Django

Jorge Barón Abad

¿Qué es Django?

Django es un framework de desarrollo para Python que se emplea para la creación de páginas web. Un framework web es un conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente.

Además se trata de una herramienta de código abierto y gratuita que cuenta con una comunidad amplia y que comparte recursos constantemente.

Instalación

Prerrequisitos:

- Python :
  - Instalar python: **sudo apt-get install python3.**
  - Comprobar si está instalado y la versión con: **python3 -V**
- Virtualenv (Para crear entornos virtuales con python)
  - Instalamos la librería **$ sudo apt-get install python3-venv (Si nuestra versión es 3.10)**
  - Creamos el directorio 2daw: **$ mkdir 2daw**
  - Nos movemos a ese directorio: **$ cd 2daw**
  - Creamos el entorno: **$ python3 -m venv myvenv**
  - Activamos nuestro entorno: **$ source myvenv/bin/activate**

(Este comando se ejecutará siempre que queramos entrar en el entorno)

Instalación

Instalar Django:

- Actualizamos gestor de paquetes pip a la última versión

  (myvenv) ~$ python -m pip install --upgrade pip

- Dentro de la carpeta 2daw, creamos el archivo "**requirements.txt**"
- En el archivo requirements.txt especificamos la versión de Django que queremos escribiendo en él:

  Django~=5.1.1

- Instalamos todos los paquetes necesarios para esa versión de Django, lanzando el comando en el directorio:

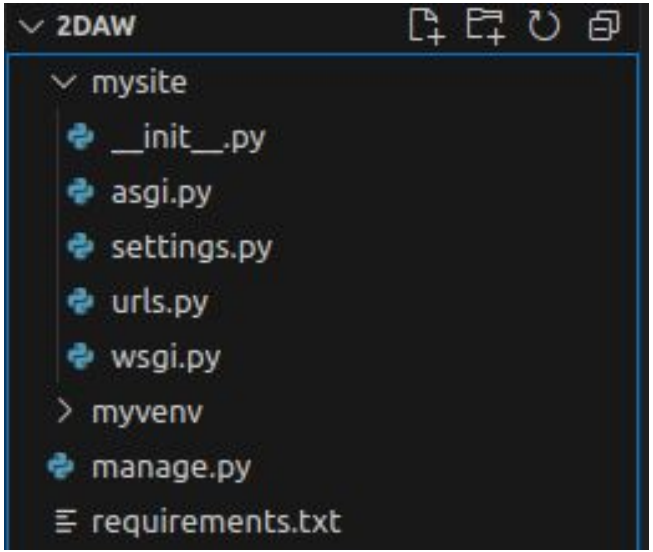
  pip install -r requirements.txt

Nuestro Primer Proyecto

Para empezar nuestro proyecto lanzamos el siguiente comando:

(myvenv) ~/2daw$ django-admin startproject mysite .

De esta forma creamos la estructura inicial de nuestro proyecto parecida a esta**:**



Configurar el proyecto

Para configurar el proyecto debemos editar el archivo **.settings.py:**

- TIME_ZONE = ‘Europe/Madrid’ -> Configuramos la hora y fecha del proyecto
- LANGUAGE_CODE = ‘es’ -> Configuramos el lenguaje del proyecto
- Estáticos! -> Configuramos la ruta de los archivos estáticos(CSS por ejemplo)
  - STATIC_URL = '/static/'
  - STATIC_ROOT = BASE_DIR / 'static'
- ALLOWED_HOSTS = ['127.0.0.1', '.pythonanywhere.com','0.0.0.0'] -> direcciones ips que necesitamos para que funcione el proyecto

Configurar el proyecto: Base de Datos

Nos aseguramos que en el archivo **settings.py** comprobamos que la configuración de la base de datos esta de la siguiente forma:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Creamos nuestra base de datos con el comando:

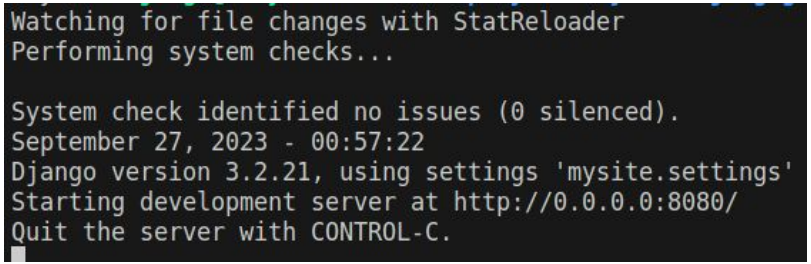
```
(myvenv) ~/2daw$ python manage.py migrate
```

Arrancar Servidor Web

Para arrancar el servidor Web lanzamos el siguiente comando:

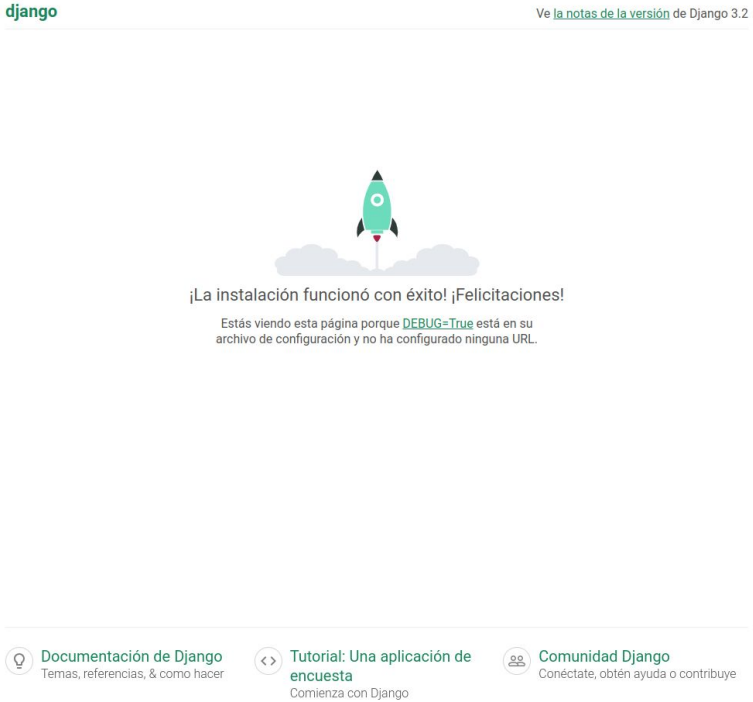
```
(myvenv) ~/2daw$ python manage.py runserver 0.0.0.0:8080
```

