



**APRENDE
DJANGO
DESDE CERO
HASTA AVANZADO**

A C A D E M I A X

Contenido

1 Introducción	5
1.1 Bienvenida	5
1.1.1 Libro vivo	6
1.1.2 Alcance	6
1.2 Prerequisitos	6
1.3 ¿Cómo evitar bloqueos?	8
2 Primeros pasos	8
2.1 ¿Qué es Django?	8
2.2 Instalación de Django	10
2.3 Creación de un proyecto Django	12
2.4 Estructura de un proyecto Django	13
2.5 El servidor de desarrollo	15
2.6 Creación de una aplicación Django	16
2.7 Estructura de una aplicación en Django	18
3 Modelos en Django	20
3.1 El ORM	20
3.2 SQLite	21
3.3 Configuración de base de datos	22
3.4 Creación de modelos	24
3.5 Campos de modelos	25
3.6 Migración	27
3.7 Relaciones entre modelos	29
4 Shell en Django	33
4.1 Guardar modelo en shell	33
4.2 Leer modelo en shell	35
4.3 Filtrar modelo en shell	37

Contenido

4.4	Borrar modelo en shell	38
5	Admin de Django	40
5.1	Acceder al panel	40
5.2	Crear superusuario	40
5.3	Registrar modelo	41
6	APIs en Django	43
6.1	Creación de APIs	43
6.2	Serialización de datos	45
6.3	Parametros en url (str, int, slug, uuid, path)	46
6.4	Parametros en url para consultar modelos	49
6.5	Parametros en peticiones	50
6.6	Funciones de atajo	51
6.6.1	redirect()	51
6.6.2	get_object_or_404()	52
6.6.3	get_list_or_404()	54
7	Vistas en Django	55
7.1	Creación de vistas	55
7.2	Renderizado de plantillas	57
7.3	Manejo de formularios	59
7.4	Redirecciones y URLs	62
7.5	Archivos estáticos	63
7.6	Manejo de errores	65
8	Templates en Django	67
8.1	Variables en plantillas	67
8.2	Filtros en plantillas	70
8.3	Tags de plantillas	72
8.4	Iteración en plantillas	74
8.5	Condicionales en plantillas	75

Contenido

8.6 Herencia de plantillas	77
9 Siguientes pasos	79
9.1 Herramientas	79
9.2 Recursos	80
9.3 ¿Que viene después?	81
9.4 Preguntas de entrevista	82

1 Introducción

1.1 Bienvenida

Bienvenid@ a este libro de Academia X en donde aprenderás Django práctico.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de este libro. He trabajado como consultor en proyectos para Nintendo, Google, entre otros proyectos top-tier, trabajé como líder de un equipo de desarrollo por 3 años, y soy Ingeniero Ex-Amazon. En mis redes me conocen como Programador X y comparto videos semanalmente en YouTube desde diversas locaciones del mundo con el objetivo de guiar y motivar a mis estudiantes mientras trabajo haciendo lo que más me gusta: la programación.

En este libro vas a aprender estos temas:

- Primeros pasos
- Modelos en Django
- Shell en Django
- Admin de Django
- APIs en Django
- Vistas en Django
- Templates en Django

La motivación de este libro es darte todo el conocimiento técnico que necesitas para elevar la calidad de tus proyectos y al mismo tiempo puedas perseguir metas más grandes, ya sea utilizar esta tecnología para tus pasatiempos creativos, aumentar tus oportunidades laborales, o si tienes el espíritu emprendedor, incluso crear tu propio negocio en línea. Confío en que este libro te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

1 INTRODUCCIÓN

1.1.1 Libro vivo

Esta publicación fue planeada, editada, y revisada manualmente por Xavier Reyes Ochoa. La fundación del contenido fue generada parcialmente por inteligencia artificial usando ChatGPT (May 12 Version) de OpenAI. Puedes ver más detalles en <https://openai.com/>

Esto es a lo que llamo un trabajo **VIVO**, esto quiere decir que será actualizado en el tiempo a medida que existan cambios en la tecnología. La versión actual es 1.0.0 editada el 26 de julio de 2023. Si tienes correcciones importantes, puedes escribirnos a nuestra sección de contacto en <https://www.academia-x.com>

1.1.2 Alcance

El objetivo de este libro es llenar el vacío que existe sobre esta tecnología en Español siguiendo el siguiente enfoque:

- Se revizan los temas con un enfoque práctico incluyendo ejemplos y retos.
- Se evita incluir material de relleno ya que no es eficiente.
- Se evita entrar en detalle en temas simples o avanzados no-prácticos.

Si deseas profundizar en algún tema, te dejo varios recursos oficiales, populares, y avanzados en la lección de recursos.

1.2 Prerequisitos

Antes de aprender Django, necesitas lo siguiente:

1. Un computador: cualquier computador moderno tiene las capacidades de correr este lenguaje. Te recomiendo un monitor de escritorio o laptop ya que dispositivos móviles o ipads no son cómodos para programar.

