

IES Fernando Aguilar Quignon

1º ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

KARLITA

José María Riol Sánchez
3 de mayo de 2021

Índice

1. Instalación - PIP, Python Django	2
2. Empezar Proyecto Nuevo	3
3. Primera migración	4
4. Crear superusuario + Administración de Django	5
5. Primera aplicación	7
6. Primer modelo	8
7. Crear Objetos en Python Shell + Registrar Model en Admin.	9
8. Personalizar Modelo en el Admin	11
9. Primera vista	13
10. Configuración de Plantillas.	15
11. Escribir Formulario.	16
12. Formulario en una Vista.	16
13. Método HTTP POST en Formulario.	17
14. Validaciones Formulario pt.I	18
15. Guardar Datos del Formulario con el Modelo	20
16. Model Form.	23
17. Validaciones Model Form	25
18. Contexto en la vista, plantilla	28
19. ModelForm en la vista	31
20. Custom Form para Contacto	35
21. Configurar Email	38
22. Configuración de Archivos Estáticos	40
23. Configuración Bootstrap	42
24. Plantillas: Herencia, Include Tag, Blocks	45
25. Django Crispy Forms	49
26. Estilo: Bootstrap 1	51
27. Estilo: CSS Custom	56

Introducción

En este documento vamos a recoger todo el progreso que vayamos haciendo en la elaboración del curso PROBAR DJANGO | Crear una Aplicación Web en Udemy. Dividiremos el documento de ahora en adelante en secciones por cada vídeo, y dentro de cada sección tendremos subsecciones donde tendremos todo lo relacionado con el trabajo realizado, el log de cabezazos y demás información que se pida en la tarea.

Todos los commits de este curso podrán verse en el siguiente enlace → [Repositorio de Karlita](#)

1. Instalación - PIP, Python Django

En este vídeo nos enseña como instalar Python, en nuestro caso ya tenemos instalado Python3, como instalar pip, hacer un entorno virtual e instalar Django. Refrescamos como hacer un entorno virtual. Para crear uno, una vez estamos en el directorio deseado hacemos uso del comando:

```
python3 -m venv curso_django
```

Esto nos creará un directorio con la estructura de sistema de archivos de como Python guarda los módulos que se han instalado, de esa manera no afectarán a los otros módulos que tengamos instalados en nuestro sistema.

Para activarlo hacemos uso del comando:

```
source curso_django/bin/activate
```

```
joserisa@pc09-t3:/VMs/joserisa$ source curso_django/bin/activate
(curso_django) joserisa@pc09-t3:/VMs/joserisa$ █
```

Podemos ver que el prompt ha cambiado. Ahora dentro de este entorno virtual, con pip descargamos django con el siguiente comando:

```
pip install django
```

```
(curso_django) joserisa@pc09-t3:/VMs/joserisa$ pip install django
Collecting django
  Downloading https://files.pythonhosted.org/packages/a8/9b/fe94c509e514f6c227308e81076506eb9d67f2fbfb8061ce5cdfbde0432e3/Django-3.2-py3-none-any.whl (7.9MB)
    100% |██████████| 7.9MB 164kB/s
Collecting asgiref<4,>=3.3.2 (from django)
  Downloading https://files.pythonhosted.org/packages/17/8b/05e225d1154b8f5358e6a6d277679c9741ec0339d1e451c9cef687a9170/asgiref-3.3.4-py3-none-any.whl
Collecting pytz (from django)
  Downloading https://files.pythonhosted.org/packages/70/94/784178ca5dd892a98f113cd923372024dc04b8d40abe77ca76b5fb90ca6/pytz-2021.1-py2.py3-none-any.whl (510kB)
    100% |██████████| 512kB 1.8MB/s
Collecting sqlparse==0.2.2 (from django)
  Downloading https://files.pythonhosted.org/packages/14/05/6e8eb62ca685b10e34051a80d7ea94b7137369d8c0be5c3b9d9b6e3f5dae/sqlparse-0.4.1-py3-none-any.whl (42kB)
    100% |██████████| 512B 2.7MB/s
Collecting typing_extensions; python_version < "3.8" (from asgiref<4,>=3.3.2->django)
  Downloading https://files.pythonhosted.org/packages/60/7a/e881b5abb54db0e6e671ab088d079c57ce54e8a01a3ca443f561ccadb37e/typing_extensions-3.7.4.3-py3-none-any.whl
Installing collected packages: typing-extensions, asgiref, pytz, sqlparse, django
Successfully installed asgiref-3.3.4 django-3.2 pytz-2021.1 sqlparse-0.4.1 typing_extensions-3.7.4.3
(curso_django) joserisa@pc09-t3:/VMs/joserisa$ █
```

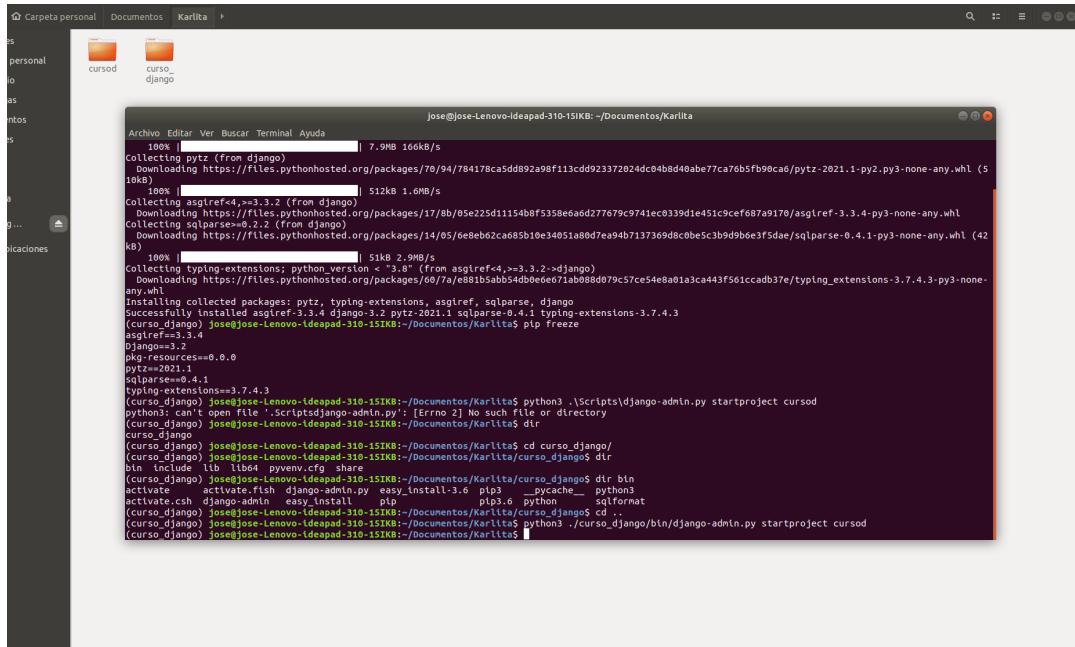
Con estas configuraciones ya estamos preparados para empezar un nuevo proyecto.

2. Empezar Proyecto Nuevo

En este capítulo nos muestra como crear un entorno virtual para el curso, nosotros ya lo tenemos creado del vídeo anterior.

Procedemos a crear el nuevo proyecto en nuestro caso usando el siguiente comando dentro de nuestra carpeta Karlita.

```
python3 ./curso_django/bin/django-admin.py startproject cursod
```

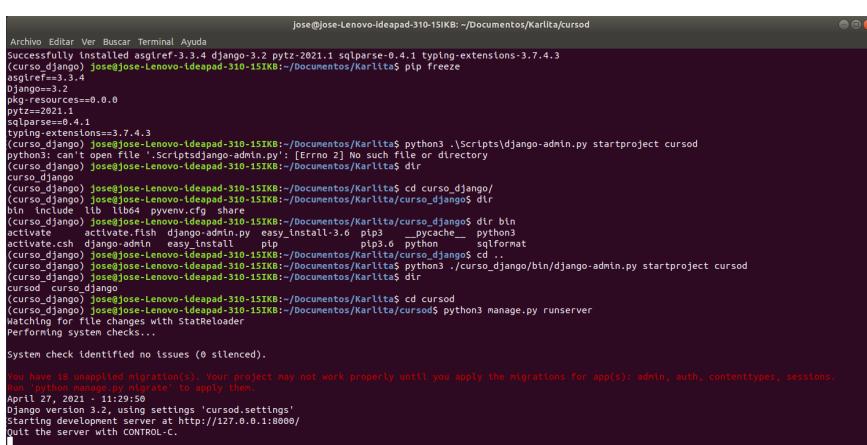


```
jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita$ python3 ./curso_django/bin/django-admin.py startproject cursod
Successfully installed asgiref==3.3.4 django==3.2 pytz==2021.1 sqlparse==0.4.1 typing-extensions==3.7.4.3
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita$ cd cursod
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita/cursod$ ls
__pycache__ manage.py
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita/cursod$ pip freeze
sqlparse==0.4.1
typing-extensions==3.7.4.3
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita/cursod$ cd ..
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita$ pip freeze
sqlparse==0.4.1
typing-extensions==3.7.4.3
(cursod) jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita$
```

Podemos observar que se nos creó la carpeta cursod. Nos movemos a esa carpeta y ejecutamos:

```
python3 manage.py runserver
```

Y tendremos el siguiente efecto:

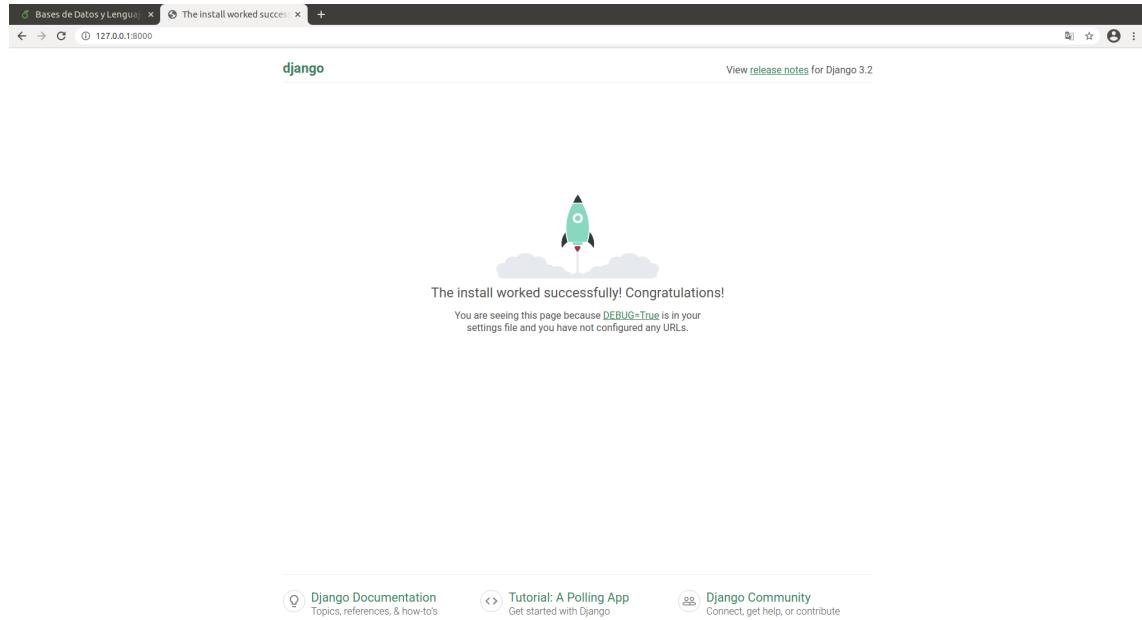


```
jose@jose-Lenovo-Ideapad-310-15IKB: ~/Documentos/Karlita/cursod$ python3 manage.py runserver
Performing system checks...
System check identified no issues (0 silenced).

You have 10 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.

Django Version 3.2, using settings 'cursod.settings'
Starting development server at http://127.0.0.1:8000/
quit the server with CONTROL-C.
```

Nos interesa la IP, `http://127.0.0.1:8000/` la copiamos, abrimos el navegador y la pegamos, nos aparece la siguiente pantalla:



Tenemos pues el primer proyecto funcionando.

No hacemos ningún commit porque no hemos hecho nada de código, solamente hemos configurado un par de cosas y comprobado que funciona.

3. Primera migración

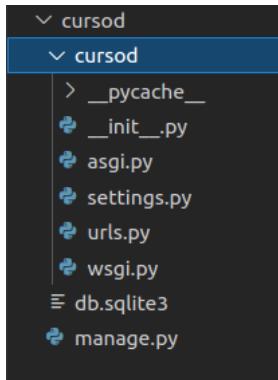
Debido al mensaje: `You have 18 unapplied migration(s)` debemos empezar a hacer nuestras migraciones, para que esté todo sincronizado y funcionando bien. Debemos crear el vínculo entre la BD y nuestro proyecto, para ello cerramos el server y ejecutamos:

```
python3 manage.py migrate
```

y nos lleva a cabo lo siguiente:

```
^C(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursos$ python3 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 admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursos$ ^C
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursos$
```

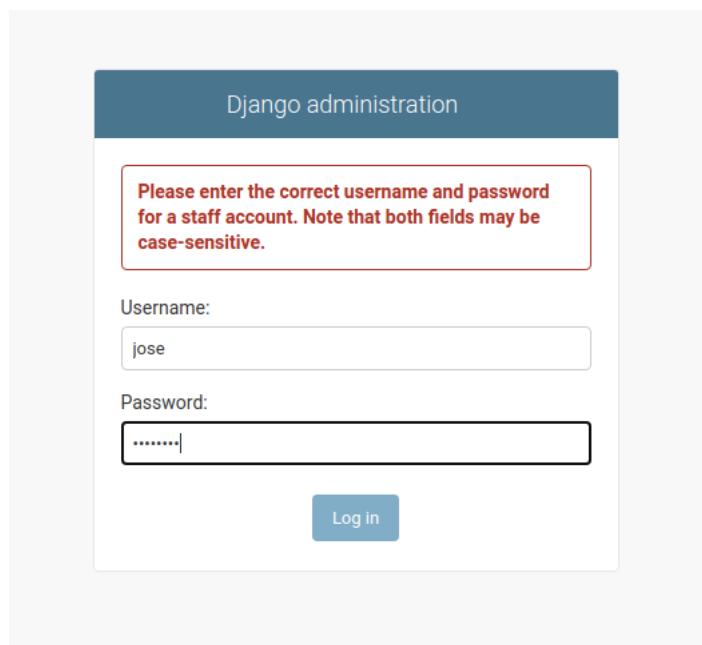
En nuestro caso usamos CODE para añadir la carpeta de trabajo cursod y supongo que trabajar desde ahí.