Debería aparecer lo siguiente:



Para consultar debemos acceder a la siguiente URL: http://0.0.0.0:8080/

Arrancar Servidor Web



Pasos en Casa desde un proyecto ya creado por git

- Descargar proyecto con GIT
- **sudo apt-get install python3-venv -> Sino está instalado ya**
- No situamos en la carpeta **2daw**
- **python3 -m venv myvenv -> C**reamos el entorno
- **source myvenv/bin/activate**
- **python -m pip install --upgrade pip**
- **pip install -r requirements.txt**
- **python manage.py migrate -> Creamos base de datos**
- **python manage.py runserver 0.0.0.0:8080 -> Lanzamos el servidor**

Pasos para un proyecto desde Cero

- Crear un repositorio en github sólo con .README
- Clonarlo en una carpeta de vuestro ordenador
- Situaros en la carpeta del ordenador
- **sudo apt-get install python3-venv -> Sino está instalado ya**
- Creamos la carpeta de nuestro proyecto -> **mkdir carpeta_proyecto**
- Nos situamos en la carpeta -> **cd carpeta_proyecto**
- Creamos el archivo **.gitignore** correspondiente a proyectos Django
- Creamos el archivo **requirements.txt** con la versión de Django
- **python3 -m venv myvenv ->** Creamos el entorno
- **source myvenv/bin/activate**
- **python -m pip install --upgrade pip**
- **pip install -r requirements.txt**
- Creamos la configuración de Django: **django-admin startproject mysite .**
- Realizamos los cambios correspondientes en **settings.py**
    - **python manage.py migrate** -> Hacemos la migración
- **python manage.py startapp <nombre_app>**
- Y probamos que funciona: **python manage.py runserver 127.0.0.1:8080**

Creando Aplicación de prueba

Ahora vamos a crear una aplicación de prueba en nuestro entorno de prueba.

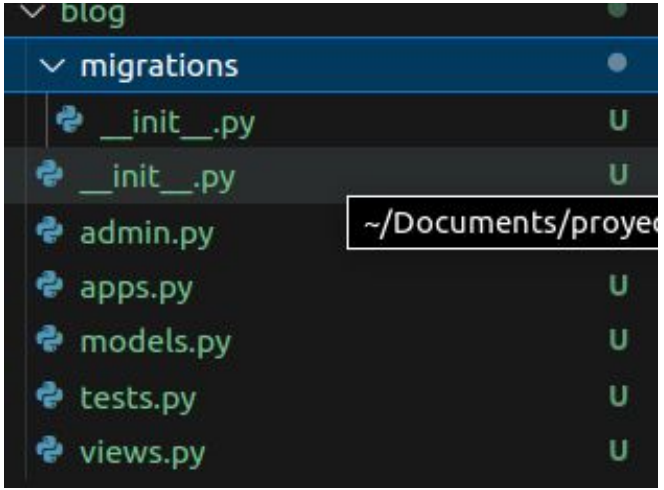
Esta aplicación consistirá en un blog, donde los usuarios pueden añadir articulos.

Creamos aplicación

Lanzamos el siguiente comando para crear la aplicación blog:

(myvenv) ~/2daw$ python manage.py startapp blog

Y debería crearse en nuestro proyecto la siguiente estructura:



Registramos aplicación

Ahora debemos indicarle a Django que vamos a usar esa aplicación. En el archivo **settings.py** en el apartado de **INSTALLED_APPS** debemos añadir la aplicación **blog:**

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

Primeros Pasos de la Aplicación

Una vez creada la aplicación y registrada, el primer paso que debemos seguir es definir las clases y tablas de la base de datos que vamos a usar en la aplicación.

Por ejemplo, si vamos a crear una aplicación que es un Blog, dónde añadiremos artículos(Post), crearemos una clase Post y en la base de datos deberá haber una tabla donde se guarden los artículos y sus datos: título del artículo, creador, fecha de creación, fecha de publicación, contenido,etc..

Modelos (Models)

En Django (y gran mayoría de frameworks) estas clases, se van a llamar **Modelos (Models)** , porque tiene una característica especial, estas clases están asociadas a una tabla de una base de datos.

En un resumen sencillo, por ejemplo si tenemos la clase Persona en python:

```
class Persona:

    nombre = ""
    edad = 0

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

Dicha clase se convertirá en una tabla en la base de datos, que se llama Persona con las columnas nombre y edad, donde guardaremos las personas que acceden a la aplicación.

Nombre	Edad
Pepito	45
Juanito	30

Creamos nuestro modelo

Vamos a crear nuestro primer Modelo. Lo primero es abrir el archivo **blog/models.py** , en este archivo se añadiran todos los modelos que necesitamos. Por ejemplo el modelo de **Post**

```
from django.conf import settings
from django.db import models
from django.utils import timezone


class Post(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)
    published_date = models.DateTimeField(blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

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

Creamos nuestro modelo

Explicamos nuestro modelo:

- **class Post(models.Model): ->** Siempre deben heredar de la clase padre **Model** para indicar que es un modelo
- **models.ForeignKey** -> Indicamos que es una clave foránea que hace referencia a otro modelo
- **models.CharField(max_length=200)** -> Indicamos que es de tipo char con un máximo de 200 caracteres
- **models.TextField()** -> Indicamos que es de tipo Texto
- **models.DateTimeField(default=timezone.now)** -> Indicamos que es de tipo DateTime

En python, no es necesario indicar el tipo de los atributos, por lo tanto ¿por qué lo definimos de esta forma tan peculiar?

Porque esta clase (Modelo) representa una tabla en la base de datos, y en la base de datos, cada atributo representa una columna en dicha tabla, y en dicha tabla si debemos especificar el tipo de cada columna.

Creamos nuestras migraciones

Bueno, ahora que hemos creado nuestro modelo.¿Qué hacemos?

