# 18 Introducción a Django



A medida que Internet ha evolucionado, la línea que separa los sitios web de las aplicaciones móviles se ha difuminado. Tanto los sitios web como las aplicaciones ayudan a los usuarios a interactuar con los datos de diversas formas. Afortunadamente, puedes utilizar Django para construir un único proyecto que sirva tanto para un sitio web dinámico como para un conjunto de aplicaciones móviles. *Django* es el más popular de Python *web framework*, un conjunto de herramientas diseñadas para construir aplicaciones web interactivas. En este capítulo, aprenderás a utilizar Django para construir un proyecto llamado Learning Log, un sistema de diario online que te permite llevar un registro de la información que has aprendido sobre diferentes temas.

Escribiremos una especificación para este proyecto y, a continuación, definiremos modelos para los datos con los que trabajará la aplicación. Utilizaremos el sistema de administración de Django para introducir algunos datos iniciales, y luego escribiremos vistas y plantillas para que Django pueda construir las páginas del sitio.

Django puede responder a peticiones de páginas y facilitar la lectura y escritura en una base de datos, gestionar usuarios y mucho más. En los Capítulos 19 y 20, perfeccionarás el proyecto Learning Log, y luego lo desplegarás en un servidor activo para que tú (y todo el mundo) podáis utilizarlo.

## Configurar un proyecto

Cuando empiezas a trabajar en algo tan importante como una aplicación web, primero tienes que describir los objetivos del proyecto en una especificación, o *spec*. Una vez que tengas un conjunto claro de objetivos, puedes empezar a identificar tareas manejables para alcanzar esos objetivos.

En esta sección, escribiremos una especificación para Learning Log y empezaremos a trabajar en la primera fase del proyecto. Esto implicará configurar un entorno virtual y construir los aspectos iniciales de un proyecto Django.

### Escribir una especificación

Una especificación completa detalla los objetivos del proyecto, describe su funcionalidad y habla de su aspecto e interfaz de usuario. Como cualquier buen proyecto o plan de empresa, una especificación debe mantenerte centrado y ayudarte a mantener tu proyecto en el buen camino. No escribiremos aquí una especificación completa del proyecto, pero estableceremos algunos objetivos claros para mantener centrado el proceso de desarrollo. Estas son las especificaciones que utilizaremos:

Escribiremos una aplicación web llamada Learning Log que permita a los usuarios registrar los temas que les interesan y hacer anotaciones en el diario a medida que aprenden sobre cada tema. La página de inicio de Learning Log describirá el sitio e invitará a los usuarios a registrarse o a iniciar sesión. Una vez conectado, un usuario puede crear nuevos temas, añadir nuevas entradas y leer y editar las entradas existentes.

Cuando investigas un tema nuevo, llevar un diario de lo que has aprendido puede ayudarte a hacer un seguimiento de la información nueva y de la que ya has encontrado. Esto es especialmente cierto cuando se estudian temas técnicos. Una buena aplicación, como la que vamos a crear, puede ayudar a que este proceso sea más eficaz.

### Crear un entorno virtual

Para trabajar con Django, primero configuraremos un entorno virtual. Un *virtual environment* es un lugar de tu sistema donde puedes instalar paquetes y aislarlos de todos los demás paquetes de Python. Separar las bibliotecas de un proyecto de las de otros proyectos es beneficioso y será necesario cuando despleguemos Learning Log en un servidor en el Capítulo 20.

Crea un nuevo directorio para tu proyecto llamado *learning\_log*, cambia a ese directorio en un terminal, e introduce el siguiente código para crear un entorno virtual:

learning\_log$ python -m venv ll\_env  
learning\_log$

Aquí estamos ejecutando el módulo de entorno virtual venv y utilizándolo para crear un entorno llamado *ll\_env* (ten en cuenta que este nombre empieza por dos s minúsculas *L*, no dos unos). Si utilizas un comando como python3 al ejecutar programas o instalar paquetes, asegúrate de utilizar ese comando aquí.

### Activar el Entorno Virtual

Ahora tenemos que activar el entorno virtual, utilizando el siguiente comando:

learning\_log$ source ll\_env/bin/activate  
(ll\_env)learning\_log$

Este comando ejecuta el script *activate* en *ll\_env/bin/*. Cuando el entorno esté activo, verás el nombre del entorno entre paréntesis. Esto indica que puedes instalar nuevos paquetes en el entorno y utilizar paquetes ya instalados. Los paquetes que instales en *ll\_env* no estarán disponibles cuando el entorno esté inactivo.

## Nota

Si utilizas Windows, utiliza el comando ll\_env\Scripts\activate (sin la palabra source) para activar el entorno virtual. Si utilizas PowerShell, es posible que tengas que escribir Activate en mayúsculas.

Para dejar de utilizar un entorno virtual, introduce deactivate:

(ll\_env)learning\_log$ deactivate  
learning\_log$

El entorno también quedará inactivo cuando cierres el terminal en el que se está ejecutando.

### Instalar Django

Con el entorno virtual activado, introduce lo siguiente para actualizar pip e instalar Django:

(ll\_env)learning\_log$ pip install --upgrade pip  
(ll\_env)learning\_log$ pip install django  
Collecting django  
--snip--  
Installing collected packages: sqlparse, asgiref, django  
Successfully installed asgiref-3.5.2 django-4.1 sqlparse-0.4.2  
(ll\_env)learning\_log$

Dado que descarga recursos de diversas fuentes, pip se actualiza con bastante frecuencia. Es una buena idea actualizar pip siempre que crees un nuevo entorno virtual.

Ahora estamos trabajando en un entorno virtual, por lo que el comando para instalar Django es el mismo en todos los sistemas. No es necesario utilizar comandos más largos, como python -m pip install nombre\_paquete, ni incluir la bandera --user. Ten en cuenta que Django sólo estará disponible cuando el entorno *ll\_env* esté activo.

## Nota

Django lanza una nueva versión aproximadamente cada ocho meses, por lo que es posible que veas una versión más reciente cuando instales Django. Lo más probable es que este proyecto funcione tal y como está escrito aquí, incluso en versiones más recientes de Django. Si quieres asegurarte de utilizar la misma versión de Django que ves aquí, utiliza el comando pip install django==4.1.\*. Esto instalará la última versión de Django 4.1. Si tienes algún problema relacionado con la versión que estás utilizando, consulta los recursos en línea de este libro en <https://ehmatthes.github.io/pcc_3e>.

### Crear un proyecto en Django

Sin salir del entorno virtual activo (recuerda buscar *ll\_env* entre paréntesis en el prompt del terminal), introduce los siguientes comandos para crear un nuevo proyecto:

❶ (ll\_env)learning\_log$ django-admin startproject ll\_project .  
❷ (ll\_env)learning\_log$ ls  
ll\_env ll\_project manage.py  
❸ (ll\_env)learning\_log$ ls ll\_project  
\_\_init\_\_.py asgi.py settings.py urls.py wsgi.py

El comando startproject ❶ indica a Django que cree un nuevo proyecto llamado *ll\_project*. El punto (.) al final del comando crea el nuevo proyecto con una estructura de directorios que facilitará el despliegue de la aplicación en un servidor cuando hayamos terminado de desarrollarla.

## Nota

No olvides este punto, o podrías encontrarte con algunos problemas de configuración cuando despliegues la aplicación. Si olvidas el punto, borra los archivos y carpetas creados (excepto *ll\_env*) y vuelve a ejecutar el comando.

Ejecutar el comando ls (dir en Windows) ❷ muestra que Django ha creado un nuevo directorio llamado *ll\_project*. También ha creado un archivo *manage.py*, que es un pequeño programa que recibe comandos y los envía a la parte relevante de Django. Utilizaremos estos comandos para gestionar tareas, como trabajar con bases de datos y ejecutar servidores.

El directorio *ll\_project* contiene cuatro archivos ❸; los más importantes son *settings.py*, *urls.py* y *wsgi.py*. El archivo *settings.py* controla la forma en que Django interactúa con tu sistema y gestiona tu proyecto. Modificaremos algunos de estos ajustes y añadiremos algunos ajustes propios a medida que evolucione el proyecto. El archivo *urls.py* indica a Django qué páginas debe construir en respuesta a las peticiones del navegador. El archivo *wsgi.py* ayuda a Django a servir los archivos que crea. El nombre del archivo es un acrónimo de "web server gateway interface".

