

Informe de Laboratorio 05

Tema: Python - Django

Nota

Estudiante	Escuela	Asignatura
Jose Alonzo Gordillo Mendoza jgordillome@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 20220577

Laboratorio	Tema	Duración
05	Python - Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 08 Mayo 2023	Al 15 Junio 2023

1. Tarea

- Crea un blog sencillo en un entorno virtual utilizando la guía: 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/JoseGordilloMendoza/lab05-PW2.git>

3. Ejercicios

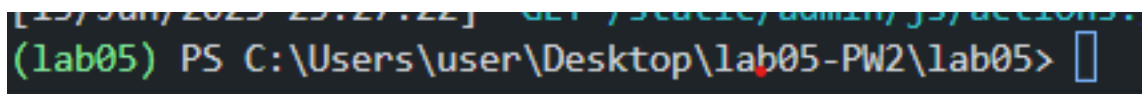
3.1. Estructura de laboratorio 05

- La distribución de archivos será la siguiente (teniendo en cuenta solo el entorno virtual y los archivos más importantes, por ejemplo lo descargado en el entorno es demasiado extenso como para incluirlo, el latex y un README estarán en la carpeta anterior):

```
lab05-PW2/
|----lab05/
|----blog
|----__pycache__
|----migrations
|----static
|----css
|----blog.css
|----templates
|----blog
|----base.html
|----post_detail.html
|----post_edit.html
|----post_list.html
|----__init__.py
|----admin.py
|----apps.py
|----forms.py
|----models.py
|----tests.py
|----urls.py
|----views.py
|----Lib
|----mysite
|----__pycache__
|----__init__.py
|----settings.py
|----urls.py
|----wsgi.py
|----Scripts
|----db.sqlite3
|----manage.py
|----pyenv.cfg
```

3.2. Análisis de archivos

- En primer lugar hay que hablar del entorno virtual, que como vimos en la clase pasada, nos ayudara a crear nuestra blog en este caso:



```
[15/Jan/2023 23:27:22] GET /static/admin/js/actions.  
(lab05) PS C:\Users\user\Desktop\lab05-PW2\lab05>
```

Listing 1: Analizando bloque 1 de settings.py

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

Estas son las configuraciones de las aplicaciones instaladas, la ultima aplicacion "blog.es" la que creamos para nuestro blog personal.

Listing 2: Settings.py - cambio de idioma y region

```
LANGUAGE_CODE = 'es-es'  
TIME_ZONE = 'America/Lima'
```

En este bloque vemos como hemos modificado el idioma que se manejara asi como la region que eran originalmente ingles y UTC respectivamente.

Listing 3: Settings.py - cambio 3

```
STATIC_URL = 'static/'  
STATIC_ROOT = BASE_DIR / 'static'
```

Se agrego la ultima linea al "settings.py."original para especificar la ubicación de la carpeta raíz donde se almacenarán los archivos estáticos recopilados en el proyecto.

- Una vez modificado esto veamos la aplicacion "bloggreada a traves del comando "python manage.py startapp blog"que crea un nuevo directorio; por tanto analicemos en primer lugar a "models.py"

Listing 4: bloque4

```
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)  
    created_date = models.DateTimeField(  
        blank=True, null=True)  
  
    def publish(self):  
        self.published_date = timezone.now()  
        self.save()  
  
    def __str__(self):  
        return self.title
```

Este es nuestro modelo Post, antes de verlo mejor, tengamos en cuenta las importaciones que hacemos del mismo django, usando sus modelos y el timezone especialmente para obtener la fecha actual; entonces veamos que se tienen los atributos author, title, created_date y created_date para definir sus respectivos aspectos usando al usuario que inicio sesion, un titulo y texto a ingresar, la hora de creacion; ademas de metodos que publicaran/guardaran los posts y nos devolveras el titulo de objeto como representacion.

- Archivo admin

Listing 5: admin.py

```
from django.contrib import admin

from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Como vemos importamos el modelo Post antes presentado, y se registra en la interfaz de administración de Django. Esto permitirá que el modelo Post sea administrado y se pueda crear, editar y eliminar instancias de Post a través de la interfaz de administración. Para iniciar sesión, se deberá crear un superusuario (superuser), con la línea de comando "manage.py createsuperuser".

- Archivo urls

Listing 6: urls.py

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

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new', views.post_new, name='post_new'),
    path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
    path('delete_post/<int:id>', views.deletePost, name = "delete_post")
]
```

Estas son las rutas que usaremos en nuestro blog, la primera siendo la ruta raíz o la pagina principal, la segunda ruta se utiliza para mostrar detalles de un post específico, la tercer ruta se encargará de mostrar el formulario de creación de un nuevo post, luego la cuarta que se utilizara para editar un post existente y la ultima con la que borraremos el post.

