### Añadir nuevos temas

Empecemos por permitir a los usuarios añadir un nuevo tema. Añadir una página basada en un formulario funciona de forma muy parecida a añadir las páginas que ya hemos construido: definimos una URL, escribimos una función de vista y escribimos una plantilla. La única diferencia significativa es la adición de un nuevo módulo llamado *forms.py*, que contendrá los formularios.

#### El Tema ModeloFormulario

Cualquier página que permita a un usuario introducir y enviar información en una página web implica un elemento HTML llamado *form*. Cuando los usuarios introducen información, necesitamos *validate* que la información proporcionada sea el tipo correcto de datos y no sea maliciosa, como código diseñado para interrumpir nuestro servidor. A continuación, necesitamos procesar y guardar la información válida en el lugar adecuado de la base de datos. Django automatiza gran parte de este trabajo.

La forma más sencilla de construir un formulario en Django es utilizar un ModelForm, que utiliza la información de los modelos que definimos en el Capítulo 18 para construir un formulario automáticamente. Escribe tu primer formulario en el archivo *forms.py*, que debe crearse en el mismo directorio que *models.py*:

**forms.py**

from django import forms  
  
from .models import Topic  
  
❶ class TopicForm(forms.ModelForm):  
 class Meta:  
❷ model = Topic  
❸ fields = ['text']  
❹ labels = {'text': ''}

Primero importamos el módulo forms y el modelo con el que trabajaremos, Topic. A continuación, definimos una clase llamada TopicForm, que hereda de forms.ModelForm ❶.

La versión más sencilla de ModelForm consiste en una clase anidada Meta que indica a Django en qué modelo debe basarse el formulario y qué campos debe incluir. Aquí especificamos que el formulario debe basarse en el modelo Topic ❷, y que sólo debe incluir el campo text ❸. La cadena vacía en el diccionario de etiquetas indica a Django que no genere una etiqueta para el campo text ❹.

#### La URL del nuevo\_tema

La URL de una nueva página debe ser corta y descriptiva. Cuando el usuario quiera añadir un nuevo tema, le enviaremos a *http://localhost:8000/new\_topic/*. Éste es el patrón de URL para la página new\_topic; añádelo a *learning\_logs/urls.py*:

**learning\_logs/urls.py**

--snip--  
urlpatterns = [  
 --snip--  
 # Page for adding a new topic.  
 path('new\_topic/', views.new\_topic, name='new\_topic'),  
]

Este patrón de URL envía peticiones a la función de vista new\_topic(), que escribiremos a continuación.

#### La función de vista new\_topic()

La función new\_topic() debe gestionar dos situaciones diferentes: las solicitudes iniciales de la página new\_topic, en cuyo caso debe mostrar un formulario en blanco; y el procesamiento de los datos enviados en el formulario. Una vez procesados los datos de un formulario enviado, debe redirigir al usuario de nuevo a la página topics:

**views.py**

from django.shortcuts import render, redirect  
  
from .models import Topic  
from .forms import TopicForm  
  
--snip--  
def new\_topic(request):  
 """Add a new topic."""  
❶ if request.method != 'POST':  
 # No data submitted; create a blank form.  
❷ form = TopicForm()  
 else:  
 # POST data submitted; process data.  
❸ form = TopicForm(data=request.POST)  
❹ if form.is\_valid():  
❺ form.save()  
❻ return redirect('learning\_logs:topics')  
  
 # Display a blank or invalid form.  
❼ context = {'form': form}  
 return render(request, 'learning\_logs/new\_topic.html', context)

Importamos la función redirect, que utilizaremos para redirigir al usuario de vuelta a la página topics después de que envíe su tema. También importamos el formulario que acabamos de escribir, TopicForm.

#### Peticiones GET y POST

Los dos tipos principales de peticiones que utilizarás cuando construyas aplicaciones son GET y POST. Las peticiones *GET* se utilizan para páginas que sólo leen datos del servidor. Normalmente utilizas las peticiones *POST* cuando el usuario necesita enviar información a través de un formulario. Especificaremos el método POST para procesar todos nuestros formularios. (Existen algunos otros tipos de peticiones, pero no los utilizaremos en este proyecto).

La función new\_topic() toma como parámetro el objeto request. Cuando el usuario solicite inicialmente esta página, su navegador enviará una solicitud GET. Una vez que el usuario haya rellenado y enviado el formulario, su navegador enviará una solicitud POST. Dependiendo de la petición, sabremos si el usuario está solicitando un formulario en blanco (GET) o pidiéndonos que procesemos un formulario cumplimentado (POST).

