FUNDAMENTOS DE DJANGO

INTRODUCCION A DJANGO

Introducción a django

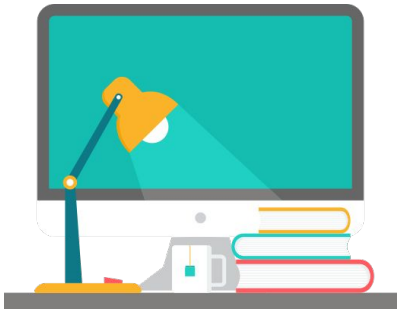
¿QUE ES DJANGO?

Django es un framework de desarrollo web de código abierto, escrito en el lenguaje de programación **Python**, el framework fue nombrado en alusión al guitarrista de jazz gitano **Django Reinhardt**. En junio de 2008 y fue anunciado que la recién formada **Django** Software Foundation se haría cargo de **Django** en el futuro

¿Que es Framework y porqué usarlo?

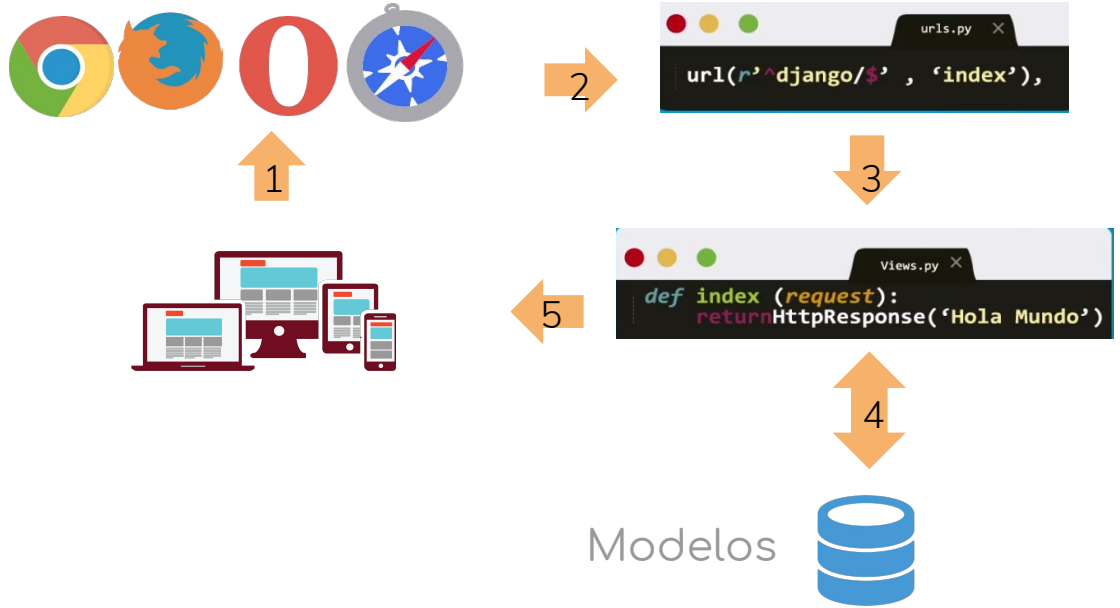
En un concepto muy simple, un **Framework** es un esquema, un esqueleto o estructura para el desarrollo y/o la implementación de un sistema Web .

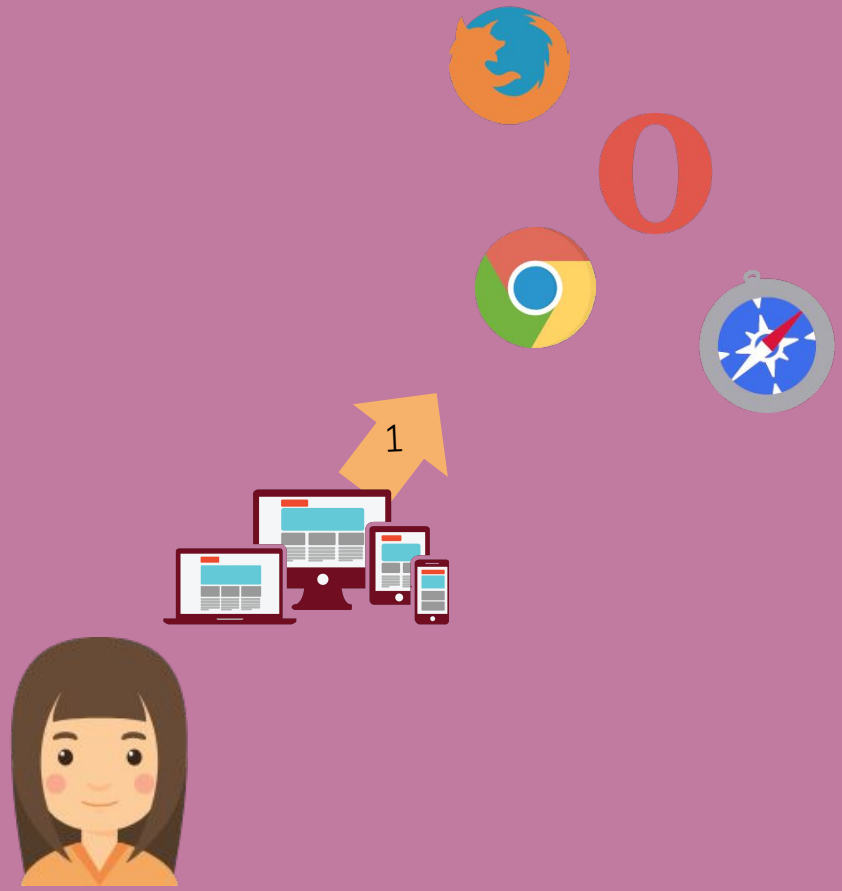
Los **frameworks** hacen posible ahorrar tiempo en la construcción de páginas web.



