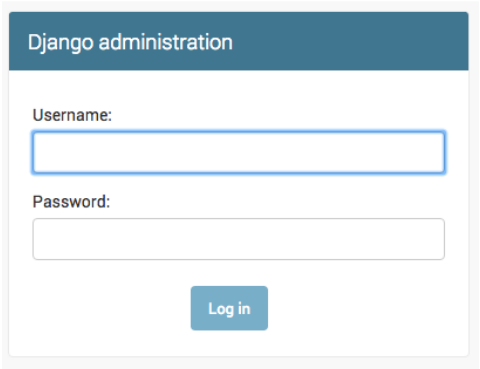
Crear un usuario para poder ingresar al Panel de Administración

Ejecute manage.py createsuperuser para crear un administrador.

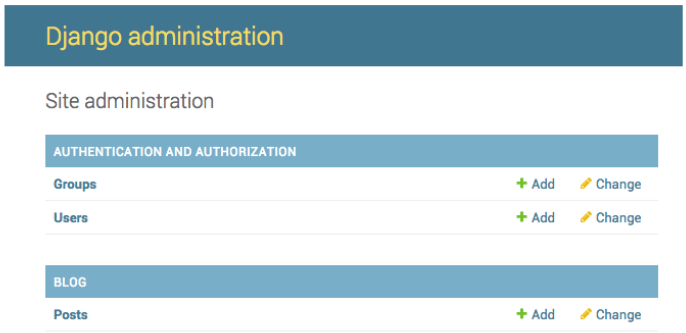
Cuando te lo pida, escribe tu nombre de usuario (en minúscula, sin espacios), email y contraseña. No te preocupes si no puedes ver la contraseña que estás tecleando - así es como debe ser. Tecléalo y pulsa intro para continuar. Luego, verás algo así (donde username y email serán los que escribiste anteriormente):

```
Username: ola
Email address: ola@example.com
Password:
Password (again):
Superuser created successfully.
```

Para verificar que el usuario se ha creado correctamente solo debemos ir con el link de la pagina y agregar /admin lo que nos llevara a ver una pantalla para ingresar usuario y contraseña



Una vez ingresado nuestro nuevo usuario debería levarnos al panel de administrador de Django.



Que es un modelo en Django

Un modelo en Django es un tipo especial de objeto que se guarda en la base de datos. Una base de datos es una colección de datos. Es un lugar en el cual almacenarás la información sobre usuarios, tus entradas de blog, etc. Al diseñar sus modelos, tiene sentido tener modelos separados para cada "objeto" (grupo de información relacionada

Los modelos están definidos, normalmente, en el archivo models.py de la aplicación. Son implementados como subclases de django.db.models.Model, y pueden incluir campos, métodos y metadata. El fragmento de código más abajo muestra un modelo "típico", llamado MyModelName

Crear un modelo en Django.

En el archivo models.py definimos todos los objetos llamados Models. Este es un lugar en el cual definiremos nuestra entrada del blog.
Abre models.py en el editor, borra todo, y escribe código como este:

```
from django.db import models

# Create your models here.
class portafolios(models.Model):
    titulo=models.charField(Max_legnth=200)
    contenido=models.textField()
    iamgen=models.imagenField(upload_to='blog')
    fechadecreacion=models.DateTimeField(auto_now_add=True)
    fechamodificacion=models.DateTimeField(auto_now=True)
#manage.py makemigrations blog
```

class portafolio(models.Model):, esta línea define nuestro modelo (es un objeto).

- class es una palabra clave que indica que estamos definiendo un objeto.
- Portafolio es el nombre de nuestro modelo. Podemos darle un nombre diferente (pero debemos evitar espacios en blanco y caracteres especiales). Siempre inicia el nombre de una clase con una letra mayúscula.
- models.Model significa que Portafolio es un modelo de Django, así Django sabe que debe guardarlo en la base de datos.

Ahora definimos las propiedades de las que hablábamos: title, text, created_date, published_date y author. Para ello tenemos que definir el tipo de cada campo (¿es texto? ¿un número? ¿una fecha? ¿una relación con otro objeto como un User (usuario)?)

- models.CharField, así es como defines un texto con un número limitado de caracteres.
- models.TextField, este es para texto largo sin límite. Suena perfecto para el contenido de la entrada del blog, ¿no?
- models.DateTimeField, este es fecha y hora.
- modelos.ForeignKey, este es una relación (link) con otro modelo.

Migrar el Modelo a la base del Panel de Administración.

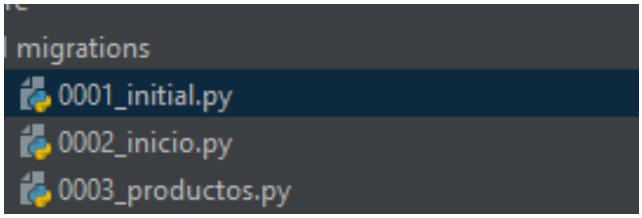
manage.py makemigrations core

Mediante la ejecución de makemigrations, le estás diciendo a Django que has hecho algunos cambios a sus modelos y que desea que los cambios se almacenarán como la migración.

Las migraciones son cómo almacena Django cambios a sus modelos (y por tanto el esquema de base de datos) - son simplemente los archivos en el disco.

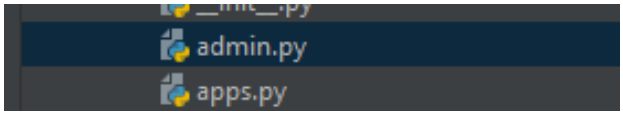
Ejecutar el comando python manage.py makemigrations core para crear migraciones

Si la migración se llevo a cabo deberá aparecerte un archivo asi



Integrar el Modelo al Panel de Administración.