En este caso tampoco hacemos ningún commit, no hubo cambio alguno.

4. Crear superusuario + Administración de Django

Debido a que cuando arrancamos el server y procedemos a hacer login dice que nuestro usuario no está registrado.



Por tanto, volvemos a cerrar el servidor y procedemos a crear un super usuario que nos permita entrar, ejecutamos el comando:

```
python3 manage.py createsuperuser
```

y completamos con la información que nos pida.

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$ python3 manage.py createsuperuser
Username (leave blank to use 'jose'):
Email address: josemaria.riol.sanchez.alu@iesfernandoaguilar.es
Password:
Password (again):
Superuser created successfully.
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$
```

Si volvemos a ejecutar nuestro server e intentamos entrar:

The screenshot shows the Django admin dashboard at 127.0.0.1:8000/admin/. The title bar says "Django administration". On the left, there's a sidebar titled "AUTENTICATION AND AUTHORIZATION" with "Groups" and "Users" listed. On the right, there's a "Recent actions" panel showing "My actions" and "None available".

Podemos ver que ahora sí nos deja entrar.

Vamos a poner ahora la página en español, nos vamos a settings.py y nos vamos a la línea LANGUAGE_CODE = 'es' y al recargar nos saldrá la página en español.

The screenshot shows the Django admin dashboard at 127.0.0.1:8000/admin/. The title bar says "Administración de Django". The sidebar now says "AUTENTICACIÓN Y AUTORIZACIÓN" with "Grupos" and "Usuarios" listed. The right side has sections for "Acciones recientes" (Recent actions) and "Mis acciones" (My actions), both currently empty.

Navegamos dentro de grupos y usuarios y podemos ver la lista de los mismos.

The screenshot shows the "Añadir usuario" (Add user) form. It asks for a username ("prueba") which is required to be between 150 characters long, containing letters, digits, and specific symbols. It also asks for a password and a confirmation of the password. Below the form, it says "Para verificar, introduzca la misma contraseña anterior." (To verify, enter the same password again.).

Podemos ver que podemos añadir usuarios, grupos. Si le damos a añadir usuario y creamos un usuario cualquiera podremos ver que la lista de usuarios se actualizará.

The screenshot shows the "Selección de usuario a modificar" (Select user to modify) page. It lists two users: "jose" and "prueba". The "jose" entry shows the email "josemaria.riol.sanchez.alu@iesfernandoaguilar.es". There are checkboxes for "NOMBRE DE USUARIO" and "DIRECCIÓN DE CORREO ELECTRÓNICO". A "seleccionados 0 de 2" message is displayed. At the bottom, it says "2 usuarios".

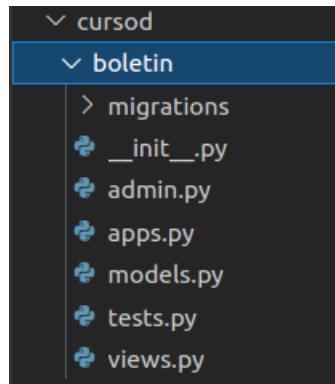
5. Primera aplicación

En esta ocasión crearemos nuestra primera aplicación mediante un comando. Cerramos el server si está abierto y escribimos:

```
python3 manage.py startapp boletin
```

```
^C(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$ python3 manage.py startapp boletin
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$
```

Se nos creará una carpeta con el nombre boletín dentro de nuestro "cursod" con varios ficheros.



Ahora tenemos que registrarlo en el settings.py de nuestro proyecto. Entramos en settings.py y añadimos boletin de la siguiente forma:

```
29
30
31     # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'boletin',
41 ]
```

6. Primer modelo

Procedemos a modificar models.py de boletín.

```
cursod > boletin > models.py > ...
1  from __future__ import unicode_literals
2
3  from django.db import models
4
5  # Create your models here.
6  class Registrado(models.Model):
7      nombre= models.CharField(max_length=100, blank=True, null=True)
8      email= models.EmailField()
9      timestamp= models.DateTimeField(auto_now_add=True, auto_now=False)
10
11     def __unicode__(self):
12         return self.email
13
14     def __str__(self):
15         return self.email
```

Una vez hayamos terminado con models.py debemos hacer una migración, haciendo uso de dos comandos:

```
python3 manage.py makemigrations
```

Con esto empaquetaremos los cambios.

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$ python3 manage.py makemigrations
Migrations for 'boletin':
  boletin/migrations/0001_initial.py
    - Create model Registrado
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$
```

El siguiente comando sería `python3 manage.py migrate`

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, boletin, contenttypes, sessions
Running migrations:
  Applying boletin.0001_initial... OK
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$
```

Y ya habríamos terminado por aquí.

7. Crear Objetos en Python Shell + Registrar Model en Admin.

Vamos a crear un objeto de Python a partir del shell, hacemos uso del comando:

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/cursod$ python3 manage.py shell
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> 
```

Hacemos una importación desde nuestro modelo, y posteriormente ponemos una variable nueva "gente", creado como un set de objetos.

```
>>> from boletin.models import Registrado
>>> gente = Registrado.objects.all()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
AttributeError: type object 'Registrado' has no attribute 'objets'
>>> gente
<QuerySet []>
>>> 
```

Si queremos añadir una persona, lo haríamos así:

```
>>> personal = Registrado.objects.create(nombre='joselito', email='j@email.com')
>>> personal
<Registrado: j@email.com>
>>> 
```

El problema es que tal y como hemos hecho esto, no hay forma de ver los cambios fuera del shell, así que vamos a llevarlo a cabo a través de la administración de Django. Nos metemos en admin.py de boletin y registramos nuestro modelo de la siguiente forma:

```
cursod > boletin > admin.py > ...
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Registrado
5
6  admin.site.register[Registrado] 
```

Volvemos a ejecutar nuestro servidor y entramos en él, observamos que tenemos boletin, Registrados y dentro de Registrados 1 registrado.

The screenshot shows the Django Admin interface with the following structure:

- Header:** Administración de Django
- Sitio administrativo**
- AUTENTICACIÓN Y AUTORIZACIÓN** section with links for Grupos and Usuarios, each with 'Añadir' and 'Modificar' buttons.
- BOLETIN** section with a link for Registrados, which also has 'Añadir' and 'Modificar' buttons.

Administración de Django

Inicio > Boletín > Registrados

AUTENTICACIÓN Y AUTORIZACIÓN	
Grupos	+ Añadir
Usuarios	+ Añadir
BOLETIN	
Registrados	+ Añadir

Seleccione registrado a modificar

Acción: Ir seleccionados 0 de 1

<input type="checkbox"/> REGISTRADO	j@email.com
-------------------------------------	-------------

1 registrado

Viene solamente el email por el unicode que escribimos, si lo cambiamos a nombre se reflejará el cambio.

Administración de Django

Inicio > Boletín > Registrados

AUTENTICACIÓN Y AUTORIZACIÓN	
Grupos	+ Añadir
Usuarios	+ Añadir
BOLETIN	
Registrados	+ Añadir

Seleccione registrado a modificar

Acción: Ir seleccionados 0 de 1

<input type="checkbox"/> REGISTRADO	joselito
-------------------------------------	----------

1 registrado

Al pinchar en el nombre vemos los dos campos, uno en negrita y otro no, porque pusimos que uno sí fuera obligatorio.

Administración de Django

Inicio > Boletín > Registrados > joselito

AUTENTICACIÓN Y AUTORIZACIÓN	
Grupos	+ Añadir
Usuarios	+ Añadir
BOLETIN	
Registrados	+ Añadir

Modificar registrado

joselito

Nombre:

Email:

[Eliminar](#)

En el unicode no es recomendable poner un campo que no sea obligatorio, puesto que al registrarse alguien, si deja un campo en blanco se vería en blanco en la página y no es lo idóneo. Dejamos models.py como estaba con el email, aunque a nosotros nos da error y no se registra el usuario, pero dejando en el unicode el email, va bien.

Django de forma automática de un id aunque nosotros no lo pusiéramos en el código.

8. Personalizar Modelo en el Admin

Queremos personalizar el display de la administración, lo debemos hacer en el código de admin.py.

```
admin.py ● models.py
cursod > boletin > admin.py > ...
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Registrado
5
6  class AdminRegistrado(admin.ModelAdmin):
7      list_display = ["__unicode__", "nombre", "timestamp"]
8      class Meta:
9          model = Registrado
10
11 admin.site.register(Registrado, AdminRegistrado)
```

Volvemos a la página, recargamos y tenemos:

ACCION	UNICODE	NOMBRE	TIMESTAMP
<input type="checkbox"/>	lu@email.com	-	28 de Abril de 2021 a las 09:50
<input type="checkbox"/>	p@email.com	k	28 de Abril de 2021 a las 09:50
<input type="checkbox"/>	j@email.com	joselito	28 de Abril de 2021 a las 09:34

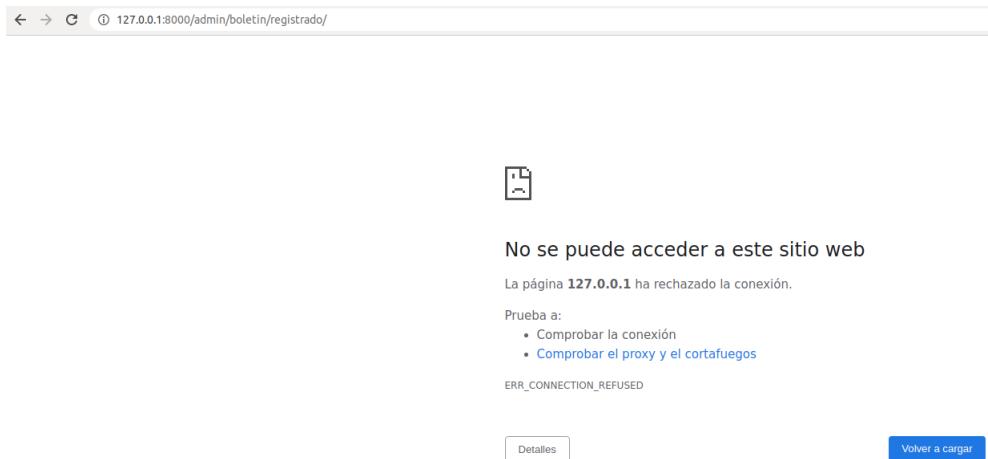
3 registrados

Podemos cambiar a nombre en list_display y si volvemos a recargar la página nos paraecerán los nombres, pero lo vamos a dejar con el email puesto.

Añadimos en admin.py las líneas:

```
list_filter = ["timestamp"]
list_editable = [.email]
search_fields = [.email, "nombre"]
```

Cuando recargamos la página nos da error.



Miramos la consola y vemos lo siguiente.

```
fn(*args, **kwargs)
  File "/home/jose/Dокументos/Karlita/curso_django/lib/python3.6/site-packages/django/core/management/commands/runserver.py", line 118, in inner_run
    self.check(display_num_errors=True)
  File "/home/jose/Dокументos/Karlita/curso_django/lib/python3.6/site-packages/django/core/management/base.py", line 469, in check
    raise SystemCheckError(msg)
django.core.management.base.SystemCheckError: System check identified some issues:

ERRORS:
<class 'boletin.admin.AdminRegistrado'>: (admin.E124) The value of 'list_editable[0]' refers to the first field in 'list_display' ('email'), which
cannot be used unless 'list_display_links' is set.

System check identified 1 issue (0 silenced).

/home/jose/Dокументos/Karlita/cursod/boletin/admin.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

Exception in thread django-main-thread:
Traceback (most recent call last):
  File "/usr/lib/python3.6/threading.py", line 916, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.6/threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "/home/jose/Dокументos/Karlita/curso_django/lib/python3.6/site-packages/django/utils/autoreload.py", line 64, in wrapper
    fn(*args, **kwargs)
  File "/home/jose/Dокументos/Karlita/curso_django/lib/python3.6/site-packages/django/core/management/commands/runserver.py", line 118, in inner_run
    self.check(display_num_errors=True)
  File "/home/jose/Dокументos/Karlita/curso_django/lib/python3.6/site-packages/django/core/management/base.py", line 469, in check
    raise SystemCheckError(msg)
django.core.management.base.SystemCheckError: System check identified some issues:

ERRORS:
<class 'boletin.admin.AdminRegistrado'>: (admin.E124) The value of 'list_editable[0]' refers to the first field in 'list_display' ('email'), which
cannot be used unless 'list_display_links' is set.
```

No podemos tener email en editable si es el primer elemento de la lista display, porque contiene un enlace hacia el usuario, por tanto es algo que no se puede modificar. Si ponemos nombre en editable, no hay problema y nos dejaría editarla.

<input type="checkbox"/>	EMAIL	NOMBRE
<input type="checkbox"/>	lu@email.com	
<input type="checkbox"/>	p@email.com	k
<input type="checkbox"/>	j@email.com	joselito
3 registrados		

Si añadimos list_display_links podemos solventar de otra forma este error y además podemos conservar en editable el email, haremos que el link esté en el nombre.

	EMAIL	NOMBRE
<input type="checkbox"/>	lu@email.com	-
<input type="checkbox"/>	p@email.com	k
<input type="checkbox"/>	j@email.com	joselito

3 registrados

Vemos que podemos editar ahora el email y el enlace es el nombre ahora, podemos observar que quien no tiene nombre le añade un guión. En el vídeo no le aparece guión y explica como solventarlo, aunque a nosotros no nos haga falta cambiar nada, lo haremos por continuar el curso.