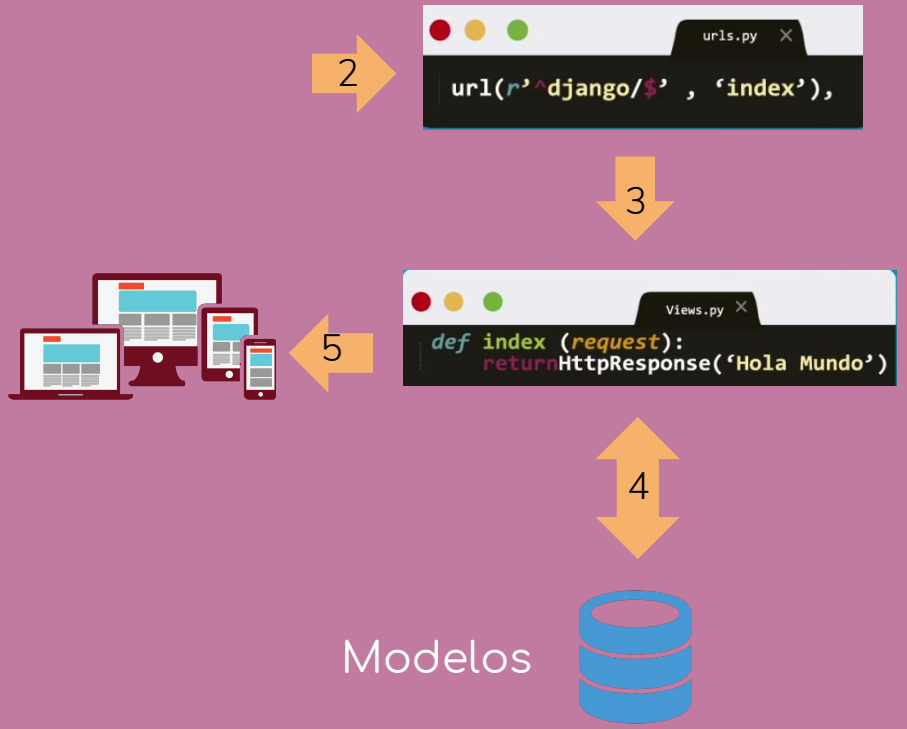
¿Cómo trabaja Django con las páginas web?

En la imagen se puede apreciar el flujo de trabajo de **Django**, pues este se encuentra numerado y detallado en las siguientes secciones.





1. El usuario accede a la página web desde un navegador que podría encontrarse instalado en un celular, tablet, computadora de escritorio o laptop



2. El navegador se dirige a una dirección de sitio Web, ejemplo (https://www.google.com/) esa solicitud es realizada en el archivo **urls.py** del proyecto django

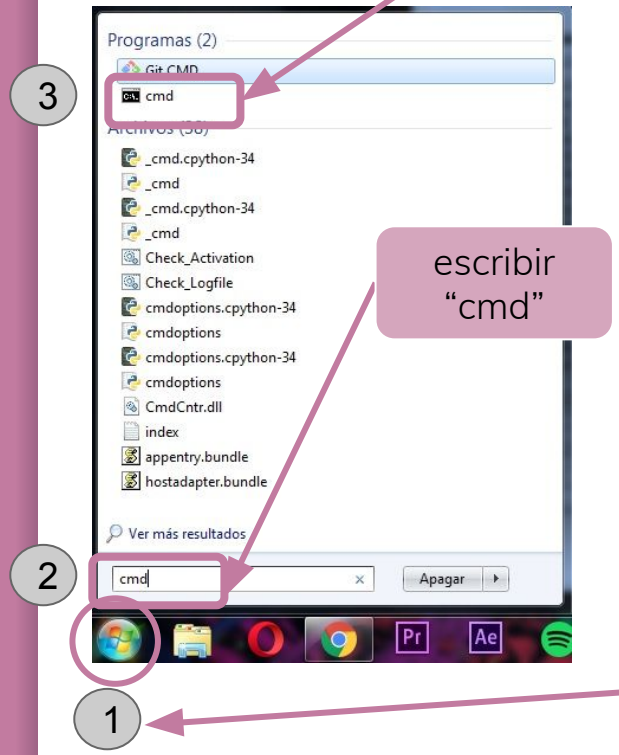
3. Lo que hace esa petición es ir a la vista del archivo llamado **views.py** este contiene la parte visual que la pagina mostrara

4. Si la vista(**views.py**) requiere de más datos, estos los solicitara de una base de datos

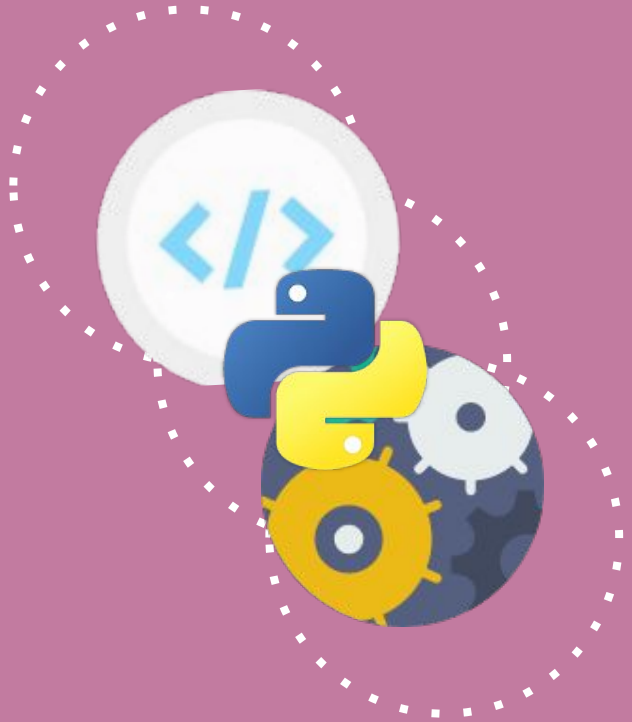
5. Finalmente los datos solicitados serán mostrados en tu celular, tablet, laptop

AMBIENTE DE DESARROLLO

INSTALACION DE PIP



Para instalar PIP se debe ir al botón de inicio de windows y escribir cmd, esto no abrirá una ventana que es conocida como la terminal, es aquí donde se debe ingresar el comando de instalación de PIP.



¿Que es PIP?

PIP

Es una herramienta para administrar e instalar paquetes de **python**, esto permite que aislar las dependencias(librerías, recursos) de nuestros proyectos.

INSTALACION DE PIP

Podemos instalar PIP con los siguientes comandos:

LINUX

sudo apt-get install python-pip

WINDOWS

1. Abrir el enlace
2. Presionar CTRL + S o CTRL + G , esto nos permitirá guardar el archivo en nuestra computadora.
3. El archivo debe ser guardado en la ruta C:\PythonXY\ , con el nombre **get-pip.py** , o en todo caso en la ruta en la que esté instalado python
4. Ejecuta el comando **python.exe get-pip.py**
5. Instalar un paquete con el comando pip install nombre_del_paquete

INSTALACION DE PIP

```
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: C8DC-FB9B

 Directorio de C:\Python34

17/08/2018  23:52    <DIR>          .
17/08/2018  23:52    <DIR>          ..
17/08/2018  23:31    <DIR>          DLLs
17/08/2018  23:31    <DIR>          Doc
17/08/2018  23:52         1.663.173 get-pip.py
17/08/2018  23:31    <DIR>          include
17/08/2018  23:31    <DIR>          Lib
17/08/2018  23:31    <DIR>          libs
09/03/2014  20:25            31.073 LICENSE.txt
09/03/2014  20:14           334.665 NEWS.txt
09/03/2014  20:24            27.136 python.exe
09/03/2014  20:24            27.648 pythonw.exe
09/03/2014  20:14             6.982 README.txt
17/08/2018  23:54    <DIR>          Scripts
17/08/2018  23:31    <DIR>          tcl
17/08/2018  23:31    <DIR>          Tools
               6 archivos      2.090.677 bytes
              10 dirs  57.988.546.560 bytes libres

C:\Python34>
```

