

# Creación de una landing page en Python



**PYTHON**

**django**

## Misión

Ser capaces de desarrollar una aplicación web completa usando un framework MVC (Modelo, Vista, Controlador)

## Planteamiento

Crear una prueba de vida de haber seguido el curso gratuito de Udemy sobre Django pero escribiendo el código en Python3 y subiendo a GitHub el estado de la aplicación tras cada capítulo.

Además, se pide un log de cabezazos. Es decir, cada vez que te encuentres un problema o una divergencia entre lo que ocurre en el vídeo y en la realidad, lo describas y describas también cómo lo has solucionado.

## ¿Qué se valora?

- Que los commit estén coherentemente distribuidos en el tiempo
- Que la presentación del trabajo esté bien ligada en un pdf o en una web (un site de Google por ejemplo) con pantallazos y explicaciones.

Empecemos por lo fundamental, ¿qué es...

## **PIP?**

PIP (Pip Instalador de Paquetes o Pip Instalador de Python) es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.

## **Python?**

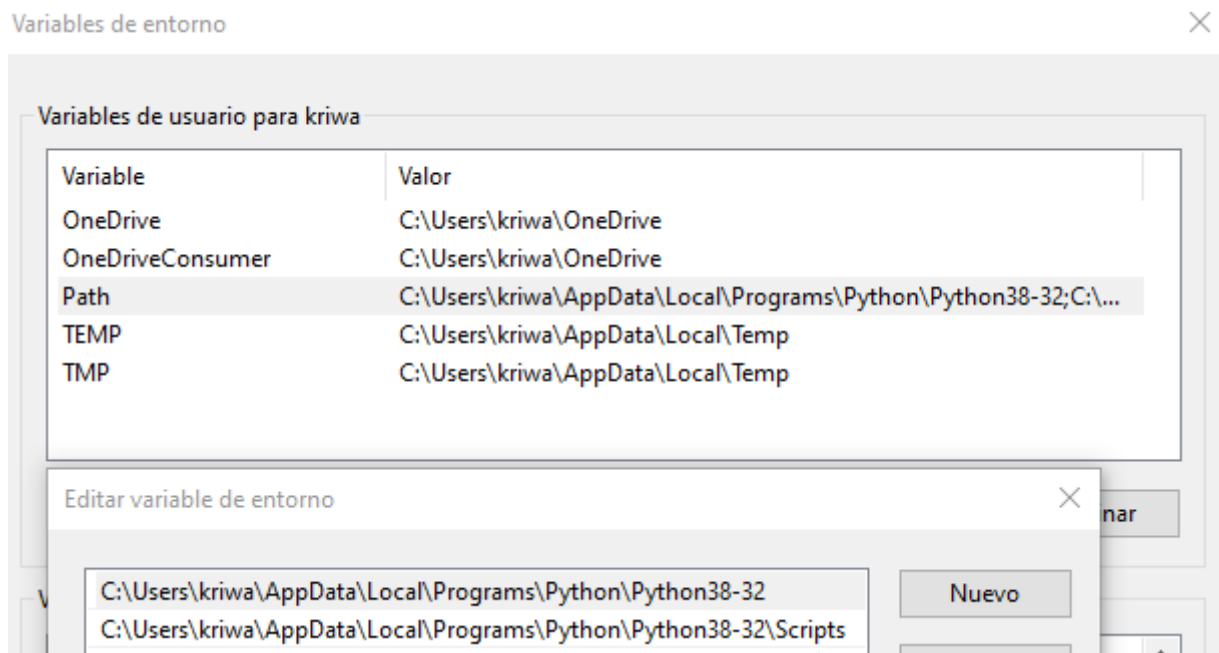
Es un lenguaje de programación interpretado, cuya filosofía hace hincapié en la legibilidad de su código y la orientación a objetos. Es un lenguaje interpretado, dinámico y multiplataforma, administrado por la Python Software Foundation.

## **Django?**

Es un framework (o entorno de trabajo) de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo–vista–controlador (MVC), un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. La meta fundamental de Django es facilitar la creación de sitios web complejos.

## 1. Instalación

Lo primero, descargarnos **Python** para **Windows** <https://www.python.org/downloads/>. Una vez hecho, tendremos que configurar las variables de entorno de nuestro sistema para que se pueda abrir *python* a través de la línea de comandos (cmd). Basta con averiguar la ruta donde hemos instalado nuestro *python* y crear una nueva variable de usuario (ver foto abajo)



Una vez configurado esto, podemos instalar *pip* desde cmd con el comando 'python -m pip install -U pip'

```
C:\Users\kriwa>python -m pip install -U pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/
/pip-21.0.1-py3-none-any.whl (1.5MB)
  | 1.5MB 2.2MB/s
Installing collected packages: pip
  Found existing installation: pip 19.2.3
  Uninstalling pip-19.2.3:
    Successfully uninstalled pip-19.2.3
  Successfully installed pip-21.0.1
```

Con *pip* ya instalado, nuestro siguiente paso consiste en instalar **Virtual Enviroment**. Para ello, escribimos 'pip install virtualenv'

```
C:\Users\kriwa>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.4.4-py2.py3-none-any.whl (7.2 MB)
    | 7.2 MB 656 kB/s
Collecting six<2,>=1.9.0
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting appdirs<2,>=1.4.3
  Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.1-py2.py3-none-any.whl (335 kB)
    | 335 kB 819 kB/s
Collecting filelock<4,>=3.0.0
  Downloading filelock-3.0.12-py3-none-any.whl (7.6 kB)
Installing collected packages: six, filelock, distlib, appdirs, virtualenv
Successfully installed appdirs-1.4.4 distlib-0.3.1 filelock-3.0.12 six-1.15.0 virtualenv-20.4.4
```

La finalidad de crear un **Virtual Enviroment** es poder instalar **Django** dentro de él. A modo de prueba, creamos uno con 'virtualenv test\_env' en nuestra carpeta personal. Lo activamos entrando en '.\Scripts\activate'.

Una vez activado, basta con escribir 'pip install django' y comenzará el proceso de instalación dentro de nuestro **Virtual Enviroment**.

```
(test_env) C:\Users\kriwa\test_env>pip install django
Collecting django
  Downloading Django-3.2-py3-none-any.whl (7.9 MB)
    | 7.9 MB 1.6 MB/s
Collecting pytz
  Downloading pytz-2021.1-py2.py3-none-any.whl (510 kB)
    | 510 kB 1.3 MB/s
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.1-py3-none-any.whl (42 kB)
    | 42 kB 1.1 MB/s
Collecting asgiref<4,>=3.3.2
  Downloading asgiref-3.3.4-py3-none-any.whl (22 kB)
Installing collected packages: sqlparse, pytz, asgiref, django
Successfully installed asgiref-3.3.4 django-3.2 pytz-2021.1 sqlparse-0.4.1
```

Vamos a crear un proyecto nuevo con **Django**, vamos a escribir lo siguiente 'python .\Scripts\django-admin.py startproject test\_project'. Con esto indicamos que queremos iniciar un nuevo proyecto con su nombre. Para asegurarnos de su correcta instalación, podemos hacer un listado del contenido de nuestro **Virtual Enviroment**.

```
(test_env) C:\Users\kriwa\test_env>python .\Scripts\django-admin.py startproject test_project
.\Scripts\django-admin.py:17: RemovedInDjango40Warning: django-admin.py is deprecated in favor of django-admin.
  warnings.warn(

(test_env) C:\Users\kriwa\test_env>dir
El volumen de la unidad C es Boot
El número de serie del volumen es: 6867-34E3

Directorio de C:\Users\kriwa\test_env

03/05/2021  12:58    <DIR>        .
03/05/2021  12:58    <DIR>        ..
03/05/2021  12:50         42     .gitignore
03/05/2021  12:50    <DIR>        lib
03/05/2021  12:50       416     pyenv.cfg
03/05/2021  12:52    <DIR>        Scripts
03/05/2021  12:58    <DIR>        test_project
                2 archivos          458 bytes
                5 dirs  19.267.997.696 bytes libres
```

## 2. Empezar proyecto nuevo

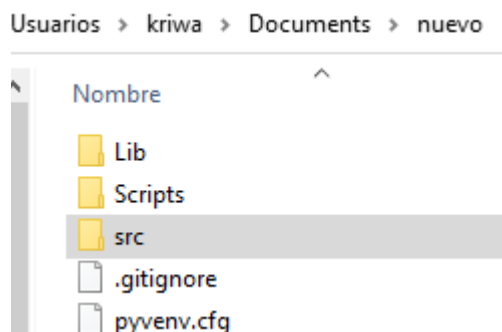
Para empezar un proyecto nuevo, abrimos nuestro cmd y creamos un nuevo directorio (en mi caso dentro de 'Documents' voy a crear una carpeta llamada 'nuevo'). Entramos en ella e instalamos el **Virtual Enviroment**. Después, lo activamos como hicimos en el ejemplo anterior (ver imagen inferior)

```
C:\Users\kriwa>cd Documents
C:\Users\kriwa\Documents>mkdir nuevo && cd nuevo
C:\Users\kriwa\Documents\nuevo>virtualenv .
created virtual environment CPython3.8.1.final.0-32 in 3981ms
creator CPython3Windows(dest=C:\Users\kriwa\Documents\nuevo, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\kriwa\AppData\Local\pypa\virtualenv)
added seed packages: pip==21.0.1, setuptools==56.0.0, wheel==0.36.2
activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
C:\Users\kriwa\Documents\nuevo>.\Scripts\activate
(nuevo) C:\Users\kriwa\Documents\nuevo>
```

Lo siguiente será instalar **Django** dentro de nuestro **Virtual Enviroment**. De nuevo, con el comando 'pip install django'

```
(nuevo) C:\Users\kriwa\Documents\nuevo>pip install django
Collecting django
  Using cached Django-3.2-py3-none-any.whl (7.9 MB)
Collecting pytz
  Using cached pytz-2021.1-py2.py3-none-any.whl (510 kB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.3.2
  Using cached asgiref-3.3.4-py3-none-any.whl (22 kB)
Installing collected packages: sqlparse, pytz, asgiref, django
Successfully installed asgiref-3.3.4 django-3.2 pytz-2021.1 sqlparse-0.4.1
```

Ya solo nos falta empezar un nuevo proyecto. Usamos 'python .\Scripts\django-admin.py startproject nuevo'. Como veréis, vamos a tener muchos directorios con el nombre 'nuevo'. Para no crear confusión, dentro de nuestro proyecto que creamos en 'Documents', vamos a renombrar la carpeta raíz del proyecto 'nuevo' a 'src'.



Volvemos al cmd y vamos a situarnos en nuestro 'src' para ejecutar el servidor de nuestro desarrollo de entorno. Escribiremos 'pyhton manage.py runserver'

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 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.
May 03, 2021 - 18:33:13
Django version 3.2, using settings 'nuevo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Con ello nuestro servidor empezará a funcionar. Simplemente copiamos la ruta del servidor y la añadimos a nuestro navegador web.

django

View [release notes](#) for Django 3.2



The install worked successfully! Congratulations!

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

La instalación se instaló sin problemas. ¡Felicidades!

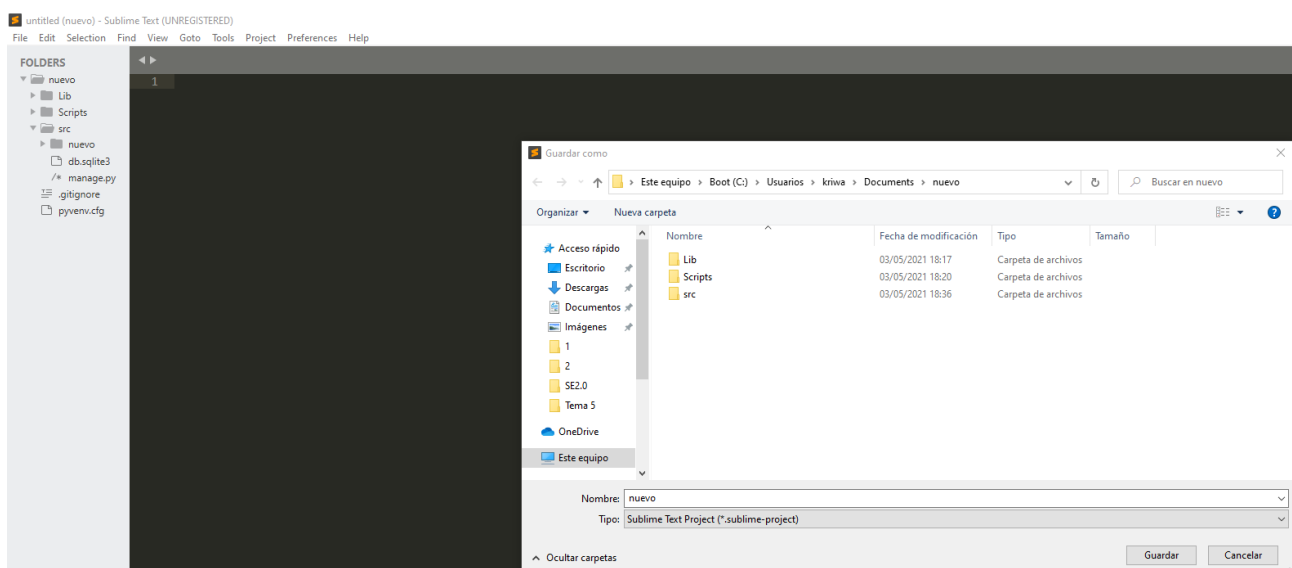
### 3. Primera migración

Ya tenemos nuestro proyecto funcionando. Ahora podemos empezar con las migraciones. Las migraciones nos sirven para comunicarnos con nuestra base de datos, es el vínculo entre la base de datos y nuestro proyecto.

