

**TALLER # 1**

**TEMA: GIT-GITHUB**

Carrera: Desarrollo de software

**Curso: Tercero H**

**Nombre : Jorddy Quito Apolinario**

**Docente: Carlos Luis Pazmiño Palma**

**Foto:**



Contenido

[Manual de Django 3](#_Toc111484411)

[1) ¿Qué es Django? 3](#_Toc111484412)

[2) ¿Qué es la máquina virtual en Django? 3](#_Toc111484413)

[3) ¿Qué es MVT en Django?? 3](#_Toc111484414)

[4) Crear un proyecto con la máquina virtual. 3](#_Toc111484415)

[5) Descargar los instaladores de Django al proyecto 3](#_Toc111484416)

[6) Crear un proyecto para programar en Django 3](#_Toc111484417)

[7) Ejecutar el proyecto y el mensaje de felicitaciones. 4](#_Toc111484418)

[8) Crear una Apps core. 4](#_Toc111484419)

[9) ¿Qué es la Carpeta Templates? 6](#_Toc111484420)

[10) ¿Qué es la Carpeta stactic? 6](#_Toc111484421)

[11)Crear un archivo base HTML en la APPS core 6](#_Toc111484422)

[12) Como se llaman a los CSS desde el archivo base html. 8](#_Toc111484423)

[13) Como consume un archivo hijo html al utilizar la herencia del 9](#_Toc111484424)

[archivo base html. 9](#_Toc111484425)

[14) Crear un view que llame al html hijo 9](#_Toc111484426)

[15) Crear la urls que llame al views. 9](#_Toc111484427)

[16) Integrar la aplicación APPS core al proyecto principal 9](#_Toc111484428)

[17) Crear las tablas del sistema de usuarios para utilizar el panel de 14](#_Toc111484429)

[administración. 14](#_Toc111484430)

[18) Crear un usuario para poder ingresar al Panel de Administración 14](#_Toc111484431)

[19) Que es un modelo en Django 14](#_Toc111484432)

[20)Crear un modelo en Django. 15](#_Toc111484433)

[21)Migrar el Modelo a la base del Panel de Administración. 15](#_Toc111484434)

[22)Integrar el Modelo al Panel de Administración. 16](#_Toc111484435)

[23)Ingresar información al modelo por el Panel de Administración. 16](#_Toc111484436)

[24)Realizar la consulta de todo lo ingresado en el modelo desde el 16](#_Toc111484437)

[views. 16](#_Toc111484438)

[25)Mostrar los datos guardados en el modelo al html hijo. 16](#_Toc111484439)

# **Manual de Django**

## **1) ¿Qué es Django?**

**Django es un framework web diseñado para realizar aplicaciones de cualquier complejidad**en unos tiempos muy razonables. Un framework web es un conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente. Está escrito en Python y tiene una comunidad muy amplia, que está en continuo crecimiento.

## **2) ¿Qué es la máquina virtual en Django?**

Un entorno virtual nos permitirá instalar todos los paquetes y librerías necesarias para tu proyecto de manera independiente, es decir, podremos probar distintas versiones de Django sin entrar en conflicto con otros proyectos. Piensa en un entorno virtual como una máquina virtual, capaz de aislar todos los paquetes instalados.

## **3) ¿Qué es MVT en Django?**

Las siglas en ingles MTV corresponden a Modelo (Modelo)*,*Template (Plantilla)*y*View (Vista).  es un patrón de diseño de software. Es una colección de tres componentes importantes Vista de modelo y Plantilla. El modelo ayuda a manejar la base de datos.

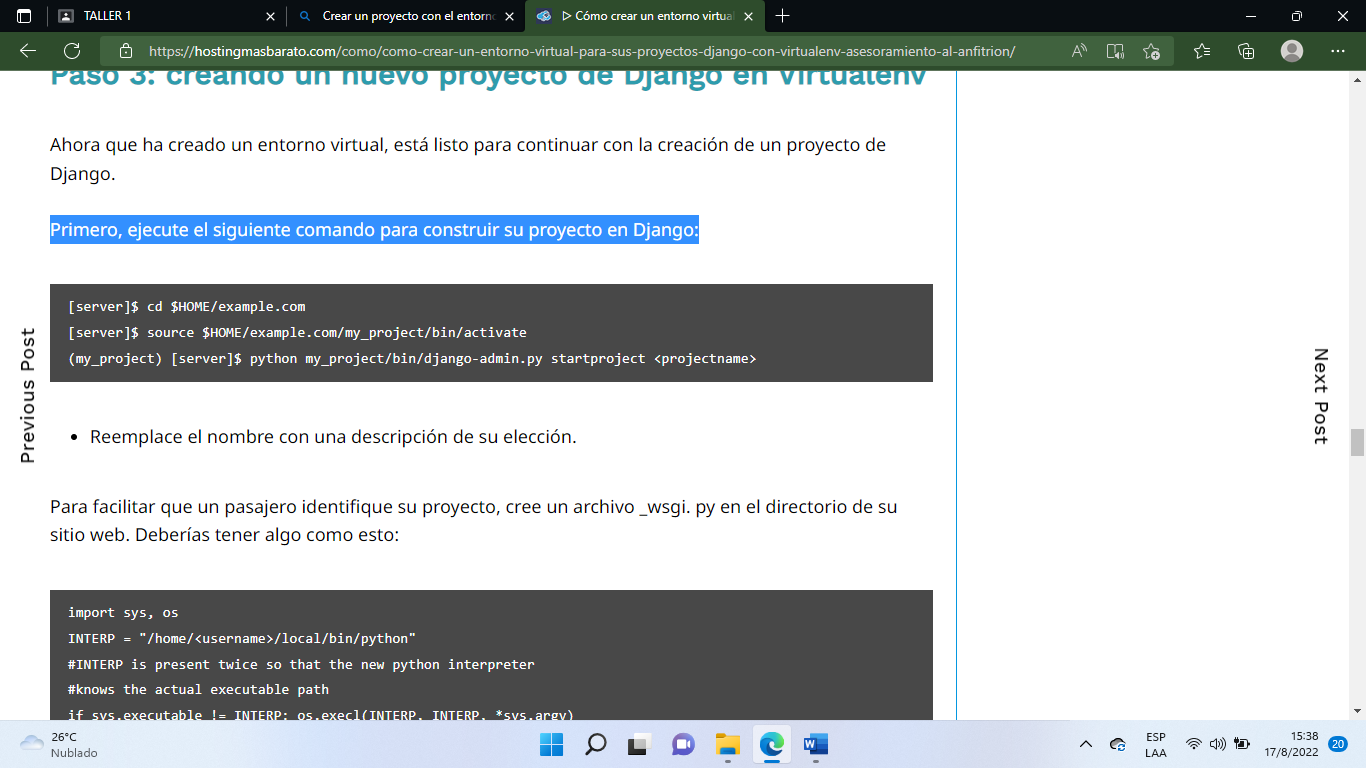
**Modelo**: Maneja todo lo relacionado con la información, esto incluye como acceder a esta, la validación, relación entre los datos y su comportamiento.

**Vista**: Es un enlace entre el modelo y el template. Decide qué información será mostrada y por cual *template.*

**Template**: Decide como será mostrada la información.

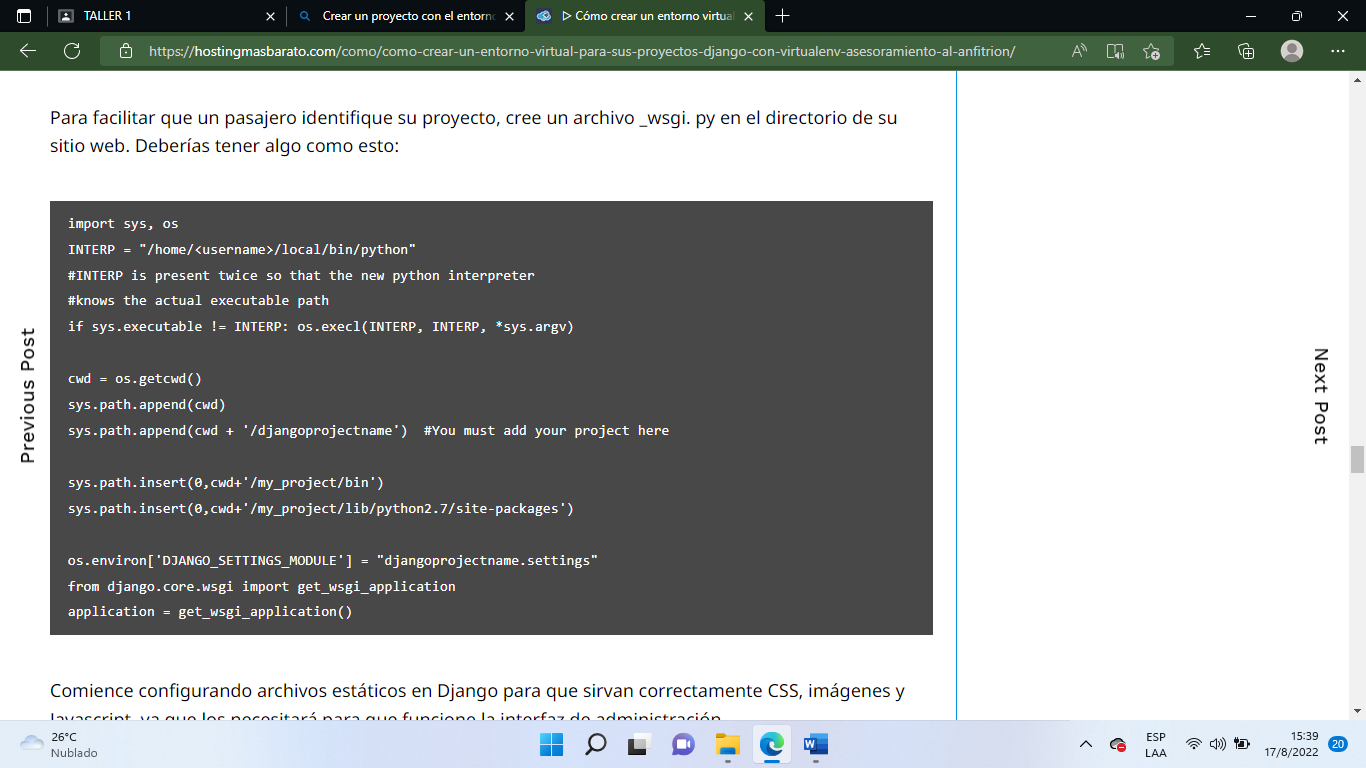
**4) Crear un proyecto con la máquina virtual.**

Primero, ejecute el siguiente comando para construir su proyecto en Django:



* Reemplace el nombre con una descripción de su elección.

Para facilitar que un pasajero identifique su proyecto, cree un archivo \_wsgi. py en el directorio de su sitio web. Deberías tener algo como esto:



Comience configurando archivos estáticos en Django para que sirvan correctamente CSS, imágenes y Javascript, ya que los necesitará para que funcione la interfaz de administración.

* Vaya al archivo .py del proyecto, ubicado en la siguiente ubicación:

example.com/projectname/settings.py

Luego, desplácese hacia abajo hasta donde pueda encontrar Static\_URL, debe configurarse en / static /.

Agregue otra línea para establecer la ubicación del directorio estático de la siguiente manera:

STATIC\_ROOT = os.path.dirname(BASE\_DIR) + '/static/'

En su directorio público, asegúrese de crear el siguiente directorio / static donde Django almacenará todos sus archivos estáticos:

(my\_project) [server]$ cd $HOME/example.com/public

(my\_project) [server]$ mkdir static

Ejecute el comando collectstatic para preparar los archivos estáticos para la interfaz de administración.

(my\_project) [server]$ cd $HOME/example.com/projectname/

(my\_project) [server]$ python manage.py collectstatic

Configure su base de datos en el archivo setting.py de la siguiente manera:

DATABASES = {

'default': {

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

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

Edite la información para que coincida con sus credenciales reales.

DATABASES = {

'default': {

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

'NAME': 'mydatabase',

'USER': 'mydatabaseuser',

'PASSWORD': 'mypassword',

'HOST': 'mysql.example.com',

'PORT': '3306',

}

}

En el archivo settings.py, actualice el campo etiquetado ALLOWED-HOSTS con su nombre de la siguiente manera:

ALLOWED\_HOSTS = ['example.com' , 'www.example.com']

Asegúrese de que la información esté en el directorio de su proyecto Django:

(my\_project) [server]$ cd ~/example.com/<projectname>

Una vez que la configuración esté completa, ejecute migrate en su directorio:

(my\_project) [server]$ python manage.py migrate

Operations to perform:

Apply all migrations: admin, auth, contenttypes, sessions

Running migrations:

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Luego crea tu superusuario:

(my\_project) [server]$ python manage.py createsuperuser

Username (leave blank to use 'username'): my\_django\_user

Email address: email@example.com

Password:

Password (again):

Superuser created successfully.

Vaya al directorio y agregue el siguiente texto; Archivo /tmp/restart.txt:

(my\_project) [server]$ cd /home/username/example.com

(my\_project) [server]$ mkdir tmp

(my\_project) [server]$ cd tmp

(my\_project) [server]$ touch restart.txt

Después de realizar estos cambios, asegúrese de ejecutar este comando desde el directorio de su proyecto:

(my\_project) [server]$ touch tmp/restart.txt

Confirme que el proyecto Django se haya instalado correctamente en su sitio:

Ahora debería poder acceder a la página de administración de Django

# **5) Descargar los instaladores de Django al proyecto**

# **6) Crear un proyecto para programar en Django**

Para crear un nuevo proyecto de django con la estructura de carpetas adecuada, simplemente usamos django-admin.py com el comando startproject.

