

# Informe de Laboratorio 05

## Tema: Django

Nota

Estudiante	Escuela	Asignatura
Ower Frank Lopez Arela olopez@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 20222083

Laboratorio	Tema	Duración
05	Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 30 Mayo 2023	Al 14 Junio 2023

### 1. Tarea

- Crea un blog sencillo en un entorno virtual utilizando la guía: [https://tutorial.djangogirls.org/es/django\\_start\\_project/](https://tutorial.djangogirls.org/es/django_start_project/)
- Especificar paso a paso la creación del blog en su informe.



### 2. URL de Repositorio Github

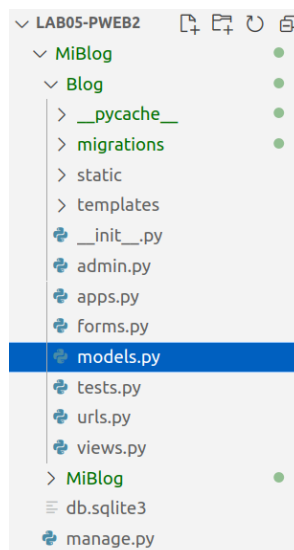
- URL para el laboratorio 05 en el Repositorio GitHub.
- <https://github.com/OwerFrankLopezArela/LAB05-PWEB2.git>

## 3. Ejercicios

### 3.1. Estructura de laboratorio 05

- El contenido que se entrega en este laboratorio es el siguiente:

```
+----LAB05-PWEB2
+----MiBlog
|----Blog
|----admin.py
|----apps.py
|----forms.py
|----__init__.py
|----migrations
|----models.py
|----__pycache__
|----static
|----style.css
|----templates
|----editPost.html
|----index.html
|----newPost.html
|----post.html
|----tests.py
|----urls.py
+----views.py
|----db.sqlite3
|----manage.py
+----MiBlog
|----asgi.py
|----__init__.py
|----__pycache__
|----settings.py
|----urls.py
+----wsgi.py
```



### 3.2. Crea un blog sencillo en un entorno virtual:

- Especificar paso a paso la creación del blog en su informe..
- Crear un video tutorial donde realice las operaciones CRUD (URL public reproducible online.
- Adjuntar URL del video en el informe.

#### Listing 1: CREACION DEL PROYECTO

El primer paso es iniciar un nuevo proyecto de Django. Básicamente, significa que vamos a lanzar unos scripts proporcionados por Django que nos crearán el esqueleto de un proyecto de Django. Son solo un montón de directorios y archivos que usaremos más tarde.

Los nombres de algunos archivos y directorios son muy importantes para Django. No deberías renombrar los archivos que estamos a punto de crear. Moverlos a un lugar diferente tampoco es buena idea. Django necesita mantener una cierta estructura para poder encontrar cosas importantes.

```
(myvenv) C:\Users\Name\djangogirls> django-admin.exe startproject mysite .
```

#### Listing 2: CAMBIAR LA CONFIGURACION

En este proceso, se realizaron varios cambios en el archivo `mysite/settings.py`. Primero, se modificó la configuración de la zona horaria (`TIME_ZONE`) para reflejar la zona horaria deseada. Luego, se cambió el código de idioma (`LANGUAGE_CODE`) para ajustar el idioma de los botones y notificaciones de Django.

Además, se agregó una ruta para archivos estáticos (`STATIC_ROOT`) al final del archivo. También se actualizó la configuración de `ALLOWED_HOSTS` para incluir las direcciones IP locales y los nombres de host correspondientes a PythonAnywhere y Cloud9 si se está utilizando. Se proporcionó una nota adicional para usuarios de Chromebook.

```
> LANGUAGE_CODE = 'es-es'

> STATIC_URL = '/static/'
> STATIC_ROOT = BASE_DIR / 'static'

> ALLOWED_HOSTS = ['127.0.0.1', '.pythonanywhere.com']
```

#### Listing 3: CONFIGURANDO LA BASE DE DATOS

En este proceso, se configura la base de datos predeterminada (`sqlite3`) en el archivo `mysite/settings.py`. Se utiliza la configuración por defecto que viene con Django. Luego, se ejecuta el comando `python manage.py migrate` en la consola, estando en el directorio que contiene el archivo `manage.py`. Este comando aplica todas las migraciones necesarias para la base de datos, creando las tablas correspondientes. Si todo va bien, se muestra en la consola una serie de mensajes que indican la ejecución exitosa de las migraciones. Después de este paso, se puede iniciar el servidor web para verificar el funcionamiento del sitio web.

