Curso de Django

Módulo 3: Vistas y Plantillas.

Lcdo. Diego Medardo Saavedra García. Mgtr.

2023-07-28

Tabla de contenidos

1 Módulo 3: Vistas y Plantillas.

1.1 Creación de Vistas en Django

Las vistas en Django son funciones que procesan las solicitudes del usuario y devuelven una respuesta HTTP. Cada vista debe recibir una solicitud como argumento y devolver una respuesta.

Ejemplo de una vista que muestra un mensaje de bienvenida:

Paso 1: En el archivo "views.py" de la aplicación "publicaciones", agrega el código de la vista de bienvenida:

```
# publicaciones/views.py

from django.http import HttpResponse

def vista_bienvenida(request):
    return HttpResponse("¡Bienvenido al blog!")
```

Paso 3: Configura la URL para la vista en el archivo "urls.py" de la aplicación "publicaciones":

```
# publicaciones/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('bienvenida/', views.vista_bienvenida, name='vista_bienvenida'),
]
```

Paso 4: Configura la URL de la aplicación **publicaciones** en el archivo **urls.py** del proyecto:

```
# blog/urls.py

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

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

```
    path('publicaciones/', include('publicaciones.urls')), # Agrega esta línea para inclui
]
```

Paso 5: Ahora, ejecuta el servidor de desarrollo con el siguiente comando:

```
python manage.py runserver
```

Paso 6: Abre tu navegador web e ingresa a la siguiente dirección:

http://127.0.0.1:8000/publicaciones/bienvenida/

Deberías ver el mensaje "¡Bienvenido al blog!" en el navegador.

1.2 Otra forma de generar un Hola Mundo en Django.

Paso 1: Creación de la vista en views.py:

En esta etapa, se crea una vista llamada "HolaMundoView" utilizando la clase TemplateView. Esta vista simplemente renderiza la plantilla "hola_mundo.html" que muestra un mensaje "Hola Mundo!".

```
# views.py

from django.shortcuts import render
from django.views.generic import TemplateView

class HolaMundoView(TemplateView):
    template_name = 'hola_mundo.html'
```

Paso 2: Configuración de las URLs en urls.py:

En el archivo urls.py de la aplicación "publicaciones", se define la URL para la vista "HolaMundoView". También se incluyen las URLs de la aplicación en las URLs globales del proyecto.

```
# urls.py de publicaciones

from django.urls import path
from .views import HolaMundoView

urlpatterns = [
    path('hola_mundo/', HolaMundoView.as_view(), name='hola_mundo'), ]
```

Paso 3 Creación de la plantilla hola_mundo.html:

La plantilla “hola_mundo.html” es un archivo HTML simple que muestra el mensaje “Hola Mundo!” en un encabezado h1.

```
<!DOCTYPE html>
<html>
<head>
<title>Hola Mundo</title>
</head>
<body>
<h1>Hola Mundo!</h1>
</body>
</html>
```

Paso 4: Configuración de las URLs globales del proyecto en urls.py:

En el archivo urls.py del proyecto principal, se incluye la URL de la aplicación “publicaciones” utilizando la función “include”. Esto permitirá acceder a las URLs de la aplicación a través de la URL base “publicaciones/”.

```
# urls.py del proyecto
from django.contrib import admin
from django.urls import path, include

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

Con estos pasos, hemos creado una aplicación simple que muestra el mensaje “Hola Mundo!” en la página cuando accedemos a la URL “publicaciones/hola_mundo/”. Además, hemos configurado la conexión entre las URLs de la aplicación y las URLs globales del proyecto. A partir de aquí, podemos agregar más funcionalidades y vistas a nuestra aplicación utilizando los modelos “Publicacion” y “Comentario”.

1.3 Mostrar Publicaciones y Comentarios

Para mostrar las publicaciones y comentarios agregados en los modelos, primero, asegúrate de que hayas definido correctamente los modelos “Publicacion” y “Comentario” en el archivo models.py de la aplicación “publicaciones” como se mostró en ejemplos anteriores.

1.4 Crea una vista para mostrar las publicaciones:

En el archivo views.py de la aplicación "publicaciones", crea una vista llamada "ListaPublicacionesView" para mostrar todas las publicaciones:

```
# publicaciones/views.py

from django.views.generic import ListView
from .models import Publicacion

class ListaPublicacionesView(ListView):
    model = Publicacion
    template_name = 'lista_publicaciones.html'
    context_object_name = 'publicaciones'
```

Crea una plantilla para mostrar la lista de publicaciones:

Crea un archivo llamado "lista_publicaciones.html" dentro de la carpeta "templates" de la aplicación "publicaciones":

```
<!-- publicaciones/templates/lista_publicaciones.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Lista de Publicaciones</title>
</head>
<body>
    <h1>Lista de Publicaciones</h1>
    <ul>
        {% for publicacion in publicaciones %}
            <li>{{ publicacion.titulo }}</li>
            <ul>
                {% for comentario in publicacion.comentarios.all %}
                    <li>{{ comentario.texto }}</li>
                {% endfor %}
            </ul>
        {% endfor %}
    </ul>
</body>
</html>
```

En este ejemplo, estamos utilizando una estructura de bucles for en la plantilla para mostrar las publicaciones y sus comentarios asociados.