En nuestro cmd, cerramos el servidor (Ctrl + D) y escribimos 'python manage.py migrate'.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying 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
```

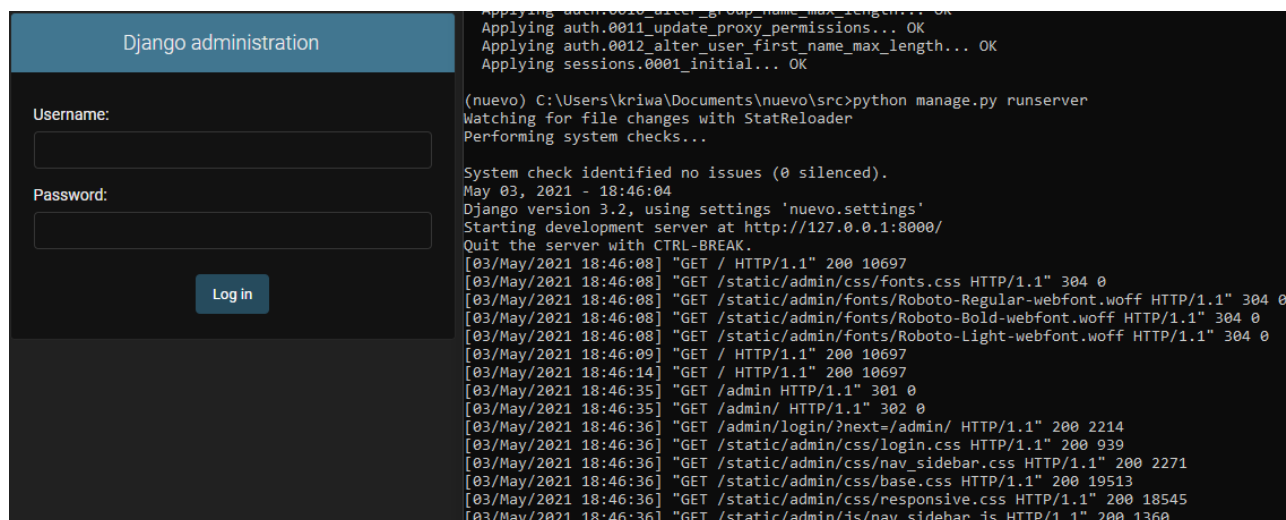
A continuación, abrimos nuestro editor de texto (en mi caso Sublime Text) y vamos a trabajar con nuestro proyecto. Pinchamos en 'Project' y añadimos la carpeta del proyecto 'nuevo'. Una vez abierto, lo guardamos dentro de nuestra carpeta 'raíz'.





## 4. Superusuarios y Administración de Django

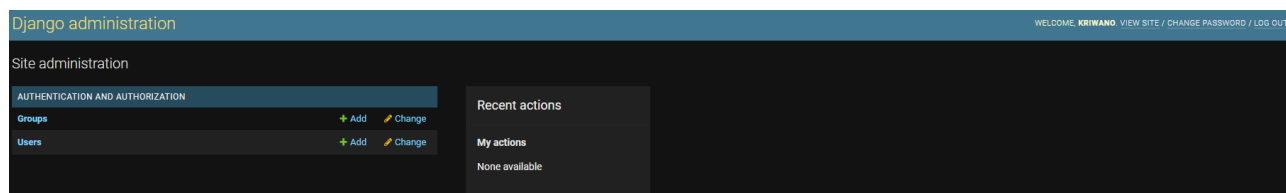
Los proyectos de **Django** vienen automáticamente con una interfaz ya escrita. Volvemos a activar nuestro servidor y recargamos en nuestro navegador web.



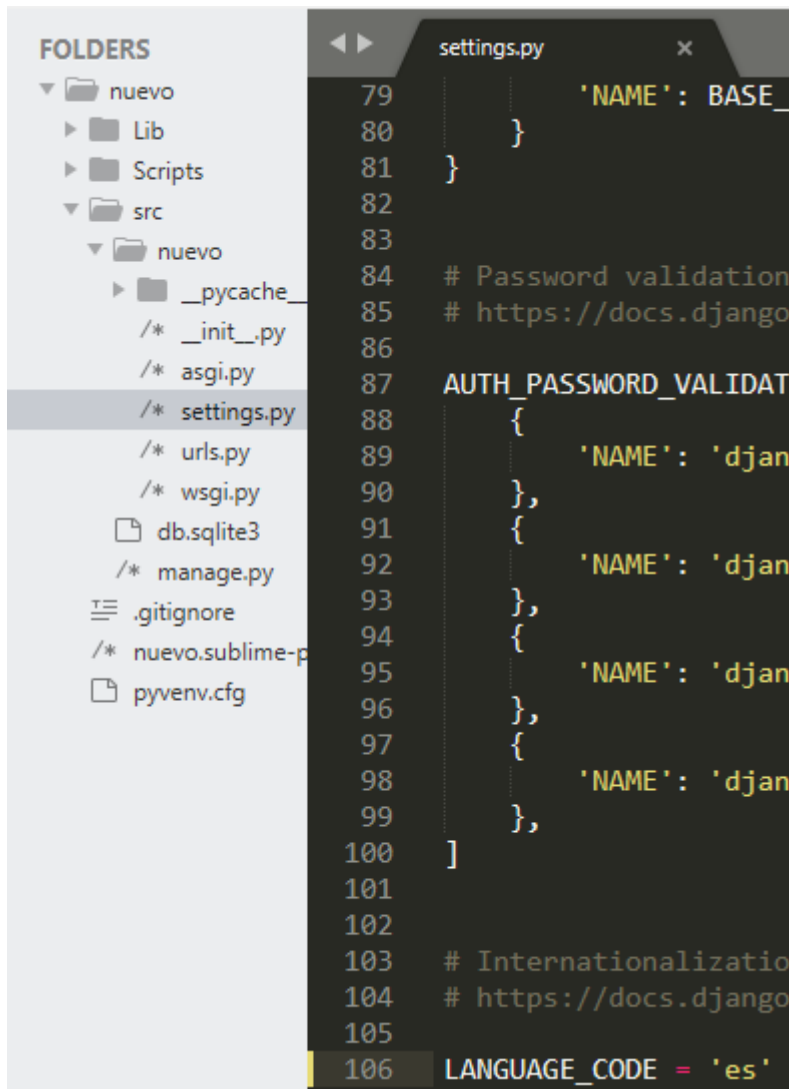
Para poder iniciar sesión dentro de **Django**, cerramos servidor y vamos a crear un *superusuario*, capaz de entrar. Para ello, ponemos en nuestro cmd 'python manage.py createsuperuser' y añadimos nuestro usuario, email y contraseña.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py createsuperuser
Username (leave blank to use 'kriwa'): kriwano
Email address: ivanperezprofesional@hotmail.com
Password:
Password (again):
Superuser created successfully.
```

Volvemos a iniciar el servidor y ya podemos acceder a nuestra pagina. Esta es la pagina principal de la interfaz administrativa..



Si queremos que la pagina esté en ingles, podemos entrar en el archivo 'settings.py' y donde pone 'LANGUAGE\_CODE' cambiar el idioma al español.



```
79     'NAME': BASE_
80     }
81     }
82
83
84     # Password validation
85     # https://docs.django
86
87     AUTH_PASSWORD_VALIDAT
88     {
89         'NAME': 'djan
90     },
91     {
92         'NAME': 'djan
93     },
94     {
95         'NAME': 'djan
96     },
97     {
98         'NAME': 'djan
99     },
100 ]
101
102
103 # Internationalizatio
104 # https://docs.django
105
106 LANGUAGE_CODE = 'es'
```

## AUTENTICACIÓN Y AUTORIZACIÓN

Grupos	+ Añadir	✎ Modificar
Usuarios	+ Añadir	✎ Modificar

## Acciones recientes

## Mis acciones

Ninguno disponible

Si, por ejemplo, queremos crear otro usuario, podemos entrar en la sección de ‘Usuarios’ y ‘Añadir usuario’, con un menú como el de la imagen de abajo.

### Añadir usuario

Primero, ingrese un nombre de usuario y contraseña. Luego, podrá editar más opciones del usuario.

**Nombre de usuario:**   
Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/\_

**Contraseña:**   
Su contraseña no puede asemejarse tanto a su otra información personal.  
Su contraseña debe contener al menos 8 caracteres.  
Su contraseña no puede ser una clave utilizada comúnmente.  
Su contraseña no puede ser completamente numérica.

**Contraseña (confirmación):**   
Para verificar, introduzca la misma contraseña anterior.

Si volvemos a ‘Usuarios’ podremos ver que ya tenemos el usuario creado. No es ‘personal’, no tendrá todos los permisos como nuestro *superusuario*.

Seleccione usuario a modificar

ÁÑADIR USUARIO +

Buscar

Acción: Ir seleccionados 0 de 2

	NOMBRE DE USUARIO	DIRECCIÓN DE CORREO ELECTRÓNICO	NOMBRE	APELLIDOS	ES STAFF
<input checked="" type="checkbox"/>	kriwano	ivanperezprofesional@hotmail.com			✓
<input checked="" type="checkbox"/>	usuario				✗

2 usuarios

**FILTRO**

Por es staff

Todo

Sí

No

Por estado de superusuario

Todo

Sí

No

Por activo

Todo

Sí

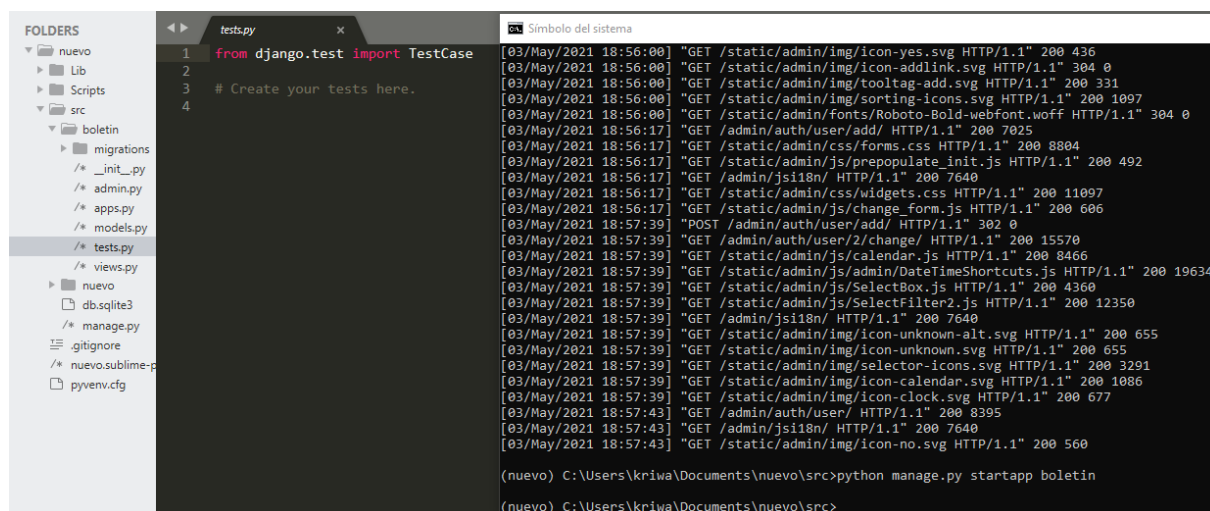
No

## 5. Primera aplicación

Vamos a crear nuestra primera aplicación dentro de nuestro proyecto. La diferencia entre un sitio web y una aplicación web es muy subjetiva, un sitio web puede ser escrito solo en HTML, pero una aplicación web añade una interacción con el usuario.