Dentro de nuestra terminal se debe ir al lugar donde se instaló python, dentro de esta carpeta podemos encontrar varios archivos de la instalación del lenguaje de programación **Python**

Con la ejecución del comando **python.exe get-pip.py** con el que comenzara la instalación de pip en nuestra computadora

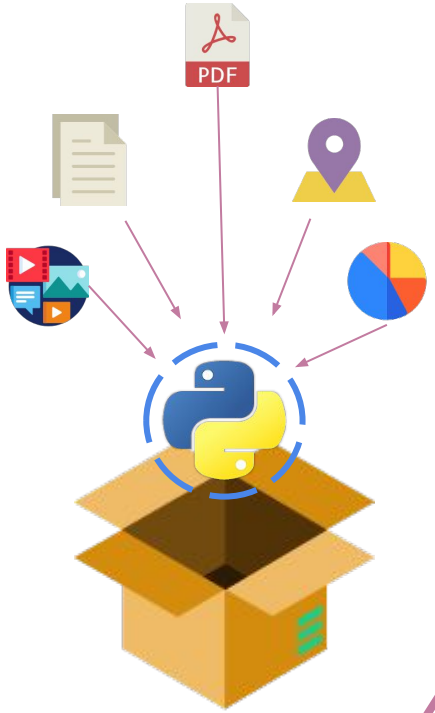
```
C:\Python34>python.exe get-pip.py
Collecting pip
  Using cached https://files.pythonhosted.org/packages/5f/25/e52d3f31441505a5f3
f41213346e5b6c221c9e086a166f3703d2ddaf940/pip-18.0-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 18.0
    Uninstalling pip-18.0:
      Successfully uninstalled pip-18.0
Successfully installed pip-18.0

C:\Python34>
```

Entorno Virtual (Virtualenv)

Para instalar **Django**, y crear nuestro proyecto denominado blog, primero instalaremos una herramienta **VIRTUALENV** este nos ayudará a mantener un entorno de desarrollo ordenado de nuestro proyecto. Es posible omitir este paso, pero es recomendable no hacerlo - ¡comenzar con la mejor configuración posible nos ayudará a evitar muchos problemas en el futuro!

INSTALACION DE VIRTUALENV



Podemos instalar virtualenv con el siguiente comando:

pip install virtualenv

Activar virtualenv

En **windows** -> .\Scripts\activate

En **Linux** -> source env/bin/activate

```
(env) CursoDjango$ pip install django
```

Listar paquetes del virtualenv

pip list o pip freeze --local

Desinstalar un paquete

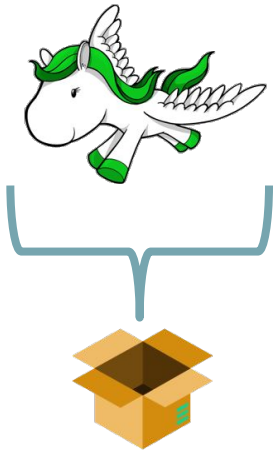
pip uninstall nombre_paquete

Detalle de un paquete

Pip show nombre_paquete

(env) nos indica que el entorno virtual actualmente está activo

Entorno Virtual



Virtualenv

Para instalar **Django** en nuestro entorno virtual se lo debe hacer con el siguiente comando:

```
(env)CursoDjango / pip install django
```

Instalación De paquetes de terceros

Django cuenta con un sin fin de librerías para diferentes propósitos en awesome django podemos ver toda la lista.

La instalación de las librerías se la realiza con el comando pip seguido del nombre del paquete ejemplo: pip install nombre_paquete

```
(env)CursoDjango$ pip install Pillow
```

```
(env)CursoDjango$ pip install django-geoposition
```

```
(env)CursoDjango$ pip install -U channels
```

```
(env)CursoDjango$ pip install djangorestframework
```

ORGANIZACIÓN DE UN PROYECTO

Vamos a Crear nuestro BLOG

¿QUE ES UN BLOG?

Un **blog** o **bitácora** es un sitio web que incluye, a modo de diario personal de su **autor** o autores, contenidos de su interés, que suelen estar actualizados con frecuencia y a menudo son comentados por los lectores.

Organización de un proyecto

Para crear el proyecto **blog** tenemos que crear una carpeta para tener dentro organizado nuestro proyecto **blog**, con la ayuda de la terminal crearemos una carpeta

```
$ mkdir ProyectoBlog
```

Este comando nos permitirá crear una carpeta con el nombre **ProyectoBlog**, podemos ingresar a este con el comando **cd**

```
$cd ProyectoBlog
```

Dentro de esta carpeta comenzaremos a crear el proyecto denominado **blog**

Organización de un proyecto

INICIAR UN PROYECTO DJANGO

Para crear el proyecto **blog** en **django** se debe ejecutar el siguiente comando en la consola:

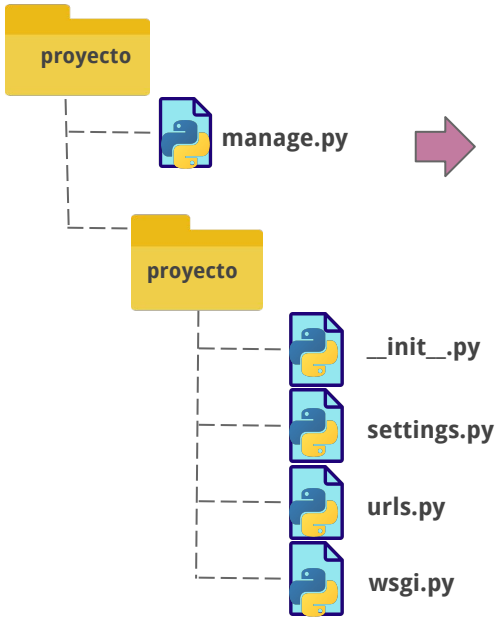
```
(env) CursoDjango $ django-admin startproject blog
```

Dónde "**blog**" viene siendo el nombre de nuestro proyecto, puedes cambiarlo por cualquier otro.