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

```
}  
}
```

---

#### Listing 4: INICIAR EL SERVIDOR

---

Para iniciar el servidor web, primero me aseguro de estar en el directorio que contiene el archivo "manage.py".<sup>en</sup> la carpeta "djangogirls". En mi consola, ejecuto el comando "python manage.py runserver". En mi caso, estoy utilizando un Chromebook, así que utilizo el comando "python manage.py runserver 0.0.0.0:8080". Si estuviera en Windows y me encontrara con un error UnicodeDecodeError, usaría en su lugar el comando "python manage.py runserver 0:8000". Luego, abro mi navegador preferido (Firefox, Chrome, Safari, Internet Explorer, o cualquier otro) y escribo la dirección "http://127.0.0.1:8000/" para verificar que mi sitio web se está ejecutando correctamente.

```
> (myenv) ~/djangogirls$ python manage.py runserver  
> http://127.0.0.1:8000/
```

---

#### Listing 5: CREAMOS UNA APLICACION

---

Para mantener todo en orden, crearemos una aplicación separada dentro de nuestro proyecto. Es muy bueno tener todo organizado desde el principio. Para crear una aplicación, necesitamos ejecutar el siguiente comando en la consola (dentro de la carpeta de djangogirls donde está el archivo manage.py):

```
(myenv) C:\Users\Name\djangogirls> python manage.py startapp blog
```

Y aquí notamos que se ha creado un nuevo directorio blog y ahora contiene una cantidad de archivos. Los directorios y archivos en nuestro proyecto deberían verse así:

```
djangogirls  
  blog  
    __init__.py  
    admin.py  
    apps.py  
    migrations  
      __init__.py  
    models.py  
    tests.py  
    views.py  
  db.sqlite3  
  manage.py  
  mysite  
    __init__.py  
    settings.py  
    urls.py  
    wsgi.py  
  requirements.txt
```

Después de crear una aplicación, también necesitamos decirle a Django que debe utilizarla. Eso se hace en el fichero mysite/settings.py – ábrelo en el editor. Tenemos que encontrar INSTALLED\_APPS y agregar una línea que contiene 'blog.apps.BlogConfig', justo por encima de ]. El producto final debe tener este aspecto:

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

---

### Listing 6: CREAMOS EL MODELO DEL POST

---

En primer lugar, importo las configuraciones de Django necesarias para el proyecto. Luego, importo los modelos y utilidades de Django que voy a utilizar en este archivo. Dentro de este archivo, defino una clase llamada "Post" que hereda de la clase "models.Model" de Django. Esta clase representa un modelo de datos para un post en un blog u otra aplicación similar. Tiene los siguientes atributos:

- "title.es un campo de tipo CharField que almacena el título del post, con una longitud máxima de 200 caracteres.
- "text.es un campo de tipo TextField que almacena el contenido del post.
- "created\_date.es un campo de tipo DateTimeField que guarda la fecha y hora de creación del post. Por defecto, se establece en el valor actual de la zona horaria.
- "published\_date.es un campo de tipo DateTimeField que guarda la fecha y hora de publicación del post. También se establece en el valor actual de la zona horaria de forma predeterminada.
- "publish.es un método que actualiza el campo "published\_date" con la fecha y hora actuales y guarda el objeto en la base de datos.
- "str.es un método especial que devuelve una representación en cadena del objeto. En este caso, devuelve el título del post.

```
from django.conf import settings  
from django.db import models  
from django.utils import timezone  
  
# Create your models here.  
  
class Post(models.Model):  
    title = models.CharField(max_length=200)  
    title = models.CharField(max_length=200)  
    text = models.TextField()  
    created_date = models.DateTimeField(  
        default=timezone.now)  
    published_date = models.DateTimeField(  
        default=timezone.now)  
  
    def publish(self):  
        self.published_date = timezone.now()  
        self.save()  
  
    def __str__(self):  
        return self.title
```

---

### Listing 7: CREACION DEL FORMULARIO

---

Seguidamente, definimos una clase llamada `CreateNewPost` que representa un formulario en Django. El formulario se crea a partir del modelo `Post` importado desde otro archivo. El formulario tiene dos campos: `title` y `text`, que corresponden a los campos del modelo `Post`. Estos campos se especifican en el atributo `fields` de la clase `Meta`. El formulario se utilizará para recopilar datos y crear nuevas instancias del modelo `Post`. En pocas palabras, este código define un formulario que permite a los usuarios ingresar un título y texto para crear nuevos posts en el sistema.

```
from django import forms
from .models import Post

class CreateNewPost(forms.ModelForm):

    class Meta:
        model = Post
        fields = ['title', 'text',]
```

---

### Listing 8: CREACION DE LAS RUTAS DE URL

---

Este código define las rutas de URL para diferentes funcionalidades de una aplicación web utilizando el framework Django. Cada ruta está asociada a una vista específica, permitiendo el acceso a diferentes acciones como ver la página principal, crear un nuevo post, ver detalles de un post, editar un post existente y eliminar un post. Estas rutas permiten la navegación y la interacción con la aplicación de manera estructurada y brindan una forma de acceder a las diferentes funcionalidades ofrecidas por la aplicación web.