Configura las URLs para mostrar la lista de publicaciones:

En el archivo urls.py de la aplicación "publicaciones", agrega la configuración para mostrar la lista de publicaciones:

```
# publicaciones/urls.py

from django.urls import path
from .views import ListaPublicacionesView

urlpatterns = [
    path('publicaciones/', ListaPublicacionesView.as_view(), name='lista_publicaciones'),
]
```

Actualiza las URLs del proyecto:

En el archivo urls.py del proyecto "blog", actualiza las URLs de la aplicación "publicaciones" para que se muestren en la ruta principal:

```
# blog/urls.py

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

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

Ejecuta el servidor de desarrollo:

Ahora, ejecuta el servidor de desarrollo nuevamente con el siguiente comando:

```
python manage.py runserver
```

Accede a la URL http://localhost:8000/publicaciones/ en tu navegador y deberías ver la lista de publicaciones y sus comentarios asociados.

Si no has agregado publicaciones o comentarios en la base de datos, es posible que no veas datos en la lista.

¡Listo! Ahora has configurado el proyecto “blog” para mostrar las publicaciones y comentarios agregados en los modelos “Publicacion” y “Comentario” utilizando el modelo Template View de Django.

1.5 Sistema de Plantillas de Django (Jinja2)

- Django utiliza el motor de plantillas Jinja2 para gestionar la presentación de los datos en las vistas.
- Jinja2 es un poderoso motor de plantillas que permite incrustar código Python y generar HTML de forma dinámica.
- En el contexto del proyecto “blog” que hemos estado desarrollando, el sistema de plantillas de Django se encargará de renderizar las vistas y mostrar las publicaciones con sus comentarios en la plantilla “lista_publicaciones.html”.

1.6 Conceptos Principales del Sistema de Plantillas de Django:

Templates: Los templates son archivos HTML que contienen código Python que define cómo se mostrarán los datos en la interfaz de usuario.

En nuestro caso, el archivo “lista_publicaciones.html” será un template donde mostraremos la lista de publicaciones y sus comentarios.

Contexto: El contexto es un diccionario de Python que contiene los datos que se van a renderizar en el template.

En este contexto, proporcionaremos la lista de publicaciones y sus comentarios para que sean mostrados en la plantilla.

Variables de Plantilla: En los templates de Django, podemos utilizar variables de plantilla para acceder a los datos proporcionados en el contexto.

Por ejemplo, podemos utilizar la variable publicaciones para acceder a la lista de publicaciones y sus comentarios.

Directivas de Control: Jinja2 permite utilizar directivas de control, como bucles y condicionales, en los templates para generar contenido de forma dinámica.

Esto nos permite iterar sobre la lista de publicaciones y mostrar cada una de ellas con sus comentarios.

1.7 Pasos para Utilizar el Sistema de Plantillas de Django en el Proyecto "blog":

Pasos para Utilizar el Sistema de Plantillas de Django en el Proyecto "blog":

Paso 1: Crear el archivo "lista_publicaciones.html"

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "lista_publicaciones.html" donde definiremos la estructura HTML y utilizaremos las variables de plantilla para mostrar los datos.

```
<!-- lista_publicaciones.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Publicaciones</title>
</head>
<body>
    <h1>Lista de Publicaciones</h1>
    <ul>
        {% for publicacion in publicaciones %}
        <li>
            <h2>{{ publicacion.titulo }}</h2>
            <p>{{ publicacion.contenido }}</p>
            <h3>Comentarios:</h3>
            <ul>
                {% for comentario in publicacion.comentarios.all %}
                <li>{{ comentario.texto }}</li>
                {% endfor %}
            </ul>
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Paso 2: Definir las Vistas

En el archivo "views.py" de la aplicación "publicaciones", definimos las vistas que serán responsables de obtener los datos de la base de datos (en este caso, las publicaciones y sus comentarios) y pasarlos al template.

```
# views.py
from django.shortcuts import render
```

```
from .models import Publicacion

def lista_publicaciones(request):
    publicaciones = Publicacion.objects.all()
    context = {'publicaciones': publicaciones}
    return render(request, 'lista_publicaciones.html', context)
```

Paso 3: Utilizar el Contexto

En las vistas, creamos un contexto que contiene los datos que queremos mostrar en el template. En nuestro caso, el contexto contendrá la lista de publicaciones y sus comentarios.