Estructura de un proyecto

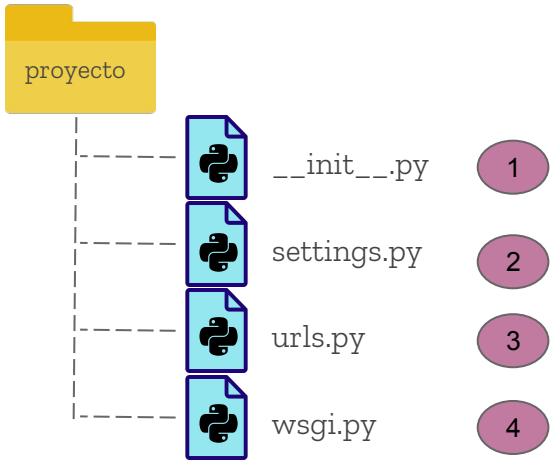
ESTRUCTURA DE UN PROYECTO DJANGO

La ejecución del comando anterior creará la siguiente estructura de carpetas y archivos.



Es el archivo de ejecución para el proyecto **django**

ESTRUCTURA DE UN PROYECTO



1.- Archivo de configuración hace posible que la carpeta proyecto sea un módulo.

2.- Archivo de configuración, este contiene la conexión a base de datos, archivos estáticos, y demás configuraciones.

3.- Archivo de configuración de las rutas o direcciones web.

4.- Es el punto de entrada para configuraciones con el servidor.

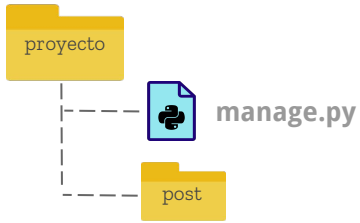
Estructura de las aplicaciones

CREAR UNA APP (Aplicación)EN DJANGO

Para crear una aplicación en un proyecto **django** se debe ejecutar el siguiente comando:

```
(env) CursoDjango/apps$ django-admin startapp post
```

publicación es el nombre que se le da a la aplicación



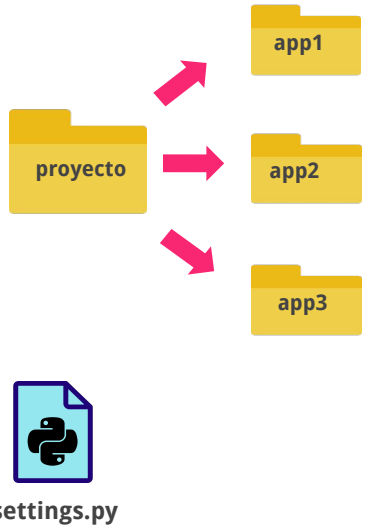
Estructura de las aplicaciones

ESTRUCTURA DE UNA APP (Aplicación) DJANGO

- **__init__.py** Convierte la carpeta home en Módulo
- **admin.py** El administrador django
- **models.py** Todos los modelos de la app
- **views.py** Todas nuestras Vistas
- **tests.py** Todas nuestras pruebas unitarias
- **forms.py** Nuestros Formularios
- **urls.py** Urls de la app

Configuración de un proyecto django

CONFIGURACIÓN DE UNA APP(Aplicación) DJANGO




```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.humanize',
    'apps.app1,
    'apps.app2,
    'apps.app3',
]
```

Las aplicaciones que creadas deben ser importadas en el archivo **settings.py**, solo asi se podra hacer uso de ellas.

MODELOS Y BASE DE DATOS

Conexión a base de datos

Pagina Web

Django

Base de Datos

Los datos que se manejan en el proyecto deben ser guardados en una **base de datos**(Es un almacén donde guardan grandes cantidades de información, para luego poder utilizarlo según nos convenga).

Conexión a base de datos

Django

Base de Datos

.

.

Las configuraciones para la base de datos del proyecto **blog** se encuentran en el archivo **settings.py**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'proyectoblog'),
    }
}
```

Campos y relaciones

Lo que necesitamos ahora es crear algo para almacenar todos los posts en nuestro **blog**. Pero para poder hacerlo tenemos que hablar un poco de acerca de algo llamado objetos .

Hay un concepto en el mundo de la programación llamado **programación orientada a objetos** . La idea es que en lugar de escribir todo como una aburrida secuencia de instrucciones de programación podemos modelar cosas y definir cómo interactúan con las demás.

Campos y relaciones

Entonces ¿Qué es un **objeto**? Es un conjunto de **propiedades** y acciones. Suena raro, pero te daremos un ejemplo.

Si queremos un modelar un **gato** crearemos un objeto Gato que tiene algunas propiedades, como son por ejemplo color , edad , estado de ánimo (es decir, bueno, malo, sueño), dueño (que es un objeto Persona).

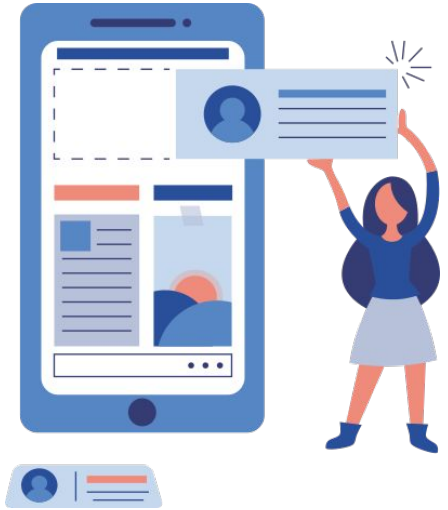
Campos y relaciones

Y luego el Gato tiene algunas acciones: ronronear , rasguñar o alimentarse (en la cual daremos al gato algunos ComidaDeGato , que podría ser un objeto independiente con propiedades, como por ejemplo, sabor).

```
Gato
---------
color
edad
dueño
ronronear()
rasguñar()
Alimentarse
(comida_de_gato)
ComidaDeGato
```

Básicamente se trata de describir cosas reales en el código con propiedades (llamadas propiedades del objeto) y las acciones (llamadas métodos).

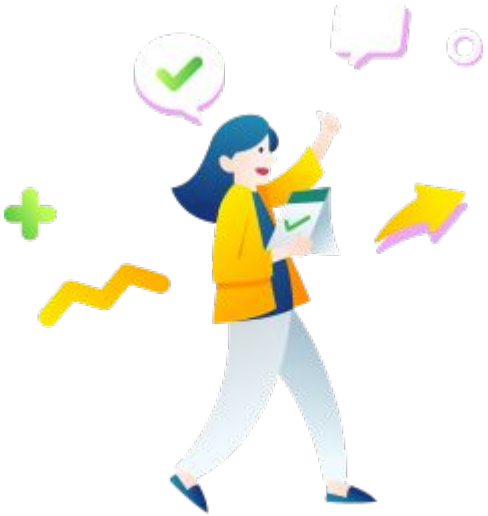
POST DEL BLOG



¿Qué es un **post** de un **blog**? ¿Qué características debe tener?

Bueno, seguro que nuestros posts necesitan un texto con su contenido y un título, ¿cierto? También sería bueno saber quién lo escribió, así que necesitamos un autor. Por último, queremos saber cuándo el **post** fue creado y publicado.

Datos para el blog



Entonces el post tendrá los siguientes campos:

```
BLOG
----------------
titulo
contenido
imagen
autor
fecha
```

Migraciones

Dentro de la aplicación **post** vamos a crear un modelo(Un modelo en Django es un tipo especial de objeto que se guarda en la base de datos), el cual contendrá los datos para las publicaciones

En el archivo **post/models.py** definimos todos los objetos llamados Models - este es un lugar en el cual definiremos nuestro modelo post. Vamos abrir **post/models.py** , quitamos todo y escribimos un código como el siguiente:

Modelos

```
from django.db import models

class Post(models.Model):
    titulo = models.CharField(max_length=255)
    contenido = models.TexTField()
    imagen = models.ImageField(upload_to="imagenes/post")
    autor = models.CharField(max_length=255)
    fecha = models.DateTimeField(default=timezone.now)

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

1 2 3 4 5 6 7 8 9

Las siguientes líneas nos permitirán crear nuestro modelo llamado Post el cual nos permitirá poder almacenar registros en nuestro **blog**

MODELOS

La primera línea comienza con la palabra **from** son las importaciones propias de django para crear el modelo Post.

```
1  from django.db import models
```

Con esta línea lo que le decimos a **django** es que importe de django.db la librería **models**, esto nos permitirá crear nuestro modelo al que llamaremos **Post**

MODELOS

La segunda línea define nuestro modelo se hace uso de la palabra reservada **class** seguido del nombre de la clase Post

```
2   class Post(models.Model):
```

importación que hicimos de **models** la ocuparemos para la creación de nuestro modelo el cual va referenciado entre los paréntesis.

MODELOS

Ahora definimos las propiedades del **Post**

```
titulo = models.CharField(max_length=255)
contenido = models.TexTField()
imagen = models.ImageField(upload_to=”imagenes/post”)
autor = models.CharField(max_length=255)
fecha = models.DateTimeField(default=timezone.now)
```

3 4 5 6 7

models.CharField -> Define un texto con el límite de 255 caracteres como máximo.

models.TextField -> Define los textos largos sin limite de caracteres.

models.ImageField -> Define la ruta en la que se guardará la imagen de post el cual será en la carpeta images/post

models.DatetimeField -> Define la hora y fecha del registro de nuestro post

MODELOS

Ahora definimos una función para nuestro **Post**

Esta función nos permitirá poder visualizar de manera correcta el título de nuestro post

```
def __str__(self):      8
    return self.titulo  9
```

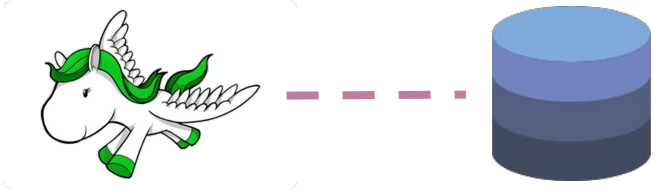
def __str__(self): -> es una función la cual hace posible ver el título de nuestro post

MIGRACIONES

Creando tablas

Creando modelos para nuestros post

En hora buena estamos a punto de crear nuestro modelo y poder registrar los posts



Django Base de Datos

Ejecutando el comando **python manage.py makemigrations post**, hara que nuestro modelo post pase a ser creado en una base de datos

```
(env) ~/CursoDjango$ python manage.py makemigrations blog
Migrations for 'blog':
  0001_initial.py:
   - Create model Post
```

Creando tablas

Migrando la tabla post

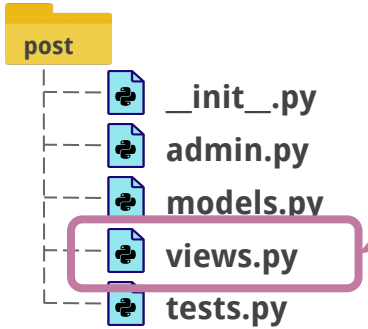
Ahora que tenemos el archivo de migración de nuestro modelo, podemos hacer la migración a nuestra **base de datos**, con el siguiente comando

```
(env) ~/CursoDjango$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Applying blog.0001_initial... OK
```

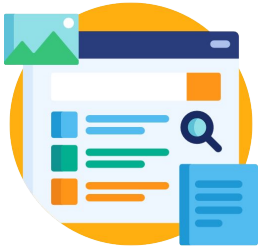
Veremos que todo salio con exito si obtenemos este resultado

VISTAS Y MOTOR DE TEMPLATES

Vistas de Django



Recordemos que en nuestra aplicación post tenemos el archivo **views.py**, vamos a hacer uso de este para crear la parte visual de nuestro proyecto



Una **vista** puede ser entendida como la parte visual, y lo que mostraremos específicamente sera los post que creemos en nuestro **blog**

Vistas de Django

```
from django.shortcuts import render
from .models import Post
```

```
def listar_post(request):
    posts = Post.objects.all()
    mensaje = "Necesitamos más mujeres en Tecnología"
    return render(request,'blog_post.html',{
                    'posts':posts , 'mensaje':mensaje})
```

django.shortcuts -> Hace posible que podamos visualizar nuestra vista

import Post -> Esta importación hará posible visualizar los datos registrados en nuestro blog

def -> define la creación de nuestra vista

listar_post -> define el nombre de la vista, para listar los posts

return -> Hace posible que nos regrese una respuesta de la función listar_post

render -> Hace posible que podamos observar nuestra vista mucho más amigable.

Importando vistas

Hemos definido nuestra primera vista, ahora para poder verlo, debemos crear e importar nuestra vista en el archivo **urls.py**

```
from django.urls import include, path      1
from .views import listar_post             2

urlpatterns = [                            3
    path('index/', views.listar_post, name='main-view'),   4

]
```

1. Con esta línea lo que le decimos a **django** es que importe de django.urls, y con esto podremos visualizar nuestra vista.

2. Importamos nuestra vista **lista_post** que se encuentra en el archivo **models.py.**

3. urlpatterns es el contenedor que almacena nuestra vista

4. Esta línea hará posible ver la vista de los posts en la dirección url 'index'

Plantillas en django

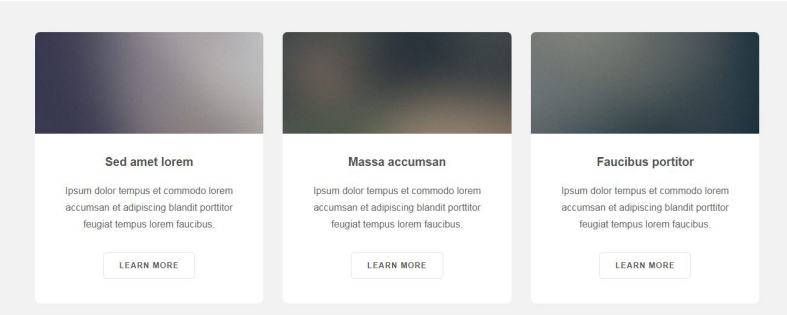
Tenemos todo listo para mostrar los posts de nuestro blog, solo queda realizar unos cuantos pasos.