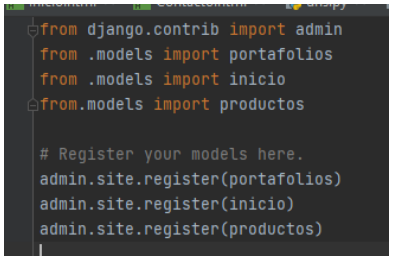
Para activar los dos modelos recién creados editamos el script **admin.py**



notar que ya importa django.contrib.admin:

from django.contrib import portafolio

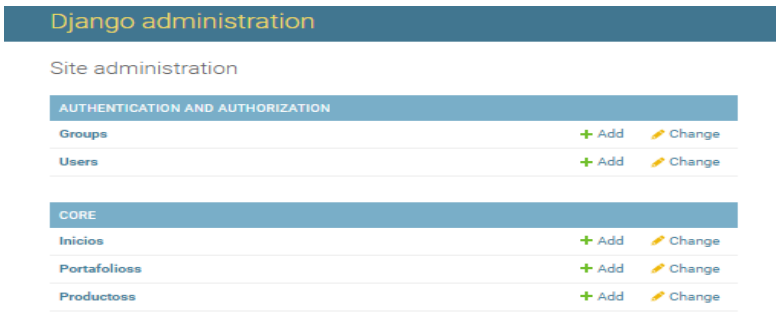
Registra los modelos copiando el texto siguiente al final del archivo. Este simple código esta importando los modelos y después llama a admin.site.register para registrar a cada uno de ellos.



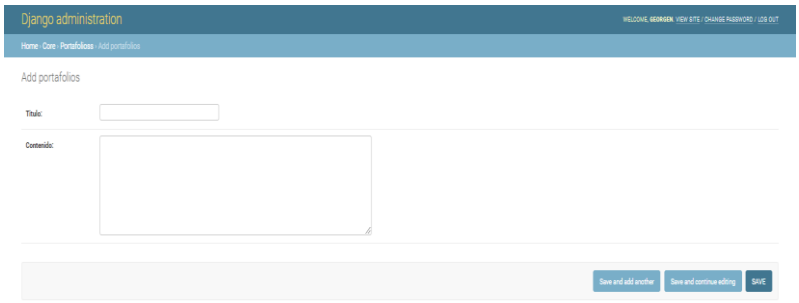
```
from django.contrib import admin
from .models import portafolios
from .models import inicio
from.models import productos

# Register your models here.
admin.site.register(portafolios)
admin.site.register(inicio)
admin.site.register(productos)
```

Ingresar información al modelo por el Panel de Administración.

Vuelve a tu navegador. Entra con las credenciales de super usuario que escogiste; verás el panel de administrador de Django.

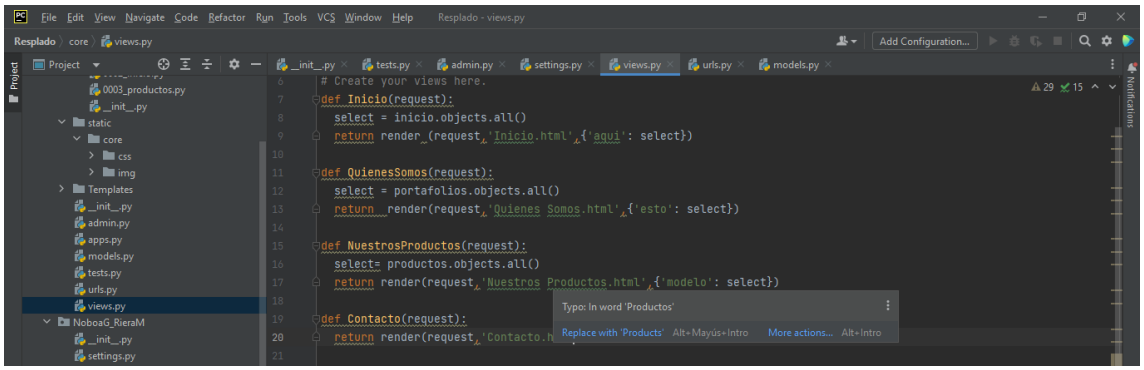


Escoge cualquier modelo que deseas ingresar una información adicional en este caso portafolio



Ingresa los datos que quieras registrar y guardamos eso seria todo

Realizar la consulta de todo lo ingresado en el modelo desde el views.



linea Portafolios.objects.all():

En primer lugar, tenemos nuestro modelo definido Portafolios. Aquí no hay nada extraño: cuando quieras buscar datos, usa el modelo para esto.

Luego, tenemos objects. Técnicamente, esto es un administrador (manager). Los administradores se encargan de todas las operaciones a "nivel de tablas" sobre los datos incluidos, y lo más importante, las consultas.

Todos los modelos automáticamente obtienen un administrador `objects`; debes usar el mismo cada vez que quieras consultar sobre una instancia del modelo.

Finalmente, tenemos all(). Este es un método del administrador objects que retorna todas las filas de la base de datos. Aunque este objeto se parece a una lista, es realmente un QuerySet — un objeto que representa algún conjunto de filas de la base de datos.

En la función render() de nuestra view estas variables serán reemplazadas por sus valores asociados cuando la plantilla sea renderizada. En este caso seria el html donde queremos que se vea nuestro modelo

Mostrar los datos guardados en el modelo al html hijo.