Paso 4: Renderizar el Template

Finalmente, en las vistas, utilizamos el método render() para renderizar el template "lista_publicaciones.html" con el contexto que creamos. Esto generará el contenido HTML dinámico que mostrará las publicaciones y sus comentarios.

Paso 5: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", definimos la URL que se utilizará para acceder a la vista que renderiza el template "lista_publicaciones.html".

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
]
```

Con estos pasos, habremos integrado el sistema de plantillas de Django (Jinja2) en nuestro proyecto "blog" y podremos mostrar de forma dinámica las publicaciones y sus comentarios en la plantilla "lista_publicaciones.html". Al acceder a la URL "/lista/", se mostrará la lista de publicaciones con sus comentarios.

2 CRUD de Publicaciones

2.1 Crear Publicaciones

Paso 1: Crear el formulario de Publicación

En el archivo "forms.py" de la aplicación "publicaciones", creamos un formulario para la creación de publicaciones.

```python
# forms.py
from django import forms
from .models import Publicacion

class PublicacionForm(forms.ModelForm):
    class Meta:
        model = Publicacion
        fields = ['titulo', 'contenido', 'autor']
```

Paso 2: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para crear una nueva publicación y renderizar el formulario.

```python
# views.py
from django.shortcuts import render, redirect
from .forms import PublicacionForm

def crear_publicacion(request):
    if request.method == 'POST':
        form = PublicacionForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_publicaciones')
    else:
        form = PublicacionForm()
    return render(request, 'crear_publicacion.html', {'form': form})
```

Paso 3: Crear la plantilla para el formulario de creación

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "crear_publicacion.html" que contendrá el formulario de creación de publicaciones.

```html
<!-- crear_publicacion.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Crear Publicación</title>
</head>
<body>
    <h1>Crear Nueva Publicación</h1>
    <form method="post">
        {% csrf_token %}
```

```
        {{ form.as_p }}
        <button type="submit">Crear</button>
    </form>
</body>
</html>
```

Paso 4: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de creación de publicaciones.

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
]
```

2.2 Leer Publicaciones

Paso 1: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para mostrar la lista de publicaciones.

```
# views.py
from django.shortcuts import render
from .models import Publicacion

def lista_publicaciones(request):
    publicaciones = Publicacion.objects.all()
    return render(request, 'lista_publicaciones.html', {'publicaciones': publicaciones})
```

Paso 2: Actualizar el archivo "lista_publicaciones.html"

En la plantilla "lista_publicaciones.html", podemos acceder a las publicaciones y mostrarlas en una lista.

```
<!-- lista_publicaciones.html -->
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Lista de Publicaciones</title>
</head>
<body>
    <h1>Lista de Publicaciones</h1>
    <ul>
        {% for publicacion in publicaciones %}
        <li>
            <h2>{{ publicacion.titulo }}</h2>
            <p>{{ publicacion.contenido }}</p>
            <h3>Comentarios:</h3>
            <ul>
                {% for comentario in publicacion.comentarios.all %}
                <li>{{ comentario.texto }}</li>
                {% endfor %}
            </ul>
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

2.3 Actualizar Publicaciones

Paso 1: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para actualizar una publicación existente.

```
# views.py
from django.shortcuts import render, get_object_or_404, redirect
from .forms import PublicacionForm
from .models import Publicacion

def actualizar_publicacion(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    if request.method == 'POST':
        form = PublicacionForm(request.POST, instance=publicacion)
        if form.is_valid():
            form.save()
```

```
            return redirect('lista_publicaciones')
    else:
        form = PublicacionForm(instance=publicacion)
    return render(request, 'actualizar_publicacion.html', {'form': form})
```

Paso 2: Crear la plantilla para el formulario de actualización

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "actualizar_publicacion.html" que contendrá el formulario de actualización de publicaciones.

```
<!-- actualizar_publicacion.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Actualizar Publicación</title>
</head>
<body>
    <h1>Actualizar Publicación</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Guardar Cambios</button>
    </form>
</body>
</html>
```

Paso 3: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de actualización de publicaciones.

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
]
```

2.4 Eliminar Publicaciones

Paso 1: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para eliminar una publicación existente.

```
# views.py
from django.shortcuts import get_object_or_404, redirect
from .models import Publicacion

def eliminar_publicacion(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    if request.method == 'POST':
        publicacion.delete()
        return redirect('lista_publicaciones')
    return render(request, 'eliminar_publicacion.html', {'publicacion': publicacion})
```

Paso 2: Crear la plantilla para confirmar la eliminación

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "eliminar_publicacion.html" que contendrá la confirmación para eliminar la publicación.

```
<!-- eliminar_publicacion.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Eliminar Publicación</title>
</head>
<body>
    <h1>Eliminar Publicación</h1>
    <p>¿Estás seguro de que deseas eliminar la publicación "{{ publicacion.titulo }}"?</p>
    <form method="post">
        {% csrf_token %}
        <button type="submit">Eliminar</button>
    </form>
</body>
</html>
```

