Programación Web II

Trabajo Práctico N°3

Estudiante Enrique Armando Bogarin
2do Año - Tecnicatura Universitaria en Desarrollo de Software
Prof. y Lic. Julio César Casco

Repositorio del proyecto:
https://github.com/Boga-in/Web2TP3

Objetivos

El objetivo de este trabajo práctico es desarrollar un proyecto llamado Blog y una aplicación denominada post utilizando Django y el patrón de diseño MVT.
Se espera que implementen funcionalidades esenciales para la gestión de publicaciones (post) en un blog, trabajando con vistas, plantillas y modelos, así como la integración con GitHub y el uso de un entorno virtual.
Poder implementar un middleware personalizado, habilitar y utilizar el middleware de autenticación que provee Django.

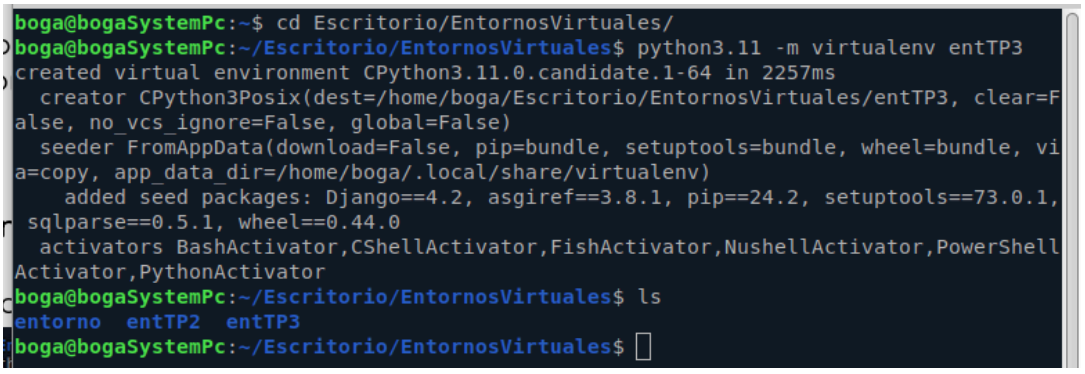
Introducción

En este proyecto planeo completar los objetivos listados con anterioridad en un proyecto Django "Blog" donde se podrá listar los posteos que podemos tanto crear como eliminar.
La aplicación contará con un inicio como pantalla principal y un navbar que guiará al usuario a las funcionalidades. Un login y un registro para los usuarios que usen la aplicación.

Comenzaré con un pequeño tutorial de como configurar el entorno y seguiré con el desarrollo del proyecto y la aplicación.