* *(mientorno)$****django-admin.py startproject miproyecto***

Dentro del proyecto de Django creado, está el script manage.py, que nos permitirá realizar distintas acciones sobre nuestro proyecto, como por ejemplo lanzar el servidor de desarrollo o sincronizar la base de datos. Por comodidad cambiaremos su permiso para que sea ejecutable.

* *(mientorno)$ cd miproyecto*
* *(mientorno)$ chmod +x manage.py*

Esto nos permite ejecutar directamente el archivo manage.py, en lugar de tener que invocarlo con python (es decir, en lugar de $ python manage.py args, ahora podremos hacer directamente $ ./manage.py args)

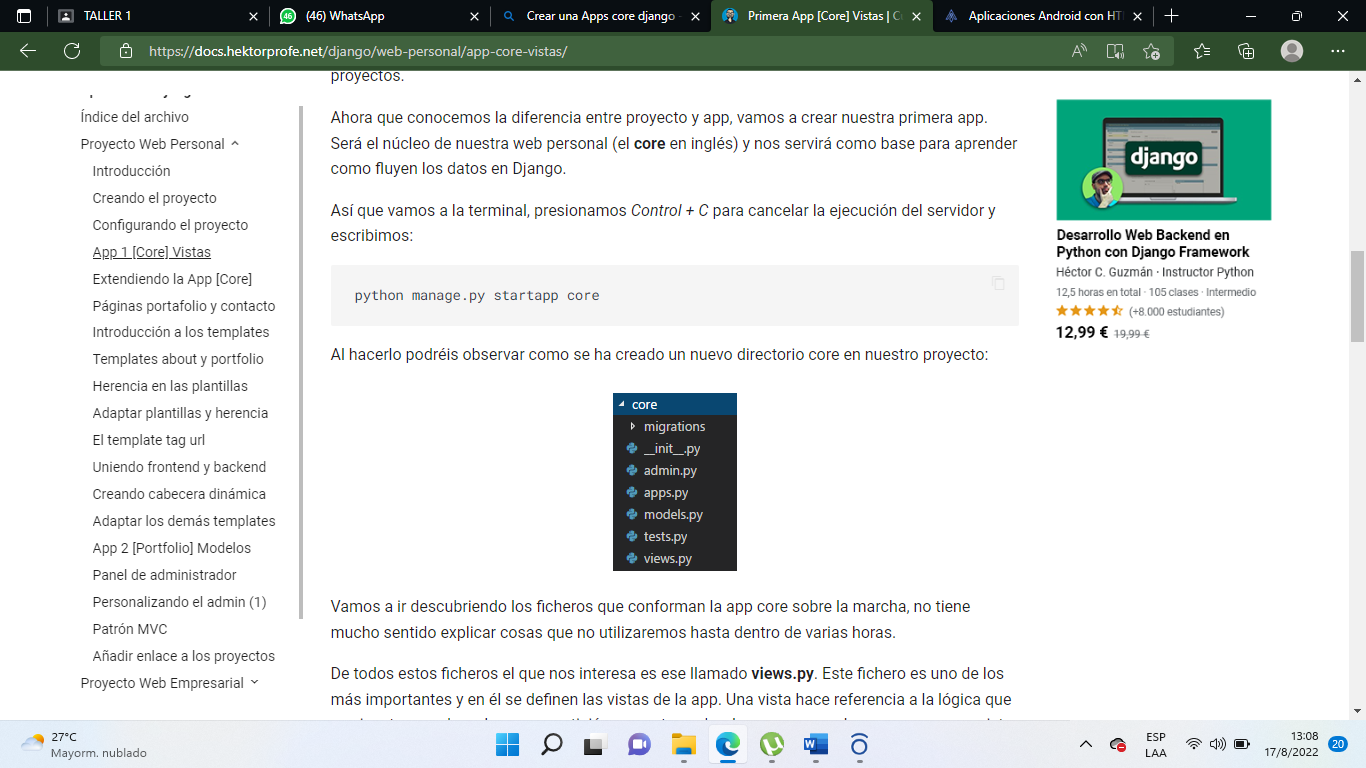
Veamos como funciona:

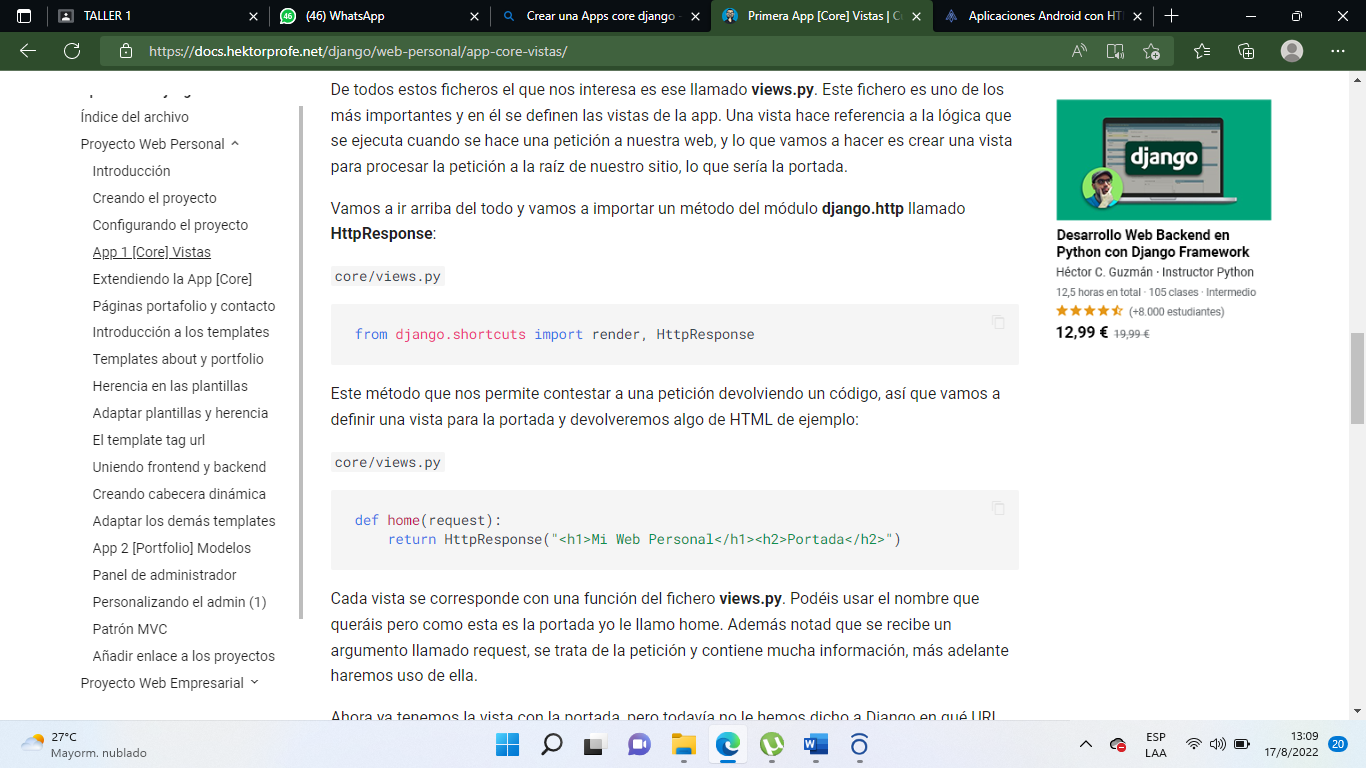
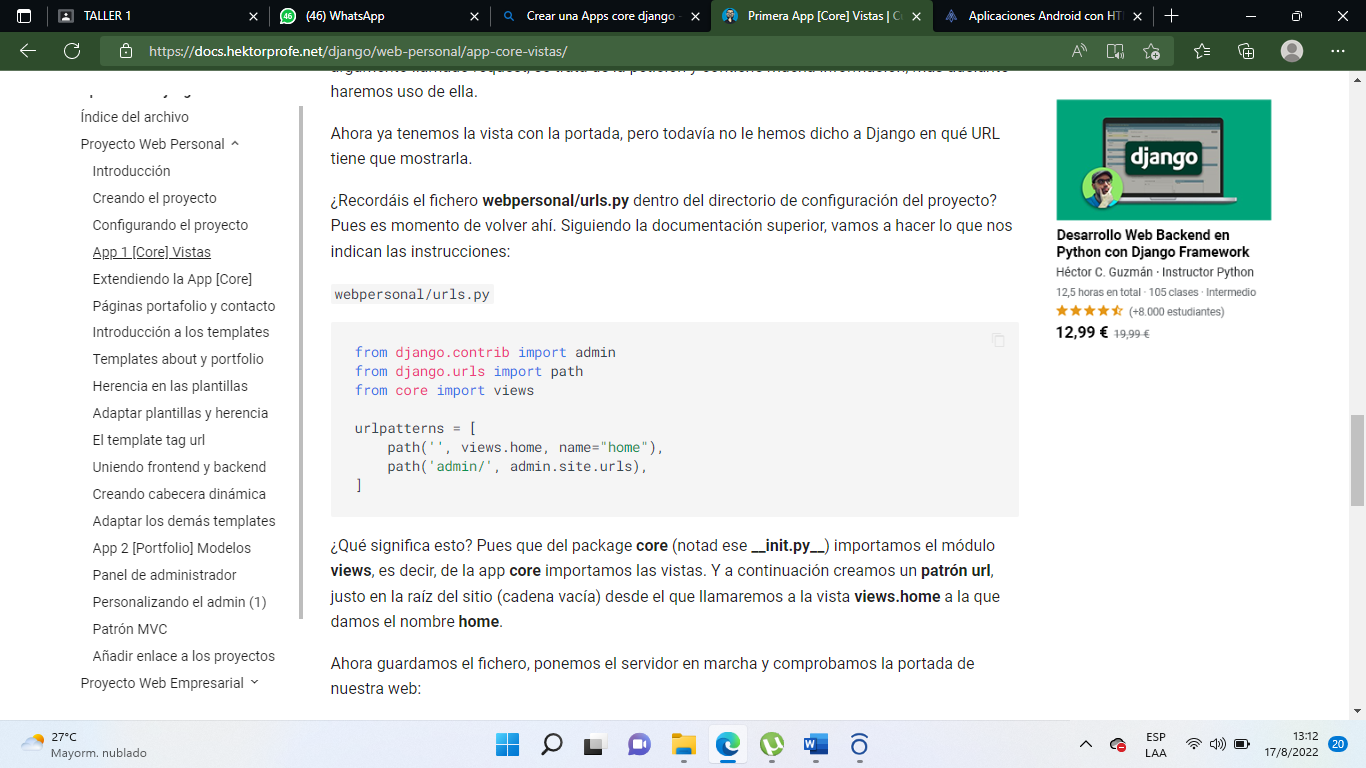
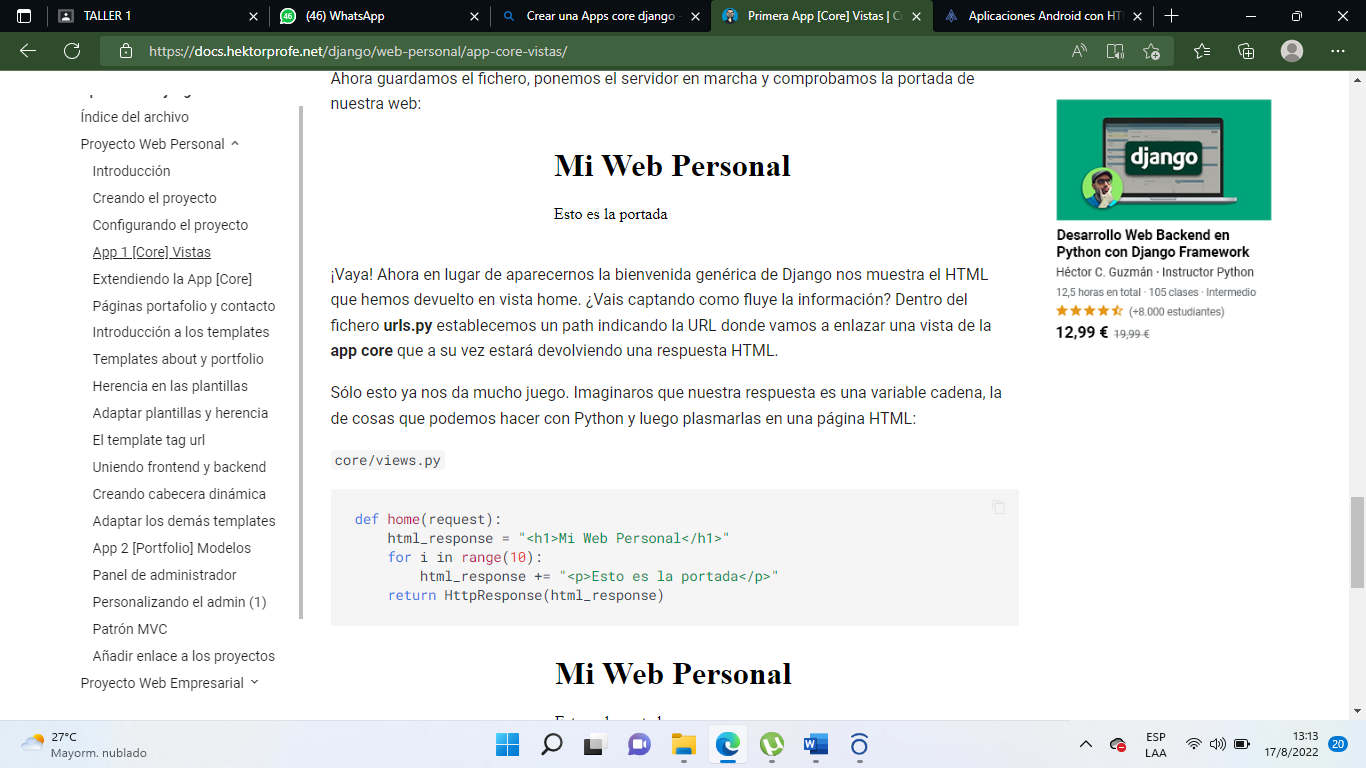
* *(mientorno)$ ./manage.py runserver*

