Práctica Opcional #1:Añadiendo mas funcionalidad

Agregar mas a tu sitio web

Nuestro blog ha recorrido un gran camino hasta aquí, pero aún hay espacio para la mejora. Lo siguiente, vamos a agregar nuevas características para borradores y su publicación. También vamos a agregar eliminación de post que ya no queremos mas.

Guarda nuevos post como borradores

Actualmente, nosotros creamos nuevos post usando nuestro formulario *New post* y el post es publicado directamente. En lugar de publicar el post vamos a guardarlo como borrador, **elimina** esta línea en `blog/views.py` en los métodos `post_new` y `post_edit`:

```
post.published_date = timezone.now()
```

De esta manera los nuevos post serán guardados como borradores y se podrán revisar después en vez de ser publicados instantáneamente. Todo lo que necesitamos es una manera de listar los borradores.

Página con los post no publicados

El la práctica 12 vimos los *querysets*, posteriormente en la práctica 13 creamos una vista `post_list` que muestra solamente los post publicados (aquellos que tienen un `publication_date` no vacío).

Es tiempo de hacer algo similiar, pero con borradores.

Vamos a añadir un enlace en `blog/templates/blog/base.html` en el encabezado. No queremos mostrar nuestro borradores a todo el mundo, entonces vamos a colocarlo dentro de la verificación `{% if user.is_authenticated %}`, justo después del botón de agregar posts.

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

Ahora las URLs: en `blog/urls.py` vamos a agregar:

```
url(r'^drafts/$', views.post_draft_list, name='post_draft_list'),
```

Tiempo de crear una nueva vista en `blog/views.py`:

```
def post_draft_list(request):
    posts = Post.objects.filter(published_date__isnull=True).order_by('created_date')
    return render(request, 'blog/post_draft_list.html', {'posts': posts})
```

Esta línea `posts = Post.objects.filter(published_date__isnull=True).order_by('created_date')` se asegura de que solamente vamos a tomar post no publicados (`published_date__isnull=True`) y los ordena por `created_date` (`order_by('created_date')`).

Ok, el último paso es el template. Crea un archivo `blog/templates/blog/post_draft_list.html` y agrega lo siguiente:

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

{% block content %}
    {% for post in posts %}
        <div class="post">
            <p class="date">created: {{ post.created_date|date:'d-m-Y' }}</p>
            <h1><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a></h1>
            <p>{{ post.text|truncatechars:200 }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

Se ve muy similar a nuestro `post_list.html`.

Ahora cuando vayas a `http://127.0.0.1:8000/drafts/` vas a ver la lista de post no publicados.

Agrega un botón de publicación

Sería bueno tener un botón en el detalle del post que publique el post inmediatamente, para esto vamos a abrir `blog/templates/blog/post_detail.html` y cambiar estas líneas:

```
{% if post.published_date %}
    <div class="date">
        {{ post.published_date }}
    </div>
```

```
{% endif %}
```

por estas:

```
{% if post.published_date %}
    <div class="date">
        {{ post.published_date }}
    </div>
{% else %}
    <a class="btn btn-default" href="{% url 'post_publish' pk=post.pk %}">Publish</a>
{% endif %}
```

Como puedes ver, hemos agregado la línea `{% else %}`. Esto significa, que la condición de `{% if post.published_date %}` no es cumplida (entonces no hay `published_date`), entonces queremos agregar la línea `<a class="btn btn-default" href="{% url 'post_publish' pk=post.pk %}">Publish</a>`. Nota que estamos pasando la variable `pk` en el `{% url %}`.

Tiempo de crear una URL (en `blog/urls.py`):

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

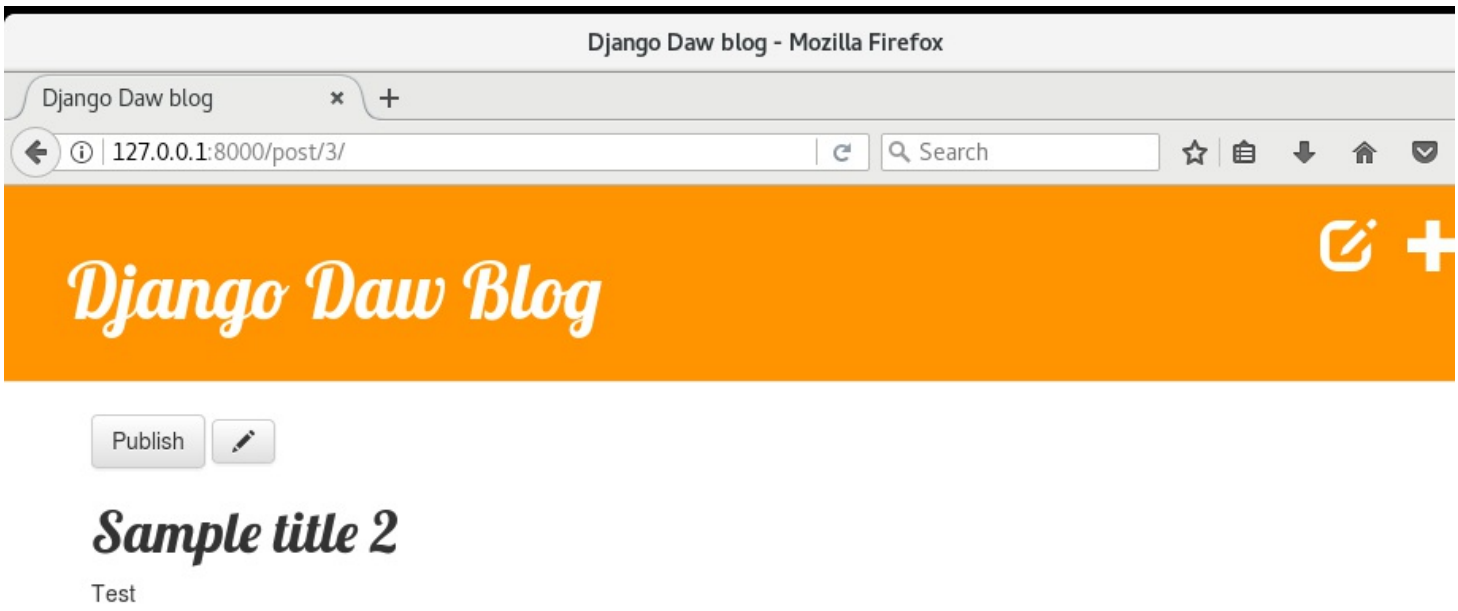
Y finalmente una vista (como siempre, en `blog/views.py`):

```
def post_publish(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.publish()
    return redirect('post_detail', pk=pk)
```

Recuerda, cuando creamos el modelo `Post` escribimos un método `publish`. Se veía como esto:

```
  def publish(self):
      self.published_date = timezone.now()
      self.save()
```

Ahora finalmente podemos usarlo. Y de nuevo al publicar el post, somos redirigidos inmediatamente a la página `post_detail`.



Eliminar post

Vamos a abrir `blog/templates/blog/post_detail.html` de nuevo y vamos a añadir esta línea

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

Justo debajo de la línea con el botón editar.

Ahora necesitamos una URL (`blog/urls.py`):

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

Ahora, abre `blog/views.py` y agrega este código:

```
def post_remove(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.delete()
    return redirect('post_list')
```

La única cosa nueva es en realidad, eliminar el post. Cada modelo en Django puede ser eliminado con `.delete()`.

Y en este momento, después de eliminar un post, queremos ir a la página web con la lista de posts, por eso usamos el `redirect`.

Probémoslo, ve a la página con los post e intentemos eliminarlo.