Paso 3: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de eliminación de publicaciones.

```python
# urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),
]
```

2.5 CRUD de Comentarios

El CRUD de comentarios sigue un proceso similar al CRUD de publicaciones. A continuación, se describen los pasos para cada operación: Crear Comentarios

Paso 1: Crear el formulario de Comentario

En el archivo "forms.py" de la aplicación "publicaciones", creamos un formulario para la creación de comentarios.

```python
# forms.py
from django import forms
from .models import Comentario

class ComentarioForm(forms.ModelForm):
    class Meta:
        model = Comentario
        fields = ['publicacion', 'autor', 'contenido']
```

Paso 2: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para crear un nuevo comentario y renderizar el formulario.

```python
# views.py
from django.shortcuts import render, redirect
from .forms import ComentarioForm

def crear_comentario(request):
    if request.method == 'POST':
        form = ComentarioForm(request.POST)
```

```
        if form.is_valid():
            form.save()
            return redirect('lista_publicaciones')
    else:
        form = ComentarioForm()
    return render(request, 'crear_comentario.html', {'form': form})
```

Paso 3: Crear la plantilla para el formulario de creación

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "crear_comentario.html" que contendrá el formulario de creación de comentarios.

```
<!-- crear_comentario.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Crear Comentario</title>
</head>
<body>
    <h1>Crear Nuevo Comentario</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Crear</button>
    </form>
</body>
</html>
```

Paso 4: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de creación de comentarios.

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),
    path('crear_comentario/', views.crear_comentario, name='crear_comentario'),
```

```
]
```

2.6 Leer Comentarios

`Paso 1:` Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para mostrar la lista de comentarios.

```
# views.py
from django.shortcuts import render
from .models import Comentario

def lista_comentarios(request):
    comentarios = Comentario.objects.all()
    return render(request, 'lista_comentarios.html', {'comentarios': comentarios})
```

Paso 2: Actualizar el archivo "lista_comentarios.html"

En la plantilla "lista_comentarios.html", podemos acceder a los comentarios y mostrarlos en una lista.

```
<!-- lista_comentarios.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Comentarios</title>
</head>
<body>
    <h1>Lista de Comentarios</h1>
    <ul>
        {% for comentario in comentarios %}
        <li>
            <p>Comentario de {{ comentario.autor }} en {{ comentario.publicacion.titulo }}
            <p>{{ comentario.contenido }}</p>
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Paso 3: Configurar las URLs

En el archivo “urls.py” de la aplicación “publicaciones”, agregamos una URL para la vista de lista de comentarios.

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),
    path('crear_comentario/', views.crear_comentario, name='crear_comentario'),
    path('lista_comentarios/', views.lista_comentarios, name='lista_comentarios'),
]
```

2.7 Actualizar Comentarios

Paso 1: Actualizar el archivo “views.py”

En el archivo “views.py” de la aplicación “publicaciones”, definimos una vista para actualizar un comentario existente.

```
# views.py
from django.shortcuts import render, get_object_or_404, redirect
from .forms import ComentarioForm
from .models import Comentario

def actualizar_comentario(request, pk):
    comentario = get_object_or_404(Comentario, pk=pk)
    if request.method == 'POST':
        form = ComentarioForm(request.POST, instance=comentario)
        if form.is_valid():
            form.save()
            return redirect('lista_comentarios')
    else:
        form = ComentarioForm(instance=comentario)
    return render(request, 'actualizar_comentario.html', {'form': form})
```

Paso 2: Crear la plantilla para el formulario de actualización

En el directorio “templates” de la aplicación “publicaciones”, creamos el archivo “actualizar_comentario.html” que contendrá el formulario de actualización de comentarios.

```
<!-- actualizar_comentario.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Actualizar Comentario</title>
</head>
<body>
    <h1>Actualizar Comentario</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Guardar Cambios</button>
    </form>
</body>
</html>
```

Paso 3: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de actualización de comentarios.

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

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),
    path('crear_comentario/', views.crear_comentario, name='crear_comentario'),
    path('lista_comentarios/', views.lista_comentarios, name='lista_comentarios'),
    path('actualizar_comentario/<int:pk>/', views.actualizar_comentario, name='actualizar_
]
```

2.8 Eliminar Comentarios

Paso 1: Actualizar el archivo "views.py"