Nuestra aplicación web sera una pagina de aterrizaje o *landing page*, pero además dentro vamos a crear aplicaciones. Una aplicación de nuestro proyecto va a ser un boletín y solo tendrá que ver con cosas de ese boletín. Hay que crear una aplicación para cada una de las funciones de nuestro proyecto.

Para crear nuestro boletín, cerramos el servidor y añadimos 'python manage.py startapp boletin'. Si volvemos a nuestro editor de texto, observaremos que se ha creado un nuevo directorio 'migrations' con varios archivos de *python*.



The screenshot shows a code editor with a file explorer on the left and a terminal on the right. The file explorer shows a project named 'nuevo' with a subdirectory 'src' containing a 'boletin' app. The 'boletin' app has a 'migrations' directory and several Python files. The terminal shows the output of the command 'python manage.py startapp boletin', which lists the files created for the new app and the command used to create it.

```
1 from django.test import TestCase
2
3 # Create your tests here.
4
```

```
[03/May/2021 18:56:00] "GET /static/admin/img/icon-yes.svg HTTP/1.1" 200 436
[03/May/2021 18:56:00] "GET /static/admin/img/icon-addlink.svg HTTP/1.1" 304 0
[03/May/2021 18:56:00] "GET /static/admin/img/tooltag-add.svg HTTP/1.1" 200 331
[03/May/2021 18:56:00] "GET /static/admin/img/sorting-icons.svg HTTP/1.1" 200 1097
[03/May/2021 18:56:00] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 304 0
[03/May/2021 18:56:17] "GET /admin/auth/user/add/ HTTP/1.1" 200 7025
[03/May/2021 18:56:17] "GET /static/admin/css/forms.css HTTP/1.1" 200 8804
[03/May/2021 18:56:17] "GET /static/admin/js/prepopulate_init.js HTTP/1.1" 200 492
[03/May/2021 18:56:17] "GET /admin/jsi18n/ HTTP/1.1" 200 7640
[03/May/2021 18:56:17] "GET /static/admin/css/widgets.css HTTP/1.1" 200 11097
[03/May/2021 18:56:17] "GET /static/admin/js/change_form.js HTTP/1.1" 200 606
[03/May/2021 18:57:39] "POST /admin/auth/user/add/ HTTP/1.1" 302 0
[03/May/2021 18:57:39] "GET /admin/auth/user/2/change/ HTTP/1.1" 200 15570
[03/May/2021 18:57:39] "GET /static/admin/js/calendar.js HTTP/1.1" 200 8466
[03/May/2021 18:57:39] "GET /static/admin/js/admin/DateTimeShortcuts.js HTTP/1.1" 200 19634
[03/May/2021 18:57:39] "GET /static/admin/js/SelectBox.js HTTP/1.1" 200 4360
[03/May/2021 18:57:39] "GET /static/admin/js/SelectFilter2.js HTTP/1.1" 200 12350
[03/May/2021 18:57:39] "GET /admin/jsi18n/ HTTP/1.1" 200 7640
[03/May/2021 18:57:39] "GET /static/admin/img/icon-unknown-alt.svg HTTP/1.1" 200 655
[03/May/2021 18:57:39] "GET /static/admin/img/icon-unknown.svg HTTP/1.1" 200 655
[03/May/2021 18:57:39] "GET /static/admin/img/selector-icons.svg HTTP/1.1" 200 3291
[03/May/2021 18:57:39] "GET /static/admin/img/icon-calendar.svg HTTP/1.1" 200 1086
[03/May/2021 18:57:39] "GET /static/admin/img/icon-clock.svg HTTP/1.1" 200 677
[03/May/2021 18:57:43] "GET /admin/auth/user/ HTTP/1.1" 200 8395
[03/May/2021 18:57:43] "GET /admin/jsi18n/ HTTP/1.1" 200 7640
[03/May/2021 18:57:43] "GET /static/admin/img/icon-no.svg HTTP/1.1" 200 560

(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py startapp boletin

(nuevo) C:\Users\kriwa\Documents\nuevo\src>
```

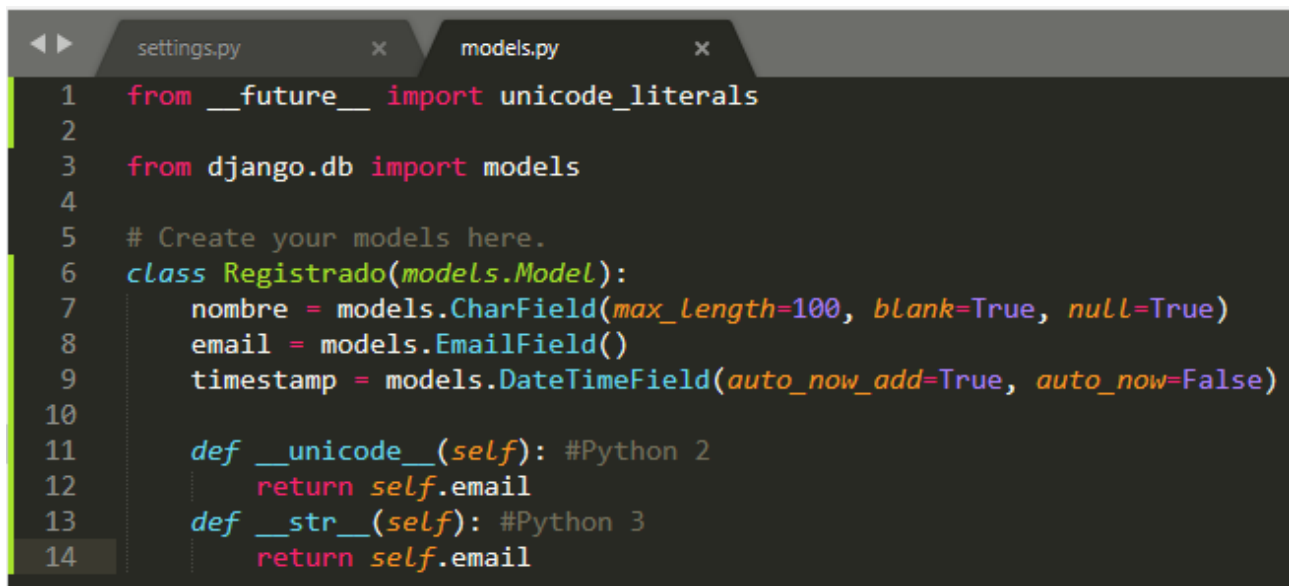
Vemos que ya tenemos nuestra aplicacion 'boletín' pero nos falta registrarlo en nuestro archivo 'settings.py'. Buscamos el campo de 'INSTALLED\_APPS' y lo añadimos.

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

## 6. Primer modelo

Los modelos son como creamos tablas en la base de datos para guardar objetos y todos sus atributos. Tenemos que escribir un modelo para crear esos campos, que serán los mismos que en nuestra base de datos.

Vamos a crear un registro de la gente que va a recibir nuestro boletín. Para ello escribimos dentro de 'models.py' lo siguiente:



```
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): #Python 2
12         return self.email
13     def __str__(self): #Python 3
14         return self.email
```

Vamos a crear un campo para el nombre del usuario, su email y un timestamp.

Dentro del usuario es obligatorio especificar la longitud máxima del nombre (max\_length=100). Si no añadiésemos ni el 'blank' ni el 'null', por defecto sería 'False'.

Al final, definimos el unicode, tanto para Python 2 como Python 3

Por ultimo, vamos a realizar la migración. Cerramos el servidor y en el cmd escribimos 'python manage.py makemigrations'. Va a buscar todas las modificaciones que hayamos hecho en nuestro modelo y va a empaquetarlas.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py makemigrations
Migrations for 'boletin':
  boletin\migrations\0001_initial.py
    - Create model Registrado

(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, boletin, contenttypes, sessions
Running migrations:
  Applying boletin.0001_initial... OK
```

Y con 'python manage.py migrate' se comunican las modificaciones a nuestra base de datos.

## 7. Crear objetos y registrar modelo en Administración

Una vez definido nuestro modelo podemos crear objetos y guardarlos en nuestra base de datos. Una manera de crearlos es utilizando el shell de Python.

Cerramos el servidor y escribimos 'python manage.py shell'. Tendremos que importar nuestro modelo, en la consola ponemos 'from boletin.models import Registrado'. También podemos crear una variable nueva, por ejemplo 'gente = Registrado.objects.all()'. Veremos que nuestra QuerySet es una lista vacía porque no tenemos nada guardado.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py shell
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
(InteractiveConsole)
>>> from boletin.models import Registrado
>>> gente = Registrado.objects.all()
>>> gente
<QuerySet []>
>>> persona1 = Registrado.objects.create(nombre='karlita', email='k@email.com')
>>> persona1
<Registrado: k@email.com>
>>> gente
<QuerySet [<Registrado: k@email.com>]>
```

Para guardar objetos podemos hacer 'persona1 = Registrado.objects.create(nombre='karlita', email='k@email.com')'

Si volvemos a escribir 'persona1' nos mostrará el email que hemos añadido, esto se debe a que en nuestro modelo hemos especificado en el 'unicode' que nos devuelva el email (podemos cambiarlo por el nombre si quisiésemos)

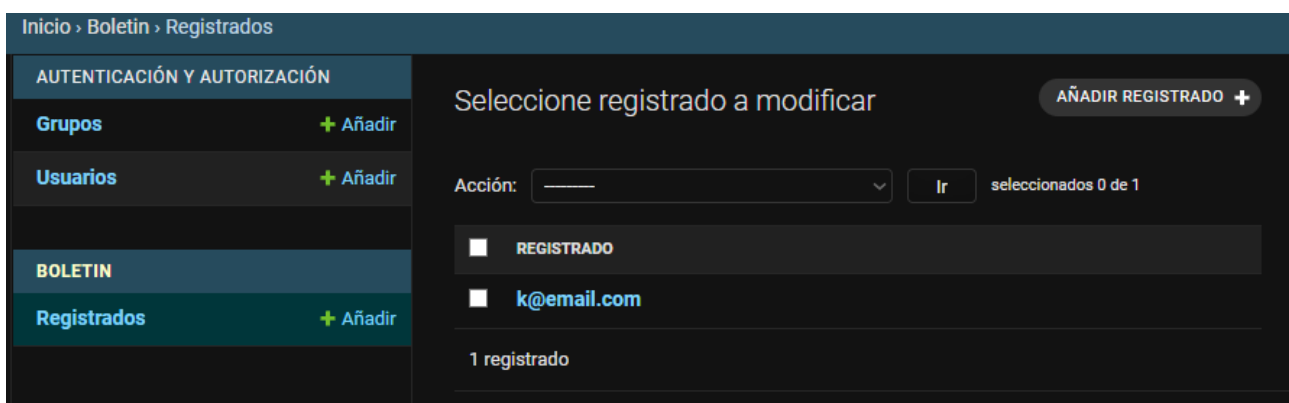
Si volvemos a escribir 'gente' veremos que ya hay un 'Registrado' en nuestra base de datos pero fuera del shell no tenemos ninguna manera de verlo. Para verlo vamos a registrarlo en la administración de Django.

Dentro de nuestro editor de texto nos dirigimos al archivo 'admin.py' y vamos a realizar una importación relativa con 'from .models import Registrado'.

