Práctica 15: Formularios en Django

Lo último que haremos en nuestro website es crear un apartado para agregar y editar posts en el blog. Django `admin` está bien, pero es bastante difícil de personalizar y hacerlo bonito. Con `forms` tendremos un poder absoluto sobre nuestra interfaz.

Lo bueno de Django forms es que podemos definirlo desde cero o creando un `ModelForm` y se guardará el resultado del formulario en el modelo.

Esto es exactamente lo que queremos hacer: crearemos un formulario para nuestro modelo `Post`.

Como cada parte importante de Django, forms tienen su propio archivo: `forms.py`.

Tenemos que crear un archivo con este nombre en el directorio `blog`.

```
blog
└── forms.py
```

Ok, vamos a abrirlo y vamos a escribir el siguiente código:

```
from django import forms

from .models import Post

class PostForm(forms.ModelForm):

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

Primero necesitamos importar Django forms (`from django import forms`) y, obviamente, nuestro modelo `Post` (`from .models import Post`).

`PostForm`, como probablemente sospechas, es el nombre del formulario. Necesitamos decirle a Django que este formulario es un `ModelForm` (así Django hará algo de magia para nosotros) - `forms.ModelForm` es responsable de ello.

A continuación, tenemos `class Meta`, donde le decimos a Django qué modelo debe utilizar para crear este formulario (`model = Post`).

Finalmente, podemos decir que campo(s) queremos en nuestro formulario. En este escenario sólo queremos `title` y `text` - `author` será la persona que se ha autenticado (¡tú!) y `created_date` se definirá automáticamente cuando creemos un post (es decir en el código).

Y eso es todo. Todo lo que necesitamos hacer ahora es usar el formulario en una *view* y mostrarla en una plantilla.

Una vez más vamos a crear: un enlace a la página, una dirección URL, una vista y una plantilla.

Enlace a una página con el formulario

Es hora de abrir `blog/templates/blog/base.html`. Vamos a añadir un enlace en `div` llamado `page-header`:

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

Ten en cuenta que queremos llamar a nuestra nueva vista `post_new`. La clase `"glyphicon glyphicon-plus"` es proveida por el tema de bootstrap que estamnos usando, y mostrara un signo de mas (`+`) por nosotros.

Después de agregar la línea, tu archivo html debería tener este aspecto:

```
{% load staticfiles %}
<html>
    <head>
        <title>Django Daw 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⊂=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="/">Django Daw Blog</a></h1>
        </div>
        <div class="content container">
            <div class="row">
                <div class="col-md-8">
                    {% block content %}
                    {% endblock %}
                </div>
            </div>
        </div>
```

```
        </body>
    </html>
```

Luego de guardar y actualizar la página `http://127.0.0.1:8000` obviamente verás un error `NoReverseMatch` familiar.

URL

Abrimos `blog/urls.py` y añadimos una línea:

```
        url(r'^post/new/$', views.post_new, name='post_new'),
```

Y el código final tendrá este aspecto:

```
    from django.conf.urls import url
    from . import views

    urlpatterns = [
        url(r'^$', views.post_list),
        url(r'^post/(?P<pk>\d+)/$', views.post_detail, name='post_detail'),
        url(r'^post/new/$', views.post_new, name='post_new'),
    ]
```

Después de actualizar el sitio, veremos un `AttributeError`, puesto que no tenemos la vista `post_new` implementada. Vamos a añadirla ahora.

Vista post_new

Es el momento de abrir el archivo `blog/views.py` y agregar las siguientes líneas al resto de las filas `from`:

```
    from .forms import PostForm
```

y nuestra *vista*:

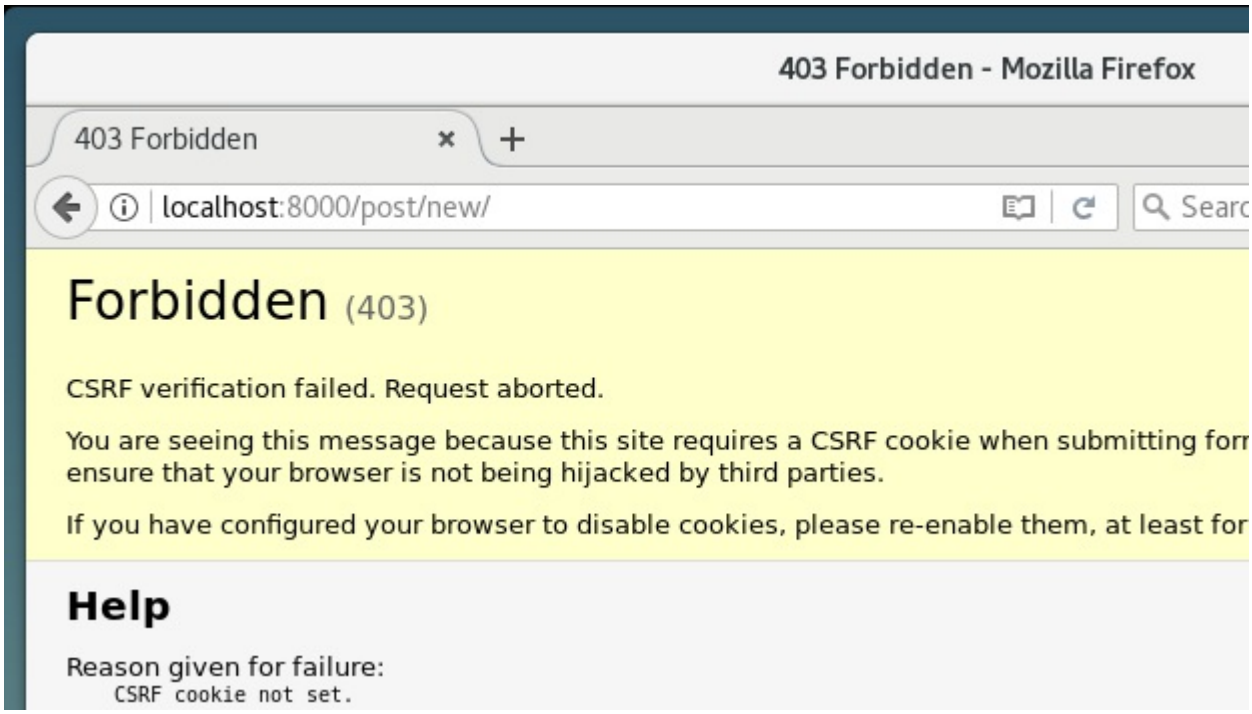
```
    def post_new(request):
        form = PostForm()
        return render(request, 'blog/post_edit.html', {'form': form})
```

Para crear un nuevo formulario `Post`, tenemos que llamar a `PostForm()` y pasarlo a la plantilla. Volveremos a esta *vista* pero, por ahora, vamos a crear rápidamente una plantilla (template) para el formulario.

Plantilla

Tenemos que crear un archivo `post_edit.html` en el directorio `blog/templates/blog`. Para hacer que un formulario funcione necesitamos varias cosas:

- tenemos que mostrar el formulario. Podemos hacerlo, por ejemplo, con un simple `{{ form.as_p }}`.
- la línea anterior tiene que estar dentro de una etiqueta de formulario HTML: `<form method="POST">...</form>`
- necesitamos un botón `Guardar`. Lo hacemos con un botón HTML: `<button type='submit'>Save</button>`
- y finalmente justo después de la apertura de `<form...>` necesitamos agregar `{% csrf_token %}`. Esto es muy importante ya que hace que tus formularios sean seguros, Django se quejará si te olvidas de esta parte cuando intentes guardar el formulario.

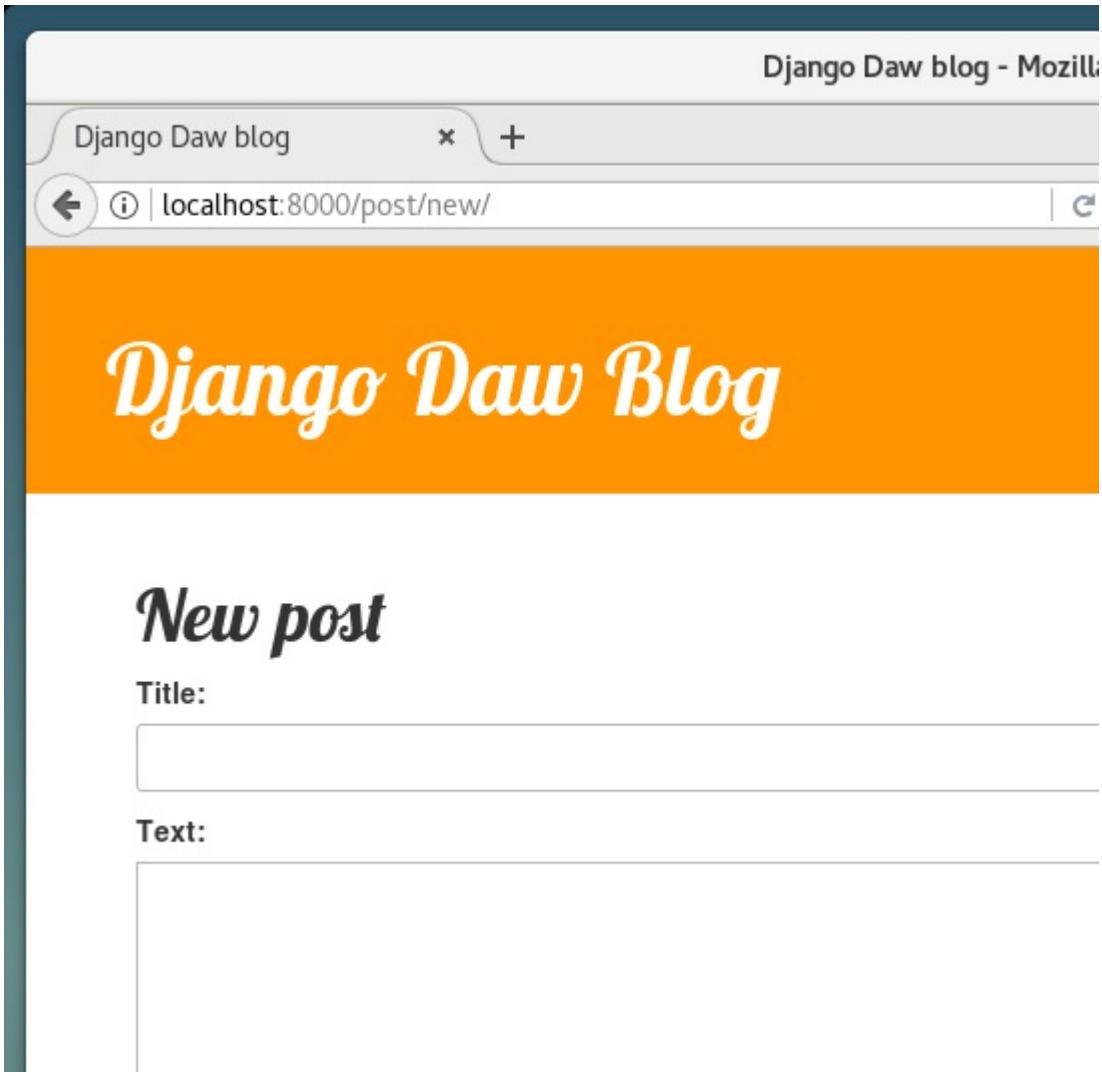


Bueno, vamos a ver cómo quedará el HTML en `post_edit.html`:

```
{% extends 'blog/base.html' %}

{% block content %}
    <h1>New post</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>
{% endblock %}
```

Es hora de actualizar. Si, tu formulario se muestra!.



Un momento. Si escribes algo en los campos `title` y `text` y tratas de guardar los cambios - ¿qué pasará?

Nada!. Una vez más estamos en la misma página y el texto se ha ido... no se añade ningún post nuevo. Entonces, ¿Qué ha ido mal?

La respuesta es: nada. Tenemos que trabajar un poco más en nuestra *vista*.

Guardar el formulario

Abre `blog/views.py` una vez más. Actualmente, lo que tenemos en la vista `post_new` es:

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

Cuando enviamos el formulario somos redirigidos a la misma vista, pero esta vez tenemos algunos datos adicionales en `request`, más específicamente en `request.POST` (el nombre no tiene nada que ver con un post del blog, se refiere a que estamos "publicando" -en inglés, posting- datos). ¿Recuerdas que en el archivo HTML la definición de `<form>` tenía la variable `method="POST"` ? Todos los campos del formulario están ahora en `request.POST`. No deberías renombrar la variable `POST` (el único nombre que también es válido para la variable `method` es `GET`, pero no tenemos tiempo para explicar cuál es la diferencia).

En nuestra *view* tenemos dos posibles situaciones a contemplar. Primero: cuando accedemos a la página por primera vez y queremos un formulario en blanco. Segundo: cuando volvemos a la *view* con los datos del formulario que acabamos de escribir. Así que tenemos que añadir una condición (utilizaremos `if` para eso).

```
    if request.method == "POST":
        [...]
    else:
        form = PostForm()
```

Es hora de llenar los puntos `[...]`. Si el `method` es `POST` queremos construir el `PostForm` con los datos del formulario, lo haremos con:

```
    form = PostForm(request.POST)
```

Fácil! Lo siguiente es verificar si el formulario es correcto (todos los campos necesarios están definidos y no hay valores incorrectos). Lo hacemos con

`form.is_valid()`.

Comprobamos que el formulario es válido y, si es así, ¡lo podemos salvar!

```
    if form.is_valid():
        post = form.save(commit=False)
        post.author = request.user
        post.published_date = timezone.now()
        post.save()
```

Básicamente, tenemos que hacer dos cosas aquí: guardamos el formulario con `form.save` y añadimos un autor (ya que no había ningún campo de `author` en el `PostForm` y este campo es obligatorio). `commit=False` significa que no queremos guardar el modelo `Post` todavía - queremos añadir el autor primero. La mayoría de las veces utilizarás `form.save()`, sin `commit=False`, pero en este caso, tenemos que hacerlo. `post.save()` conservará los cambios (añadiendo el autor) y se creará una nuevo post en el blog.

Por último, sería genial si podemos inmediatamente ir a la página `post_detail` del nuevo post de blog, para hacerlo necesitamos importar algo más:

```
    from django.shortcuts import redirect
```

Agrégalo al principio del archivo. Y ahora podemos decir: vé a la página `post_detail` del post recién creado.