### Crear la base de datos

Django almacena la mayor parte de la información de un proyecto en una base de datos, así que lo siguiente que tenemos que hacer es crear una base de datos con la que Django pueda trabajar. Introduce el siguiente comando (aún en un entorno activo):

(ll\_env)learning\_log$ python manage.py migrate  
❶ Operations to perform:  
 Apply all migrations: admin, auth, contenttypes, sessions  
Running migrations:  
 Applying contenttypes.0001\_initial... OK  
 Applying auth.0001\_initial... OK  
 --snip--  
 Applying sessions.0001\_initial... OK  
❷ (ll\_env)learning\_log$ ls  
db.sqlite3 ll\_env ll\_project manage.py

Cada vez que modificamos una base de datos, decimos que estamos *migrating* la base de datos. Ejecutar el comando migrate por primera vez le dice a Django que se asegure de que la base de datos coincide con el estado actual del proyecto. La primera vez que ejecutemos este comando en un nuevo proyecto que utilice SQLite (más sobre SQLite en un momento), Django creará una nueva base de datos para nosotros. Aquí, Django informa de que preparará la base de datos para almacenar la información que necesita para gestionar las tareas administrativas y de autenticación ❶.

Al ejecutar el comando ls se muestra que Django ha creado otro archivo llamado *db.sqlite3* ❷. *SQLite* es una base de datos que se ejecuta desde un único archivo; es ideal para escribir aplicaciones sencillas porque no tendrás que prestar mucha atención a la gestión de la base de datos.

## Nota

En un entorno virtual activo, utiliza el comando python para ejecutar los comandos manage.py, aunque utilices algo diferente, como python3, para ejecutar otros programas. En un entorno virtual, el comando python hace referencia a la versión de Python que se utilizó para crear el entorno virtual.

### Ver el proyecto

Asegurémonos de que Django ha configurado el proyecto correctamente. Introduce el comando runserver para ver el proyecto en su estado actual:

(ll\_env)learning\_log$ python manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
❶ System check identified no issues (0 silenced).  
May 19, 2022 - 21:52:35  
❷ Django version 4.1, using settings 'll\_project.settings'  
❸ Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.

Django debería iniciar un servidor llamado *development server*, para que puedas ver el proyecto en tu sistema y comprobar si funciona correctamente. Cuando solicitas una página introduciendo una URL en un navegador, el servidor Django responde a esa solicitud construyendo la página apropiada y enviándola al navegador.

Django comprueba primero que el proyecto está configurado correctamente ❶; a continuación, informa de la versión de Django en uso y del nombre del archivo de configuración en uso ❷. Por último, informa de la URL en la que se está sirviendo el proyecto ❸. La URL *http://127.0.0.1:8000/* indica que el proyecto está a la escucha de peticiones en el puerto 8000 de tu ordenador, que se denomina localhost. El término  *localhost* se refiere a un servidor que sólo procesa peticiones en tu sistema; no permite que nadie más vea las páginas que estás desarrollando.

Abre un navegador web e introduce la URL *http://localhost:8000/*, o *http://127.0.0.1:8000/* si la primera no funciona. Deberías ver algo como [la Figura 18-1](#figure18-1): una página que Django crea para informarte de que todo funciona correctamente hasta el momento. Mantén el servidor en funcionamiento por ahora, pero cuando quieras detenerlo, pulsa CTRL-C en el terminal donde se emitió el comando runserver.

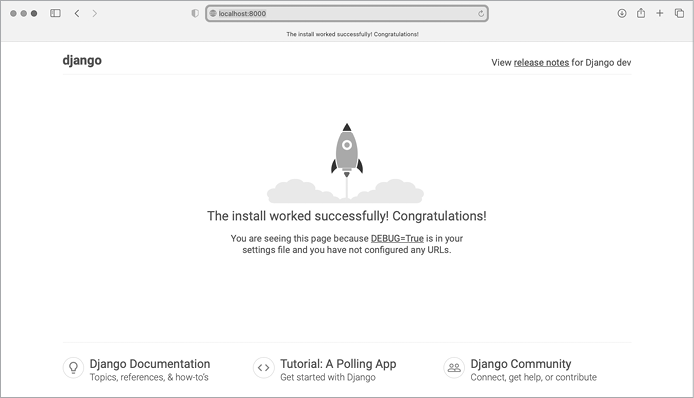


Figura 18-1: Hasta ahora todo funciona correctamente.

## Nota

Si recibes el mensaje de error "Ese puerto ya está en uso", dile a Django que utilice un puerto diferente introduciendo python manage.py runserver 8001 y luego pasando por números superiores hasta que encuentres un puerto abierto.

## Pruébalo tú mismo

18-1. Nuevos proyectos: Para hacerte una mejor idea de lo que hace Django, crea un par de proyectos vacíos y observa lo que crea Django. Crea una nueva carpeta con un nombre sencillo, como *tik\_gram* o *insta\_tok* (fuera de tu directorio *learning\_log* ), navega hasta esa carpeta en un terminal y crea un entorno virtual. Instala Django y ejecuta el comando django-admin.py startproject tg\_project . (asegurándote de incluir el punto al final del comando).

Observa los archivos y carpetas que crea este comando, y compáralos con el Registro de Aprendizaje. Hazlo varias veces, hasta que te familiarices con lo que crea Django al iniciar un nuevo proyecto. Después, borra los directorios del proyecto si lo deseas.

## Iniciar una aplicación

Un proyecto Django *project* está organizado como un grupo de *apps* individuales que trabajan juntos para hacer que el proyecto funcione como un todo. Por ahora, crearemos una app para que realice la mayor parte del trabajo de nuestro proyecto. Añadiremos otra app en el Capítulo 19 para gestionar las cuentas de usuario.

Deberías dejar el servidor de desarrollo funcionando en la ventana de terminal que abriste antes. Abre una nueva ventana de terminal (o pestaña) y navega hasta el directorio que contiene *manage.py*. Activa el entorno virtual y, a continuación, ejecuta el comando startapp:

learning\_log$ source ll\_env/bin/activate  
(ll\_env)learning\_log$ python manage.py startapp learning\_logs  
❶ (ll\_env)learning\_log$ ls  
db.sqlite3 learning\_logs ll\_env ll\_project manage.py  
❷ (ll\_env)learning\_log$ ls learning\_logs/  
\_\_init\_\_.py admin.py apps.py migrations models.py tests.py views.py

El comando startapp appname indica a Django que cree la infraestructura necesaria para construir una app. Cuando mires ahora en el directorio del proyecto, verás una nueva carpeta llamada *learning\_logs* ❶. Utiliza el comando ls para ver qué ha creado Django ❷. Los archivos más importantes son *models.py*, *admin.py*, y *views.py*. Utilizaremos *models.py* para definir los datos que queremos gestionar en nuestra aplicación. Veremos *admin.py* y *views.py* un poco más adelante.

### Definir modelos

Pensemos un momento en nuestros datos. Cada usuario tendrá que crear una serie de temas en su registro de aprendizaje. Cada entrada que hagan estará vinculada a un tema, y estas entradas se mostrarán como texto. También necesitaremos almacenar la marca de tiempo de cada entrada para poder mostrar a los usuarios cuándo hicieron cada una.

Abre el archivo *models.py* y mira el contenido existente:

**models.py**

from django.db import models  
  
# Create your models here.

Se está importando un módulo llamado models, y se nos invita a crear modelos propios. Un *model* indica a Django cómo trabajar con los datos que se almacenarán en la aplicación. Un modelo es una clase; tiene atributos y métodos, como todas las clases de las que hemos hablado. Este es el modelo para los temas que almacenarán los usuarios:

from django.db import models  
  
class Topic(models.Model):  
 """A topic the user is learning about."""  
❶ text = models.CharField(max\_length=200)  
❷ date\_added = models.DateTimeField(auto\_now\_add=True)  
  
❸ def \_\_str\_\_(self):  
 """Return a string representation of the model."""  
 return self.text

Hemos creado una clase llamada Topic, que hereda de Model-una clase padre incluida en Django que define la funcionalidad básica de un modelo. Añadimos dos atributos a la clase Topic: text y date\_added.

El atributo text es un CharField, un dato formado por caracteres o texto ❶. Utilizas CharField cuando quieres almacenar una pequeña cantidad de texto, como un nombre, un título o una ciudad. Cuando definimos un atributo CharField, tenemos que decirle a Django cuánto espacio debe reservar en la base de datos. Aquí le damos un max\_length de 200 caracteres, que debería ser suficiente para contener la mayoría de los nombres de temas.

El atributo date\_added es un DateTimeField, un dato que registrará una fecha y una hora ❷. Pasamos el argumento auto\_now\_add=True, que indica a Django que establezca automáticamente este atributo con la fecha y hora actuales cada vez que el usuario cree un nuevo tema.

