# 20 Estilizar y desplegar una aplicación



Learning Log ya es totalmente funcional, pero no tiene estilo y sólo se ejecuta en tu máquina local. En este capítulo, estilizarás el proyecto de forma sencilla pero profesional y luego lo desplegarás en un servidor en vivo para que cualquier persona del mundo pueda crear una cuenta y utilizarlo.

Para el estilo, utilizaremos la biblioteca *Bootstrap*, una colección de herramientas para dar estilo a las aplicaciones web de modo que tengan un aspecto profesional en todos los dispositivos modernos, desde un pequeño teléfono a un gran monitor de escritorio. Para ello, utilizaremos la aplicación django-bootstrap5, que también te servirá para practicar el uso de aplicaciones creadas por otros desarrolladores de Django.

Desplegaremos Learning Log utilizando *Platform.sh*, un sitio que te permite enviar tu proyecto a uno de sus servidores, poniéndolo a disposición de cualquier persona con conexión a Internet. También empezaremos a utilizar un sistema de control de versiones llamado Git para realizar un seguimiento de los cambios en el proyecto.

Cuando hayas terminado con Learning Log, serás capaz de desarrollar aplicaciones web sencillas, darles un aspecto profesional y desplegarlas en un servidor activo. También podrás utilizar recursos de aprendizaje más avanzados a medida que desarrolles tus habilidades.

## Estilo de Learning Log

Hemos ignorado a propósito el estilo hasta ahora para centrarnos primero en la funcionalidad de Learning Log. Es una buena forma de enfocar el desarrollo, porque una aplicación sólo es útil si funciona. Una vez que una aplicación funciona, su aspecto es fundamental para que la gente quiera utilizarla.

En esta sección, instalaremos la aplicación django-bootstrap5 y la añadiremos al proyecto. Luego la utilizaremos para dar estilo a las páginas individuales del proyecto, de modo que todas las páginas tengan un aspecto coherente.

### La aplicación django-bootstrap5

Utilizaremos django-bootstrap5 para integrar Bootstrap en nuestro proyecto. Esta aplicación descarga los archivos de Bootstrap necesarios, los coloca en una ubicación adecuada de tu proyecto y hace que las directivas de estilo estén disponibles en las plantillas de tu proyecto.

Para instalar django-bootstrap5, ejecuta el siguiente comando en un entorno virtual activo:

(ll\_env)learning\_log$ pip install django-bootstrap5  
--snip--  
Successfully installed beautifulsoup4-4.11.1 django-bootstrap5-21.3  
 soupsieve-2.3.2.post1

A continuación, tenemos que añadir django-bootstrap5 a INSTALLED\_APPS en *settings.py*:

**settings.py**

--snip--  
INSTALLED\_APPS = [  
 # My apps.  
 'learning\_logs',  
 'accounts',  
  
 # Third party apps.  
 'django\_bootstrap5',  
  
 # Default django apps.  
 'django.contrib.admin',  
 --snip--

Crea una nueva sección llamada Third party apps, para aplicaciones creadas por otros desarrolladores, y añade 'django\_bootstrap5' a esta sección. Asegúrate de colocar esta sección después de My apps pero antes de la sección que contiene las aplicaciones por defecto de Django.

### Utilizar Bootstrap para dar estilo al registro de aprendizaje

Bootstrap es una gran colección de herramientas de estilo. También tiene una serie de plantillas que puedes aplicar a tu proyecto para crear un estilo general. Es mucho más fácil utilizar estas plantillas que utilizar herramientas de estilo individuales. Para ver las plantillas que ofrece Bootstrap, ve a <https://getbootstrap.com> y haz clic en **Ejemplos**. Utilizaremos la plantilla *Navbar static*, que proporciona una sencilla barra de navegación superior y un contenedor para el contenido de la página.

La Figura[20-1](#figure20-1) muestra el aspecto que tendrá la página de inicio después de que apliquemos la plantilla de Bootstrap a *base.html* y modifiquemos ligeramente *index.html*.

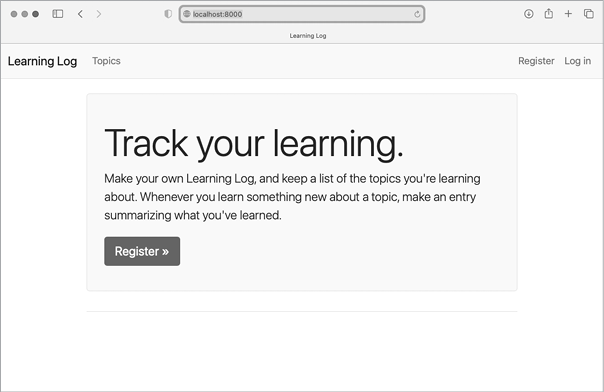


Figura 20-1: Página de inicio del Registro de aprendizaje utilizando Bo

otstrap

### Modificación de base.html

Tenemos que reescribir *base.html* utilizando la plantilla de Bootstrap. Desarrollaremos el nuevo *base.html* por secciones. Se trata de un archivo grande; tal vez quieras copiarlo de los recursos en línea, disponibles en [https://ehmatthes.github.io/pcc\_3e.](https://ehmatthes.github.io/pcc_3e) Si copias el archivo, deberías leer la siguiente sección para comprender los cambios realizados.

#### Definición de las cabeceras HTML

El primer cambio que haremos en *base.html* define las cabeceras HTML del archivo. También añadiremos algunos requisitos para utilizar Bootstrap en nuestras plantillas, y daremos un título a la página. Elimina todo lo que hay en *base.html* y sustitúyelo por el siguiente código:

**base.html**

❶ <!doctype html>  
❷ <html lang="en">  
❸ <head>  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
❹ <title>Learning Log</title>  
  
❺ {% load django\_bootstrap5 %}  
 {% bootstrap\_css %}  
 {% bootstrap\_javascript %}  
  
</head>

Primero declaramos este archivo como un documento HTML ❶ escrito en inglés ❷. Un archivo HTML se divide en dos partes principales: la *head* y la *body*. La cabecera del archivo comienza con una etiqueta de apertura <head> ❸. La cabecera de un archivo HTML no contiene nada del contenido de la página; sólo indica al navegador lo que necesita saber para mostrar la página correctamente. Incluimos un elemento <title> para la página, que se mostrará en la barra de título del navegador siempre que Learning Log esté abierto ❹.

Antes de cerrar la sección head, cargamos la colección de etiquetas de plantilla disponibles en django-bootstrap5 ❺. La etiqueta de plantilla {% bootstrap\_css %} es una etiqueta personalizada de django-bootstrap5; carga todos los archivos CSS necesarios para implementar los estilos de Bootstrap. La etiqueta que sigue habilita todo el comportamiento interactivo que podrías utilizar en una página, como las barras de navegación plegables. La etiqueta de cierre </head> aparece en la última línea.

Todas las opciones de estilo de Bootstrap están ahora disponibles en cualquier plantilla que herede de *base.html*. Si quieres utilizar etiquetas de plantilla personalizadas de django-bootstrap5, cada plantilla deberá incluir la etiqueta {% load django\_bootstrap5 %}.

#### Definir la barra de navegación

El código que define la barra de navegación en la parte superior de la página es bastante largo, porque tiene que funcionar igual de bien en las pantallas estrechas de los teléfonos y en las anchas de los monitores de escritorio. Trabajaremos la barra de navegación por secciones.

Aquí está la primera parte de la barra de navegación:

**base.html**

--snip--  
</head>  
<body>  
  
❶ <nav class="navbar navbar-expand-md navbar-light bg-light mb-4 border">  
 <div class="container-fluid">  
❷ <a class="navbar-brand" href="{% url 'learning\_logs:index' %}">  
 Learning Log</a>  
  
❸ <button class="navbar-toggler" type="button" data-bs-toggle="collapse"  
 data-bs-target="#navbarCollapse" aria-controls="navbarCollapse"  
 aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
  
❹ <div class="collapse navbar-collapse" id="navbarCollapse">  
❺ <ul class="navbar-nav me-auto mb-2 mb-md-0">  
❻ <li class="nav-item">  
❼ <a class="nav-link" href="{% url 'learning\_logs:topics' %}">  
 Topics</a></li>  
 </ul> <!-- End of links on left side of navbar -->  
 </div> <!-- Closes collapsible parts of navbar -->  
  
 </div> <!-- Closes navbar's container -->  
 </nav> <!-- End of navbar -->  
  
❽ {% block content %}{% endblock content %}  
  
</body>  
</html>

El primer elemento nuevo es la etiqueta de apertura <body>. La etiqueta *body* de un archivo HTML contiene el contenido que los usuarios verán en una página. A continuación tenemos un elemento <nav>, que abre el código de la barra de navegación en la parte superior de la página ❶. Todo lo que contiene este elemento está estilizado según las reglas de estilo de Bootstrap definidas por los selectores navbar, navbar-expand-md, y el resto que ves aquí. Un *selector* determina a qué elementos de una página se aplica una determinada regla de estilo. Los selectores navbar-light y bg-light dan estilo a la barra de navegación con un fondo de tema claro. El mb en mb-4 es la abreviatura de *margin-bottom*; este selector garantiza que aparezca un pequeño espacio entre la barra de navegación y el resto de la página. El selector border proporciona un fino borde alrededor del fondo claro para diferenciarlo un poco del resto de la página.