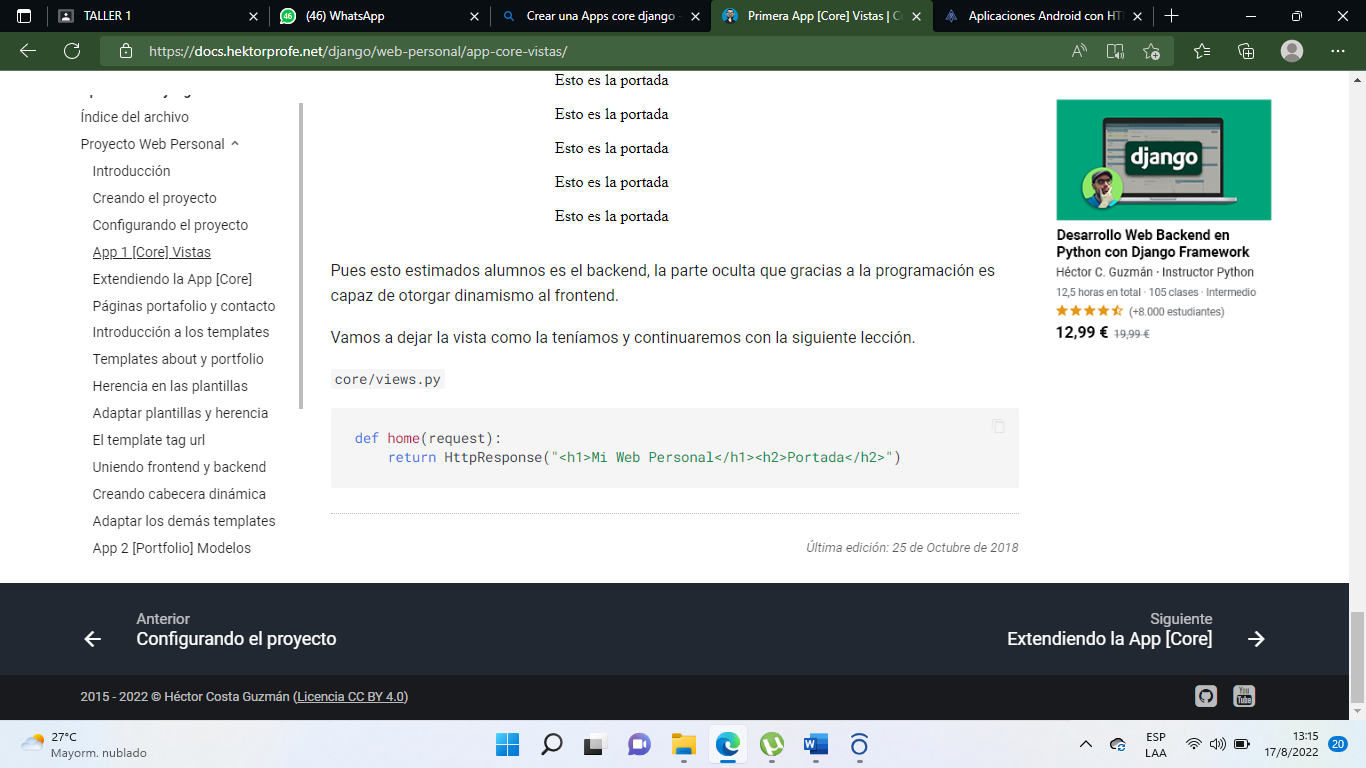
Con esto habremos lanzado el servidor de desarrollo (para producción utilizaremos un entorno más sofisticado), que es suficiente para comprobar el funcionamiento de nuestra aplicación mientras trabajamos en ella

# **7) Ejecutar el proyecto y el mensaje de felicitaciones.**

# **8) Crear una Apps core.**







# **9) ¿Qué es la Carpeta Templates?**

En nuestro proyecto de Django debemos crear una nueva carpeta llamada templates dentro de cada aplicación, y luego en la carpeta templates debemos crear una nueva carpeta con el nombre de la aplicación para que este sea el namespace. Los Templates son plantillas que reciben datos y generan código HTML. Para los Templates debemos manejar una sintaxis un poco diferente.

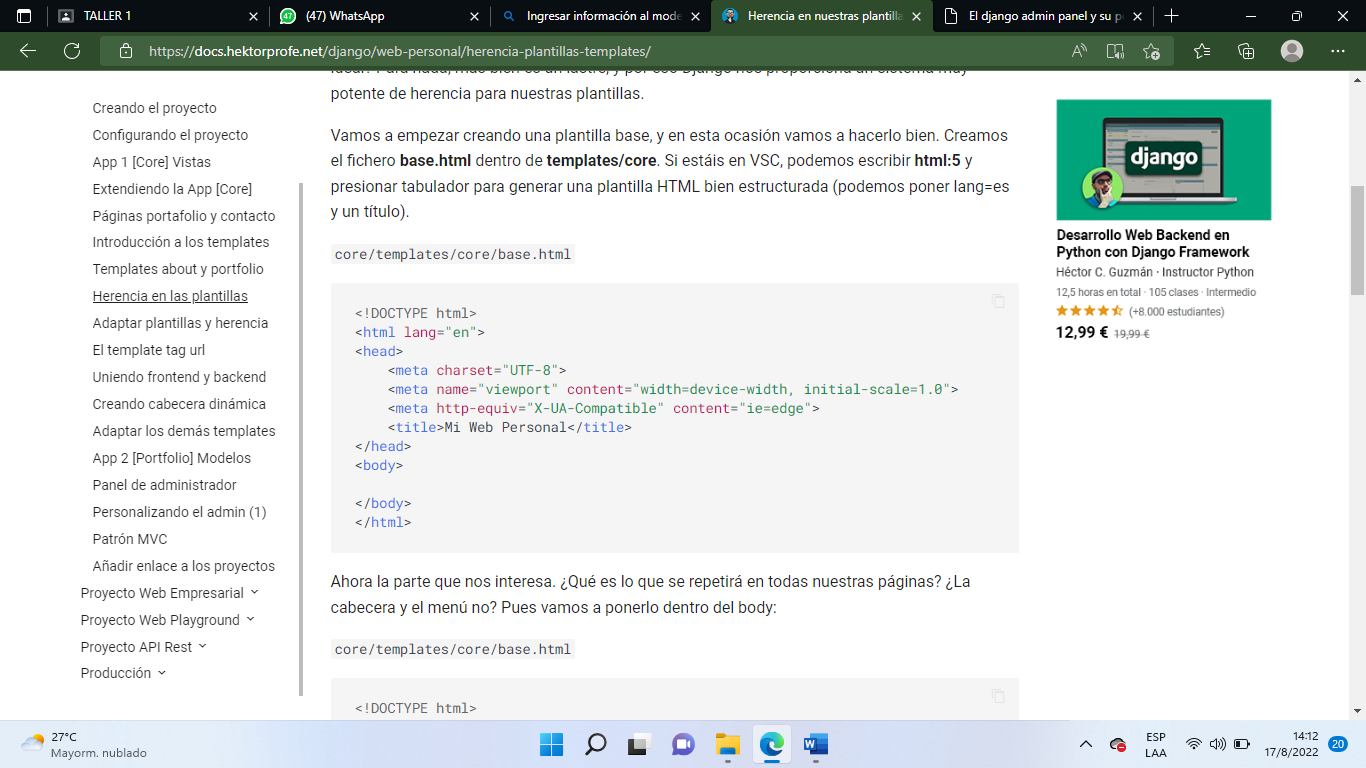
# **10) ¿Qué es la Carpeta stactic?**

Cada aplicación Django tendrá una carpeta static y una carpeta plantillas. Dentro de estas carpetas, habrá otra carpeta con el nombre de la aplicación. Dentro de estas carpetas, colocaremos todos los archivos y plantillas estáticos.

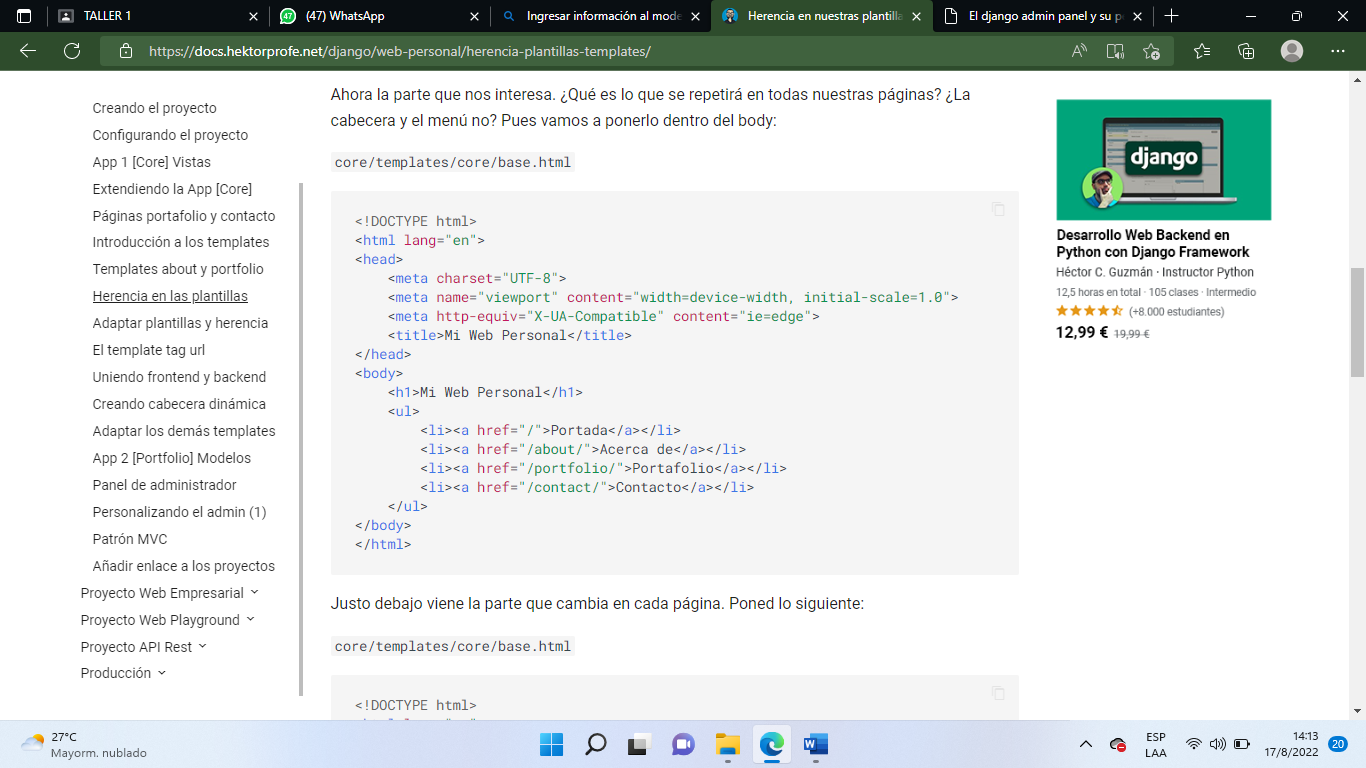
Este enfoque nos permite tener varios archivos con el mismo nombre. Por ejemplo, cada aplicación puede tener un archivo index.html y un style.css, y será más fácil acceder a ellos.

# **11) Crear un archivo base HTML en la APPS core**

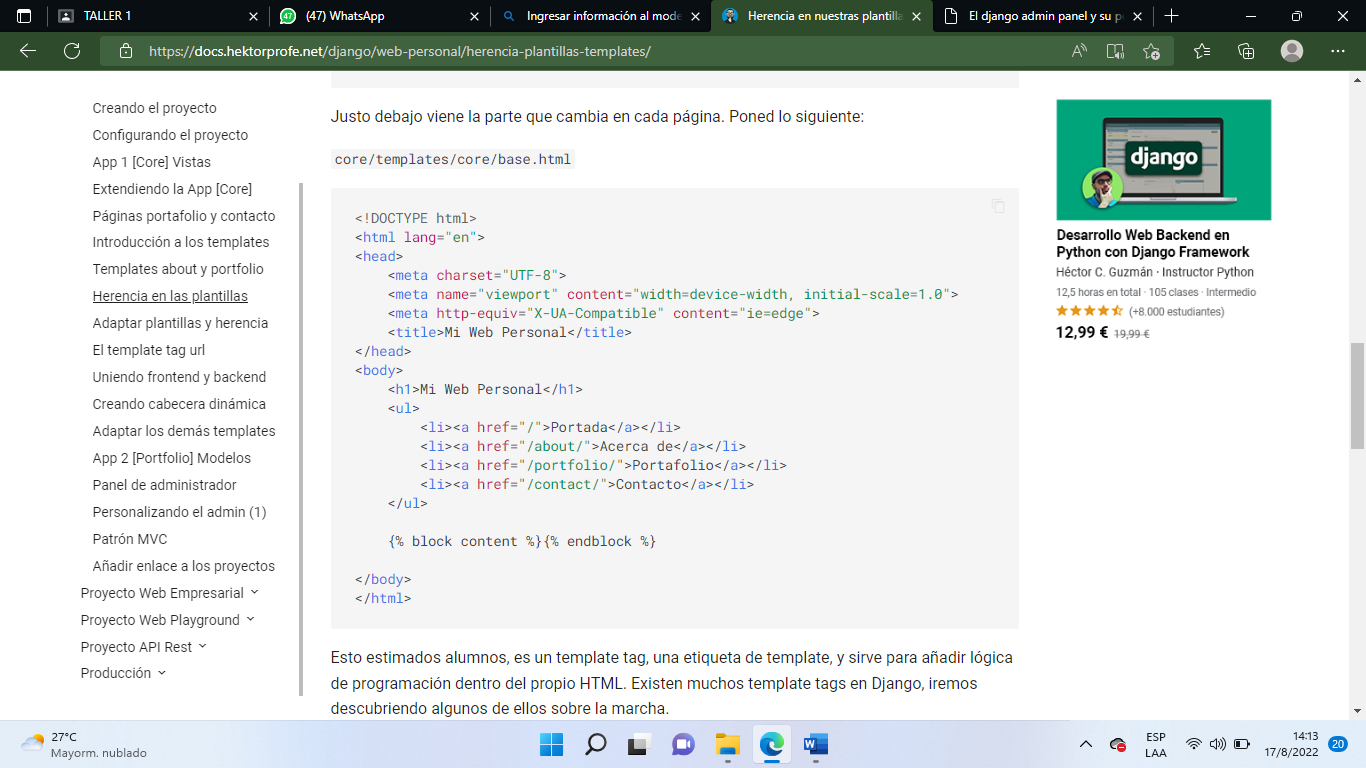
Vamos a empezar creando una plantilla base, y en esta ocasión vamos a hacerlo bien. Creamos el fichero **base.html** dentro de **templates/core**. Si estáis en VSC, podemos escribir **html:5** y presionar tabulador para generar una plantilla HTML bien estructurada (podemos poner lang=es y un título).