Es una buena idea decirle a Django cómo quieres que represente una instancia de un modelo. Si un modelo tiene un método \_\_str\_\_(), Django llama a ese método cada vez que necesita generar una salida que haga referencia a una instancia de ese modelo. Aquí hemos escrito un método \_\_str\_\_() que devuelve el valor asignado al atributo text ❸.

Para ver los distintos tipos de campos que puedes utilizar en un modelo, consulta la página "Referencia de campos de modelo" en [https://docs.djangoproject.com/en/4.1/ref/models/fields.](https://docs.djangoproject.com/en/4.1/ref/models/fields) No necesitarás toda la información ahora mismo, pero te será muy útil cuando desarrolles tus propios proyectos Django.

### Activar modelos

Para utilizar nuestros modelos, tenemos que decirle a Django que incluya nuestra aplicación en el proyecto general. Abre *settings.py* (en el directorio *ll\_project* ); verás una sección que indica a Django qué aplicaciones están instaladas en el proyecto:

**settings.py**

--snip--  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]  
--snip--

Añade nuestra app a esta lista modificando INSTALLED\_APPS para que tenga este aspecto:

--snip--  
INSTALLED\_APPS = [  
 # My apps.  
 'learning\_logs',  
  
 # Default django apps.  
 'django.contrib.admin',  
 --snip--  
]  
--snip--

Agrupar aplicaciones en un proyecto ayuda a seguirlas cuando el proyecto crece e incluye más aplicaciones. Aquí empezamos una sección llamada My apps, que por ahora sólo incluye 'learning\_logs'. Es importante que coloques tus propias aplicaciones antes de las aplicaciones por defecto, por si necesitas sustituir algún comportamiento de las aplicaciones por defecto por tu propio comportamiento personalizado.

A continuación, tenemos que decirle a Django que modifique la base de datos para que pueda almacenar información relacionada con el modelo Topic. Desde el terminal, ejecuta el siguiente comando:

(ll\_env)learning\_log$ python manage.py makemigrations learning\_logs  
Migrations for 'learning\_logs':  
 learning\_logs/migrations/0001\_initial.py  
 - Create model Topic  
(ll\_env)learning\_log$

El comando makemigrations indica a Django que averigüe cómo modificar la base de datos para que pueda almacenar los datos asociados a los nuevos modelos que hayamos definido. La salida aquí muestra que Django ha creado un archivo de migración llamado *0001\_initial.py*. Esta migración creará una tabla para el modelo Topic en la base de datos.

Ahora aplicaremos esta migración y haremos que Django modifique la base de datos por nosotros:

(ll\_env)learning\_log$ python manage.py migrate  
Operations to perform:  
 Apply all migrations: admin, auth, contenttypes, learning\_logs, sessions  
Running migrations:  
 Applying learning\_logs.0001\_initial... OK

La mayor parte de la salida de este comando es idéntica a la salida de la primera vez que ejecutamos el comando migrate. Tenemos que comprobar la última línea de esta salida, donde Django confirma que la migración para learning\_logs funcionó OK.

Siempre que queramos modificar los datos que gestiona Learning Log, seguiremos estos tres pasos: modificar *models.py*, llamar a makemigrations en learning\_logs, y decirle a Django que migrate el proyecto.

### El sitio de administración de Django

Django facilita el trabajo con tus modelos a través de su sitio de administración. El sitio *admin site* de Django sólo está pensado para ser utilizado por los administradores del sitio; no está pensado para usuarios normales. En esta sección, configuraremos el sitio de administración y lo utilizaremos para añadir algunos temas a través del modelo Topic.

#### Configurar un superusuario

Django te permite crear un *superuser*, un usuario que tiene todos los privilegios disponibles en el sitio. Los *privileges* de un usuario controlan las acciones que puede realizar. La configuración de privilegios más restrictiva permite a un usuario leer únicamente la información pública del sitio. Los usuarios registrados suelen tener el privilegio de leer sus propios datos privados y alguna información seleccionada disponible sólo para los miembros. Para administrar eficazmente un proyecto, el propietario del sitio suele necesitar acceso a toda la información almacenada en él. Un buen administrador es cuidadoso con la información sensible de sus usuarios, porque éstos depositan mucha confianza en las aplicaciones a las que acceden.

Para crear un superusuario en Django, introduce el siguiente comando y responde a las indicaciones:

(ll\_env)learning\_log$ python manage.py createsuperuser  
❶ Username (leave blank to use 'eric'): ll\_admin  
❷ Email address:  
❸ Password:  
Password (again):  
Superuser created successfully.  
(ll\_env)learning\_log$

Cuando emitas el comando createsuperuser, Django te pedirá que introduzcas un nombre de usuario para el superusuario ❶. Aquí estoy utilizando ll\_admin, pero puedes introducir cualquier nombre de usuario que desees. Puedes introducir una dirección de correo electrónico o simplemente dejar este campo en blanco ❷. Tendrás que introducir tu contraseña dos veces ❸.

## Nota

Alguna información sensible puede ocultarse a los administradores de un sitio. Por ejemplo, Django no almacena la contraseña que introduces; en su lugar, almacena una cadena derivada de la contraseña, denominada *hash*. Cada vez que introduces tu contraseña, Django realiza un hash de tu entrada y la compara con el hash almacenado. Si los dos hashes coinciden, estás autenticado. Al exigir que los hashes coincidan, Django se asegura de que si un atacante consigue acceder a la base de datos de un sitio, podrá leer los hashes almacenados, pero no las contraseñas. Cuando un sitio está configurado correctamente, es casi imposible obtener las contraseñas originales a partir de los hashes.

#### Registrar un modelo en el sitio de administración

Django incluye algunos modelos en el sitio de administración de forma automática, como User y Group, pero los modelos que creemos deben añadirse manualmente.

Cuando iniciamos la aplicación learning\_logs, Django creó un archivo *admin.py* en el mismo directorio que *models.py*. Abre el archivo *admin.py*:

**admin.py**

from django.contrib import admin  
  
# Register your models here.

Para registrar Topic en el sitio de administración, introduce lo siguiente:

from django.contrib import admin  
  
from .models import Topic  
  
admin.site.register(Topic)

Este código importa primero el modelo que queremos registrar, Topic. El punto delante de models indica a Django que busque *models.py* en el mismo directorio que *admin.py*. El código admin.site.register() le dice a Django que gestione nuestro modelo a través del sitio admin.

Ahora utiliza la cuenta de superusuario para acceder al sitio admin. Ve a *http://localhost:8000/admin/* e introduce el nombre de usuario y la contraseña del superusuario que acabas de crear. Deberías ver una pantalla similar a la que se muestra en la [Figura 18-2](#figure18-2). Esta página te permite añadir nuevos usuarios y grupos, y modificar los existentes. También puedes trabajar con datos relacionados con el modelo Topic que acabamos de definir.

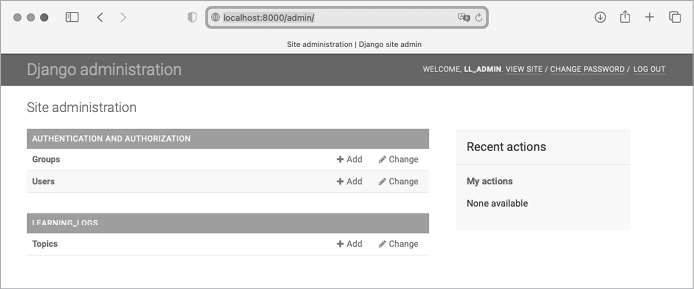


Figura 18-2: El sitio de administración con Topic incluido

## Nota

Si ves un mensaje en tu navegador indicando que la página web no está disponible, asegúrate de que aún tienes el servidor Django ejecutándose en una ventana de terminal. Si no es así, activa un entorno virtual y vuelve a emitir el comando python manage.py runserver. Si tienes problemas para ver tu proyecto en cualquier punto del proceso de desarrollo, cerrar cualquier terminal abierta y volver a emitir el comando runserver es un buen primer paso para solucionar el problema.

#### Añadir temas

Ahora que Topic se ha registrado en el sitio de administración, vamos a añadir nuestro primer tema. Haz clic en **Temas** para ir a la página Temas, que está casi vacía, porque aún no tenemos temas que gestionar. Haz clic en **Añadir tema** y aparecerá un formulario para añadir un nuevo tema. Introduce Chess en la primera casilla y haz clic en **Guardar**. Volverás a la página de administración de Temas y verás el tema que acabas de crear.

