

Proyecto Web Personal

Este primer proyecto es nuestro punto de partida e introducción a Django y en él aprenderemos sus fundamentos mientras damos vida a un diseño frontend completo.

De la misma forma que un ingeniero no puede crear el motor de un coche sin tener alguna referencia sobre como será el coche en sí, no se puede crear un backend sin tener una referencia sobre como será el frontend, o por lo menos no en Django. Esa es la razón por la que he diseñado de antemano todos los frontends de este curso.

Para cada proyecto empezaremos echándole una ojeada a los diseños, identificaremos las funcionalidades que debemos implementar y finalmente les daremos vida paso a paso. Además también os enseñaré una demostración del proyecto final funcionando sobre Django para daros una visión objetiva de todo lo que haremos.

Demo de la Web Personal y recursos

Puedes ver el proyecto final en el siguiente enlace: <https://webpersonal.pythonanywhere.com/>. Recuerda que sólo los alumnos matriculados pueden descargar los diseños frontend preparados para realizar el curso paso a paso.

Las webs personales son como tarjetas de presentación virtuales. Si analizamos la web podemos encontrar distintos elementos:

- Un menú superior con cuatro páginas (portada, acerca de, portafolio y contacto).
- Si navegamos por las distintas páginas de prueba veremos que se repite siempre una cabecera con una imagen que contiene el título de la página y un subtítulo.
- Debajo tendríamos el contenido específico de cada página
- Y abajo del todo un pie o footer con unos enlaces sociales y el típico copyright.

La tarea que se nos ha encomendado es llevar a esta web a Django y añadirle un backend para manejar el portafolio, de manera que desde un panel de administrador nuestro cliente pueda gestionar proyectos dinámicamente. Todas las demás páginas las podremos dejar tal cual las tenemos en la maqueta.

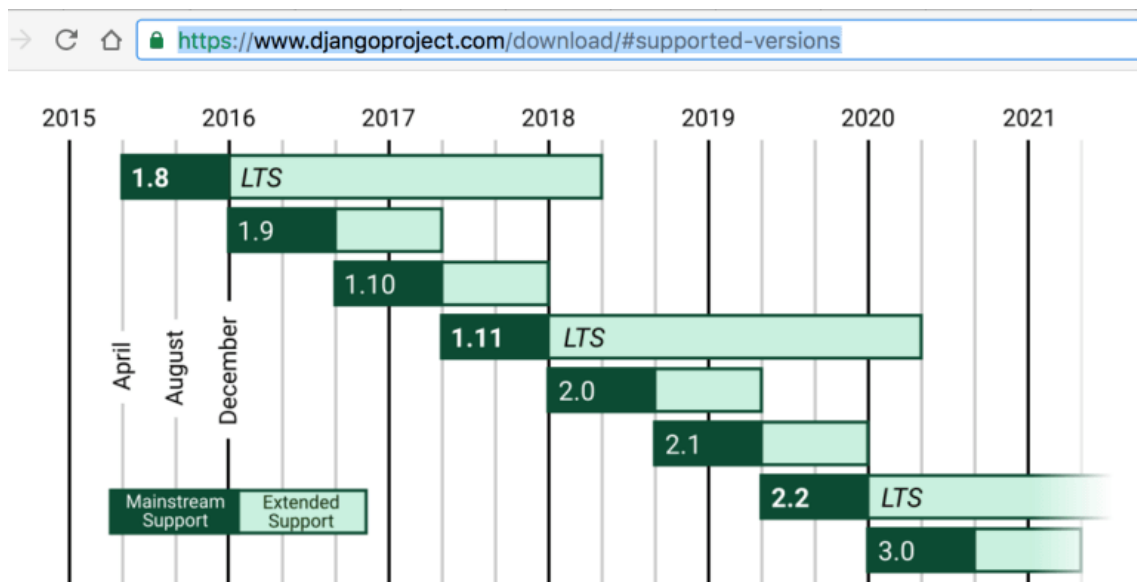
Es una web simple pero resultona y tiene todo lo necesario para aprender los fundamentos de Django, haciendo especial hincapié en el proceso de creación de los templates jerarquizados.

Creando el proyecto

Antes de crear el proyecto vamos a comentar el plan de desarrollo oficial de Django Framework.

Como sabéis este es un curso sobre Django 2.0. El caso es que tanto esta versión como sus futuras actualizaciones tendrán un ciclo de vida de 2 años hasta Diciembre de 2019, momento cuando "en teoría" se publicará Django 3.0.

Por suerte la última actualización de esta serie, la 2.2, tendrá soporte extendido hasta 2022, eso significa que seguirán arreglando bugs y fallos de seguridad pero no añadirán nuevas funcionalidades. A medida que se publiquen las actualizaciones de la versión 2 (la 2.1 y la 2.2) os compartiré los cambios más importantes en la *Sección 6. Anexo* del curso.



Ahora sí, dando por hecho que todos podemos crear entornos virtuales y tenemos un editor, voy a proceder a abrir una terminal y a crear un nuevo entorno. Lo voy a llamar **django2** haciendo referencia a que en él instalaré esta versión. Tened en cuenta que Django 2 requiere Python 3.4 o superior, así que mientras utilizéis una versión mayor a esa no deberíais tener problemas. Actualmente la versión más actual de Python es la 3.6.4 así que utilizaré esa, vosotros podéis utilizar la más actual:

```
conda create -n django2 python=3.6.4
activate django2
```

Ahora, a diferencia de lo que digo en el vídeo, se han reportado fallos de seguridad en la versión 2.0 así que os recomiendo instalar la versión 2.1 o superior de Django:

```
(django2) pip install django==2.1
```

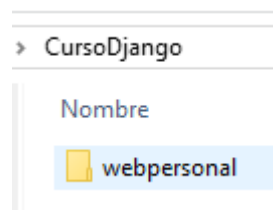
Para saber si Django está bien instalado podemos hacer lo siguiente, desde el entorno virtual:

```
(django2) python -m django --version
```

Ahora, estando en el directorio **CursoDjango** y siempre con nuestro entorno virtual activo, ejecutaremos las palabras mágicas para crear un nuevo proyecto con Django:

```
(django2) django-admin startproject webpersonal
```

Si no ocurre nada es buena señal, porque si abrimos nuestro directorio ahí tendremos creada una jerarquía de carpetas cuya raíz es el nombre que le hemos puesto **webpersonal**:



Una vez tenemos nuestro primer proyecto vamos a abrir el directorio en Visual Studio Code desde **Archivo > Abrir carpeta** o arrastrándolo al programa.

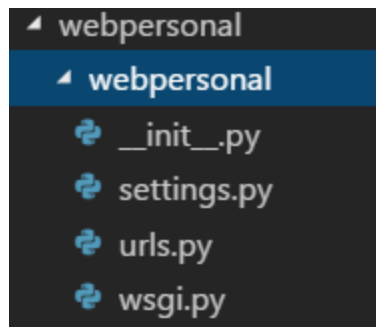
Configurando el proyecto

Con el proyecto creado, el siguiente paso es realizar la configuración inicial para ponernos en marcha, pero antes vamos a hablar brevemente de esta jerarquía de ficheros que se han creado automáticamente.

Dejando de banda el directorio **.vscode**, que crea nuestro editor, lo primero que notaréis es que la propia carpeta **webpersonal** contiene a su vez otra carpeta **webpersonal** junto a un fichero llamado **manage.py**. Este fichero es un script que sirve para gestionar todo nuestro proyecto desde la terminal, y lo vamos a estar utilizando un montón.

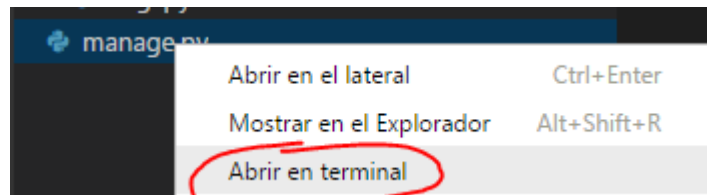
Por otro lado el subdirectorio **webpersonal** es el que contiene los scripts base del proyecto, los cuales tienen toda la configuración inicial y de despliegue.

Resumiendo mucho, el fichero **__init.py__** nos indica que la carpeta es un paquete, **settings.py** es el que contiene la configuración del proyecto, **urls.py** es el fichero encargado de manejar las direcciones de la web (sí esas que se escribirán en la barra del navegador) y por último **wsgi.py**, un script que contiene la información para desplegar el proyecto en producción, algo que trataremos en la *Sección 5. Despliegue*.



Creo que es mejor que descubramos sobre la marcha estos ficheros, así que por ahora vamos a centrarnos sólo en el manage. Desde Visual Studio Code, vamos a abrir el directorio donde se encuentra **manage.py**.

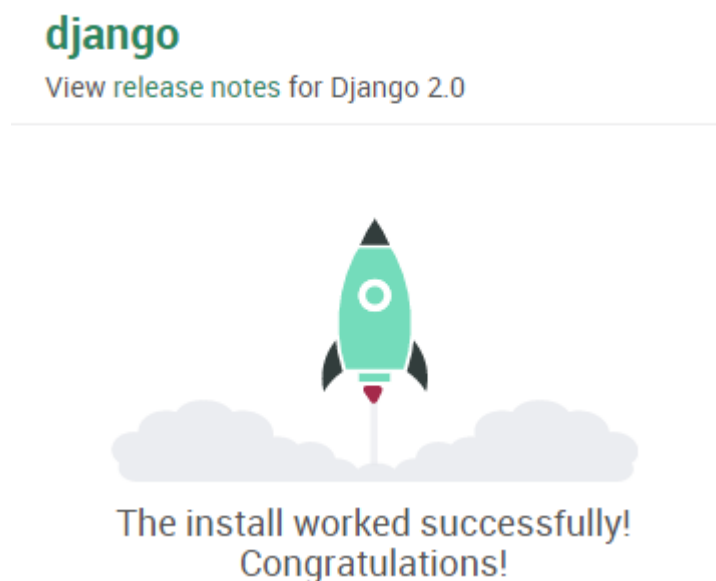
Lo vamos a ejecutar abriéndolo y ejecutando el script con clic derecho, de esa forma el propio Visual Studio Code nos activará el entorno virtual, es un pequeño truco.



Ahora vamos a ejecutar el script junto con el comando runserver, que significa correr o poner en marcha el servidor:

```
(django2) python manage.py runserver
```

Esto pondrá en marcha el servidor de desarrollo de Django y podremos acceder a la web que se ha generado haciendo clic en la dirección mientras presionamos *Control*:



Así veremos nuestro primer proyecto y cómo Django nos da la bienvenida. Fijaros que nos muestra un mensaje:

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

Estás viendo esta página porque el `DEBUG` está en `True` en tu settings y no has configurado ninguna URL.