Creación de Entorno y Proyecto

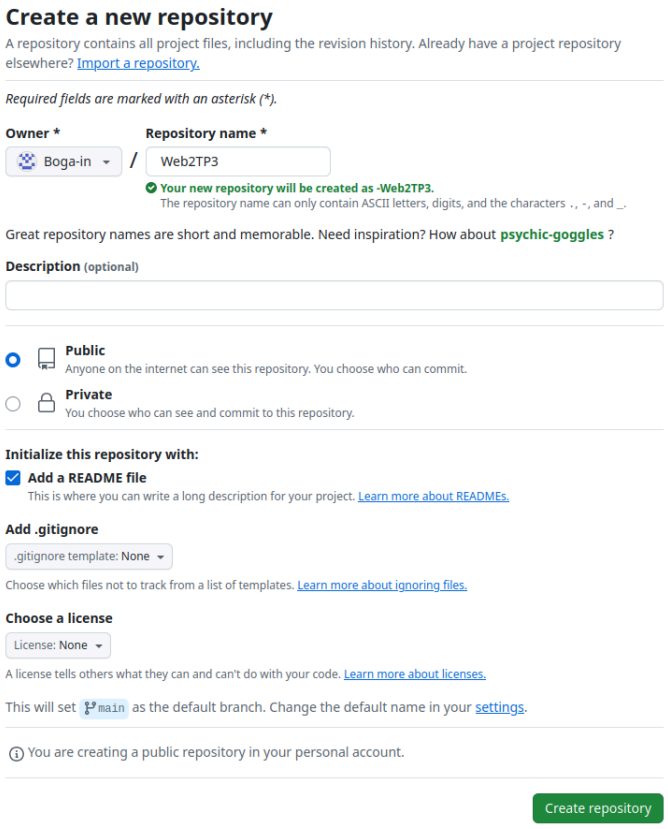
1- Crearemos un entorno virtual



2- Instalaremos, ya dentro del entorno, la version Django que utilizaremos en nuestro proyecto

```
boga@bogaSystemPc:~/Escritorio/ProyectosDjango$ cd
boga@bogaSystemPc:~$ source /home/boga/Escritorio/EntornosVirtuales/entTP3/bin/activate
(entTP3) boga@bogaSystemPc:~$ cd Escritorio/ProyectosDjango/
(entTP3) boga@bogaSystemPc:~/Escritorio/ProyectosDjango$ ls
 Integradora   Tablero_de_Mensajes   Web2TP3
'PWII TP1'     Tarea_De_Mensajes     WebIITP2
(entTP3) boga@bogaSystemPc:~/Escritorio/ProyectosDjango$ pip install django==4.2
Requirement already satisfied: django==4.2 in /home/boga/Escritorio/EntornosVirtuales/entTP3
/lib/python3.11/site-packages (4.2)
Requirement already satisfied: asgiref<4,>=3.6.0 in /home/boga/Escritorio/EntornosVirtuales/
entTP3/lib/python3.11/site-packages (from django==4.2) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in /home/boga/Escritorio/EntornosVirtuales/en
tTP3/lib/python3.11/site-packages (from django==4.2) (0.5.1)
(entTP3) boga@bogaSystemPc:~/Escritorio/ProyectosDjango$
```

3- Crearemos nuestro repositorio de Github donde alojaremos nuestro proyecto



4- Haremos un clone de nuestro repositorio en la carpeta donde tendremos nuestro proyecto

- git clone git@github.com:Boga-in/Web2TP3.git

5- Crearemos nuestro proyecto con el administrador de DJango

- django-admin startproject Blog

6- Seguiremos con la creación de de la App en la carpeta donde alojamos al proyecto

- python manage.py startapp post

7- Haremos un add de todo el proyecto creado a Git

- git add .

8- Entonces un commit con un mensaje correspondiente para luego hacer su debido push al repositorio

- git commit -m “Primer commit base del proyecto ”

Con estos pasos podremos comenzar a desarrollar nuestra app.

Modelo de Post

Contará con cuatro atributos base, dos de ellos limitados a 50 y 150 caracteres cada uno y uno siendo un atributo de tipo DateTime al que se le asigna la fecha y hora del momento en que se cree un nuevo post.
El autor en cambio usa el modelo de User de Django para hacer referencia y luego permita eliminar y editar solo si sos el autor de dicho posteo.

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

class Post(models.Model):
   titulo = models.CharField(max_length=50)
   contenido = models.TextField(max_length=150)
   fecha_publicacion = models.DateTimeField(default=timezone.now)
   autor = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='blog_publicado')

   def __str__(self):
       return f"{self.titulo} de {self.autor}: {self.contenido}"
```

Creación del Formulario

Para esta aplicación usaremos un formulario que reciba los datos requeridos correspondientes al modelo

```
from django import forms
from .models import Post
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class PostForm(forms.ModelForm):
   class Meta:
       model = Post
       fields = ['titulo', 'contenido']
       widgets = {
           'titulo': forms.TextInput(attrs={'class': 'form-control'}),
           'contenido': forms.Textarea(attrs={'class': 'form-control',
'rows': 4}),
       }
       labels = {
           'titulo': 'Titulo',
           'contenido': 'contenido',
       }
```

Funcionalidades implementadas

`RegistroUsuarioView` (Vista para el Registro de Usuarios)

Tipo: CBV (CreateView)

Método GET: Muestra un formulario de registro.

Método POST: Procesa el formulario y crea un nuevo usuario en la base de datos.

Parámetros:

`form`: El formulario que recibe los datos del usuario para el registro.

Métodos adicionales:

`form_valid()`: Guarda el usuario y lo autentica automáticamente al momento de registrarlo.

```
class RegistroUsuarioView(CreateView):

   model = User

   form_class = RegistroForm

   template_name = 'registro.html'

   success_url = reverse_lazy('post:index')

   def form_valid(self, form):

       response = super().form_valid(form)

       user = form.save()

       login(self.request, user)

       return response
```

`LoginUsuarioView` (Vista para Iniciar Sesión)

Tipo: CBV (LoginView)

Método GET: Muestra el formulario de login.

Método POST: Autentica al usuario usando el formulario.

Parámetros:

`request`: Representa la solicitud enviada por el cliente.

Métodos adicionales:

`get_success_url()`: Define a qué página redirigir al usuario una vez autenticado.