La etiqueta <div> de la línea siguiente abre un contenedor redimensionable que contendrá la barra de navegación general. El término *div* es la abreviatura de *division*; construyes una página web dividiéndola en secciones y definiendo reglas de estilo y comportamiento que se aplican a cada sección. Cualquier regla de estilo o comportamiento que se defina en una etiqueta de apertura <div> afectará a todo lo que veas hasta su correspondiente etiqueta de cierre, escrita como </div>.

A continuación establecemos el nombre del proyecto, Learning Log, para que aparezca como primer elemento de la barra de navegación ❷. Esto también servirá como enlace a la página de inicio, tal y como ha estado haciendo en la versión mínimamente estilizada del proyecto que construimos en los dos capítulos anteriores. El selector navbar-brand da estilo a este enlace para que destaque del resto de enlaces y ayude a añadir algo de marca al sitio.

A continuación, la plantilla Bootstrap define un botón que aparece si la ventana del navegador es demasiado estrecha para mostrar toda la barra de navegación en horizontal ❸. Cuando el usuario pulsa el botón, los elementos de navegación aparecen en una lista desplegable. La referencia collapse hace que la barra de navegación se contraiga cuando el usuario encoge la ventana del navegador o cuando el sitio se muestra en dispositivos con pantallas pequeñas.

A continuación, abrimos una nueva sección (<div>) de la barra de navegación ❹. Esta es la parte de la barra de navegación que puede colapsarse en función del tamaño de la ventana del navegador.

Bootstrap define los elementos de navegación como elementos de una lista desordenada ❺, con reglas de estilo que hacen que no se parezca en nada a una lista. Cada enlace o elemento que necesites en la barra puede incluirse como elemento en una lista desordenada ❻. Aquí, el único elemento de la lista es nuestro enlace a la página de temas ❼. Fíjate en la etiqueta de cierre </li> al final del enlace; toda etiqueta de apertura necesita su correspondiente etiqueta de cierre.

El resto de las líneas que se muestran aquí cierran todas las etiquetas que se han abierto. En HTML, un comentario se escribe así:

<!-- This is an HTML comment. -->

Las etiquetas de cierre no suelen tener comentarios, pero si eres nuevo en HTML, puede ser realmente útil etiquetar algunas de tus etiquetas de cierre. Una sola etiqueta que falte o una etiqueta de más puede desvirtuar el diseño de toda una página. Incluimos el bloque content ❽ y también las etiquetas de cierre </body> y </html>.

No hemos terminado con la barra de navegación, pero ahora tenemos un documento HTML completo. Si runserver está actualmente activo, detén el servidor actual y reinícialo. Ve a la página de inicio del proyecto, y deberías ver una barra de navegación que tiene algunos de los elementos mostrados en la [Figura 20-1](#figure20-1). Ahora vamos a añadir el resto de elementos a la barra de navegación.

#### Añadir enlaces a cuentas de usuario

Todavía tenemos que añadir los enlaces asociados a las cuentas de usuario. Empezaremos por añadir todos los enlaces relacionados con las cuentas, excepto el formulario de cierre de sesión.

Haz los siguientes cambios en *base.html*:

**base.html**

--snip--  
 </ul> <!-- End of links on left side of navbar -->  
  
 <!-- Account-related links -->  
❶ <ul class="navbar-nav ms-auto mb-2 mb-md-0">  
  
❷ {% if user.is\_authenticated %}  
 <li class="nav-item">  
❸ <span class="navbar-text me-2">Hello, {{ user.username }}.  
 </span></li>  
❹ {% else %}  
 <li class="nav-item">  
 <a class="nav-link" href="{% url 'accounts:register' %}">  
 Register</a></li>  
 <li class="nav-item">  
 <a class="nav-link" href="{% url 'accounts:login' %}">  
 Log in</a></li>  
 {% endif %}  
  
 </ul> <!-- End of account-related links -->  
  
 </div> <!-- Closes collapsible parts of navbar -->  
 --snip--

Comenzamos un nuevo grupo de enlaces utilizando otra etiqueta de apertura <ul> ❶. Puedes tener tantos grupos de enlaces como necesites en una página. El selector ms-auto es la abreviatura de *margin-start-automatic*: este selector examina los demás elementos de la barra de navegación y elabora un margen izquierdo (de inicio) que empuja este grupo de enlaces hacia el lado derecho de la ventana del navegador.

El bloque if es el mismo bloque condicional que utilizamos antes para mostrar mensajes adecuados a los usuarios, dependiendo de si están conectados ❷. Ahora el bloque es un poco más largo porque hay algunas reglas de estilo dentro de las etiquetas condicionales. El saludo para los usuarios autentificados se envuelve en un elemento <span> ❸. Un *span element* estiliza trozos de texto o elementos de una página que forman parte de una línea más larga. Mientras que los elementos div crean sus propias divisiones en una página, los elementos span son continuos dentro de una sección mayor. Esto puede resultar confuso al principio, porque muchas páginas tienen elementos div profundamente anidados. Aquí, estamos utilizando el elemento span para dar estilo al texto informativo de la barra de navegación: en este caso, el nombre del usuario conectado.

En el bloque else, que se ejecuta para los usuarios no autentificados, incluimos los enlaces para registrar una nueva cuenta e iniciar sesión ❹. Deben tener el mismo aspecto que el enlace a la página de temas.

Si quisieras añadir más enlaces a la barra de navegación, añadirías otro elemento <li> a uno de los grupos <ul> que hemos definido, utilizando directivas de estilo como las que has visto aquí.

Ahora vamos a añadir el formulario de cierre de sesión a la barra de navegación.

#### Añadir el formulario de cierre de sesión a la barra de navegación

Cuando escribimos por primera vez el formulario de cierre de sesión, lo añadimos a la parte inferior de *base.html*. Ahora vamos a ponerlo en un lugar mejor, en la barra de navegación:

**base.html**

--snip--  
 </ul> <!-- End of account-related links -->  
  
 {% if user.is\_authenticated %}  
 <form action="{% url 'accounts:logout' %}" method='post'>  
 {% csrf\_token %}  
❶ <button name='submit' class='btn btn-outline-secondary btn-sm'>  
 Log out</button>  
 </form>  
 {% endif %}  
  
 </div> <!-- Closes collapsible parts of navbar -->  
 --snip--

El formulario de cierre de sesión debe colocarse después del conjunto de enlaces relacionados con la cuenta, pero dentro de la sección plegable de la barra de navegación. El único cambio en el formulario es la adición de varias clases de estilo Bootstrap en el elemento <button>, que aplican elementos de estilo Bootstrap al botón de cierre de sesión ❶.

Recarga la página de inicio, y deberías poder iniciar y cerrar sesión utilizando cualquiera de las cuentas que hayas creado.

Aún tenemos que añadir algo más a *base.html*. Tenemos que definir dos bloques que las páginas individuales pueden utilizar para colocar el contenido específico de esas páginas.

#### Definir la parte principal de la página

El resto de *base.html* contiene la parte principal de la página:

**base.html**

--snip--  
 </nav> <!-- End of navbar -->  
  
❶ <main class="container">  
❷ <div class="pb-2 mb-2 border-bottom">  
 {% block page\_header %}{% endblock page\_header %}  
 </div>  
❸ <div>  
 {% block content %}{% endblock content %}  
 </div>  
 </main>  
  
</body>  
</html>

Primero abrimos una etiqueta <main> ❶. El elemento *main* se utiliza para la parte más significativa del cuerpo de una página. Aquí asignamos el selector bootstrap container, que es una forma sencilla de agrupar elementos en una página. Colocaremos dos elementos div en este contenedor.

El primer elemento div contiene un bloque page\_header ❷. Utilizaremos este bloque para titular la mayoría de las páginas. Para que esta sección destaque del resto de la página, colocaremos algo de relleno debajo del encabezamiento. *Padding* se refiere al espacio entre el contenido de un elemento y su borde. El selector pb-2 es una directiva de bootstrap que proporciona una cantidad moderada de relleno en la parte inferior del elemento estilizado. Un *margin* es el espacio entre el borde de un elemento y otros elementos de la página. El selector mb-2 proporciona una cantidad moderada de margen en la parte inferior de este div. Queremos un borde en la parte inferior de este bloque, así que utilizamos el selector border-bottom, que proporciona un borde fino en la parte inferior del bloque page\_header.

A continuación, definimos un elemento div más que contiene el bloque content ❸. No aplicamos ningún estilo específico a este bloque, por lo que podemos dar estilo al contenido de cualquier página como creamos conveniente para esa página. Al final del archivo *base.html* hay etiquetas de cierre para los elementos main, body y html.

Cuando cargues la página de inicio de Learning Log en un navegador, deberías ver una barra de navegación de aspecto profesional que coincida con la que se muestra en la [Figura 20-1](#figure20-1). Prueba a cambiar el tamaño de la ventana para que sea muy estrecha; un botón debería sustituir a la barra de navegación. Haz clic en el botón, y todos los enlaces deberían aparecer en una lista desplegable.