9. Primera vista

Para ello entramos en el archivo views.py y lo editamos.

```
cursod > boletin > ✎ views.py > ...
1  from django.shortcuts import render
2
3  # Create your views here.
4  def inicio(request):
5      return render(request, "inicio.html", {})
```

Podemos observar que en el return tenemos inicio.html que sería una plantilla y tenemos un diccionario vacío. Pasamos ahora al fichero urls.py. Aquí tenemos ya una divergencia entre el vídeo y la realidad, ya que su forma de poner la URL es distinta de la mía, yo tengo que seguir la forma que tiene mi url para que funcione sin problema.

```
"""cursod URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

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

Debemos importar ahora nuestras views poniendo import en el mismo fichero en el que nos encontramos.

```
16  from django.contrib import admin
17  from django.urls import path
18
19  from boletin import views
20  #from boletin.views inicio
21
22  urlpatterns = [
23      path('admin/', admin.site.urls),
24      path('', views.inicio, name='inicio')
25
26  ]
27  |
```

Cuando volvemos a la página, nos vemos lo siguiente:

TemplateDoesNotExist at /

inicio.html

Request Method: GET
Request URL: http://127.0.0.1:8000/
Django Version: 3.2
Exception Type: TemplateDoesNotExist
Exception Value: inicio.html
Exception Location: /home/jose/Documentos/Karlita/curso_django/lib/python3.6/site-packages/django/template/loader.py, line 19, in get_template
Python Executable: /home/jose/Documentos/Karlita/curso_django/bin/python3
Python Version: 3.6.9
Python Path: ['/home/jose/Documentos/Karlita/cursod', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/lib-dynload', '/home/jose/Documentos/Karlita/curso_django/lib/python3.6/site-packages']
Server time: Wed, 28 Apr 2021 10:47:43 +0000

Template-loader postmortem

Django tried loading these templates, in this order:

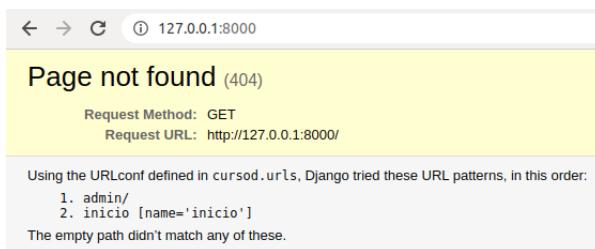
Using engine django:

- django.template.loaders.app_directories.Loader: /home/jose/Documentos/Karlita/curso_django/lib/python3.6/site-packages/django/contrib/admin/templates/inicio.html (Source does not exist)
- django.template.loaders.app_directories.Loader: /home/jose/Documentos/Karlita/curso_django/lib/python3.6/site-packages/django/contrib/auth/templates/inicio.html (Source does not exist)

Traceback [Switch to copy-and-paste view](#)

```
/home/jose/Documentos/Karlita/curso_django/lib/python3.6/site-packages/django/core/handlers/exception.py, line 47, in inner
47.             response = get_response(request)
```

Es normal el error porque no hemos creado la plantilla. Si cambiamos la línea que contiene `path('', views.inicio, name='inicio')` por `path('inicio', views.inicio, name='inicio')`, al refrescar nos dice que la página no existe.



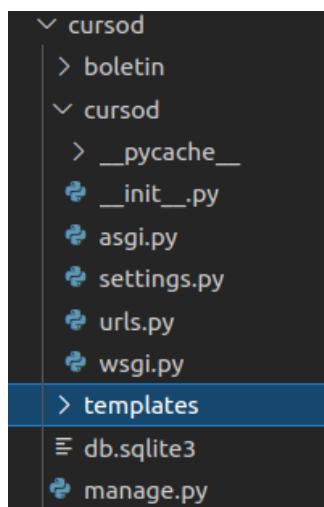
Si entramos en la dirección `http://127.0.0.1:8000/inicio` nos saldrá el mismo error de que la plantilla no existe, aunque esta vez en `/inicio`.

10. Configuración de Plantillas.

Tenemos el error de que no encuentra las plantillas. Si nos vamos a nuestro fichero settings.py y bajamos hasta donde pone TEMPLATES, podemos ver que podemos añadirlas ahí. El directorio de plantillas está inicialmente vacío. Para las rutas aquí tendríamos otra divergencia entre el vídeo y la realidad. Nuestra forma de hacerlo sería tal que así: 'DIRS': [BASE_DIR / 'templates']

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Debemos añadir ahora en el directorio de cursod una carpeta con ese nombre.



Así sabrá dónde encontrar nuestras plantillas. Si ahora dentro de la carpeta templates, creamos un archivo de nombre inicio.html y escribimos alguna cosa como:

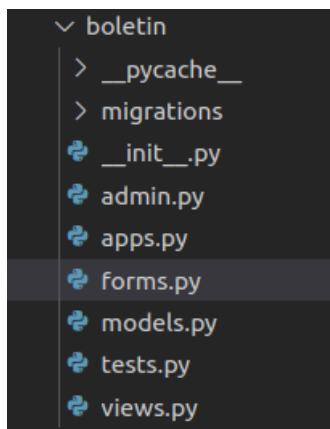
```
cursod > templates > < inicio.html > h1
1   <h1>Hola mundo</h1>
```

Si volvemos a la página, esta vez no habrá error:



11. Escribir Formulario.

Para los formularios debemos crear dentro de nuestra carpeta boletín el archivo forms.py



Una vez creado el fichero, escribimos lo siguiente:

```
cursod > boletin > forms.py > ...
1  from django import forms
2
3  class RegForm(forms.Form):
4      nombre = forms.CharField(max_length=100)
5      edad = forms.IntegerField()
```

12. Formulario en una Vista.

Vamos a views.py

```
cursod > boletin > views.py
1  from django.shortcuts import render
2
3  from .forms import Reg
4  Form
5  # Create your views here.
6  def inicio(request):
7      form = RegForm()
8      context = {"el_form":form,}
9      return render(request, "inicio.html", context)
```

Luego vamos a nuestra plantilla.

```
cursod > templates > inicio.html > ...
1  <h1>Hola mundo</h1>
2
3  {{ el_form }}
```

Si nos vamos para nuestra página, veremos el formulario.

The screenshot shows a browser window with the URL 127.0.0.1:8000. The page title is "Hola mundo". Below the title is a form with two input fields: "Nombre:" and "Edad:". Below the fields is a submit button with the value "Regístrate".

Hacemos un par de cambios en el html.

```
cursod > templates > inicio.html > form > input
1  <h1>Hola mundo</h1>
2  <form>
3  {{ el_form.as_p }}
4  <input type='submit' value= 'Regístrate' />
5  </form>
```

Y la página luce tal que así

The screenshot shows a browser window with the URL 127.0.0.1:8000. The page title is "Hola mundo". Below the title is a form with two input fields: "Nombre:" and "Edad:". Below the fields is a submit button with the value "Regístrate".

13. Método HTTP POST en Formulario.

Si intentamos meter datos en el formulario ahora los datos se ponen en la URL. Haciendo los siguientes cambios...

```
cursod > templates > inicio.html > form
1  <h1>Hola mundo</h1>
2  <form method ='POST' action ="">{{ csrf_token %}}
3  {{ el_form.as_p }}
4  <input type='submit' value= 'Regístrate' />
5  </form>
```

Podemos ver que en la URL ya no aparece nada, porque esta vez se hizo el método POST, lo podemos ver en la terminal.

```
System check identified no issues (0 silenced).
May 01, 2021 - 16:51:46
Django version 3.2, using settings 'cursod.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[01/May/2021 16:51:54] "GET / HTTP/1.1" 200 276
Not Found: /favicon.ico
[01/May/2021 16:51:56] "GET /favicon.ico HTTP/1.1" 404 2110
[01/May/2021 16:55:33] "GET / HTTP/1.1" 200 297
[01/May/2021 17:07:58] "GET / HTTP/1.1" 200 444
[01/May/2021 17:08:03] "POST / HTTP/1.1" 200 444
```

Esto forma parte del ciclo de petición-respuesta.

14. Validaciones Formulario pt.I

En el vídeo al hacer el cambio en este fichero añadiendo request.POST

```
cursod > boletin > ✎ views.py
 1   from django.shortcuts import render
 2
 3   from .forms import RegForm
 4   # Create your views here.
 5   def inicio(request):
 6       form = RegForm(request.POST)
 7       context = {"el_form":form,}
 8       return render(request, "inicio.html", context)]
```

en la página de Karlita le sale los campos obligatorios, a mí por lo menos no.



Igualmente seguimos el vídeo para "quitarlos".

Añadiendo or None ya se arregla.

```
cursod > boletin > ✎ views.py
1   from django.shortcuts import render
2
3   from .forms import RegForm
4   # Create your views here.
5   def inicio(request):
6       form = RegForm(request.POST or None)
7       context = {"el_form":form,}
8       return render(request, "inicio.html", context)
```

Nos enseña un truco, poniendo print(dir(form)) miramos la consola y vemos lo que podemos todo lo que podemos hacer con form.

```
System check identified no issues (0 silenced).
May 01, 2021 - 17:23:39
Django version 3.2, using settings 'cursod.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
 '__getitem__', '__gt__', '__hash__', '__html__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__',
 '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', '__weakref__', '__bound_fields_cache__', 'clean_fields', 'clean_form',
 'errors', 'html_output', 'post_clean', 'add_error', 'add_initial_prefix', 'add_prefix', 'as_p', 'as_table',
 'as_ul', 'auto_id', 'base_fields', 'changed_data', 'clean', 'data', 'declared_fields', 'default_renderer',
 'empty_permitted', 'error_class', 'errors', 'field_order', 'fields', 'files', 'full_clean', 'get_initial_for_field',
 'has_changed', 'has_error', 'hidden_fields', 'initial', 'is_bound', 'is_multipart', 'is_valid', 'label_suffix',
 'media', 'non_field_errors', 'order_fields', 'prefix', 'renderer', 'use_required_attribute', 'visible_fields']
[01/May/2021 17:23:41] "POST / HTTP/1.1" 200 468
```

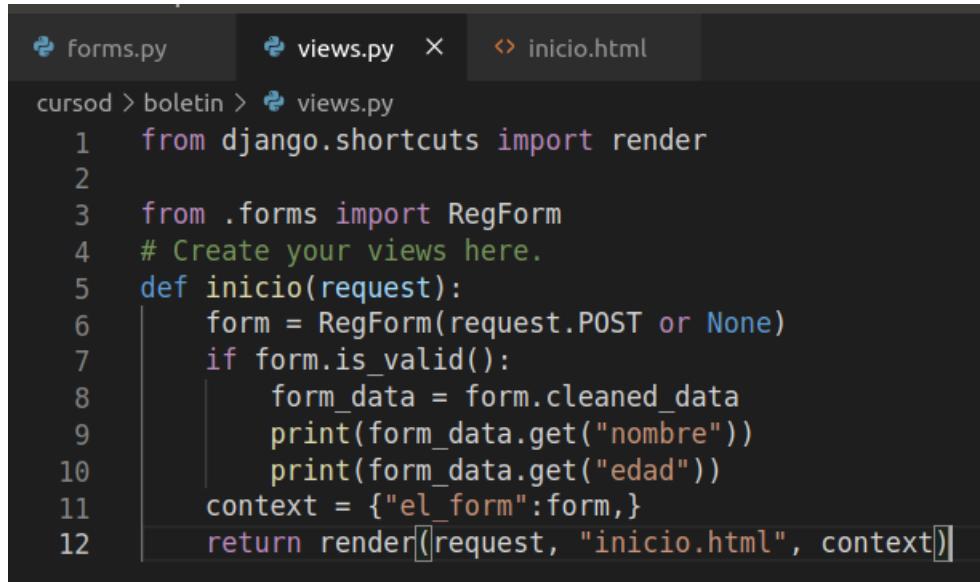
Añadiendo el if, cuando metemos datos en la página web en la consola nos aparecen en un diccionario. Divergencia en el vídeo, nuestro print debe ir entre paréntesis para que funcione.

```
cursod > boletin > ✎ views.py
1   from django.shortcuts import render
2
3   from .forms import RegForm
4   # Create your views here.
5   def inicio(request):
6       form = RegForm(request.POST or None)
7       if form.is_valid():
8           print (form.cleaned_data)
9       context = {"el_form":form,}
10      return render(request, "inicio.html", context)
```

Metemos los datos karli y 1.

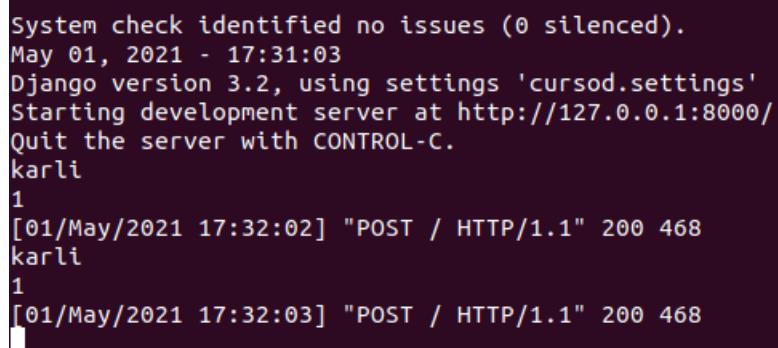
```
System check identified no issues (0 silenced).
May 01, 2021 - 17:27:44
Django version 3.2, using settings 'cursod.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
{'nombre': 'karli', 'edad': 1}
[01/May/2021 17:28:09] "POST / HTTP/1.1" 200 468
{'nombre': 'karli', 'edad': 1}
[01/May/2021 17:28:10] "POST / HTTP/1.1" 200 468
```

Cambiamos el code y guardamos en una variable los datos para luego ponerlos en dos prints.



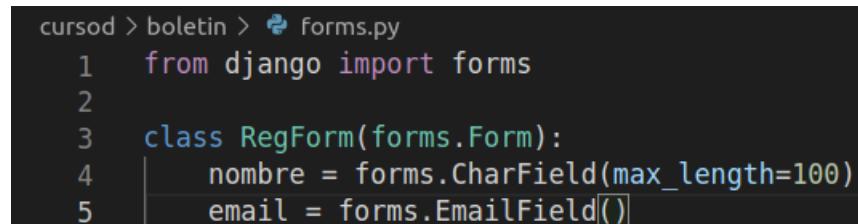
```
cursod > boletin > views.py
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  # Create your views here.
5  def inicio(request):
6      form = RegForm(request.POST or None)
7      if form.is_valid():
8          form_data = form.cleaned_data
9          print(form_data.get("nombre"))
10         print(form_data.get("edad"))
11         context = {"el_form":form,}
12         return render(request, "inicio.html", context)
```

De esta forma al meter los datos los veremos en la consola de forma limpia.