1 INTRODUCCIÓN

2. Sistema operativo: conocimiento básico de cómo utilizar el sistema operativo en el que se ejecutará Django (por ejemplo, Windows, MacOS, Linux). Te recomiendo tener el sistema operativo actualizado.
3. Conocimiento básico de la línea de comandos: se utiliza para ejecutar programas en Django.
4. Un editor de texto: lo necesitas para escribir código de Django. Los editores de texto más populares son Visual Studio Code, Sublime Text, Atom y Notepad ++.
5. Un navegador web y conexión al internet: es útil para investigar más sobre esta tecnología cuando tengas dudas. Los navegadores más populares son Google Chrome, Mozilla Firefox, Safari y Microsoft Edge. Se recomienda tener el navegador actualizado.
6. Conocimientos de programación en Python: es importante tener una comprensión sólida de los conceptos fundamentales de programación, como variables, estructuras de control de flujo (bucles, condicionales), funciones y objetos, ya que Django se basa en Python.
7. Tener conocimientos básicos de desarrollo web: incluyendo el lenguaje de marcado HTML, la hoja de estilos CSS y el lenguaje de programación JavaScript, te ayudará a entender mejor la estructura y la interacción de una aplicación web.
8. Base de datos: tener una comprensión básica de las bases de datos y el lenguaje SQL será beneficioso, ya que Django utiliza modelos para interactuar con bases de datos relacionales.
9. pip3: es necesario tener pip instalado. pip es el sistema de gestión de paquetes de Python que permite instalar, actualizar y administrar bibliotecas y dependencias externas en tus proyectos.

Si ya tienes estos requisitos, estarás en una buena posición para comenzar a aprender Django y profundizar en sus características y aplicaciones.

Si todavía no tienes conocimiento sobre algunos de estos temas, te recomiendo buscar tutoriales básicos en blogs a través de Google, ver videos en YouTube, o

2 PRIMEROS PASOS

hacer preguntas a ChatGPT. Alternativamente, puedes empezar ya e investigar en línea a medida que encuentres bloqueos entendiendo los conceptos en este libro.

1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a este libro:

1. No solo leas este libro. La práctica es esencial en este campo. Practica todos los días y no pases de lección hasta que un concepto esté 100% claro.
2. No tienes que memorizarlo todo, solo tienes que saber donde están los temas para buscarlos rápidamente cuando tengas dudas.
3. Cuando tengas preguntas usa [Google](#), [StackOverFlow](#), y [ChatGPT](#)
4. Acepta que en esta carrera, mucho de tu tiempo lo vas utilizar investigando e innovando, no solo escribiendo código.
5. No tienes que aprender inglés ahora pero considera aprenderlo en un futuro porque los recursos más actualizados están en inglés y también te dará mejores oportunidades laborales.
6. Si pierdas la motivación, recuerda tus objetivos. Ninguna carrera es fácil pero ya tienes los recursos para llegar muy lejos. Te deseo lo mejor en este campo!

2 Primeros pasos

2.1 ¿Qué es Django?

¡Hola! ¿Estás listo para sumergirte en el emocionante mundo del desarrollo web? ¿Quieres construir aplicaciones web de manera fácil y rápida? ¡Entonces Django es la herramienta para ti!

Django es un framework web de alto nivel, escrito en Python, que te permite construir aplicaciones web de manera rápida y fácil. ¿No sabes qué es un framework? Es

2 PRIMEROS PASOS

como una caja de herramientas que contiene todo lo que necesitas para construir una aplicación web, desde la estructura básica hasta las herramientas avanzadas.

Imagínate que estás construyendo una casa desde cero: necesitas planos, materiales, herramientas y mucho tiempo. Pero, ¿qué pasaría si pudieras comenzar con una casa preconstruida? Sería más rápido y fácil agregar tus propias personalizaciones y hacerlo tuyo. Así es como funciona Django: te brinda una estructura preconstruida para comenzar, ahorrándote tiempo y esfuerzo.

Además, Django es altamente escalable y seguro. Puedes construir aplicaciones desde simples blogs hasta complejas plataformas de comercio electrónico. Con características como la autenticación integrada, la gestión de la base de datos y la capacidad de manejar grandes cantidades de tráfico, Django es una de las opciones más populares para construir aplicaciones web hoy en día.

¡Este es un ejemplo de Django que entenderás a detalle más adelante!



A screenshot of a code editor window. The title bar has three colored circles (red, yellow, green). The main area contains the following Python code:

```
from django.shortcuts import render
from django.http import HttpResponse

def bienvenido(request):
    return HttpResponse("¡Bienvenid@ a mi aplicación web
construida con Django!")
```

Como puedes ver, en solo unas pocas líneas, puedes crear una función que devuelve una respuesta personalizada a tu usuario. Ahora eso es potente.