### Estilizar la página de inicio con una jumbotrona

Para actualizar la página de inicio, utilizaremos un elemento de Bootstrap llamado *jumbotron*, una gran caja que destaca del resto de la página. Normalmente, se utiliza en las páginas de inicio para contener una breve descripción del proyecto general y una llamada a la acción que invite al espectador a participar.

Aquí tienes el archivo *index.html* revisado:

**index.html**

{% extends "learning\_logs/base.html" %}  
  
❶ {% block page\_header %}  
❷ <div class="p-3 mb-4 bg-light border rounded-3">  
 <div class="container-fluid py-4">  
❸ <h1 class="display-3">Track your learning.</h1>  
  
❹ <p class="lead">Make your own Learning Log, and keep a list of the  
 topics you're learning about. Whenever you learn something new  
 about a topic, make an entry summarizing what you've learned.</p>  
  
❺ <a class="btn btn-primary btn-lg mt-1"  
 href="{% url 'accounts:register' %}">Register &raquo;</a>  
 </div>  
 </div>  
{% endblock page\_header %}

Primero le decimos a Django que vamos a definir lo que va en el bloque page\_header ❶. Un jumbotrón se implementa como un par de elementos div a los que se aplica un conjunto de directivas de estilo ❷. El div exterior tiene ajustes de relleno y margen, un color de fondo claro y esquinas redondeadas. El div interior es un contenedor que cambia con el tamaño de la ventana y también tiene algo de relleno. El selector py-4 añade relleno a la parte superior e inferior del elemento div. Siéntete libre de ajustar los números de estas opciones y observa cómo cambia la página de inicio.

Dentro del jumbotrón hay tres elementos. El primero es un mensaje breve, Track your learning, que da a los nuevos visitantes una idea de lo que hace Learning Log ❸. El elemento <h1> es un encabezado de primer nivel, y el selector display-3 añade un aspecto más fino y alto a este encabezado en particular. También incluimos un mensaje más largo que proporciona más información sobre lo que el usuario puede hacer con su Registro de aprendizaje ❹. Está formateado como un párrafo lead, que pretende destacarse de los párrafos normales.

En lugar de utilizar sólo un enlace de texto, creamos un botón que invita a los usuarios a registrar una cuenta en Learning Log ❺. Se trata del mismo enlace que en la cabecera, pero el botón destaca en la página y muestra al espectador lo que tiene que hacer para empezar a utilizar el proyecto. Los selectores que ves aquí estilizan esto como un botón grande que representa una llamada a la acción. El código &raquo; es un *HTML entity* que se parece a dos corchetes combinados (>>). Por último, proporcionamos etiquetas div de cierre y cerramos el bloque page\_header. Con sólo dos elementos div en este archivo, no es especialmente útil etiquetar las etiquetas div de cierre. No vamos a añadir nada más a esta página, así que no necesitamos definir el bloque content en esta plantilla.

La página de inicio tiene ahora el aspecto de [la Figura 20-1](#figure20-1). ¡Se trata de una mejora significativa con respecto a la versión sin estilo del proyecto!

### Estilizar la página de inicio de sesión

Hemos refinado el aspecto general de la página de inicio de sesión, pero el propio formulario de inicio de sesión aún no tiene ningún estilo. Hagamos que el formulario tenga un aspecto coherente con el resto de la página modificando *login.html*:

**login.html**

{% extends 'learning\_logs/base.html' %}  
❶ {% load django\_bootstrap5 %}  
  
❷ {% block page\_header %}  
 <h2>Log in to your account.</h2>  
{% endblock page\_header %}  
  
{% block content %}  
  
 <form action="{% url 'accounts:login' %}" method='post'>  
 {% csrf\_token %}  
❸ {% bootstrap\_form form %}  
❹ {% bootstrap\_button button\_type="submit" content="Log in" %}  
 </form>  
  
{% endblock content %}

Primero cargamos las etiquetas de la plantilla bootstrap5 en esta plantilla ❶. A continuación, definimos el bloque page\_header, que indica al usuario de qué página se trata ❷. Observa que hemos eliminado el bloque {% if form.errors %} de la plantilla; django-bootstrap5 gestiona los errores de formulario automáticamente.

Para mostrar el formulario, utilizamos la etiqueta de plantilla {% bootstrap\_form %} ❸; ésta sustituye al elemento {{ form.as\_div }} que utilizábamos en el capítulo 19. La etiqueta de plantilla {% booststrap\_form %} inserta las reglas de estilo de Bootstrap en los elementos individuales del formulario a medida que éste se muestra. Para generar el botón de envío, utilizamos la etiqueta {% bootstrap\_button %} con argumentos que lo designan como botón de envío, y le damos la etiqueta Log in ❹.

[La Figura 20-2](#figure20-2) muestra ahora el formulario de acceso. La página está mucho más limpia, con un estilo coherente y un propósito claro. Prueba a iniciar sesión con un nombre de usuario o contraseña incorrectos; verás que incluso los mensajes de error tienen un estilo coherente y se integran bien con el conjunto del sitio.

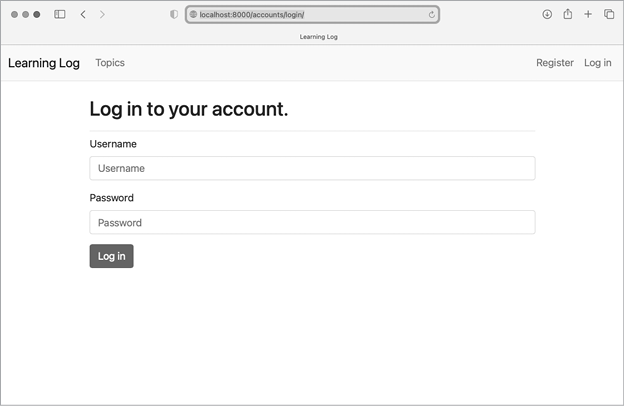


Figura 20-2: La página de inicio de sesión estilizada con Bootstrap

### Estilizar la página de temas

Asegurémonos de que las páginas para ver la información también tienen el estilo adecuado, empezando por la página de temas:

**topics.html**

{% extends 'learning\_logs/base.html' %}  
  
{% block page\_header %}  
❶ <h1>Topics</h1>  
{% endblock page\_header %}  
  
{% block content %}  
  
❷ <ul class="list-group border-bottom pb-2 mb-4">  
 {% for topic in topics %}  
❸ <li class="list-group-item border-0">  
 <a href="{% url 'learning\_logs:topic' topic.id %}">  
 {{ topic.text }}</a>  
 </li>  
 {% empty %}  
❹ <li class="list-group-item border-0">No topics have been added yet.</li>  
 {% endfor %}  
 </ul>  
  
 <a href="{% url 'learning\_logs:new\_topic' %}">Add a new topic</a>  
  
{% endblock content %}

No necesitamos la etiqueta {% load bootstrap5 %}, porque no estamos utilizando ninguna etiqueta de plantilla personalizada de bootstrap5 en este archivo. Movemos el encabezamiento Topics al bloque page\_header y lo convertimos en un elemento <h1> en lugar de un simple párrafo ❶.

El contenido principal de esta página es una lista de temas, así que utilizamos el componente *list group* de Bootstrap para representar la página. Esto aplica un conjunto sencillo de directivas de estilo a la lista general y a cada elemento de la lista. Cuando abrimos la etiqueta <ul>, incluimos primero la clase list-group para aplicar las directivas de estilo por defecto a la lista ❷. Personalizamos aún más la lista poniendo un borde en la parte inferior de la lista, un poco de relleno debajo de la lista (pb-2), y un margen debajo del borde inferior (mb-4).

Cada elemento de la lista necesita la clase list-group-item, y personalizamos el estilo por defecto eliminando el borde alrededor de los elementos individuales ❸. El mensaje que se muestra cuando la lista está vacía necesita estas mismas clases ❹.

Ahora, cuando visites la página de temas, deberías ver una página con un estilo que coincide con el de la página de inicio.

### Estilizar las entradas de la página de temas

En la página de temas, utilizaremos el componente de tarjeta de Bootstrap para hacer que cada entrada destaque. Un *card* es un conjunto anidable de divs con estilos flexibles y predefinidos que son perfectos para mostrar las entradas de un tema:

**topic.html**

{% extends 'learning\_logs/base.html' %}  
  
❶ {% block page\_header %}  
 <h1>{{ topic.text }}</h1>  
{% endblock page\_header %}  
  
{% block content %}  
 <p>  
 <a href="{% url 'learning\_logs:new\_entry' topic.id %}">Add new entry</a>  
 </p>  
  
 {% for entry in entries %}  
❷ <div class="card mb-3">  
 <!-- Card header with timestamp and edit link -->  
❸ <h4 class="card-header">  
 {{ entry.date\_added|date:'M d, Y H:i' }}  
❹ <small><a href="{% url 'learning\_logs:edit\_entry' entry.id %}">  
 edit entry</a></small>  
 </h4>  
 <!-- Card body with entry text -->  
❺ <div class="card-body">{{ entry.text|linebreaks }}</div>  
 </div>  
 {% empty %}  
❻ <p>There are no entries for this topic yet.</p>  
 {% endfor %}  
  
{% endblock content %}

Primero colocamos el tema en el bloque page\_header ❶. A continuación, eliminamos la estructura de lista desordenada utilizada anteriormente en esta plantilla. En lugar de hacer de cada entrada un elemento de lista, abrimos un elemento div con el selector card ❷. Esta ficha tiene dos elementos anidados: uno para contener la marca de tiempo y el enlace para editar la entrada, y otro para contener el cuerpo de la entrada. El selector card se encarga de la mayor parte del estilo que necesitamos para este div; personalizamos la tarjeta añadiendo un pequeño margen en la parte inferior de cada tarjeta (mb-3).