Vamos a crear un segundo tema para tener más datos con los que trabajar. Vuelve a hacer clic en **Añadir tema** e introduce Rock Climbing. Haz clic en **Guardar** y volverás a la página principal de Temas. Ahora verás que aparecen Ajedrez y Escalada.

### Definir el modelo de entrada

Para que un usuario pueda registrar lo que ha estado aprendiendo sobre ajedrez y escalada, necesitamos definir un modelo para los tipos de entradas que los usuarios pueden hacer en sus registros de aprendizaje. Cada entrada debe estar asociada a un tema concreto. Esta relación se denomina *many-to-one relationship*, lo que significa que muchas entradas pueden asociarse a un tema.

Aquí tienes el código del modelo Entry. Colócalo en tu archivo *models.py* :

**models.py**

from django.db import models  
  
class Topic(models.Model):  
 --snip--  
  
❶ class Entry(models.Model):  
 """Something specific learned about a topic."""  
❷ topic = models.ForeignKey(Topic, on\_delete=models.CASCADE)  
❸ text = models.TextField()  
 date\_added = models.DateTimeField(auto\_now\_add=True)  
  
❹ class Meta:  
 verbose\_name\_plural = 'entries'  
  
 def \_\_str\_\_(self):  
 """Return a simple string representing the entry."""  
❺ return f"{self.text[:50]}..."

La clase Entry hereda de la clase base Model de Django, al igual que Topic ❶. El primer atributo, topic, es una instancia de ForeignKey ❷. Un *foreign key* es un término de base de datos; es una referencia a otro registro de la base de datos. Es el código que conecta cada entrada con un tema concreto. A cada tema se le asigna un *key*, o ID, cuando se crea. Cuando Django necesita establecer una conexión entre dos datos, utiliza las claves asociadas a cada información. En breve utilizaremos estas conexiones para recuperar todas las entradas asociadas a un determinado tema. El argumento on\_delete=models.CASCADE indica a Django que cuando se elimine un tema, se eliminen también todas las entradas asociadas a ese tema. Esto se conoce como *cascading delete*.

A continuación hay un atributo llamado text, que es una instancia de TextField ❸. Este tipo de campo no necesita un límite de tamaño, porque no queremos limitar el tamaño de las entradas individuales. El atributo date\_added nos permite presentar las entradas en el orden en que fueron creadas, y colocar una marca de tiempo junto a cada entrada.

La clase Meta está anidada dentro de la clase Entry ❹. La clase Meta contiene información adicional para gestionar un modelo; aquí, nos permite establecer un atributo especial que indica a Django que utilice Entries cuando necesite hacer referencia a más de una entrada. Sin esto, Django se referiría a varias entradas como Entrys.

El método \_\_str\_\_() indica a Django qué información debe mostrar cuando hace referencia a entradas individuales. Como una entrada puede ser un cuerpo de texto largo, \_\_str\_\_() devuelve sólo los primeros 50 caracteres de text ❺. También añadimos una elipsis para aclarar que no siempre mostramos la entrada completa.

### Migrar el modelo de entrada

Como hemos añadido un nuevo modelo, tenemos que volver a migrar la base de datos. Este proceso te resultará bastante familiar: modificas *models.py*, ejecutas el comando python manage.py makemigrations nombre\_de\_la\_aplicación, y luego ejecutas el comando python manage.py migrate.

Migra la base de datos y comprueba el resultado introduciendo los siguientes comandos:

(ll\_env)learning\_log$ python manage.py makemigrations learning\_logs  
Migrations for 'learning\_logs':  
❶ learning\_logs/migrations/0002\_entry.py  
 - Create model Entry  
(ll\_env)learning\_log$ python manage.py migrate  
Operations to perform:  
 --snip--  
❷ Applying learning\_logs.0002\_entry... OK

Se genera una nueva migración llamada *0002\_entry.py*, que indica a Django cómo modificar la base de datos para almacenar información relacionada con el modelo Entry ❶. Cuando emitimos el comando migrate, vemos que Django aplicó esta migración y todo funcionó correctamente ❷.

### Registro de entrada en el sitio de administración

También tenemos que registrar el modelo Entry. Este es el aspecto que debería tener ahora *admin.py*:

**admin.py**

from django.contrib import admin  
  
from .models import Topic, Entry  
  
admin.site.register(Topic)  
admin.site.register(Entry)

Vuelve a *http://localhost/admin/*, y deberías ver Entradas listadas bajo *Learning\_Logs*. Haz clic en el enlace **Añadir** para Entradas, o haz clic en **Entradas** y luego elige **Añadir entrada**. Deberías ver una lista desplegable para seleccionar el tema para el que estás creando una entrada y un cuadro de texto para añadir una entrada. Selecciona **Ajedrez** en la lista desplegable y añade una entrada. Ésta es la primera entrada que hice:

La apertura es la primera parte de la partida, más o menos las diez primeras jugadas. En la apertura, conviene hacer tres cosas: sacar los alfiles y los caballos, intentar controlar el centro del tablero y enrocar el rey.

Por supuesto, esto son sólo directrices. Será importante aprender cuándo seguir estas directrices y cuándo hacer caso omiso de estas sugerencias.

Cuando hagas clic en **Guardar**, volverás a la página principal de administración de las entradas. Aquí, verás la ventaja de utilizar text[:50] como representación de cadena para cada entrada; es mucho más fácil trabajar con varias entradas en la interfaz de administración si sólo ves la primera parte de una entrada, en lugar del texto completo de cada entrada.

Haz una segunda entrada para Ajedrez y otra para Escalada, para que tengamos algunos datos iniciales. Aquí tienes una segunda entrada para Ajedrez:

En la fase inicial de la partida, es importante sacar a relucir tus alfiles y caballos. Estas piezas son lo bastante poderosas y maniobrables como para desempeñar un papel importante en las jugadas iniciales de una partida.

Y aquí tienes una primera entrada para Escalada:

Uno de los conceptos más importantes en la escalada es mantener el peso sobre los pies tanto como sea posible. Existe el mito de que los escaladores pueden colgarse todo el día de los brazos. En realidad, los buenos escaladores han practicado formas específicas de mantener su peso sobre los pies siempre que sea posible.

Estas tres entradas nos darán algo con lo que trabajar mientras seguimos desarrollando el Registro de Aprendizaje.

### El caparazón de Django

Ahora que hemos introducido algunos datos, podemos examinarlos programáticamente a través de una sesión de terminal interactiva. Este entorno interactivo se llama Django *shell*, y es un entorno estupendo para probar y solucionar problemas de tu proyecto. Aquí tienes un ejemplo de sesión de terminal interactiva:

(ll\_env)learning\_log$ python manage.py shell  
❶ >>> from learning\_logs.models import Topic  
>>> Topic.objects.all()  
<QuerySet [<Topic: Chess>, <Topic: Rock Climbing>]>

El comando python manage.py shell, ejecutado en un entorno virtual activo, lanza un intérprete de Python que puedes utilizar para explorar los datos almacenados en la base de datos de tu proyecto. Aquí, importamos el modelo Topic del módulo learning\_logs.models ❶. A continuación, utilizamos el método Topic.objects.all() para obtener todas las instancias del modelo Topic; la lista que se devuelve se denomina *queryset*.

Podemos hacer un bucle sobre un conjunto de consultas igual que lo haríamos sobre una lista. Así puedes ver el ID que se ha asignado a cada objeto tema:

>>> topics = Topic.objects.all()  
>>> for topic in topics:  
... print(topic.id, topic)  
...  
1 Chess  
2 Rock Climbing

Asignamos el conjunto de consultas a topics y, a continuación, imprimimos el atributo id de cada tema y la representación de cadena de cada tema. Podemos ver que Chess tiene un ID de 1 y Rock Climbing tiene un ID de 2.

Si conoces el ID de un objeto concreto, puedes utilizar el método Topic.objects.get() para recuperar ese objeto y examinar cualquier atributo que tenga. Veamos los valores text y date\_added de Chess:

>>> t = Topic.objects.get(id=1)  
>>> t.text  
'Chess'  
>>> t.date\_added  
datetime.datetime(2022, 5, 20, 3, 33, 36, 928759,  
 tzinfo=datetime.timezone.utc)

También podemos mirar las entradas relacionadas con un tema determinado. Anteriormente, definimos el atributo topic para el modelo Entry. Se trataba de un ForeignKey, una conexión entre cada entrada y un tema. Django puede utilizar esta conexión para obtener cada entrada relacionada con un tema determinado, de la siguiente manera:

❶ >>> t.entry\_set.all()  
<QuerySet [<Entry: The opening is the first part of the game, roughly...>, <Entry:  
In the opening phase of the game, it's important t...>]>

Para obtener datos a través de una relación de clave foránea, utiliza el nombre en minúsculas del modelo relacionado seguido de un guión bajo y la palabra set ❶. Por ejemplo, supongamos que tienes los modelos Pizza y Topping, y que Topping está relacionado con Pizza a través de una clave ajena. Si tu objeto se llama my\_pizza, y representa una única pizza, puedes obtener todos los ingredientes de la pizza utilizando el código my\_pizza.topping\_set.all().