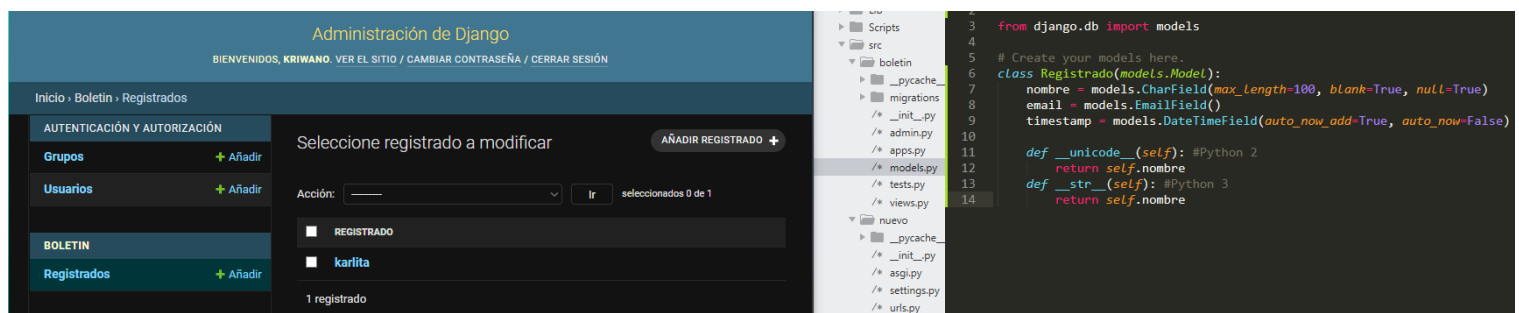
Para registrarlo en la administracion, escribimos 'admin.site.register(Registrado)'

```
settings.py x models.py x admin.py x
1 from django.contrib import admin
2
3 # Register your models here.
4 from .models import Registrado
5
6
7 admin.site.register(Registrado)
```

Si ahora recargamos nuestra pagina web veremos que hay un registrado dentro de nuestro boletín, el que hemos creado en la shell de Python. Vemos que solo nos da el email (por el unicode)



Si quisieramos que nos mostrase el nombre del usuario registrado, solo tendríamos que cambiar el unicode de 'email' a 'nombre' (ver ejemplo abajo)



Nosotros vamos a dejarlo con el email.



## 8. Personalizar modelo en el Admin

Podemos personalizar el display de la Administración de Django, para ello volvemos al 'admin.py' y debajo de la importación vamos a escribir un class que sera 'AdminRegistrado', con varias listas:

- El display, va a mostrar nuestro unicode (el email), el nombre y el timestamp.
- El filter, para poder filtrar por cualquier fecha.
- El editable, un campo para editar nuestro nombre.
- Un search\_fields, para buscar por email o nombre

Ademas, dentro de 'AdminRegistrado' creamos otra clase 'Meta' con 'model = Registrado'

The image shows a Django Admin interface at the top and a Sublime Text editor at the bottom. The Admin interface displays a table with two records, each with an email, a name field, and a timestamp. A filter sidebar on the right allows filtering by timestamp. The Sublime Text editor shows the Python code for the 'AdminRegistrado' class, which inherits from 'admin.ModelAdmin' and defines 'list\_display', 'list\_filter', 'list\_editable', and 'search\_fields'. It also includes a 'Meta' class with 'model = Registrado'.

Seleccione registrado a modificar

Acción:  Ir seleccionados 0 de 2

EMAIL	NOMBRE	TIMESTAMP
yo@yo.com	<input type="text"/>	3 de Mayo de 2021 a las 17:30
k@email.com	karlita	3 de Mayo de 2021 a las 17:21

2 registrados

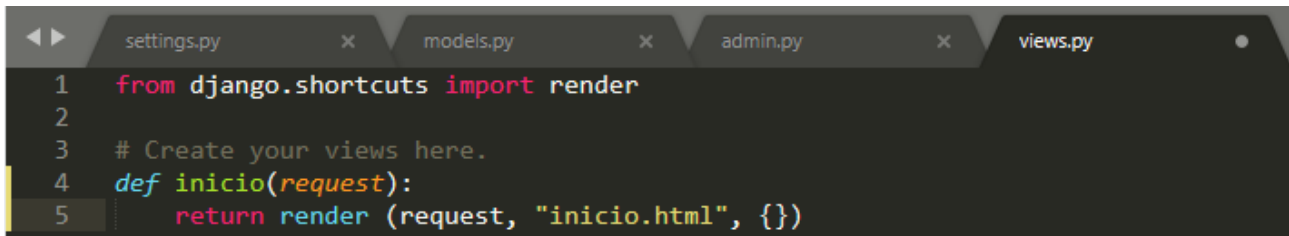
FILTRO

- Por timestamp
- Cualquier fecha
- Hoy
- Últimos 7 días
- Este mes
- Este año

```
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 = ["email", "nombre", "timestamp"]
8     # list_display_links = ["nombre"]
9     list_filter = ["timestamp"]
10    list_editable = ["nombre"]
11    search_fields = ["email", "nombre"]
12
13    class Meta:
14        model = Registrado
15
16
17 admin.site.register(Registrado, AdminRegistrado)
```

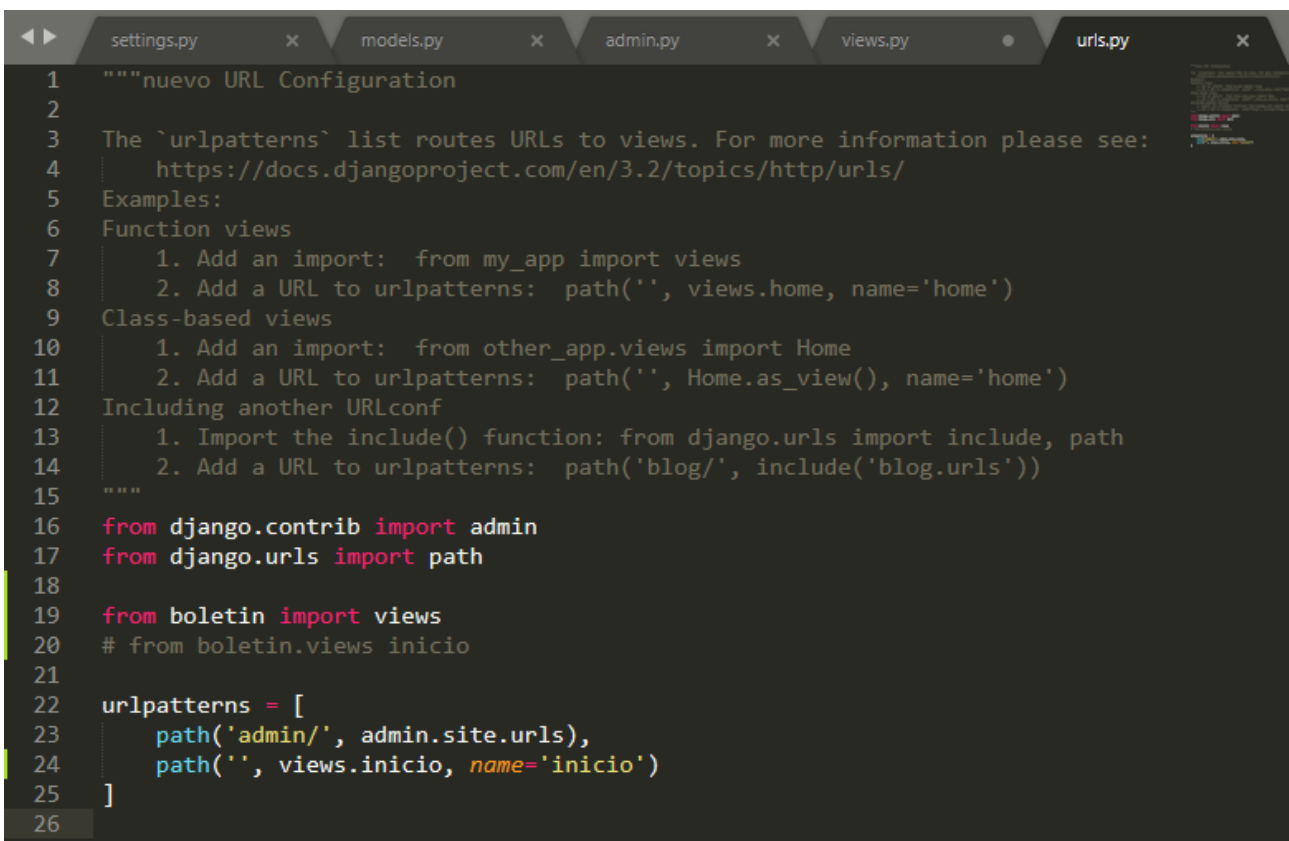
## 9. Primera vista

Ahora vamos a escribir nuestra primera vista, por lo tanto vamos a abrir el 'views.py' y escribimos lo siguiente:



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

Como veréis, hemos añadido una url como plantilla para nuestra vista, pero no está todavía configurada. Así que vamos a abrir otro fichero dentro de 'nuevo' llamada 'urls.py'



```
1 """nuevo URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/3.2/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import:  from my_app import views
8     2. Add a URL to urlpatterns:  path('', views.home, name='home')
9 Class-based views
10    1. Add an import:  from other_app.views import Home
11    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
15 """
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
```

Importamos nuestra vista 'from boletin import views' y abajo añadimos la ruta de nuestra vista, de nombre 'inicio'

Si lo dejamos así e intentamos entrar en el navegador web, nos aparecerá un bonito dedo en forma de texto.



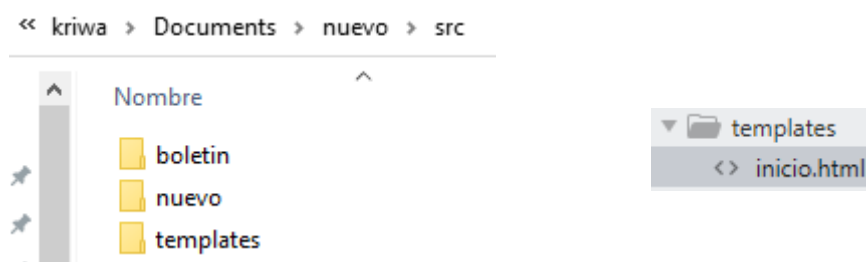
El problema está en que no tenemos una plantilla 'inicio.html' creada.

## 10. Configuración de plantillas

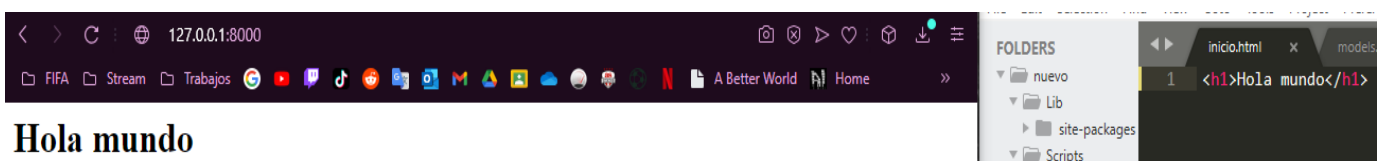
Tenemos que las plantillas están guardadas dentro de 'settings.py'

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(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',  
            ],  
        },  
    ],  
]
```

Vamos a crear una ruta dentro de la plantilla dirigiendo el html a la carpeta "templates". Como esa carpeta no existe, la crearemos dentro de nuestro proyecto y dentro de ella creamos 'inicio.html'



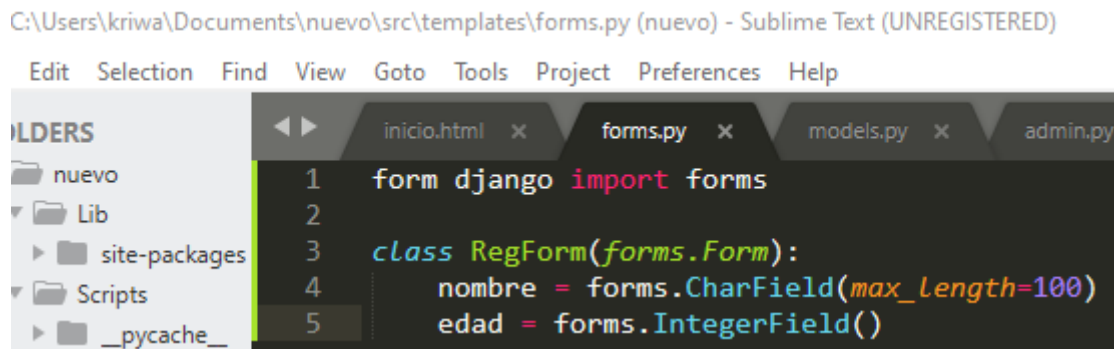
Para comprobar si funciona, vamos a hacer una prueba escribiendo el universal 'Hola Mundo'.



Perfecto.

## 11. Escribir formulario

Dentro de nuestra aplicación creamos un archivo nuevo que se llama 'forms.py'



The screenshot shows a Sublime Text editor window titled 'C:\Users\kriwa\Documents\nuevo\src\templates\forms.py (nuevo) - Sublime Text (UNREGISTERED)'. The editor has a menu bar with 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Preferences', and 'Help'. On the left, a 'FOLDERS' sidebar shows the project structure: 'nuevo' (expanded), 'Lib', 'site-packages', 'Scripts', and '\_\_pycache\_\_'. The main editor area shows the following Python code in 'forms.py':

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

Tenemos que hacer una importación y abajo escribimos nuestro formulario. Al ser un formulario le podemos añadir los campos que queramos sin que existan en nuestro modelo. Vamos a poner nuestro nombre y la edad.

## 12. Formulario en una vista

Si queremos añadir el formulario a nuestra vista abrimos el 'views.py' e importamos RegForm

```
from django.shortcuts import render

from .forms import RegForm

# Create your views here.
def inicio(request):
    form = RegForm()
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Creamos una variable nueva 'form = RegForm()' y un diccionario 'context', que lo vamos a meter en nuestro html.