```
System check identified no issues (0 silenced).
May 01, 2021 - 17:31:03
Django version 3.2, using settings 'cursod.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
karli
1
[01/May/2021 17:32:02] "POST / HTTP/1.1" 200 468
karli
1
[01/May/2021 17:32:03] "POST / HTTP/1.1" 200 468
```

15. Guardar Datos del Formulario con el Modelo



```
cursod > boletin > forms.py
1  from django import forms
2
3  class RegForm(forms.Form):
4      nombre = forms.CharField(max_length=100)
5      email = forms.EmailField()
```

Podemos ver que en nuestra página hay cambios por esto.



Hola mundo

Nombre:

- Este campo es obligatorio.

Email:

En form quitamos la línea print de edad porque ya no nos sirve y hacemos un par de cambios más.

```
cursod > boletin > views.py
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  from .models import Registrado
5  # Create your views here.
6  def inicio(request):
7      form = RegForm(request.POST or None)
8      if form.is_valid():
9          form_data = form.cleaned_data
10         abc = form_data.get("email")
11         obj = Registrado.objects.create(email = abc)
12         context = {"el_form":form,}
13         return render(request, "inicio.html", context)
```

Vamos a la página a meter unos datos de ejemplo.



Hola mundo

Nombre:

- Este campo es obligatorio.

Email:

Podemos ver que ha sido creado y guardado en nuestra BD. Aunque pusimos nombre no tiene, porque en las views solo tenemos el email.

EMAIL	NOMBRE
laura@email.com	
otro@email.com	
lu@email.com	
p@email.com	k
j@email.com	joselito

Hacemos los siguientes cambios.

```
cursod > boletin > forms.py      views.py      inicio.html      models.py
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  from .models import Registrado
5  # Create your views here.
6  def inicio(request):
7      form = RegForm(request.POST or None)
8      if form.is_valid():
9          form_data = form.cleaned_data
10         abc = form_data.get("email")
11         abc2 = form_data.get("nombre")
12         obj = Registrado.objects.create(email = abc, nombre =abc2)
13
14         #obj = Registrado()
15         #obj.email = abc
16         #obj.save()
17         context = {"el_form":form,}
18         return render(request, "inicio.html", context)
```

Vamos a la página y volvemos a meter otro usurario, Laura2.

Ahora si tenemos el nombre también porque así lo pusimos en views.

16. Model Form.

Si echamos un vistazo al formulario que hace Django por nuestro modelo:

Añadir registrado

Nombre:

Email:

```
<!-- Container -->
<div id="container"> </div>
<!-- Header -->
<div id="header"></div> </div>
<!-- END Header -->
<div class="breadcrumbs"></div>
<div class="main shifted" id="main"> </div>
<div class="sticky toggle-nav-sidebar" id="toggle-nav-sidebar" aria-label="Activar navegación lateral" ></div>
<nav class="sticky" id="nav-sidebar" ></nav>
<div class="content">
  <!-- Content -->
  <div id="content" class="colM">
    <h1>Añadir registrado

```

Nosotros lo haremos mediante model form para que sea dinámico y aproveche los campos de nuestro modelo.

```
cursod > boletin > forms.py
1  from django import forms
2
3  from .models import Registrado
4
5  class RegModelForm(forms.ModelForm):
6      class Meta:
7          model = Registrado
8          fields = ["email"]
9
10 class RegForm(forms.Form):
11     nombre = forms.CharField(max_length=100)
12     email = forms.EmailField()
```

```

forms.py      admin.py      views.py      inicio.html
cursod > boletin > admin.py
1   from django.contrib import admin
2
3   # Register your models here.
4   from .forms import RegModelForm
5   from .models import Registrado
6
7   class AdminRegistrado(admin.ModelAdmin):
8       list_display = ["email", "nombre", "timestamp"]
9       form = RegModelForm
10      #list_display_links = ["nombre"]
11      list_filter =[ "timestamp"]
12      list_editable = [ "nombre"]
13      search_fields = [ "email", "nombre"]
14      #class Meta:
15      #    model = Registrado
16
17      admin.site.register(Registrado, AdminRegistrado)
18

```

Podemos ver en nuestra página los cambios.

Si añadimos nombre:

```

forms.py      admin.py      views.py      inicio.html
cursod > boletin > forms.py
1   from django import forms
2
3   from .models import Registrado
4
5   class RegModelForm(forms.ModelForm):
6       class Meta:
7           model = Registrado
8           fields = [ "nombre", "email"]
9
10  class RegForm(forms.Form):
11      nombre = forms.CharField(max_length=100)
12      email = forms.EmailField()

```

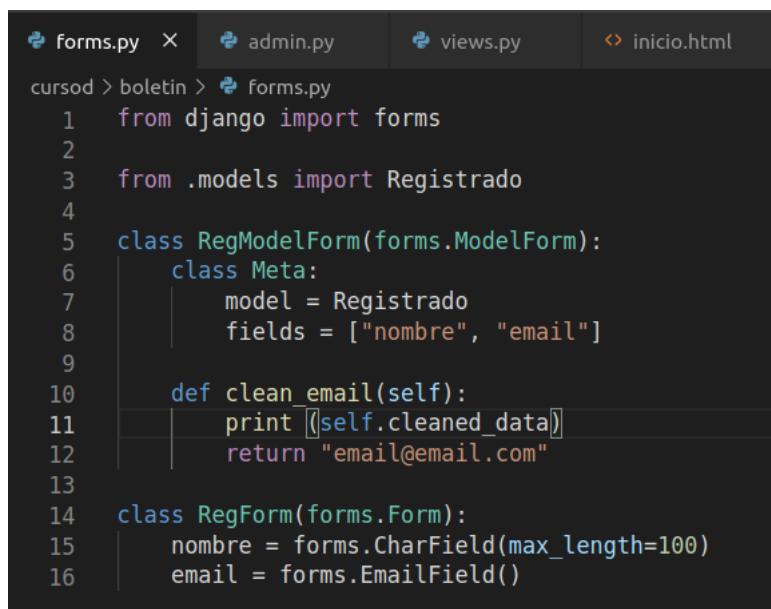
En la página lo vemos así:

The screenshot shows a Django admin page titled "Añadir registrado". On the left, there's a sidebar with "REGISTROS Y AUTORIZACIÓN" and buttons for "Añadir" (Add). The main area is titled "Añadir registrado" and contains two input fields: "Nombre:" and "Email:". Below these fields is a large empty text area.

Así tenemos nuestro model form.

17. Validaciones Model Form

Vamos a poner nuestras propias validaciones.



```
forms.py
admin.py
views.py
inicio.html

cursod > boletin > forms.py
1  from django import forms
2
3  from .models import Registrado
4
5  class RegModelForm(forms.ModelForm):
6      class Meta:
7          model = Registrado
8          fields = ["nombre", "email"]
9
10     def clean_email(self):
11         print([self.cleaned_data])
12         return "email@email.com"
13
14     class RegForm(forms.Form):
15         nombre = forms.CharField(max_length=100)
16         email = forms.EmailField()
```

Esto provoca que al registrar un nuevo usuario quede como email@email.com



The screenshot shows a Django admin list view for "Registrado". At the top, a green bar displays the message "El registrado "email@email.com" fue agregado correctamente.". Below is a table with columns: Acción, EMAIL, NOMBRE, and TIMESTAMP. One row is shown: "email@email.com" with timestamp "1 de Mayo de 2021 a las 18:15".

```

forms.py  X  admin.py  views.py  inicio.html
cursod > boletin > forms.py
1  from django import forms
2
3  from .models import Registrado
4
5  class RegModelForm(forms.ModelForm):
6      class Meta:
7          model = Registrado
8          fields = ["nombre", "email"]
9
10     def clean_email(self):
11         email = self.cleaned_data.get("email")
12         if not ".edu" in email:
13             raise forms.ValidationError("Por favor utiliza un email con la extensión .EDU")
14         return email
15
16 class RegForm(forms.Form):
17     nombre = forms.CharField(max_length=100)
18     email = forms.EmailField()

```

Estos cambios provocan lo siguiente en el formulario.

Añadir registrado

Por favor corrija el siguiente error.

Nombre:	<input type="text"/>
Email:	<input type="text" value="karlita@email.com"/>

Si ponemos EDU@mail.com se lo traga, queremos que sea la extensión, para ello:

```

forms.py  X  admin.py  views.py  inicio.html
cursod > boletin > forms.py
1  from django import forms
2
3  from .models import Registrado
4
5  class RegModelForm(forms.ModelForm):
6      class Meta:
7          model = Registrado
8          fields = ["nombre", "email"]
9
10     def clean_email(self):
11         email = self.cleaned_data.get("email")
12         email_base, proveedor = email.split("@")
13         dominio, extension = proveedor.split(".")
14         if not extension == ".edu":
15             raise forms.ValidationError("Por favor utiliza un email con la extensión .EDU")
16         return email
17
18 class RegForm(forms.Form):
19     nombre = forms.CharField(max_length=100)
20     email = forms.EmailField()

```

Si ahora ponemos edu@yo.com pues no se lo traga.

Añadir registrado

Por favor corrija el siguiente error.

Nombre:

Email: Por favor utiliza un email con la extensión .EDU

Con la extensión correcta si nos deja meterlo. Así tenemos nuestras validaciones personales.

✓ El registrado "yo@yo.edu" fue agregado correctamente.

Seleccione registrado a modificar

Acción: Ir seleccionados 0 de 8

<input type="checkbox"/> EMAIL	NOMBRE
<input type="checkbox"/> yo@yo.edu	<input type="text"/>

Podríamos hacer lo mismo con los nombres.

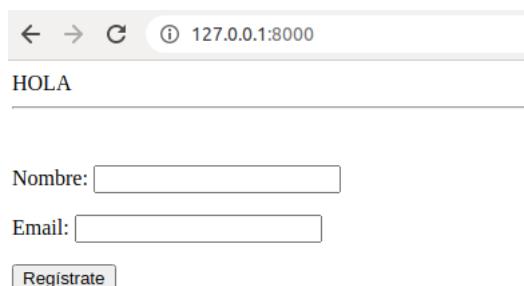
```
def clean_email(self):
    email = self.cleaned_data.get("email")
    email_base, proveeder = email.split("@")
    dominio, extension = proveeder.split(".")
    if not extension == "edu":
        raise forms.ValidationError("Por favor utiliza un email con la extensión .EDU")
    return email
def clean_nombre(self):
    nombre = self.cleaned_data.get("nombre")
    #validaciones
    return nombre
```

18. Contexto en la vista, plantilla

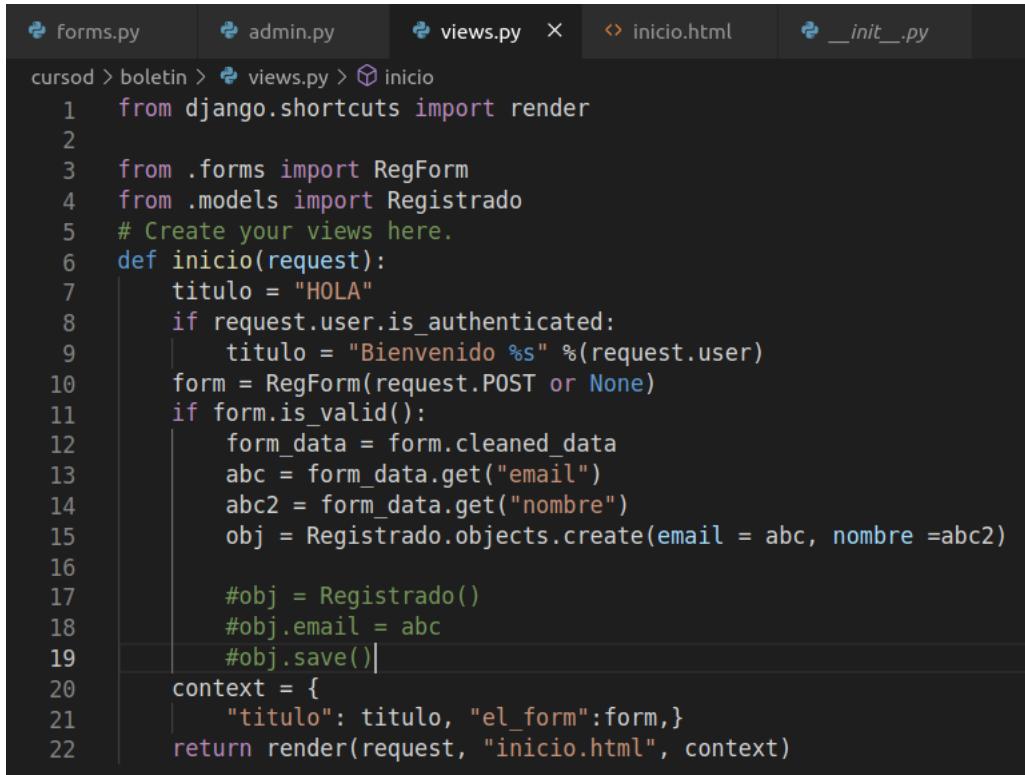
```
forms.py admin.py views.py X inicio.html __init__.py
cursod > boletin > views.py > inicio
1   from django.shortcuts import render
2
3   from .forms import RegForm
4   from .models import Registrado
5   # Create your views here.
6   def inicio(request):
7       titulo = "HOLA"
8       form = RegForm(request.POST or None)
9       if form.is_valid():
10           form_data = form.cleaned_data
11           abc = form_data.get("email")
12           abc2 = form_data.get("nombre")
13           obj = Registrado.objects.create(email = abc, nombre =abc2)
14
15           #obj = Registrado()
16           #obj.email = abc
17           #obj.save()
18           context = {
19               "titulo": titulo, "el_form":form,}
20           return render[request, "inicio.html", context]
```