- Archivos views

Listing 7: Primer parte de views.py

```
def post_list(request):
    posts=
        Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
```

```
form = PostForm()
return render(request, 'blog/post_edit.html', {'form': form})
```

En primer lugar `post_list` maneja la vista de la lista de publicaciones en el blog. Filtra las publicaciones por fecha de publicación anterior o igual a la fecha y hora actual, luego, ordena las publicaciones por fecha de publicación ascendente, para luego pasarlas publicaciones filtradas como contexto. Por otra parte `post_detail` maneja la vista de detalle de una publicación específica en el blog. Por último `post_new` maneja la vista para crear una nueva publicación en el blog. Si la solicitud es de tipo POST, se crea una instancia de `PostForm` con los datos proporcionados en la solicitud. Luego, se verifica si el formulario es válido utilizando `is_valid()`. Si es válido, se guarda la instancia de `Post`. Se establece el autor de la publicación como el usuario actual y la fecha de publicación como la fecha y hora actual. Finalmente, se guarda la publicación y se redirige a la vista de detalle de la publicación creada. Si la solicitud no es de tipo POST, se crea una instancia vacía de `PostForm`.

Listing 8: Segunda parte de `views.py`

```
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})

def deletePost(request, id):
    post = Post.objects.get(id=id)
    post.delete()
    return redirect('/')
```

Esta función `post_edit` maneja la vista para editar una publicación existente en el blog. Toma como argumentos la solicitud (`request`) y el parámetro `pk` que representa la clave primaria de la publicación que se desea editar, en esta se obtiene una instancia de la publicación específica, si la publicación no existe, se devuelve una página de error 404. Dentro del bloque `if`, se crea una instancia de `PostForm` con los datos enviados en la solicitud. Se utiliza el argumento `instance=post` para vincular el formulario con la instancia de la publicación que se está editando. Se verifica si el formulario es válido. Si es válido, se guardan los cambios en la instancia de `Post`. Luego, se guarda la publicación y se redirige a la vista de detalle de la publicación editada. Si la solicitud no es de tipo POST, se crea una instancia de `PostForm` con los datos de la publicación existente. Es decir, esta función permite editar una publicación existente en el blog. La última función `deletePost` elimina un objeto `Post` de la base de datos según el `id` proporcionado y redirige al usuario a la página principal del blog.

- Plantillas HTML (haremos uso de variables con llaves, así como el extenderse de archivos que sirven como base)

Listing 9: `base.html`

```
{% load static %}
<html>
<head>
```

```
<title>BLOG</title>
<link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
<link
      href="//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext"
      rel="stylesheet" type="text/css">
<link rel="stylesheet" href="{% static 'css/blog.css' %}">
</head>
<body>
  <div class="page-header">
    <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon
      glyphicon-plus"></span></a>
    <h1><a href="/">MI PRIMER BLOG EN DJANGO</a></h1>
  </div>
  <div class="content container">
    <div class="row">
      <div class="col-md-8">
        {% block content %}
        {% endblock %}
      </div>
    </div>
  </div>
</body>
</html>
```

Esta plantilla HTML proporciona una estructura básica para las páginas del blog. Define un encabezado con un enlace para crear una nueva publicación y un título del blog, y proporciona una estructura de columnas para el contenido principal.

Listing 10: post_detail.html

```
{% extends 'blog/base.html' %}
{% block content %}
  <div class="post">
    {% if post.published_date %}
      <div class="date">
        {{ post.published_date }}
      </div>
    {% endif %}
    {% if user.is_authenticated %}
      <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk
        %}"><span class="glyphicon glyphicon-pencil"></span></a>
    {% endif %}
    <h2>{{ post.title }}</h2>
    <p>{{ post.text|linebreaksbr }}</p>
  </div>
{% endblock %}
```

En esta plantilla vemos que se extiende (extends) la plantilla base y define el contenido específico para la página de detalle de una publicación de blog. Muestra la fecha de publicación (si está disponible), un botón de edición para usuarios autenticados, el título de la publicación y el texto de la publicación con saltos de línea convertidos en elementos HTML `
`.