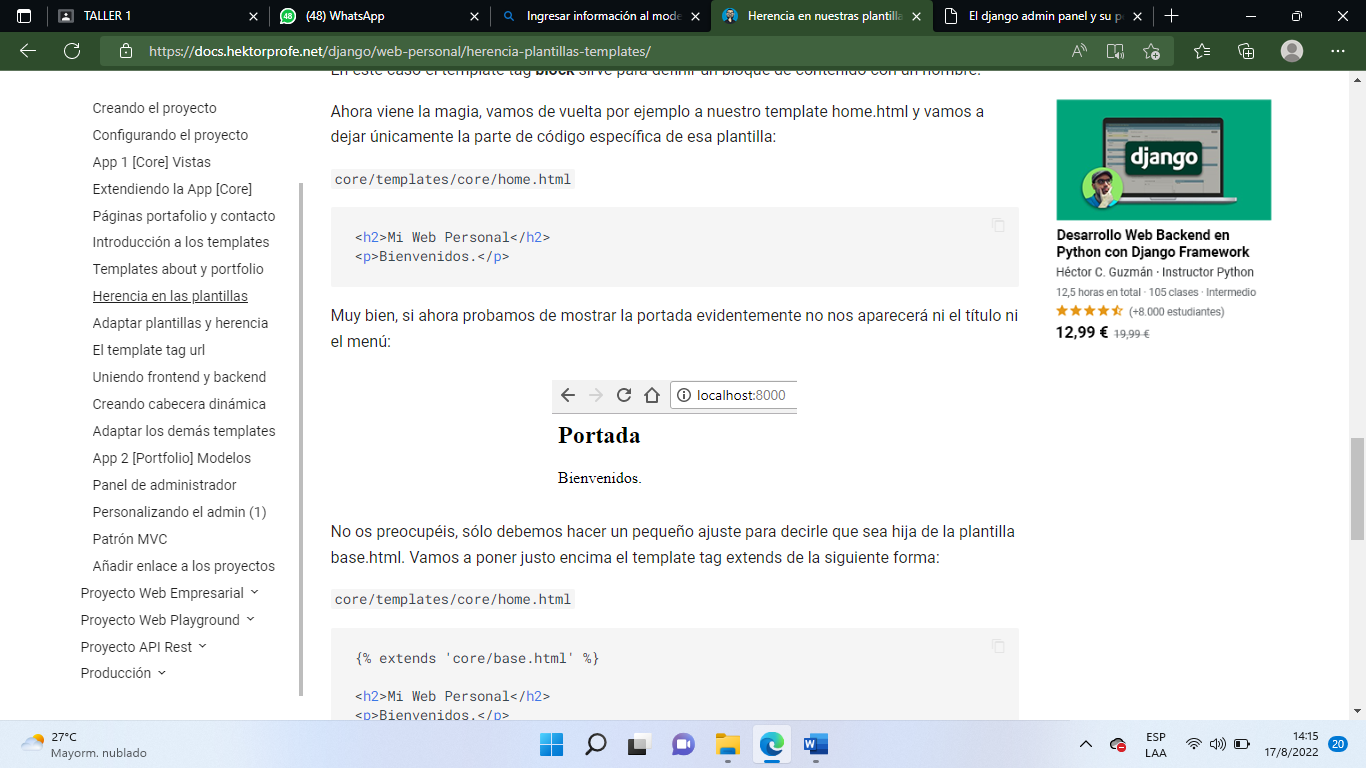
La cabecera y el menú . Pues vamos a ponerlo dentro del body:



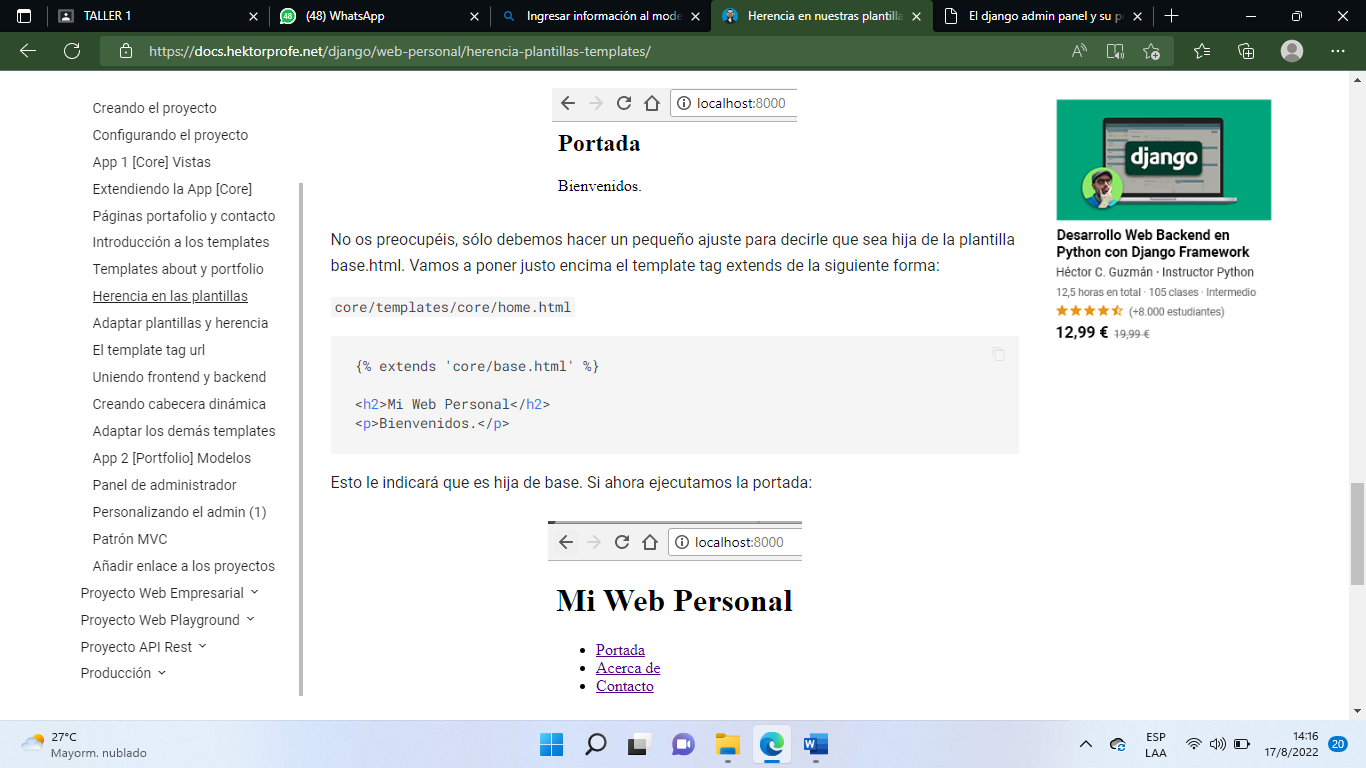
Justo debajo viene la parte que cambia en cada página. Poned lo siguiente:



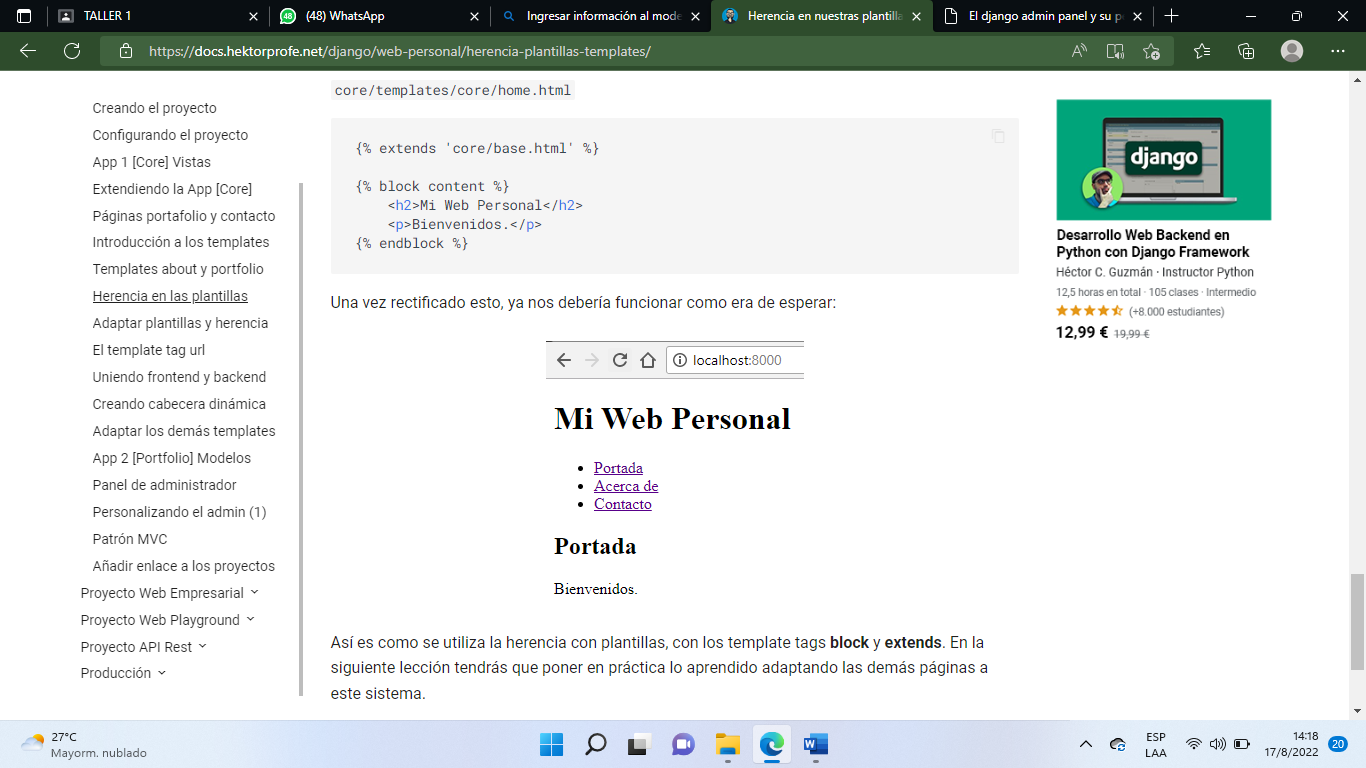
vamos de vuelta por ejemplo a nuestro template home.html y vamos a dejar únicamente la parte de código específica de esa plantilla



Sólo debemos hacer un pequeño ajuste para decirle que sea hija de la plantilla base.html. Vamos a poner justo encima el template tag extends de la siguiente forma:



Sólo se muestra el título y el menú. El caso es que no está funcionando como debería, pero vamos bien. Nuestra portada se dibuje justo dentro del bloque content de la base, y eso lo haremos creando de nuevo el bloque content y poniendo nuestro HTML dentro:



# **12) Como se llaman a los CSS desde el archivo base html**.

Como HTML, CSS (*Cascading Style Sheets*) u Hojas de estilo en cascada en español, no es realmente un lenguaje de programación, tampoco es un lenguaje de marcado. Es un *lenguaje de hojas de estilo,* es decir, te permite aplicar estilos de manera selectiva a elementos en documentos HTML.