En el archivo "views.py" de la aplicación "publicaciones", definimos una vista para eliminar un comentario existente.

```python
# views.py
from django.shortcuts import get_object_or_404, redirect
from .models import Comentario

def eliminar_comentario(request, pk):
    comentario = get_object_or_404(Comentario, pk=pk)
    if request.method == 'POST':
        comentario.delete()
        return redirect('lista_comentarios')
    return render(request, 'eliminar_comentario.html', {'comentario': comentario})
```

Paso 2: Crear la plantilla para confirmar la eliminación

En el directorio "templates" de la aplicación "publicaciones", creamos el archivo "eliminar_comentario.html" que contendrá la confirmación para eliminar el comentario.

```html
<!-- eliminar_comentario.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Eliminar Comentario</title>
</head>
<body>
    <h1>Eliminar Comentario</h1>
    <p>¿Estás seguro de que deseas eliminar el comentario de "{{ comentario.autor }}" en "
    <form method="post">
        {% csrf_token %}
        <button type="submit">Eliminar</button>
    </form>
</body>
</html>
```

Paso 3: Configurar las URLs

En el archivo "urls.py" de la aplicación "publicaciones", agregamos una URL para la vista de eliminación de comentarios.

```python
# urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
```

```
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),
    path('crear_comentario/', views.crear_comentario, name='crear_comentario'),
    path('lista_comentarios/', views.lista_comentarios, name='lista_comentarios'),
    path('actualizar_comentario/<int:pk>/', views.actualizar_comentario, name='actualizar_
    path('eliminar_comentario/<int:pk>/', views.eliminar_comentario, name='eliminar_coment
]
```

2.9 Migrar y Ejecutar el Servidor

Paso 1: Aplicar las migraciones

Después de agregar los modelos y las vistas, es necesario aplicar las migraciones para crear las tablas correspondientes en la base de datos.

Ejecutamos el siguiente comando:

```
python manage.py makemigrations
python manage.py migrate
```

Paso 2: Ejecutar el servidor

Finalmente, para ver nuestro proyecto en funcionamiento, ejecutamos el servidor de desarrollo de Django.

Ejecutamos el siguiente comando:

```
python manage.py runserver
```

Con esto, podemos acceder a nuestro sistema CRUD de publicaciones y comentarios en el navegador, utilizando las URLs configuradas en las vistas y templates.

Por ejemplo,

1. Para ver la lista de publicaciones, accedemos a “/lista/”
2. Para crear una nueva publicación, accedemos a “/crear/”.
3. Para ver la lista de comentarios, accedemos a “/lista_comentarios/”
4. Para crear un nuevo comentario, accedemos a “/crear_comentario/”.

3 Correcciones

3.1 Integración de Botones.

Para integrar los botones de actualizar y eliminar en la lista de publicaciones y comentarios, necesitamos realizar algunos cambios en las plantillas y en las vistas.

A continuación, describo los pasos necesarios para cada uno:

3.2 Para las Publicaciones:

Paso 1: Actualizar "lista_publicaciones.html" en el directorio "templates" de la aplicación "publicaciones". Agregar los enlaces para actualizar y eliminar cada publicación.

```
<!-- lista_publicaciones.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Publicaciones</title>
</head>
<body>
    <h1>Lista de Publicaciones</h1>
    <ul>
        {% for publicacion in publicaciones %}
        <li>
            <h2>{{ publicacion.titulo }}</h2>
            <p>{{ publicacion.contenido }}</p>
            <a href="{% url 'detalle_publicacion' pk=publicacion.pk %}">Ver detalles</a>
            <a href="{% url 'actualizar_publicacion' pk=publicacion.pk %}">Actualizar</a>
            <a href="{% url 'eliminar_publicacion' pk=publicacion.pk %}">Eliminar</a>
            <ul>
                {% for comentario in publicacion.comentarios %}
                <li>{{ comentario.texto }}</li>
                <a href="{% url 'actualizar_comentario' pk=comentario.pk %}">Actualizar</a
                <a href="{% url 'eliminar_comentario' pk=comentario.pk %}">Eliminar</a>
                {% endfor %}
            </ul>
        </li>
        {% endfor %}
    </ul>
</body>
```

```
</html>
```