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

urlpatterns = [
    path('', views.index, name = 'index'),
    path('new_post/', views.newPost, name = "new_post" ),
    path('post/<int:id>', views.post, name = "post"),
    path('edit_post/<int:id>', views.editPost, name = "edit_post"),
    path('delete_post/<int:id>', views.deletePost, name = "delete_post")
]
```

---

### Listing 9: CREACION DE LAS FUNCIONES DE VISTA

---

Este código define las funciones de vista para diferentes acciones en una aplicación web utilizando el framework Django. La función `index` recupera todos los posts del modelo `Post` y los muestra en la página principal. La función `newPost` maneja tanto la visualización del formulario para crear un nuevo post como el procesamiento de la solicitud POST para guardar el nuevo post en la base de datos. La función `post` muestra los detalles de un post específico. La función `editPost` maneja tanto la visualización del formulario para editar un post existente como el procesamiento de la solicitud POST para guardar los cambios realizados. La función `deletePost` elimina un post específico de la base de datos. Estas funciones interactúan con las plantillas HTML correspondientes y utilizan el modelo `Post` y el formulario `CreateNewPost` para realizar operaciones de lectura, escritura y eliminación en la base de datos.

## Listing 10: CREACION DE LAS PLANTILLAS DE NUESTRO BLOG

A continuación, se tuvo que crear un archivo HTML que sirve como plantilla para la página de edición de un blog. En el encabezado del archivo, se carga el archivo de estilos CSS y se define el ícono de la página. El título de la página se establece como "EDICION DE BLOG". En el cuerpo del archivo, se encuentra un encabezado con el logo de la universidad y el título del blog. En el contenido principal, se encuentra un formulario que permite al usuario editar el título y el texto del blog. Se utiliza el token CSRF para proteger contra ataques de falsificación de solicitudes entre sitios. Los campos del formulario se generan utilizando las propiedades del formulario pasado desde la vista. Finalmente, hay un botón "Enviar" que envía el formulario cuando se hace clic en él.

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="{% static 'styles/style.css' %}" />
  <link rel="icon" type="image/png"
    href="https://cdn-icons-png.flaticon.com/512/7273/7273751.png">
  <title>EDICION DE BLOG</title>
</head>
<body>
  <header class="header">
    <div class="logo">
      
    </div>
    <div class="title">
      <h1>Mi Blog Personal</h1>
    </div>
  </header>

  <main class="main">
    <div class="container">
      <form method="post" class="my-form">
        {% csrf_token %}
        <div class="form-field">
          <label for="{% form.title.id_for_label %}"
            class="form-label">Titulo:</label>
          {{ form.title }}
        </div>
        <div class="form-field">
          <label for="{% form.text.id_for_label %}"
            class="form-label">Texto:</label>
          {{ form.text }}
        </div>
        <button class="my-button">Enviar</button>
      </form>
    </div>
  </main>
</body>
</html>
```

## Listing 11: CREACION DE LA PLANTILLA DE LA PAGINA PRINCIPAL

Después, se tuvo que crear otro archivo HTML que sirve como plantilla para la página principal del blog. En el encabezado del archivo, se carga el archivo de estilos CSS y se define el ícono de la página. El título de la página se establece como "MI BLOG". En el cuerpo del archivo, se encuentra un encabezado con el logo de la universidad y el título del blog. Además, hay un enlace "Nuevo Blog" que redirige al formulario de creación de un nuevo post. En el contenido principal, se encuentra una lista no ordenada que muestra todos los posts disponibles. Se utiliza un bucle "for" para recorrer cada post y generar un elemento de lista con un enlace que muestra el título del post. El enlace también redirige a la página de detalles de ese post en particular. Junto al título, se muestra la fecha de creación del post.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="{% static 'styles/style.css' %}" />
    <link rel="icon" type="image/png"
      href="https://cdn-icons-png.flaticon.com/512/7273/7273751.png">
    <title>MI BLOG</title>
  </head>
  <body>
    <header class="header">
      <div class="logo">
        
      </div>
      <div class="title">
        <h1>Mi Blog Personal</h1>
      </div>
      <div>
        <a href="{% url 'new_post' %}" class="add-button">Nuevo Blog</a>
      </div>
    </header>

    <main class="main">
      <ul class="article-list">
        {% for post in posts %}
        <li class="article-item">
          <article class="article">
            <h2 class="article__h2">
              <a href="{% url 'post' post.id %}" class="article__a">{{ post.title }}</a>
            </h2>
            <a href="{% url 'post' post.id %}" class="article__aa">{{
              post.created_date }}</a>
          </article>
        </li>
        {% endfor %}
      </ul>
    </main>
  </body>
```



&lt;/html&gt;

---

**Listing 12: CREACION DE LA PLANTILLA PARA GENERAR UN BLOG**

---

Después, se tuvo que crear otro archivo HTML que sirve como plantilla para la página de creación de un nuevo blog. En el encabezado del archivo, se carga el archivo de estilos CSS y se define el ícono de la página. El título de la página se establece como "NUEVO BLOG". En el cuerpo del archivo, se encuentra un encabezado con el logo de la universidad y el título del blog. Además, hay un enlace "Volver" que redirige a la página principal del blog. En el contenido principal, se encuentra un formulario que permite al usuario ingresar el título y el texto del nuevo blog. Se utiliza el token CSRF para proteger contra ataques de falsificación de solicitudes entre sitios. Los campos del formulario se generan utilizando las propiedades del formulario pasado desde la vista. Finalmente, hay un botón "Enviar" que envía el formulario cuando se hace clic en él.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="{% static 'styles/style.css' %}" />
    <link rel="icon" type="image/png"
      href="https://cdn-icons-png.flaticon.com/512/7273/7273751.png">
    <title>NUEVO BLOG</title>
  </head>
  <body>
    <header class="header">
      <div class="logo">
        
      </div>
      <div class="title">
        <h1>Mi Blog Personal</h1>
      </div>
      <div>
        <a href="{% url 'index' %}" class="add-button">Volver</a>
      </div>
    </header>

    <main class="main">
      <div class="container">
        <form method="post" class="my-form">
          {% csrf_token %}
          <div class="form-field">
            <label for="{% form.title.id_for_label %}" class="form-label">Titulo:</label>
            {{ form.title }}
          </div>
          <div class="form-field">
            <label for="{% form.text.id_for_label %}" class="form-label">Texto:</label>
            {{ form.text }}
          </div>
          <button class="my-button">Enviar</button>
        </form>
      </div>
    </main>
  </body>
</html>
```

```
</div>
</main>
</body>
</html>
```