```python
class LoginUsuarioView(LoginView):

   template_name = 'login.html'

   redirect_authenticated_user = True

   success_url = reverse_lazy('post:index')

   def get_success_url(self):

       return self.success_url
```

`LogoutUsuarioView` (Vista para Cerrar Sesión)

Tipo: CBV (LogoutView)

Método GET: Cierra la sesión del usuario actual y lo redirige a la página principal o al login.

Parámetros:

`next_page`: URL a la que redirige después de cerrar sesión.

```python
class LogoutUsuarioView(LogoutView):

 next_page = reverse_lazy('post:index')
```

`CrearPostView` (Vista para Crear Publicaciones)

Tipo: CBV (CreateView)

Método GET: Muestra el formulario para crear una publicación.

Método POST: Procesa el formulario para crear una nueva publicación.

Parámetros:

`form`: El formulario que recibe los datos de la publicación.

Métodos adicionales:

`form_valid()`: Asigna automáticamente al usuario autenticado como autor del post y luego lo guarda.

```python
class CrearPostView(LoginRequiredMixin, CreateView):

    model = Post

    form_class = PostForm

    template_name = 'crear_post.html'

    context_object_name = 'publicacion'

    success_url = reverse_lazy('post:index')

    def form_valid(self, form):

        form.instance.autor = self.request.user

        return super().form_valid(form)
```

`EditarPublicacionView` (Vista para Editar Publicaciones)

Tipo: CBV (UpdateView)

Método GET: Muestra el formulario precargado con los datos actuales de la publicación para editar.

Método POST: Procesa el formulario para actualizar la publicación en la base de datos.

Parámetros:

`form`: El formulario de edición de la publicación.

Métodos adicionales:

`test_func()`: Verifica si el usuario autenticado es el autor del post que quiere editar.

```
class EditarPublicacionView(UserPassesTestMixin,
UpdateView):

   model = Post

   form_class = PostForm

   template_name = 'editar_publicacion.html'

   context_object_name = 'publicacion'

   success_url = reverse_lazy('post:lista_publicaciones')

   def test_func(self):

       publicacion = self.get_object()

       return self.request.user == publicacion.autor
```

`ListaPublicacionesView` (Vista para Listar Publicaciones)

Tipo: CBV (ListView)

Método GET: Muestra una lista de todas las publicaciones ordenadas por fecha de publicación (más reciente primero).

Parámetros:

`publicaciones`: Contiene la lista de todas las publicaciones.

```python
class ListaPublicacionesView(ListView):

    model = Post

    template_name = 'lista_publicaciones.html'

    context_object_name = 'publicaciones'

    ordering = ['-fecha_publicacion']
```

`DetallePublicacionView` (Vista para Ver el Detalle de una Publicación)

Tipo: CBV (DetailView)

Método GET: Muestra los detalles completos de una publicación seleccionada (título, contenido, fecha y autor).

Parámetros:

`publicacion`: Representa el objeto de la publicación específica.

```python
class DetallePublicacionView(DetailView):

    model = Post

    template_name = 'detalle_publicacion.html'

    context_object_name = 'publicacion'
```

`EliminarPublicacionView` (Vista para Eliminar una Publicación)

Tipo: CBV (DeleteView)

Método GET: Muestra una página de confirmación para eliminar la publicación.

Método POST: Elimina la publicación de la base de datos.

Parámetros:

`publicacion`: Representa el objeto de la publicación que se va a eliminar.

Métodos adicionales:

`test_func()`: Verifica si el usuario autenticado es el autor del post que quiere eliminar.