Listing 11: post_edit.html

```
{% extends 'blog/base.html' %}
{% block content %}
    <h2>New post</h2>
    <form method="POST" class="post-form"> {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Guardar</button>
    </form>
{% endblock %}
```

Esta plantilla extiende la plantilla base nuevamente y define el contenido específico para la página de creación de una nueva publicación en el blog. Muestra un encabezado "New post" seguido de un formulario que permite al usuario ingresar los detalles de la nueva publicación y guardarla al hacer clic en el botón "Guardar".

Listing 12: post_list.html

```
{% extends 'blog/base.html' %}
{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h2><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a></h2>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

Esta plantilla también extiende la plantilla base y define el contenido específico para la página de lista de publicaciones del blog. Utiliza un bucle para mostrar cada publicación individualmente. Cada publicación se muestra con su fecha de publicación, título (como un enlace a la página de detalle de la publicación) y texto.

Archivos de estilos, blogs.css

Listing 13: blog.css

```
.page-header {
    background-color: #C25100;
    margin-top: 0;
    padding: 20px 20px 20px 40px;
}
.page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1
a:active {
    color: #ffffff;
    font-size: 36pt;
    text-decoration: none;
}
.content {
    margin-left: 40px;
}
h1, h2, h3, h4 {
    font-family: 'Impact', cursive;
}
.date {
```

```
        color: #828282;}
    .save {
        float: right;}

    .post-form textarea, .post-form input {
        width: 100%;}
    .top-menu, .top-menu:hover, .top-menu:visited {
        color: #ffffff;
        float: right;
        font-size: 26pt;
        margin-right: 20px;}
    .post {
        margin-bottom: 70px; }

    .post h2 a, .post h2 a:visited {
        color: #000000;
    }
```

Estos son los estilos que se usaron en el blog, que usan adicionalmente Bootstrap; lo importante es mencionar que este es un archivo estatico, que maneja clases de los html anteriormente vistos.

Archivo forms

Listing 14: forms.py

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

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('title', 'text',)
}
```

Este bloque de código define un formulario llamado PostForm que se utiliza para crear o editar una publicación en el blog. El formulario está vinculado al modelo Post y contiene los campos title y text. Este formulario facilita la creación y edición de publicaciones en el blog al proporcionar un conjunto de campos predefinidos y lógica de validación integrada.

3.3. Demostracion de pagina

Ahoa veamos a la pagina como tal:

Primero tenemos que correr nuestro server con el comando "python manage.py run-server", una vez dentro veamos la pantall principal:

MI PRIMER BLOG EN DJANGO



13 de junio de 2023 a las 21:58

Sample title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

13 de junio de 2023 a las 21:58

Post nuevo

Probando en post nuevo

13 de junio de 2023 a las 22:03

POST 3

Contenido de post 3

Como vemos tenemos tres posts iniciales y un botón para poder agregar más posts, pero si queremos entrar a un post, simplemente le damos en su título:

MI PRIMER BLOG EN DJANGO



13 de junio de 2023 a las 23:27



Sample title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Esta es la vista que tenemos de un post específico, como vemos tenemos un lapicito con el que podemos editar el post, siempre y cuando seamos un usuario verificado y dueño del post, a su derecha hay un tacho que es para eliminar el post siguiendo la misma lógica de estar "logueado":

MI PRIMER BLOG EN DJANGO



New post

Title:

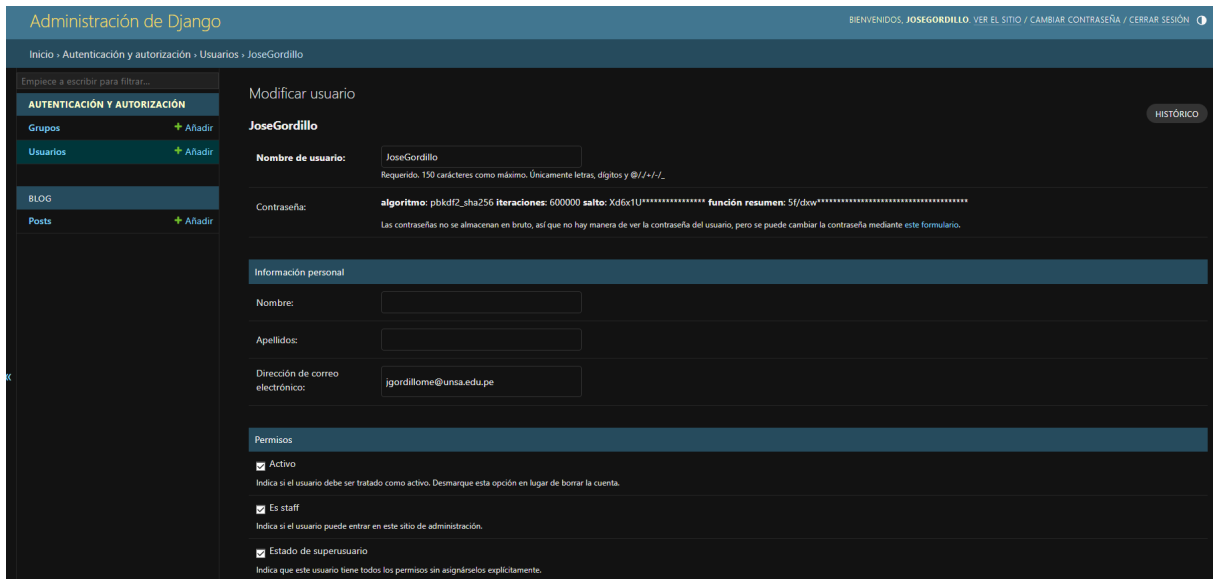
Sample title

Text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Guardar

Ahora veamos en el apartado de administrador nuestro usuario:



Administración de Django BIENVENIDOS, JOSEGORDILLO VER EL SITIO / CAMBIAR CONTRASEÑA / CERRAR SESIÓN

Inicio » Autenticación y autorización » Usuarios » JoseGordillo

Empezar a escribir para filtrar...

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos [+ Añadir](#)

Usuarios [+ Añadir](#)

BLOG

Posts [+ Añadir](#)

Modificar usuario HISTÓRICO

JoseGordillo

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

Contraseña: **algoritmo:** pbkdf2_sha256 **iteraciones:** 600000 **salto:** Xd6x1U***** **función resumen:** 5l/dxw*****
 Las contraseñas no se almacenan en bruto, así que no hay manera de ver la contraseña del usuario, pero se puede cambiar la contraseña mediante este formulario.

Información personal

Nombre:

Apellidos:

Dirección de correo electrónico: jgordillome@unsa.edu.pe

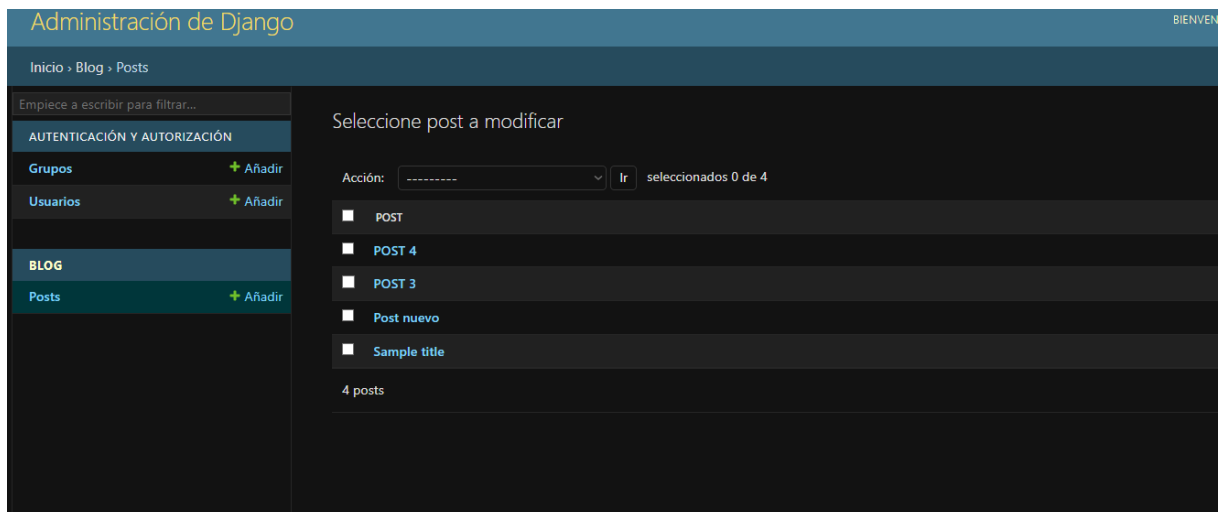
Permisos

☒ **Activo**
 Indica si el usuario debe ser tratado como activo. Desmarque esta opción en lugar de borrar la cuenta.