El primer elemento de la tarjeta es una cabecera, que es un elemento <h4> con el selector card-header ❸. Esta cabecera contiene la fecha en que se hizo la entrada y un enlace para editarla. La etiqueta <small> alrededor del enlace edit\_entry hace que aparezca un poco más pequeño que la marca de tiempo ❹. El segundo elemento es un div con el selector card-body ❺, que coloca el texto de la entrada en una simple caja en la tarjeta. Observa que el código Django para incluir la información en la página no ha cambiado; sólo lo han hecho los elementos que afectan a la apariencia de la página. Como ya no tenemos una lista desordenada, hemos sustituido las etiquetas de elemento de lista alrededor del mensaje de lista vacía por simples etiquetas de párrafo ❻.

La[Figura 20-3](#figure20-3) muestra la página de temas con su nuevo aspecto. La funcionalidad de Learning Log no ha cambiado, pero tiene un aspecto mucho más profesional y atractivo para los usuarios.

Si quieres utilizar una plantilla Bootstrap diferente para un proyecto, sigue un proceso similar al que hemos hecho hasta ahora en este capítulo. Copia la plantilla que quieras utilizar en *base.html*, y modifica los elementos que contienen contenido real para que la plantilla muestre la información de tu proyecto. A continuación, utiliza las herramientas de estilo individuales de Bootstrap para dar estilo al contenido de cada página.

## Nota

El proyecto Bootstrap tiene una documentación excelente. Visita la página principal en <https://getbootstrap.com> y haz clic en **Docs** para saber más sobre lo que ofrece Bootstrap.

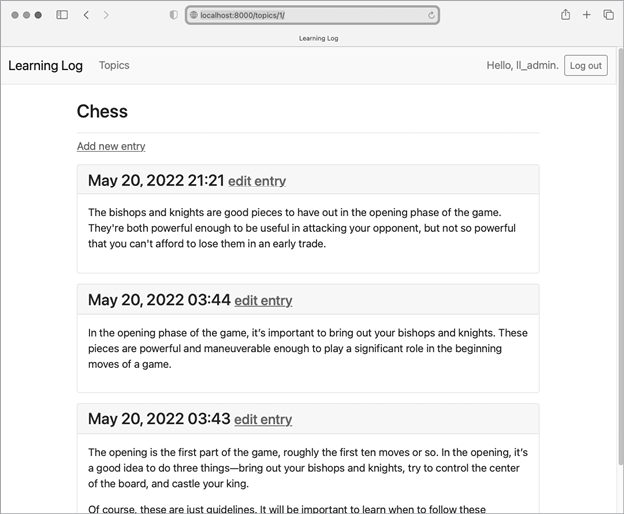


Figura 20-3: Página del tema con

estilo Bootstrap

## Pruébalo tú mismo

20-1. Otros formularios: Hemos aplicado los estilos de Bootstrap a la página login. Haz cambios similares en el resto de las páginas basadas en formularios, incluyendo new\_topic, new\_entry, edit\_entry, y register.

20-2. Blog con estilo: Utiliza Bootstrap para dar estilo al proyecto Blog que creaste en el Capítulo 19.

## Despliegue del Registro de Aprendizaje

Ahora que tenemos un proyecto de aspecto profesional, vamos a desplegarlo en un servidor en vivo para que cualquier persona con conexión a Internet pueda utilizarlo. Utilizaremos Platform.sh, una plataforma basada en web que te permite gestionar el despliegue de aplicaciones web. Pondremos en marcha Learning Log en Platform.sh.

### Crear una cuenta en Platform.sh

Para crear una cuenta, ve a <https://platform.sh> y haz clic en el botón **Prueba gratuita**. Platform.sh tiene un nivel gratuito que, en el momento de escribir este artículo, no requiere tarjeta de crédito. El periodo de prueba te permite desplegar una aplicación con recursos mínimos, lo que te permite probar tu proyecto en un despliegue en vivo antes de comprometerte con un plan de alojamiento de pago.

## Nota

Los límites específicos de los planes de prueba suelen cambiar periódicamente, ya que las plataformas de alojamiento luchan contra el spam y el abuso de recursos. Puedes ver los límites actuales de la prueba gratuita en [https://platform.sh/free-trial.](https://platform.sh/free-trial)

### Instalación de la CLI Platform.sh

Para desplegar y gestionar un proyecto en Platform.sh, necesitarás las herramientas disponibles en la interfaz de línea de comandos (CLI). Para instalar la última versión de la CLI, visita <https://docs.platform.sh/development/cli.html> y sigue las instrucciones correspondientes a tu sistema operativo.

En la mayoría de los sistemas, puedes instalar la CLI ejecutando el siguiente comando en un terminal:

$ curl -fsS https://platform.sh/cli/installer | php

Una vez finalizada la ejecución de este comando, deberás abrir una nueva ventana de terminal antes de poder utilizar la CLI.

## Nota

Es probable que este comando no funcione en un terminal estándar en Windows. Puedes utilizar el Subsistema de Windows para Linux (WSL) o un terminal Git Bash. Si necesitas instalar PHP, puedes utilizar el instalador XAMPP de [https://apachefriends.org.](https://apachefriends.org) Si tienes alguna dificultad para instalar la CLI Platform.sh, consulta las instrucciones de instalación más detalladas del Apéndice E.

### Instalación de platformshconfig

También necesitarás instalar un paquete adicional, platformshconfig. Este paquete ayuda a detectar si el proyecto se está ejecutando en tu sistema local o en un servidor Platform.sh. En un entorno virtual activo, emite el siguiente comando:

(ll\_env)learning\_log$ pip install platformshconfig

Utilizaremos este paquete para modificar la configuración del proyecto cuando se esté ejecutando en el servidor activo.

### Crear un archivo requirements.txt

El servidor remoto necesita saber de qué paquetes depende Learning Log, así que utilizaremos pip para generar un archivo que los enumere. De nuevo, desde un entorno virtual activo, emite el siguiente comando:

(ll\_env)learning\_log$ pip freeze > requirements.txt

El comando freeze indica a pip que escriba los nombres de todos los paquetes instalados actualmente en el proyecto en el archivo *requirements.txt*. Abre este archivo para ver los paquetes y números de versión instalados en tu proyecto:

**requirements.txt**

asgiref==3.5.2  
beautifulsoup4==4.11.1  
Django==4.1  
django-bootstrap5==21.3  
platformshconfig==2.4.0  
soupsieve==2.3.2.post1  
sqlparse==0.4.2

Learning Log ya depende de versiones específicas de siete paquetes diferentes, por lo que necesita un entorno adecuado para ejecutarse correctamente en un servidor remoto. (Instalamos tres de estos paquetes manualmente, y cuatro de ellos se instalaron automáticamente como dependencias de estos paquetes).

Cuando despleguemos Learning Log, Platform.sh instalará todos los paquetes enumerados en *requirements.txt*, creando un entorno con los mismos paquetes que estamos utilizando localmente. Por ello, podemos estar seguros de que el proyecto desplegado funcionará igual que en nuestro sistema local. Este enfoque de la gestión de un proyecto es fundamental cuando empiezas a crear y mantener varios proyectos en tu sistema.

## Nota

Si el número de versión de un paquete que aparece en tu sistema difiere del que se muestra aquí, quédate con la versión que tienes en tu sistema.

### Requisitos adicionales para el despliegue

El servidor en vivo requiere dos paquetes adicionales. Estos paquetes se utilizan para servir el proyecto en un entorno de producción, donde muchos usuarios pueden estar haciendo peticiones al mismo tiempo.

En el mismo directorio donde está guardado *requirements.txt*, crea un nuevo archivo llamado *requirements\_remote.txt*. Añádele los dos paquetes siguientes:

**requirements\_remote.txt**

# Requirements for live project.  
gunicorn  
psycopg2

El paquete gunicorn responde a las peticiones a medida que llegan al servidor remoto; éste ocupa el lugar del servidor de desarrollo que hemos estado utilizando localmente. El paquete psycopg2 es necesario para que Django gestione la base de datos Postgres que utiliza Platform.sh. *Postgres* es una base de datos de código abierto muy adecuada para aplicaciones de producción.

### Añadir archivos de configuración

Cada plataforma de alojamiento requiere alguna configuración para que un proyecto se ejecute correctamente en sus servidores. En esta sección, añadiremos tres archivos de configuración:

*.platform.app.yaml* Este es el archivo de configuración principal del proyecto. Indica a Platform.sh qué tipo de proyecto estamos intentando desplegar y qué tipo de recursos necesita nuestro proyecto, e incluye comandos para construir el proyecto en el servidor.