Veamos qué es **HTML**

HTML es un simple código que es interpretado por tu navegador web - tal es el caso de Google Chrome, Firefox , Opera, Safari.

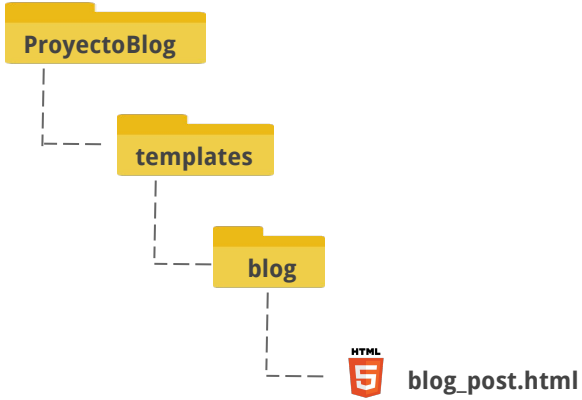
Tu primera plantilla

Crearemos nuestra plantilla, y con este podremos visualizar nuestros post de una manera más elegante, cada post podrá ser visualizado dentro de un section(es un contenedor gráficamente podemos verlo como una caja, como en la imagen que tenemos abajo)



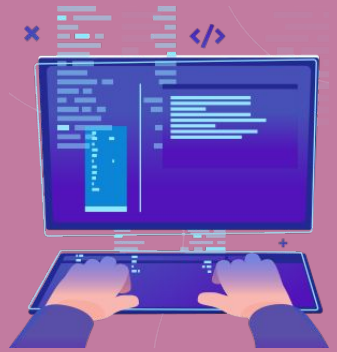
Plantillas en Django

Dentro de nuestra carpeta **ProyectoBlog** crearemos las siguientes carpetas esto con el fin de tener organizado un poco más nuestro proyecto:

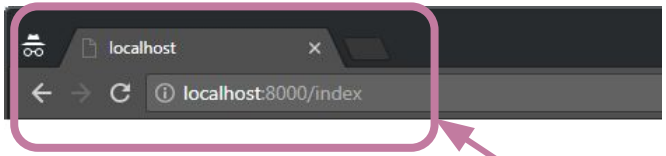


Nuestras plantillas las guardaremos en la carpeta blog que se encuentra dentro de la **templates**

Plantillas en Django



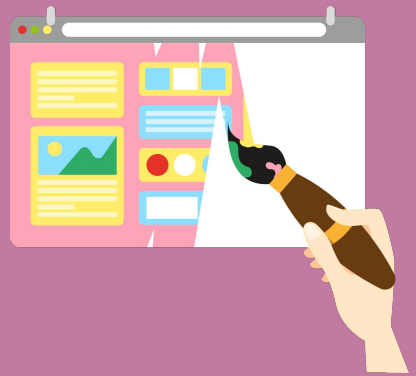
Dentro de la carpeta ProyectoBlog/templates/blog, crearemos un archivo con el nombre **blog_post.html** (lo dejaremos en blanco por ahora)



Dirección url de nuestra vista

Si abrimos nuestro navegador web sea **Chrome, Firefox, Opera, Safari** e ingresamos a localhost:8000/index podremos notar que no se ve nada, y es que nuestro archivo **blog_post.html** no tiene nada escrito vamos a escribir código y hacer que los datos de nuestro post puede verse.

Plantillas en Django



Vamos a crear nuestra primera **plantilla**

```
<html>
    <head>
        <title>Blog</title>
    </head>
    <body>
        <p>Bienvenido</p>
        <p>a mi Blog</p>
    </body>
</html>
```

La etiqueta más básica, **<html >**, es siempre el principio de cualquier página web y **</html >** es siempre el final. Como puedes ver, todo el contenido de la página web va desde el principio de la etiqueta <html > y hasta la etiqueta de cierre **</html >**

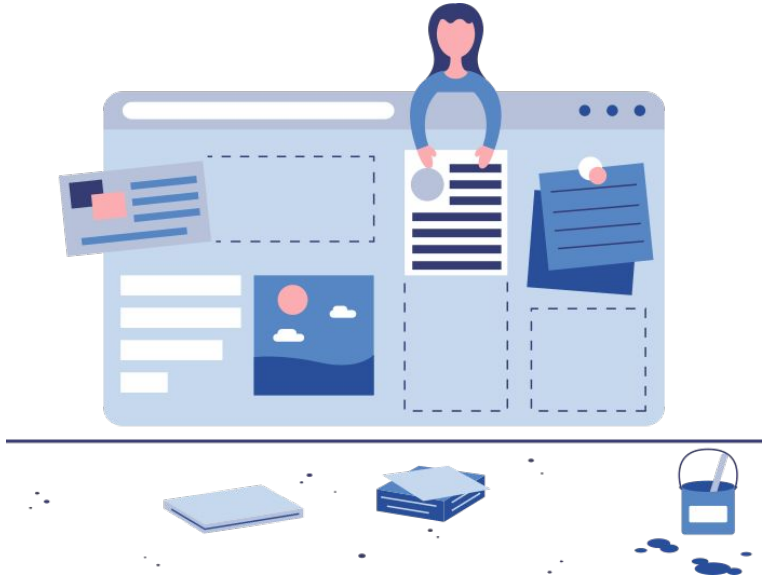
<head> </head> es el contenedor del título de la página web

<title></title> Es la etiqueta para asignar título a la página Web

<body></body> Etiqueta que contiene el cuerpo de la página web.

** ** es una etiqueta para los elementos de párrafo; ** ** cierra cada párrafo

Sistema de plantillas



Plantillas de Django

Template tags

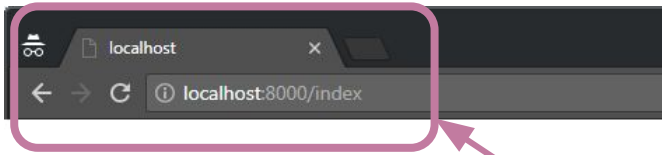
Para poder mostrar el mensaje que nos proporciona la vista **listar_post** utilizaremos **tags**

{{ comentario }}

Un tag no puede mostrarse sola debe ir dentro de una etiqueta, para nuestro blog utilizaremos la etiqueta ** ** quedando así nuestra etiqueta.