```
    return redirect('post_detail', pk=post.pk)
```

`post_detail` es el nombre de la vista a la que queremos ir. Recuerda que esta *view* requiere una variable `pk`. Para pasarlo a las vistas utilizamos `pk=post.pk`, donde `post` es el post recién creado.

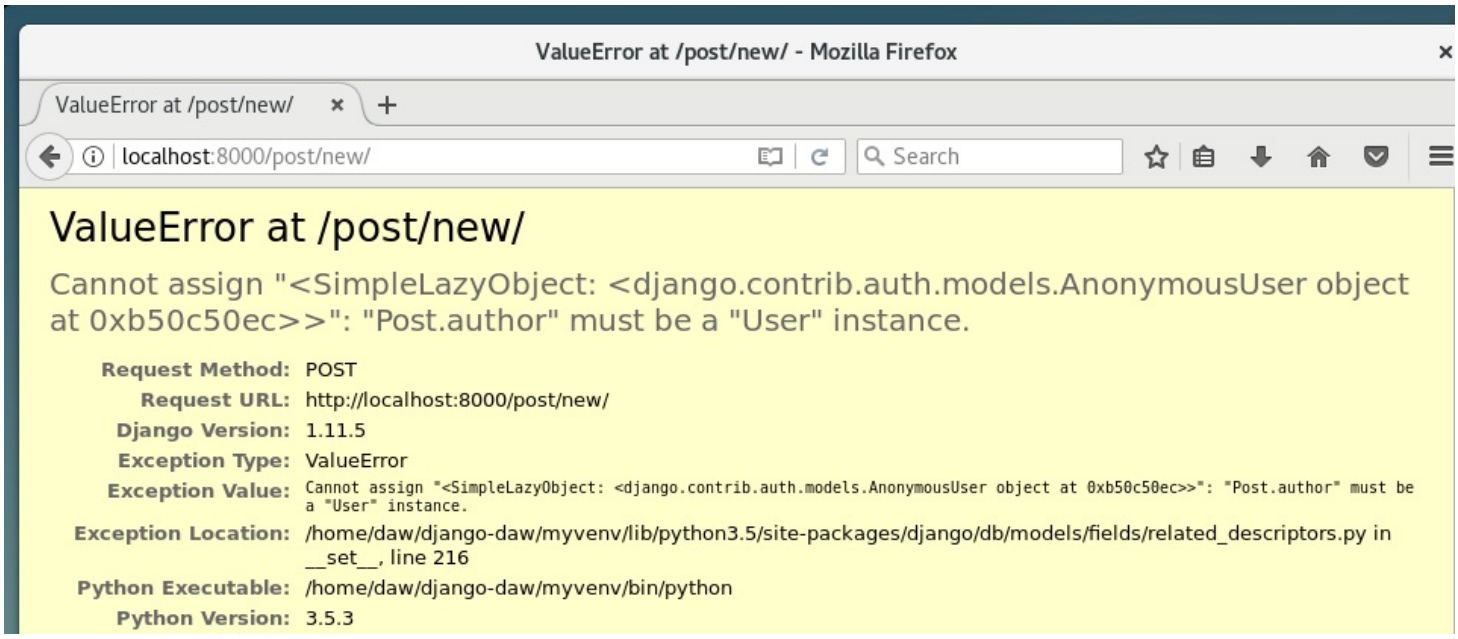
Bien, hablamos mucho, pero probablemente queremos ver como se ve ahora la *vista*:

```
    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})
```

Vamos a ver si funciona. Ve a la página http://127.0.0.1:8000/post/new/, añade un `title` y un `text`, guardalo... ¡y voilà! Se añade el nuevo post al blog y se nos redirige a la página de `post_detail`.

Probablemente has visto que no hemos definido la fecha de publicación. Vamos a introducir un *botón publicar* en otra práctica ma adelante.

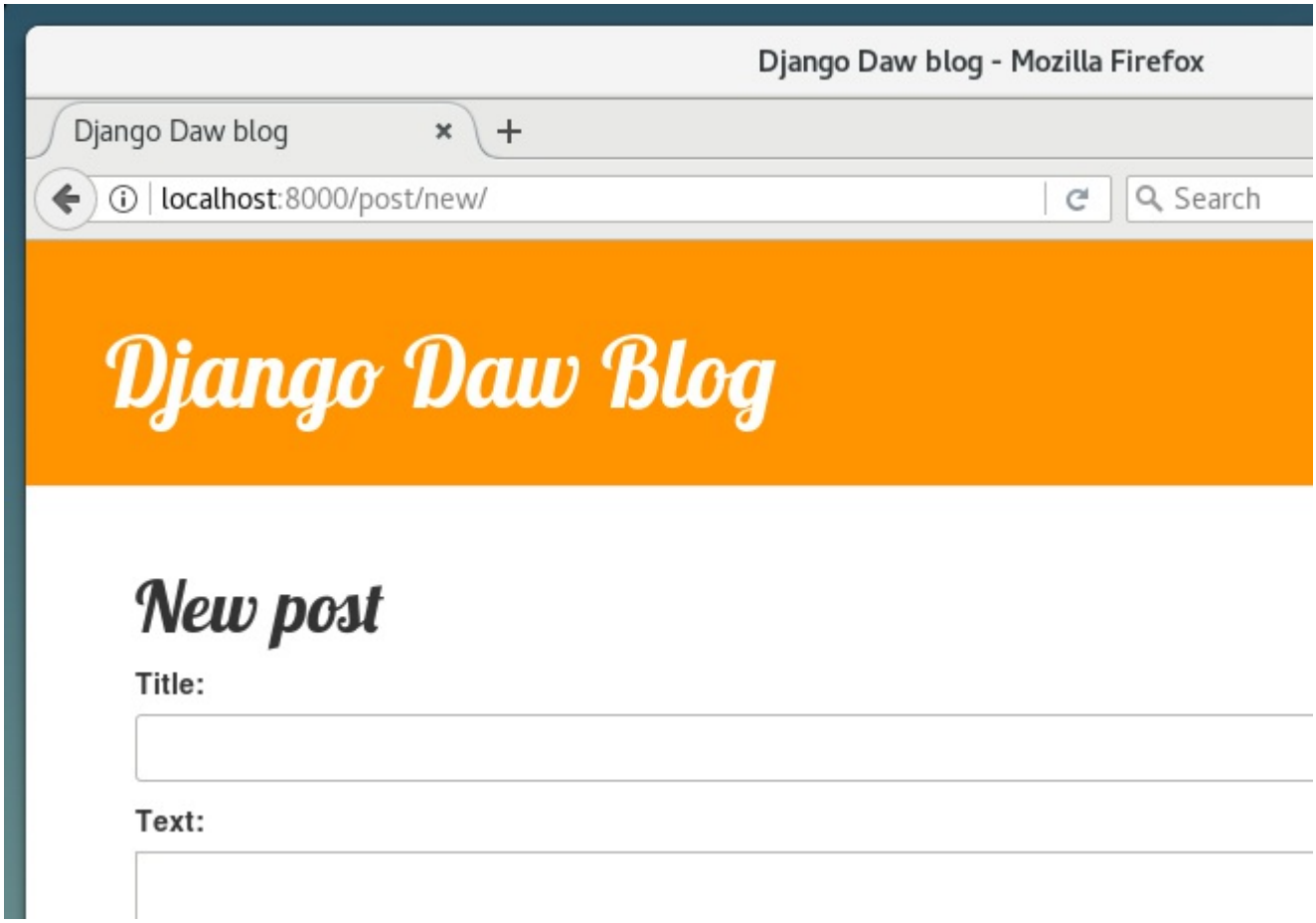
> Como recientemente hemos utilizado la interfaz de administrador de Django, el sistema piensa que hemos iniciado sesión. Hay algunas situaciones que podrían llevarnos a desconectarnos (cerrando el navegador, reiniciando la base de datos, etc.). Si estás recibiendo errores al crear un post que indican la falta de inicio de sesión de usuario, dirígete a la página de administración `http://127.0.0.1:8000/admin` e inicia sesión nuevamente. Esto resolverá el problema temporalmente. Hay un arreglo permanente en una sección más adelante.



Validación de formularios

Ahora, veamos qué tan bueno es Django forms. Un post del blog debe tener los campos `title` y `text`. En nuestro modelo `Post` no dijimos (a diferencia de `published_date`) que estos campos son requeridos, así que Django, por defecto, espera que estén definidos.

Trata de guardar el formulario sin `title` y `text`. Adivina qué pasará!.



Django se encarga de validar que todos los campos en el formulario estén correctos.

Formulario de edición

Ahora sabemos cómo agregar un nuevo formulario. Pero, ¿Qué pasa si queremos editar uno existente? Es muy similar a lo que acabamos de hacer. Vamos a crear

algunas cosas importantes rápidamente (si no entiendes algo, pregunta o revisa lo prácticas anteriores, son temas que ya se han cubierto).

Abre el archivo `blog/templates/blog/post_detail.html` y añade esta línea:

```
    <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

para que la plantilla quede:

```
{% extends 'blog/base.html' %}

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
        <h1>{{ post.title }}</h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

En el archivo `blog/urls.py` añadimos esta línea:

```
    url(r'^post/(?P<pk>\d+)/edit/$', views.post_edit, name='post_edit'),
```

Vamos a reusar la plantilla `blog/templates/blog/post_edit.html`, así que lo último que nos falta es una *view*.

Abramos el archivo `blog/views.py` y añadamos al final esta línea:

```
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})
```

Esto se ve casi exactamente igual a nuestra view `post_new`, ¿no? Pero no del todo. Primero: pasamos un parámetro extra `pk` de los urls. Luego: obtenemos el modelo `Post` que queremos editar con `get_object_or_404(Post, pk=pk)` y después, al crear el formulario pasamos este post como una `instancia` tanto al guardar el formulario:

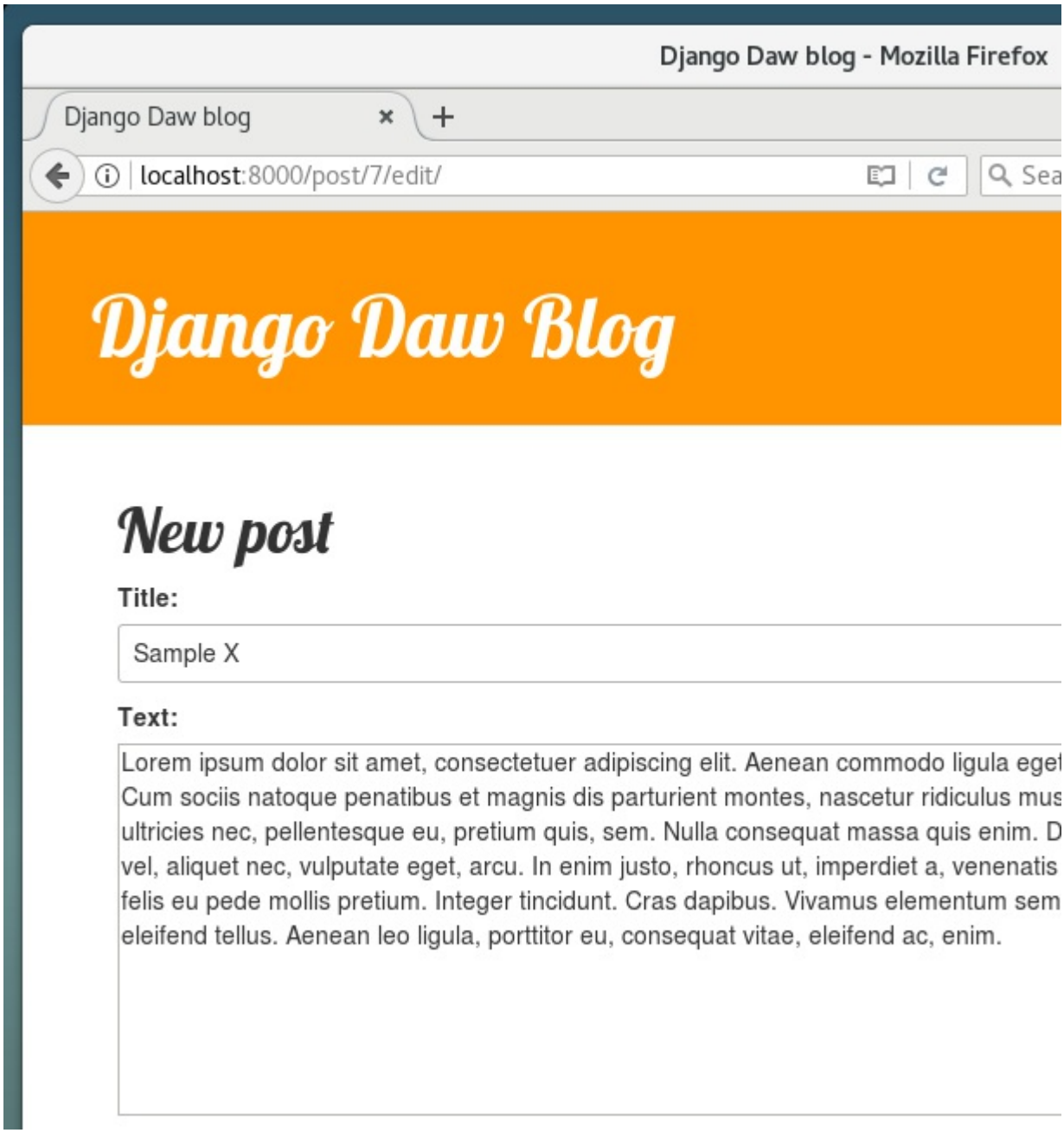
```
    form = PostForm(request.POST, instance=post)
```

como al abrir un formulario con este post para editarlo:

```
    form = PostForm(instance=post)
```

Dirígete a la página `post_detail`. Debe haber ahí un botón para editar en la esquina superior derecha:

Al dar click ahí, debes ver el formulario con nuestro post del blog:



Siéntete libre de cambiar el título o el texto y guarda los cambios.

Si necesitas más información sobre los formularios de Django, debes leer la documentación: https://docs.djangoproject.com/en/1.11/topics/forms/

Seguridad

Ser capaz de crear nuevos post solo con presionar un botón esta de lujo. Pero ahora, cualquiera que visite tu sitio será capaz de crear nuevos posts, y eso probablemente es algo que no deseas. Cambiemos esto para que el botón solo aparezca para ti y para nadie más.

En `blog/templates/blog/base.html`, encuentra nuestro `page-header` `div` y la etiqueta ancla que pusiste ahí antes. Se debe de ver algo así:

```
    <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

Agregaremos otra etiqueta `{% if %}` a esto, lo cual hará el enlace aparecer solo para usuarios que hayan iniciado sesión como administradores — en este momento, solo tú. Cambia la etiqueta `<a>` para que se vea así:

```
{% if user.is_authenticated %}
    <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
{% endif %}
```

Este `{% if %}` hará que el enlace sea enviado al navegador solo si el usuario ha iniciado sesión. Esto no protege contra la creación de posts completamente, pero es un buen primer paso. Se vera mas sobre seguridad en una práctica (opcional) posterior.

Agregaremos este cambio al icono de edición que creamos una sección atrás, así otras personas no serán capaces de editar posts existentes.

Abre `blog/templates/blog/post_detail.html` y encuentra está línea:

```
    <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

cambiala a esto:

```
    {% if user.is_authenticated %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
    {% endif %}
```

Ya que seguramente haz iniciado sesión, actualiza la pagina, no veras nada diferente. Carga la pagina en un navegador diferente o una ventana de incognito, veras que el enlace no se muestra y que el icono tampoco aparece.

Una cosa más: ¡Tiempo de implementación!

Veamos si todo esto funciona en PythonAnywhere. Tiempo de hacer otro despliegue.

- Primero, haz un commit con tu nuevo código y súbelo a GitHub:

```
$ git status
$ git add --all .
$ git status
$ git commit -m "Added views to create/edit blog post inside the site."
$ git push
```

- Luego, en una consola Bash de PythonAnywhere

```
$ cd ~/<tu-pythonanywhere-username>.pythonanywhere.com
$ git pull
[...]
```

- Finalmente, ve a la pestaña Web y haz click en **Reload**.

Y eso debería ser todo. Felicidades!!