Distintas formas de llamar a un archivo .css.  
1- Colocando el siguiente código entre las etiquetas <head> y </head>

<link rel=»StyleSheet» href=»estilo.css» type=»text/css»>

2- Otra forma de usar css es metiendo el código dentro del mismo archivo .html pero el código css dentro de las etiquetas <STYLE> y </STYLE>, estas etiquetas también van ubicadas dentro de las cabeceras HEAD ejemplo:

<style>  
#centro{ margin: auto;  width: 200px;  }  
</style>

3- Una tercera forma de llamar a un .css es con la sentencia @import que nos ofrece CSS

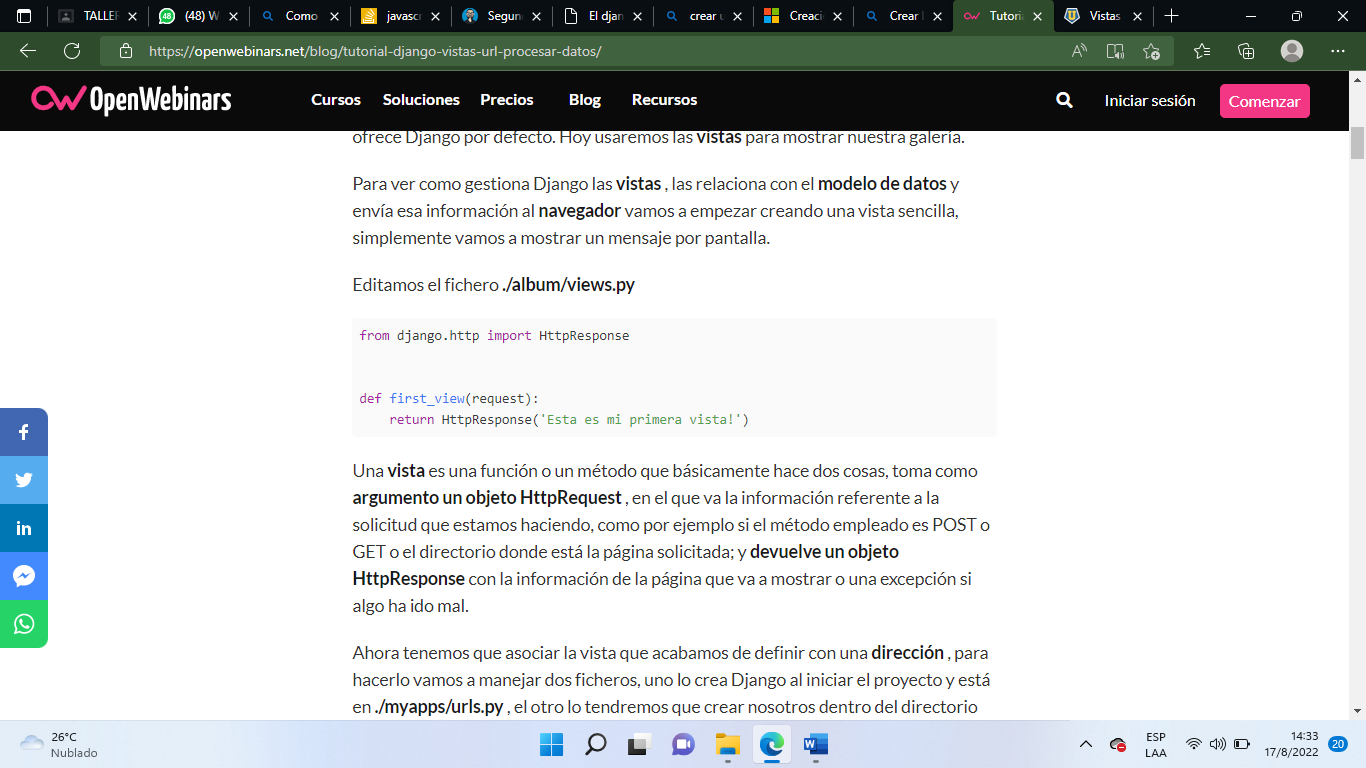
<STYLE>  
@import url(estilos/index.css);  
</STYLE>

**13) Como consume un archivo hijo html al utilizar la herencia del** **archivo base html.**

**14) Crear un view que llame al html hijo**

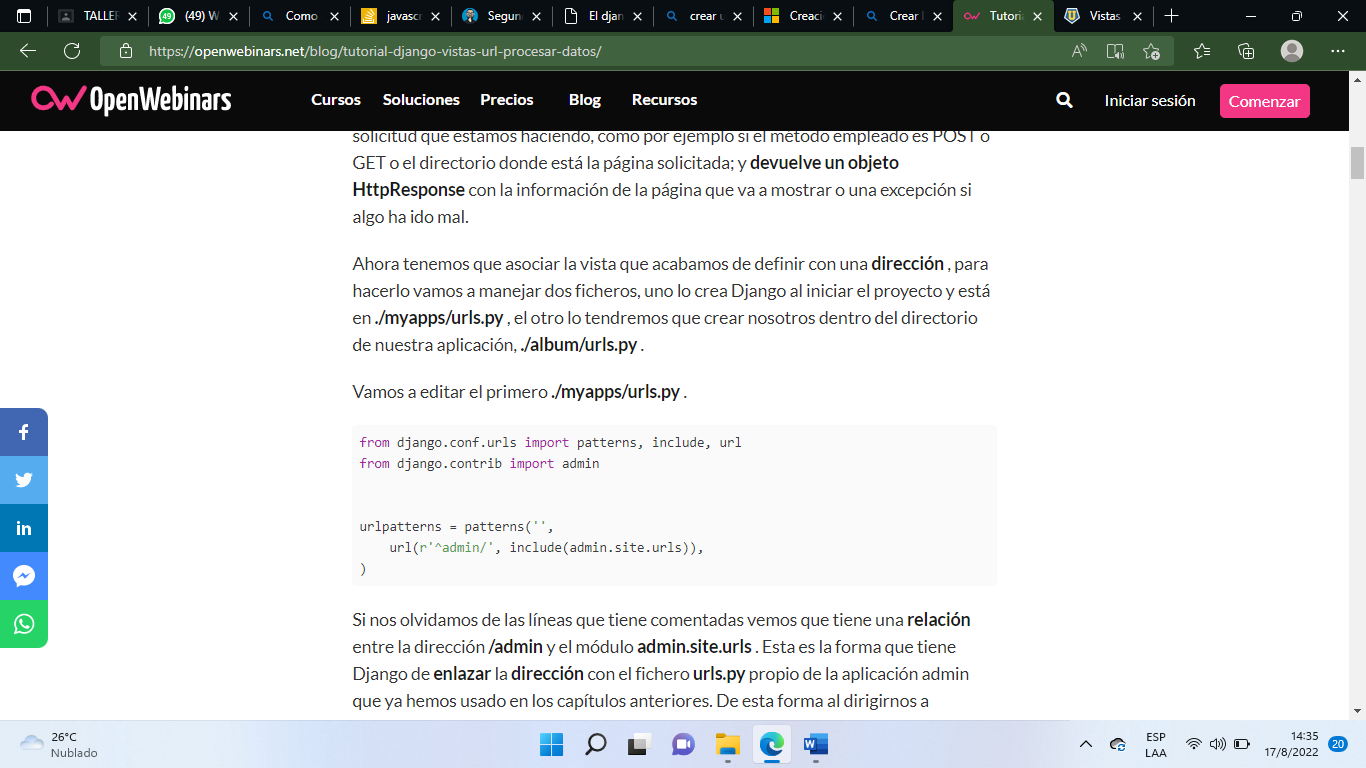
**15) Crear la urls que llame al views.**

Editamos el fichero ./album/views.py



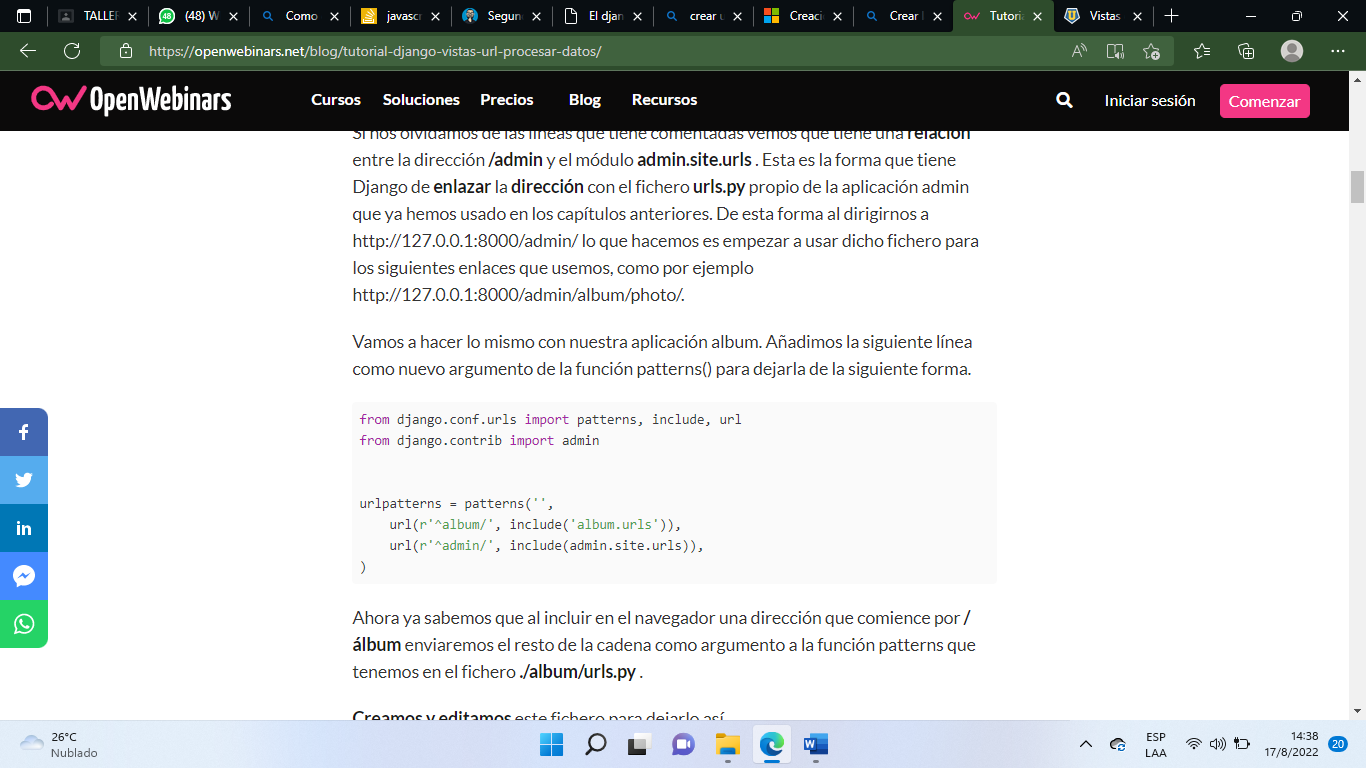
Ahora tenemos que asociar la vista que acabamos de definir con una **dirección,** para hacerlo vamos a manejar dos ficheros, uno lo crea Django al iniciar el proyecto y está en **./myapps/urls.py**, el otro lo tendremos que crear nosotros dentro del directorio de nuestra aplicación, **./album/urls.py**.

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



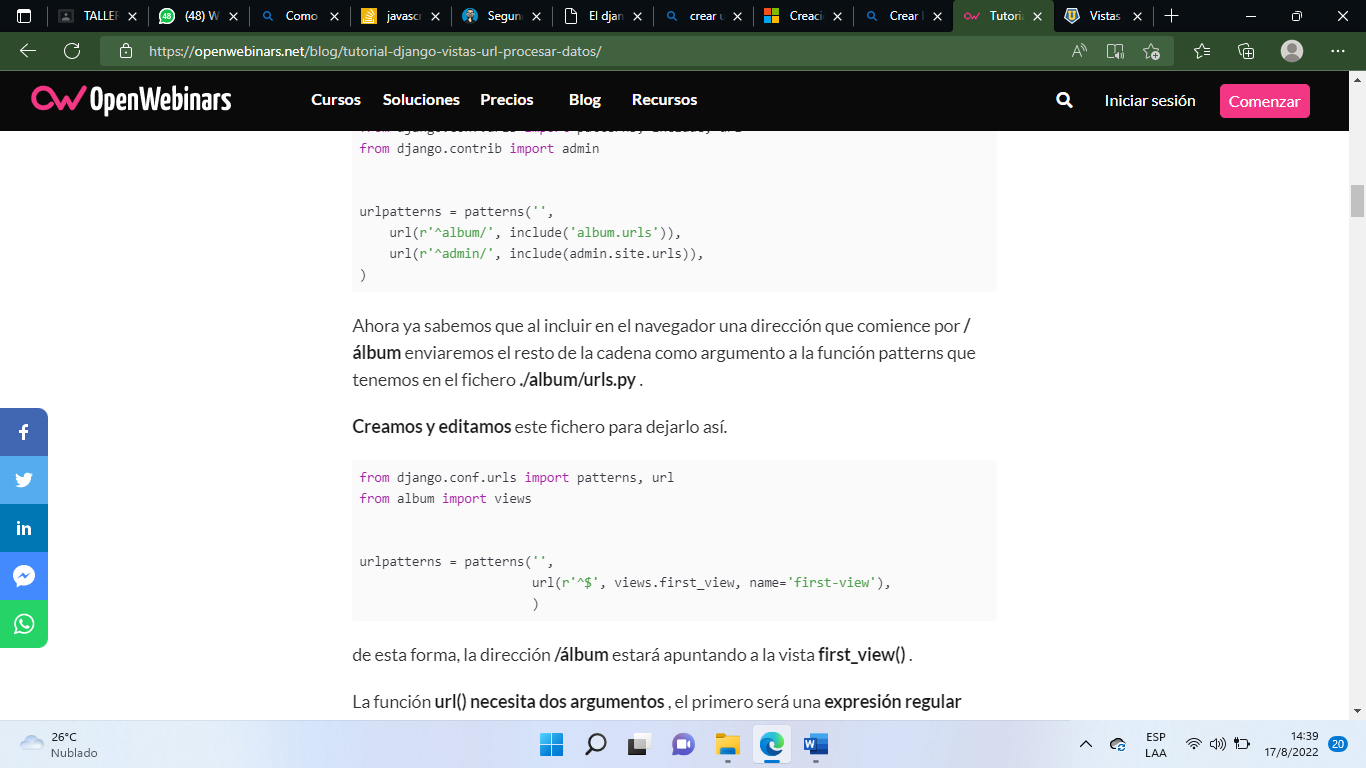
Si nos olvidamos de las líneas que tiene comentadas vemos que tiene una **relación**entre la dirección **/admin**y el módulo **admin.site.urls**. Esta es la forma que tiene Django de **enlazar**la **dirección**con el fichero **urls.py**propio de la aplicación admin que ya hemos usado en los capítulos anteriores. De esta forma al dirigirnos a http://127.0.0.1:8000/admin/ lo que hacemos es empezar a usar dicho fichero para los siguientes enlaces que usemos, como por ejemplo http://127.0.0.1:8000/admin/album/photo/.