*.platform/routes.yaml* Este archivo define las rutas a nuestro proyecto. Cuando Platform.sh recibe una solicitud, ésta es la configuración que ayuda a dirigir estas solicitudes a nuestro proyecto específico.

*.platform/services.yaml* Este archivo define cualquier servicio adicional que necesite nuestro proyecto.

Todos estos son archivos YAML (YAML Ain't Markup Language). *YAML* es un lenguaje diseñado para escribir archivos de configuración; está hecho para ser leído fácilmente tanto por humanos como por ordenadores. Puedes escribir o modificar a mano un archivo YAML típico, pero un ordenador también puede leerlo e interpretarlo sin ambigüedades.

Los archivos YAML son magníficos para la configuración del despliegue, porque te dan un gran control sobre lo que ocurre durante el proceso de despliegue.

#### Hacer visibles los archivos ocultos

La mayoría de los sistemas operativos ocultan los archivos y carpetas que empiezan por un punto, como *.platform*. Cuando abras un explorador de archivos, no verás este tipo de archivos y carpetas por defecto. Pero como programador, necesitarás verlos. He aquí cómo ver los archivos ocultos, dependiendo de tu sistema operativo:

* En Windows, abre el Explorador de Windows y, a continuación, abre una carpeta como *Desktop*. Haz clic en la pestaña **Ver** y asegúrate de que están marcadas las opciones **Extensiones de nombre de archivo** y **Elementos ocultos**.
* En macOS, puedes pulsar ⌘-SHIFT-. (punto) en cualquier ventana del Finder para ver los archivos y carpetas ocultos.
* En sistemas Linux como Ubuntu, puedes pulsar CTRL-H en cualquier explorador de archivos para ver los archivos y carpetas ocultos. Para que esta configuración sea permanente, abre un explorador de archivos como Nautilus y haz clic en la pestaña de opciones (indicada por tres líneas). Selecciona la casilla **Mostrar** archivos ocultos.

#### El archivo de configuración .platform.app.yaml

El primer archivo de configuración es el más largo, porque controla el proceso general de despliegue. Lo mostraremos por partes; puedes introducirlo a mano en tu editor de texto o descargar una copia de los recursos en línea en [https://ehmatthes.github.io/pcc\_3e.](https://ehmatthes.github.io/pcc_3e)

Aquí tienes la primera parte de *.platform.app.yaml*, que debes guardar en el mismo directorio que *manage.py*:

**.platform.app.yaml**

❶ name: "ll\_project"  
type: "python:3.10"  
  
❷ relationships:  
 database: "db:postgresql"  
  
# The configuration of the app when it's exposed to the web.  
❸ web:  
 upstream:  
 socket\_family: unix  
 commands:  
❹ start: "gunicorn -w 4 -b unix:$SOCKET ll\_project.wsgi:application"  
❺ locations:  
 "/":  
 passthru: true  
 "/static":  
 root: "static"  
 expires: 1h  
 allow: true  
  
# The size of the persistent disk of the application (in MB).  
❻ disk: 512

Cuando guardes este archivo, asegúrate de incluir el punto al principio del nombre del archivo. Si omites el punto, Platform.sh no encontrará el archivo y tu proyecto no se desplegará.

No es necesario que lo entiendas todo en *.platform.app.yaml* en este momento; destacaré las partes más importantes de la configuración. El archivo comienza especificando el name del proyecto, que llamaremos 'll\_project' para ser coherentes con el nombre que utilizamos al iniciar el proyecto ❶. También tenemos que especificar la versión de Python que estamos utilizando (3.10 en el momento de escribir esto). Puedes encontrar una lista de las versiones compatibles en [https://docs.platform.sh/languages/python.html.](https://docs.platform.sh/languages/python.html)

A continuación hay una sección llamada relationships que define otros servicios que necesita el proyecto ❷. Aquí la única relación es con una base de datos Postgres. Después está la sección web ❸. La sección commands:start indica a Platform.sh qué proceso debe utilizar para atender las peticiones entrantes. Aquí estamos especificando que gunicorn gestionará las peticiones ❹. Este comando sustituye al comando python manage.py runserver que hemos estado utilizando localmente.

La sección locations indica a Platform.sh dónde debe enviar las solicitudes entrantes ❺. La mayoría de las peticiones deben pasarse a gunicorn; nuestros archivos *urls.py* dirán a gunicorn exactamente cómo manejar esas peticiones. Las solicitudes de archivos estáticos se gestionarán por separado y se actualizarán una vez cada hora. La última línea muestra que estamos solicitando 512 MB de espacio en disco en uno de los servidores de Platform.sh ❻.

El resto de [.platform.app.yaml](http://.platform.app.yaml) es como sigue:

--snip--  
disk: 512  
  
# Set a local read/write mount for logs.  
❶ mounts:  
 "logs":  
 source: local  
 source\_path: logs  
  
# The hooks executed at various points in the lifecycle of the application.  
❷ hooks:  
 build: |  
❸ pip install --upgrade pip  
 pip install -r requirements.txt  
 pip install -r requirements\_remote.txt  
  
 mkdir logs  
❹ python manage.py collectstatic  
 rm -rf logs  
❺ deploy: |  
 python manage.py migrate

La sección mounts ❶ nos permite definir directorios donde podemos leer y escribir datos mientras se ejecuta el proyecto. Esta sección define un directorio *logs/* para el proyecto desplegado.

La sección hooks ❷ define acciones que se llevan a cabo en varios momentos del proceso de despliegue. En la sección build, instalamos todos los paquetes necesarios para servir el proyecto en el entorno en vivo ❸. También ejecutamos collectstatic ❹, que reúne todos los archivos estáticos necesarios para el proyecto en un solo lugar para que puedan ser servidos de forma eficiente.

Por último, en la sección deploy ❺, especificamos que las migraciones se ejecuten cada vez que se despliegue el proyecto. En un proyecto sencillo, esto no tendrá ningún efecto cuando no haya habido cambios.

Los otros dos archivos de configuración son mucho más cortos; vamos a escribirlos ahora.

#### El archivo de configuración routes.yaml

Un *route* es la ruta que toma una petición cuando es procesada por el servidor. Cuando Platform.sh recibe una solicitud, necesita saber dónde enviarla.

Crea una nueva carpeta llamada *.platform*, en el mismo directorio que *manage.py*. Asegúrate de incluir el punto al principio del nombre. Dentro de esa carpeta, crea un archivo llamado *routes.yaml* e introduce lo siguiente:

**.platform/routes.yaml**

# Each route describes how an incoming URL will be processed by Platform.sh.  
  
"https://{default}/":  
 type: upstream  
 upstream: "ll\_project:http"  
  
"https://www.{default}/":  
 type: redirect  
 to: "https://{default}/"

Este archivo se asegura de que las peticiones como *https://project\_url.com* y *www.project\_url.com* se dirijan todas al mismo lugar.

#### El archivo de configuración services.yaml

Este último archivo de configuración especifica los servicios que nuestro proyecto necesita para funcionar. Guarda este archivo en el directorio *.platform/*, junto a *routes.yaml*:

**.platform/routes.yaml**

# Each service listed will be deployed in its own container as part of your  
# Platform.sh project.  
  
db:  
 type: postgresql:12  
 disk: 1024

Este archivo define un servicio, una base de datos Postgres.

### Modificar settings.py para Platform.sh

Ahora tenemos que añadir una sección al final de *settings.py* para modificar algunos ajustes para el entorno Plataforma.sh. Añade este código al final de *settings.py*:

**settings.py**

--snip--  
# Platform.sh settings.  
❶ from platformshconfig import Config  
  
config = Config()  
❷ if config.is\_valid\_platform():  
❸ ALLOWED\_HOSTS.append('.platformsh.site')  
  
❹ if config.appDir:  
 STATIC\_ROOT = Path(config.appDir) / 'static'  
❺ if config.projectEntropy:  
 SECRET\_KEY = config.projectEntropy  
  
 if not config.in\_build():  
❻ db\_settings = config.credentials('database')  
 DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.postgresql',  
 'NAME': db\_settings['path'],  
 'USER': db\_settings['username'],  
 'PASSWORD': db\_settings['password'],  
 'HOST': db\_settings['host'],  
 'PORT': db\_settings['port'],  
 },  
 }

Normalmente colocamos las declaraciones import al principio de un módulo, pero en este caso, es útil mantener todos los ajustes específicos del remoto en una sola sección. Aquí importamos Config de platformshconfig ❶, que ayuda a determinar los ajustes en el servidor remoto. Sólo modificamos la configuración si el método config.is\_valid\_platform() devuelve True ❷, lo que indica que la configuración se está utilizando en un servidor Platform.sh.

Modificamos ALLOWED\_HOSTS para permitir que el proyecto sea servido por hosts que terminen en *.platformsh.site* ❸. Todos los proyectos desplegados en el nivel gratuito se servirán utilizando este host. Si la configuración se carga en el directorio de la aplicación desplegada ❹, configuramos STATIC\_ROOT para que los archivos estáticos se sirvan correctamente. También establecemos un SECRET\_KEY más seguro en el servidor remoto ❺.