Pues debemos ejecutar un comando para crear un archivo con la sentencias que deben ejecutarse en la base de datos asociada a nuestro modelo. Este archivo se llama **migration** y debemos crearlo con el comando **makemigrations:**

```
(myvenv) ~/2daw$ python manage.py makemigrations blog
```

Y debería aparecer lo siguiente:

```
Migrations for 'blog':
  blog/migrations/0001_initial.py
    - Create model Post
```

Creamos nuestras migraciones

En el proyecto debemos tener ahora una carpeta que se llama **migrations,** y en ella con un archivo **0001_initial.py** con este códgio:

```
class Migration(migrations.Migration):
initial = True
dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
]

operations = [
        migrations.CreateModel(
                name='Post',
                fields=[
                        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                        ('title', models.CharField(max_length=200)),
                        ('text', models.TextField()),
                        ('created_date', models.DateTimeField(default=django.utils.timezone.now)),
                        ('published_date', models.DateTimeField(blank=True, null=True)),
                        ('author', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL)),
                ],
        ),
]
```

Ejecutamos nuestras migraciones

Ahora que tenemos las operaciones que crean nuestra tabla en la base de datos, ejecutamos el comando correspondiente para ejecutar las **migraciones** y crear nuestra tabla:

```
(myvenv) ~/2daw$ python manage.py migrate blog
```

Debería aparecer lo siguiente:

```
Operations to perform:
  Apply all migrations: blog
Running migrations:
    Applying blog.0001_initial… OK
```

Administración Django

Ahora que hemos creado la tabla en la base de datos y todo ha ido perfecto, ¿Cómo gestiono esta tabla en mi base de datos?

Para ello debemos indicarle a Django que vamos a administrar dicho modelo y acceder a nuestro panel de administración de la aplicación.

Por lo tanto accedemos al archivo **admin.py** y añadimos lo siguiente:

```
from .models import Post

admin.site.register(Post)
```

Administración Django

Ahora ejecutamos el servidor de Django y accedemos al panel de administración:

```
python manage.py runserver
```

Y accedemos a la url: http://127.0.0.1:8000/admin/.

Y debería aparecer la siguiente página web.

Pero….. ¿Cuál es el usuario y la contraseña?

Pues todavía no lo hemos creado. Para ello, lanzamos el comando y lo creamos:

```
(myvenv) ~/2daw$ python manage.py createsuperuser
```

```
Nombre de usuario (leave blank to use 'jorge'): jorge
Dirección de correo electrónico: jorge.baron@iespoligonosur.org
Password:
Password (again):
Superuser created successfully.
```

Administración Django

Si os sale el siguiente error

```
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, ses
sions.
Run 'python manage.py migrate' to apply them.
```

Debéis lanzar el siguiente comando:

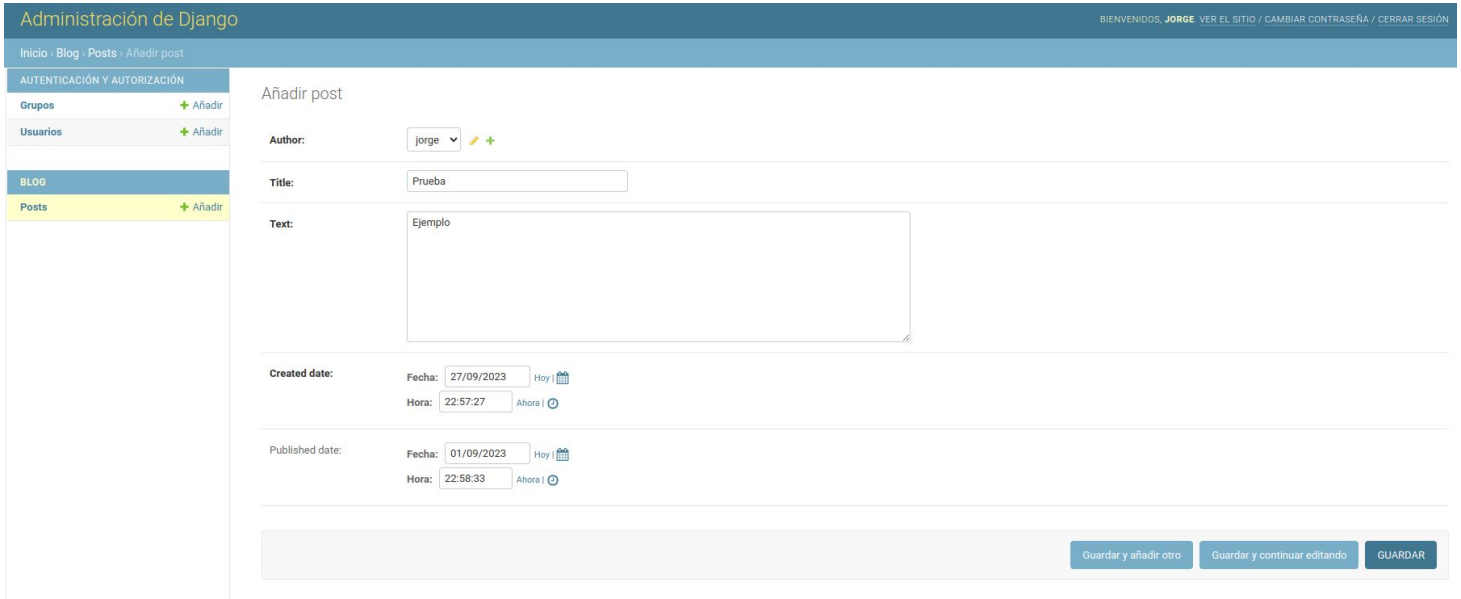
python manage.py migrate

Administración Django

Volvemos a ejecutar el servidor de Django si lo hemos parado anteriormente y accedemos de nuevo a la url http://127.0.0.1:8000/admin/ y ahora introducimos el usuario y la contraseña que hemos creado y debería aparecer lo siguiente