¿Settings... eh? Vamos un momento a nuestro fichero `settings.py`. Ahí vamos a encontrar una configuración muuuy extensa, llena de variables, listas, directorios... Por ahora quedémonos con la de **`DEBUG = True`**, como véis nos indica una descripción justo encima "*AVISO DE SEGURIDAD: no poner en marcha con el debug en producción*".

```
webpersonal/settings.py
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

Hablemos un poco sobre esto porque es muy importante. El modo **`DEBUG`**, o en español el modo de **`DEPURACIÓN`**, es un modo de ejecución especial en el que siempre que ocurra un fallo, Django nos mostrará un montón de información para analizarla y poder solucionarlo. Óbviamente esta información es confidencial, ya que muestra valores de variables, rutas de directorios y otros datos importantes. Es por eso que en cuanto publiquemos o mejor dicho, despleguemos un proyecto de Django en Internet, lo primero que haremos siempre es desactivar el modo **`DEBUG`** poniéndolo a *False*.

Otra cosa interesante de Django que os va a gustar mucho, es su interfaz viene traducida a a un montón de idiomas. Podemos cambiar el idioma por defecto cambiando la variable del fichero settings: `LANGUAGE_CODE` a *es*. Si queréis algo más específico consultad la documentación del enlace justo encima:

```
webpersonal/settings.py
# Internationalization #
https://docs.djangoproject.com/en/dev/topics/i18n/
LANGUAGE_CODE = 'es'
```

Sólo habiendo hecho este cambio el servidor de Django se reiniciará aplicando los cambios, y si recargamos la pantalla de antes... ¡Sorpresa! Ahora nos sale en español:

django

Ve la notas de la versión de Django 2.0



¡La instalación funcionó con éxito!
¡Felicitaciones!

Estás viendo esta página porque `DEBUG=True`
está en tu archivo de configuración y no has
configurado ningún URL.

El siguiente paso que tenemos que llevar a cabo, justo después de crear un proyecto, es crear la base de datos inicial. Lo tenemos que hacer para que Django pueda gestionar un montón de cosas por nosotros, como por ejemplo las sesiones, los usuarios o los grupos. Para hacerlo vamos de vuelta al fichero **settings.py** y buscaremos el diccionario llamado *DATABASES*.

```
webpersonal/settings.py
# Database #
https://docs.djangoproject.com/en/dev/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Lo que tenemos aquí es una configuración predeterminada utilizando la base de datos SQLite3. Sí, Django trabaja con bases de datos SQL, y es compatible con unas cuantas. Si accedemos al enlace de la documentación veremos exactamente cuales en el apartado *ENGINES*:

ENGINE

Default: '' (Empty string)

The database backend to use. The built-in database backends are:

- `'django.db.backends.postgresql'`
- `'django.db.backends.mysql'`
- `'django.db.backends.sqlite3'`
- `'django.db.backends.oracle'`

PostgreSQL, MySQL, SQLite3 y Oracle. ¿Y qué pasa con SQL Server de Microsoft? Pues no está soportada por defecto, aunque sí existen módulos externos como Django-MSSQL, pero claro, ya requiere una configuración más avanzada.

Nosotros vamos a trabajar siempre con bases de datos SQLite3. Este motor se maneja en un fichero y no requiere instalar ningún servidor de base de datos, es lo mejor para ir aprendiendo.

La configuración por defecto para utilizar SQLite3 es simplemente el *ENGINE* y una ruta al directorio donde se creará la base de datos, que por defecto es el directorio principal del proyecto y se llamará **db.sqlite3**. Si os interesa conectar con MySQL, PostgreSQL u Oracle significa que estáis familiarizados con estos sistemas, echad un vistazo al ejemplo de la documentación para conectar PostgreSQL, no cambiará mucho en los otros servidores:

```
webpersonal/settings.py
DATABASES = {
    'default': {
        'ENGINE':
        'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432'
    }
}
```

Así pues vamos a sincronizar la base de datos inicialmente, ¿cómo lo hacemos? Pues fijaros que si volvemos a nuestra terminal, justo en el momento que hemos puesto en marcha el servidor nos aparece un mensaje:

```
You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for
app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
```

Tienes 14 migraciones sin aplicar. Tu proyecto puede no funcionar correctamente hasta que apliques las migraciones para las apps admin, auth, contenttypes y sessions. Ejecuta 'python manage.py migrate' para aplicarlas.

Bueno, pues ahí tenemos el comando. Nos pide migrar esas "apps", de las que en un momento vamos a hablar. Por ahora simplemente ejecutémoslo (parando el servidor *Control + C*):

```
(django2) python manage.py migrate
```

Con esto tenemos migrada la configuración inicial y a Django listo para sacarle todo el jugo.

Por ahora lo vamos a dejar aquí, nos vemos en la siguiente lección.

Primera App [Core] Vistas

En esta lección vamos a hablar de las *apps*.

Django apuesta por un sistema de reutilización de código organizado en *apps*, algo así como aplicaciones internas que implementan funcionalidades específicas.

¿Recordáis el mensaje que aparecía antes de migrar la base de datos? Allí se hablaba de las apps *admin*, *auth*, *contenttypes* y *sessions*. Bueno, pues esas son algunas de las apps integradas en Django que sirven para gestionar el panel de administrador y la autenticación de usuarios entre otras cosas.

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',
]
```

No sé si lo recordaréis, pero al migrar algunas de estas apps no aparecían: Messages y StaticFiles. Eso es porque no necesariamente todas las apps requieren utilizar la base de datos, aunque por contra sí requieran estar activadas en esta lista.

La genialidad de Django recae en que que a parte de incluir muchas apps genéricas también nos permite crear las nuestras propias, y eso estimados alumnos es la mejor idea de este framework, pues una app no tiene que limitarse a un solo proyecto, sino que se puede reutilizar en varios. Sin ir más lejos, en los repositorios de PyPy existen miles de apps de Django creadas por la comunidad y que en pocos minutos podríamos estar utilizando sin mucha complicación.

En este curso vamos a crear un montón de apps, desde un portafolio hasta un blog, pasando por gestores de contenidos con páginas dinámicas y otras apps para manejar el registro de usuarios y sus perfiles. Todas y cada una de ellas son reutilizables y os servirán para un montón de proyectos.

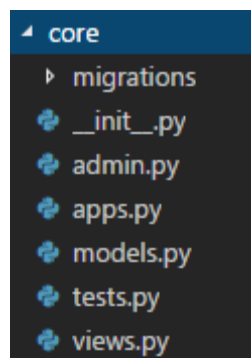
Por lo tanto podríamos concluir en que, mientras una app es una aplicación web que implementa una funcionalidad y por sí misma no sirve para nada, un proyecto es un conjunto de configuraciones a las que se "conectan" esas apps para que todo unido de lugar a un sitio web completo. Un proyecto puede contener múltiples apps, y una app puede ser incluida en múltiples proyectos.

Ahora que conocemos la diferencia entre proyecto y app, vamos a crear nuestra primera app. Será el núcleo de nuestra web personal (el **core** en inglés) y nos servirá como base para aprender como fluyen los datos en Django.

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:



Vamos a ir descubriendo los ficheros que conforman la app core sobre la marcha, no tiene mucho sentido explicar cosas que no utilizaremos hasta dentro de varias horas.

De todos estos ficheros el que nos interesa es ese llamado **views.py**. Este fichero es uno de los más importantes y en él se definen las vistas de la app. Una vista hace referencia a la lógica que se ejecuta cuando se hace una petición a nuestra web, y lo que vamos a hacer es crear una vista para procesar la petición a la raíz de nuestro sitio, lo que sería la portada.

Vamos a ir arriba del todo y vamos a importar un método del módulo **django.http** llamado **HttpResponse**:

```
core/views.py
from django.shortcuts import render, HttpResponse
```

Este método que nos permite contestar a una petición devolviendo un código, así que vamos a definir una vista para la portada y devolveremos algo de HTML de ejemplo:

```
core/views.py
def home(request):
    return HttpResponse("<h1>Mi Web Personal</h1><h2>Portada</h2>")
```

Cada vista se corresponde con una función del fichero **views.py**. Podéis usar el nombre que queráis pero como esta es la portada yo le llamo home. Además notad que se recibe un argumento llamado request, se trata de la petición y contiene mucha información, más adelante haremos uso de ella.

Ahora ya tenemos la vista con la portada, pero todavía no le hemos dicho a Django en qué URL tiene que mostrarla.

¿Recordáis el fichero **webpersonal/urls.py** dentro del directorio de configuración del proyecto? Pues es momento de volver ahí. Siguiendo la documentación superior, vamos a hacer lo que nos indican las instrucciones:

```
webpersonal/urls.py
from django.contrib import admin
from django.urls import path
from core import views

urlpatterns = [
    path('', views.home, name="home"),
    path('admin/', admin.site.urls),
]
```

¿Qué significa esto? Pues que del package **core** (notad ese **__init.py__**) importamos el módulo **views**, es decir, de la app **core** importamos las vistas. Y a continuación creamos un **patrón url**, justo en la raíz del sitio (cadena vacía) desde el que llamaremos a la vista **views.home** a la que damos el nombre **home**.

Ahora guardamos el fichero, ponemos el servidor en marcha y comprobamos la portada de nuestra web:

Mi Web Personal

Esto es la portada

¡Vaya! Ahora en lugar de aparecernos la bienvenida genérica de Django nos muestra el HTML que hemos devuelto en vista home. ¿Vais captando como fluye la información? Dentro del fichero **urls.py** establecemos un path indicando la URL donde vamos a enlazar una vista de la **app core** que a su vez estará devolviendo una respuesta HTML.

Sólo esto ya nos da mucho juego. Imaginaros que nuestra respuesta es una variable cadena, la de cosas que podemos hacer con Python y luego plasmarlas en una página HTML:

```
core/views.py
def home(request):
    html_response = "<h1>Mi Web Personal</h1>"
    for i in range(10):
        html_response += "<p>Esto es la portada</p>"
    return HttpResponse(html_response)
```

Mi Web Personal

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Esto es la portada

Pues esto estimados alumnos es el backend, la parte oculta que gracias a la programación es capaz de otorgar dinamismo al frontend.

Vamos a dejar la vista como la teníamos y continuaremos con la siguiente lección.