Por último, configuramos la base de datos de producción ❻. Esto sólo se configura si el proceso de compilación ha terminado de ejecutarse y se está sirviendo el proyecto. Todo lo que ves aquí es necesario para que Django pueda hablar con el servidor Postgres que Platform.sh configuró para el proyecto.

### Utilizar Git para rastrear los archivos del proyecto

Como se comentó en el Capítulo 17, Git es un programa de control de versiones que te permite tomar una instantánea del código de tu proyecto cada vez que implementas con éxito una nueva característica. Si algo va mal, puedes volver fácilmente a a la última instantánea funcional de tu proyecto; por ejemplo, si accidentalmente introduces un error mientras trabajas en una nueva función. Cada instantánea se denomina *commit*.

Utilizando Git, puedes probar a implementar nuevas funciones sin preocuparte de romper tu proyecto. Cuando despliegues en un servidor activo, debes asegurarte de que estás desplegando una versión funcional de tu proyecto. Para leer más sobre Git y el control de versiones, consulta el Apéndice D.

#### Instalar Git

Puede que Git ya esté instalado en tu sistema. Para averiguarlo, abre una nueva ventana de terminal y emite el comando git --version:

(ll\_env)learning\_log$ git --version  
git version 2.30.1 (Apple Git-130)

Si aparece un mensaje indicando que Git no está instalado, consulta las instrucciones de instalación en el Apéndice D.

#### Configurar Git

Git hace un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para ello, Git necesita saber tu nombre de usuario y tu correo electrónico. Debes proporcionar un nombre de usuario, pero puedes inventarte un correo electrónico para tus proyectos de prácticas:

(ll\_env)learning\_log$ git config --global user.name "eric"  
(ll\_env)learning\_log$ git config --global user.email "eric@example.com"

Si olvidas este paso, Git te pedirá esta información cuando hagas tu primera confirmación.

#### Ignorar archivos

No necesitamos que Git rastree todos los archivos del proyecto, así que le diremos que ignore algunos archivos. Crea un archivo llamado *.gitignore* en la carpeta que contiene *manage.py*. Observa que este nombre de archivo comienza con un punto y no tiene extensión. Este es el código que va en *.gitignore*:

**.gitignore**

ll\_env/  
\_\_pycache\_\_/  
\*.sqlite3

Le decimos a Git que ignore todo el directorio *ll\_env*, porque podemos volver a crearlo automáticamente en cualquier momento. Tampoco rastreamos el directorio *\_\_pycache\_\_*, que contiene los archivos *.pyc* que se crean automáticamente cuando se ejecutan los archivos *.py*. No hacemos un seguimiento de los cambios en la base de datos local, porque es una mala costumbre: si alguna vez utilizas SQLite en un servidor, podrías sobrescribir accidentalmente la base de datos activa con tu base de datos de prueba local cuando envíes el proyecto al servidor. El asterisco en \*.sqlite3 indica a Git que ignore cualquier archivo que termine con la extensión *.sqlite3*.

## Nota

Si utilizas macOS, añade .DS\_Store a tu archivo *.gitignore*. Se trata de un archivo que almacena información sobre la configuración de carpetas en macOS, y no tiene nada que ver con este proyecto.

#### Confirmar el proyecto

Tenemos que inicializar un repositorio Git para Learning Log, añadir todos los archivos necesarios al repositorio y confirmar el estado inicial del proyecto. He aquí cómo hacerlo:

❶ (ll\_env)learning\_log$ git init  
Initialized empty Git repository in /Users/eric/.../learning\_log/.git/  
❷ (ll\_env)learning\_log$ git add .  
❸ (ll\_env)learning\_log$ git commit -am "Ready for deployment to Platform.sh."  
[main (root-commit) c7ffaad] Ready for deployment to Platform.sh.  
 42 files changed, 879 insertions(+)  
 create mode 100644 .gitignore  
 create mode 100644 .platform.app.yaml  
 --snip--  
 create mode 100644 requirements\_remote.txt  
❹ (ll\_env)learning\_log$ git status  
On branch main  
nothing to commit, working tree clean  
(ll\_env)learning\_log$

Ejecutamos el comando git init para inicializar un repositorio vacío en el directorio que contiene Learning Log ❶. A continuación, utilizamos el comando git add ., que añade todos los archivos que no se están ignorando al repositorio ❷.(No olvides el punto.) A continuación, emitimos el comando git commit -am " mensaje de confirmación": la bandera -a indica a Git que incluya todos los archivos modificados en esta confirmación, y la bandera -m indica a Git que registre un mensaje de registro ❸.

Emitir el comando git status ❹ indica que estamos en la rama *main* y que nuestro árbol de trabajo es *clean*. Este es el estado que querrás ver cada vez que envíes tu proyecto a un servidor remoto.

### Crear un proyecto en Plataforma.sh

Llegados a este punto, el proyecto Learning Log sigue ejecutándose en nuestro sistema local y también está configurado para ejecutarse correctamente en un servidor remoto. Utilizaremos la CLI Plataforma.sh para crear un nuevo proyecto en el servidor y, a continuación, enviaremos nuestro proyecto al servidor remoto.

Asegúrate de que estás en un terminal, en el directorio *learning\_log/*, y emite el siguiente comando:

(ll\_env)learning\_log$ platform login  
Opened URL: http://127.0.0.1:5000  
Please use the browser to log in.  
--snip--  
❶ Do you want to create an SSH configuration file automatically? [Y/n] Y

Este comando abrirá una pestaña del navegador donde podrás iniciar sesión. Una vez iniciada la sesión, puedes cerrar la pestaña del navegador y volver al terminal. Si se te pide que crees un archivo de configuración SSH ❶, introduce Y para que puedas conectarte al servidor remoto más tarde.

Ahora crearemos un proyecto. Hay mucha salida, así que veremos el proceso de creación por secciones. Comienza emitiendo el comando create:

(ll\_env)learning\_log$ platform create  
\* Project title (--title)  
Default: Untitled Project  
❶ > ll\_project  
  
\* Region (--region)  
The region where the project will be hosted  
 --snip--  
 [us-3.platform.sh] Moses Lake, United States (AZURE) [514 gC02eq/kWh]  
❷ > us-3.platform.sh  
\* Plan (--plan)  
Default: development  
Enter a number to choose:  
 [0] development  
 --snip--  
❸ > 0  
  
\* Environments (--environments)  
The number of environments  
Default: 3  
❹ > 3  
  
\* Storage (--storage)  
The amount of storage per environment, in GiB  
Default: 5  
❺ > 5

El primer mensaje nos pide un nombre para el proyecto ❶, así que utilizaremos el nombre ll\_project. El siguiente mensaje nos pregunta en qué región queremos que esté el servidor ❷. Elige el servidor que te quede más cerca; en mi caso, us-3.platform.sh. Para el resto de las preguntas, puedes aceptar los valores predeterminados: un servidor en el plan de desarrollo más bajo ❸, tres entornos para el proyecto ❹, y 5 GB de almacenamiento para el proyecto en general ❺.

Hay tres preguntas más a las que responder:

Default branch (--default-branch)  
The default Git branch name for the project (the production environment)  
Default: main  
❶ > main  
  
Git repository detected: /Users/eric/.../learning\_log  
❷ Set the new project ll\_project as the remote for this repository? [Y/n] Y  
  
The estimated monthly cost of this project is: $10 USD  
❸ Are you sure you want to continue? [Y/n] Y  
  
The Platform.sh Bot is activating your project  
  
 ▀▄ ▄▀  
 █▄█▀███▀█▄█  
 ▀█████████▀  
 ▄▀ ▀▄  
  
The project is now ready!

Un repositorio Git puede tener varias ramas; Platform.sh nos pregunta si la rama por defecto para el proyecto debe ser main ❶. Después nos pregunta si queremos conectar el repositorio local del proyecto con el repositorio remoto ❷. Por último, se nos informa de que este proyecto costará unos 10$ al mes si lo mantenemos en funcionamiento más allá del periodo de prueba gratuito ❸. Si aún no has introducido una tarjeta de crédito, no deberías preocuparte por este coste. Platform.sh simplemente suspenderá tu proyecto si superas los límites de la prueba gratuita sin añadir una tarjeta de crédito.

### Enviar a Plataforma.sh

El último paso antes de ver la versión en vivo del proyecto es empujar nuestro código al servidor remoto. Para ello, ejecuta el siguiente comando:

(ll\_env)learning\_log$ platform push  
❶ Are you sure you want to push to the main (production) branch? [Y/n] Y  
--snip--  
The authenticity of host 'git.us-3.platform.sh (...)' can't be established.  
RSA key fingerprint is SHA256:Tvn...7PM  
❷ Are you sure you want to continue connecting (yes/no/[fingerprint])? Y  
Pushing HEAD to the existing environment main  
 --snip--  
 To git.us-3.platform.sh:3pp3mqcexhlvy.git  
 \* [new branch] HEAD -> main

Cuando emitas el comando platform push, se te pedirá una confirmación más de que quieres empujar el proyecto ❶. También es posible que aparezca un mensaje sobre la autenticidad de Platform.sh, si es la primera vez que te conectas al sitio ❷. Introduce Y para cada uno de estos mensajes, y verás pasar un montón de resultados. Esta salida probablemente te parecerá confusa al principio, pero si algo va mal, te resultará muy útil durante la resolución de problemas. Si hojeas la salida, podrás ver dónde Platform.sh instala los paquetes necesarios, recopila archivos estáticos, aplica migraciones y configura URLs para el proyecto.