Vamos a hacer lo mismo con nuestra aplicación album. Añadimos la siguiente línea como nuevo argumento de la función patterns() para dejarla de la siguiente forma.



Ahora ya sabemos que al incluir en el navegador una dirección que comience por **/álbum**enviaremos el resto de la cadena como argumento a la función patterns que tenemos en el fichero **./album/urls.py**.

Creamos y editamoseste fichero para dejarlo así.



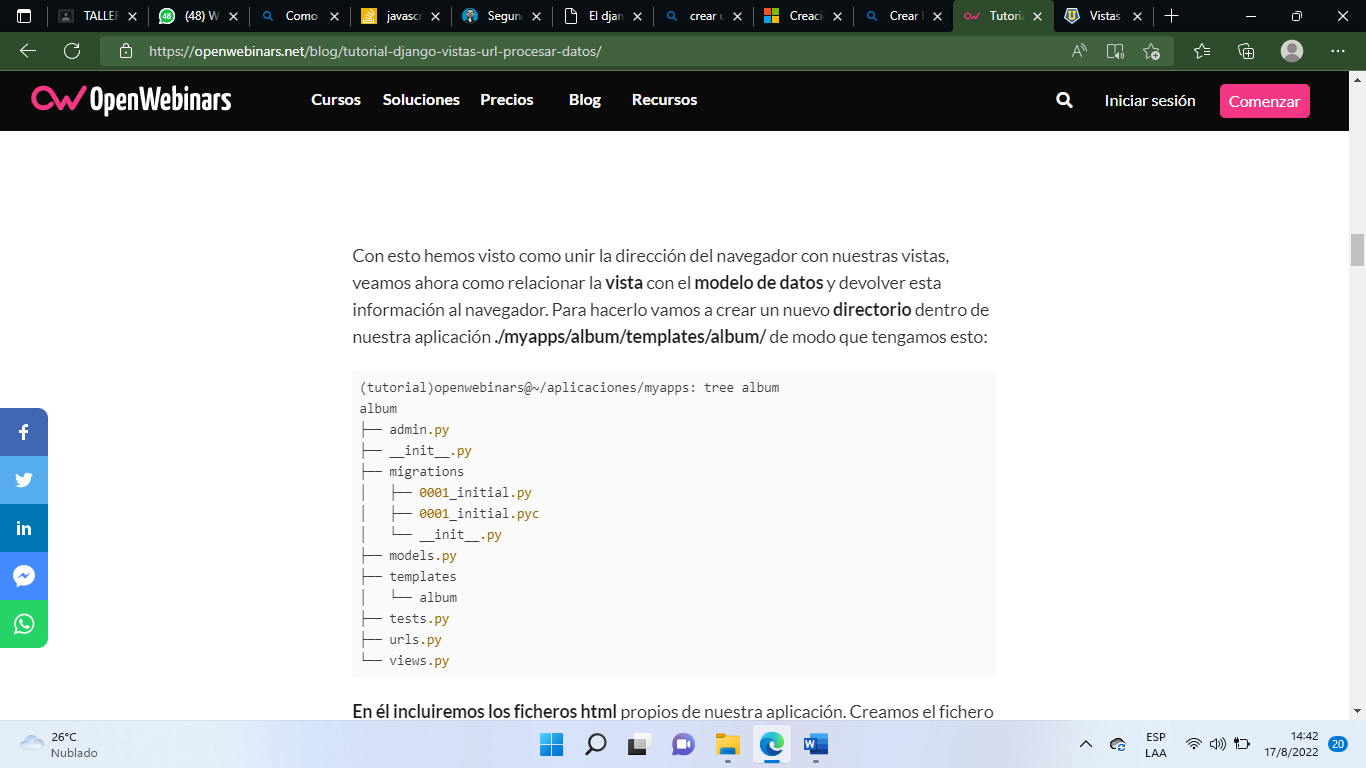
de esta forma, la dirección **/álbum**estará apuntando a la vista **first\_view()**.

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 funcióna 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 únicay 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.

Si arrancamos el servidor de pruebas y buscamos en el navegador la dirección http://127.0.0.1:8000/album/ veremos lo siguiente:



Con esto hemos visto como unir la dirección del navegador con nuestras vistas, veamos ahora como relacionar la vista con el modelo de datos y devolver esta información al navegador. Para hacerlo vamos a crear un nuevo directorio dentro de nuestra aplicación ./myapps/album/templates/album/ de modo que tengamos esto:

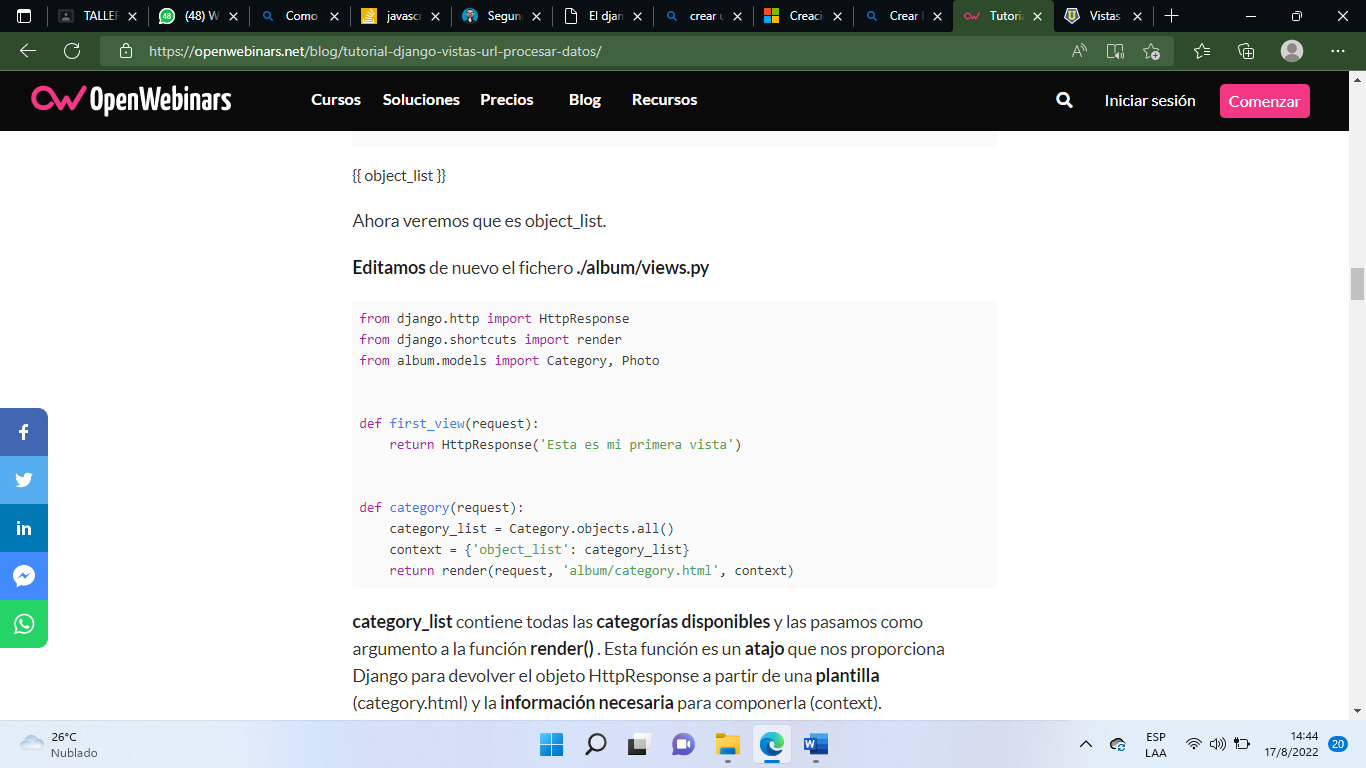


En él incluiremos los ficheros html propios de nuestra aplicación. Creamos el fichero category.html de la siguiente forma:

{{ object\_list }}

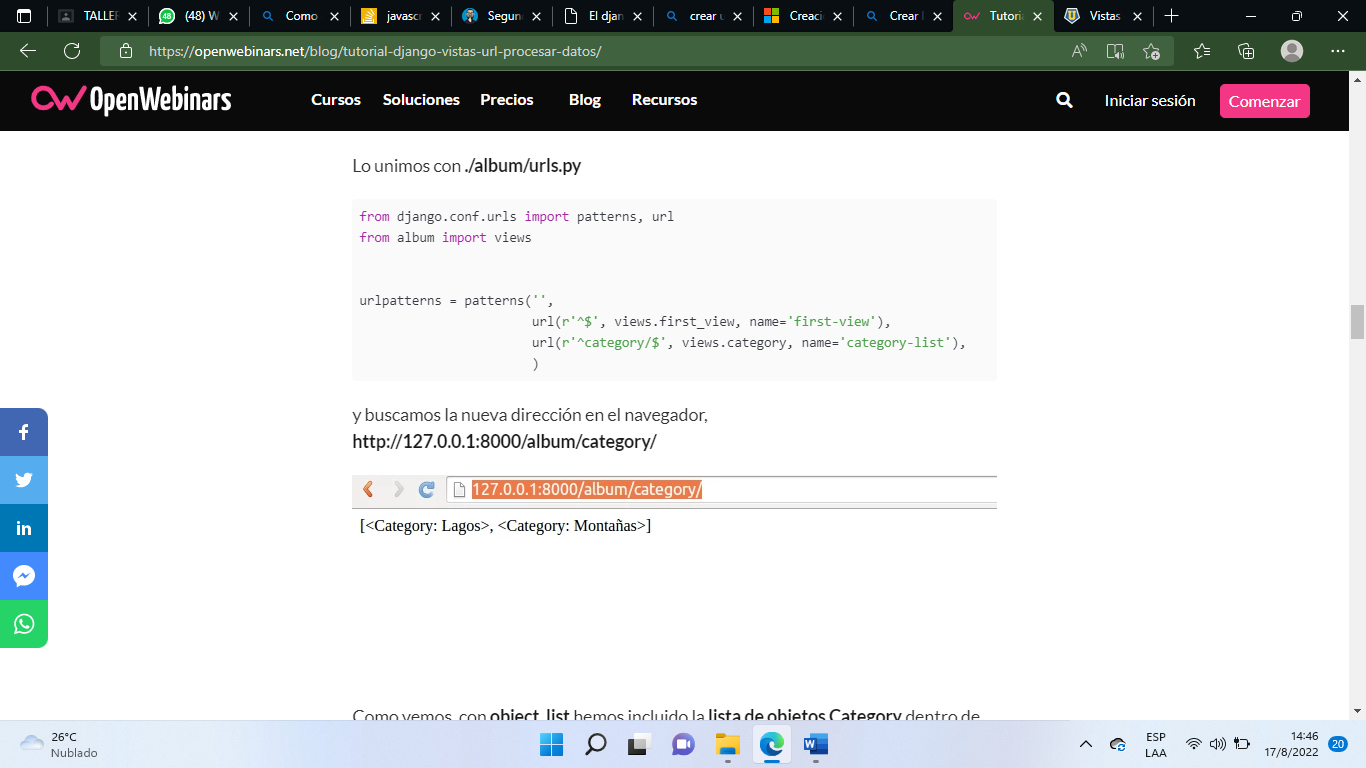
Ahora veremos que es object\_list.

Editamos de nuevo el fichero ./album/views.py



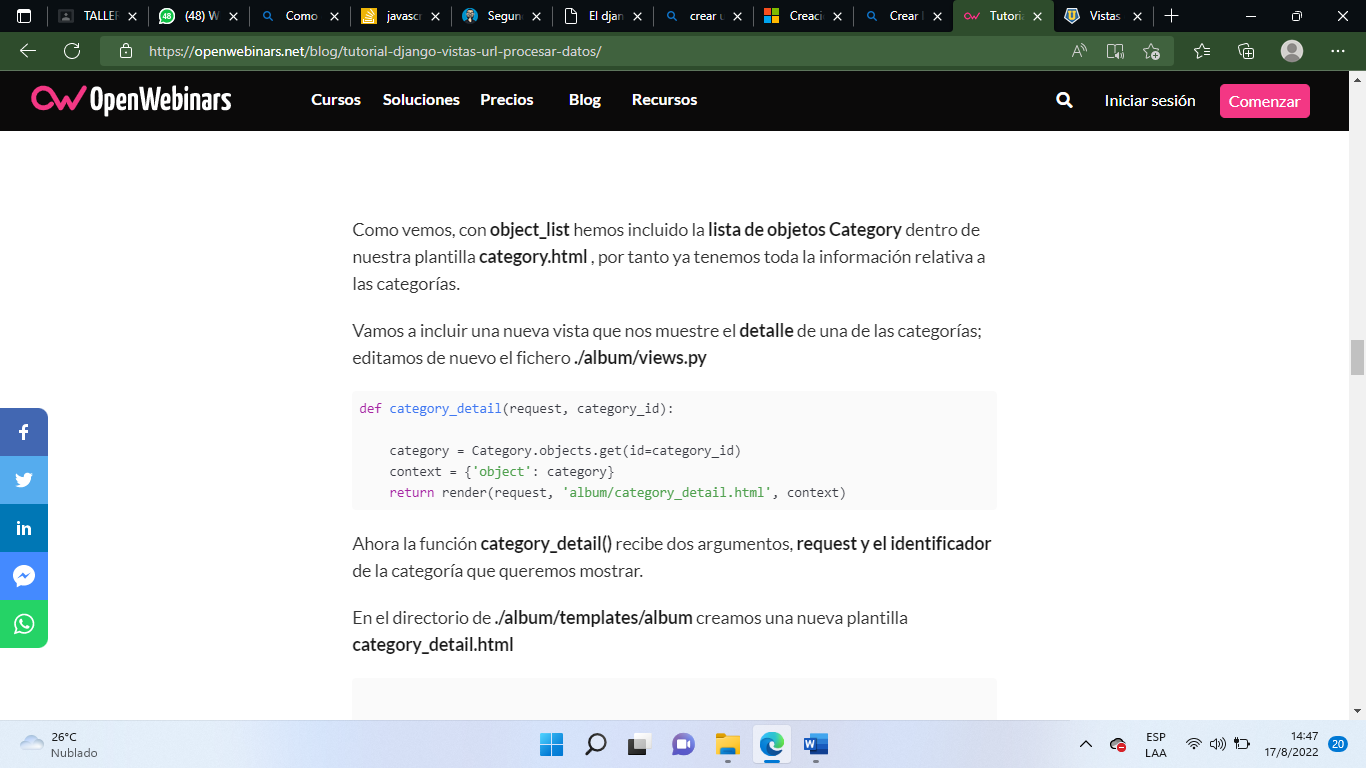
category\_list contiene todas las categorías disponibles y las pasamos como argumento a la función render() . Esta función es un atajo que nos proporciona Django para devolver el objeto HttpResponse a partir de una plantilla (category.html) y la información necesaria para componerla (context).