### Listing 13: CREACION DE LA PLANTILLA PARA GUARDAR LOS BLOGS

A continuación, se ha creado otro archivo HTML que sirve como plantilla para la página de guardar un blog existente. En el encabezado del archivo, se carga el archivo de estilos CSS y se define el ícono de la página. El título de la página se establece como "GUARDAR MI BLOG". En el cuerpo del archivo, se encuentra un encabezado con el logo de la universidad y el título del blog. En el contenido principal, se muestra la información del blog, incluyendo la fecha de creación, el título y el texto del blog. Además, se incluyen enlaces que permiten editar, eliminar y volver a la página principal del blog. Estos enlaces utilizan las rutas definidas en las vistas correspondientes.

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="{% static 'styles/style.css' %}" />
    <link rel="icon" type="image/png"
      href="https://cdn-icons-png.flaticon.com/512/7273/7273751.png">
    <title>GUARDAR MI BLOG</title>
  </head>
  <body>
    <header class="header">
      <div class="logo">
        
      </div>
      <div class="title">
        <h1>Mi Blog Personal</h1>
      </div>
    </header>
    <main class="main">
      <ul class="article-list">
        <li class="article-item">
          <article class="article">
            <a class="article__aa">{{ post.created_date }}</a>
            <h2 class="article__h2">
              <a class="article__a">{{ post.title }}</a>
            </h2>
            <p class="article__p">{{ post.text }}</p>
            <a href="{% url 'edit_post' post.id %}" class="add-button">Editar</a>
            <a href="{% url 'delete_post' post.id %}" class="add-button">Eliminar</a>
            <a href="{% url 'index' %}" class="add-button">Volver</a>
          </article>
        </li>
      </ul>
    </main>
  </body>
</html>
```

---

**Listing 14: CREAMOS NUESTRO CSS**

---

Aquí, se proporciona estilos visuales y de diseño para la página web. Define estilos para el cuerpo del documento, incluyendo el espaciado, la altura mínima y el diseño flexible. Establece estilos para el encabezado, alineando los elementos y asignando colores de fondo y texto. También define estilos para los botones de agregar, especificando el color de fondo, el color del texto y los efectos al pasar el cursor sobre ellos. Además, establece estilos para la lista de artículos, definiendo el espaciado, los bordes, el tamaño de fuente y los colores de los enlaces. El código también proporciona estilos para los formularios, incluyendo el diseño, el espaciado, las etiquetas y los campos de entrada. Por último, se establecen estilos para los botones, especificando el color de fondo, el color del texto y los efectos al pasar el cursor sobre ellos. En resumen, el código CSS define la apariencia y el diseño visual de la página web, asegurando una experiencia estética y coherente para los usuarios.

```
...  
.header__a:hover {  
  color: hsl(0 0% 100%);  
  background-color: hsl(25 100% 45%);  
}  
  
.main {  
  background-color: hsl(0 0% 90%);  
  height: 100%;  
  flex-grow: 1;  
}  
  
.article-list {  
  list-style: none;  
  padding: 0;  
  margin: 80px;  
}  
...
```

---

**Listing 15: ADMINISTRADOR DE DJANGO**

---

Para agregar, editar y borrar los posts que hemos modelado, usaremos el administrador (admin) de Django. Abrimos el fichero blog/admin.py en el editor y reemplazamos su contenido con esto:

```
from django.contrib import admin  
from .models import Post  
  
# Register your models here.  
  
admin.site.register(Post)
```

Para iniciar sesión en el sitio web, necesitarás crear un superusuario, que es un usuario con privilegios especiales que puede controlar todo el sitio. Para hacer esto, regresa a la línea de comandos y escribe "python manage.py createsuperuser", luego presiona Enter. Esto ejecutará un comando que te guiará a través del proceso de creación del superusuario. Deberás proporcionar un nombre de usuario, una dirección de correo electrónico y una contraseña para el superusuario. Una vez que hayas completado estos pasos, podrás usar las credenciales del superusuario para acceder a las funciones y configuraciones avanzadas del sitio web.