Paso 2: Actualizar “views.py” en la aplicación “publicaciones”. Agregar las vistas para actualizar y eliminar las publicaciones.

```
# views.py
from django.shortcuts import render, get_object_or_404, redirect
from .forms import PublicacionForm, ComentarioForm
from .models import Publicacion

from django.shortcuts import render, redirect, get_object_or_404
from .forms import PublicacionForm, ComentarioForm
from .models import Publicacion, Comentario

def lista_publicaciones(request):
    publicaciones = Publicacion.objects.all()
    return render(request, 'lista_publicaciones.html', {'publicaciones': publicaciones})

def crear_publicacion(request):
    if request.method == 'POST':
        form = PublicacionForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_publicaciones')
    else:
        form = PublicacionForm()
    return render(request, 'crear_publicacion.html', {'form': form})

def detalle_publicacion(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    comentarios = Comentario.objects.filter(publicacion=publicacion)

    if request.method == 'POST':
        comentario_form = ComentarioForm(request.POST)
        if comentario_form.is_valid():
            comentario = comentario_form.save(commit=False)
            comentario.publicacion = publicacion
            comentario.save()
            return redirect('detalle_publicacion', pk=pk)
    else:
        comentario_form = ComentarioForm()
```

```
    context = {
        'publicacion': publicacion,
        'comentarios': comentarios,
        'comentario_form': comentario_form,
    }
    return render(request, 'detalle_publicacion.html', context)

def actualizar_publicacion(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    if request.method == 'POST':
        form = PublicacionForm(request.POST, instance=publicacion)
        if form.is_valid():
            form.save()
            return redirect('lista_publicaciones')
    else:
        form = PublicacionForm(instance=publicacion)
    return render(request, 'crear_publicacion.html', {'form': form})

def eliminar_publicacion(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    if request.method == 'POST':
        publicacion.delete()
        return redirect('lista_publicaciones')
    return render(request, 'eliminar_publicacion.html', {'publicacion': publicacion})
```

Paso 3: Actualizar "urls.py" en la aplicación "publicaciones". Agregar las URLs correspondientes para las vistas de actualizar y eliminar publicaciones.

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

urlpatterns = [
    # Rutas de Publicaciones
    path('lista/', views.lista_publicaciones, name='lista_publicaciones'),
    path('crear/', views.crear_publicacion, name='crear_publicacion'),
    path('detalle/<int:pk>/', views.detalle_publicacion, name='detalle_publicacion'),
    path('actualizar/<int:pk>/', views.actualizar_publicacion, name='actualizar_publicacio
    path('eliminar/<int:pk>/', views.eliminar_publicacion, name='eliminar_publicacion'),

    # Rutas de Comentarios
```

```
    path('<int:pk>/actualizar_comentario/', views.actualizar_comentario, name='actualizar_
    path('<int:pk>/eliminar_comentario/', views.eliminar_comentario, name='eliminar_coment
]
```

3.3 Para los Comentarios:

El proceso es similar al de las publicaciones, solo que debemos aplicarlo para los comentarios.

Paso 1: Actualizar "lista_publicaciones.html" en el directorio "templates" de la aplicación "publicaciones". Agregar los enlaces para actualizar y eliminar cada comentario.

```
<!-- lista_publicaciones.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Publicaciones</title>
</head>
<body>
    <h1>Lista de Publicaciones</h1>
    <ul>
        {% for publicacion in publicaciones %}
        <li>
            <h2>{{ publicacion.titulo }}</h2>
            <p>{{ publicacion.contenido }}</p>
            <a href="{% url 'detalle_publicacion' pk=publicacion.pk %}">Ver detalles</a>
            <a href="{% url 'actualizar_publicacion' pk=publicacion.pk %}">Actualizar</a>
            <a href="{% url 'eliminar_publicacion' pk=publicacion.pk %}">Eliminar</a>
            <ul>
                {% for comentario in publicacion.comentarios %}
                <li>{{ comentario.texto }}</li>
                <a href="{% url 'actualizar_comentario' pk=comentario.pk %}">Actualizar</a
                <a href="{% url 'eliminar_comentario' pk=comentario.pk %}">Eliminar</a>
                {% endfor %}
            </ul>
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Tambien es necesario la creación de un nuevo template llamado detalle_publicacion.html en el directorio "templates" de la aplicación "publicaciones". Agregar los enlaces para actualizar

y eliminar cada comentario.

```
<!-- detalle_publicacion.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Detalle de Publicación</title>
</head>
<body>
    <h1>{{ publicacion.titulo }}</h1>
    <p>{{ publicacion.contenido }}</p>
    <h3>Comentarios:</h3>
    <ul>
        {% for comentario in comentarios %}
        <li>{{ comentario.contenido }}</li>
        <a href="{% url 'actualizar_comentario' comentario.pk %}">Actualizar Comentario</a
        <a href="{% url 'eliminar_comentario' comentario.pk %}">Eliminar Comentario</a>
        {% endfor %}
    </ul>

    <a href="{% url 'actualizar_publicacion' publicacion.pk %}">Actualizar Publicación</a>
    <a href="{% url 'eliminar_publicacion' publicacion.pk %}">Eliminar Publicación</a>

    <!-- Agregar formulario para agregar comentario -->
    <h3>Agregar Comentario:</h3>
    <form method="post">
        {% csrf_token %}
        {{ comentario_form.as_p }}
        <button type="submit">Enviar Comentario</button>
    </form>
</body>
</html>
```