## Nota

Puede que veas un error de algo que puedes diagnosticar fácilmente, como una errata en uno de los archivos de configuración. Si esto ocurre, corrige el error en tu editor de texto, guarda el archivo y vuelve a ejecutar el comando git commit. Entonces podrás volver a ejecutar platform push.

### Ver el proyecto en vivo

Una vez completado el push, puedes abrir el proyecto:

(ll\_env)learning\_log$ platform url  
Enter a number to open a URL  
 [0] https://main-bvxea6i-wmye2fx7wwqgu.us-3.platformsh.site/  
 --snip--  
 > 0

El comando platform url enumera las URL asociadas a un proyecto desplegado; se te dará a elegir entre varias URL que son todas válidas para tu proyecto. Elige una y tu proyecto se abrirá en una nueva pestaña del navegador. Tendrá el mismo aspecto que el proyecto que hemos estado ejecutando localmente, pero puedes compartir esta URL con cualquier persona del mundo, y podrá acceder a tu proyecto y utilizarlo.

## Nota

Cuando despliegues tu proyecto utilizando una cuenta de prueba, no te sorprendas si a veces una página tarda más de lo normal en cargarse. En la mayoría de las plataformas de alojamiento, los recursos gratuitos que están inactivos suelen suspenderse y sólo se reinician cuando llegan nuevas solicitudes. La mayoría de las plataformas responden mucho mejor en los planes de alojamiento de pago.

### Perfeccionar el despliegue de Platform.sh

Ahora perfeccionaremos el despliegue creando un superusuario, igual que hicimos localmente. También haremos que el proyecto sea más seguro cambiando la configuración DEBUG a False, para que los mensajes de error no muestren a los usuarios ninguna información adicional que puedan utilizar para atacar al servidor.

#### Crear un superusuario en Plataforma.sh

Se ha creado la base de datos del proyecto en vivo, pero está completamente vacía. Todos los usuarios que hemos creado antes sólo existen en nuestra versión local del proyecto.

Para crear un superusuario en la versión en vivo del proyecto, iniciaremos una sesión SSH (secure socket shell) en la que podremos ejecutar comandos de gestión en el servidor remoto:

(ll\_env)learning\_log$ platform environment:ssh  
  
 \_\_\_ \_ \_ \_\_ \_  
| \_ \ |\_\_ \_| |\_ / \_|\_\_\_ \_ \_ \_ \_\_ \_\_| |\_  
| \_/ / \_` | \_| \_/ \_ \ '\_| ' \ \_(\_-< ' \  
|\_| |\_\\_\_,\_|\\_\_|\_| \\_\_\_/\_| |\_|\_|\_(\_)\_\_/\_||\_|  
  
 Welcome to Platform.sh.  
  
❶ web@ll\_project.0:~$ ls  
accounts learning\_logs ll\_project logs manage.py requirements.txt  
 requirements\_remote.txt static  
❷ web@ll\_project.0:~$ python manage.py createsuperuser  
❸ Username (leave blank to use 'web'): ll\_admin\_live  
Email address:  
Password:  
Password (again):  
Superuser created successfully.  
❹ web@ll\_project.0:~$ exit  
logout  
Connection to ssh.us-3.platform.sh closed.  
❺ (ll\_env)learning\_log$

Cuando ejecutes por primera vez el comando platform environment:ssh, puede que aparezca otro mensaje sobre la autenticidad de este host. Si ves este mensaje, introduce Y y deberías acceder a una sesión de terminal remota.

Después de ejecutar el comando ssh, tu terminal actúa igual que un terminal en el servidor remoto. Observa que tu prompt ha cambiado para indicar que estás en una sesión web asociada al proyecto llamado ll\_project ❶. Si ejecutas el comando ls, verás los archivos que se han enviado al servidor Platform.sh.

Emite el mismo comando createsuperuser que utilizamos en el Capítulo 18 ❷. Esta vez, he introducido un nombre de usuario de administrador, ll\_admin\_live, que es distinto del que utilicé localmente ❸. Cuando hayas terminado de trabajar en la sesión de terminal remota, introduce el comando exit ❹. Tu prompt te indicará que estás trabajando de nuevo en tu sistema local ❺.

Ahora puedes añadir */admin/* al final de la URL de la aplicación en vivo e iniciar sesión en el sitio de administración. Si otras personas ya han empezado a utilizar tu proyecto, ¡ten en cuenta que tendrás acceso a todos sus datos! Tómate en serio esta responsabilidad y los usuarios seguirán confiándote sus datos.

## Nota

Los usuarios de Windows utilizarán los mismos comandos que se muestran aquí (como ls en lugar de dir), porque estás ejecutando un terminal Linux a través de una conexión remota.

#### Asegurar el proyecto en vivo

Hay un problema de seguridad evidente en la forma en que se despliega actualmente nuestro proyecto: la configuración DEBUG = True en *settings.py*, que proporciona mensajes de depuración cuando se producen errores. Las páginas de error de Django te proporcionan información de depuración vital cuando estás desarrollando un proyecto; sin embargo, dan demasiada información a los atacantes si las dejas activadas en un servidor activo.

Para ver lo malo que es esto, ve a la página de inicio de tu proyecto desplegado. Accede a la cuenta de un usuario y añade */topics/999/* al final de la URL de la página de inicio. Suponiendo que no hayas hecho miles de temas, deberías ver una página con el mensaje *DoesNotExist at /topics/999/*. Si te desplazas hacia abajo, deberías ver un montón de información sobre el proyecto y el servidor. No querrás que tus usuarios vean esto, y desde luego no querrás que esta información esté disponible para cualquier persona interesada en atacar el sitio.

Podemos evitar que esta información se muestre en el sitio en vivo configurando DEBUG = False en la parte de *settings.py* que sólo se aplica a la versión desplegada del proyecto. De este modo, seguirás viendo la información de depuración localmente, donde esa información es útil, pero no se mostrará en el sitio en vivo.

Abre *settings.py* en tu editor de texto, y añade una línea de código a la parte que modifica la configuración de Platform.sh:

**settings.py**

--snip--  
if config.is\_valid\_platform():  
 ALLOWED\_HOSTS.append('.platformsh.site')  
 DEBUG = False  
 --snip--

Todo el trabajo para establecer la configuración de la versión desplegada del proyecto ha dado sus frutos. Cuando queramos ajustar la versión en vivo del proyecto, sólo tenemos que cambiar la parte pertinente de la configuración que establecimos antes.

#### Confirmar y enviar cambios

Ahora tenemos que confirmar los cambios realizados en *settings.py* y enviar los cambios a Platform.sh. Aquí tienes una sesión de terminal que muestra la primera parte de este proceso:

❶ (ll\_env)learning\_log$ git commit -am "Set DEBUG False on live site."  
[main d2ad0f7] Set DEBUG False on live site.  
 1 file changed, 1 insertion(+)  
❷ (ll\_env)learning\_log$ git status  
On branch main  
nothing to commit, working tree clean  
(ll\_env)learning\_log$

Emitimos el comando git commit con un mensaje de confirmación breve pero descriptivo ❶. Recuerda que la bandera -am se asegura de que Git confirma todos los archivos que han cambiado y registra el mensaje de registro. Git reconoce que un archivo ha cambiado y consigna este cambio en el repositorio.

Ejecutar git status muestra que estamos trabajando en la rama main del repositorio y que ahora no hay nuevos cambios que confirmar ❷. Es importante comprobar el estado antes de enviar a un servidor remoto. Si no ves un estado limpio, entonces algunos cambios no se han confirmado y no se enviarán al servidor. Puedes intentar emitir de nuevo el comando commit; si no estás seguro de cómo resolver el problema, lee el Apéndice D para comprender mejor cómo trabajar con Git.

Ahora vamos a empujar el repositorio actualizado a Plataforma.sh:

(ll\_env)learning\_log$ platform push  
Are you sure you want to push to the main (production) branch? [Y/n] Y  
Pushing HEAD to the existing environment main  
--snip--  
 To git.us-3.platform.sh:wmye2fx7wwqgu.git  
 fce0206..d2ad0f7 HEAD -> main  
(ll\_env)learning\_log$

Platform.sh reconoce que el repositorio se ha actualizado, y reconstruye el proyecto para asegurarse de que se han tenido en cuenta todos los cambios. No reconstruye la base de datos, por lo que no hemos perdido ningún dato.

Para asegurarte de que este cambio ha surtido efecto, visita de nuevo la URL */topics/999/*. Sólo deberías ver el mensaje *Server Error (500)*, sin ningún tipo de información sensible sobre el proyecto.