```
core/views.py
def home(request):
    return HttpResponse("<h1>Mi Web Personal</h1><h2>Portada</h2>")
```

Extendiendo la App [Core]

Según nuestra maqueta tenemos varias páginas más, vamos añadirlas siguiendo la misma lógica. Vamos al fichero **core/views.py** y crearemos una vista llamada *about* (acerca de en inglés):

```
core/views.py
def about(request):
```

```

return HttpResponse("""
    <h1>Mi Web Personal</h1>
    <h2>Acerca de</h2>
    <p>Me llamo Héctor y me encanta Django!</p>
""")

```

Ahora tenemos que enlazarla a una dirección, así que vamos a **webpersonal/urls.py** y la añadimos a la lista *urlpatterns*:

```

webpersonal/urls.py
from django.contrib import admin
from django.urls import path
from core import views

urlpatterns = [
    path('', views.home, name="home"),
    path('about/', views.about, name="about"),
    path('admin/', admin.site.urls),
]

```

Si ahora ponemos de nuevo el servidor en marcha y probamos la dirección **/about/** debería salirnos:



En este punto, sabiendo un poco de HTML y con algo de imaginación, podemos crear una pequeña web con un menú de enlaces: *core/views.py*

```

html_base = """
    <h1>Mi Web Personal</h1>
    <ul>
        <li><a href="/">Portada</a></li>
        <li><a href="/about/">Acerca de</a></li>
    </ul>
"""

def home(request):
    return HttpResponse(html_base + """
        <h2>Bienvenidos</h2>
        <p>Esto es la portada.</p>
    """)

def about(request):
    return HttpResponse(html_base + """
        <h2>Acerca de</h2>
        <p>Me llamo Héctor y me encanta Django!</p>
    """)

```

Mi Web Personal

- [Portada](#)
- [Acerca de](#)

Acerca de

Me llamo Héctor y me encanta Django

Como véis es una forma bien interesante de ir estructurando nuestras páginas.

Páginas portafolio y contacto

Para practicar lo que hemos hecho hasta ahora os reto a crear dos nuevas páginas:

- Una para el *Portafolio* (**vista portfolio**) en la url **portfolio/**.
- Y para el Contacto (**vista contact**) en la url **contact/**.

Podéis poner el contenido que queráis, pero no olvidéis actualizar el menú para poder navegar entre ellas, recordad que debéis crear cada vista y enlazarla a un path en el fichero webpersonal/urls.py.

Solución

```
core/views.py
html_base = """
<h1>Mi Web Personal</h1>
<ul>
  <li><a href="/">Portada</a></li>
  <li><a href="/about/">Acerca de</a></li>
  <li><a href="/contact/">Contacto</a></li>
</ul>
"""

def home(request):
    return HttpResponse(html_base + """
    <h2>Bienvenidos</h2>
    <p>Esto es la portada.</p>
    """)

def about(request):
    return HttpResponse(html_base + """
    <h2>Acerca de</h2>
    <p>Me llamo Héctor y me encanta Django!</p>
    """)

def contact(request):
    return HttpResponse(html_base + """
    <h2>Contacto</h2>
    <p>Aquí os dejo mi email y mis redes sociales:</p>
    <ul>
```

```

        <li><a href="mailto:hola@hektorprofe.net">Email</a></li>
        <li><a href="https://github.com/hcosta">Github</a></li>
        <li><a href="https://youtube.com">Youtube</a></li>
    </ul>
    """
webpersonal/urls.py

from django.contrib import admin
from django.urls import path
from core import views

urlpatterns = [
    path('', views.home, name="home"),
    path('about/', views.about, name="about"),
    path('contact/', views.contact, name="contact"),
    path('admin/', admin.site.urls),
]

```

Mi Web Personal

- [Portada](#)
- [Acerca de](#)
- [Contacto](#)

Contacto

Aquí os dejo mi email y mis redes sociales para que podáis contactarme.

- [Email](#)
- [Github](#)
- [Youtube](#)

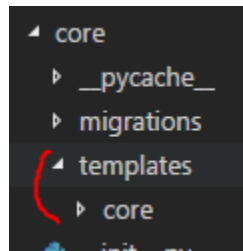
Introducción a las plantillas: Templates

Hasta ahora hemos estado devolviendo HTML plano utilizando el método **HttpResponse**. Como podéis suponer esto no es muy práctico, por eso Django nos ofrece la posibilidad de utilizar plantillas HTML (en inglés templates) mucho más cómodas y repletas de funcionalidades.

Para utilizar una plantilla lo primero es crearla, pero no la podemos crearla donde nos apetezca, debemos hacerlo siguiendo una lógica. Lo primero es crear un directorio **templates** en nuestra app, que dentro debe contener otro directorio con el mismo nombre que la app, en nuestro caso **core**.

Tenemos que hacerlo así porque Django funciona mezclando los directorios templates de las apps, de manera que al final él tiene un solo directorio **templates** y dentro otro para cada app.

Dentro de este subdirectorio **templates/core** de la app vamos a comenzar creando un fichero **home.html**:



Dentro vamos a poner todo el código HTML tal cual será devuelto al llamar la vista home, algo más o menos así:

```
core/templates/core/home.html
<h1>Mi Web Personal</h1>

<ul>
  <li><a href="/">Portada</a></li>
  <li><a href="/about/">Acerca de</a></li>
  <li><a href="/contact/">Contacto</a></li>
</ul>

<h2>Bienvenidos</h2>

<p>Esto es la portada.</p>
```

Evidentemente lo suyo sería crear bien la estructura de nuestro HTML, pero por ahora vamos a dejarlo así.

Lo que nos interesa es cambiar nuestra vista para que en lugar de devolver la respuesta `HttpResponse` devuelva este template HTML, y para lograrlo vamos utilizar el método `render` del módulo `http` de Django, que ya viene incluido por defecto.

```
core/views.py
def home(request):
    return render(request, "core/home.html")
```

Una vez hecho vamos a probar si carga la portada, pero como podréis observar no funcionará:

TemplateDoesNotExist at /
core/home.html

Por defecto Django optimiza el uso de la memoria así que no carga las plantillas de una app que no esté instalada en **settings.py**. Para cargar la app `core` y sus plantillas en memoria debemos ir al fichero **webpersonal/settings.py** y añadir la app en la lista `INSTALLED_APPS` justo abajo del todo:

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

```

    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'core', # <====
]

```

Ahora probamos de nuevo y ya funcionará como debe.

Templates about, portfolio y contact

Ahora es tu turno de crear los templates de las otras tres páginas. El resultado final será el mismo, pero ahora podrás dejar bien limpio el fichero **core/views.py** y borrar todo el código que ya no necesites.

Solución

core/templates/core/about.html

```
<h1>Mi Web Personal</h1>
```

```
<ul>
```

```
    <li><a href="/">Portada</a></li>
```

```
    <li><a href="/about/">Acerca de</a></li>
```

```
    <li><a href="/portfolio/">Portafolio</a></li>
```

```
    <li><a href="/contact/">Contacto</a></li>
```

```
</ul>
```

```
<h2>Acerca de</h2>
```

```
<p>Me llamo Héctor y me encanta Django!</p>
```

core/templates/core/portfolio.html

```
<h1>Mi Web Personal</h1>
```

```
<ul>
```

```
    <li><a href="/">Portada</a></li>
```

```
    <li><a href="/about/">Acerca de</a></li>
```

```
    <li><a href="/portfolio/">Portafolio</a></li>
```

```
    <li><a href="/contact/">Contacto</a></li>
```

```
</ul>
```

```
<h2>Portafolio</h2>
```

```
<p>Algunos de mis trabajos.</p>
```

core/templates/core/contact.html

```
<h1>Mi Web Personal</h1>
```

```
<ul>
```

```
    <li><a href="/">Portada</a></li>
```

```
    <li><a href="/about/">Acerca de</a></li>
```

```
    <li><a href="/portfolio/">Portafolio</a></li>
```

```
    <li><a href="/contact/">Contacto</a></li>
```

```
</ul>
```

```
<h2>Contacto</h2>
```

```
<p>Aquí os dejo mi email y mis redes sociales:</p>
```

```
<ul>
```

```
    <li><a href="mailto:hola@hektorprofe.net">Email</a></li>
```

```
    <li><a href="https://github.com/hcosta">Github</a></li>
```



```

        <li><a href="https://youtube.com">Youtube</a></li>
</ul>
core/views.py
from django.shortcuts import render, HttpResponseRedirect

def home(request):
    return render(request, "core/home.html")

def about(request):
    return render(request, "core/about.html")

def portfolio(request):
    return render(request, "core/portfolio.html")

def contact(request):
    return render(request, "core/contact.html")
webpersonal/urls.py
from django.contrib import admin
from django.urls import path
from core import views

urlpatterns = [
    path('', views.home, name="home"),
    path('about/', views.about, name="about"),
    path('portfolio/', views.portfolio, name="portfolio"),
    path('contact/', views.contact, name="contact"),
    path('admin/', admin.site.urls),
]

```

Herencia en nuestras plantillas

Siempre digo a mis estudiantes una frase recurrente:

En programación, si estás repitiendo código, es que lo estás haciendo mal.

¿Os habéis fijado que la cabecera con el menú es exactamente la misma en los cuatro templates?

```

<h1>Mi Web Personal</h1>

<ul>
    <li><a href="/">Portada</a></li>
    <li><a href="/about/">Acerca de</a></li>
    <li><a href="/portfolio/">Portafolio</a></li>
    <li><a href="/contact/">Contacto</a></li>
</ul>

```

Ahora imaginad que queremos cambiar el menú y añadir una nueva sección. ¿Qué problema tendríamos? Pues que deberíamos ir una a una cambiando exactamente lo mismo. ¿Es eso ideal? Para nada, más bien es un lastre, y por eso Django nos proporciona un sistema muy potente de herencia para nuestras plantillas.

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).

```
core/templates/core/base.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Mi Web Personal</title>
</head>
<body>

</body>
</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
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Mi Web Personal</title>
</head>
<body>
  <h1>Mi Web Personal</h1>
  <ul>
    <li><a href="/">Portada</a></li>
    <li><a href="/about/">Acerca de</a></li>
    <li><a href="/portfolio/">Portafolio</a></li>
    <li><a href="/contact/">Contacto</a></li>
  </ul>
</body>
</html>
```

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

```
core/templates/core/base.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Mi Web Personal</title>
</head>
<body>
  <h1>Mi Web Personal</h1>
  <ul>
    <li><a href="/">Portada</a></li>
    <li><a href="/about/">Acerca de</a></li>
    <li><a href="/portfolio/">Portafolio</a></li>
```

```

        <li><a href="/contact/">Contacto</a></li>
    </ul>

    {% block content %}{% endblock %}

</body>
</html>

```

Esto estimado alumnos, es un template tag, una etiqueta de template, y sirve para añadir lógica de programación dentro del propio HTML. Existen muchos template tags en Django, iremos descubriendo algunos de ellos sobre la marcha.

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

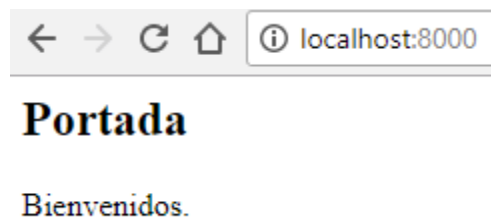
Ahora viene la magia, vamos de vuelta por ejemplo a nuestro template home.html y vamos a dejar únicamente la parte de código específica de esa plantilla:

```

core/templates/core/home.html
<h2>Mi Web Personal</h2>
<p>Bienvenidos.</p>

```

Muy bien, si ahora probamos de mostrar la portada evidentemente no nos aparecerá ni el título ni el menú:



No os preocupéis, 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:

```

core/templates/core/home.html
{% extends 'core/base.html' %}

<h2>Mi Web Personal</h2>
<p>Bienvenidos.</p>

```

Esto le indicará que es hija de base. Si ahora ejecutamos la portada:



Mi Web Personal

- [Portada](#)
- [Acerca de](#)
- [Contacto](#)

Vaya, parece que esta vez sólo se muestra el título y el menú. El caso es que no está funcionando como debería, pero vamos bien. ¿Sabéis qué nos falta? Decirle que 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:

```
core/templates/core/home.html
{% extends 'core/base.html' %}

{% block content %}
    <h2>Mi Web Personal</h2>
    <p>Bienvenidos.</p>
{% endblock %}
```

Una vez rectificado esto, ya nos debería funcionar como era de esperar:



Mi Web Personal

- [Portada](#)
- [Acerca de](#)
- [Contacto](#)

Portada

Bienvenidos.

Así es como se utiliza la herencia con plantillas, con los template tags **block** y **extends**. En la siguiente lección tendrás que poner en práctica lo aprendido adaptando las demás páginas a este sistema.

Adaptar las plantillas con herencia

Para este ejercicio deberás adaptar las plantillas *about*, *portfolio* y *contact* para que extiendan de *base*. También te reto a crear un nuevo bloque llamado *title* a

través del cual debes lograr que cambie el título en cada página dentro de la etiqueta **<title>**. Las páginas deberán tener el título siguiendo este patrón:

- Portada | Mi Web Personal
- Acerca de | Mi Web Personal
- Portafolio | Mi Web Personal
- Contacto | Mi Web Personal

Solución

```
core/templates/core/base.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %}{% endblock %} | Mi Web Personal</title>
</head>
<body>
    <h1>Mi Web Personal</h1>
    <ul>
        <li><a href="/">Portada</a></li>
        <li><a href="/about/">Acerca de</a></li>
        <li><a href="/portfolio/">Portafolio</a></li>
        <li><a href="/contact/">Contacto</a></li>
    </ul>

    {% block content %}{% endblock %}

</body>
</html>
core/templates/core/portada.html
{% extends 'core/base.html' %}

{% block title %}Portada{% endblock %}

{% block content %}
    <h2>Mi Web Personal</h2>
    <p>Bienvenidos.</p>
{% endblock %}
core/templates/core/about.html
{% extends 'core/base.html' %}

{% block title %}Acerca de{% endblock %}

{% block content %}
    <h2>Acerca de</h2>
    <p>Me llamo Héctor y me encanta Django!</p>
{% endblock %}
core/templates/core/portfolio.html
{% extends 'core/base.html' %}

{% block title %}Portafolio{% endblock %}

{% block content %}
    <h2>Portafolio</h2>
    <p>Algunos de mis trabajos.</p>
{% endblock %}
```

```
core/templates/core/contact.html
{% extends 'core/base.html' %}

{% block title %}Contacto{% endblock %}

{% block content %}
  <h2>Contacto</h2>
  <p>Aquí os dejo mi email y mis redes sociales:</p>

  <ul>
    <li><a href="mailto:hola@hektorprofe.net">Email</a></li>
    <li><a href="https://github.com/hcosta">Github</a></li>
    <li><a href="https://youtube.com">Youtube</a></li>
  </ul>
{% endblock %}
```

Template tag {% url %}

Antes de continuar no puedo evitar introducir una buena práctica para que en el futuro nunca cometáis mis errores del pasado. Me refiero al template tag **{% url %}**.

Este tag nos permite hacer referencia directamente a una view desde nuestros templates y es la forma correcta de escribir enlaces relativos dentro de nuestra web.

Vamos de vuelta a nuestro template **base.html** y vamos a sustituir los enlaces "hard code" escritos directamente, por url automatizadas.

La forma es muy sencilla **{% url 'nombre_en_el_path %}'**:

```
core/templates/core/base.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>{% block title %}{% endblock %}</title>

</head>
<body>
  <h1>Mi Web Personal</h1>
  <ul>
    <li><a href="{% url 'home' %}">Portada</a></li>
    <li><a href="{% url 'about' %}">Acerca de</a></li>
    <li><a href="{% url 'portfolio' %}">Portafolio</a></li>
    <li><a href="{% url 'contact' %}">Contacto</a></li>
  </ul>

  {% block content %}{% endblock %}
```

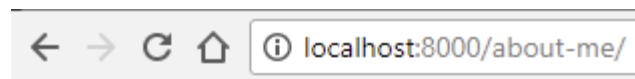
```
</body>
</html>
```

En la práctica es lo mismo, pero si vamos a nuestro **urls.py** y cambiamos una dirección, por ejemplo **about** por **about-me**:

```
webpersonal/urls.py
from django.contrib import admin
from django.urls import path
from core import views

urlpatterns = [
    path('', views.home, name="home"),
    path('about-me/', views.about, name="about"),
    path('portfolio/', views.portfolio, name="portfolio"),
    path('contact/', views.contact, name="contact"),
    path('admin/', admin.site.urls),
]
```

Sin cambiar absolutamente nada en el template, la url del menú se habrá actualizado y seguirá funcionando perfecto:



Mi Web Personal

- [Portada](#)
- [Acerca de](#)
- [Contacto](#)

Acerca de

Me llamo Héctor y me encanta Django

Por tanto la enseñanza de la lección es:

Nunca utilices hard-code para tus enlaces, en su lugar usa el template tag url.

Uniando el Frontend con el Backend

Con todo lo que hemos hecho hasta ahora hemos creado un backend funcional utilizando templates con herencia simple. Pero como recordaréis ya contamos con un frontend, así que vamos a utilizarlo ¿no?

Cuando vayamos a fusionar un frontend y un backend, mi mejor consejo es tomarlo con calma. Pararse a identificar los elementos comunes y dinámicos de las páginas para aplicar la lógica de la herencia de plantillas.

Por suerte el diseño de este primer proyecto es muy sencillo, todas las páginas tienen la misma estructura y lo único que cambia es el contenido. Vamos a empezar trasladando el código de la plantilla **base.html**. tomando como referencia la maqueta **index.html**.

Lo primero que haremos será comentar el código que tenemos en la base <!-- -> de principio a fin, así podremos tomarlo como referencia. core/templates/core/base.html

```
<!--
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %}{% endblock %}</title>

</head>
<body>
    <h1>Mi Web Personal</h1>
    <ul>
        <li><a href="{% url 'home' %}">Portada</a></li>
        <li><a href="{% url 'about' %}">Acerca de</a></li>
        <li><a href="{% url 'portfolio' %}">Portafolio</a></li>
        <li><a href="{% url 'contact' %}">Contacto</a></li>
    </ul>

    {% block content %}{% endblock %}

</body>
</html>
-->
```

Justo debajo podemos pegar todo el código de la maqueta **index.html**.

Empezando por arriba, el primer contenido dinámico que encontramos es el título, vamos a ponerlo bien:

```
core/templates/core/base.html
<title>{% block title %}{% endblock %} | Juan Pérez
(Ingeniero)</title>
```

A continuación vamos a actualizar el menú para utilizar el template tag url, de esa forma nos aseguramos que los enlaces siempre funcionarán y llevarán a la vista correspondiente:

```
core/templates/core/base.html
<div class="collapse navbar-collapse" id="navbarResponsive">
    <ul class="navbar-nav ml-auto">
        <li class="nav-item">
            <a class="nav-link" href="{% url 'home' %}">Portada</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{% url 'about-me' %}">Acerca
de</a>
        </li>
```



```

        <li class="nav-item">
            <a class="nav-link" href="{% url 'portfolio'
%}">Portafolio</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{% url 'contact'
%}">Contacto</a>
        </li>
    </ul>
</div>

```

Tampoco nos olvidemos del enlace del propio nombre de la página, que debería llevar a la portada:

```

core/templates/core/base.html
<a class="navbar-brand" href="{% url 'home' %}">Juan Pérez</a>

```

Justo debajo del menú nos topamos con un nuevo contenido dinámico: la cabecera. Esta parte consta de un título, un subtítulo y una imagen de fondo. Estos contenidos irán cambiando en cada página, de ahí lo de dinámicos. Luego nos pondremos a ello, por ahora vamos a finalizar con el bloque de contenido, que lo pondremos justo entre la cabecera y el pie de página.

```

core/templates/core/base.html
<!-- Contenido -->
{% block content %}{% endblock %}

```

Hecho esto ya podemos borrar el código de referencia que teníamos arriba comentado, sino nos dará error porque detecta bloques repetidos.

Vamos a probar qué tal se ve nuestra página:



Bueno, parece que ha cambiado un poco, pero claramente no es lo que estábamos esperando. ¿Qué está sucediendo? Pues que nos se están cargando los recursos estáticos.

Los recursos estáticos son esos contenidos que forman parte del maquetado frontend, por ejemplo los css, los javascripts y las imágenes, de ahí que no se vea nada. ¿Por qué sucede esto?

Bueno, resulta que el servidor de Django, ese que utilizamos al hacer un:

```
(django2) python manage.py runserver
```

Es sólo un servicio para utilizar durante el desarrollo y no está pensado para manejar ficheros estáticos. De esto se encargaría por ejemplo Nginx o Apache que son servidores pensados para la fase de producción.

¿Entonces no podemos ver como queda nuestra web en la fase de desarrollo? Pues sí, sí que podemos, pero debemos realizar algunas configuraciones.

Lo primero es crear un directorio en nuestra app core para almacenar los contenidos estáticos, la lógica es la misma que con el directorio templates, así que crearemos un directorio **static** y dentro otro llamado **core**, el nombre de la app, y dentro vamos a copiar todos los directorios de la maqueta que incluyen este tipo de ficheros:

CursoDjango > webpersonal > core > static > core	
Nombre	Fecha de i
css	23/01/201
img	30/01/201
js	19/01/201
vendor	19/01/201

Ahora sólo nos falta decirle a nuestro template que cargue los ficheros estáticos. Vamos de vuelta a nuestro **base.html**, y justo antes de llamar nuestros css y javascripts ejecutaremos el siguiente template tag:

```
core/templates/core/base.html
<!-- Estilos y fuentes del template -->

{% load static %}
```

Acto seguido sólo tenemos que sustituir los enlaces con los recursos mediante el tag **static** y pasándole la ruta con la app por delante, en nuestro caso **core**:

!!! note {% raw %} ``html tab="core/templates/core/base.html"

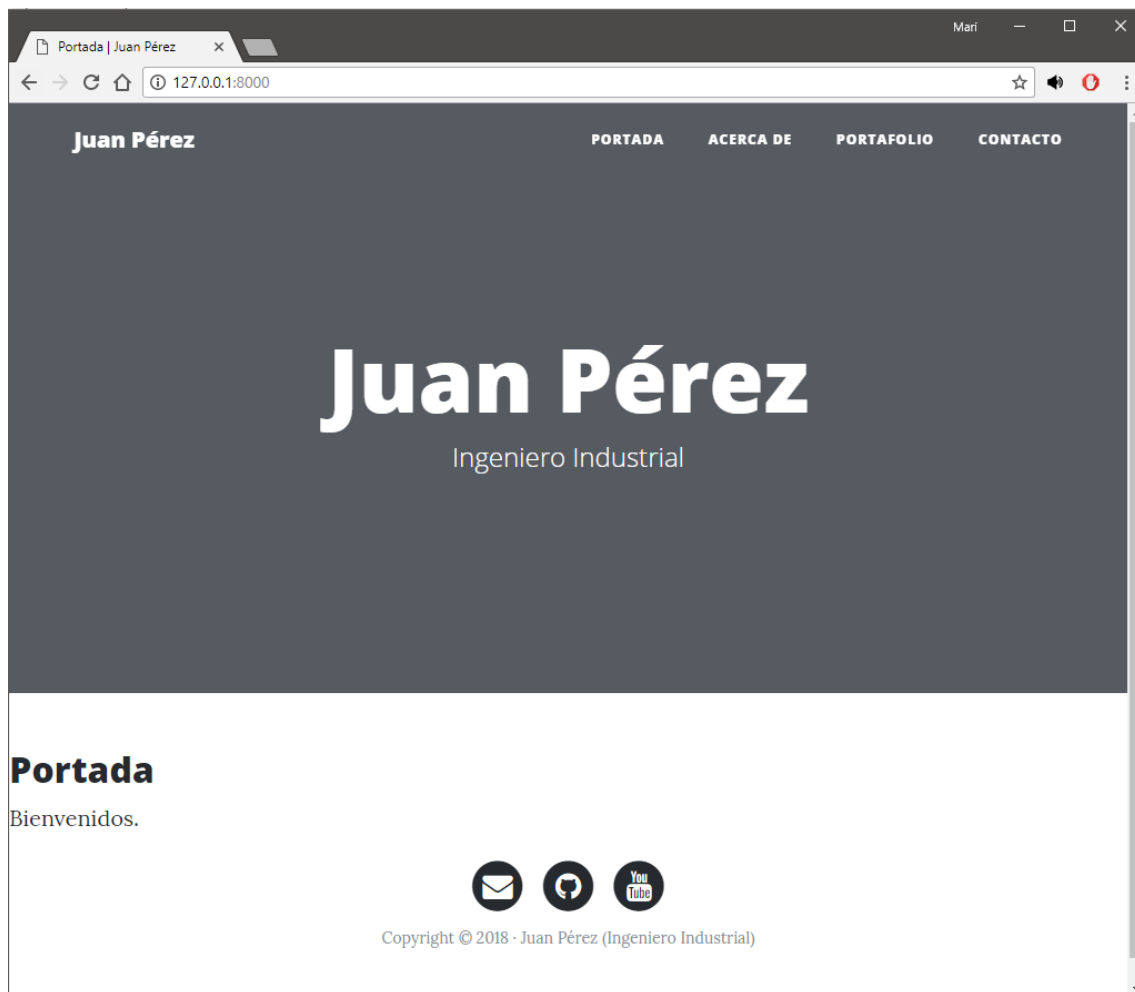
```
{% load static %}
```

No olvidemos los Javascripts de la parte inferior:

```
`core/templates/core/base.html`
``html
<!-- Bootstrap y Javascripts -->

<script src="{% static 'core/vendor/jquery/jquery.min.js'
%}"></script>
<script src="{% static
'core/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
<script src="{% static 'core/js/clean-blog.min.js' %}"></script>
```

Una vez lo tenemos vamos a probar de nuevo nuestra web:



Ahora ya nos carga los recursos, menos la imagen de la cabecera que no la hemos adaptado. Podríamos hacerlo ahora, pero como se nos alargaría mucho la lección lo haremos en la siguiente.

Creando la cabecera dinámica

Ya tenemos bastante adaptado el template a falta de algunos pequeños ajustes.

Si estudiamos un poco el código veremos que hay tres componentes cambiantes en la cabecera: la imagen, el título y el subtítulo.

¿Se os ocurre alguna forma de cambiar estos valores? Estoy seguro de que sí, por ejemplo utilizando bloques como hicimos con el título de la página. Podríamos crear dos, uno para el fondo y otro para los textos:

```
core/templates/core/base.html
<!-- Cabecera -->
<header class="masthead"
    style="background-image: url('{{ block background }}' {% endblock
%})">
    <div class="overlay"></div>
```

```

<div class="container">
  <div class="row">
    <div class="col-lg-8 col-md-10 mx-auto">
      <div class="site-heading">
        {% block headers %}{% endblock %}
      </div>
    </div>
  </div>
</div>
</header>

```

Ahora desde el template **home.html** sólo tendríamos que establecer los valores de ambos bloques:

```

core/templates/core/home.html
{% extends 'core/base.html' %}

{% load static %}

{% block title %}Portada{% endblock %}

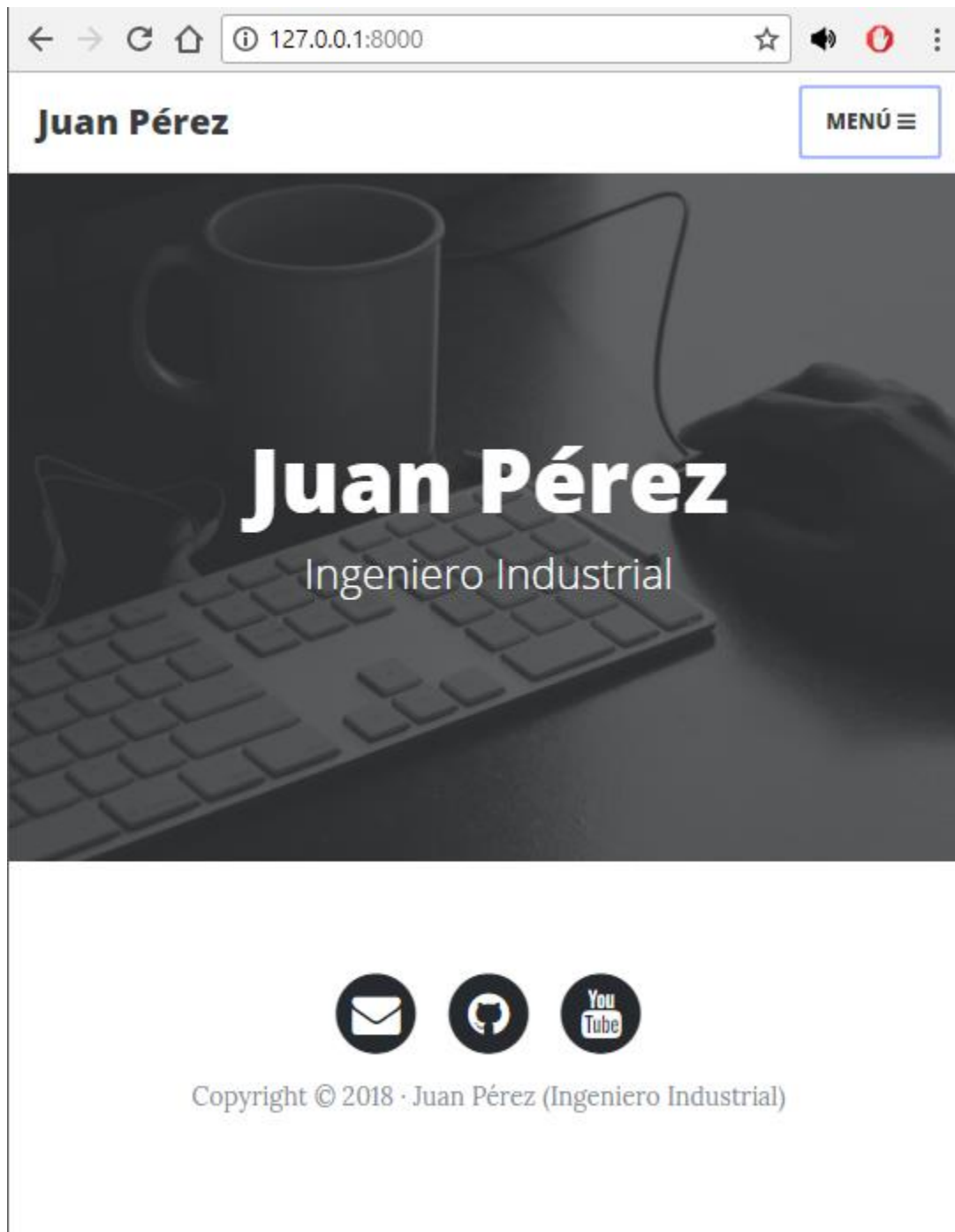
{% block background %}{% static 'core/img/home-bg.jpg' %}{% endblock %}

{% block headers %}
  <h1>Juan Pérez</h1>
  <span class="subheading">Ingeniero Industrial</span>
{% endblock %}

```

Es muy importante que dejemos el bloque background en una línea y sin utilizar espacios, ya que si ponemos saltos no se inyectará correctamente el código css con la imagen en el template **base.html**.

Hecho esto ya deberíamos ver como nuestra portada aparece perfecta:



Ahora es tu turno de adaptar los demás templates con los contenidos de prueba de las maquetas.

Adaptar los demás templates

En este ejercicio deberás tomar como referencia las maquetas *about.html*, *portfolio.html* y *contact.html* y lograr que las secciones de nuestra web con Django se vean exactamente iguales.

Recordatorio

No olvides el tag **<hr>** dentro del footer. Esta línea debe mostrarse en todas las páginas menos la portada, a ver si se te ocurre alguna forma de hacerlo. Pistas: variable **{{request.path}}** y template tags: **{% if %} {% endif %}**.

Solución

```
core/templates/core/base.html
<!-- Contenido -->
<div class="container">
    {% block content %}{% endblock %}
</div>
```

El **<hr>** se puede poner en un block, pero esta solución es más elegante.

```
core/templates/core/base.html
<!-- Contenido -->

<div class="container">
    {% block content %}{% endblock %}
</div>

{% if request.path != "/" %}<hr>{% endif %}

<!-- Pié de página -->
core/templates/core/about.html
{% extends 'core/base.html' %}
{% load static %}

{% block title %}Acerca de{% endblock %}

{% block background %}{% static 'core/img/about-bg.jpg' %}{% endblock %}

{% block headers %}
    <h1>Acerca de</h1>
    <span class="subheading">Biografía</span>
{% endblock %}

{% block content %}
<div class="row">
    <div class="col-lg-3 col-md-4 offset-lg-1">
        
    </div>
    <div class="col-lg-7 col-md-8">
        <h2 class="section-heading">Juan Pérez</h2>
        <p>Nacido en... lorem ipsum dolor sit amet, consectetur
adipisicing elit. Saepe nostrum ullam eveniet pariatur
voluptates odit, fuga atque ea nobis sit soluta odio.</p>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Saepe nostrum ullam eveniet pariatur voluptates odit...</p>
    </div>
</div>
{% endblock %}
core/templates/core/portfolio.html
{% extends 'core/base.html' %}

{% load static %}

{% block title %}Portafolio{% endblock %}
```

```

{% block background %}{% static 'core/img/portfolio-bg.jpg' %}{%
endblock %}

{% block headers %}
    <h1>Portafolio</h1>
    <span class="subheading">Currículo</span>
{% endblock %}

{% block content %}
    <!-- Proyecto -->
    <div class="row project">
        <div class="col-lg-3 col-md-4 offset-lg-1">
            
        </div>
        <div class="col-lg-7 col-md-8">
            <h2 class="section-heading title">Segundo proyecto</h2>
            <p>As we got further and further away, it [the Earth]
                diminished in size. Finally it shrank to the size of a
                marble,
                the most beautiful you can imagine. That beautiful,
                warm....</p>
            <p><a href="http://google.com">Más información</a></p>
        </div>
    </div>
    <!-- Proyecto -->
    <div class="row project">
        <div class="col-lg-3 col-md-4 offset-lg-1">
            
        </div>
        <div class="col-lg-7 col-md-8">
            <h2 class="section-heading title">Primer proyecto</h2>
            <p>As we got further and further away, it [the Earth]
                diminished in size. Finally it shrank to the size of a
                marble,
                the most beautiful you can imagine. That beautiful,
                warm....</p>
            <p><a href="http://google.com">Más información</a></p>
        </div>
    </div>
{% endblock %}
core/templates/core/contact.html
{% extends 'core/base.html' %}

{% load static %}

{% block title %}Contacto{% endblock %}

{% block background %}{% static 'core/img/contact-bg.jpg' %}{%
endblock %}

{% block headers %}
    <h1>Contacto</h1>
    <span class="subheading">Asesoría</span>
{% endblock %}