```
(myvenv) C:\Users\Name\djangogirls> python manage.py createsuperuser
```

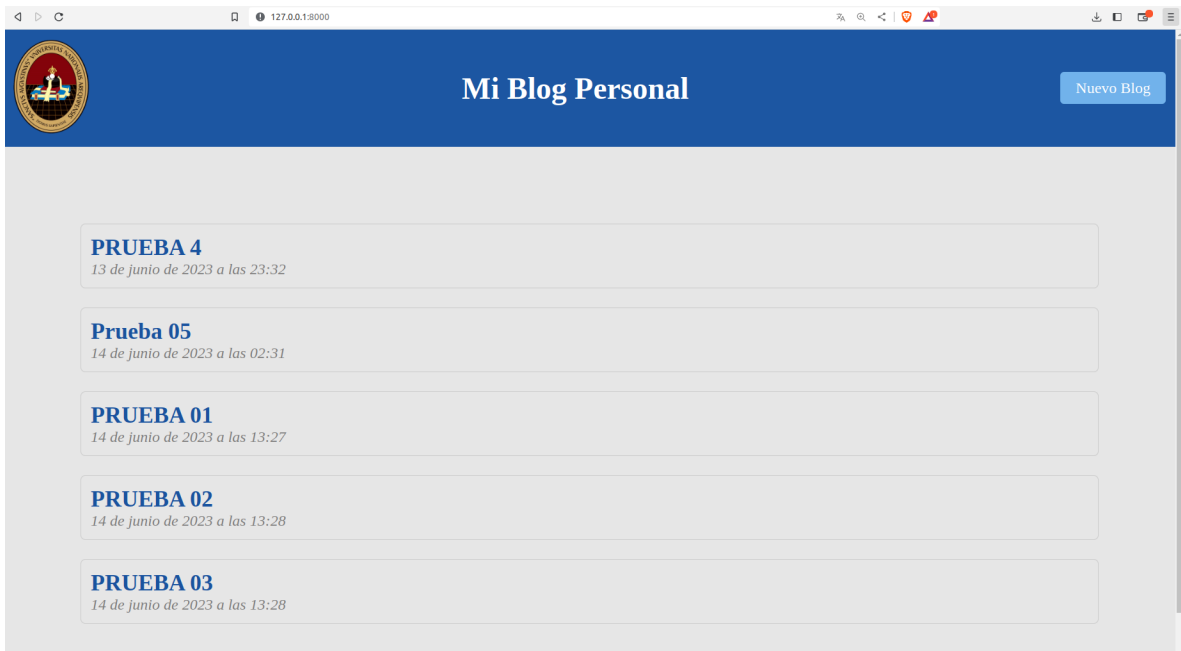
Luego, verás algo así (donde username y email serán los que escribiste anteriormente):