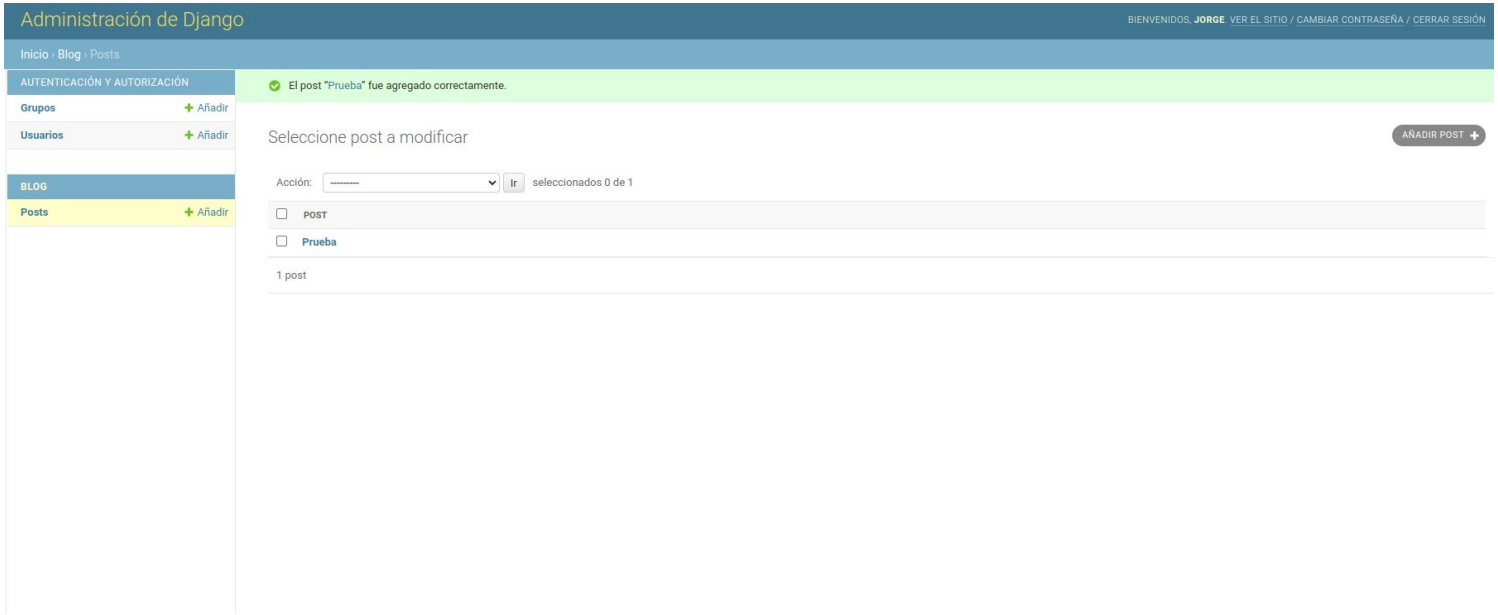
Administración Django

Pulsamos en el botón **añadir** que aparece en la misma línea que **Post.** Nos aparecerá un formulario para crear Post, rellenamos y le damos a guardar:



Administración Django

Una vez creado nos debería aparecer un listado con los Post creados. En este caso sólo 1.



Django Urls

Ok. Ya hemos creado nuestro primer modelo, hemos accedido al panel de Admin para acceder a los Post en la base de Datos.¿Qué nos toca ahora?

Puedes debemos mostrar dichos Posts en nuestra aplicación Web.

Ahora mismo nuestra aplicación sólo tiene una página de Inicio, y además es una página por defecto cuando accedemos a la URL: http://127.0.0.1:8000/ .

También tenemos la URL del panel de administración: http://127.0.0.1:8000/admin/

Nuestro objetivo ahora es crear más URLs para acceder a las distintas páginas de nuestra Web.

Django URLs

Para editar las URLs, debemos abrir el archivo **mysite/urls.py** que tiene el siguiente código**:**

```
from django.contrib import admin
from django.urls import path

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

Ahora mismo lo único que tenemos son las urls de la aplicación de admin.Cuando incluimos la palabra admin en la url, se utilizan las urls de la aplicación de admin.

Django URLs

Ahora debemos indicar que en nuestra aplicación blog también van a existir URLs. Modificamos el archivo de **mysite/urls.py** para que se parezca al siguiente:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
]
```

Lo que hemos hecho es indicar que la URL por defecto de la página Web use las URLs de la aplicación Blog.(Que todavía tenemos que crear)

Django urls

Ahora toca crear las urls en nuestra aplicación Blog:

- Creamos el archivo **urls.py** en dentro de la carpeta blog.
- Creamos el siguiente código

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
]
```

De esta forma indicamos que la ruta por defecto muestre la vista: **post_list**. ¿Pero qué va a pasar cuando se acceda a http://127.0.0.1:8000/ ?

Django urls

Resulta que ahora no puedo acceder a la página principal.

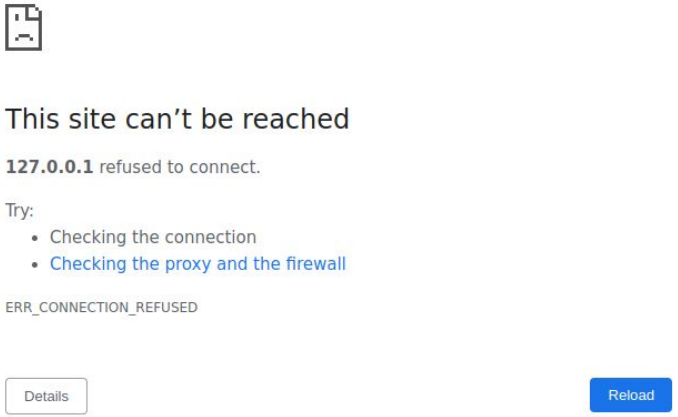
Si consulto la consola, nos aparecerá el siguiente error:

```
    path('', views.post_list, name='post_list'),
AttributeError: module 'blog.views' has no attribute 'post_list'
```

¿Qué significa esto?

Qué todavía no existe el módulo de vistas de nuestra aplicación.

Por lo tanto ahora toca crear las vistas



Django Views

Las vistas es dónde vamos a incluir el código que queremos que se ejecute y que indicará la página HTML que queremos que se muestre al cliente

Dentro de la aplicación Blog, debemos encontrar un archivo **views.py.** Lo abrimos e incluimos el siguiente código:

```
from django.shortcuts import render

# Create your views here.
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Hemos creado una función que tiene el mismo nombre, que el “name” que especificamos en la URL, que devuelve una página .html.

Ahora volvemos a acceder a la URL principal http://127.0.0.1:8000/ ¿Qué pasara ahora?