```
class EliminarPublicacionView(UserPassesTestMixin, DeleteView):
   model = Post
   template_name = 'eliminar_publicacion.html'
   success_url = reverse_lazy('post:lista_publicaciones')
   def test_func(self):
       publicacion = self.get_object()
       return self.request.user == publicacion.autor
```

ENLACES

A continuacion el archivo urls.py

```
from django.urls import path
from .views import CrearPostView, index, RegistroUsuarioView,
LoginUsuarioView, LogoutUsuarioView
from .views import (ListaPublicacionesView, DetallePublicacionView,
EliminarPublicacionView, EditarPublicacionView)
app_name = 'post'

urlpatterns = [
   path('', index, name='index'),
   path('crear/', CrearPostView.as_view(), name='crear_post'),
   path('registro/', RegistroUsuarioView.as_view(), name='registro'),
   path('login/', LoginUsuarioView.as_view(), name='login'),
```

```
    path('logout/', LogoutUsuarioView.as_view(), name='logout'),
    path('lista_publicaciones/', ListaPublicacionesView.as_view(),
name='lista_publicaciones'),
    path('publicacion/<int:pk>/', DetallePublicacionView.as_view(),
name='detalle_publicacion'),
    path('publicacion/<int:pk>/eliminar/',
EliminarPublicacionView.as_view(), name='eliminar_publicacion'),
    path('publicacion/<int:pk>/editar/',
EditarPublicacionView.as_view(), name='editar_publicacion'),
]
```

Templates

1- Base.html : Es la plantilla base que proporciona la estructura común para todas las páginas. Todas las demás plantillas extienden de esta.
Contenido:

- **Bloque `title`**: Puede ser sobrescrito por las plantillas hijas para establecer el título específico de cada página.
- **Bloque `content`**: Define la sección donde se inyectará el contenido único de cada página.
- **Barra de navegación**: Cambia según si el usuario está autenticado o no, mostrando enlaces como "Crear Post", "Iniciar sesión", "Registrarse" o "Cerrar sesión".

2. **`lista_publicaciones.html:`** Muestra una lista de publicaciones en tarjetas. Cada publicación incluye su título, autor, y fecha de publicación. Los autores pueden editar o eliminar sus propias publicaciones.

- Uso de DTL:
    - **Bucle `{% for %}`**: Itera sobre todas las publicaciones y las presenta en tarjetas.

- Condicional `{% if %}`: Verifica si el usuario autenticado es el autor del post, lo que permite mostrar los botones de edición y eliminación.
- Filtro `date`: Formatea la fecha de publicación.

3. **`crear_post.html`:**Función: Permite a los usuarios crear una nueva publicación.

- Uso de DTL:
  - Formulario dinámico: Utiliza `{{ form.as_p }}` para mostrar automáticamente el formulario definido en el archivo `forms.py`.
  - Token CSRF: Protección contra ataques CSRF con `{% csrf_token %}`.

4. **`detalle_publicacion.html`:**Función: Muestra el detalle completo de una publicación, incluyendo opciones de editar o eliminar si el usuario es el autor.

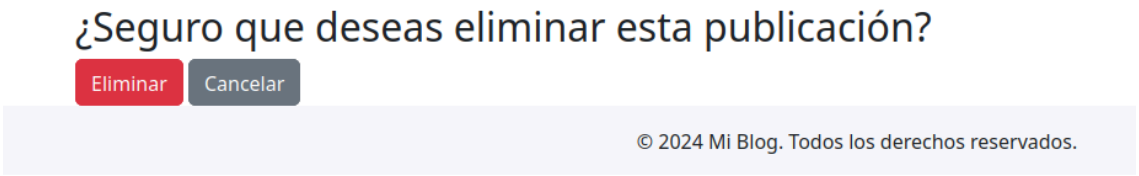
- Uso de DTL:
  - Condicional `{% if %}`: Verifica si el usuario autenticado es el autor de la publicación para mostrar los botones de editar y eliminar.
  - Filtro `date`: Formateo de la fecha de publicación.

5. **`editar.html`:** Permite al autor editar una publicación existente.

- Uso de DTL:
  - **Formulario dinámico**: Usa `{{ form.as_p }}` para mostrar automáticamente el formulario de edición del post.
  - **Token CSRF**: Incluye protección contra ataques CSRF.

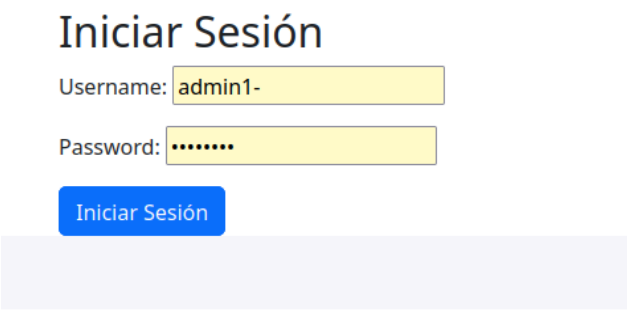
6. eliminar_publicacion.html: Confirma la eliminación de una publicación.

- **Uso de DTL:**
  - **Condicional**: Verifica si el usuario autenticado es el autor de la publicación antes de eliminarla.



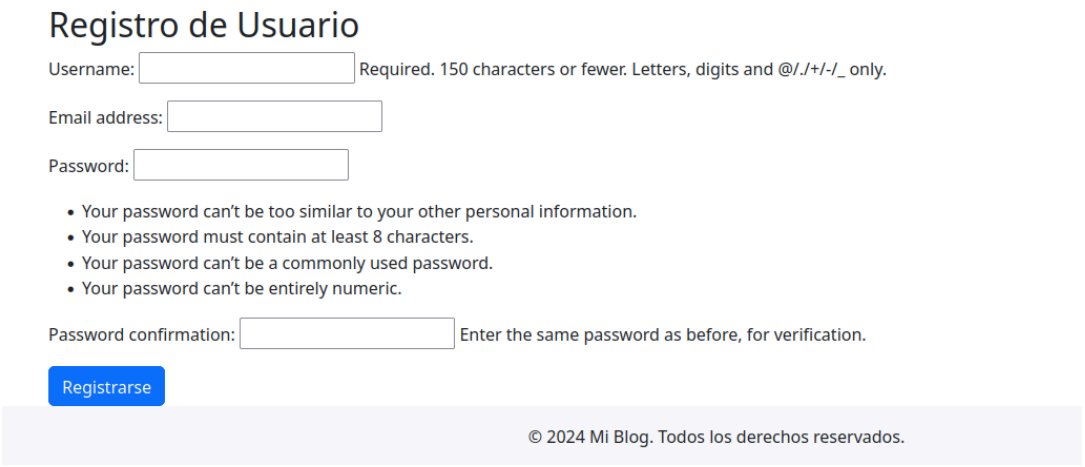
7. login.html: Permite a los usuarios iniciar sesión.

- **Uso de DTL:**
  - **Formulario dinámico**: Muestra los campos del formulario de inicio de sesión con `{{ form.as_p }}`.



8. registro.html: Permite a los usuarios registrarse en el sistema.

- Uso de DTL:
  - **Formulario dinámico**: Utiliza `{{ form.as_p }}` para generar el formulario de registro automáticamente.



MIDDLEWARE