Volvemos a 'inicio.html' para cargar la variable en nuestra plantilla.

```
<h1>Hola mundo</h1>

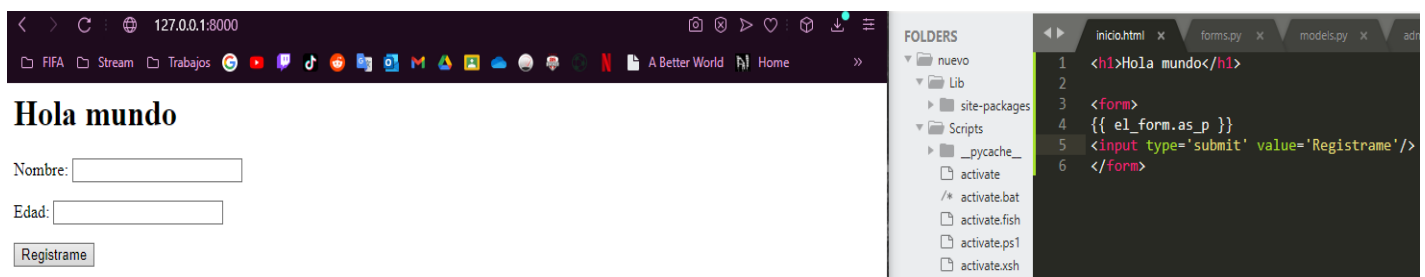
{{ el_form }}
```

Si recargamos la pagina veremos que tenemos dos campos para rellenar (nombre y edad)

**Hola mundo**

Nombre:  Edad:

Si queremos añadir un botón de registro, podemos crear un input dentro de 'inicio.html' (ver imagen inferior)

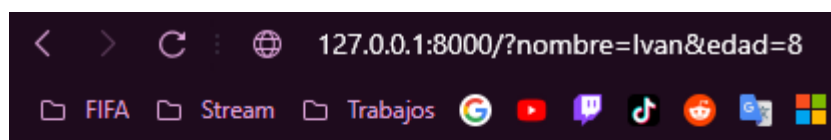


## 13. Método HTTP POST en Formulario

Antes de trabajar en la base de datos tenemos que saber unos conceptos muy importantes. El CRUD (Create Read Update Delete) consiste de 4 funciones principales de cada aplicación web y cada uno tiene un método HTTP para llevar a cabo la función.

Lo que pretendemos hacer con nuestro formulario anteriormente creado es guardar información dentro de la base de datos, para ello tenemos que especificar el método. Para la C de Create sería el método POST.

Si nosotros queremos añadir el nombre y la edad pinchando en el botón 'Regístrate', observaremos que no ocurre nada. Aparece en el navegador como método GET pero no se almacena en ningún objeto.



# Hola mundo

Nombre:

Edad:

Para arreglar esto, volvemos a nuestro 'inicio.html' y dentro de la etiqueta 'form' añadimos el método POST. Si queremos utilizar POST tenemos que añadir el csrf\_token a nuestro formulario (ayuda contra la falsificación de peticiones).

```
<h1>Hola mundo</h1>

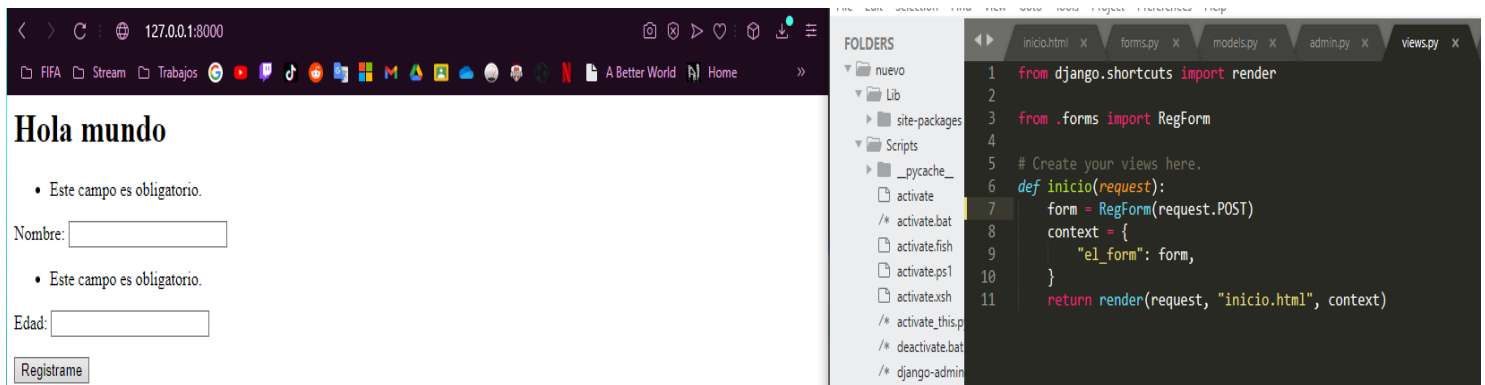
<form method="POST" action="">{% csrf_token %}
{{ el_form.as_p }}
<input type='submit' value='Regístrate'/>
</form>
```

Si volvemos a intentar añadir nuestro nombre y edad, observaremos en el cmd que aparece el método POST, pero no hemos creado el objeto aún.

```
[09/May/2021 11:23:59] "GET /?nombre=Ivan&edad=8 HTTP/1.1" 200 440
[09/May/2021 11:24:04] "POST /?nombre=Ivan&edad=8 HTTP/1.1" 200 440
```

## 14. Validaciones Formulario pt.1

Para analizar si los datos del formulario son datos limpios, primero tendremos que hacer algo con nuestro método POST en 'views.py' Vamos a añadir a nuestro formulario la petición del POST (request.POST)



Si recargamos la web veremos que nos indica los dos campos como obligatorios para rellenar. En el modelo el campo de 'nombre' no es obligatorio pero en el formulario si. Si queremos quitar ese mensaje, basta con incluir 'or None' dentro de 'form'.

```
form = RegForm(request.POST or None)
```



Podemos hacer que el formulario saque los datos limpios cuando nos registremos, para ello ponemos la condición en 'views.py' de que si el formulario es valido, nos muestre nuestros datos de manera clara.

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

Simbolo del sistema - python manage.py runserver

Performing system checks...

System check identified no issues (0 silenced).

May 09, 2021 - 11:44:11

Django version 3.2, using settings 'nuevo.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

[09/May/2021 11:44:15] "GET / HTTP/1.1" 200 440

{'nombre': 'Ivan', 'edad': 8}

[09/May/2021 11:44:37] "POST / HTTP/1.1" 200 463

Si queremos quitar el diccionario, volvemos a escribir:

```
1 from django.shortcuts import render
2
3 from .forms import RegForm
4
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        print (form_data.get("nombre"))
11        print (form_data.get("edad"))
12    context = {
13        "el_form": form,
14    }
15    return render(request, "inicio.html", context)
```

Simbolo del sistema - python manage.py runserver

[09/May/2021 11:48:23] "POST / HTTP/1.1" 200 463

Ivan

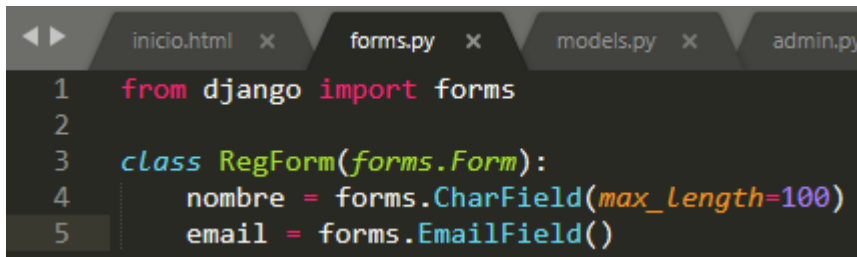
8

[09/May/2021 11:48:38] "POST / HTTP/1.1" 200 463

## 15. Guardar datos del formulario con el modelo.

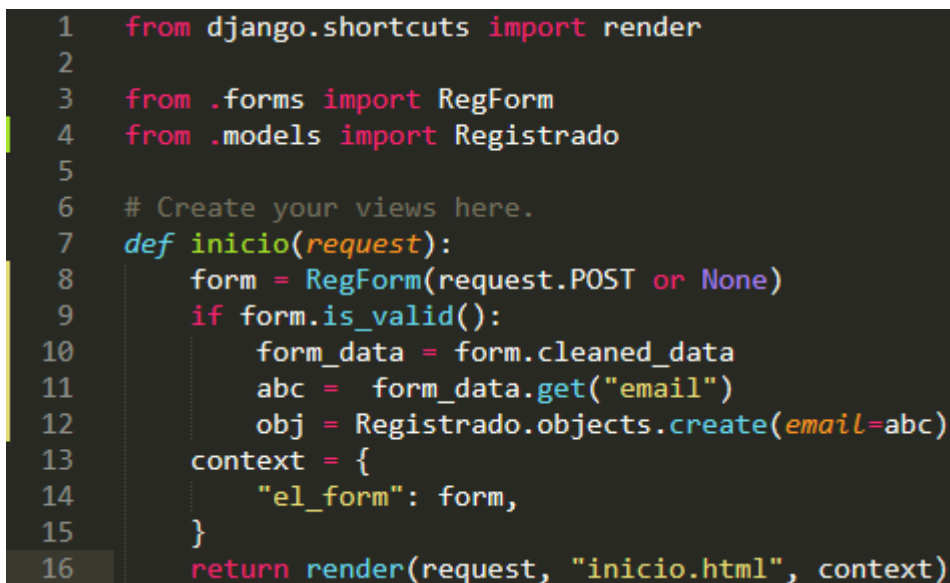
Ahora vamos a guardar objetos con los datos del formulario utilizando nuestro modelo.

Lo primero que vamos a hacer es sustituir el campo 'edad' por nuestro campo 'email'.