Utilizamos una prueba if para determinar si el método de solicitud es GET o POST ❶. Si el método de solicitud no es POST, probablemente la solicitud sea GET, por lo que tenemos que devolver un formulario en blanco. (Si es otro tipo de solicitud, sigue siendo seguro devolver un formulario en blanco.) Creamos una instancia de TopicForm ❷, la asignamos a la variable form, y enviamos el formulario a la plantilla del diccionario context ❼. Como no incluimos argumentos al instanciar TopicForm, Django crea un formulario en blanco que el usuario puede rellenar.

Si el método de solicitud es POST, el bloque else se ejecuta y procesa los datos enviados en el formulario. Creamos una instancia de TopicForm ❸ y le pasamos los datos introducidos por el usuario, que se asignan a request.POST. El objeto form que se devuelve contiene la información enviada por el usuario.

No podemos guardar la información enviada en la base de datos hasta que hayamos comprobado que es válida ❹. El método is\_valid() comprueba que se hayan rellenado todos los campos obligatorios (todos los campos de un formulario son obligatorios por defecto) y que los datos introducidos coincidan con los tipos de campo esperados; por ejemplo, que la longitud de text sea inferior a 200 caracteres, como especificamos en *models.py* en el Capítulo 18. Esta validación automática nos ahorra mucho trabajo. Si todo es válido, podemos llamar a save() ❺, que escribe los datos del formulario en la base de datos.

Una vez guardados los datos, podemos abandonar esta página. La función redirect() toma el nombre de una vista y redirige al usuario a la página asociada a esa vista. Aquí utilizamos redirect() para redirigir el navegador del usuario a la página topics ❻, donde el usuario debería ver el tema que acaba de introducir en la lista de temas.

La variable context se define al final de la función de vista, y la página se renderiza utilizando la plantilla *new\_topic.html*, que crearemos a continuación. Este código se coloca fuera de cualquier bloque if; se ejecutará si se ha creado un formulario en blanco, y se ejecutará si se determina que un formulario enviado no es válido. Un formulario inválido incluirá algunos mensajes de error por defecto para ayudar al usuario a enviar datos aceptables.

#### La plantilla new\_topic

Ahora crearemos una nueva plantilla llamada *new\_topic.html* para mostrar el formulario que acabamos de crear:

**new\_topic.html**

{% extends "learning\_logs/base.html" %}  
  
{% block content %}  
 <p>Add a new topic:</p>  
  
❶ <form action="{% url 'learning\_logs:new\_topic' %}" method='post'>  
❷ {% csrf\_token %}  
❸ {{ form.as\_div }}  
❹ <button name="submit">Add topic</button>  
 </form>  
  
{% endblock content %}

Esta plantilla extiende *base.html*, por lo que tiene la misma estructura base que el resto de páginas del Registro de Aprendizaje. Utilizamos las etiquetas </form> para definir un formulario HTML ❶. El argumento action indica al navegador dónde enviar los datos enviados en el formulario; en este caso, los enviamos a la función de vista new\_topic(). El argumento method indica al navegador que envíe los datos como una petición POST.

Django utiliza la etiqueta de plantilla {% csrf\_token %} ❷ para evitar que los atacantes utilicen el formulario para obtener acceso no autorizado al servidor. (Este tipo de ataque se denomina *cross-site request forgery*.) A continuación, mostramos el formulario; aquí puedes ver lo sencillas que Django puede hacer ciertas tareas, como mostrar un formulario. Sólo tenemos que incluir la variable de plantilla {{ form.as\_div }} para que Django cree automáticamente todos los campos necesarios para mostrar el formulario ❸. El modificador as\_div indica a Django que muestre todos los elementos del formulario como elementos HTML </div>; ésta es una forma sencilla de mostrar el formulario de forma ordenada.

Django no crea un botón de envío para los formularios, así que definimos uno antes de cerrar el formulario ❹.

#### Enlace a la página nuevo\_tema

A continuación, incluimos un enlace a la página new\_topic en la página topics:

**topics.html**

{% extends "learning\_logs/base.html" %}  
  
{% block content %}  
  
 <p>Topics</p>  
  
 <ul>  
 --snip--  
 </ul>  
  
 <a href="{% url 'learning\_logs:new\_topic' %}">Add a new topic</a>  
  
{% endblock content %}

Coloca el enlace después de la lista de temas existentes. [La Figura 19-1](#figure19-1) muestra el formulario resultante; prueba a utilizarlo para añadir algunos temas nuevos por tu cuenta.



Figura 19-1: Página para añadir un nuevo

tema

[anterior](c19_2.html)[Subtema 3 de 23: (Ver todo)](c19.html)[siguiente](c19_4.html)