```
Username: ola
Email address: ola@example.com
Password:
Password (again):
Superuser created successfully.
```

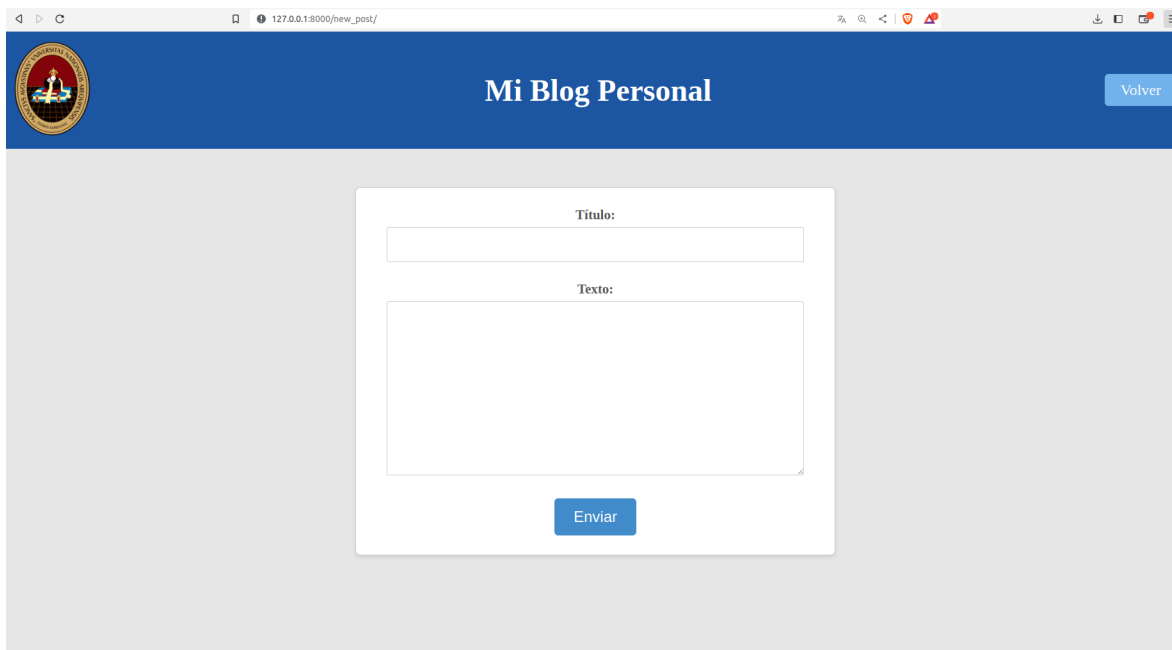


## 4. Mi Blog:

- Demostracion de **MI Blog Personal**
- Ejecutamos en nuestra consola lo siguiente 'python manage.py runserver' para correr el servidor web. En el navegador escribimos la dirección `http://127.0.0.1:8000/`.
- El `index.html` o pagina principal



- El newPost.html, esta pagina sirve para crear nuevos Blogs



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/new\_post/'. The page has a blue header with the university logo on the left, the title 'Mi Blog Personal' in the center, and a 'Volver' button on the right. The main content area is light gray and contains a white form with two input fields: 'Título:' and 'Texto:'. Below the 'Texto:' field is a blue 'Enviar' button.

- El post.html, esta pagina sirve para hacer el envio o tener la opcion de editar o eliminar nuestros Blogs

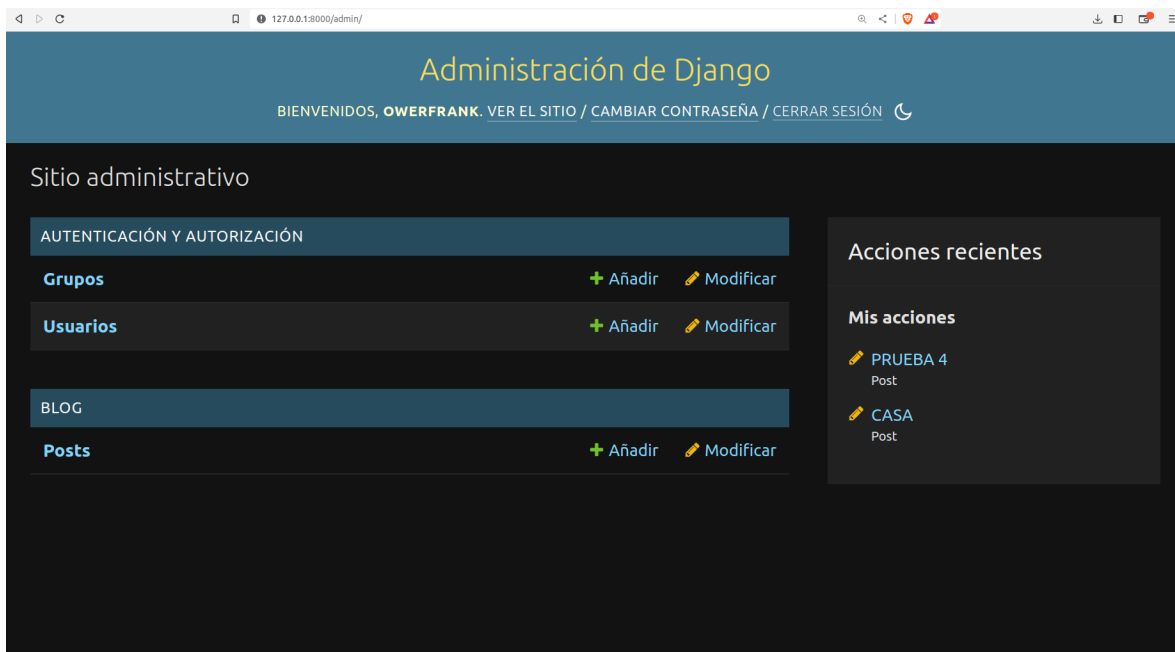


The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/post/6'. The page has a blue header with the university logo on the left, the title 'Mi Blog Personal' in the center, and a 'Volver' button on the right. The main content area is light gray and contains a white box with a timestamp '13 de junio de 2023 a las 23:32', the title 'PRUEBA 4', and the text 'ESTE ES UN TEXTO DE PRUEBA EDITADO ESTE ES UN TEXTO DE PRUEBA EDITADO ESTE ES UN TEXTO DE PRUEBA EDITADO'. Below the text are three buttons: 'Editar', 'Eliminar', and 'Volver'.

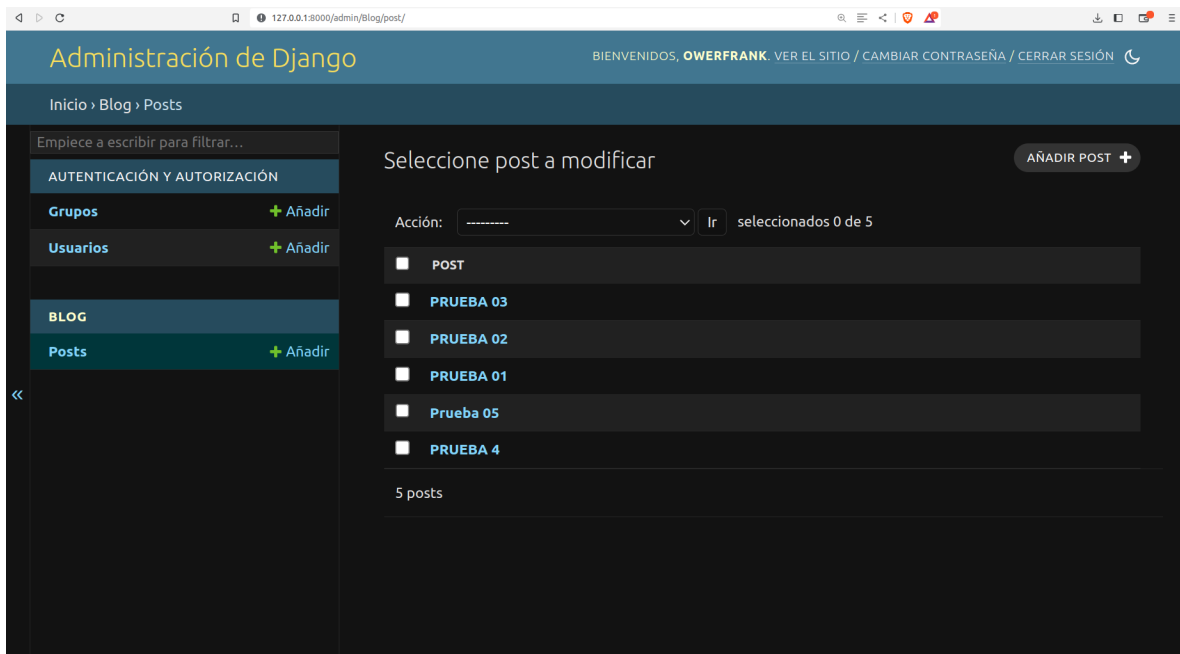
- El editPost.html, esta pagina sirve para editar nuestros Blogs



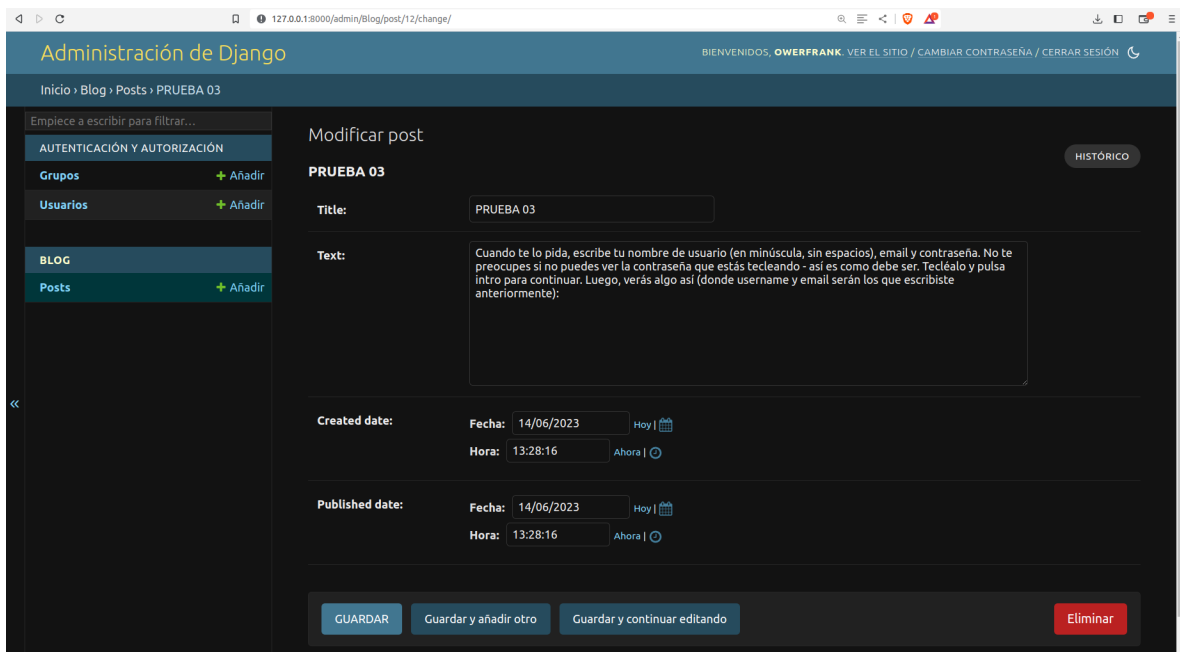
- Ahora veremos en la pagina /admin, veremos los blogs creados



- Ahora veremos en la pagina /admin, la lista de blogs creados



- Ahora veremos en la pagina /admin, la modificacion de la lista



## 5. Cuestionario

1. ¿Cuál es un estándar de codificación para Python? Ejemplo: Para PHP en el proyecto Pear <https://pear.php.net/manual/en/standards.php>

En Python, uno de los estándares de codificación más reconocidos y utilizados es PEP 8 (Python

Enhancement Proposal 8). PEP 8 es una guía de estilo que brinda recomendaciones sobre cómo escribir código Python legible y consistente. Algunas pautas clave de PEP 8 son las siguientes:

- **Indentación:** Se recomienda utilizar sangrías de 4 espacios para indentar el código.
- **Longitud de línea:** Se sugiere limitar la longitud de una línea de código a 79 caracteres. Esto ayuda a mantener la legibilidad en pantallas y editores de texto estándar.
- **Nombres de variables y funciones:** Se aconseja utilizar nombres descriptivos en minúsculas para variables y funciones. Si se necesita separar palabras, se recomienda utilizar guiones bajos (\_).
- **Espacios en blanco:** Es importante utilizar líneas en blanco para separar funciones y clases, así como bloques de código dentro de las funciones. Esto mejora la legibilidad del código.