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

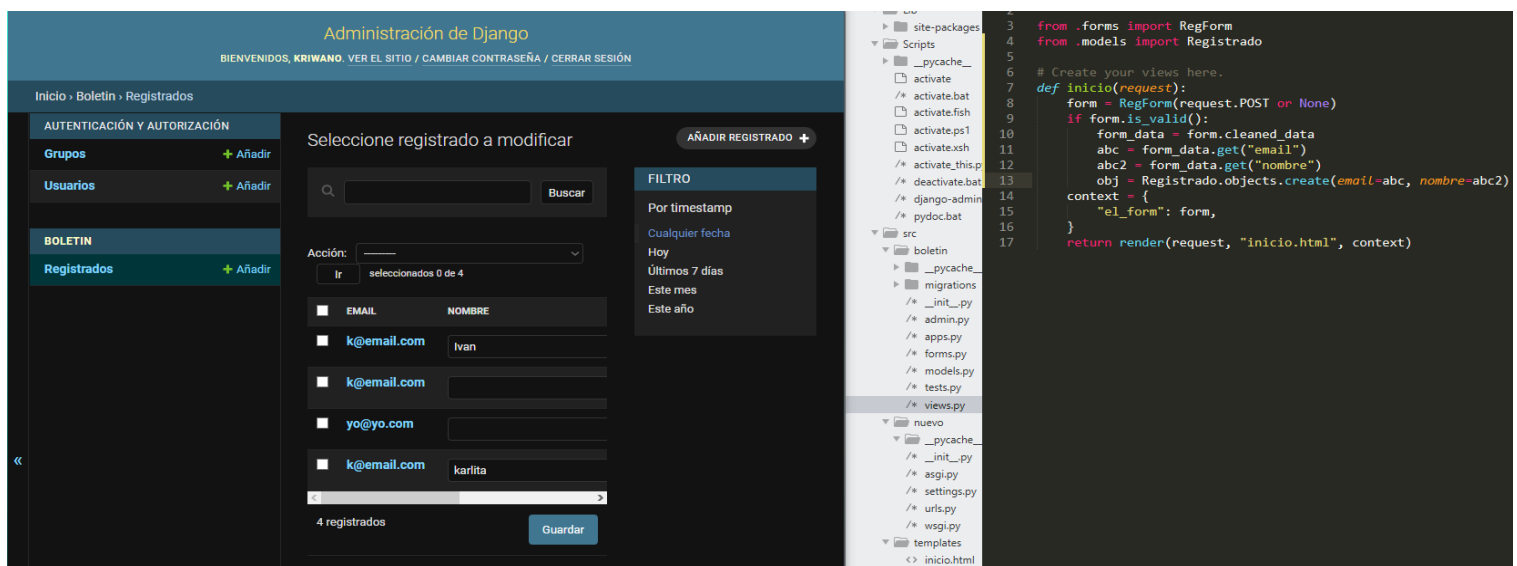
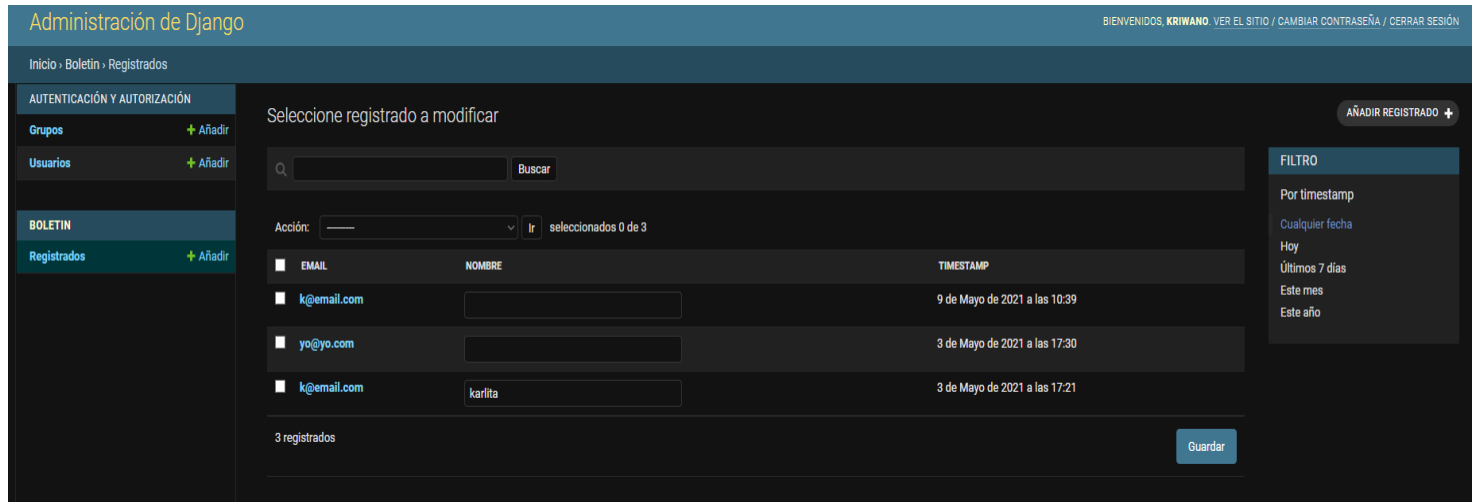
Vamos a retocar el 'views.py'. Creamos una variable nueva 'abc' con nuestro email y otra 'obj' para crear un objeto con el email del modelo previamente importado.



```
1 from django.shortcuts import render
2
3 from .forms import RegForm
4 from .models import Registrado
5
6 # Create your views here.
7 def inicio(request):
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         obj = Registrado.objects.create(email=abc)
13     context = {
14         "el_form": form,
15     }
16     return render(request, "inicio.html", context)
```

{Tuve error a la hora de añadir el objeto settings.DATABASES is improperly configured. Please supply the NAME value. Al parecer el nombre de la base estaba comentado dentro de settings.py, no se si viene por defecto cuando se crea Django o es un error comun}

Si entramos en la administración de Django, dentro de 'Registrados' veremos que se ha añadido correctamente el email. Vemos que falta el nombre, eso es porque en la vista no hemos hecho que aparezca, para ello creamos un segundo objeto.



Ahora ya nos muestra el nombre.

## 16. Model Form

Vamos a modificar 'forms.py' para crear nuestro propio model form. De momento solo tendrá el campo de 'email'.

```
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()
```

También habrá que tocar en la Administración de Django, nos vamos a 'admin.py' y creamos el model form.

```
1 from django.contrib import admin
2
3 # Register your models here.
4 from .models import Registrado
5 from .forms import RegModelForm
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
18
19 admin.site.register(Registrado, AdminRegistrado)
```

El resultado será como este:

The screenshot shows the Django Admin interface. The header is blue with the text 'Administración de Django' and 'BIENVENIDOS, KRIWANO. VER EL SITIO / CAMBIAR CONTRASEÑA / CERRAR SESIÓN'. The breadcrumb trail is 'Inicio > Boletín > Registrados > Añadir registrado'. The left sidebar has two main sections: 'AUTENTICACIÓN Y AUTORIZACIÓN' with links for 'Grupos' and 'Usuarios' (both with '+ Añadir' buttons), and 'BOLETÍN' with a link for 'Registrados' (with '+ Añadir' button). The main content area is titled 'Añadir registrado' and contains an 'Email:' label followed by a text input field. At the bottom, there are three buttons: 'Guardar y añadir otro', 'Guardar y continuar editando', and 'GUARDAR'.

Si queremos tener un campo para el nombre, lo insertamos en 'forms.py'

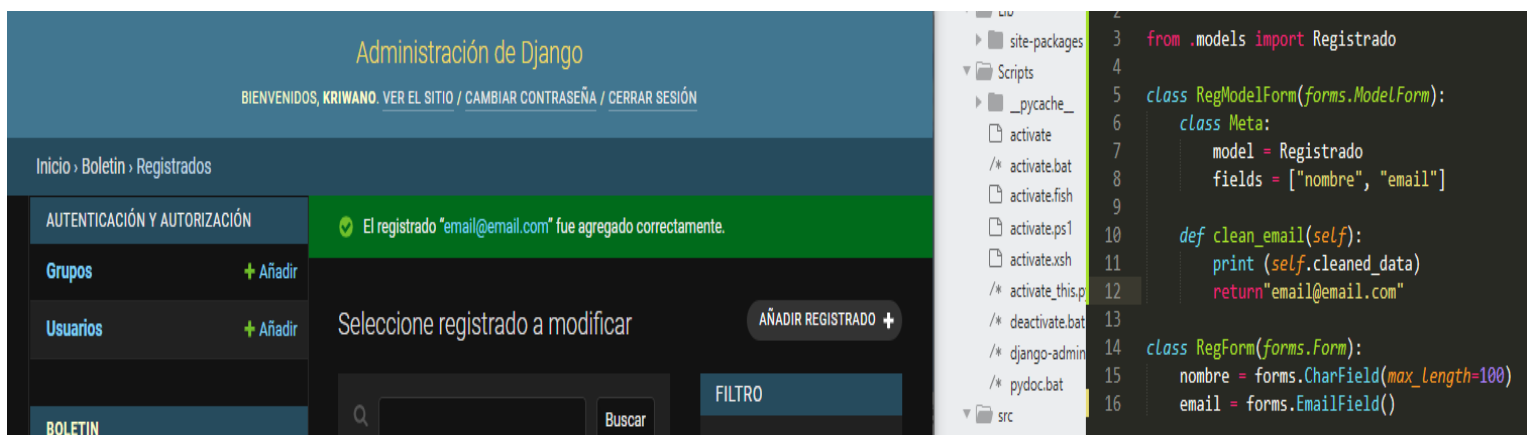
This block shows the updated Django Admin interface and the corresponding Python code in 'forms.py'. The interface on the left is similar to the previous one, but the 'Añadir registrado' form now includes a 'Nombre:' label and a text input field above the 'Email:' field. The code on the right, in a dark-themed editor, shows the following:

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

## 17. Validaciones Model Form

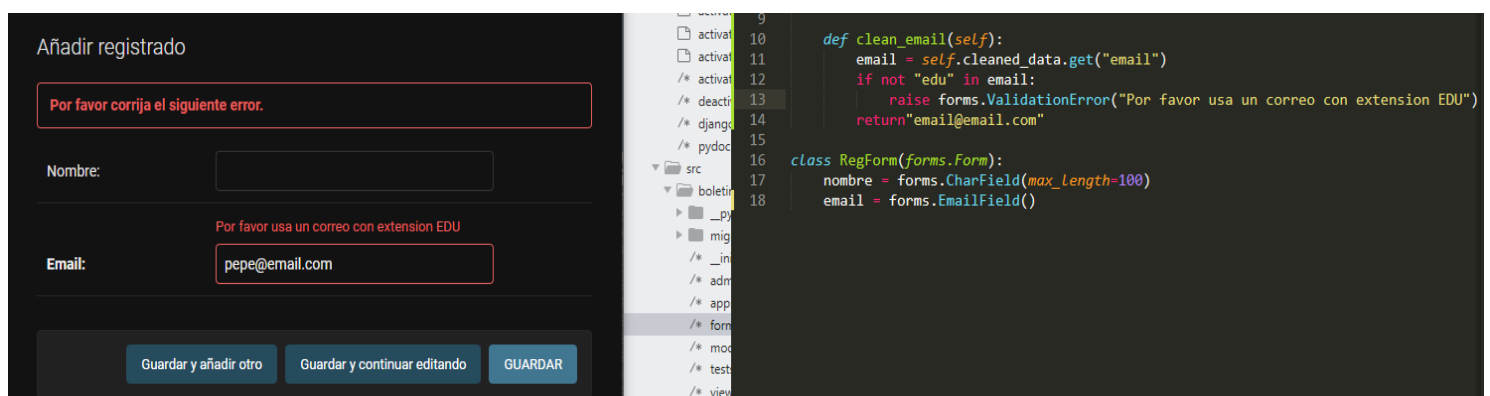
Como vimos anteriormente Django aplica sus propias validaciones según cada campo, esto es, si no escribimos un correo en el campo de 'email', no lo dará como válido. Además, tenemos que comprobar que esos datos no sean maliciosos para la base de datos.

Para ello, vamos a nuestro código y vamos a escribir métodos de Python para crear nuestras validaciones, por ejemplo, `clean_email`:



Si escribimos un correo cualquiera, nos saldrá una notificación diciendo que el email "[email@email.com](mailto:email@email.com)" ha sido agregado correctamente.

Vamos a reescribirlo un poco mejor, esta vez para que cuando escribamos un correo no educativo (.EDU) nos salga un error.



Pero esto no impide que pongamos un correo de nombre edu sin la extensión propia, si lo que queremos es que solo acepte la extensión, vamos a crear una nueva variable:

Añadir registrado

Por favor corrija el siguiente error.

Nombre:

Por favor usa un correo con extension EDU

Email:

edu@yo.com

Guardar y añadir otro

Guardar y continuar editando

GUARDAR

```
9
10
11
12
13
14
15
16
17
18
19
20
def clean_email(self):
    email = self.cleaned_data.get("email")
    email_base, proveedor = email.split("@")
    dominio, extension = proveedor.split(".")
    if not extension == "edu":
        raise forms.ValidationError("Por favor usa un correo con extension EDU")
    return email@email.com

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

Así funcionan las validaciones personalizadas.

## 18. Contexto en la vista, plantillas

Vamos a ver en detalle como funciona el contexto, las variables tanto en las plantillas como en las vistas.

En nuestra vista, vamos a agregar un titulo para nuestro html.

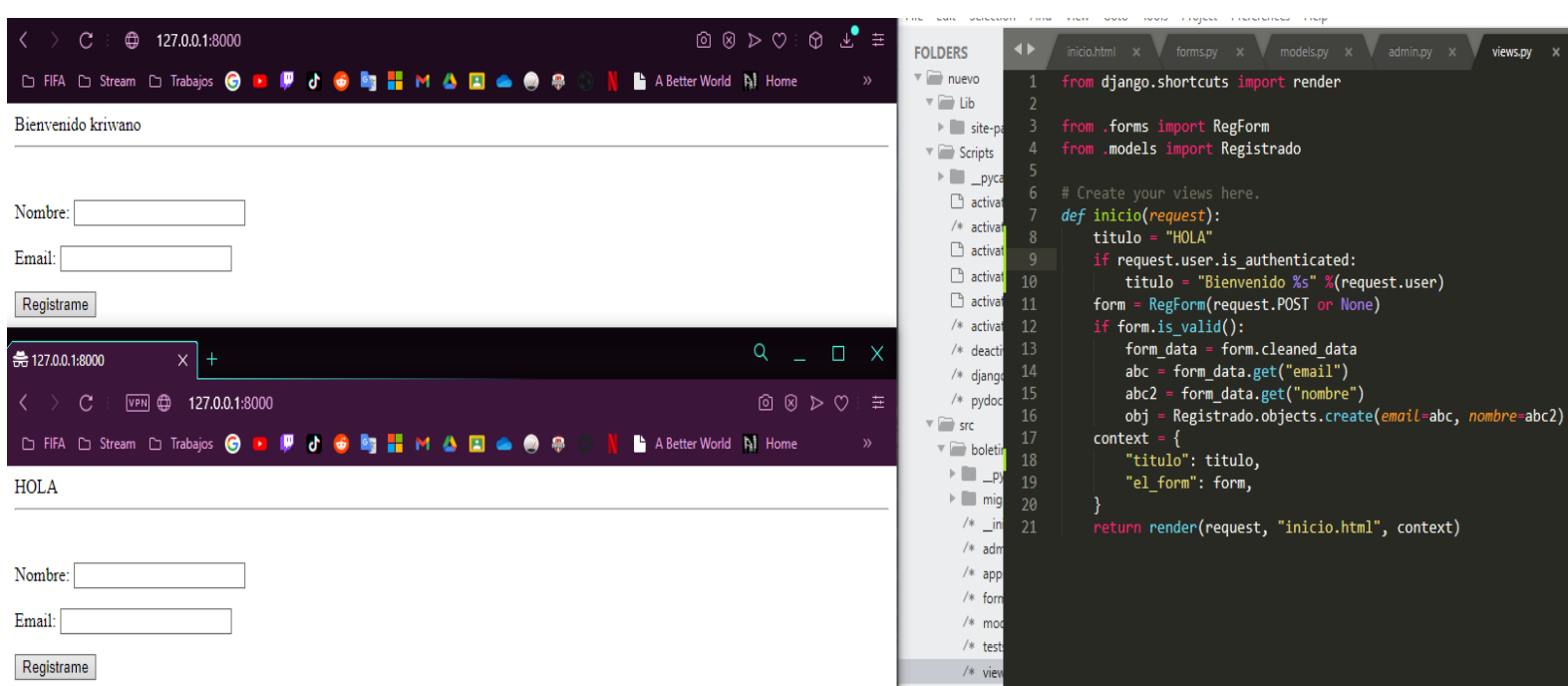
```
1  from django.shortcuts import render
2
3  from .forms import RegForm
4  from .models import Registrado
5
6  # Create your views here.
7  def inicio(request):
8      titulo = "HOLA"
9      form = RegForm(request.POST or None)
10     if form.is_valid():
11         form_data = form.cleaned_data
12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

A parte, lo añadiremos a 'inicio.html' haciendole referencia a 'titulo'.

```
1  <!--<h1>Hola mundo</h1> -->
2  {{ titulo }}
3  <hr/>
4  <br/>
5
6  <form method="POST" action="">{% csrf_token %}
7  {{ el_form.as_p }}
8  <input type='submit' value='Registrame' />
9  </form>
```



También podemos hacer que si el usuario no esta autenticado con su cuenta, le aparezca otro mensaje de bienvenida distinto:



{cuidado con poner () después del request.user.is\_authenticated, nos dará un error porque intentamos que un objeto actúe como si fuese un método o una función}

Para jugar un poco mas con nuestra plantilla, podemos agregar diferentes variables.

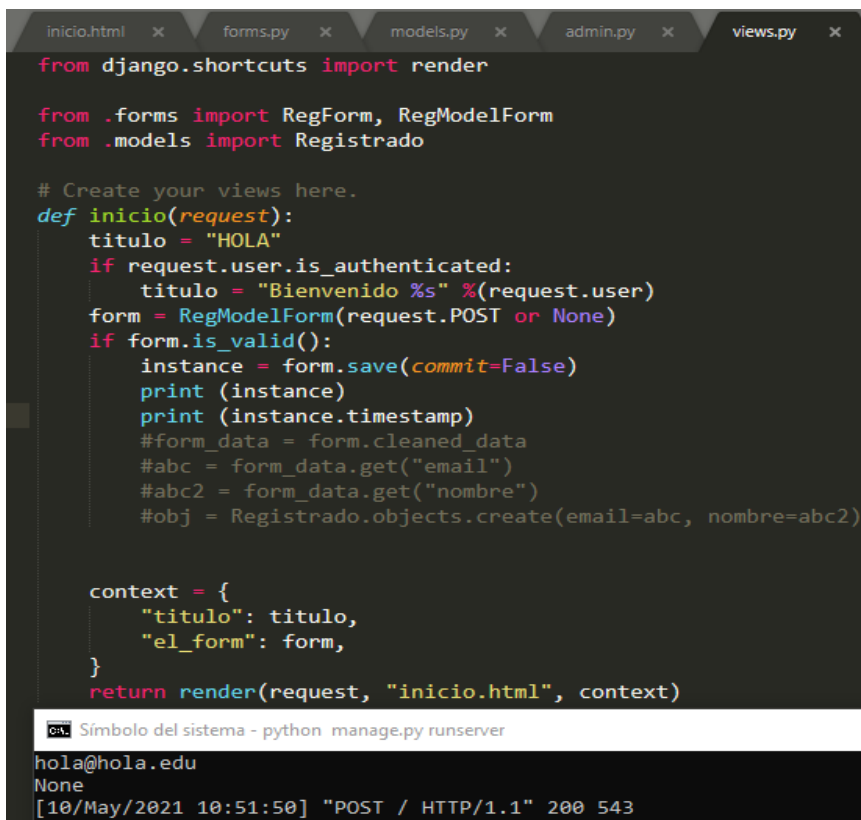


Donde 'request.user' somos nosotros y 'abc' sería una variable cualquiera.

## 19. Model Form en la vista

En nuestra vista renderizamos RegForm desde 'forms.py', se trata de un Custom Form. Vamos a sustituirlo por nuestro Model Form. Hacemos esto para guardar objetos nuevos usando el modelo.

Importamos 'RegModelForm' en la vista y creamos una instancia.



```
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)
    if form.is_valid():
        instance = form.save(commit=False)
        print (instance)
        print (instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)

    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

C:\> Simbolo del sistema - python manage.py runserver

hola@hola.edu  
None  
[10/May/2021 10:51:50] "POST / HTTP/1.1" 200 543

Podemos observar en la línea de comando que nos muestra el correo (recordad que nuestro unicode era 'email') pero no muestra el timestamp. Eso es debido a que tenemos que 'commit=False', eso impide que ese registrado nuevo se guarde, nos falta una línea más, 'instance.save()'

```

inicio.html x forms.py x models.py x admin.py x views.py x
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)
    if form.is_valid():
        instance = form.save(commit=False)
        instance.save()
        print(instance)
        print(instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)

    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)

```

hola@hola.edu  
2021-05-10 08:55:27.909109+00:00  
[10/May/2021 10:55:27] "POST / HTTP/1.1" 200 543

Confirmamos que el timestamp se acaba de guardar. Si os acordáis, en 'models.py' el campo 'nombre' no era obligatorio. Vamos a añadir una linea mas en nuestro modelo, para que en caso de que no se ponga un nombre de usuario, se agregue otro por defecto.

### Administración de Django

BIENVENIDOS, KRIWANO. [VER EL SITIO](#) / [CAMBIAR CONTRASEÑA](#) / [CERRAR SESIÓN](#)

CIÓN

+ Añadir

+ Añadir

+ Añadir

Seleccione registrado a modificar

AÑADIR REGISTRADO +

Buscar

FILTRO

Por timestamp

Cualquier fecha

Hoy

Últimos 7 días

Este mes

Este año

Acción:

Ir seleccionados 0 de 11

EMAIL	NOMBRE
a@hola.edu	PERSONA

```

3 from .forms import RegForm, RegModelForm
4 from .models import Registrado
5
6 # Create your views here.
7 def inicio(request):
8     titulo = "HOLA"
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         if not instance.nombre:
15             instance.nombre = "PERSONA"
16         instance.save()
17         print(instance)
18         print(instance.timestamp)
19         #form_data = form.cleaned_data
20         #abc = form_data.get("email")
21         #abc2 = form_data.get("nombre")
22         #obj = Registrado.objects.create(email=abc, nombre=abc2)
23
24

```

Para eso tenemos el 'commit=False', para escribir lógica o código y después de cumplirlo guardarlo.

Otra cosa que podemos hacer es modificar el contexto. Vamos a hacer que cuando nos registremos aparezca un mensaje con el nombre que se haya introducido. En caso de que no haya un nombre, podemos hacer que nos muestre el correo registrado.

Gracias pep@pe.edu!  
kriwano

Regístrate

```

3 from .forms import RegForm, RegModelForm
4 from .models import Registrado
5
6 # Create your views here.
7 def inicio(request):
8     titulo = "HOLA"
9     if request.user.is_authenticated:
10         titulo = "Bienvenido %s" %(request.user)
11     form = RegModelForm(request.POST or None)
12
13     context = {
14         "titulo": titulo,
15         "el_form": form,
16     }
17
18     if form.is_valid():
19         instance = form.save(commit=False)
20         nombre = form.cleaned_data.get("nombre")
21         email = form.cleaned_data.get("email")
22         if not instance.nombre:
23             instance.nombre = "PERSONA"
24         instance.save()
25
26         context = {
27             "titulo": "Gracias %s!" %(nombre)
28         }
29
30     if not nombre:
31         context = {
32             "titulo": "Gracias %s!" %(email)
33         }

```

Vemos que aún después de habernos registrado aparece ese botón de 'Regístrate'.  
Vamos a hacer que no se vea. Habrá que editar nuestra plantilla.

Gracias pep@pe.edu!  
kriwano

```

3 {{ titulo }}<br/>
4 {{ request.user }}<br/>
5 <hr/>
6 <br/>
7
8 {% if form %}
9 <form method="POST" action="">{% csrf_token %}
10 {{ el_form.as_p }}
11 <input type='submit' value='Regístrate' />
12 </form>
13 {% endif %}

```

## 20. Custom Form para Contacto

Para nuestra pagina principal de registro usamos el Model Form, pero en 'forms.py' teniamos un 'RegForm' que lo vamos a convertir en un formulario de contacto, por si alguien se quiere poner en contacto con nosotros.

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

El widget ese nos vale para que el mensaje sea mas grande, tenga mas espacio para escribirse.

Aparte, tendremos que crear una vista nueva para dicho contacto (recordad que hay que borrar la importación antigua y crear una nueva para ContactForm)

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

Dentro de las plantillas vamos a crear una nueva, 'forms.html' y agregar dicha a 'urls.py'

```
inicio.html x forms.html x forms.py x models.py  
{{ titulo }}<br/>  
{{ request.user }}<br/>  
<hr/>  
<br/>  
  
<form method="POST" action="">{% csrf_token %}  
{{ form.as_p }}  
<input type='submit' value='Registrate'/>  
</form>
```

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

Una vez acabado, nos quedará algo así:

kriwano

Nombre:

Email:

Mensaje:

Regístrate

Vamos a agregar unas validaciones

kriwano

Nombre: epep

Email: pep@pe.edu

hola

Mensaje:

Regístrate

```
46 def contact(request):
47     form = ContactForm(request.POST or None)
48     if form.is_valid():
49         email = form.cleaned_data.get("email")
50         mensaje = form.cleaned_data.get("mensaje")
51         nombre = form.cleaned_data.get("nombre")
52         print(email, mensaje, nombre)
53     context = {
54         "form": form,
55     }
56
57     return render(request, "forms.html", context)
```

Simbolo del sistema - python manage.py runserver

```
System check identified no issues (0 silenced).
May 10, 2021 - 11:52:04
Django version 3.2, using settings 'nuevo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[10/May/2021 11:52:04] "GET /contact/ HTTP/1.1" 200 571
pep@pe.edu hola epep
[10/May/2021 11:52:21] "POST /contact/ HTTP/1.1" 200 607
```

Si tenemos muchos campos y no queremos escribirlos todos podemos hacer:

```
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)
```

CA. Símbolo del sistema - python manage.py runserver

```
[10/May/2021 12:00:29] "POST /contact/ HTTP/1.1" 200 607
nombre
eep
email
pep@pe.edu
mensaje
hola
[10/May/2021 12:00:34] "POST /contact/ HTTP/1.1" 200 607
```

O de la siguiente manera {cambiar .iteritems por .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)
```

CA. Símbolo del sistema - python manage.py runserver

```
mensaje hola
[10/May/2021 12:07:15] "POST /contact/ HTTP/1.1" 200 607
nombre Ivan
email pep@pe.edu
mensaje hola
[10/May/2021 12:07:17] "POST /contact/ HTTP/1.1" 200 607
```

## 21. Configurar email.

Vamos a configurar nuestro correo electrónico para poder enviar mensajes a usuarios. Vamos a hacer el testing con nuestro formulario de contacto, con dicha información que nos enviaremos a nosotros mismos.