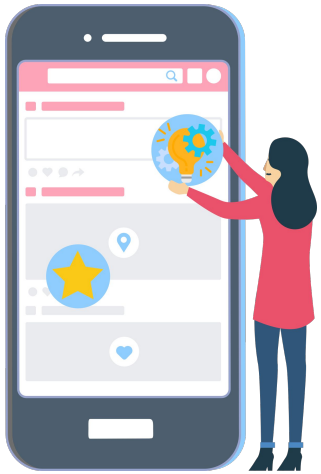
** {{ comentario }} **

Vamos a ver estos cambios en nuestro navegador web



Dirección url de nuestra vista

Visualizar nuestros posts



Vamos a agregar mas código para poder mostrar los posts que tiene nuestro **blog**

Template tags

Modificaremos un poco nuestro código html, todo los cambios lo realizaremos dentro de la etiqueta **<body></body>**

```
<html>
    <head>
        <title>Blog</title>
    </head>
    <body>

    <div>
        <h1><a href="">Bienvenido a mi Blog</a></h1>
    </div>

    {% for post in posts %}
    <div>
        <p>published: {{ post.fecha }}</p>
        <h1><a href="">{{ post.titulo }}</a></h1>
        <p>{{ post.autor }}</p>
    </div>
    {% endfor %}
    </body>
</html>
```

Esta sección de código hará posible visualizar todos nuestros posts

Template tags

```
{% for post in posts %}
 <div>
     <p>published: {{ post.fecha }}</p>
     <h1><a href="">{{ post.titulo }}</a></h1>
     <p>{{ post.autor }}</p>
 </div>
 {% endfor %}
```

1 2 3 4 5 6 7

Entendamos un poco como trabaja el código que acabamos de escribir.

La 1ra línea hace que podamos ir uno por uno en cada uno de los posts que tenemos registrados

La 2da línea abre la etiqueta ** **

La 3ra línea muestra dentro de la etiqueta ** ** las fecha del post

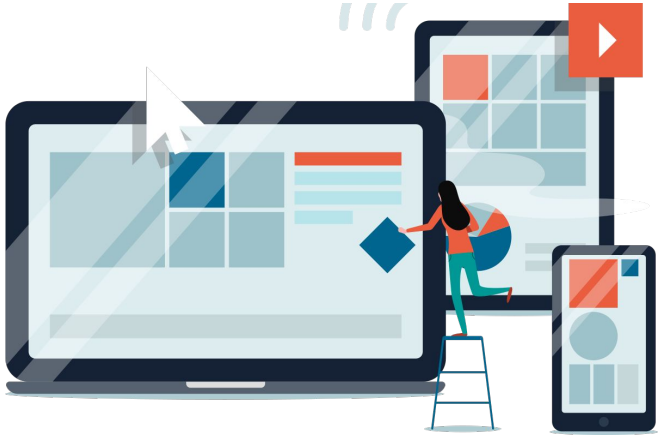
La 4ta línea nuestra dentro de la etiqueta **<h1></h1>** el título del post.

La 5ta línea nuestra dentro de la etiqueta ** ** el nombre del autor

La 6ta línea cierra nuestra etiqueta ** **

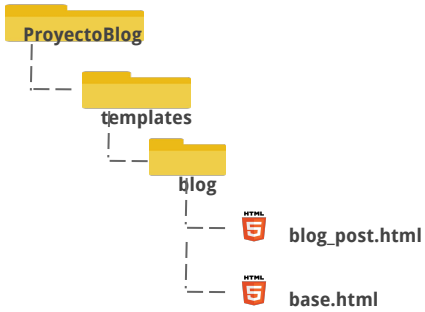
La 7ma línea cierra el **for** , termina la secuencia.

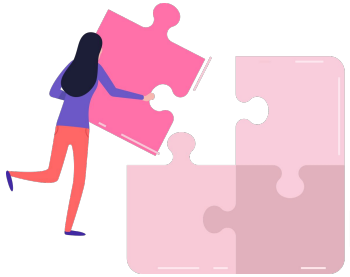
Herencia de plantillas



Django cuenta con superpoderes, tanto que nos permite reutilizar herramientas, una de ellas es la del código html para nuestro **blog**, veamos cómo utilizarlo

Template tags





Vamos a probar los superpoderes de **django,** dentro de nuestra carpeta templates/Blog vamos a crear una plantilla con el nombre **base.html**

base.html será nuestra template padre, y blog_post.html será nuestra plantilla hija, la cual podrá heredar características del padre.

Herencia de plantillas

Copiemos todo el contenido de nuestro archivo **blog_post.html** a nuestro archivo **base.html**

base.html

```
<html>
    <head>
        <title>Blog</title>
    </head>
    <body>

    <div>
        <h1><a href="">Bienvenido a mi Blog</a></h1>
    </div>

    {% for post in posts %}
    <div>
        <p>published: {{ post.fecha }}</p>
        <h1><a href="">{{ post.titulo }}</a></h1>
        <p>{{ post.autor }}</p>
    </div>
    {% endfor %}
    </body>
</html>
```

Borrar todas estas líneas

Herencia de plantillas

Quedando nuestro archivo de la siguiente manera

```
<html>
    <head>
        <title>Blog</title>
    </head>
    <body>

    <div>
        <h1><a href="">Bienvenido a mi Blog</a></h1>
    </div>

    {% block content %}
    {% endblock content %}
    </body>
</html>
```

Quedando solo estas dos lineas de código

Herencia de plantillas

Vamos a modificar nuestro archivo **blog_post.html,** haremos que herede de nuestra plantilla padre llamada **base.html**

blog_post.html

```
{% extends 'blog/base.html'%}

    {% block content %}

     {% for post in posts %}
         <div>
             <p>published: {{ post.fecha }}</p>
             <h1><a href="">{{ post.titulo }}</a></h1>
             <p>{{ post.autor }}</p>
         </div>
         {% endfor %}
    {% endblock content %}
```

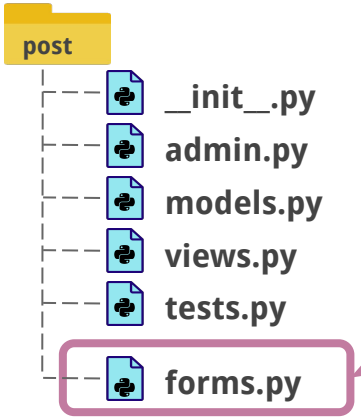
Con {% extends 'blog/post.html' %} le indicamos a esta plantilla que herede algunas etiquetas de la plantilla padre

FORMULARIOS

Formularios en Django

Vamos a crear una interfaz para poder agregar nuestros posts

Formularios en Django



Dentro de nuestra aplicación post tenemos que crear un archivo llamado **forms.py**

Formularios

Vamos a abrir este archivo y escribir el siguiente código

```
from django import forms                    1
from .models import Post                    2
class PostForm(forms.ModelForm):            3
    class Meta:                             4
        model = Post                        5
```

Entendamos un poco como trabaja el código que acabamos de escribir.