☒ **Es staff**
 Indica si el usuario puede entrar en este sitio de administración.

☒ **Estado de superusuario**
 Indica que este usuario tiene todos los permisos sin asignárselos explícitamente.

Ademas tenemos los posts que se han ido creando:



Administración de Django BIENVEN

Inicio » Blog » Posts

Empezar a escribir para filtrar...

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos [+ Añadir](#)

Usuarios [+ Añadir](#)

BLOG

Posts [+ Añadir](#)

Seleccione post a modificar

Acción: seleccionados 0 de 4

☐ POST

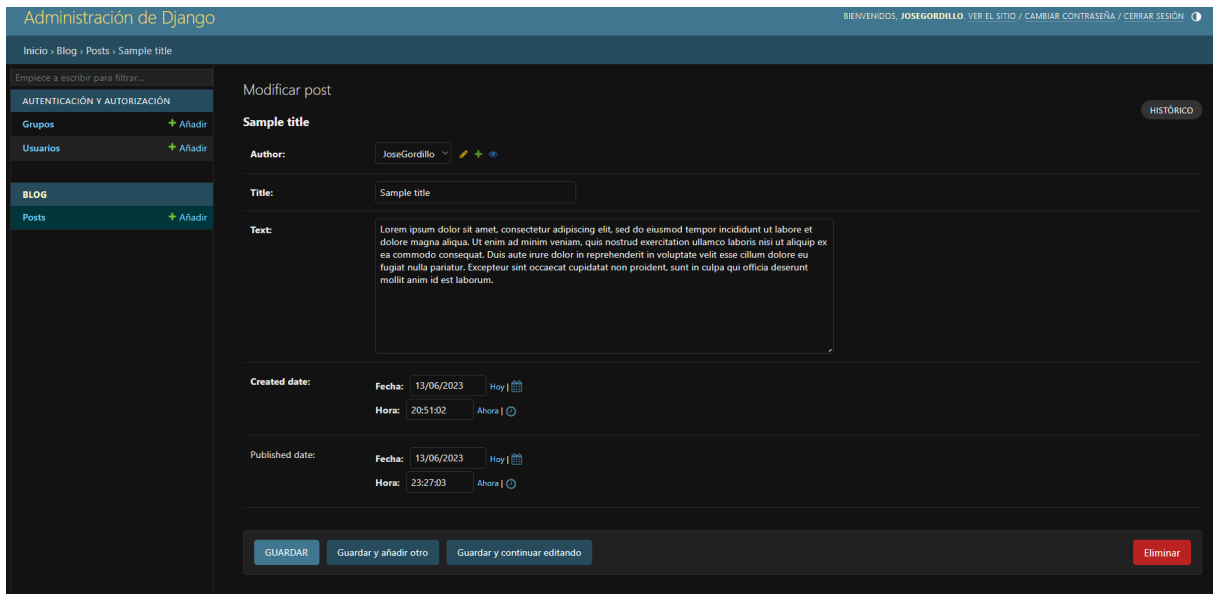
☐ POST 4

☐ POST 3

☐ Post nuevo

☐ Sample title

4 posts



Link donde esta alojado el video (FlipGrid): <https://flip.com/groups/14644257/topics/36914916/responses/429099066/comments>

4. Cuestionario

- ¿Cuál es un estándar de codificación para Python? Ejemplo: Para PHP en el proyecto Pear <https://pear.php.net/manual/en/standards.php> Uno de los estándares de codificación más ampliamente aceptados y utilizados en Python se conoce como PEP 8 (Python Enhancement Proposal 8). PEP 8 establece una guía de estilo para escribir código Python con el fin de mejorar la legibilidad y fomentar la coherencia entre proyectos. Estos son algunos ejemplos de las recomendaciones del PEP 8:

- **Convenciones de nomenclatura**

Usar snake_case para nombres de variables y nombres de funciones.

CamelCase debe usarse para nombres de clase.