```
forms.py admin.py views.py X inicio.html __init__.py
cursod > templates > inicio.html > form
1   {{titulo}}
2   <hr/>
3   <br/>
4   <form method ='POST' action ="">{% csrf_token %}
5   {{ el_form.as_p }}
6   <input type='submit' value= 'Regístrate' />
7   </form>
```

Recargamos la página.



Vamos a hacer ahora que cuando un usuario esté registrado no muestre solo el HOLA, cambie el mensaje, así lo hacemos dinámico. Aquí se produce otra divergencia con el vídeo, ya que Karlita hace `request.user.is_authenticated()`, y nos da error, porque bool no es collable. Nosotros lo pondríamos así:



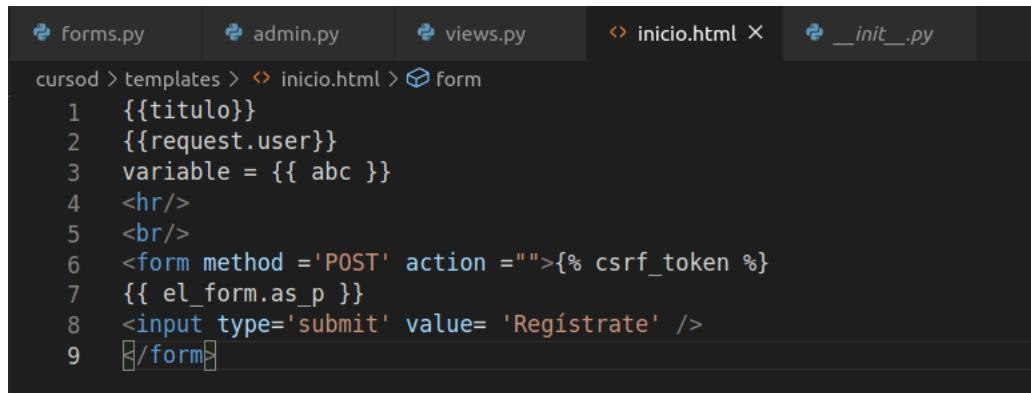
```
cursod > boletin > views.py > inicio
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  from .models import Registrado
5  # Create your views here.
6  def inicio(request):
7      titulo = "HOLA"
8      if request.user.is_authenticated:
9          titulo = "Bienvenido %s" %(request.user)
10     form = RegForm(request.POST or None)
11     if form.is_valid():
12         form_data = form.cleaned_data
13         abc = form_data.get("email")
14         abc2 = form_data.get("nombre")
15         obj = Registrado.objects.create(email = abc, nombre =abc2)
16
17         #obj = Registrado()
18         #obj.email = abc
19         #obj.save()
20     context = {
21         "titulo": titulo, "el_form":form,}
22     return render(request, "inicio.html", context)
```

Si vamos a la página:

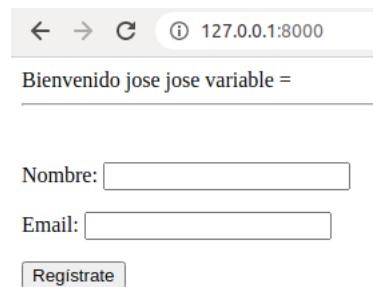


Podemos ver que pone nuestro nombre porque tenemos la sesión, si entramos como modo incógnito saldrá de nuevo el HOLA.

Hacemos unos cambios en el html y vemos la página luego.



```
cursod > templates > inicio.html > form
1  {{titulo}}
2  {{request.user}}
3  variable = {{ abc }}
4  <hr/>
5  <br/>
6  <form method ='POST' action =">{{ csrf_token }}
7  {{ el_form.as_p }}
8  <input type='submit' value= 'Regístrate' />
9  </form>
```



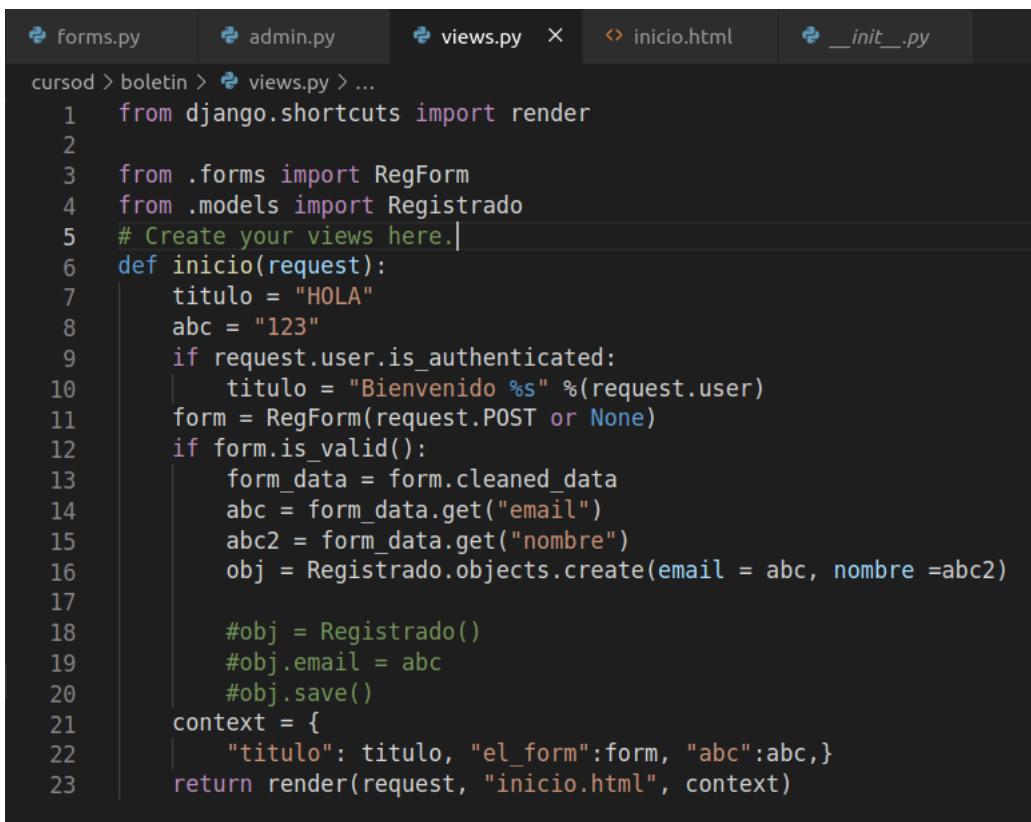
127.0.0.1:8000

Bienvenido jose jose variable =

Nombre:

Email:

Es el efecto que tiene en nuestra página. Lo podemos separar con
 para leerlo mejor. Para que abc aparezca con algún valor tenemos que declararlo en views.py de la siguiente forma:



```
cursod > boletin > views.py > ...
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  from .models import Registrado
5  # Create your views here.
6  def inicio(request):
7      titulo = "HOLA"
8      abc = "123"
9      if request.user.is_authenticated:
10          titulo = "Bienvenido %s" %(request.user)
11          form = RegForm(request.POST or None)
12          if form.is_valid():
13              form_data = form.cleaned_data
14              abc = form_data.get("email")
15              abc2 = form_data.get("nombre")
16              obj = Registrado.objects.create(email = abc, nombre =abc2)
17
18              #obj = Registrado()
19              #obj.email = abc
20              #obj.save()
21          context = {
22              "titulo": titulo, "el_form":form, "abc":abc,}
23          return render(request, "inicio.html", context)
```

Bienvenido jose
jose
variable = 123

Nombre:

Email:

Podemos ver ahora que si muestra valor y lo tenemos separado.

19. ModelForm en la vista

```

cursod > boletin > views.py > inicio
1   from django.shortcuts import render
2
3   from .forms import RegForm, RegModelForm
4   from .models import Registrado
5   # Create your views here.
6   def inicio(request):
7       titulo = "HOLA"
8       abc = "123"
9       if request.user.is_authenticated:
10           titulo = "Bienvenido %s" %(request.user)
11           form = RegModelForm(request.POST or None)
12           if form.is_valid():
13               instance = form.save(commit=False)
14               print [instance]
15               #form_data = form.cleaned_data
16               #abc = form_data.get("email")
17               #abc2 = form_data.get("nombre")
18               #obj = Registrado.objects.create(email = abc, nombre =abc2)
19
20               #obj = Registrado()
21               #obj.email = abc
22               #obj.save()
23           context = {
24               "titulo": titulo, "el_form":form, "abc":abc,}
25           return render(request, "inicio.html", context)

```

Antes de seguir hacemos la siguiente corrección en forms.py, cambiamos self. por sed.

```

def clean_nombre(sed):
    nombre = sed.cleaned_data.get("nombre")
    #validaciones
    return nombre

```

Nos vamos para la página y podemos ver que si ponemos mala extensión salta el error que pusimos de validación.

Bienvenido jose
jose

Nombre:

- Por favor utiliza un email con la extensión .EDU

Email:

Vamos a la consola y podemos ver la instancia, ya que en el modelo lo tenemos puesto así.

```
[02/May/2021 09:54:17] "POST / HTTP/1.1" 200 598
holi@holi.edu
[02/May/2021 09:57:36] "POST / HTTP/1.1" 200 512
```

Añadimos en views.py la siguiente línea: print (instance.timestamp) guardamos, recagramos la página.

```
Quit the server with CONTROL-C.
holi@holi.edu
None
[02/May/2021 09:59:57] "POST / HTTP/1.1" 200 512
```

Ahora aparece None, porque no hay timestamp porque el objeto no está guardado en la BD porque el commit lo tenemos en False. Añadimos:

```
form = RegModelForm(request.POST or None)
if form.is_valid():
    instance = form.save(commit=False)
    instance.save()
```

Ahora si tenemos timestamp

```
Quit the server with CONTROL-C.
holi@holi.edu
2021-05-02 10:02:16.531426+00:00
[02/May/2021 10:02:17] "POST / HTTP/1.1" 200 512
```

Como establecimos anteriormente, nombre no es obligatorio, por lo que añadiremos la siguiente línea:

```
instance = form.save(commit=False)
if not instance.nombre:
    instance.nombre = "PERSONA"
```

Si vamos a nuestra página y dejamos el nombre en blanco al registrarnos:

Acción: ----- Ir seleccionados 0 de 10

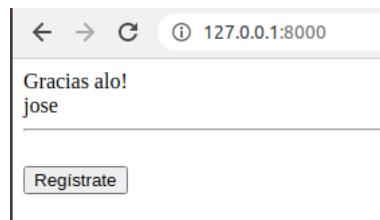
EMAIL	NOMBRE
<input type="checkbox"/> alo@alo.edu	<input type="checkbox"/> PERSONA

Obviamente se rellena por defecto el valor que le damos.

Cambiamos de sitio nuestro context y podemos abajo del save otro context nuevo. Divergencia con el vídeo, no podemos poner solo nombre, tenemos que poner instance.nombre para que no nos dé error.

```
if not instance.nombre:  
    instance.nombre = "PERSONA"  
instance.save()  
  
context = [  
    "titulo": "Gracias %s!" %(instance.nombre)  
]
```

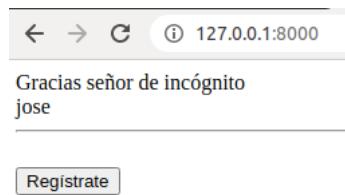
Vamos de nuevo para nuestra página, hacemos un nuevo registro y al darle a registrarnos nos sale:



El problema es que si no ponemos ningún nombre, el Gracias sale sin nada más que la exclamación, vamos a solucionar eso poniendo las siguientes líneas en views.py:

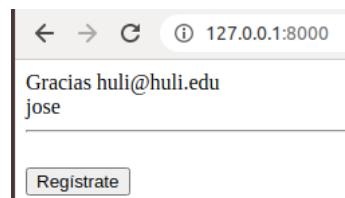
```
if not nombre:  
    context ={ "titulo": "Gracias señor de incógnito"}
```

Si volvemos a registrar a alguien sin nombre solo con el correo, pues tendríamos:



En vez de poner un mensaje podríamos poner el campo email, ya que es obligatorio y nos daría las gracias y el email del usuario. Otra divergencia, no es solo email como pone el vídeo, si no instance.email.

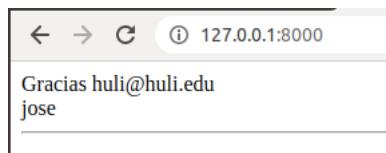
```
if not instance.nombre:  
    #context = { "titulo": "Gracias señor de incógnito"}  
    context ={"titulo": "Gracias %s" %[instance.email]}
```



Aún nos queda el botón puesto después del registro, nos vamos a templates, inicio.html y...

```
cursod > templates > inicio.html > ...  
1   {{titulo}}<br/>  
2   {{request.user}}<br/>  
3   <hr/>  
4   <br/>  
5   {% if form %}  
6   <form method='POST' action ="">{{ csrf_token }}  
7   {{ el_form.as_p }}  
8   <input type='submit' value= 'Regístrate' />  
9   </form>  
10  [% endif %]
```

Ya no hay botón.



20. Custom Form para Contacto

Con esto tenemos un formulario con más espacio.

```
class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()
    mensaje = forms.CharField(widget=forms.Textarea)
```

Creamos ahora en views.py una vista para contactos.

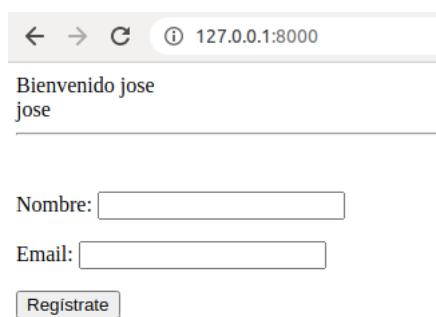
```
39
40     def contact(request):
41         form = ContactForm(requests.POST or None)
42
43         context ={"form":form,}
44         return render(request, "forms.html", context )
```

Ahora nos vamos a templates y creamos el forms.py. Copiamos el contenido de inicio y lo pegamos en forms.html, luego nos vamos a URLs y añadimos la URL para los contactos.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.inicio, name='inicio')
    path('contact/', views.contact, name='contact')

]
```

Haciendo esto, nuestro formulario desaparecerá, tenemos que quitar los if del inicio. Una vez hecho, nos quedará la página así:



Podemos ver nuestro formulario.

127.0.0.1:8000/contact/

jose

Nombre:

Email:

Mensaje:

Sería adecuado añadir validaciones para esto también, además hemos hecho que el nombre no sea obligatorio.