La 1ra línea importa la librería de django forms, esto nos permitirá crear formularios

La 2da línea importa el modelo de Post que se encuentra en el archivo models

La 3ra crea una clase con el nombre PostForm el cual contendrá nuestro formulario

La 4ta línea nos permite agregar nuestro modelo Post para usarlo como formulario.

La 5ta línea nos permite asignar los atributos del modelo post al formulario

Formularios

Enlace para nuestro formulario

```
<a href="{% url 'blog.views.post_new' %}" class="top-menu"><span
class="glyphicon glyphicon-plus" ></span> </a>
```

```
  {% extends 'blog/base.html'%}

   <a href="{% url 'blog.views.post_new' %}" class="top-menu"><span
class="glyphicon glyphicon-plus" ></span> </a>

     {% block content %}

      {% for post in posts %}
          <div>
              <p>published: {{ post.fecha }}</p>
              <h1><a href="">{{ post.titulo }}</a></h1>
              <p>{{ post.autor }}</p>
          </div>
          {% endfor %}
     {% endblock content %}
```

Enlace para

Nuestro formulario

Nuestra plantilla quedará de la siguiente manera

Formularios

```
from django.urls import include, path
from .views import listar_post
from . import views
urlpatterns = [
    path('index/', views.index, name='main-view'),
    path('^post/nuevo/$', views.post_nuevo, name='post-new')

]
```

1

2

Hemos agregado 2 nuevas líneas de código, vamos a ver el uso que le daremos:

1. Importamos todas las vistas de nuestro archivo views.py
2. Creamos una nueva url para nuestro formulario de posts.

Formularios

Creando la vista para el formulario

Vamos a abrir nuestro archivo views.py de la aplicación post y escribiremos el siguiente código

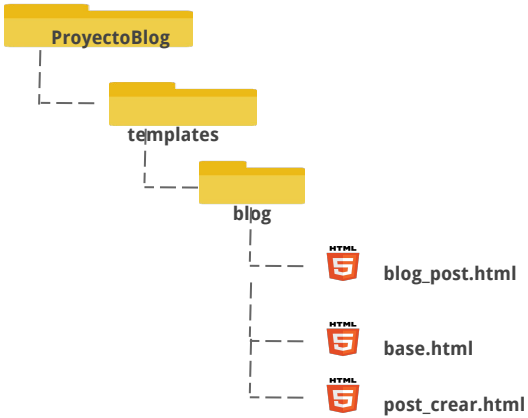
```
from .forms import PostForm
from .models import Publicacion
from django.views.generic.edit import CreateView

class CreatePostView(CreateView):

    form_class = PublicacionForm
    template_name = 'post_crear.html'
    model = Publicacion
    success_url = '/publicaciones'
```

Formularios

Vamos a crear nuestra plantilla con el nombre post_crear.html



Dentro de esta plantilla escribiremos código para poder crear nuestro formulario de posts

Herencia de plantillas

```
{% extends 'blog/base.html' %}

    {% block content %}

        <h1>New post</h1>
         <form method="POST" class="post-form" enctype="multipart/form-data">{%
csrf_token %}
            {{ form.as_p }}
            <button type="submit" class="save btn btn-default">Save</button>
        </form>
    {% endblock content %}
```

post_crear.html

Nuestra plantilla también heredará de la plantilla padre **base.html**

Tenemos que mostrar el formulario. Podemos hacerlo, por ejemplo, con un simple **{{ form.as_p }}.**

La línea anterior tiene que estar dentro de una etiqueta de formulario HTML:

<form method="POST">...</form>

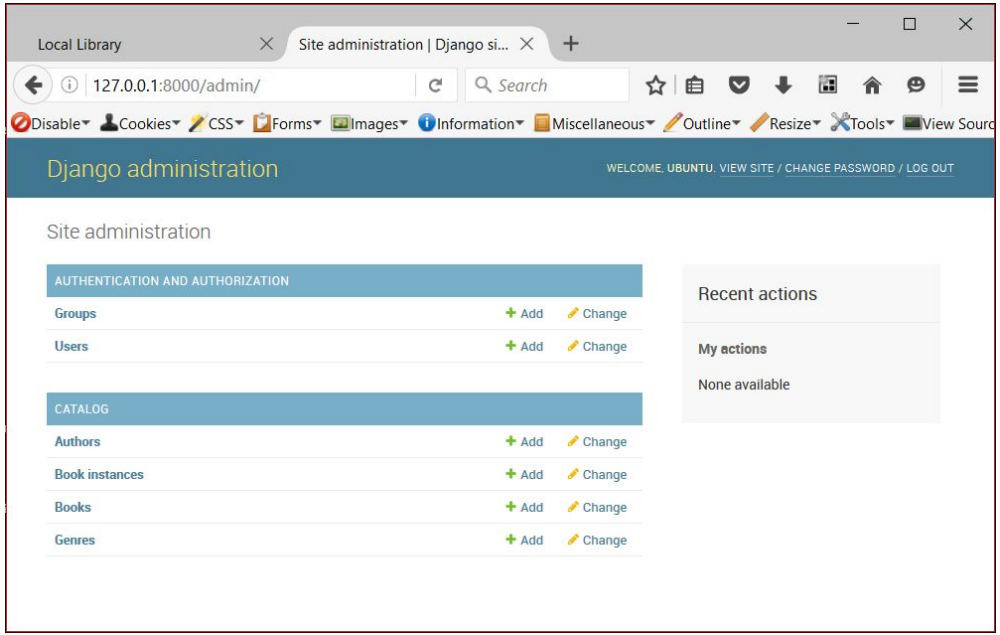
Necesitamos un botón Guardar. Lo hacemos con un botón HTML:

<button type='submit'>Save</button>

Finalmente justo después de la apertura de **<form...>** necesitamos agregar {% csrf_token %}. ¡Esto es muy importante ya que hace que tus formularios sean seguros! Django se quejará si te olvidas de esta parte cuando intentes guardar el formulario.

USUARIO Y ADMIN

Admin de Django



Django nos proporciona de un administrador desde donde podemos crear, editar y eliminar registros para poder acceder a este en nuestra consola debemos ingresar el siguiente comando.

```
(env)CursoDjango = python manage.py createsuperuser
```

Admin de Django

Al terminar de ejecutar el comando nos pedirá un usuario, debemos crear uno

Podemos colocar nuestro nombre, luego nos pedirá nuestro correo electronico y como último paso una contraseña.

```
Username (leave blank to use 'root'): rizzu
Email address: rizwan.iu11@gmail.com
Password:
Password (again):
Superuser created successfully.
```

CemTic

Inclusión de Mujeres en el Desarrollo e Innovación Tecnologica