Y finalmente actualizar el archivo actualizar_comentario.html en el directorio “templates” de la aplicación “publicaciones”. Agregar los enlaces para actualizar y eliminar cada comentario.

```
<!-- actualizar_comentario.html -->
<!DOCTYPE html>
<head>
    <title>Actualizar Comentario</title>
</head>
<body>
```

```
    <h1>Actualizar Comentario</h1>
    <form method="post">
        {% csrf_token %}
        {{ comentario_form.as_p }}
        <button type="submit">Guardar Cambios</button>
    </form>
</body>
</html>
```

Paso 2: Actualizar "views.py" en la aplicación "publicaciones". Agregar las vistas para actualizar y eliminar los comentarios.

```
# views.py
from django.shortcuts import render, get_object_or_404, redirect
from .forms import PublicacionForm, ComentarioForm
from .models import Publicacion, Comentario

def lista_publicaciones(request):
    publicaciones = Publicacion.objects.all()
    return render(request, 'lista_publicaciones.html', {'publicaciones': publicaciones})

def crear_publicacion(request):
    # Código existente

def detalle_publicacion(request, pk):
    # Código existente

def actualizar_publicacion(request, pk):
    # Código existente

def eliminar_publicacion(request, pk):
    # Código existente

def crear_comentario(request):
    if request.method == 'POST':
        form = ComentarioForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_publicaciones')
    else:
        form = ComentarioForm()
    return render(request, 'crear_comentario.html', {'form': form})
```

```
def lista_comentarios(request):
    comentarios = Comentario.objects.all()
    return render(request, 'lista_comentarios.html', {'comentarios': comentarios})

def actualizar_comentario(request, pk):
    comentario = get_object_or_404(Comentario, pk=pk)

    if request.method == 'POST':
        comentario_form = ComentarioForm(request.POST, instance=comentario)
        if comentario_form.is_valid():
            comentario_form.save()
            return redirect('detalle_publicacion', pk=comentario.publicacion.pk)
    else:
        comentario_form = ComentarioForm(instance=comentario)

    context = {
        'comentario_form': comentario_form,
    }
    return render(request, 'actualizar_comentario.html', context)

def eliminar_comentario(request, pk):
    comentario = get_object_or_404(Comentario, pk=pk)
    publicacion_pk = comentario.publicacion.pk
    comentario.delete()
    return redirect('detalle_publicacion', pk=publicacion_pk)
```

Paso 3: Actualizar "urls.py" en la aplicación "publicaciones". Agregar las URLs correspondientes para las vistas de actualizar y eliminar comentarios.

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

urlpatterns = [
    # Rutas de Comentarios
    path('<int:pk>/actualizar_comentario/', views.actualizar_comentario, name='actualizar_
    path('<int:pk>/eliminar_comentario/', views.eliminar_comentario, name='eliminar_coment
]
```

3.4 Corregir el Modelo

Finalmente para corregir el modelo y hacer que el campo autor herede los datos de **User** de Django, podemos usar un campo **ForeignKey** que apunte al modelo User.

Esto nos permitirá asociar cada publicación y comentario a un usuario específico. Aquí está el código corregido:

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

class Publicacion(models.Model):
    titulo = models.CharField(max_length=200)
    contenido = models.TextField()
    fecha_publicacion = models.DateTimeField(auto_now_add=True)
    autor = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return self.titulo

class Comentario(models.Model):
    publicacion = models.ForeignKey(Publicacion, on_delete=models.CASCADE)
    autor = models.ForeignKey(User, on_delete=models.CASCADE)
    contenido = models.TextField()
    fecha_publicacion = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Comentario de {self.autor.username} en {self.publicacion.titulo}"
```

En este código, hemos modificado el campo autor en ambos modelos para que sea un **ForeignKey** que apunta al modelo **User de Django**.

Con esto, cada publicación y comentario estará asociado a un usuario registrado en el sistema.

El argumento **on_delete=models.CASCADE** en el campo autor de Comentario asegura que si un usuario es eliminado, también se eliminarán todos sus comentarios relacionados, pero ten en cuenta que esto es opcional y depende de la lógica de negocio que desees implementar.

3.5 Backend: Construcción de una API con Django Rest Framework

En esta sección, dividiremos todo lo que hemos realizado hasta ahora en una API utilizando Django Rest Framework (DRF). Esto nos permitirá exponer nuestros modelos (Publicacion y

Comentario) como puntos finales (endpoints) para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) a través de peticiones HTTP.