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        print(form.cleaned_data)
    context = {"form": form}
    return render(request, "forms.html", context)
```

127.0.0.1:8000/contact/

jose

Nombre:

- Por favor utiliza un email con la extensión .EDU

Email: huli@huli.ca

Mensaje:

mejoramos el ContactForm.

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        email = form.cleaned_data.get("email")
        mensaje = form.cleaned_data.get("mensaje")
        nombre = form.cleaned_data.get("nombre")
        print (email, mensaje, nombre)
    context ={"form":form,}
    return render(request, "forms.html", context)
```

Comprobamos.

```
[02/May/2021 16:05:37] "POST /contact/ HTTP/1.1" 200 603
hola@gola.com vf Karlita
[02/May/2021 16:05:42] "POST /contact/ HTTP/1.1" 200 603
```

Vemos que se guarda en el símbolo del sistema.

Si tenemos muchos campos, podemos poner el código así:

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        for key in form.cleaned_data:
            print (key)
            print form.cleaned_data.get(key)
        # email = form.cleaned_data.get("email")
        #mensaje = form.cleaned_data.get("mensaje")
        #nombre = form.cleaned_data.get("nombre")
        #print (email, mensaje, nombre)
    context ={"form":form,}
    return render(request, "forms.html", context)
```

Si volvemos a hacer el cuestionario, esta vez en la consola sale así.

```
vt
[02/May/2021 16:10:02] "POST /contact/ HTTP/1.1" 200 603
nombre
Karlita
email
hola@gola.com
mensaje
vf
[02/May/2021 16:10:04] "POST /contact/ HTTP/1.1" 200 603
```

Otra forma de escribir esa parte del código es esta. Divergencia con el vídeo, no es iteritems, es items.

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        for key, value in form.cleaned_data.items():
            print(key, value)
        #for key in form.cleaned_data:
        #    print (key)
        #    # print (form.cleaned_data.get(key))
        #email = form.cleaned_data.get("email")
        #mensaje = form.cleaned_data.get("mensaje")
        #nombre = form.cleaned_data.get("nombre")
        #print (email, mensaje, nombre)
    context ={"form":form,}
    return render(request, "forms.html", context)
```

Vemos que sale ahora mejor.

```
[02/May/2021 16:14:06] "POST /contact/ HTTP/1.1" 200 603
nombre Karlita
email hola@gola.com
mensaje vf
[02/May/2021 16:14:07] "POST /contact/ HTTP/1.1" 200 603
```

21. Configurar Email

Para llevar a cabo el testeo con el correo necesitamos configurar unas cosas antes, nos vamos a settings.py. Ponemos nuestros datos personales. (Se cambiarán antes de subirlos, solo será para testeo)

```
ALLOWED_HOSTS = []

EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'xemuelvazquez@gmail.com'
EMAIL_HOST_PASSWORD =
EMAIL_PORT = 587
EMAIL_USE_TLS = True
```

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        #for key, value in form.cleaned_data.items():
        #    print(key, value)
        #for key in form.cleaned_data:
        #    print (key)
        #    # print (form.cleaned_data.get(key))
        email = form.cleaned_data.get("email")
        mensaje = form.cleaned_data.get("mensaje")
        nombre = form.cleaned_data.get("nombre")
        #print (email, mensaje, nombre)
    context ={"form":form,}
    return render(request, "forms.html", context)
```

Ahora importamos en views.py

```
cursod > boletin > views.py > contact
1   from django.conf import settings
2   from django.core.mail import send_mail
3
```

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        #for key, value in form.cleaned_data.items():
        #    print(key, value)
        #for key in form.cleaned_data:
        #    print (key)
        #    # print (form.cleaned_data.get(key))
        email = form.cleaned_data.get("email")
        mensaje = form.cleaned_data.get("mensaje")
        nombre = form.cleaned_data.get("nombre")
        send_mail(asunto,
                  mensaje_email,
                  email_from,
                  [email_to],
                  fail_silently = False)
        #print (email, mensaje, nombre)
    context = {"form":form,}
    return render(request, "forms.html", context)
```

Debemos desbloquear el captcha si usamos gmail. Con esto podremos hacer testing:

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        #for key, value in form.cleaned_data.items():
        #    print(key, value)
        #for key in form.cleaned_data:
        #    print (key)
        #    # print (form.cleaned_data.get(key))
        form_email = form.cleaned_data.get("email")
        form_mensaje = form.cleaned_data.get("mensaje")
        form_nombre = form.cleaned_data.get("nombre")
        asunto = 'Form e Contacto'
        email_from = settings.EMAIL_HOST_USER
        email_to = [email_from, "otroemail@gamil.com"]
        email_mensaje = "%s: %s enviado por %s" %(form_nombre, form_mensaje, form_email)
        send_mail(asunto,
                  email_mensaje,
                  email_from,
                  email_to,
                  fail_silently = False)
        #print (email, mensaje, nombre)
    context = {"form":form,}
    return render(request, "forms.html", context)
```

Una vez lo tenemos así, si metemos nuestros datos, nos saltará el siguiente error:

```

Request Method: POST
Request URL: http://127.0.0.1:8000/contact/
Django Version: 3.2
Exception Type: SMTPAuthenticationError
Exception Value: (535, b'5.7.8 Username and Password not accepted. Learn more at\nhttps://support.google.com/mail/?p=BadCredentials q10sm5325431wre.92 - gsmtp')
Exception Location: /usr/lib/python3.6/smtplib.py, line 642, in auth
Python Executable: /home/jose/Dокументos/Karita/curso_django/bin/python3
Python Version: 3.6.9
Python Path: ['/home/jose/Dокументos/Karita/cursod',
              '/usr/lib/python36.zip',
              '/usr/lib/python3.6',
              '/usr/lib/python3.6/lib-dynload',
              '/home/jose/Dокументos/Karita/curso_django/lib/python3.6/site-packages']
Server time: Sun, 02 May 2021 16:39:24 +0000

```

Traceback [Switch to copy-and-paste view](#)

Si ponemos el fail silently a true lo intentará, pero no saldrá error, pero no quiere decir que se haya hecho.

22. Configuración de Archivos Estáticos

Tendremos 2 servidores, uno de producción y otro con archivos estáticos, y tendremos que hacer que ambos se comuniquen. Esto no será válido para producción.

Nos aseguramos que tenemos static files.

```

17 INSTALLED_APPS: list
18 INSTALLED_APPS = [
19     'django.contrib.admin',
20     'django.contrib.auth',
21     'django.contrib.contenttypes',
22     'django.contrib.sessions',
23     'django.contrib.messages',
24     'django.contrib.staticfiles',
25     'boletin',

```

Miramos a ver como se define nuestra URL.

```

2 # Static files (CSS, JavaScript, Images)
3 # https://docs.djangoproject.com/en/3.2/howto/static-files/
4 STATIC_URL = '/static/'
5

```

Para que django busque en un directorio los archivos estáticos añadimos esto casi al final de settings.py. Divergencia con el vídeo, el directorio es como se muestra en la siguiente imagen (lo mismo pasa cuando pongamos STATIC ROOT y MEDIA):

```

STATICFILES_DIRS = [
    BASE_DIR / "static_pro" / "static",
    #'/var/www/static/',
]

```

Comentaremos la ruta absoluta para que podamos abrir el proyecto en otro ordenador.

Vamos a poner la ruta donde van a convivir los archivos estáticos, será así:

```

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'
STATIC_URL = '/media/'
STATICFILES_DIRS = [
    BASE_DIR / "static_pro" / "static",
    #'var/www/static/',
]

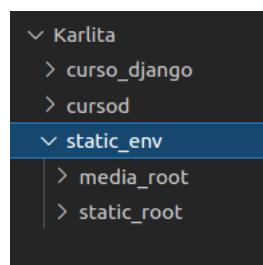
STATIC_ROOT = BASE_DIR / "..." / "static_env" / "static_root"
MEDIA_ROOT = BASE_DIR / "..." / "static_env" / "media_root"

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

```

Creamos los directorios correspondientes.



```

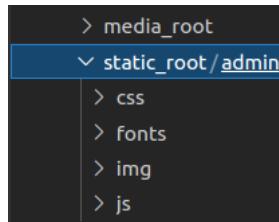
5 """
6 from django.contrib import admin
7 from django.urls import path
8 from django.conf import settings
9         from django.conf.urls.static import static
10
11         from boletin import views
12         #from boletin.views inicio
13
14         urlpatterns = [
15             path('admin/', admin.site.urls),
16             path('', views.inicio, name='inicio'),
17             path('contact/', views.contact, name='contact'),
18         ]
19         if settings.DEBUG:
20
21
22
23
24
25
26
27
28
29         if settings.DEBUG:
30             urlpatterns += static(settings.STATIC_URL, document_root= settings.STATIC_ROOT)
31             urlpatterns += static(settings.MEDIA_URL, document_root= settings.MEDIA_ROOT)

```

Una vez tenemos todo esto configurado, lo tenemos listo. Nos vamos a la consola y ejecutamos el siguiente comando:

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ python3 manage.py collectstatic
128 static files copied to '/home/jose/Documentos/Karlita/curso_django/static_env/static_root'.
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$
```

Podemos ver que se han copiado los archivos estáticos. En el vídeo la copia aparece de otra forma, pero podemos ver que se hace bien, porque además, tenemos los archivos en static_root tal y como salen en el vídeo.



23. Configuración Bootstrap

Vamos a meternos en la página de Bootstrap y vamos a descargar la plantilla que dice Navbar static, ya que es una plantilla prediseñada en vez de usar el Bootstrap CDN. Pero antes de descargar le damos click derecho e inspeccionar.

```
← → C ⓘ view-source:https://getbootstrap.com/docs/5.0/examples/navbar-static/
Ajuste de linea □
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <meta name="description" content="">
7     <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
8     <meta name="generator" content="Hugo 0.82.0">
9     <title>Top navbar example · Bootstrap v5.0</title>
10
11    <link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/navbar-static/">
12
13
14
15    <!-- Bootstrap core CSS -->
16    <link href="/docs/5.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-eOJMySd53ii+sc0/bJGFsiCz+5NDVN2yr8+0RDqr0Ql0h+rP48cklpbzKgwra6" crossorigin="anonymous">
17
18    <!-- Favicons -->
19    <link rel="apple-touch-icon" href="/docs/5.0/assets/img/favicons/apple-touch-icon.png" sizes="180x180">
20    <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png">
21    <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png">
22    <link rel="manifest" href="/docs/5.0/assets/img/favicons/manifest.json">
23    <link rel="mask-icon" href="/docs/5.0/assets/img/favicons/safari-pinned-tab.svg" color="#7952b3">
24    <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon.ico">
25    <meta name="theme-color" content="#7952b3">
26
27
28    <style>
29      .bd-placeholder-img {
30        font-size: 1.125rem;
31        text-anchor: middle;
32        -webkit-user-select: none;
33        -moz-user-select: none;
34        user-select: none;
35      }
36
37      @media (min-width: 768px) {
38        .bd-placeholder-img-lg {
39          font-size: 3.5rem;
40        }
41      }
42    </style>
43
44
45    <!-- Custom styles for this template -->
46    <link href="navbar-top.css" rel="stylesheet">
47  </head>
48  <body>
49
50    <nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
51      <div class="container-fluid">
52        <a class="navbar-brand" href="#">Top navbar
```

Podemos ver así como funciona y se compone. Dicho código lo copiamos y lo pegamos en un archivo que vamos a crear dentro de templates llamado base.html

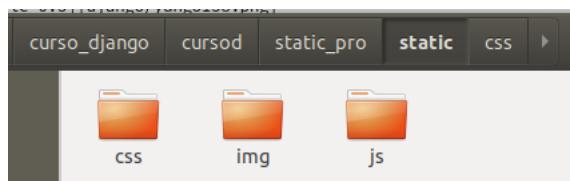
This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs >](#)

Aunque nos falta el estilo.

Nos vamos a donde pone Bootstrap core CSS, abrimos el enlace y ese contenido hacemos click derecho y lo guardamos como, en nuestra carpeta static, crearemos una carpeta llamada css y ahí lo guardamos.

Una vez tenemos guardado nuestro archivo, creamos las carpetas para ordenar futuros archivos.



Para lograr que se renderice tenemos que hacer unas configuraciones. Ponemos en base.html:

```
<!doctype html>
[% load static %]
```

Además para renderizar imágenes en la documentación de Django podemos encontrar:

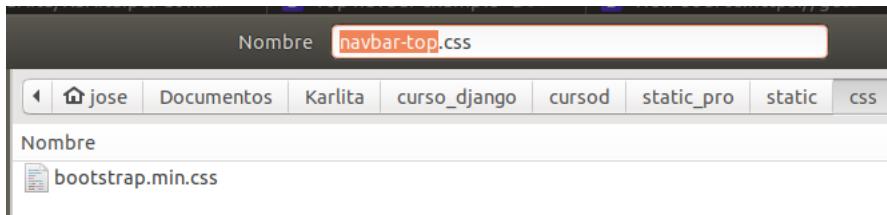
```
{% load static %}

```

Haremos lo mismo para nuestras hojas de estilo.