{% block content %}
    <div class="row">
        <div class="col-lg-8 col-md-10 mx-auto">

```



```
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.  
Saepe nostrum ullam eveniet pariatur voluptates odit, fuga  
atque ea nobis sit soluta odio, adipisci quas excepturi maxime  
quae totam ducimus consectetur?</p>  
<br>  
<p><b>Teléfono:</b> +09 876 543 210</p>  
<p><b>Honorarios:</b> 60€/h (precio base)</p>  
</div>  
</div>  
{% endblock %}
```

Segunda App [Portfolio] Modelos

Ahora ya tenemos el sitio fusionado, pero el requisito de nuestro cliente era tener un panel para gestionar el portafolio y ahora mismo sólo tenemos datos de prueba.

En esta lección vamos a añadir dinamismo al portafolio haciendo que interactúe con una base de datos, para ello crearemos una nueva app llamada **portfolio**. Durante su creación aprenderemos a definir modelos y a utilizar el panel de administrador. Os aseguro que os va a gustar.

Así que la pregunta es ¿qué gestiona el portafolio? ¿Proyectos no? ¿Y qué campos tiene un proyecto...? Un título, una descripción, un enlace y una imagen ¿verdad?.

Para que Django pueda manipular imágenes necesita un módulo externo llamado Pillow, vamos a empezar instalándolo en nuestro entorno virtual:

```
(django2) pip install Pillow
```

Una vez instalado Pillow vamos a definir la estructura de nuestros proyectos: aquí es donde entran en escena los modelos y el potente sistema de mapeado ORM modelo-objeto-relacional de Django. ¿Qué significa esto?

Significa que si seguimos las pautas de Django podremos trabajar con objetos mapeados en la base de datos, de manera que al crear instancias de una clase específica, estas quedarán guardadas como registros de forma automática.

En Django las clases que manejan estos objetos persistentes se conocen como Modelos. Lo vais a ver muy fácilmente con la práctica.

Vamos a crear nuestro modelo para almacenar *Proyectos*, pero como vamos a hacerlo en una nueva app tenemos que crearla antes:

```
(django2) python manage.py startapp portfolio
```

Para definir un nuevo modelo **Proyecto** nos vamos a dirigir al fichero **portfolio/models.py**, pero no podemos hacerlo de cualquier forma, debemos seguir unas pautas.

Crear un modelo es fácil, sólo tenemos que crear una clase heredando de una clase padre llamada **models.Model**. Es importante que el nombre de la clase siempre sea en singular, en nuestro caso sería **Project** (lo voy a poner en inglés para seguir un orden en todo el código):

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    pass
```

Esta clase representará una tabla dentro de la base de datos. Lo siguiente será crear sus columnas, que no serán otra cosa que los atributos de la clase. Por ahora crearemos el título, la imagen y la descripción, además de dos campos especiales que nos servirán para almacenar automáticamente la fecha y hora de creación del registro, así como la fecha y hora de la última edición. No os preocupéis por el campo de la dirección web, lo dejaremos para más adelante:

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title =
    description =
    image =
    created =
    updated =
```

Como podéis suponer con el nombre no basta. tenemos que indicarle a Django qué representarán cada uno de estos campos. Es decir, si son cadenas de texto, números, fechas, imágenes... Para hacerlo utilizaremos los modelos definidos dentro del módulo **models**.

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField()
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

Hay muchos tipos de datos, pero por ahora prefiero no entrar en detalles, luego os pondré un ejercicio donde tendréis que investigar sobre los tipos de datos. Por ahora tenemos una cadena con un título (charfield), una imagen (imagefield), un texto largo con la descripción (textfield) y dos fechas con hora (datetimefield).

Con esto ya tenemos nuestro modelo, pero todavía no podemos utilizarlo. Antes tenemos que añadir la app **portfolio** a las apps de nuestro proyecto y luego migrar el modelo a la base de datos.

```
webpersonal/settings.py
INSTALLED_APPS = [
```

```

'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'core',
'portfolio', # <====
]

```

Para migrar la app necesitaremos dos comandos:

El primero es **makemigrations**, que sirve para indicarle a Django que hay cambios en algún modelo, de manera que creará un fichero de migraciones. Si en el futuro tenemos algún problema, siempre podremos restaurar una migración anterior.

Vamos a crear la migración a ver qué sucede:

```
(django2) python manage.py makemigrations portfolio
```

Nos devuelve un error: "El campo CharField debe tener definido un atributo max_length". ¿Recordáis cuando os he dicho que los modelos están enlazados a la base de datos? Pues esa es una de esas restricciones que tienen los campos de una tabla SQL. Los campos de tipo cadena de caracter necesitan que les definamos una longitud máxima. Vamos a hacerlo:

```

portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

```

Ahora creamos de nuevo la migración y ya debería funcionar:

```
(django2) python manage.py makemigrations portfolio
```

Sólo nos falta aplicarla a la base de datos, que lo haremos con el comando **migrate**:

```
(django2) python manage.py migrate portfolio
```

Recordad, cada vez que hagamos un cambio en nuestro ficheros **models.py** ejecutaremos estos dos comandos para crear una migración y posteriormente aplicarla.

Ahora que tenemos nuestro modelo ¿podríamos crear algún registro no? ¿Cómo lo hacemos? Hay dos formas:

- Podemos crear una vista que procese un formulario y a partir de él crear los proyectos.


- O bien podemos utilizar el panel de administrador de Django y dejar que él se encargue de todo.

La primera forma tiene sentido en entornos abiertos donde tenemos que proporcionar formularios a los visitantes para que interactúen con la página. En cambio la segunda es ideal para entornos cerrados donde sólo algunos usuarios tienen acceso a la base de datos. Como nuestra web es personal y sólo nosotros o el cliente tendrá acceso a la base de datos, la segunda forma nos va de perlas, por ahora lo dejamos aquí.

Panel de administrador

Bien, como recordaréis nos quedamos en que había dos formas de añadir registros a nuestra base de datos. Una era crear las instancias manualmente a través de formularios en vistas y la otra utilizar el panel de administrador. Evidentemente vamos a utilizar la segunda, la primera la reservaremos para más adelante.

El panel de administrador de Django es una funcionalidad que viene creada por defecto. Para acceder tenemos que entrar a la dirección **/admin** de nuestra página:



Esto no es casual, si abris el **urls.py** del proyecto veréis que ya está configurado por defecto para abrirse en esa dirección como si se tratara de otra app:

```
webpersonal/urls.py
path('admin/', admin.site.urls),
```

No tenemos ningún usuario creado. Vamos a crear uno, pero no uno cualquiera, sino el superusuario, el jefazo de la página que tendrá pleno acceso. Os

recomiendo no utilizar nunca el nombre de usuario admin más allá del desarrollo, es mejor vuestro nombre o un nick cualquiera.

```
(django2) python manage.py createsuperuser
```

También es buena idea poner un email real. Una vez hecho ponemos de nuevo el servidor en marcha, nos identificamos y...



Sitio administrativo

AUTENTICACIÓN Y AUTORIZACIÓN		
Grupos	+ Añadir	✎ Modificar
Usuarios	+ Añadir	✎ Modificar
Acciones recientes		
Mis acciones		
Ninguno disponible		

¡Ahí estamos! Os presento al panel de administrador, la razón por la que me enamoré de Django, y que además desde Django 2.0 es 100% adaptativo, así que vuestros clientes podrán acceder desde su móvil o tablet sin ningún problema.

Al identificados como super usuario podemos editar cualquier tabla de la base de datos a voluntad, aunque por ahora sólo nos aparecen las tablas de grupos y usuarios.

<input type="checkbox"/>	USERNAME	EMAIL ADDRESS
<input type="checkbox"/>	admin	██████████@gmail.com

Muy bonito... pero ¿dónde está nuestro modelo de proyectos?

Para que nuestro modelo aparezca en el administrador tenemos que registrarlo en el fichero **portfolio/admin.py**:

```
portfolio/admin.py
from django.contrib import admin
from .models import Project

# Register your models here.
admin.site.register(Project)
```

Ahora actualizamos de nuevo... y ahí lo tenemos ¿qué os parece?



Ahora desde este menú podemos añadir, modificar y borrar proyectos a nuestro antojo:

Añadir project

Title:

Image: Ningún archiv...seleccionado

Description:

Sé que os dan muchas ganas de crear algo pero esperad un momento, vamos a tomarnos un descanso y a la vuelta le damos duro.

Personalizando el administrador (1)

¡Vamos a crear un proyecto!

Añadir project

Title:	Envase innovador con cierre fácil
Image:	<input type="button" value="Seleccionar archivo"/> FOG2.jpg
Description:	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.</p>

Ahora le damos a grabar y...

<input type="checkbox"/>	PROJECT
<input type="checkbox"/>	Project object (1)

Ya lo tenemos creado.

En este punto seguro que empezáis a tener algunas dudas, por ejemplo. ¿Si todo está en español porque hemos creado el modelo en inglés, no sería mejor hacerlo en español? o ¿Por qué en la lista de proyectos nos aparece *Project object (1)* en lugar del nombre del proyecto? o ¿Dónde se guarda la imagen que hemos añadido al proyecto?

Calma hombre calma... Lo que tenemos hasta ahora es sólo la configuración base del administrador, si queremos personalizarlo un poco tendremos que configurar algunas cosas.

Nombre de la app

Nuestra aplicación se llama Portfolio en inglés y quizá queremos que en el administrador aparezca Portafolio en español. Para lograrlo hay que cambiar dos cosas, primero añadir un campo **verbose_name** en el fichero **portfolio/apps.py**:

```
portfolio/apps.py
from django.apps import AppConfig

class PortfolioConfig(AppConfig):
    name = 'portfolio'
    verbose_name = 'Portafolio'
```

De esta forma podemos establecer una configuración extendida.

Lo segundo es establecer esta configuración en **settings.py**, lo cual se hace llamando a esta clase **PortfolioConfig** en lugar de **portfolio** a secas:

```
webpersonal/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
```

```

'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'core',
'portfolio.apps.PortfolioConfig', # <====
]

```

Ahora si volvemos a nuestro admin ya nos aparecerá traducido:



Campos en español

Vamos de vuelta a nuestro fichero de modelos e introduciremos dos nuevos conceptos para nuestros modelos, la subclase Meta y el método especial `__str__`.

Al crear la clase **Proyecto** hemos decidido ponerle **Project** para seguir una lógica en todo el proyecto, pero podemos cambiar el nombre a mostrar en el panel de forma muy sencilla creando una subclase con **Meta** información:

```

portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "proyecto"
        verbose_name_plural = "proyectos"

```

También es aconsejable poner un campo de ordenación por defecto para nuestros registros, que en nuestro caso podría ser la fecha de creación:

```

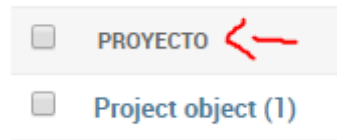
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "proyecto"
        verbose_name_plural = "proyectos"
        ordering = ["-created"] # <=====

```


Ordering es una lista porque permite ordenar con prioridades entre distintos campos. Además si añadimos un guión delante del nombre del campo, es posible ordenar de forma revertida. Al hacer **-created**, le indicamos que nos muestre primero los proyectos de más actuales a más antiguos.



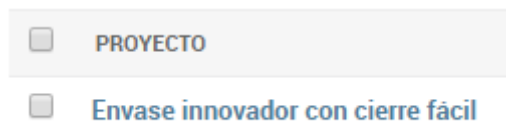
Ahora para que nos aparezca el nombre del proyecto en el desplegable simplemente podemos redefinir el método especial **__str__** para que devuelva la cadena que nosotros queramos:

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "proyecto"
        verbose_name_plural = "proyectos"
        ordering = ["-created"]

    def __str__(self):
        return self.title # <=====
```



En cuanto a los nombres de los campos, también podemos utilizar el atributo **verbose_name** para cambiarlos:

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200,
        verbose_name="Título")
    description = models.TextField(
        verbose_name="Descripción")
    image = models.ImageField(
        verbose_name="Imagen")
    created = models.DateTimeField(auto_now_add=True,
        verbose_name="Fecha de creación")
    updated = models.DateTimeField(auto_now=True,
        verbose_name="Fecha de edición")

    class Meta:
        verbose_name = "proyecto"
```

```
verbose_name_plural = "proyectos"  
ordering = ["-created"]
```

```
def __str__(self):  
    return self.title
```

Modificar proyecto

Titulo:	Envase innovador con cierre fácil
Imagen:	Actualmente: FOG2.jpg Modificar: <input type="button" value="Seleccionar archivo"/> Ningún archivo seleccionado
Descripción:	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.</p>

Campos especiales

Como podréis observar los campos de fecha y hora automatizados no aparecen en el administrador, Django los esconde para que no se puedan modificar, pero podemos mostrarlos como campos de tipo "sólo lectura".

Para hacerlo tenemos que extender un poco la configuración base del administrador de la siguiente forma:

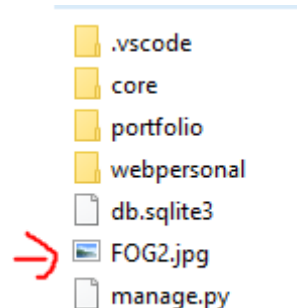
```
portfolio/admin.py  
from django.contrib import admin  
from .models import Project  
  
class ProjectAdmin(admin.ModelAdmin):  
    readonly_fields = ('created', 'updated')  
  
admin.site.register(Project, ProjectAdmin)
```

Titulo:	Envase innovador con cierre fácil
Imagen:	Actualmente: FOG2.jpg Modificar: <input type="button" value="Seleccionar archivo"/> Ningún archivo seleccionado
Descripción:	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.</p>
Fecha de creación:	31 de Enero de 2018 a las 18:48
Fecha de edición:	31 de Enero de 2018 a las 18:48

Como véis poco a poco vamos dando forma a nuestro panel de administrador. A medida que avance el curso os iré enseñando más y más funcionalidades.

Servir ficheros media

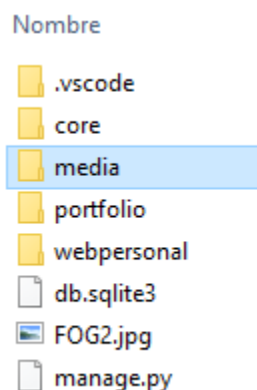
Por ahora sólo hay una cosa que no tenemos del todo bien, y es la imagen. ¿Dónde habrá ido a parar? Como no hemos configurado un directorio para las subidas, lo más seguro es que esté en la raíz de nuestro proyecto:



Esto no es nada práctico y además es peligroso. ¿Os imagináis que todas las imágenes y documentos que sube un usuario se guardasen aquí? Al final podríamos tener cientos o miles de ficheros en el mismo directorio.

Bueno, pues antes de nada una pequeña apreciación. Los ficheros que suben los usuarios no son ficheros "estáticos", no existen desde el principio. Estos se llaman ficheros "media" o multimedia.

Para que Django pueda servir ficheros Media durante el desarrollo, necesitaremos crear un directorio donde almacenar todos estos archivos. Normalmente le llamaremos **media** y lo crearemos en la raíz de nuestro proyecto.



Ahora nos dirigiremos al fichero **settings.py** y abajo del todo añadiremos estas dos variables, una para indicar la URL externa y otra para el directorio interno donde se encuentran los ficheros media (unido al core_dir del proyecto):

```
webpersonal/settings.py
# Media files
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

Ahora vamos a nuestro modelo Proyecto y vamos a añadir a la imagen un atributo llamado **upload_to**:

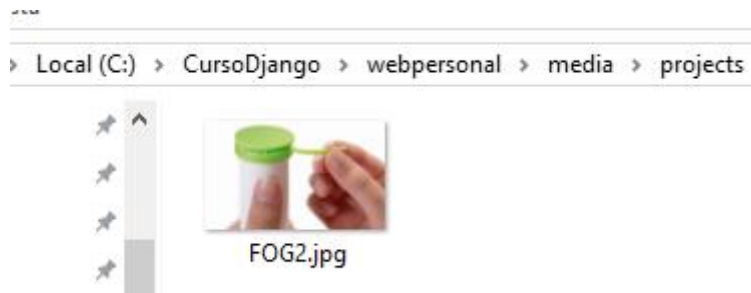
```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200,
        verbose_name="Título")
    description = models.TextField(
        verbose_name="Descripción")
    image = models.ImageField(upload_to="projects", # <=====
        verbose_name="Imagen")
    created = models.DateTimeField(auto_now_add=True,
        verbose_name="Fecha de creación")
    updated = models.DateTimeField(auto_now=True,
        verbose_name="Fecha de edición")

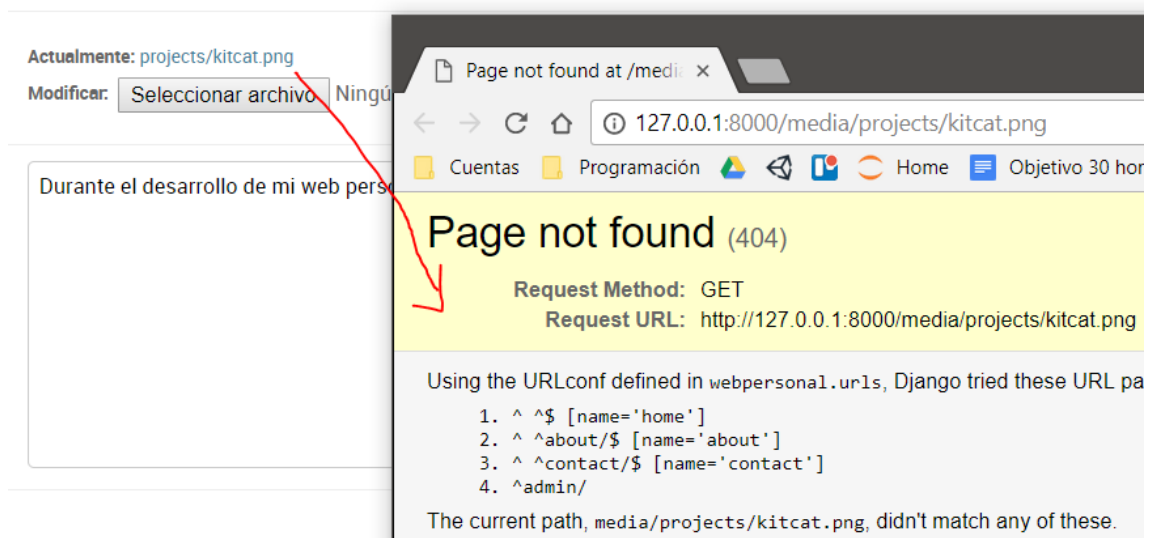
    class Meta:
        verbose_name = "proyecto"
        verbose_name_plural = "proyectos"
        ordering = ["-created"]

    def __str__(self):
        return self.title
```

Con esto le diremos a Django que suba todas las imágenes al directorio **media/projects**. Él mismo se encargará de crear el directorio si no existe. Haced la prueba, subid de nuevo la imagen y mirad los directorios:



Ahora fijaos en una cosa, si abrimos nuestro primer proyecto no podemos acceder a la imagen:



Esto sucede porque el servidor de desarrollo no puede servir estos ficheros, de eso normalmente se encargaría un servidor de producción como Apache o Nginx ya en la etapa de producción. Sin embargo y como algo temporal podemos hacer que lo haga sólo cuando tengamos el modo *DEBUG* activo.

Vamos a nuestro fichero **settings/urls.py** y vamos a editarlo de la siguiente manera:

```
webpersonal/urls.py
from django.contrib import admin
from django.urls import path
from core import views

from django.conf import settings # <=====

urlpatterns = [
    path('', views.home, name="home"),
    path('about-me/', views.about, name="about"),
    path('portfolio/', views.portfolio, name="portfolio"),
    path('contact/', views.contact, name="contact"),
    path('admin/', admin.site.urls),
]

if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT)
```

Ahora si probamos el enlace de nuevo nos aparecerá la imagen:



Lo que hemos hecho es cargar el módulo de ficheros estáticos genérico y hacer que Django sirva ficheros como algo excepcional, sólo si tenemos el modo *DEBUG* activo. Consideradlo un truco para la fase de desarrollo.

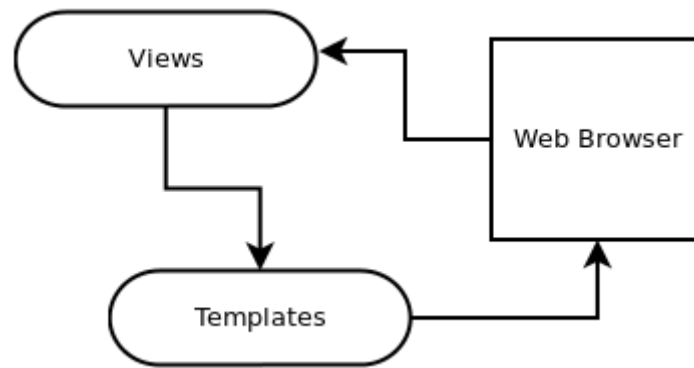
Con nuestro administrador listo y funcionando, el siguiente paso será recuperar nuestros proyectos en la vista **portfolio** y mostrarlos en el template **portfolio.html**.

Por cierto, no olvidéis borrar la imagen del directorio raíz, ya no pinta nada ahí.