Utilizaremos esta sintaxis cuando empecemos a codificar las páginas que pueden solicitar los usuarios. El intérprete de comandos es realmente útil para asegurarte de que tu código recupera los datos que deseas. Si tu código funciona como esperas en el intérprete de comandos, también debería funcionar correctamente en los archivos de tu proyecto. Si tu código genera errores o no recupera los datos que esperas, es mucho más fácil solucionar los problemas de tu código en el sencillo entorno del shell que dentro de los archivos que generan las páginas web. No nos referiremos mucho al shell, pero deberías seguir utilizándolo para practicar el trabajo con la sintaxis de Django para acceder a los datos almacenados en el proyecto.

Cada vez que modifiques tus modelos, tendrás que reiniciar la shell para ver los efectos de esos cambios. Para salir de una sesión de shell, pulsa CTRL-D; en Windows, pulsa CTRL-Z y luego ENTER.

## Pruébalo tú mismo

18-2. Entradas cortas: El método \_\_str\_\_() del modelo Entry actualmente añade una elipsis a cada instancia de Entry cuando Django lo muestra en el sitio de administración o en el shell. Añade una sentencia if al método \_\_str\_\_() que añada una elipsis sólo si la entrada tiene más de 50 caracteres. Utiliza el sitio de administración para añadir una entrada que tenga menos de 50 caracteres de longitud, y comprueba que no tiene una elipsis cuando se visualiza.

18-3. La API de Django: Cuando escribes código para acceder a los datos de tu proyecto, estás escribiendo un *query*. Hojea la documentación para consultar tus datos en <https://docs.djangoproject.com/en/4.1/topics/db/queries>*.*. Mucho de lo que veas te parecerá nuevo, pero te será bastante útil cuando empieces a trabajar en tus propios proyectos.

18-4. Pizzería: Inicia un nuevo proyecto llamado pizzeria\_project con una aplicación llamada pizzas. Define un modelo Pizza con un campo llamado name, que contendrá valores de nombre, como Hawaiian y Meat Lovers. Define un modelo llamado Topping con campos llamados pizza y name. El campo pizza debe ser una clave ajena a Pizza, y name debe poder contener valores como pineapple, Canadian bacon y sausage.

Registra ambos modelos en el sitio de administración y utilízalo para introducir algunos nombres de pizza y sus ingredientes. Utiliza el intérprete de comandos para explorar los datos introducidos.

## Crear páginas: La Página de Inicio del Registro de Aprendizaje

Hacer páginas web con Django consta de tres etapas: definir URLs, escribir vistas y escribir plantillas. Puedes hacerlas en cualquier orden, pero en este proyecto siempre empezaremos definiendo el patrón de URL. Un *URL pattern* describe la forma en que está dispuesta la URL. También le dice a Django qué debe buscar cuando hace coincidir una petición del navegador con la URL de un sitio, para saber qué página debe devolver.

Cada URL se asigna a una vista concreta. La función *view* recupera y procesa los datos necesarios para esa página. La función de vista a menudo renderiza la página utilizando un *template*, que contiene la estructura general de la página. Para ver cómo funciona esto, vamos a crear la página de inicio del Registro de Aprendizaje. Definiremos la URL de la página de inicio, escribiremos su función de vista y crearemos una plantilla sencilla.

Como sólo queremos asegurarnos de que Learning Log funciona como se supone que debe hacerlo, de momento crearemos una página sencilla. Es divertido dar estilo a una aplicación web que funciona cuando está completa; una aplicación que tiene buen aspecto pero no funciona bien no tiene sentido. Por ahora, la página de inicio sólo mostrará un título y una breve descripción.

### Asignar una URL

Los usuarios solicitan las páginas introduciendo URL en un navegador y haciendo clic en los enlaces, así que tendremos que decidir qué URL son necesarias. La URL de la página de inicio es lo primero: es la URL base que la gente utiliza para acceder al proyecto. Por el momento, la URL base, *http://localhost:8000/*, devuelve el sitio Django por defecto que nos permite saber que el proyecto se ha configurado correctamente. Cambiaremos esto asignando la URL base a la página de inicio de Learning Log.

En la carpeta principal *ll\_project*, abre el archivo *urls.py*. Deberías ver el siguiente código:

**ll\_project/urls.py**

❶ from django.contrib import admin  
from django.urls import path  
  
❷ urlpatterns = [  
❸ path('admin/', admin.site.urls),  
]

Las dos primeras líneas importan el módulo admin y una función para construir rutas URL ❶. El cuerpo del archivo define la variable urlpatterns ❷. En este archivo *urls.py*, que define las URL para el proyecto en su conjunto, la variable urlpatterns incluye conjuntos de URL de las aplicaciones del proyecto. La lista incluye el módulo admin.site.urls, que define todas las URL que se pueden solicitar desde el sitio de administración ❸.

Necesitamos incluir las URL de learning\_logs, así que añade lo siguiente:

from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('learning\_logs.urls')),  
]

Hemos importado la función include(), y también hemos añadido una línea para incluir el módulo learning\_logs.urls.

Por defecto, *urls.py* está en la carpeta *ll\_project* ; ahora necesitamos crear un segundo archivo *urls.py* en la carpeta *learning\_logs* . Crea un nuevo archivo Python, guárdalo como *urls.py* en *learning\_logs*, e introduce en él este código:

**learning\_logs/urls.py**

❶ """Defines URL patterns for learning\_logs."""  
  
❷ from django.urls import path  
  
❸ from . import views  
  
❹ app\_name = 'learning\_logs'  
❺ urlpatterns = [  
 # Home page  
❻ path('', views.index, name='index'),  
]

Para dejar claro en qué *urls.py* estamos trabajando, añadimos un docstring al principio del archivo ❶. A continuación, importamos la función path, necesaria para asignar URLs a vistas ❷. También importamos el módulo views ❸; el punto indica a Python que importe el módulo *views.py* del mismo directorio que el módulo actual *urls.py*. La variable app\_name ayuda a Django a distinguir este archivo *urls.py* de los archivos del mismo nombre de otras aplicaciones del proyecto ❹. La variable urlpatterns de este módulo es una lista de páginas individuales que pueden solicitarse a la aplicación learning\_logs ❺.

El patrón URL real es una llamada a la función path(), que toma tres argumentos ❻. El primer argumento es una cadena que ayuda a Django a enrutar correctamente la solicitud actual. Django recibe la URL solicitada e intenta dirigir la solicitud a una vista. Para ello, busca en todos los patrones de URL que hemos definido uno que coincida con la solicitud actual. Django ignora la URL base del proyecto (*http://localhost:8000/*), por lo que la cadena vacía ('') coincide con la URL base. Cualquier otra URL no coincidirá con este patrón, y Django devolverá una página de error si la URL solicitada no coincide con ningún patrón de URL existente.

El segundo argumento de path() ❻ especifica a qué función llamar en *views.py*. Cuando una URL solicitada coincide con el patrón que estamos definiendo, Django llama a la función index() de *views.py*. (Escribiremos esta función de vista en la siguiente sección.) El tercer argumento proporciona el nombre *index* a este patrón de URL para que podamos referirnos a él más fácilmente en otros archivos a lo largo del proyecto. Siempre que queramos proporcionar un enlace a la página de inicio, utilizaremos este nombre en lugar de escribir una URL.

### Escribir una vista

Una función de vista recibe información de una solicitud, prepara los datos necesarios para generar una página y, a continuación, devuelve los datos al navegador. A menudo lo hace utilizando una plantilla que define el aspecto que tendrá la página.

El archivo *views.py* en *learning\_logs* se generó automáticamente cuando ejecutamos el comando python manage.py startapp. Esto es lo que hay ahora en *views.py*:

**views.py**

from django.shortcuts import render  
  
# Create your views here.

Actualmente, este archivo sólo importa la función render(), que renderiza la respuesta basándose en los datos proporcionados por views. Abre *views.py* y añade el siguiente código para la página de inicio:

from django.shortcuts import render  
  
def index(request):  
 """The home page for Learning Log."""  
 return render(request, 'learning\_logs/index.html')

Cuando una petición de URL coincide con el patrón que acabamos de definir, Django busca una función llamada index() en el archivo *views.py* . A continuación, Django pasa el objeto request a esta función de vista. En este caso, no necesitamos procesar ningún dato para la página, por lo que el único código de la función es una llamada a render(). La función render() pasa aquí dos argumentos: el objeto original request y una plantilla que puede utilizar para construir la página. Escribamos esta plantilla.