```
class RequestLogMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        print(f"New Request: {request.method} {request.path}")
        print(f"PARAMETROS GET: {request.GET}")
        print(f"PARAMETROS POST: {request.POST}")

        response = self.get_response(request)
```

```
    return response
```

Este middleware está centrado en registrar los **parámetros GET y POST** que acompañan a la solicitud.

- **`__init__(self, get_response)`**: Se ejecuta al inicializar el middleware, guardando la función `get_response` para permitir el paso de la solicitud.
- **`__call__(self, request)`**: Se ejecuta en cada solicitud y:
    - **Registra el método HTTP y la ruta** de la solicitud.
    - Imprime los **parámetros GET y POST** enviados en la solicitud.
    - Llama al siguiente middleware o vista con `self.get_response(request)` y retorna la respuesta sin modificaciones.

PROBLEMAS ENCONTRADOS

Fue algo confuso la implementación de un usuario a la hora permitir el eliminar o editar, pero pude solucionarlo en parte gracias a foros de preguntas y las últimas clases sincrónicas.
No llegue con los tiempos debido a lo lento que avanzaba con las pruebas y errores así que no quede conforme con el trabajo.
Sigo cometiendo el error de desarrollar sin subir cambios al repositorio, por eso la app completa la subo por completo en un solo commit, espero solucionarlo en futuros trabajos.

CONCLUSIÓN

Este trabajo práctico sirvió para poder entender mejor el funcionamiento de los middlewares ya que es algo que no

tenía tan incorporado incluso ya trabajando en el área de desarrollo de software.
La creación del view también me dio un mejor entendimiento del manejo de los post creados. Lo mismo aplico al manejo de sesión con el login y el logout

Versiones usadas

```
asgiref 3.8.1
Django 4.2
sqlparse 0.5.1
Python 3.11.0rc1
pip 24.2
```

Software Utilizado

Visual Studio Code
Terminal de Linux
Git hub

Bibliografía

Unidad 3 - Campus UDC - Profesor Julio Casco