Adaptamos el directorio. Hacemos lo mismo con navbar-top.css, guardamos el archivo en css.

```
<!-- Bootstrap core CSS -->
<link href="{% static 'css/bootstrap.min.css' %}">
```



Y hacemos lo mismo de adaptar el directorio.

```
<!-- Custom styles for this template -->
<link href="{% 'navbar-top.css' %}" rel="stylesheet">
</head>
<body>
```

Si volvemos a nuestra página y recargamos, vemos:

Cerramos el servidor y hacemos el comando:

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ python3 manage.py collectstatic
130 static files copied to '/home/jose/Documentos/Karlita/curso_django/static_env/static_root'.
```

Con la importación podremos ver que tenemos las hojas del estilo en su carpeta correspondiente. Descargamos ahora el js.

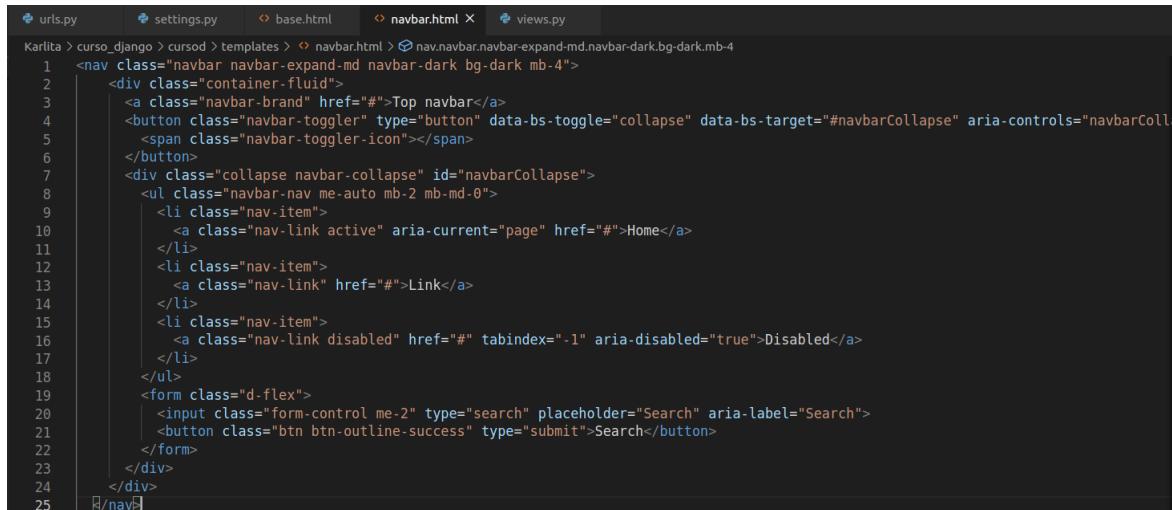
```
<script src="/docs/5.0/dist/js/bootstrap.bundle.min.js"
```

De ese enlace sacamos el archivo js y lo metemos en su carpeta correspondiente. Después ejecutamos de nuevo collectstatic, luego encendemos el server y podremos comprobar que funciona sin problema alguno.

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ python3 manage.py collectstatic
You have requested to collect static files at the destination
location as specified in your settings:
    /home/jose/Documentos/Karlita/curso_django/static_env/static_root
This will overwrite existing files!
Are you sure you want to do this?
Type 'yes' to continue, or 'no' to cancel: yes
1 static file copied to '/home/jose/Documentos/Karlita/curso_django/static_env/static_root', 130 unmodified.
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$
```

24. Plantillas: Herencia, Include Tag, Blocks

Crearemos navbar.html en nuestra carpeta templates. Cortamos en base.html la parte del navbar y lo pegamos en el archivo que acabamos de crear.



```
Karita > curso_django > cursod > templates > navbar.html > nav.navbar.navbar-expand-md.navbar-dark.bg-dark.mb-4
1  <nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
2    <div class="container-fluid">
3      <a class="navbar-brand" href="#">Top navbar</a>
4      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse">
5        <span class="navbar-toggler-icon"></span>
6      </button>
7      <div class="collapse navbar-collapse" id="navbarCollapse">
8        <ul class="navbar-nav me-auto mb-2 mb-md-0">
9          <li class="nav-item">
10            <a class="nav-link active" aria-current="page" href="#">Home</a>
11          </li>
12          <li class="nav-item">
13            <a class="nav-link" href="#">Link</a>
14          </li>
15          <li class="nav-item">
16            <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
17          </li>
18        </ul>
19        <form class="d-flex">
20          <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
21          <button class="btn btn-outline-success" type="submit">Search</button>
22        </form>
23      </div>
24    </div>
25  </nav>
```

En la parte que hemos cortado de base.html pondremos la etiqueta { % include "navbar.html" %}. Guardamos y vamos a nuestra página.



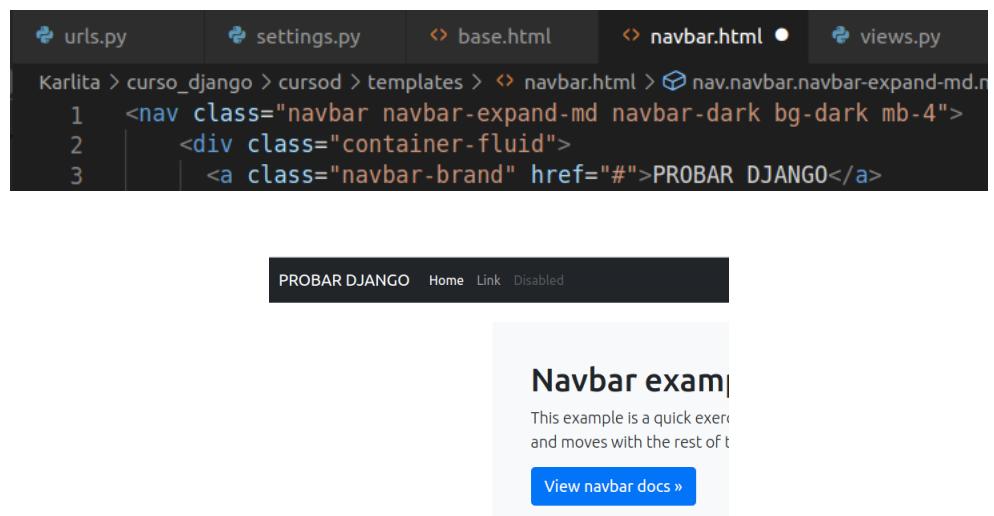
Top navbar Home Link Disabled

Navbar example

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

Y observamos que la barra de navegación se conserva.



PROBAR DJANGO Home Link Disabled

Navbar example

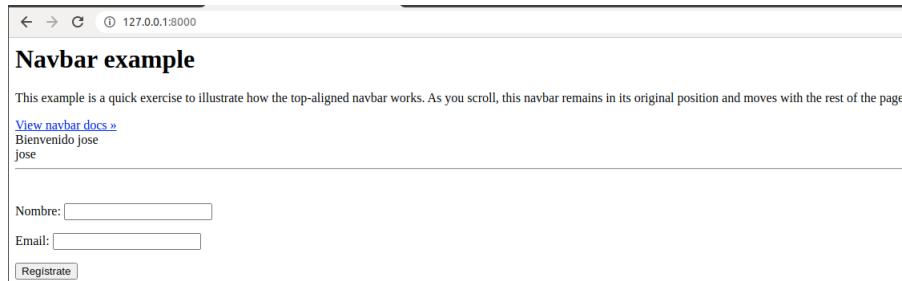
This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

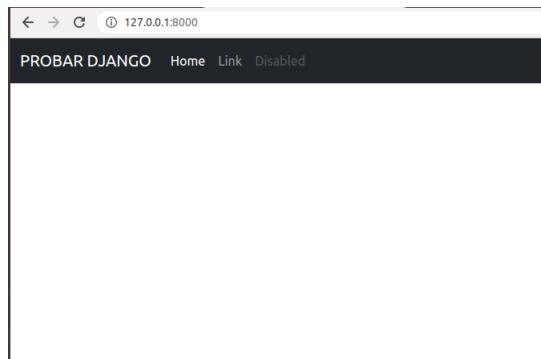
Movemos esta parte a inicio.html, y ponemos las etiquetas correspondientes en su lugar para que cargue la página sin problemas.

```
<main class="container">
  <div class="bg-light p-5 rounded">
    <h1>Navbar example</h1>
    <p class="lead">This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.</p>
    <a class="btn btn-lg btn-primary" href="/docs/5.0/components/navbar/" role="button">View navbar docs &raquo;</a>
  </div>
</main>
```

Como ese contenido está en inicio, tenemos que volver a poner inicio.html en el render de views.py.



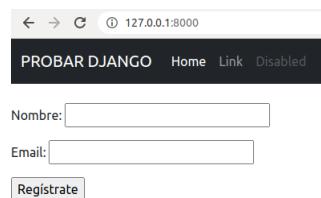
Para que se vea correctamente debemos añadir un extense a inicio.html



Solo carga la navbar, necesitamos hacer lo siguiente para que aparezca todo lo que queremos. Ponemos otra vez las etiquetas correspondientes que son las mismas que las que usamos hace un momento.

```
{% block content %}
<form method='POST' action="">{% csrf_token %}
{{ el_form.as_p }}
<input type='submit' value='Regístrate' />
</form>
{% endblock %}
```

Ahora sí aparece el formulario.

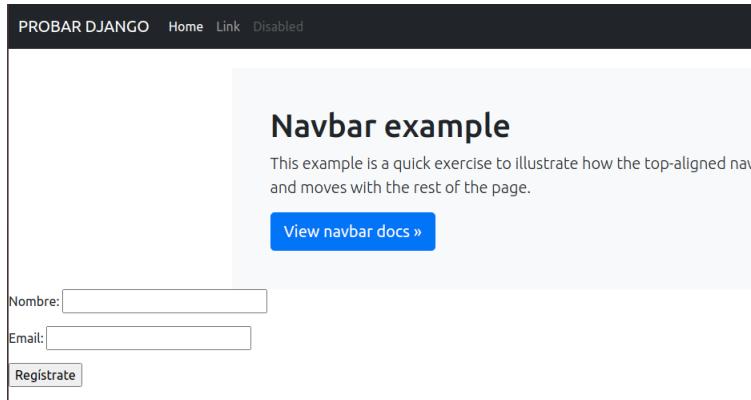


Nos falta el rectángulo gris y el botón azul, para ello añadimos las etiquetas block en base.html y en inicio.html respectivamente.

```
{% include "navbar.html" %}  
{% block container-fluid %}  
{% endblock %}  
  
{% block content %}  
{% endblock %}
```

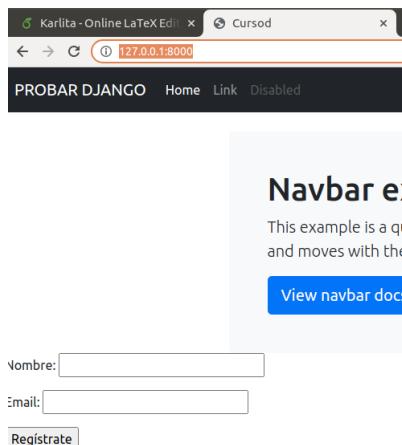
```
{% block container-fluid %}  
  
<main class="container">  
  <div class="bg-light p-5 rounded">  
    <h1>Navbar example</h1>  
    <p class="lead">This example is a quick exercise to illustrate how the  
    nav bar moves with the rest of the page.  
    <a class="btn btn-lg btn-primary" href="/docs/5.0/components/navbar/">  
      View navbar docs »  
    </a>  
  </div>  
</main>  
  
{% endblock %}
```

Arreglado.



Podemos en título añadir este bloque y luego esos bloques los ponemos en inicio también.

```
<meta name="generator" content="Hugo 0.82.0">  
<title>{{ block head_title }}{{ endblock }}{{ cursod }}</title>
```



Vemos que pone en la pestaña cursod tal y como lo tenemos puesto en el block. Si en inicio ponemos Bienvenidos entre los bloques saldrá Bienvenidos | cursod.

Si ponemos cursod entre los bloques de base, no aparecerá en la pestaña. Pero si luego en inicio ponemos el bloque super volverá a salir perfectamente.

"/>

Seguimos quitando cosas de base.html y pegamos esta parte en un archivo nuevo llamado head_css.html

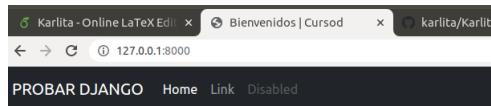
3
4 <!-- Favicons -->
5 <link rel="apple-touch-icon" href="/docs/5.0/assets/img/favicons/apple-touch-icon.png" sizes="180x180">
6 <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png">
7 <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png">
8 <link rel="manifest" href="/docs/5.0/assets/img/favicons/manifest.json">
9 <link rel="mask-icon" href="/docs/5.0/assets/img/favicons/safari-pinned-tab.svg" color="#7952b3">
10 <link rel="icon" href="/docs/5.0/assets/img/favicons/favicon.ico">
11 <meta name="theme-color" content="#7952b3">

"/>

Como aquí son archivos estáticos tenemos que ponerle a ese archivo la etiqueta de load static. Luego en base tenemos que incluir el archivo nuevo con la etiqueta include.

Además podemos poner bloques dentro de bloques:

"/>

"/>


INICIO!!

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains at the top of the page and moves with the rest of the page. You can also see how the sidebar's position remains fixed relative to the top of the page.

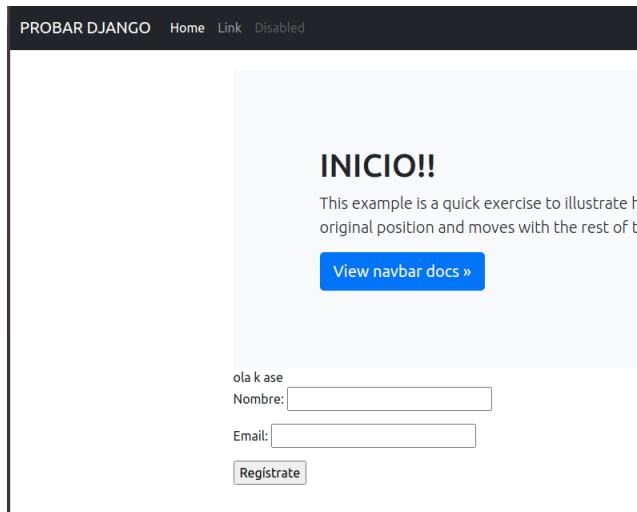
[View navbar docs »](#)

Cabe mencionar las divergencias de que nosotros no usamos jumbotron, si no que estamos usando la otra clase light p-5, además distribuimos los bloques tal y como se muestran en las imágenes para que no haya errores.

Si comentamos el contenido del bloque, desaparecerá de la página. Y si ponemos otra vez el bloque container-fluid con el block super, podemos poner todo el contenido.

```
{% block container-fluid %}  
{{ block.super }}  
ola k ase  
{% endblock %} |
```

Lo mismo con el bloque content, ponemos otro block super para él, y el resultado sería el siguiente.