Django Views

Volvemos a tener un Error, pero un poco distinto:

TemplateDoesNotExist at /

blog/post_list.html

Request Method:	GET
Request URL:	http://127.0.0.1:8000/
Django Version:	3.2.21
Exception Type:	TemplateDoesNotExist
Exception Value:	blog/post_list.html
Exception Location:	/home/jorge/Documents/proyectosPython/djangogirls/myvenv/lib/python3.10/site-packages/django/template/loader.py, line 19, in get_template
Python Executable:	/home/jorge/Documents/proyectosPython/djangogirls/myvenv/bin/python
Python Version:	3.10.12
Python Path:	['/home/jorge/Documents/proyectosPython/djangogirls', '/usr/lib/python310.zip', '/usr/lib/python3.10', '/usr/lib/python3.10/lib-dynload', '/home/jorge/Documents/proyectosPython/djangogirls/myvenv/lib/python3.10/site-packages']
Server time:	Thu, 28 Sep 2023 00:04:36 +0200

Este error se debe a que no tenemos el archivo HTML que vamos a mostrar. En Django a ese archivo se le llama **Template**

Django Templates

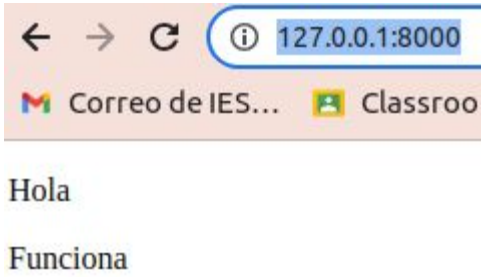
Las Templates de Django son archivos con la extensión .html , donde tendremos código HTML y otro código especial de Django para mostrar más información.

Vamos a paso a paso:

- Creamos la carpeta **templates** dentro de Blog
- Creamos la carpeta blog, dentro de templates
- Creamos el archivo **post_list.html**
- Incluimos el siguiente código en el archivo .html

Django Templates

Ahora al consultar la URL http://127.0.0.1:8000/ debería aparecer ya:



Ya hemos conseguido mostrar nuestro contenido, introducida una url, pero ¿Dónde están nuestros Posts?¿Cómo lo incluimos?

Ahora es cuando entra en juego la comunicación entre las vistas y los templates.

La vista le va a pasar la información de los Posts al Template y el Template se va a encargar de mostrarlo.

Pero……. ¿Cómo obtengo los Post en la vista?¿Cómo los saco de la Base De Datos?

Ahora presentamos dos tecnologías : **Django ORM y QuerySets**

Django ORM y QuerySets

Las Django ORM(Mapeo Objeto-Relacional): Es la tecnología que nos ha permitido crear una tabla en la base de datos a partir de una clase, y que nos permitirá explotar dichos datos.

Django QuerySets: Son los conjuntos de datos que vamos a obtener de la base de datos, ya transformados en los objetos de la clase correspondiente.

Por ejemplo si quisiera obtener todos los posts que hay en la base de datos, crearía la siguiente instrucción:

```
Post.objects.all()
```

Django View-QuerySet

Ahora que sabemos cómo obtener los Posts desde la base de datos, vamos a usarlo en las vistas, para poder pasar esa información a las templates y se puedan mostrar.

Volvemos a abrir el archivo **views.py** y lo modificamos de la siguiente forma:

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

# Create your views here.
def post_list(request):
    posts = Post.objects.all()
    return render(request, 'blog/post_list.html', {'posts_mostrar': posts})
```

¿Que hemos hecho?:

- En la variable posts, hemos guardado todos los Posts de la base de datos
- Le pasamos a la Template blog/post_list.html los Posts con el nombre ‘posts_mostrar’

Django View-Template

¿Cómo mostramos la información de los Posts en nuestra plantilla?

Modificamos nuestro archivo .html de la siguiente forma.

Hemos incluido en esta Template lo que son las **Templates Tags**

```
<!DOCTYPE html>
<html>
    <head>
        <title>Mi Blog</title>
    </head>
    <body>
        <header>
            <h1><a href="/">Django Girls Blog</a></h1>
        </header>

        {% for post in posts_mostrar %}
            <article>
                <time>published: {{ post.published_date }}</time>
                <h2><a href="">{{ post.title }}</a></h2>
                <p>{{ post.text|linebreaksbr }}</p>
            </article>
        {% endfor %}
    </body>
</html>
```

Django Template-Tags

Las **Templates Tags** son etiquetas, que pueden mezclarse con el HTML, y nos permite mostrar la información que deseamos de la vista:

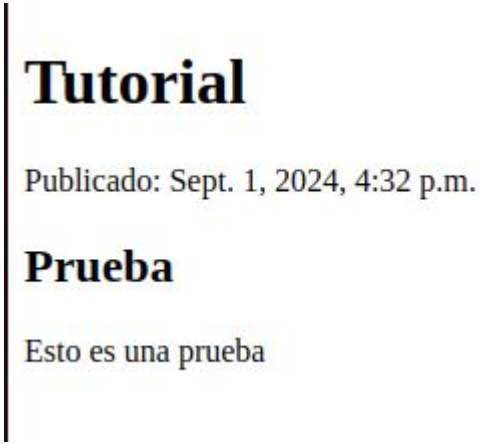
`{% for post in posts_mostrar %}` -> Hacemos un for, para recorrer todos los posts que provienen de la vista

`{{ post.published_date }}` -> Mostramos información de dicho atributo del objeto Post.

`{% endfor %}` -> Cerramos el bucle

Django Template-Tags

Si ahora accedemos a nuestra aplicación, debería aparecer información de nuestros Posts.



Django estáticos: CSS, Imágenes y JS

En una aplicación web hay una serie de archivos que siempre serán iguales, por lo tanto es necesarios que se mantengan estáticos, para que el navegador los pueda cachear y la página funcione mejor.

Esto son los casos de los archivos de imágenes, CSS y JS.

Ahora mismo nuestra aplicación no es especialmente bonita, para ello, debemos incluir los archivos css necesarios en nuestras plantillas para que resulte más bonita. Por ejemplo incluimos Bootstrap 5 de esta forma. Aunque más adelante veremos cómo hacerlo mejor.