Paso 1: Instalar Django Rest Framework

Primero, debemos instalar Django Rest Framework en nuestro entorno virtual. Ejecuta el siguiente comando:

```
pip install djangorestframework
```

Paso 2: Configurar Django Rest Framework en el Proyecto

En el archivo settings.py del proyecto, agrega 'rest_framework' a la lista de aplicaciones instaladas:

```
INSTALLED_APPS = [
    # Otras aplicaciones...
    'rest_framework',
]
```

Paso 3: Serializadores

En DRF, los serializadores se utilizan para convertir nuestros modelos de Django en formatos JSON y viceversa. Vamos a crear los serializadores para los modelos Publicacion y Comentario en un archivo serializers.py dentro de la aplicación publicaciones.

```
# serializers.py
from rest_framework import serializers
from .models import Publicacion, Comentario

class ComentarioSerializer(serializers.ModelSerializer):
    class Meta:
        model = Comentario
        fields = '__all__'

class PublicacionSerializer(serializers.ModelSerializer):
    comentarios = ComentarioSerializer(many=True, read_only=True)

    class Meta:
        model = Publicacion
        fields = '__all__'
```

En este código, creamos dos serializadores, ComentarioSerializer y PublicacionSerializer, que utilizan el modelo correspondiente y definen los campos que queremos exponer en nuestra

API. En el caso de la publicación, utilizamos comentarios para mostrar los comentarios relacionados.

Paso 4: Vistas

Vamos a modificar nuestras vistas para utilizar los serializadores y convertir nuestros datos en formato JSON. En el archivo views.py de la aplicación publicaciones, actualiza el contenido de las vistas de la siguiente manera:

```
# views.py
from rest_framework import generics
from .models import Publicacion, Comentario
from .serializers import PublicacionSerializer, ComentarioSerializer

class ListaPublicaciones(generics.ListCreateAPIView):
    queryset = Publicacion.objects.all()
    serializer_class = PublicacionSerializer

class DetallePublicacion(generics.RetrieveUpdateDestroyAPIView):
    queryset = Publicacion.objects.all()
    serializer_class = PublicacionSerializer

class ListaComentarios(generics.ListCreateAPIView):
    queryset = Comentario.objects.all()
    serializer_class = ComentarioSerializer

class DetalleComentario(generics.RetrieveUpdateDestroyAPIView):
    queryset = Comentario.objects.all()
    serializer_class = ComentarioSerializer
```

Paso 5: URLs

Ahora, vamos a configurar las URLs de nuestra API en el archivo urls.py de la aplicación publicaciones.

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

urlpatterns = [
    path('publicaciones/', views.ListaPublicaciones.as_view(), name='lista_publicaciones')
    path('publicaciones/<int:pk>/', views.DetallePublicacion.as_view(), name='detalle_publ
    path('comentarios/', views.ListaComentarios.as_view(), name='lista_comentarios'),
```

```
    path('comentarios/<int:pk>/', views.DetalleComentario.as_view(), name='detalle_comenta
]
```

En este código, configuramos las URLs de nuestras vistas utilizando las vistas basadas en clases proporcionadas por DRF. Creamos puntos finales (endpoints) para listar, crear, ver, actualizar y eliminar publicaciones y comentarios.

¡Hemos construido una API básica para nuestro proyecto "Blog" utilizando Django Rest Framework!

Ahora podemos realizar operaciones CRUD a través de las peticiones HTTP en nuestros modelos de Publicacion y Comentario.

3.6 Documentación de las API

Para documentar las API, podemos utilizar la herramienta **drf-yasg**. La información la vamos a obtener de la documentación oficial en el siguiente link

Documentación de drf-yasg

Paso 1: Instalar drf-yasg

Primero, debemos instalar drf-yasg en nuestro entorno virtual. Ejecuta el siguiente comando:

```
pip install -U drf-yasg
```

Paso 2: Configurar drf-yasg en el Proyecto

En el archivo settings.py del proyecto, agrega 'drf_yasg' a la lista de aplicaciones instaladas:

```
INSTALLED_APPS = [
    # Otras aplicaciones...
    'drf_yasg',
]
```

Paso 3: Configurar drf-yasg en el archivo urls.py

En el archivo urls.py del proyecto, agrega las siguientes líneas de código:

```
# urls.py
...
from django.urls import re_path
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi
```

```
...

schema_view = get_schema_view(
    openapi.Info(
        title="Snippets API",
        default_version='v1',
        description="Test description",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('swagger<format>/', schema_view.without_ui(cache_timeout=0), name='schema-json'),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
    ...
]
```

Paso 4: Agregar la variable autenticacion_classes.

En el archivo url.py del proyecto agregar la variable authentication_classes para que no nos pida autenticación para poder ver la documentación de las API.

```
# urls.py
...
authentication_classes = []
...
```

Paso 5: Ejecutar el servidor

Ejecuta el servidor y en el navegador ingresa a http://localhost:8000/swagger o http://localhost:8000/redoc

```
python manage.py runserver
```

Tambien puedes probar los plugins de VSCode RappidApi Client o Thunder Client para poder ver la documentación de las API desde el editor de código.

En la siguiente sección, construiremos el frontend para consumir esta API.