### Escribir una plantilla

La plantilla define el aspecto que debe tener la página, y Django rellena los datos relevantes cada vez que se solicita la página. Una plantilla te permite acceder a a cualquier dato proporcionado por la vista. Como nuestra vista para la página de inicio no proporciona datos, esta plantilla es bastante sencilla.

Dentro de la carpeta *learning\_logs*, crea una nueva carpeta llamada *templates*. Dentro de la carpeta *templates*, crea otra carpeta llamada *learning\_logs*. Esto puede parecer un poco redundante (tenemos una carpeta llamada *learning\_logs* dentro de una carpeta llamada *templates* dentro de una carpeta llamada *learning\_logs*), pero establece una estructura que Django puede interpretar sin ambigüedades, incluso en el contexto de un gran proyecto que contenga muchas aplicaciones individuales. Dentro de la carpeta *learning\_logs*, crea un nuevo archivo llamado *index.html*. La ruta del archivo será *ll\_project/learning\_logs/templates/learning\_logs/index.html*. Introduce el siguiente código en ese archivo:

**index.html**

<p>Learning Log</p>  
  
<p>Learning Log helps you keep track of your learning, for any topic you're  
interested in.</p>

Se trata de un archivo muy sencillo. Si no estás familiarizado con HTML, las etiquetas <p></p> significan párrafos. La etiqueta <p> abre un párrafo, y la etiqueta </p> lo cierra. Tenemos dos párrafos: el primero actúa como título, y el segundo describe lo que los usuarios pueden hacer con Learning Log.

Ahora, cuando solicites la URL base del proyecto, *http://localhost:8000/*, deberías ver la página que acabamos de construir en lugar de la página por defecto de Django. Django tomará la URL solicitada, y esa URL coincidirá con el patrón ''; entonces Django llamará a la función views.index(), que renderizará la página utilizando la plantilla contenida en *index.html*. La [Figura 18-3](#figure18-3) muestra la página resultante.

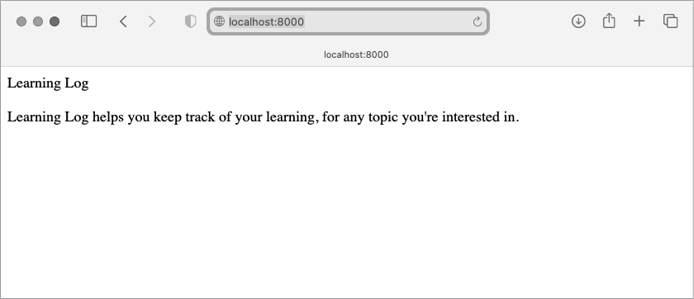


Figura 18-3: Página de inicio del Registro de Aprendizaje

Aunque pueda parecer un proceso complicado para crear una página, esta separación entre URL, vistas y plantillas funciona bastante bien. Te permite pensar en cada aspecto de un proyecto por separado. En proyectos más grandes, permite a las personas que trabajan en el proyecto centrarse en las áreas en las que son más fuertes. Por ejemplo, un especialista en bases de datos puede centrarse en los modelos, un programador en el código de las vistas y un especialista en frontend en las plantillas.

## Nota

Es posible que veas el siguiente mensaje de error:

ModuleNotFoundError: No module named 'learning\_logs.urls'

Si es así, detén el servidor de desarrollo pulsando CTRL-C en la ventana del terminal donde emitiste el comando runserver. A continuación, vuelve a emitir el comando python manage.py runserver. Deberías poder ver la página de inicio. Cada vez que te encuentres con un error como éste, intenta detener y reiniciar el servidor.

## Pruébalo tú mismo

18-5. Planificador de comidas: Piensa en una aplicación que ayude a la gente a planificar sus comidas a lo largo de la semana. Crea una nueva carpeta llamada *meal\_planner*, e inicia un nuevo proyecto Django dentro de esta carpeta. A continuación, crea una nueva aplicación llamada meal\_plans. Haz una página de inicio sencilla para este proyecto.

18-6. Página de inicio de la Pizzería: Añade una página de inicio al proyecto Pizzería que comenzaste en el Ejercicio 18-4 (página 388).

## Construir Páginas Adicionales

Ahora que hemos establecido una rutina para construir una página, podemos empezar a construir el proyecto Registro de Aprendizaje. Construiremos dos páginas que muestren datos: una página que enumere todos los temas y una página que muestre todas las entradas de un tema concreto. Para cada página, especificaremos un patrón URL, escribiremos una función de vista y escribiremos una plantilla. Pero antes, crearemos una plantilla base de la que podrán heredar todas las plantillas del proyecto.

### Herencia de plantillas

Al construir un sitio web, algunos elementos tendrán que repetirse en cada página. En lugar de escribir estos elementos directamente en cada página, puedes escribir una plantilla base que contenga los elementos repetidos y luego hacer que cada página herede de la base. Este enfoque te permite centrarte en desarrollar los aspectos únicos de cada página, y hace que sea mucho más fácil cambiar el aspecto general del proyecto.

#### La plantilla padre

Crearemos una plantilla llamada *base.html* en el mismo directorio que *index.html*. Este archivo contendrá elementos comunes a todas las páginas; todas las demás plantillas heredarán de *base.html*. El único elemento que queremos repetir en cada página en este momento es el título de la parte superior. Como incluiremos esta plantilla en todas las páginas, hagamos que el título sea un enlace a la página de inicio:

**base.html**

<p>  
❶ <a href="{% url 'learning\_logs:index' %}">Learning Log</a>  
</p>  
  
❷ {% block content %}{% endblock content %}

La primera parte de este archivo crea un párrafo que contiene el nombre del proyecto, que también actúa como enlace a la página de inicio. Para generar un enlace, utilizamos una etiqueta *template tag*, que se indica mediante llaves y signos de porcentaje ({% %}). Una etiqueta de plantilla genera información que se mostrará en una página. La etiqueta de plantilla {% url 'learning\_logs:index' %} que se muestra aquí genera una URL que coincide con el patrón de URL definido en *learning\_logs/urls.py* con el nombre 'index' ❶. En este ejemplo, learning\_logs es el *namespace* y index es un patrón de URL con nombre único en ese espacio de nombres. El espacio de nombres procede del valor que asignamos a app\_name en el archivo *learning\_logs/urls.py*.

En una página HTML sencilla, un enlace está rodeado por el *anchor* *tag* <a> :

<a href="link\_url">link text</a>

Hacer que la etiqueta de plantilla genere la URL por nosotros hace que sea mucho más fácil mantener nuestros enlaces actualizados. Sólo tenemos que cambiar el patrón de URL en *urls.py*, y Django insertará automáticamente la URL actualizada la próxima vez que se solicite la página. Todas las páginas de nuestro proyecto heredarán de *base.html*, por lo que, a partir de ahora, todas las páginas tendrán un enlace a la página de inicio.

En la última línea, insertamos un par de etiquetas block ❷. Este bloque, llamado content, es un marcador de posición; la plantilla hija definirá el tipo de información que va en el bloque content.

Una plantilla hija no tiene por qué definir todos los bloques de su padre, así que puedes reservar espacio en las plantillas padre para tantos bloques como quieras; la plantilla hija sólo utilizará los que necesite.

## Nota

En el código Python, casi siempre utilizamos cuatro espacios cuando hacemos sangrías. Los archivos de plantilla suelen tener más niveles de anidamiento que los archivos Python, por lo que es habitual utilizar sólo dos espacios para cada nivel de sangría.

#### La plantilla hija

Ahora tenemos que reescribir *index.html* para que herede de *base.html*. Añade el siguiente código a *index.html*:

**index.html**

❶ {% extends 'learning\_logs/base.html' %}  
  
❷ {% block content %}  
 <p>Learning Log helps you keep track of your learning, for any topic you're  
 interested in.</p>  
❸ {% endblock content %}

Si lo comparas con el original *index.html*, puedes ver que hemos sustituido el título Learning Log por el código para heredar de una plantilla padre ❶. Una plantilla hija debe tener una etiqueta {% extends %} en la primera línea para indicar a Django de qué plantilla padre debe heredar. El archivo *base.html*  forma parte de learning\_logs, así que incluimos *learning\_logs* en la ruta a la plantilla padre. Esta línea recoge todo lo que contiene la plantilla *base.html* y permite a *index.html* definir lo que va en el espacio reservado por el bloque content.