```
<title>Tutorial</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
```

Django Formularios

También debemos incluir formularios en nuestros proyectos, que nos permitirán al cliente comunicarse con la aplicación, para enviar datos.

Por ejemplo ahora mismo, creamos los Posts desde el panel de Administración, pero lo razonable sería incluir un formulario en el proyecto, para que los usuarios que accedan puedan crear Posts.

RESUMEN WORKFLOW

¿Qué es MVT?

Es un patrón de diseño de software para desarrollar una aplicación web.

- **Modelo:** Actuará como la interfaz de sus datos. Es responsable de mantener los datos.
- **Vista:** Controla lo que verá el usuario, responde a una petición de URL y orquesta la respuesta. Se encarga de la lógica de negocio y de llamar al procesamiento de los templates.
- **Plantilla:** Consta de partes estáticas de la salida HTML deseada, y también tiene contenido dinámico

Aportación de: Diego Reyes Rosales

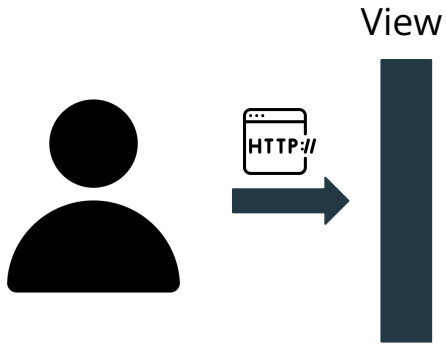
¿Cómo funciona?

El usuario hace la petición HTTP a nuestro nuestra dirección

Aportación de: Diego Reyes Rosales

¿Cómo funciona?

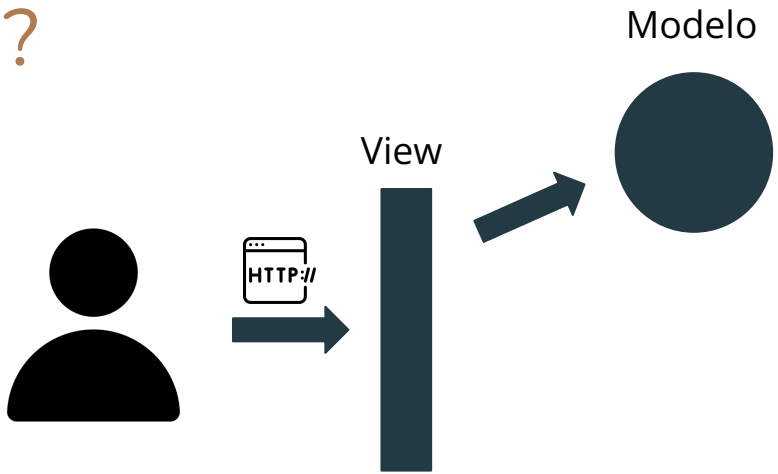
Lo recibe el VIEW (Capa de vista)



Aportación de: Diego Reyes Rosales

¿Cómo funciona?

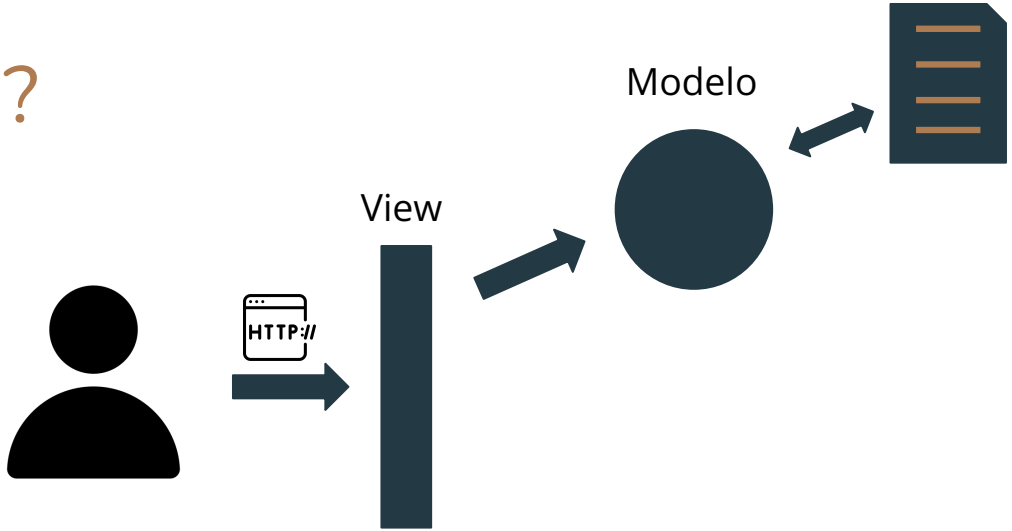
El view le pregunta a Modelo.



Aportación de: Diego Reyes Rosales

Fichero posts con todos los post de la base de datos

¿Cómo funciona?



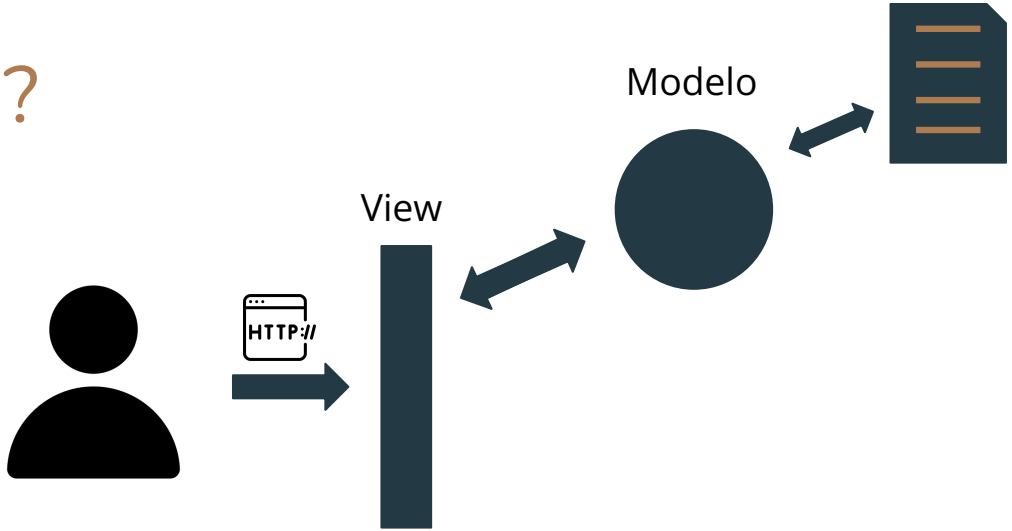
El view le pregunta a Modelo.

Y hace un archivo **posts** con todos los posts ordenados de la base de datos.

Aportación de: Diego Reyes Rosales

Fichero posts con todos los post de la base de datos

¿Cómo funciona?



El view le pregunta a Modelo.