25. Django Crispy Forms

Crispy form es una aplicación que nos permite crear formularios sin mucho código. Debemos instalarlo para usarlo, haciendo uso de `pip install - -upgrade django-crispy-forms`

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ pip install --upgrade django-crispy-forms  
Collecting django-crispy-forms  
  Downloading https://files.pythonhosted.org/packages/2a/df/f4618a94c3bf1748c6876bc178f2fb0ec7a9fc6ef17a406c80009bb130d/django_crispy_forms-1.11.2-py3-none-any.whl  
    100% |████████████████████████████████| 122KB 1.3MB/s  
Installing collected packages: django-crispy-forms  
Successfully installed django-crispy-forms-1.11.2  
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$
```

Ponemos la app de crispy.

```
INSTALLED_APPS = [  
    #apps de django  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    #apps de terceros  
    'crispy_forms',]  
    #mis apps  
    'boletin',
```

Hacemos por si acaso una migración.

```
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ python3 manage.py makemigrations  
No changes detected  
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$ python3 manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, boletin, contenttypes, sessions  
Running migrations:  
  No migrations to apply.  
(curso_django) jose@jose-Lenovo-ideapad-310-15IKB:~/Documentos/Karlita/curso_django/cursod$
```

Añadimos en settings.py también esta línea: CRISPY_TEMPLATE_PACK = 'bootstrap4' divergencia con el vídeo, Karlita usa el 3, y guardamos. Nos vamos ahora a inicio.html

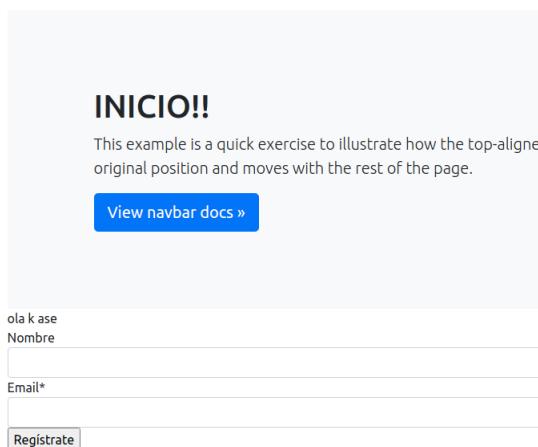


```
urls.py      settings.py      base.html      head_css.html      inicio.html ●
Karlita > curso_django > cursod > templates > inicio.html ...
1   {% extends "base.html" %} 
2   {% load crispy_forms_tags %} 
3
```

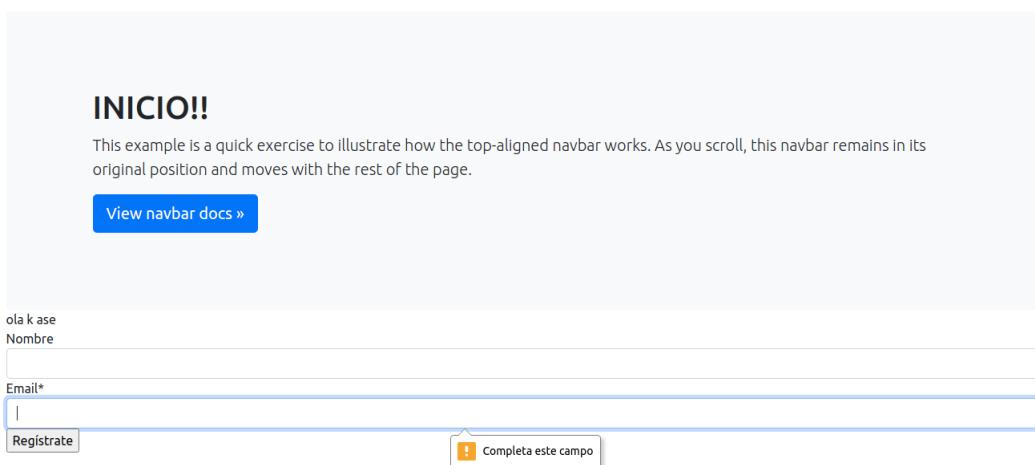
En inicio también cambiamos el_form.as_p por:

```
26  <hr/>
27  <br/>
28  {% block content %}
29  {{ block.super }}
30  <form method='POST' a
31  {{ el_form|crispy }} 
32  <input type='submit' value='>
```

Así cambia nuestro formulario.

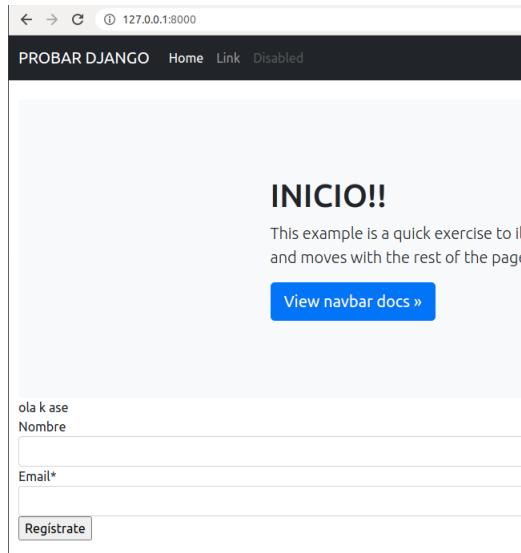


La página cambió de diseño

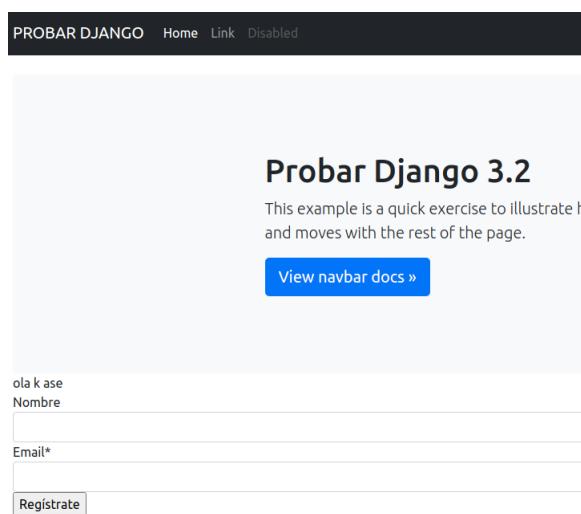


26. Estilo: Bootstrap 1

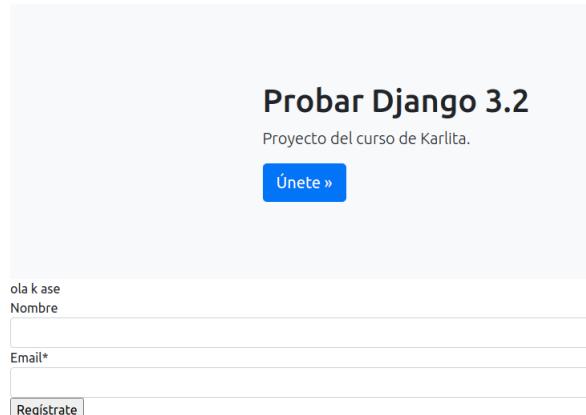
Añadimos la palabra fluid a la clase container, para cambiar los márgenes, la página se muestra ahora a la izquierda.



En nuestro caso con el navbar ya está hecho con fluid, a diferencia que en el vídeo de karlita. Cambiamos el Inicio pocho por algo con más sentido.



Hacemos lo mismo con el párrafo de abajo de Probar Django 3.2, ponemos otra cosa distinta. El texto del botón azul también lo cambiamos por otra cosa.



Vamos a hacer espacio.

```
{% block container-content %}
<div class="row">
|   <div class='col-sm-6'>
<main class="container">
|   <div class="bg-light p-5 rounded">
|       <h1>Probar Django 3.2</h1>
|       <p class="lead">Proyecto del curso de Karlita.</p>
|       <a class="btn btn-lg btn-primary" href="/docs/5.0/components/navbar/" role="button">Únete &raquo;</a>
|   {% endblock %}
</main>
|   </div>
</div>
```

The screenshot shows the "PROBAR DJANGO" website. The main content area displays the "Probar Django 3.2" registration form, which includes a title, subtitle, a "Únete »" button, and a registration form with fields for "Nombre" and "Email*".

Para emular donde va el vídeo pondremos un cuadro negro de la siguiente forma:

```
</main>
|   </div>
|   <div class='col-sm-6' style ='background-color: black; height:300px'></div>
|   </div>

{% block container-fluid %}
```

Hacemos estos cambios en el código:

```
{% block container-content %}

<div class='row'>
    <div class='col-sm-6'>
        <h1>Probar Django 3.2</h1>
        <p class='lead'>Proyecto del curso de Karlita.</p>
        <a class='btn btn-lg btn-primary' href="/docs/5.0/components/navbar/" role="button">Únete &raquo;</a>
    </div>
    <div class='col-sm-6' style ='background-color: #black; height:300px;'></div>
</div>
{% endblock %}

{{titulo}}<br/>
{{request.user}}<br/>
<hr/>
<br/>

{% block content %}
{{ block.super}}
<div class='row'>
    <div class='col-sm-6'>
<form method = 'POST' action ="">{{ csrf_token }}
{{ el_form|crispy }}
<input type='submit' value= 'Registrate' />
</form>
</div>
</div>
{% endblock %}
```

Y la página queda así.

PROBAR DJANGO Home Link Disabled

Search

Probar Django 3.2

Proyecto del curso de Karlita.

[Únete »](#)



Nombre

Email*

Quitamos el fluid de la clase container otra vez, la página quedará así.

A screenshot of a Django application interface. At the top, there is a navigation bar with the text "PROBAR DJANGO" and links for "Home", "Link", and "Disabled". On the far right of the navigation bar is a search input field with the placeholder "Search". The main content area has a light gray background. In the center, there is a large heading "Probar Django 3.2" in bold black font, followed by the subtext "Proyecto del curso de Karlita.". Below this text is a blue rectangular button with white text that says "Únete »". At the bottom of the page, there is a registration form with fields for "Nombre" and "Email*", each accompanied by a text input box. A "Regístrate" button is located at the bottom left of the form area.

Quitamos el fluid del navbar y quedará con la anchura del otro.

The screenshot shows a registration form with a single column layout. At the top, there's a navigation bar with links: PROBAR DJANGO, Home, Link, Disabled, and a search bar. Below the navigation is the title "Probar Django 3.2" and a subtitle "Proyecto del curso de Karlita.". A blue button labeled "Únete »" is present. The main content area contains two input fields: "Nombre" and "Email*", followed by a "Regístrate" button.

Ajustamos el tamaño que ocupa el formulario, hacemos cálculos y vemos que caben 3 columnas más, las añadimos:

```
{% block content %}
{{ block.super}}
<div class='row'>
| <div class='col-sm-3 pull-right'>
| <form method='POST' action="">{{ csrf_token }}
| {{ el_form|crispy }}
| <input type='submit' value='Regístrate' />
| </form>
| </div>
| <div class='col-sm-3'><p>Años de experiencia</p></div>
| <div class='col-sm-3'><p>Aprende con nosotros</p></div>
| <div class='col-sm-3'><p>José María hizo esto</p></div>
| </div>
{% endblock %}
```

The screenshot shows the same registration form but with a three-column layout. The "Regístrate" button has been moved to the right column. The columns are labeled "Nombre", "Años de experiencia", "Aprende con nosotros", and "José María hizo esto".

Le añadimos un título al formulario.

```
{% block content %}
{{ block.super}}
<div class='row'>
|   <div class='col-sm-3 pull-right'>
|       <p>{{ titulo }}</p>
<form method = 'POST' action ="">{{ csrf_token %}}
```

The screenshot shows a registration form titled "Probar Django 3.2". The title is displayed above the main content area. Below the title, there is a subtitle "Proyecto del curso de Karlita." and a blue button labeled "Únete »". The form itself has two input fields: "Nombre" and "Email*", both enclosed in a single horizontal input box. Below the inputs is a blue "Regístrate" button. At the top left of the form area, it says "Bienvenido jose". At the top right, it says "Años de experiencia".

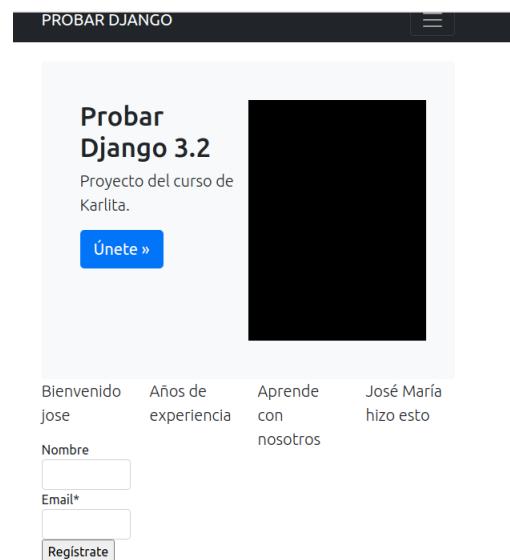
Hagamos las letras más grandes.

```
    {{titulo}}<br/>
    {{request.user}}<br/>
    <br/>

    {% block content %}
    {{ block.super}}
    <div class='row'>
        <div class='col-sm-3 pull-right'>
            <p class='lead'>{{ titulo }}</p>
        <form method = 'POST' action ="">{{ csrf_token %}}
        {{ el_form|crispy }}
        <input type='submit' value= 'Regístrate' />
    </form>
    </div>
    <div class='col-sm-3'><p class='lead'>Años de experiencia</p></div>
    <div class='col-sm-3'><p class='lead'>Aprende con nosotros</p></div>
    <div class='col-sm-3'><p class='lead'>José María hizo esto</p></div>
    </div>
    [% endblock %]
```

The screenshot shows a registration form with large, bold text. At the top left, it says "PROBAR DJANGO" followed by navigation links: "Home", "Link", and "Disabled". On the right, there is a search bar with a green "Search" button. The main content area has a large title "Probar Django 3.2" and a subtitle "Proyecto del curso de Karlita.". Below the title is a blue "Únete »" button. The form itself has two input fields: "Nombre" and "Email*", both enclosed in a single horizontal input box. Below the inputs is a blue "Regístrate" button. At the top left of the form area, it says "Bienvenido jose". At the top right, it says "Años de experiencia". To the right of the form, there is a large black rectangular placeholder or image area. At the bottom, there are four additional sections: "Aprende con nosotros", "José María hizo esto", "Bienvenido jose", and "Años de experiencia".

Se adapta a la anchura.



27. Estilo: CSS Custom