Lo unimos con **./album/urls.py**



Como vemos, con object\_list hemos incluido la lista de objetos Category dentro de nuestra plantilla category.html , por tanto ya tenemos toda la información relativa a las categorías.

Vamos a incluir una nueva vista que nos muestre el detalle de una de las categorías; editamos de nuevo el fichero ./album/views.py



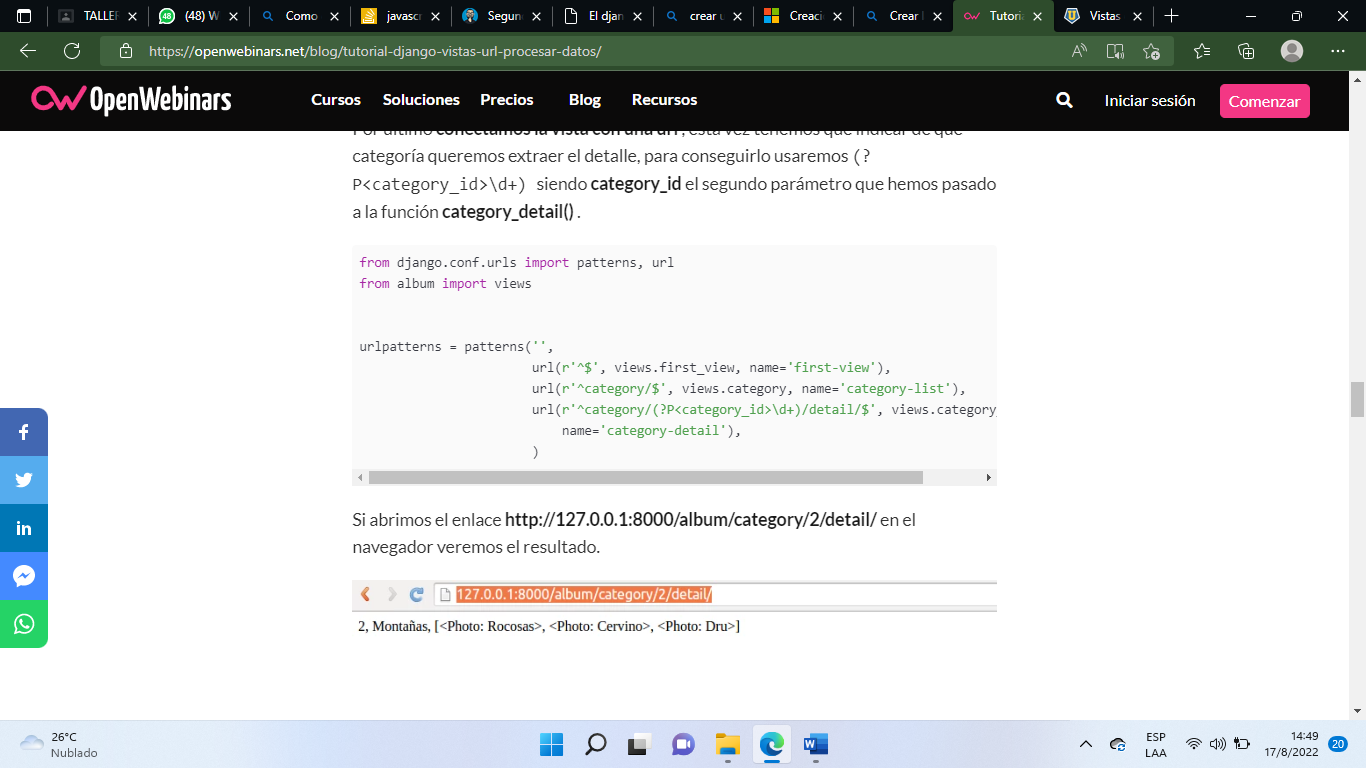
Ahora la función category\_detail() recibe dos argumentos, request y el identificador de la categoría que queremos mostrar.

En el directorio de ./album/templates/album creamos una nueva plantilla category\_detail.html

{{ object.id }}, {{ object.name }}, {{ object.photo\_set.all }}

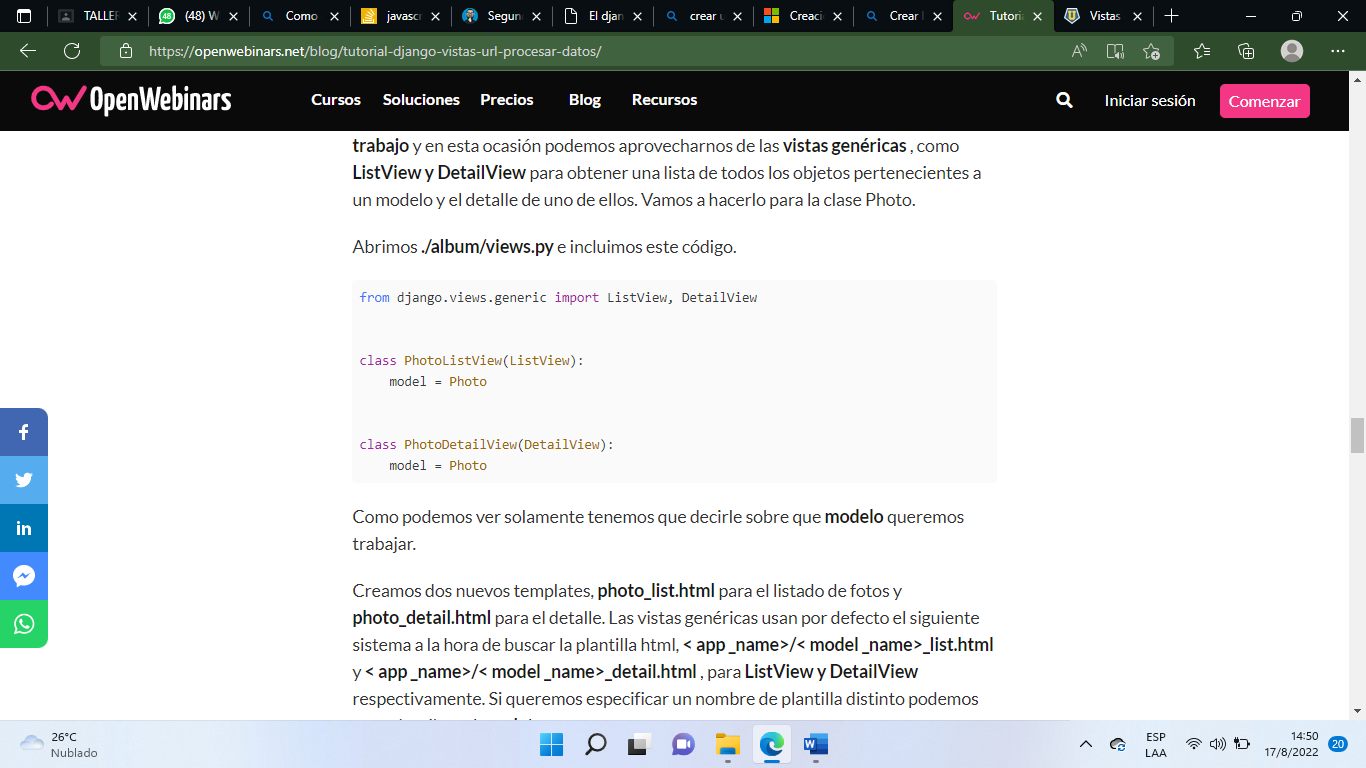
De forma similar a la vista anterior, en la variable context que pasamos a la función render() tenemos un objeto Category del que podemos extraer su id, name y las fotos que pertenecen a dicha categoría entre otras cosas.

Por último conectamos la vista con una url , esta vez tenemos que indicar de que categoría queremos extraer el detalle, para conseguirlo usaremos (?P<category\_id>\d+) siendo category\_id el segundo parámetro que hemos pasado a la función category\_detail() .



Ya vimos en posts anteriores que Django nos ofrece soluciones para facilitarnos el trabajo y en esta ocasión podemos aprovecharnos de las vistas genéricas , como ListView y DetailView para obtener una lista de todos los objetos pertenecientes a un modelo y el detalle de uno de ellos. Vamos a hacerlo para la clase Photo.

Abrimos **./album/views.py**e incluimos este código.



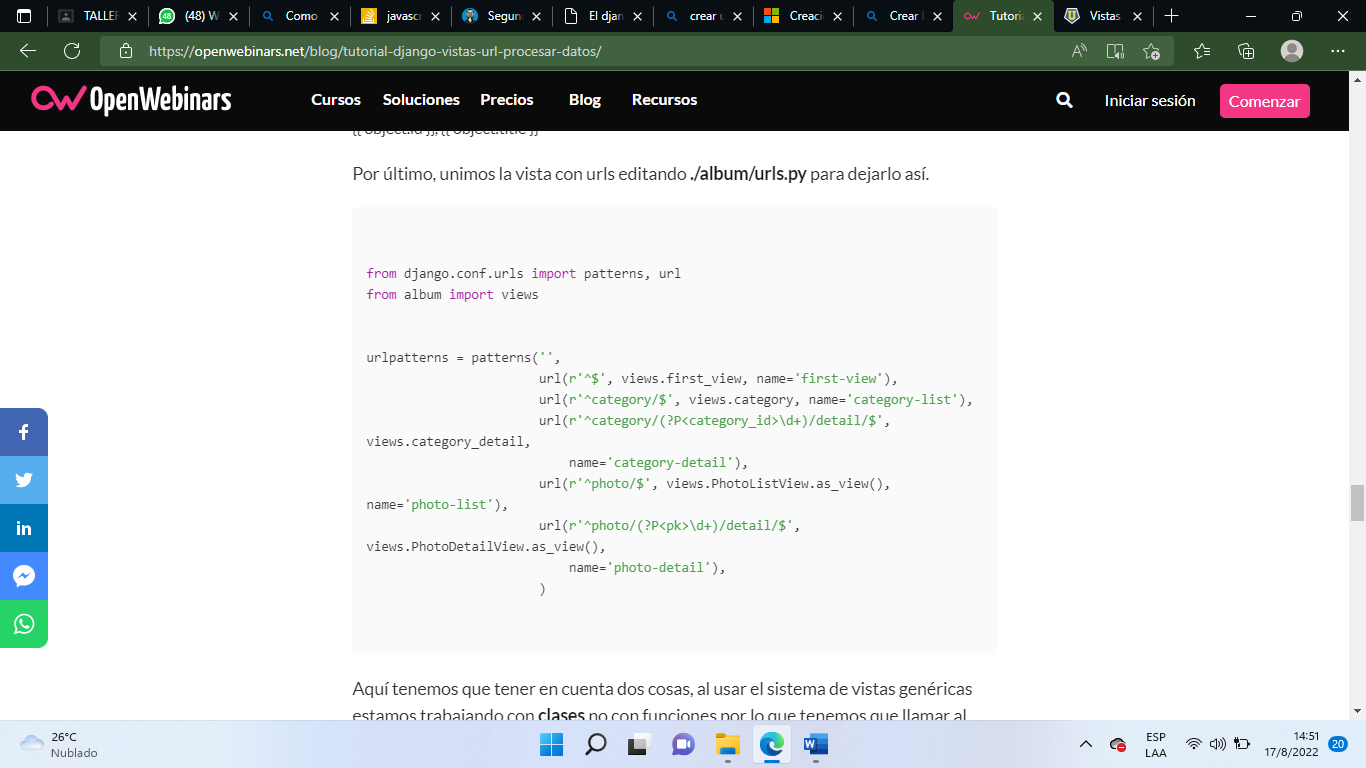
Como podemos ver solamente tenemos que decirle sobre que modelo queremos trabajar.

Creamos dos nuevos templates, photo\_list.html para el listado de fotos y photo\_detail.html para el detalle. Las vistas genéricas usan por defecto el siguiente sistema a la hora de buscar la plantilla html, < app \_name>/< model \_name>\_list.html y < app \_name>/< model \_name>\_detail.html , para ListView y DetailView respectivamente. Si queremos especificar un nombre de plantilla distinto podemos usar el atributo template\_name .

Creamos ./album/templates/album/photo\_list.html

* {{ object\_list }}
* y ./album/templates/album/photo\_detail.html
* {{ object.id }}, {{ object.title }}

Por último, unimos la vista con urls editando ./album/urls.py para dejarlo así.



**16) Integrar la aplicación APPS core al proyecto principal**

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

**18) Crear un usuario para poder ingresar al Panel de Administración**

Para poder ingresar deberás crear un superusuario un usuario que tiene control sobre todo lo que hay en el sitio.