Definimos el bloque de contenido insertando una etiqueta {% block %} con el nombre content ❷. Todo lo que no estamos heredando de la plantilla padre va dentro del bloque content. Aquí, ese es el párrafo que describe el proyecto Learning Log. Indicamos que hemos terminado de definir el contenido utilizando una etiqueta {% endblock content %} ❸. La etiqueta {% endblock %} no requiere un nombre, pero si una plantilla crece hasta contener varios bloques, puede ser útil saber exactamente qué bloque está terminando.

Puedes empezar a ver el beneficio de la herencia de plantillas: en una plantilla hija, sólo necesitamos incluir contenido que sea exclusivo de esa página. Esto no sólo simplifica cada plantilla, sino que también facilita mucho la modificación de la página. Para modificar un elemento común a muchas páginas, sólo tienes que modificar la plantilla padre. Tus cambios se trasladarán a todas las páginas que hereden de esa plantilla. En un proyecto que incluya decenas o cientos de páginas, esta estructura puede hacer mucho más fácil y rápido mejorar tu sitio.

En un proyecto grande, es habitual tener una plantilla padre llamada *base.html* para todo el sitio y plantillas padre para cada sección principal del sitio. Todas las plantillas de sección heredan de *base.html*, y cada página del sitio hereda de una plantilla de sección. De esta forma puedes modificar fácilmente el aspecto del sitio en su conjunto, de cualquier sección del sitio o de cualquier página individual. Esta configuración proporciona una forma muy eficaz de trabajar, y te anima a actualizar tu proyecto de forma constante a lo largo del tiempo.

### La Página de Temas

Ahora que tenemos un enfoque eficiente para construir páginas, podemos centrarnos en nuestras dos siguientes páginas: la página de temas generales y la página para mostrar las entradas de un único tema. La página de temas mostrará todos los temas que han creado los usuarios, y es la primera página que implicará trabajar con datos.

#### El patrón URL de los temas

En primer lugar, definimos la URL de la página de temas. Es habitual elegir un fragmento de URL simple que refleje el tipo de información que se presenta en la página. Utilizaremos la palabra *topics*, por lo que la URL *http://localhost:8000/topics/* devolverá esta página. Así es como modificamos *learning\_logs/urls.py*:

**learning\_logs/urls.py**

"""Defines URL patterns for learning\_logs."""  
--snip--  
urlpatterns = [  
 # Home page  
 path('', views.index, name='index'),  
 # Page that shows all topics.  
 path('topics/', views.topics, name='topics'),  
]

El nuevo patrón de URL es la palabra *topics*, seguida de una barra oblicua. Cuando Django examine una URL solicitada, este patrón coincidirá con cualquier URL que tenga la URL base seguida de *topics*. Puedes incluir u omitir una barra diagonal al final, pero no puede haber nada más después de la palabra *topics*, o el patrón no coincidirá. Cualquier solicitud con una URL que coincida con este patrón se pasará a la función topics() en *views.py*.

#### La vista Temas

La función topics() necesita recuperar algunos datos de la base de datos y enviarlos a la plantilla. Añade lo siguiente a *views.py*:

**views.py**

from django.shortcuts import render  
  
❶ from .models import Topic  
  
def index(request):  
 --snip--  
  
❷ def topics(request):  
 """Show all topics."""  
❸ topics = Topic.objects.order\_by('date\_added')  
❹ context = {'topics': topics}  
❺ return render(request, 'learning\_logs/topics.html', context)

Primero importamos el modelo asociado a los datos que necesitamos ❶. La función topics() necesita un parámetro: el objeto request que Django ha recibido del servidor ❷. Consultamos la base de datos pidiendo los objetos Topic, ordenados por el atributo date\_added ❸. Asignamos el conjunto de consultas resultante a topics.

A continuación, definimos un contexto que enviaremos a la plantilla ❹. Un *context* es un diccionario en el que las claves son nombres que utilizaremos en la plantilla para acceder a los datos que queremos, y los valores son los datos que necesitamos enviar a la plantilla. En este caso, hay un par clave-valor, que contiene el conjunto de temas que mostraremos en la página. Cuando construimos una página que utiliza datos, llamamos a render() con el objeto request, la plantilla que queremos utilizar y el diccionario context ❺.

#### La plantilla Temas

La plantilla para la página de temas recibe el diccionario context, para que la plantilla pueda utilizar los datos que proporciona topics(). Crea un archivo llamado *topics.html* en el mismo directorio que *index.html*. Así es como podemos mostrar los temas en la plantilla:

**topics.html**

{% extends 'learning\_logs/base.html' %}  
  
{% block content %}  
  
 <p>Topics</p>  
  
❶ <ul>  
❷ {% for topic in topics %}  
❸ <li>{{ topic.text }}</li>  
❹ {% empty %}  
 <li>No topics have been added yet.</li>  
❺ {% endfor %}  
❻ </ul>  
  
{% endblock content %}

Utilizamos la etiqueta {% extends %} para heredar de *base.html*, igual que hicimos en la página de inicio, y luego abrimos un bloque content. El cuerpo de esta página contiene una lista con viñetas de los temas que se han introducido. En HTML estándar, una lista con viñetas se denomina *unordered list* y se indica mediante las etiquetas <ul></ul>. La etiqueta de apertura <ul> inicia la lista con viñetas de temas ❶.

A continuación utilizamos una etiqueta de plantilla que equivale a un bucle for, que recorre la lista topics desde el diccionario context ❷. El código utilizado en las plantillas difiere del de Python en algunos aspectos importantes. Python utiliza la sangría para indicar qué líneas de una sentencia for forman parte de un bucle. En una plantilla, cada bucle for necesita una etiqueta {% endfor %} explícita que indique dónde se produce el final del bucle. Por tanto, en una plantilla, verás bucles escritos así:

{% for item in list %}  
 do something with each item  
{% endfor %}

Dentro del bucle, queremos convertir cada tema en un elemento de la lista con viñetas. Para imprimir una variable en una plantilla, encierra el nombre de la variable entre llaves dobles. Las llaves no aparecerán en la página; sólo indican a Django que estamos utilizando una variable de plantilla. Así, el código {{ topic.text }} ❸ será sustituido por el valor del atributo text del tema actual en cada pasada por el bucle. La etiqueta HTML <li></li> indica un *list item*. Cualquier cosa entre estas etiquetas, dentro de un par de etiquetas <ul></ul>, aparecerá como un elemento con viñetas en la lista.

También utilizamos la etiqueta de plantilla {% empty %} ❹, que indica a Django qué hacer si no hay elementos en la lista. En este caso, imprimimos un mensaje informando al usuario de que aún no se ha añadido ningún tema. Las dos últimas líneas cierran el bucle for ❺ y luego cierran la lista con viñetas ❻.

Ahora tenemos que modificar la plantilla base para incluir un enlace a la página de temas. Añade el siguiente código a *base.html*:

**base.html**

<p>  
❶ <a href="{% url 'learning\_logs:index' %}">Learning Log</a> -  
❷ <a href="{% url 'learning\_logs:topics' %}">Topics</a>  
</p>  
  
{% block content %}{% endblock content %}

Añadimos un guión después del enlace a la página de inicio ❶, y luego añadimos de nuevo un enlace a la página de temas utilizando la etiqueta de plantilla {% url %} ❷. Esta línea indica a Django que genere un enlace que coincida con el patrón de URL con el nombre 'topics' en *learning\_logs/urls.py*.

Ahora, cuando actualices la página de inicio en tu navegador, verás un enlace Temas. Cuando hagas clic en el enlace, verás una página similar a la de [la Figura 18-4](#figure18-4).

|  |
| --- |
| Figura |



18-4: Página

de temas

### Páginas de temas individuales

A continuación, necesitamos crear una página que pueda centrarse en un único tema, mostrando el nombre del tema y todas las entradas de ese tema. Definiremos un nuevo patrón URL, escribiremos una vista y crearemos una plantilla. También modificaremos la página de temas para que cada elemento de la lista con viñetas enlace a su correspondiente página de temas.