- **Comillas:** Se puede utilizar tanto comillas simples (') como comillas dobles (") para representar cadenas. Sin embargo, se recomienda ser coherente en su uso y preferir las comillas simples, a menos que se necesite utilizar una comilla simple dentro de la cadena.
- **Importaciones:** Se sugiere importar módulos en líneas separadas y evitar el uso de importaciones genéricas como `import *`. Esto facilita la identificación de las dependencias del código.

Es importante tener en cuenta que PEP 8 es una guía, no un conjunto rígido de reglas. Puede haber situaciones en las que se justifique apartarse ligeramente de estas recomendaciones para mantener la legibilidad y la coherencia dentro de un proyecto o equipo en particular.

### 1. ¿Qué diferencias existen entre EasyInstall, pip, y PyPM?

EasyInstall fue una herramienta popular en el pasado utilizada para instalar paquetes en Python. Sin embargo, su desarrollo ha sido discontinuado y ha sido reemplazada por otras herramientas más modernas y ampliamente adoptadas.

pip es la herramienta de gestión de paquetes más comúnmente utilizada en el ecosistema de Python en la actualidad. Viene preinstalada en las versiones más recientes de Python y se ha convertido en el estándar de facto para la instalación y gestión de paquetes. pip ofrece una interfaz de línea de comandos sencilla y potente que permite instalar, actualizar y desinstalar paquetes de Python, así como manejar sus dependencias. También se integra con el Python Package Index (PyPI), el repositorio oficial de paquetes de Python, para buscar y descargar paquetes fácilmente.

Por otro lado, PyPM era una herramienta específica de la distribución ActivePython, proporcionada por ActiveState. Se basaba en el proyecto PPM (Perl Package Manager) y permitía instalar paquetes Python y gestionar dependencias en entornos de ActivePython. Sin embargo, en versiones más recientes de ActivePython, han migrado a utilizar conda y conda-forge como su sistema de gestión de paquetes predeterminado, dejando de lado PyPM.

#### 1. En un proyecto Django que se debe ignorar para usar git. Vea: <https://github.com/django/django/blob/main/.gitignore>. ¿Qué otros tipos de archivos se deberían agregar a este archivo?

- **Archivos de configuración sensibles:** Cualquier archivo que contenga información confidencial o sensible, como claves secretas, contraseñas de bases de datos u otros datos de autenticación, debe ser excluido del seguimiento de Git. Esto incluye archivos como `settings.py` o cualquier archivo que contenga información sensible.
- **Archivos de logs y registros:** Los archivos de registro generados por Django o cualquier otro registro personalizado pueden crecer rápidamente y no deben ser incluidos en el control de versiones. Esto puede incluir archivos con extensiones `.log`, `.txt`, `.log.gz` u otros formatos de registro.



- Archivos de bases de datos: Los archivos de base de datos generados por Django, como archivos SQLite (con extensión `.sqlite3`) o archivos de respaldo de bases de datos, deben excluirse del seguimiento de Git. Estos archivos son generados y actualizados por el propio sistema de gestión de bases de datos y no deben ser compartidos en el repositorio.