Entra a la consola o la terminal de comandos y escribe:

* python manage.py createsuperuser

Presiona enter y escribe tu nombre de usuario (en minúsculas, sin espacios), dirección de email y contraseña cuando sean requeridos.

No te preocupes que no puedes ver tu contraseña mientras la tipeas - así es como debe ser. Simplemente escribela y presiona 'Enter' para continuar. La salida de este comando debería verse así (nombre de usuario y email deberían ser los tuyos)

El código en consola se verá así

* (myvenv) ~/djangogirls$ python manage.py createsuperuser
* Username: admin
* Email address: admin@admin.com
* Password:
* Password (again):
* Superuser created successfully.

Una vez que hayas creado el superusuario, abres el navegador y digitas la url de tu sitio añadiendo /admin al final de la URL.

Por ejemplo, si la URL es: http://127.0.0.0/

Entonces el administrador está ubicado en: <http://127.0.0.0/admin>.

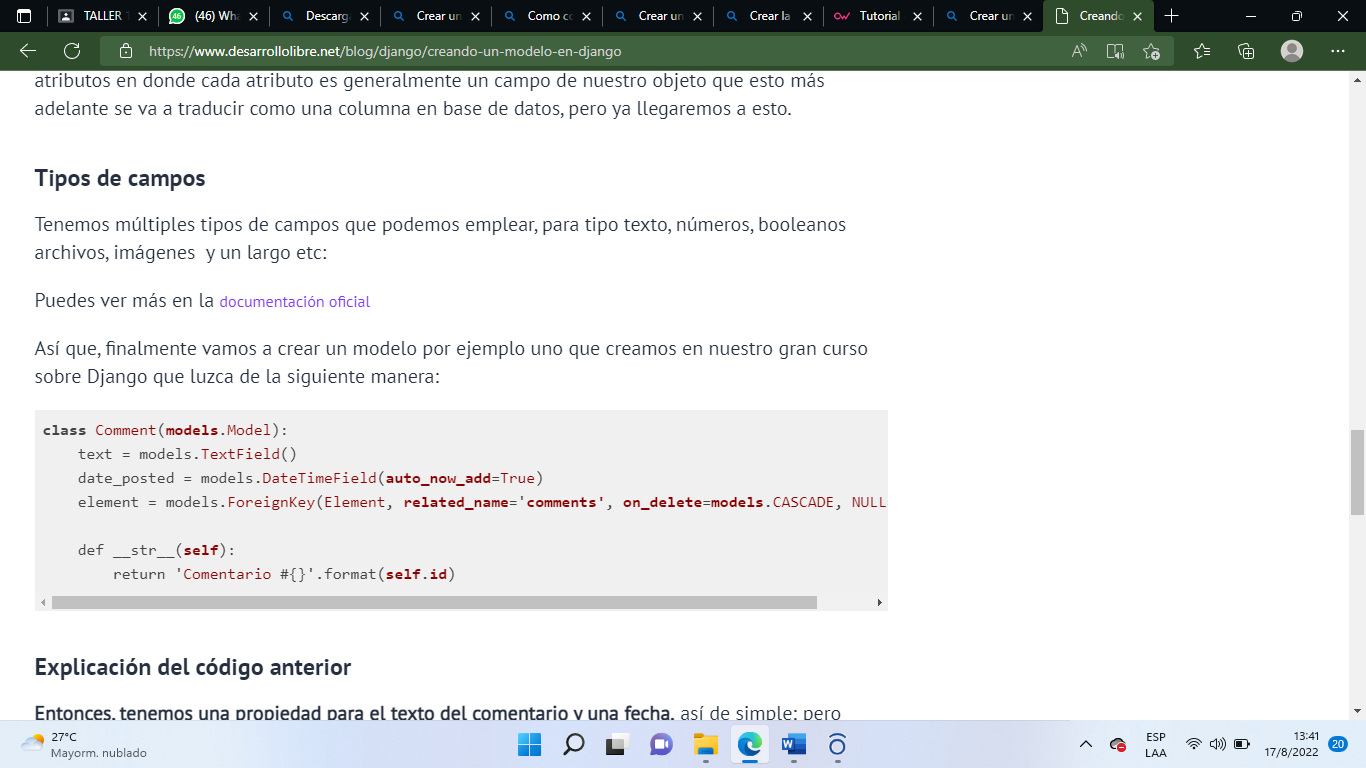
**19) Que es un modelo en Django**

Un modelo de una aplicación Django es un archivo con código Python, generalmente llamado *models.py.*Este archivo representa la estructura de la base de datos de nuestra aplicación. Tomado de la [documentación de Django](https://docs.djangoproject.com/es/1.11/intro/tutorial02/#creating-models):

Un modelo es la fuente única y definitiva de información sobre sus datos. Contiene los campos esenciales y los comportamientos de los datos que usted guarda. Django sigue el [Principio DRY](https://docs.djangoproject.com/es/1.11/misc/design-philosophies/#dry). El objetivo es definir el modelo de datos en un solo lugar y derivar cosas de este automáticamente.

# **20) Crear un modelo en Django.**

Tenemos un archivo llamado models.pyque está completamente vacío; como puedes deducir por el nombre del archivo, sirve para almacenar o definir todos los modelos de esta aplicación, por lo tanto si vienes de PHP , esto te puede chocar un poco ya que en frameworks populares como CodeIgniter o Laravel, nosotros definimos un modelo por clase, es decir, en un modelo solamente tenemos almacenados una clase y listo; pero en Django, específicamente en Python esto no es así y esto es muy bueno.



Como puedes ver, simplemente tenemos que especificar el tipo campo con la función correspondiente, por ejemplo, TextField para un campo abierto de texto, o un campo DateTimeField para un campo de fecha, especificamos el nombre de la columna que sería el mismo nombre del atributo y listo.

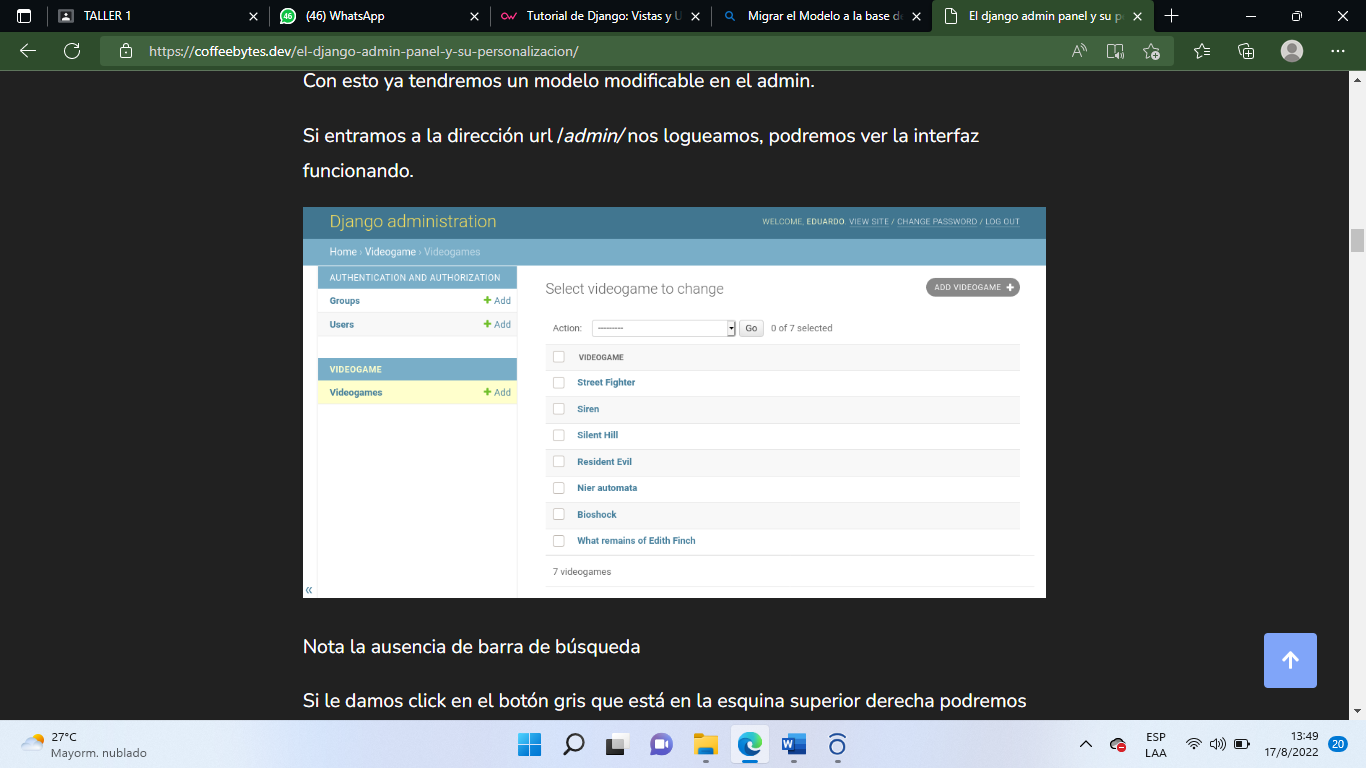
# **21) Migrar el Modelo a la base del Panel de Administración.**

 # videogameStore/admin.py

from django.contrib import admin

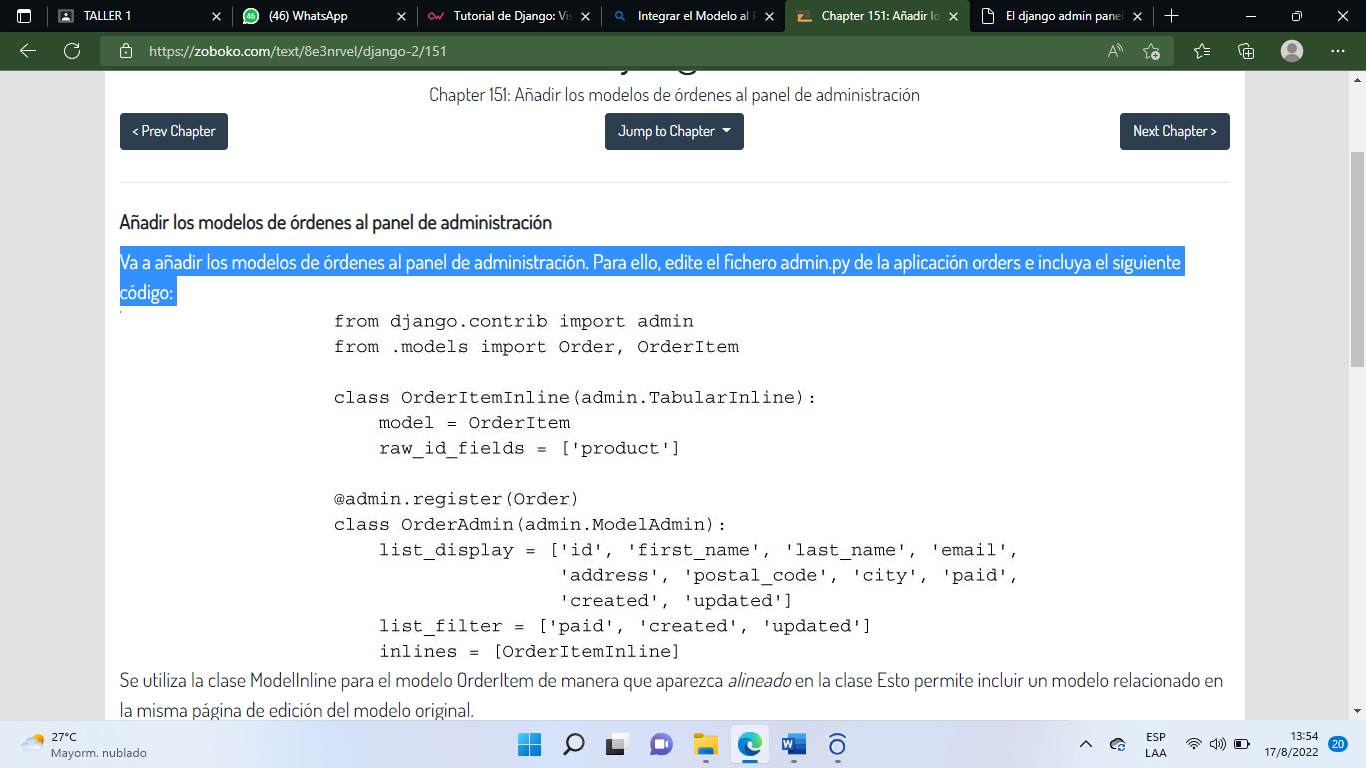
from .models import Videogame

admin.site.register(Videogame)

Con esto ya tendremos un modelo modificable en el admin.

Si entramos a la dirección url /*admin/* nos logueamos, podremos ver la interfaz funcionando.

# **22) Integrar el Modelo al Panel de Administración.**

Va a añadir los modelos al panel de administración. Para ello, edite el fichero admin.py de la aplicación orders e incluya el siguiente código:

# **23) Ingresar información al modelo por el Panel de Administración.**

Si quieres darle un toque personalizado al admin de Django, incluso poniéndole el nombre de tu negocio, o el del negocio de tu cliente. Puedes hacerlo modificando las propiedades del modelo Admin de la siguiente manera:

* # videogameStore/admin.py
* # ...
* admin.site.site\_header = 'Nombre de mi sitio'
* admin.site.index\_title = 'Panel de control de mi sitio'
* admin.site.site\_title = 'Titulo en la pestaña del navegador'

Ahora puedes apreciar que en la página principal del admin ya aparecen los cambios que hemos hecho.

# **24) Realizar la consulta de todo lo ingresado en el modelo desde el**

# **views.**

# **25) Mostrar los datos guardados en el modelo al html hijo.**