Hacer uso de MAYUSCULAS_PARA_CONSTANTES.

- **Indentado y espacio:**

Usar cuatro espacios en lugar de representaciones tabulares para la sangría.

No exceder los 79 caracteres por línea de código.

Dejar dos líneas en blanco entre las definiciones de clases y funciones.

- **Importaciones:**

Completar las importaciones en líneas separadas para cada módulo.

Evitar importar varios elementos en la misma línea.

- **Comentarios:**

Utilizar comentarios para explicar el código de manera concisa y clara.

Evitar hacer comentarios redundantes o innecesarios.

- ¿Qué diferencias existen entre EasyInstall, pip, y PyPM?

EasyInstall, pip y PyPM son herramientas utilizadas en el ecosistema de Python para la instalación y gestión de paquetes. EasyInstall fue una herramienta popular en versiones antiguas de Python, pero ha sido reemplazada por pip, que es la herramienta de facto estándar en la comunidad de Python. Pip permite instalar paquetes de Python desde el registro PyPI y otros repositorios, además de gestionar dependencias y actualizaciones de paquetes de manera sencilla.

Por otro lado, PyPM es una herramienta utilizada específicamente en el lenguaje Perl, no está relacionada con el sistema de paquetes de Python y tiene funcionalidades y objetivos diferentes. En resumen EasyInstall es una herramienta de instalación y gestión de paquetes obsoleta, pip es la herramienta estándar de facto para instalar paquetes en Python, y PyPM es una herramienta utilizada específicamente para el lenguaje Perl.

- 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?

El archivo .gitignore en un proyecto Django generalmente se utiliza para especificar los archivos y directorios que no se deben incluir en el repositorio Git. Esto ayuda a evitar que se suban al repositorio archivos innecesarios o sensibles que no deben ser compartidos públicamente. Algunos otros tipos de archivos que generalmente se deben agregar al archivo .gitignore en un proyecto Django:

- **Archivos de configuración local:** Esto puede incluir archivos de configuración utilizados para almacenar datos sensibles como claves secretas, contraseñas de base de datos, etc.

- **Archivos generados automáticamente:** Estos archivos generados no son esenciales para el repositorio y pueden ser excluidos.

- **Directorios y archivos estáticos generados:** Los archivos generados, como archivos CSS compilados o archivos minificados, no necesitan estar en el repositorio y pueden ser excluidos.

- **Archivos de entorno:** Si se utiliza un archivo de entorno (por ejemplo, .env) para configurar variables de entorno o configuraciones específicas del entorno.

- **Archivos de registro:** por ejemplo, *.log, para evitar que se llenen con registros de desarrollo o registros confidenciales.

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

Los archivos que se modificaron son:

Archivo de migraciones (migrations/): Se generaron archivos de migración en la carpeta migrations/. Estos archivos contienen instrucciones para aplicar los cambios en la base de datos y reflejar la adición de nuevos objetos o campos en el modelo.

Archivo de base de datos (db.sqlite3, postgresql, etc.): En mi caso se usa SQLite, entonces al agregar objetos en la consola de Django, los datos se almacenarán en la base de datos. Por lo tanto, el archivo de la base de datos se modificará para incluir los nuevos registros correspondientes a los objetos agregados.

5. Conclusiones

- Django es un marco de desarrollo web poderoso y de alto nivel que ofrece una amplia gama de características y funcionalidades para crear aplicaciones web complejas.
- Django se destaca por su enfoque en la eficiencia y la seguridad, ofreciendo funciones integradas para proteger las aplicaciones web contra amenazas comunes y brindando un rendimiento efectivo y escalable.
- La comunidad de Django está activa y ofrece una amplia gama de recursos, documentación y paquetes complementarios.
- Django es un marco flexible y escalable que se puede ajustar para satisfacer las diversas necesidades y requisitos del proyecto. Es posible utilizarlo para desarrollar aplicaciones web de cualquier tamaño, desde pequeñas aplicaciones hasta proyectos empresariales de gran escala.

6. Referencias

- 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://pear.php.net/manual/en/standards.php>
- <https://docs.djangoproject.com/en/4.0/>
- <https://www.youtube.com/watch?v=M4NIIs4BM1dk>
- <https://pypi.org/>
- https://pip.pypa.io/en/latest/user_guide/
- <https://packaging.python.org/en/latest/tutorials/installing-packages/>