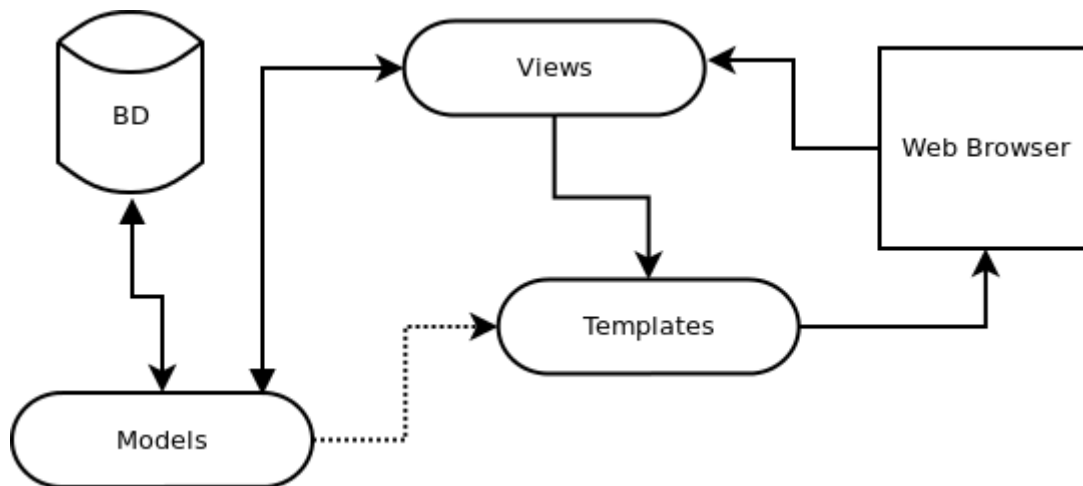
Patrón MVT: Modelo-Vista-Template

Si tenéis experiencia en el mundo de la programación seguro que habéis oído hablar del famoso patrón MVC: Modelo-Vista-Controlador. Django redefine este modelo como MVT: Modelo-Vista-Template.

Hasta ahora lo que hemos hecho no requería de interactuar con la base de datos. Podríamos decir que simplemente se recibe una petición del navegador, se ejecuta la vista correspondiente y se renderiza el Template para que el navegador muestre el HTML resultante:



Sin embargo en el momento en que aparecen las base de datos y los modelos, este proceso se extiende. Ahora se recibirá la petición, se pasará a la vista, en la vista recuperaremos los datos del modelo correspondiente, y finalmente la renderizaremos el Template pero esta vez integrando los datos dinámicos recuperados del modelo, antes de enviar el HTML final al navegador:



Vamos a hacerlo, ya veréis como en la práctica es bastante fácil.

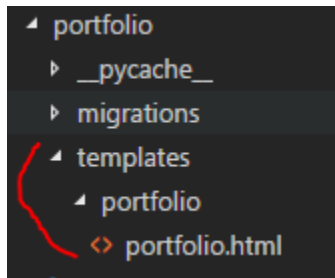
Como la lista de proyectos la enviaremos al template **portofolio.html** a través de su view **porfolio**, vamos a recuperar los datos ahí. Pero en lugar de definirla en la app Core, la vamos a trasladar al views de su propia app **portfolio/views.py**:

```

portfolio/views.py
from django.shortcuts import render

def portfolio(request):
    return render(request, "core/portfolio.html")
  
```

También crearemos una carpeta **templates/portfolio** y pondremos el template de la página **portfolio** ahí:



Y actualizamos la ruta al template:

```
portfolio/views.py
from django.shortcuts import render

def portfolio(request):
    return render(request, "portfolio/portfolio.html") # <=====
```

Ahora tenemos que readaptar las URL, pero como ahora tenemos que hacer referencia a dos apps, necesitamos importar de forma distintas las vistas:

```
webpersonal/urls.py
from django.contrib import admin
from django.urls import path

from core import views as core_views
from portfolio import views as portfolio_views

from django.conf import settings

urlpatterns = [
    path('', core_views.home, name="home"),
    path('about-me/', core_views.about, name="about"),
    path('portfolio/', portfolio_views.portfolio, name="portfolio"),
    path('contact/', core_views.contact, name="contact"),
    path('admin/', admin.site.urls),
]

if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

Esto que hemos hecho parece una tontería, pero separar las apps siempre viene bien. Nos permite organizar mejor el código y prepararlo para escalarlo en el futuro.

De vuelta a nuestra vista **portfolio**, necesitamos hacer referencia a nuestro modelo Project para recuperar sus instancias y enviarlas al template, así que lo importamos arriba del todo:

```
portfolio/views.py
from django.shortcuts import render
from .models import Project # <=====

def portfolio(request):
    return render(request, "portfolio/portfolio.html")
```


Ahora fijaros qué fácil es recuperar los registros de la tabla Projects que maneja nuestro modelo ORM a través de un su lista de objetos interna y un método `.all()` que hace referencia a todos sus objetos:

```
portfolio/views.py
from django.shortcuts import render
from .models import Project

def portfolio(request):
    projects = Project.objects.all() # <=====
    return render(request, "portfolio/portfolio.html")
```

Finalmente tenemos que inyectar estos proyectos en el template. Para hacerlo simplemente enviamos a la función render un tercer parámetro con un diccionario y los valores que queremos inyectar. De la siguiente forma:

```
portfolio/views.py
from django.shortcuts import render
from .models import Project

def portfolio(request):
    projects = Project.objects.all()
    return render(request, "portfolio/portfolio.html",
                  {'projects':projects}) # <=====
```

Teóricamente con esto habremos inyectado los proyectos, así que ahora vamos a la plantilla para debugearlos a ver si nos aparece algo:

```
portfolio/templates/portfolio/porfolio.html
{% extends 'core/base.html' %}

{% load static %}

{% block title %}Portafolio{% endblock %}

{% block background %}{% static 'core/img/portfolio-bg.jpg' %}{%
endblock %}

{% block headers %}
    <h1>Portafolio</h1>
    <span class="subheading">Currículo</span>
{% endblock %}

{% block content %}
    {% for project in projects %}
        Proyecto: {{project.title}} creado {{project.created}} <br>
    {% endfor %}
{% endblock %}
```

Probamos la página y fijaros que os mostrará algo llamado QuerySet, como una lista y dentro aparece nuestro proyecto.

Un QuerySet es la representación del resultado de una consulta a la base de datos, pero devuelta como una lista de instancias. Al ser una especie de lista, lo bueno que tiene es que podemos iterarla con otro templatetag llamado for, y dentro de cada iteración mostrar para cada Proyecto sus atributos.

Si esto lo adaptamos un poco, podemos transformar cada iteración en el proyecto que tenemos que mostrar:

```
portfolio/templates/portfolio/porfolio.html
{% extends 'core/base.html' %}

{% block title %}Portafolio{% endblock %}

{% load static %}

{% block background %}{% static 'core/img/portfolio-bg.jpg' %}{%
endblock %}

{% block headers %}
    <h1>Portafolio</h1>
    <span class="subheading">Currículo</span>
{% endblock %}

{% block content %}
    {% for project in projects %}
        <!-- Proyecto -->
        <div class="row project">
            <div class="col-lg-3 col-md-4 offset-lg-1">
                
            </div>
            <div class="col-lg-7 col-md-8">
                <h2 class="section-heading
title">{{project.title}}</h2>
                <p>{{project.description}}</p>
                {% if project.link %}
                    <p><a href="{{project.link}}">Más
información</a></p>
                {% endif %}
            </div>
        </div>
    {% endfor %}
{% endblock %}
```

Envase innovador con cierre fácil

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.

¿No es fantástico? Lo malo es que la imagen no se muestra. Si analizamos el código generado veremos la causa:

```

```

Como véis no añade la URL de los ficheros que se supone están en /media/, pero no os preocupéis. El campo ImageField tiene un atributo llamado url que nos generará su ruta correcta automáticamente teniendo en cuenta la variable **MEDIA_URL** que tenemos en **settings.py**:

```
portfolio/templates/portfolio/porfolio.html

```

Con esto debería funcionar la imagen:



Envase innovador con cierre fácil

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.

En este punto si queréis podemos volver al panel de administrador y añadir otro proyecto de prueba:

Modificar proyecto

Título:

Estudio sobre la optimización de los envases

Imagen:

Actualmente: projects/BDAYfamilia.jpg

Modificar:

[Seleccionar archivo](#)

Ningún archivo seleccionado

Descripción:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra.

Una vez creado el proyecto nos aparecerá en la parte superior, recordad que se nos ordenan de más recientes amás antiguos:



Estudio sobre la optimización de los envases plásticos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra.



Diseño de sistema abre fácil para envases plásticos de tubo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.

Fijaros en muy poco tiempo hemos transformado un template estático en una web con una sección Portafolio dinámica, donde a través de un panel de

administrador nuestros clientes pueden manejar proyectos de forma simple y cómoda, incluso a través de su teléfono móvil. Sólo nos faltaría crear un usuario para nuestro cliente desde el administrador, pero esto lo trabajaremos más a fondo en el siguiente proyecto.

Por ahora damos la web personal acabada, lo que resta de sección lo dedicaremos a algunos ejercicios prácticos.

Añadir enlace a los proyectos

No sé si lo recordaréis, pero nuestro proyecto en principio debía tener un enlace optativo para mostrar más información, así que os toca crearlo. Tenéis que seguir los siguientes pasos:

- Modificar el modelo `Project` y añadir un campo `URLField`. Tened en cuenta que es un campo opcional, así que deberéis establecer sus atributos `null` y `blank` en `True`.
- Crear la migración y migrar la app **portfolio** para aplicar los cambios.
- Modificar el Template para que muestre un enlace llamado **Más información** que llevará a la dirección establecida. Debéis contemplar la posibilidad de que no haya un enlace y en ese caso no mostrar nada. Para ello utilizad el template tag `{% if condicion %} {% endif %}`.
- Finalmente añade una dirección en algún proyecto, deja otro vacío para ver si no se muestra en enlace.

Solución

```
portfolio/models.py
from django.db import models

class Project(models.Model):
    title = models.CharField(max_length=200,
                             verbose_name="Título")
    description = models.TextField(
        verbose_name="Descripción")
    image = models.ImageField(upload_to="projects",
                              verbose_name="Imagen")
    link = models.URLField(null=True, blank=True, # <=====
                           verbose_name="Dirección Web")
    created = models.DateTimeField(auto_now_add=True,
                                   verbose_name="Fecha de creación")
    updated = models.DateTimeField(auto_now=True,
                                   verbose_name="Fecha de edición")

    class Meta:
        verbose_name = "proyecto"
        verbose_name_plural = "proyectos"
        ordering = ["-created"]

    def __str__(self):
        return self.title

(django2) python manage.py makemigrations portfolio
(django2) python manage.py migrate portfolio
```

Dirección Web:

Actualmente: <https://www.google.es/>

Cambiar:

```
portfolio/templates/portfolio/porfolio.html
{% extends 'core/base.html' %}

{% block title %}Portafolio{% endblock %}

{% load static %}

{% block background %}{% static 'core/img/portfolio-bg.jpg' %}{%
endblock %}

{% block headers %}
    <h1>Portafolio</h1>
    <span class="subheading">Currículo</span>
{% endblock %}

{% block content %}
    {% for project in projects %}
        <!-- Proyecto -->
        <div class="row project">
            <div class="col-lg-3 col-md-4 offset-lg-1">
                
            </div>
            <div class="col-lg-7 col-md-8">
                <h2 class="section-heading
title">{{project.title}}</h2>
                <p>{{project.description}}</p>
                {% if project.link %}
                    <p><a href="{{project.link}}">Más
información</a></p>
                {% endif %}
            </div>
        </div>
    {% endfor %}
{% endblock %}
```



Estudio sobre la optimización de los envases plásticos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra.

Más información



Diseño de sistema abre fácil para envases plásticos de tubo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempor quam sed leo viverra, a posuere massa volutpat. Nulla facilisi. Fusce ullamcorper risus id placerat pharetra. Duis posuere pulvinar velit, id malesuada mi dignissim vitae. Vivamus elit est, dictum vitae congue finibus, maximus et urna.

No