#### El patrón URL de temas

El patrón URL para la página de temas es un poco diferente de los patrones URL anteriores porque utilizará el atributo id del tema para indicar qué tema se ha solicitado. Por ejemplo, si el usuario quiere ver la página detallada del tema Ajedrez (donde id es 1), la URL será *http://localhost:8000/topics/1/*. Aquí tienes un patrón para que coincida con esta URL, que debes colocar en *learning\_logs/urls.py*:

**learning\_logs/urls.py**

--snip--  
urlpatterns = [  
 --snip--  
 # Detail page for a single topic.  
 path('topics/<int:topic\_id>/', views.topic, name='topic'),  
]

Examinemos la cadena 'topics/<int:topic\_id>/' en este patrón de URL. La primera parte de la cadena indica a Django que busque URL que tengan la palabra *topics* después de la URL base. La segunda parte de la cadena, /<int:topic\_id>/, busca un número entero entre dos barras diagonales y asigna el valor entero a un argumento llamado topic\_id.

Cuando Django encuentra una URL que coincide con este patrón, llama a la función de vista topic() con el valor asignado a topic\_id como argumento. Utilizaremos el valor de topic\_id para obtener el tema correcto dentro de la función.

#### La vista Tema

La función topic() necesita obtener el tema y todas las entradas asociadas de la base de datos, de forma muy similar a lo que hicimos anteriormente en el shell de Django:

**views.py**

--snip--  
❶ def topic(request, topic\_id):  
 """Show a single topic and all its entries."""  
❷ topic = Topic.objects.get(id=topic\_id)  
❸ entries = topic.entry\_set.order\_by('-date\_added')  
❹ context = {'topic': topic, 'entries': entries}  
❺ return render(request, 'learning\_logs/topic.html', context)

Esta es la primera función de la vista que requiere un parámetro distinto del objeto request. La función acepta el valor capturado por la expresión /<int:topic\_id>/ y lo asigna a topic\_id ❶. A continuación, utilizamos get() para recuperar el tema, tal y como hicimos en el intérprete de comandos Django ❷. A continuación, obtenemos todas las entradas asociadas a este tema y las ordenamos según date\_added ❸. El signo menos delante de date\_added ordena los resultados en orden inverso, lo que mostrará primero las entradas más recientes. Almacenamos el tema y las entradas en el diccionario context ❹ y llamamos a render() con el objeto request, la plantilla *topic.html* y el diccionario context ❺.

## Nota

Las frases de código en ❷ y ❸ se llaman *queries*, porque consultan la base de datos en busca de información específica. Cuando escribas consultas como éstas en tus propios proyectos, es útil probarlas primero en el shell de Django. Obtendrás una respuesta mucho más rápida en el intérprete de comandos que si escribes una vista y una plantilla y luego compruebas los resultados en un navegador.

#### La plantilla de temas

La plantilla necesita mostrar el nombre del tema y las entradas. También necesitamos informar al usuario si aún no se han realizado entradas para este tema.

**topic.html**

{% extends 'learning\_logs/base.html' %}  
  
{% block content %}  
  
❶ <p>Topic: {{ topic.text }}</p>  
  
 <p>Entries:</p>  
❷ <ul>  
❸ {% for entry in entries %}  
 <li>  
❹ <p>{{ entry.date\_added|date:'M d, Y H:i' }}</p>  
❺ <p>{{ entry.text|linebreaks }}</p>  
 </li>  
❻ {% empty %}  
 <li>There are no entries for this topic yet.</li>  
 {% endfor %}  
 </ul>  
  
{% endblock content %}

Extendemos *base.html*, como haremos para todas las páginas del proyecto. A continuación, mostramos el atributo text del tema que se ha solicitado ❶. La variable topic está disponible porque está incluida en el diccionario context. A continuación, iniciamos una lista con viñetas ❷ para mostrar cada una de las entradas y hacemos un bucle a través de ellas ❸, como hicimos antes con los temas.

Cada viñeta contiene dos datos: la marca de tiempo y el texto completo de cada entrada. Para la marca de tiempo ❹, mostramos el valor del atributo date\_added. En las plantillas Django, una línea vertical (|) representa una plantilla *filter*-una función que modifica el valor de una variable de plantilla durante el proceso de representación. El filtro date:'M d, Y H:i' muestra marcas de tiempo en el formato *January 1, 2022 23:00*. La siguiente línea muestra el valor del atributo text de la entrada actual. El filtro linebreaks ❺ garantiza que las entradas de texto largas incluyan saltos de línea en un formato que entiendan los navegadores, en lugar de mostrar un bloque de texto ininterrumpido. Volvemos a utilizar la etiqueta de plantilla {% empty %} ❻ para imprimir un mensaje que informe al usuario de que no se han realizado entradas.

#### Enlaces desde la Página de Temas

Antes de ver la página de temas en un navegador, tenemos que modificar la plantilla de temas para que cada tema enlace con la página adecuada. Éste es el cambio que tienes que hacer en *topics.html*:

**topics.html**

--snip--  
 {% for topic in topics %}  
 <li>  
 <a href="{% url 'learning\_logs:topic' topic.id %}">  
 {{ topic.text }}</a></li>  
 </li>  
 {% empty %}  
--snip--

Utilizamos la etiqueta de plantilla de URL para generar el enlace adecuado, basándonos en el patrón de URL de learning\_logs con el nombre 'topic'. Este patrón de URL requiere un argumento topic\_id, así que añadimos el atributo topic.id a la etiqueta de plantilla de URL. Ahora cada tema de la lista de temas es un enlace a una página de tema, como *http://localhost:8000/topics/1/*.

Cuando actualices la página de temas y hagas clic en un tema, deberías ver una página parecida a [la Figura 18-5](#figure18-5).

## Nota

Hay una sutil pero importante diferencia entre topic.id y topic\_id. La expresión topic.id examina un tema y recupera el valor del ID correspondiente. La variable topic\_id es una referencia a ese ID en el código. Si te encuentras con errores al trabajar con IDs, asegúrate de que estás utilizando estas expresiones de la forma adecuada.

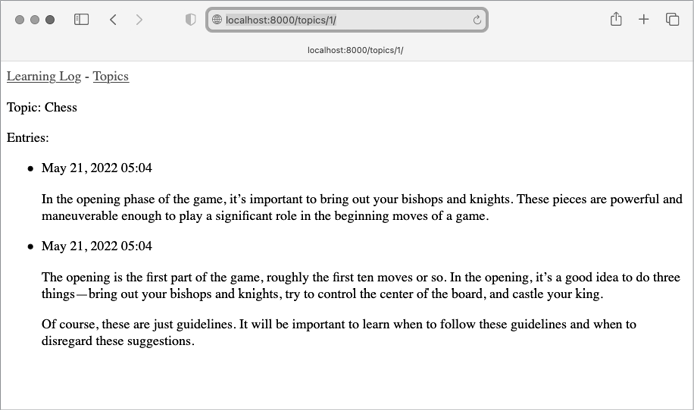


Figura 18-5: La página de detalles de un tema, mostrando todas las entradas de un tema

## Pruébalo tú mismo

18-7. Documentación de plantillas: Hojea la documentación sobre plantillas de Django en [https://docs.djangoproject.com/en/4.1/ref/templates.](https://docs.djangoproject.com/en/4.1/ref/templates) Puedes consultarla cuando trabajes en tus propios proyectos.

18-8. Páginas de la Pizzería: Añade una página al proyecto Pizzería del Ejercicio 18-6(página 392) que muestre los nombres de las pizzas disponibles. Luego enlaza cada nombre de pizza a una página que muestre los ingredientes de la pizza. Asegúrate de utilizar la herencia de plantillas para construir tus páginas de forma eficiente.

## Resumen

En este capítulo, has aprendido cómo empezar a construir una aplicación web sencilla utilizando el framework Django. Has visto una breve especificación del proyecto, has instalado Django en un entorno virtual, has configurado un proyecto y has comprobado que el proyecto estaba configurado correctamente. Configuraste una aplicación y definiste modelos para representar los datos de tu aplicación. Aprendiste sobre bases de datos y cómo Django te ayuda a migrar tu base de datos después de hacer un cambio en tus modelos. Creaste un superusuario para el sitio de administración y lo utilizaste para introducir algunos datos iniciales.

También exploraste el shell de Django, que te permite trabajar con los datos de tu proyecto en una sesión de terminal. Aprendiste a definir URLs, crear funciones de vista y escribir plantillas para crear páginas para tu sitio. También utilizaste la herencia de plantillas para simplificar la estructura de las plantillas individuales y facilitar la modificación del sitio a medida que evoluciona el proyecto.

En el Capítulo 19, crearás páginas intuitivas y fáciles de usar que permitan a los usuarios añadir nuevos temas y entradas y editar las entradas existentes sin pasar por el sitio de administración. También añadirás un sistema de registro de usuarios, que les permita crear una cuenta y crear su propio registro de aprendizaje. Éste es el corazón de una aplicación web: la capacidad de crear algo con lo que pueda interactuar cualquier número de usuarios.