### Crear páginas de error personalizadas

En el Capítulo 19, configuramos el Registro de Aprendizaje para que devolviera un error 404 si el usuario solicitaba un tema o una entrada que no le pertenecía. Ahora también has visto un error de servidor 500. Un error 404 suele significar que tu código Django es correcto, pero el objeto solicitado no existe. Un error 500 suele significar que hay un error en el código que has escrito, como un error en una función en *views.py*. Actualmente, Django devuelve la misma página de error genérica en ambas situaciones, pero podemos escribir nuestras propias plantillas de páginas de error 404 y 500 que coincidan con la apariencia general de Learning Log. Estas plantillas pertenecen al directorio raíz de plantillas.

#### Crear plantillas personalizadas

En la carpeta *learning\_log*, crea una nueva carpeta llamada *templates*. Luego crea un nuevo archivo llamado *404.html*; la ruta a este archivo debe ser *learning\_log/templates/404.html*. Aquí tienes el código de este archivo:

**404.html**

{% extends "learning\_logs/base.html" %}  
  
{% block page\_header %}  
 <h2>The item you requested is not available. (404)</h2>  
{% endblock page\_header %}

Esta sencilla plantilla proporciona la información genérica de la página de error 404, pero con un estilo acorde con el resto del sitio.

Crea otro archivo llamado *500.html* utilizando el siguiente código:

**500.html**

{% extends "learning\_logs/base.html" %}  
  
{% block page\_header %}  
 <h2>There has been an internal error. (500)</h2>  
{% endblock page\_header %}

Estos nuevos archivos requieren un ligero cambio en *settings.py*.

**settings.py**

--snip--  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [BASE\_DIR / 'templates'],  
 'APP\_DIRS': True,  
 --snip--  
 },  
]  
--snip--

Este cambio indica a Django que busque en el directorio raíz de plantillas las plantillas de páginas de error y cualquier otra plantilla que no esté asociada a una aplicación concreta.

#### Enviar los cambios a Platform.sh

Ahora tenemos que confirmar los cambios que acabamos de hacer y enviarlos a Platform.sh:

❶ (ll\_env)learning\_log$ git add .  
❷ (ll\_env)learning\_log$ git commit -am "Added custom 404 and 500 error pages."  
 3 files changed, 11 insertions(+), 1 deletion(-)  
 create mode 100644 templates/404.html  
 create mode 100644 templates/500.html  
❸ (ll\_env)learning\_log$ platform push  
--snip--  
 To git.us-3.platform.sh:wmye2fx7wwqgu.git  
 d2ad0f7..9f042ef HEAD -> main  
(ll\_env)learning\_log$

Emitimos el comando git add . ❶ porque hemos creado algunos archivos nuevos en el proyecto. Luego confirmamos los cambios ❷ y empujamos el proyecto actualizado a Plataforma.sh ❸.

Ahora, cuando aparezca una página de error, tendrá el mismo estilo que el resto del sitio, lo que facilitará la experiencia del usuario cuando se produzcan errores.

### Desarrollo en curso

Es posible que quieras seguir desarrollando Learning Log después de su lanzamiento inicial en un servidor activo, o que quieras desarrollar tus propios proyectos para desplegarlos. Al hacerlo, existe un proceso bastante coherente para actualizar tus proyectos.

En primer lugar, realizarás los cambios necesarios en tu proyecto local. Si tus cambios dan lugar a algún archivo nuevo, añade esos archivos al repositorio Git utilizando el comando git add . (asegurándote de incluir el punto al final del comando). Cualquier cambio que requiera una migración de la base de datos necesitará este comando, porque cada migración genera un nuevo archivo de migración.

En segundo lugar, confirma los cambios en tu repositorio utilizando git commit -am "mensaje de confirmación". A continuación, empuja tus cambios a Plataforma.sh, utilizando el comando platform push. Visita tu proyecto en vivo y asegúrate de que los cambios que esperas ver han surtido efecto.

Es fácil cometer errores durante este proceso, así que no te sorprendas cuando algo vaya mal. Si el código no funciona, revisa lo que has hecho e intenta detectar el error. Si no encuentras el error o no sabes cómo deshacerlo, consulta las sugerencias para obtener ayuda en el Apéndice C. No seas tímido a la hora de pedir ayuda: todo el mundo aprendió a construir proyectos haciéndose las mismas preguntas que probablemente te harás tú, así que alguien estará encantado de ayudarte. Resolver cada problema que surja te ayudará a desarrollar constantemente tus habilidades hasta que estés construyendo proyectos significativos y fiables y respondiendo también a las preguntas de otras personas.

### Eliminar un proyecto en Plataforma.sh

Es una buena práctica realizar el proceso de despliegue varias veces con el mismo proyecto o con una serie de proyectos pequeños, para cogerle el truco al despliegue. Pero necesitarás saber cómo eliminar un proyecto que se ha desplegado. Platform.sh también limita el número de proyectos que puedes alojar gratuitamente, y no querrás saturar tu cuenta con proyectos de práctica.

Puedes eliminar un proyecto utilizando la CLI:

(ll\_env)learning\_log$ platform project:delete

Se te pedirá que confirmes que deseas realizar esta acción destructiva. Responde a las indicaciones y tu proyecto será eliminado.

El comando platform create también dio al repositorio Git local una referencia al repositorio remoto en los servidores de Platform.sh. También puedes eliminar este remoto desde la línea de comandos:

(ll\_env)learning\_log$ git remote  
platform  
(ll\_env)learning\_log$ git remote remove platform

El comando git remote lista los nombres de todas las URL remotas asociadas al repositorio actual. El comando git remote remove nombre\_remoto elimina estas URL remotas del repositorio local.

También puedes eliminar los recursos de un proyecto iniciando sesión en el sitio web Platform.sh y visitando tu panel de control en [https://console.platform.sh.](https://console.platform.sh) Esta página enumera todos tus proyectos activos. Haz clic en los tres puntos de la casilla de un proyecto, y haz clic en **Editar Plan**. Esta es una página de precios para el proyecto; haz clic en el botón **Eliminar proyecto** en la parte inferior de la página, y se te mostrará una página de confirmación donde podrás seguir adelante con la eliminación. Aunque hayas eliminado tu proyecto utilizando la CLI, es una buena idea que te familiarices con el panel de control de cualquier proveedor de alojamiento en el que te despliegues.

## Nota

Eliminar un proyecto en Platform.sh no hace nada a tu versión local del proyecto. Si nadie ha utilizado tu proyecto desplegado y sólo estás practicando el proceso de despliegue, es perfectamente razonable borrar tu proyecto en Platform.sh y volver a desplegarlo. Sólo ten en cuenta que si las cosas dejan de funcionar, puede que te hayas topado con las limitaciones de nivel libre del host.

## Pruébalo tú mismo

20-3. Blog en vivo: Despliega el proyecto Blog en el que has estado trabajando en Plataforma.sh. Asegúrate de configurar DEBUG en False, para que los usuarios no vean las páginas de error completas de Django cuando algo vaya mal.

20-4.Registro de aprendizaje ampliado: Añade una característica al Registro de Aprendizaje, y empuja el cambio a tu despliegue en vivo. Prueba con un cambio sencillo, como escribir más sobre el proyecto en la página de inicio. Después, prueba a añadir una función más avanzada, como dar a los usuarios la opción de hacer público un tema. Esto requeriría un atributo llamado public como parte del modelo Topic (debería estar configurado como False por defecto) y un elemento de formulario en la página new\_topic que permita al usuario cambiar un tema de privado a público. Entonces tendrías que migrar el proyecto y revisar *views.py* para que cualquier tema que sea público también sea visible para los usuarios no autenticados.

## Resumen

En este capítulo has aprendido a dar a tus proyectos un aspecto sencillo pero profesional utilizando la biblioteca Bootstrap y la aplicación django-bootstrap5. Con Bootstrap, los estilos que elijas funcionarán de forma consistente en casi cualquier dispositivo que la gente utilice para acceder a tu proyecto.

Has aprendido sobre las plantillas de Bootstrap y has utilizado la plantilla *Navbar static* para crear un aspecto sencillo para Learning Log. Utilizaste un jumbotrón para destacar el mensaje de una página de inicio, y aprendiste a aplicar un estilo coherente a todas las páginas de un sitio.

En la parte final del proyecto, aprendiste a desplegar un proyecto en un servidor remoto para que cualquiera pueda acceder a él. Creaste una cuenta Platform.sh e instalaste algunas herramientas que ayudan a gestionar el proceso de despliegue. Utilizaste Git para enviar el proyecto de trabajo a un repositorio, y luego enviaste el repositorio a un servidor remoto en Platform.sh. Por último, aprendiste a proteger tu aplicación configurando DEBUG = False en el servidor activo. También creaste páginas de error personalizadas, para que los inevitables errores que surjan se vean bien gestionados.

Ahora que has terminado el Registro de Aprendizaje, puedes empezar a construir tus propios proyectos. Empieza de forma sencilla y asegúrate de que el proyecto funciona antes de añadirle complejidad. Disfruta de tu aprendizaje continuo, ¡y buena suerte con tus proyectos!