- Archivos estáticos generados: Si se utilizan herramientas como Django's `collectstatic` para recolectar archivos estáticos (como archivos CSS, JS o imágenes) en un solo directorio, esos archivos generados pueden agregarse a la lista de exclusión.
- Archivos de entorno virtual: Si se utiliza un entorno virtual para el proyecto, los directorios y archivos asociados (como `venv` o `env`) pueden ser agregados al archivo `.gitignore`. Esto evitará que se incluyan en el control de versiones y permitirá a otros desarrolladores crear su propio entorno virtual.

### 1. Utilice `python manage.py shell` para agregar objetos. ¿Qué archivos se modificaron al agregar más objetos?

Cuando utilizas `python manage.py shell` en Django para agregar objetos, los cambios generalmente afectan a dos tipos de archivos: los archivos de modelos y los archivos de migraciones.

- Archivos de modelos: Al agregar objetos en Django mediante el shell interactivo, es probable que estés creando instancias de modelos definidos en tu aplicación. Estos modelos se definen en archivos Python, donde se especifica la estructura de la base de datos y las operaciones relacionadas con los datos. Al agregar más objetos, es posible que debas modificar el archivo de modelos correspondiente para agregar las definiciones de los nuevos objetos creados.
- Archivos de migraciones: Django utiliza migraciones para realizar cambios en el esquema de la base de datos de manera controlada y consistente. Cuando agregas objetos a través del shell y estos objetos están asociados a modelos con campos nuevos o modificados, Django genera automáticamente nuevas migraciones para reflejar esos cambios. Estas migraciones se almacenan en el directorio `migrations` de tu aplicación y suelen tener nombres como `000X_nombre_de_migracion.py`, donde X es un número secuencial. Los archivos de migración contienen código Python que describe las operaciones necesarias para aplicar los cambios en el esquema de la base de datos, como crear nuevas tablas o agregar columnas.

Entonces, al agregar objetos en Django mediante el shell interactivo, es probable que necesites modificar los archivos de modelos para incluir las definiciones de los nuevos objetos y se generarán nuevas migraciones para reflejar los cambios en el esquema de la base de datos. Estas modificaciones garantizan que los objetos agregados se persistan correctamente y se pueda realizar un seguimiento de los cambios en la estructura de la base de datos.

## 6. Referencias

- [https://www.w3schools.com/python/python\\_reference.asp](https://www.w3schools.com/python/python_reference.asp)
- <https://docs.python.org/3/tutorial/>
- <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Models>
- [https://tutorial.djangogirls.org/es/django\\_models/](https://tutorial.djangogirls.org/es/django_models/)
- <https://pear.php.net/manual/en/standards.php>
- <https://docs.djangoproject.com/en/4.0/>
- <https://www.youtube.com/watch?v=M4NI54BM1dk>

- <https://pypi.org/>
- [https://pip.pypa.io/en/latest/user\\_guide/](https://pip.pypa.io/en/latest/user_guide/)
- <https://packaging.python.org/en/latest/tutorials/installing-packages/>