Y hace un archivo **posts** con todos los posts ordenados de la base de datos.

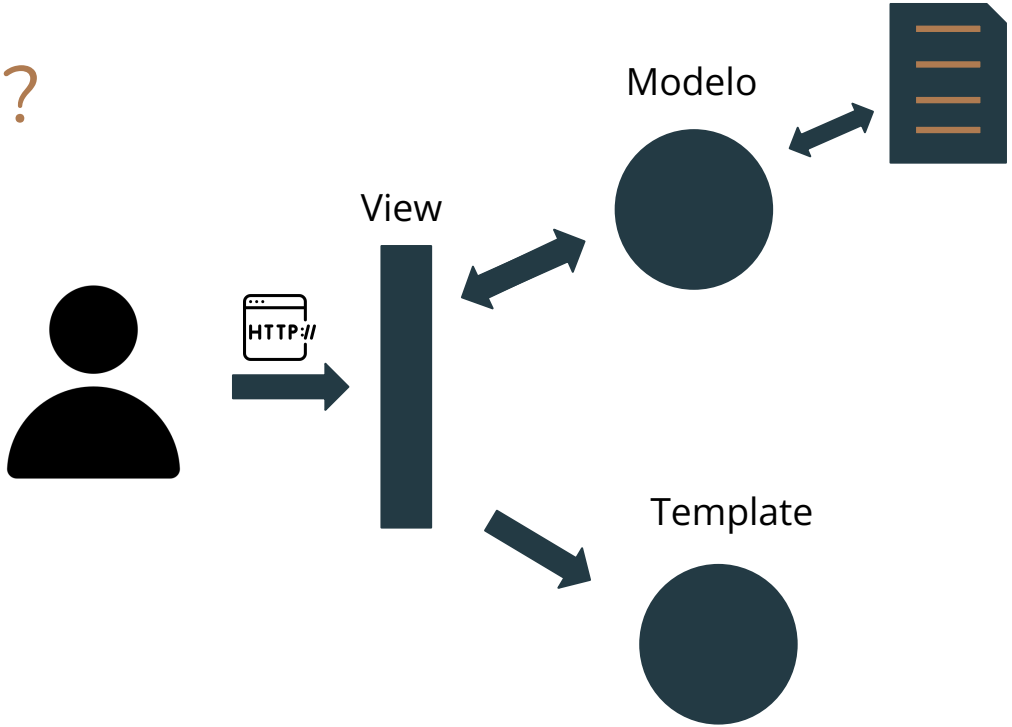
Y se lo devuelve a View

Aportación de: Diego Reyes Rosales

Fichero posts con todos los post de la base de datos

¿Cómo funciona?

Ahora View le pregunta a Templates por el HTML, y estáticos (css)

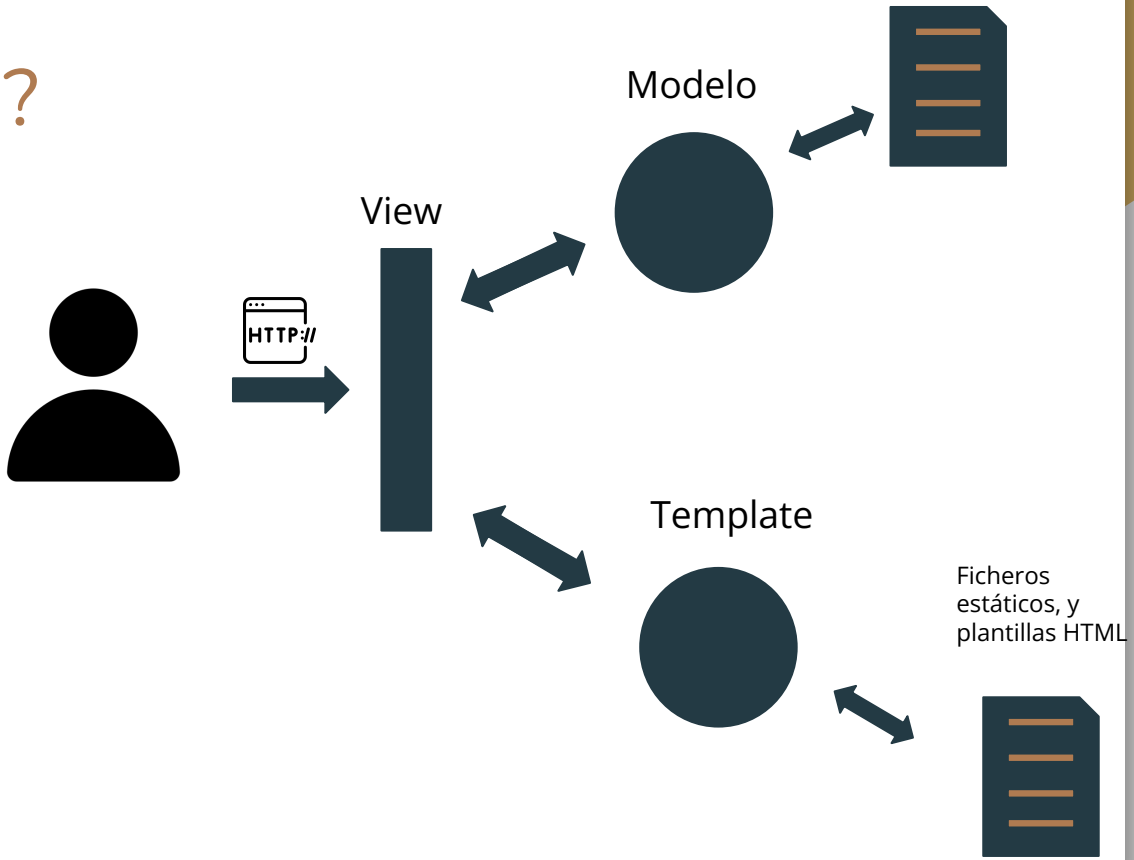


Aportación de: Diego Reyes Rosales

¿Cómo funciona?

Ahora View le pregunta a Templates por el HTML, y estáticos (css)

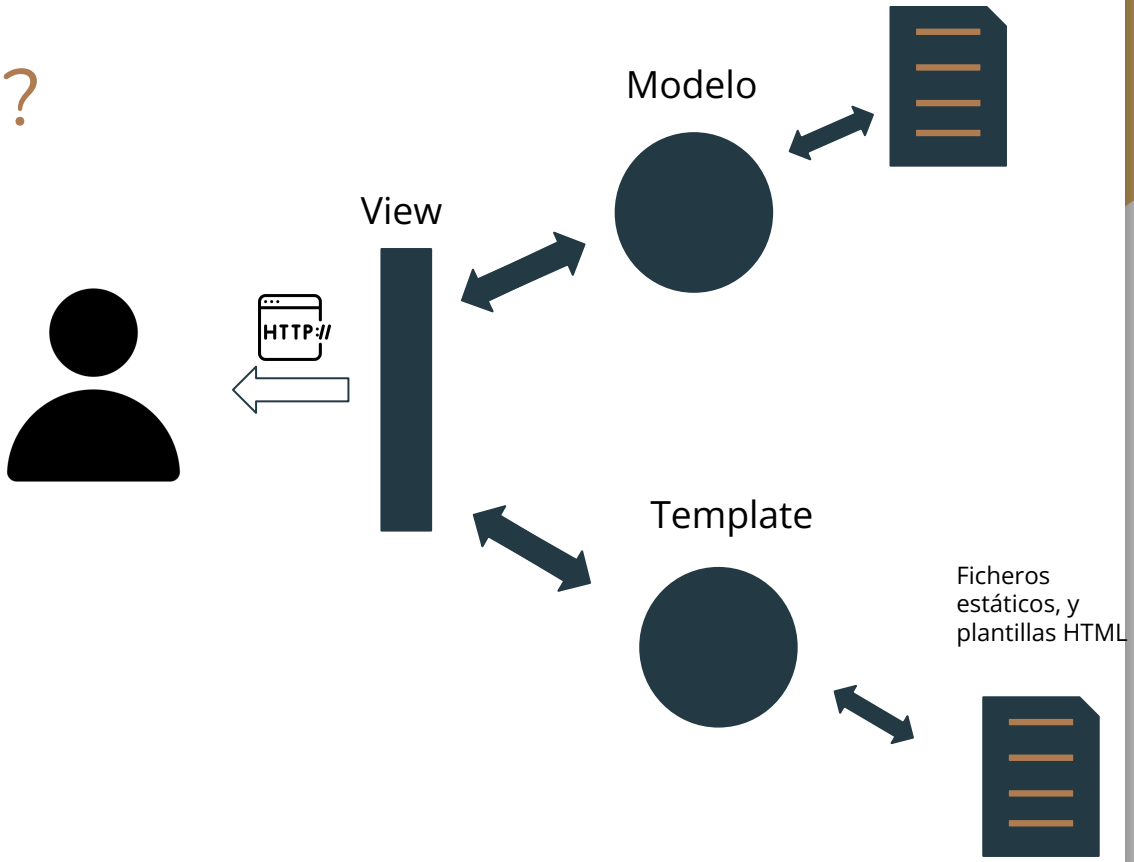
Templates le da todas las plantillas y css a View.



Aportación de: Diego Reyes Rosales

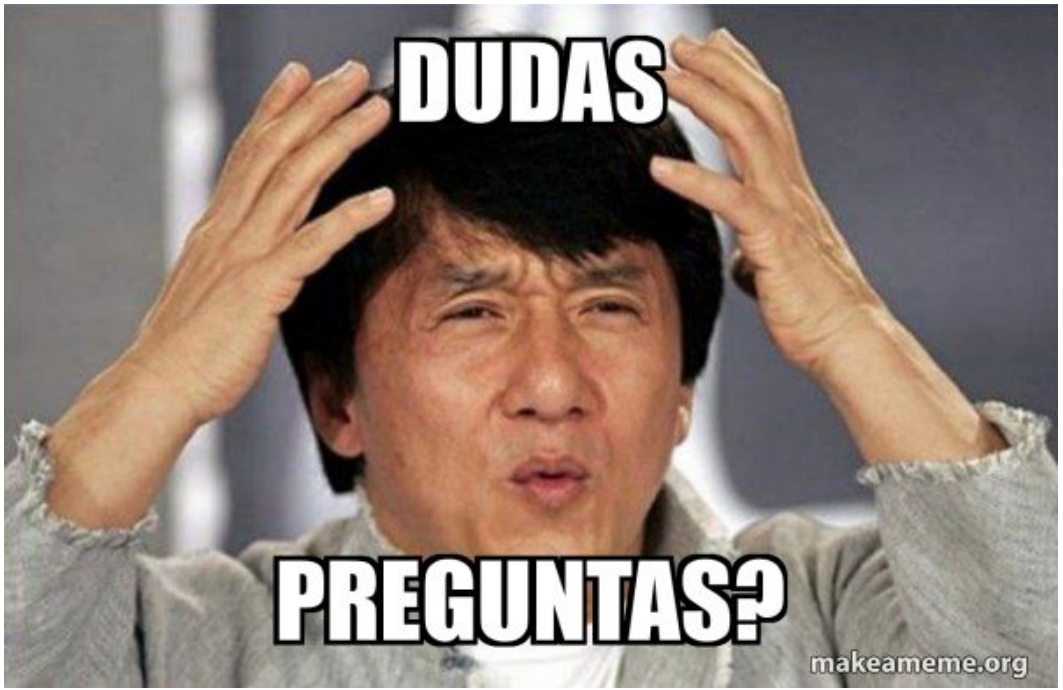
¿Cómo funciona?

Finalmente se le muestra al usuario la página con la plantilla html y css montado con los posts.



Aportación de: Diego Reyes Rosales

¿DUDAS?



Agradecimientos

Eduardo Serrano Luque