Primero de todo hay que agregar opciones en 'settings.py' (en un futuro habrá que desbloquear el captcha de Gmail)

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

En 'views.py' vamos a ir de nuevo a nuestro 'contact', después de importar nuestros settings (el `Fail_silently` nos sirve para que muestre el error del servidor ya que estamos haciendo un testing)

```
from django.conf import settings
from django.core.mail import send_mail
```

```
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 de ContactFor'
        email_from = settings.EMAIL_HOST_USER
        email_to = [email_from, "otroemail@gmail.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)
```



Vamos a hacer algunas pruebas {hay que configurar tu cuenta de Gmail para que acepte enviar mensajes desde otras aplicaciones, habilitar que apps tengan acceso a tu cuenta y habilitar el IMAP}

```
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 de Contact'
        email_from = settings.EMAIL_HOST_USER
        email_to = [email_from, "otroemail@gmail.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)
```

Si dejamos el 'fail\_silently' en False, se metemos un correo Gmail falso dará error, pero si lo dejamos en True, intentará conectarse al servidor sin éxito.

## 22. Configuración de archivos estáticos

Cuando hablamos de archivos estáticos nos referimos a css, imágenes y JavaScript. Tenemos que asegurarnos que dentro de 'settings.py' tenemos 'staticfiles' en 'INSTALLED\_APPS' y a continuación especificar la ruta donde guardaremos los archivos:

```
STATIC_URL = '/static/'
# /static/imagenes/img1.jpg

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static_pro", "static"),
    # '/var/www/static',
]
```

Vamos a crear una carpeta nueva dentro de nuestra raíz 'src' que se llame 'static\_pro', y dentro de ella otra llamada 'static'. Además, dentro de nuestro entorno virtual vamos a añadir una carpeta 'static\_env' para trabajar con dichos archivos. Esto nos vale para emular el tener nuestros archivos estáticos en otro servidor para producción.

También tenemos que especificar 'STATIC\_ROOT' donde ya vivirán los archivos en producción en otro servidor. Habrá que crear una carpeta 'static\_root' dentro de 'static\_env'.

```
STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env", "static_root")
```

También podemos hacer lo mismo para 'media', archivos estáticos subidos por terceros.

```
STATIC_URL = '/static/'
STATIC_URL = '/media/'
# /static/imagenes/img1.jpg

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static_pro", "static"),
    # '/var/www/static',
]

STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env", "static_root")
MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env", "media_root")
```

Lo siguiente sería configurar las URLs (solo para desarrollo, para producción no vale)

Si 'DEBUG = True' estamos en desarrollo

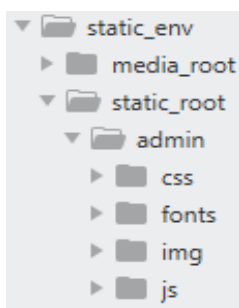
```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True
```

```
from django.contrib import admin  
from django.urls import path  
  
from django.conf import settings  
from django.conf.urls.static import static  
  
from boletin import views  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', views.inicio, name='home'),  
    path('contact/', views.contact, name='contact'),  
]  
  
if settings.DEBUG:  
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)  
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Tenemos que ejecutar el comando para enviar nuestros archivos estáticos al servidor.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py collectstatic  
128 static files copied to 'C:\Users\kriwa\Documents\nuevo\static_env\static_root'.
```

Si vamos a la carpeta 'static\_root' veremos que se ha creado unas carpetas con fuentes, css, imagenes... pero todavía no tenemos archivos estáticos.



## 23. Configuración Bootstrap

Bootstrap es un framework para diseño adaptable, cuando cambiemos el tamaño del navegador según el dispositivo, este se adapta perfectamente. Vamos a añadir un poco de diseño a nuestro proyecto.

Vamos a coger el código fuente de uno de los ejemplos de Bootstrap y añadirlo a nuestro proyecto. Creamos una nueva plantilla dentro de 'Templates' llamada 'base.html' y vamos a pegar todo el código ahí.

En 'views.py' estabamos renderizando 'inicio.html', lo cambiamos a 'base.html'. Volvemos a ejecutar el servidor, abrimos nuestra pagina web y se nos abrirá con el nuevo diseño.



### Navbar example

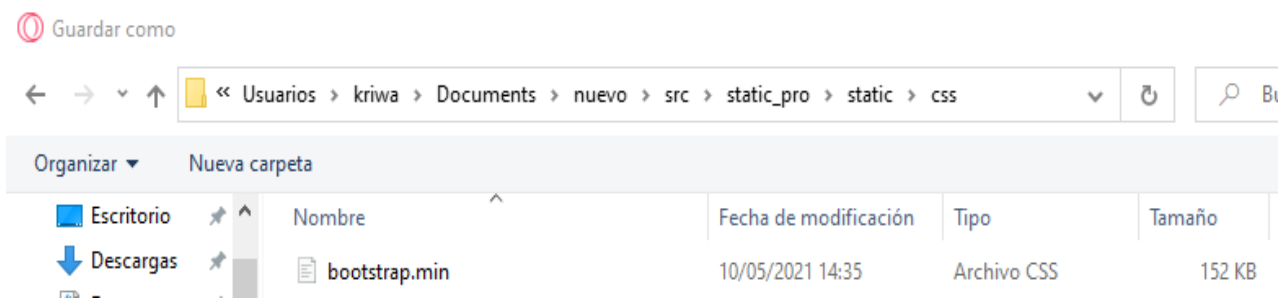
This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

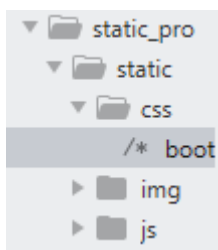
Está renderizando la plantilla pero el css no. Nos faltan por configurar un par de cosas. Tenemos que añadir las hojas de estilo a nuestro proyecto.

```
<!-- Bootstrap core CSS -->
<link href="/docs/5.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55wLwNGmvv4Um1vskellj0" crossorigin="anonymous">
```

Lo abrimos y lo guardamos en una nueva carpeta



Vamos a realizar esto mismo para img y js.



Tenemos que añadir una etiqueta 'load static' para poder utilizar esas hojas de estilo dentro de 'base.html'.

```
<!doctype html>
{% load static %}
```

Y referenciarlo a nuestras carpetas

```
<!-- Bootstrap core CSS -->
<link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet" integrity="
sha384-wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55WWLwNGmvvx4Um1vskeMj0"
crossorigin="anonymous">
```

Haremos esto para todos los archivos .css

Volvemos a recargar nuestra pagina y veremos como se han cargado correctamente los estilos.



## 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 »](#)

Como práctica de buenas maneras, y aunque no tengamos nada nuestro todavía subido, vamos a volver a escribir el comando para subir nuestros archivos estáticos.

```
(nuevo) C:\Users\kriwa\Documents\nuevo\src>python manage.py collectstatic

You have requested to collect static files at the destination
location as specified in your settings:

    C:\Users\kriwa\Documents\nuevo\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

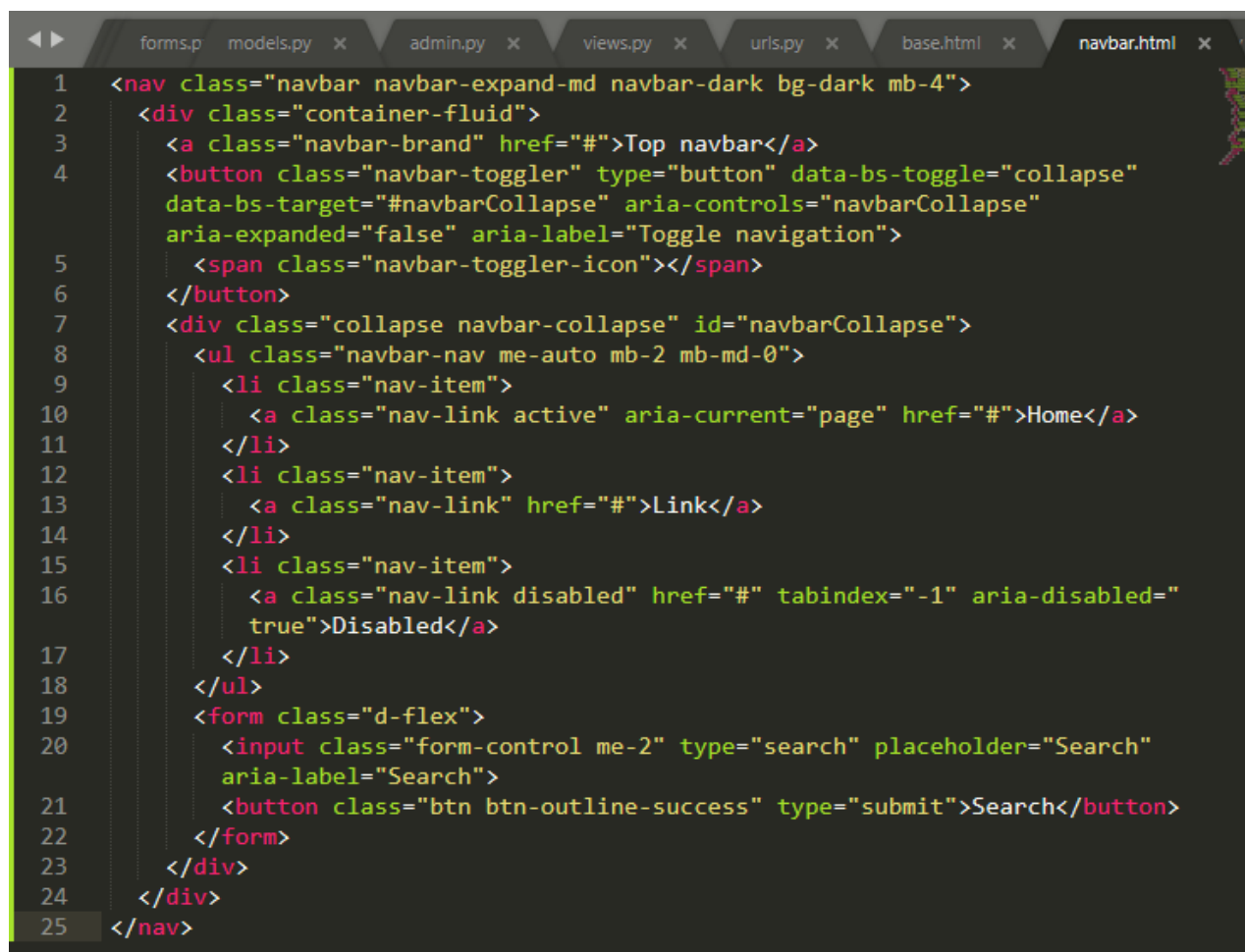
2 static files copied to 'C:\Users\kriwa\Documents\nuevo\static_env\static_root', 128 unmodified.
```

Ahora volveríamos a hacer lo mismo para JavaScript.

## 24. Plantillas

Las plantillas son renderizadas en las vistas junto a la petición y al contexto. Si abrimos 'base.html' veremos que hay una barra de navegación, fundamental en estos casos. Lo que vamos a hacer es que las demás plantillas hereden este elemento para que se renderizen siempre sin repetir código.

Como ejemplo, creamos una plantilla nueva y pegamos la parte que queremos heredar.



```
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"
5       data-bs-target="#navbarCollapse" aria-controls="navbarCollapse"
6       aria-expanded="false" aria-label="Toggle navigation">
7       <span class="navbar-toggler-icon"></span>
8     </button>
9     <div class="collapse navbar-collapse" id="navbarCollapse">
10      <ul class="navbar-nav me-auto mb-2 mb-md-0">
11        <li class="nav-item">
12          <a class="nav-link active" aria-current="page" href="#">Home</a>
13        </li>
14        <li class="nav-item">
15          <a class="nav-link" href="#">Link</a>
16        </li>
17        <li class="nav-item">
18          <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="
19            true">Disabled</a>
20        </li>
21      </ul>
22      <form class="d-flex">
23        <input class="form-control me-2" type="search" placeholder="Search"
24          aria-label="Search">
25        <button class="btn btn-outline-success" type="submit">Search</button>
26      </form>
27    </div>
28  </div>
29</nav>
```

En 'base.html' donde ha quedado el hueco vacío ponemos una etiqueta haciendo referencia a 'navbar.html'

```
{% include "navbar.html" %}
```

Una vez familiarizado con el código y como funciona la herencia entre plantillas, podemos seguir haciendo mas ejemplos. Vamos a hacer lo mismo ahora pero con 'navbar example'