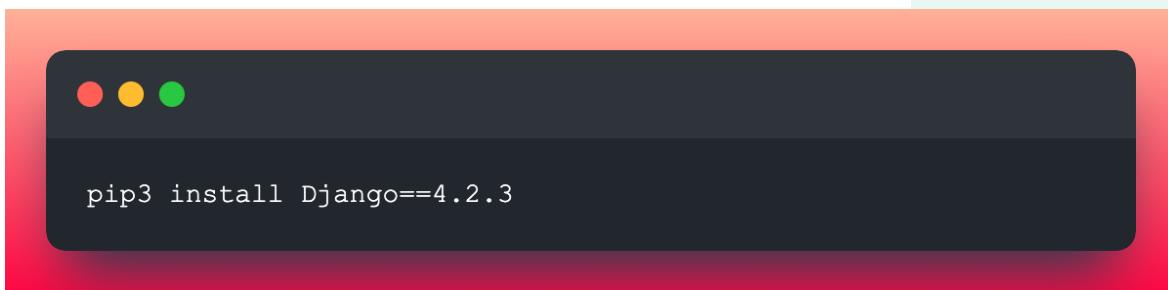
¿Estoy emocionado por enseñarte a construir tus propias aplicaciones web con Django? ¡Empecemos!

2.2 Instalación de Django

¿Estamos listos para aprender cómo instalar Django? ¡Genial! Porque hoy te voy a enseñar paso a paso cómo hacerlo.

Puedes dirigirte a la página oficial de Django (<https://www.djangoproject.com/>) y revisar las instrucciones de descarga de la última versión.

En esta publicación instalaremos la última versión oficial que es la 4.2.3 (LTS). Para esto utilizaremos el gestor de paquetes de Python, pip3:



Cuando se utiliza pip3 para instalar un paquete, descarga los archivos de distribución del paquete y los instala en el entorno de Python asociado a pip3. Los archivos suelen almacenarse en un directorio de site-packages a nivel del sistema o en el directorio de site-packages de un entorno virtual. Esto permite que el paquete sea accesible desde cualquier parte del sistema o dentro del entorno virtual, y no solo en el directorio desde el que se ejecutó pip3.

Una vez que se instale correctamente, puedes verificar la versión de Django instalada con cualquiera de estos comandos:

2 PRIMEROS PASOS

```
python3 -m django --version  
django-admin --version  
pip3 list -v
```

También se puede verificar corriendo este código de Python:

```
import django  
django.get_version()
```

Y listo. ¡Ya tienes Django instalado en tu computadora! Ahora puedes empezar a crear tus proyectos con este framework fantástico.

Reto:

Crea un proyecto Django nuevo utilizando la terminal. No te preocupes si no tienes idea de cómo hacerlo, aquí te dejo una pista:

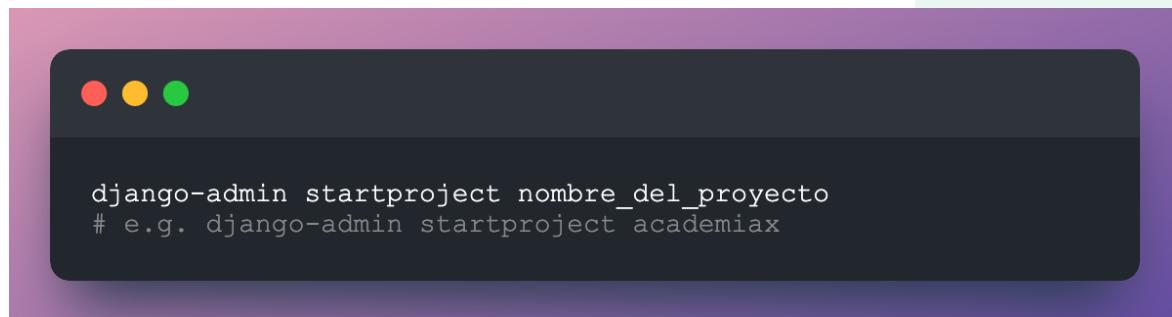
```
django-admin startproject nombre_del_proyecto
```

¡Que la fuerza de Django te acompañe!

2.3 Creación de un proyecto Django

¡Estoy emocionado que te aventures en la creación de tu proyecto en Django!

Ahora bien, para crear un nuevo proyecto Django, simplemente tienes que ir a la carpeta donde deseas crear tu proyecto en tu terminal y utilizar el siguiente comando:



¡Y listo! Con tan solo una línea de código, habrás creado tu proyecto. Pero espera, aún hay más que hacer. Si ingresas a la carpeta de tu nuevo proyecto, verás que ya tienes algunas carpetas y archivos por defecto. En particular, hay una carpeta llamada `nombre_del_proyecto` con el archivo `settings.py` donde se encuentran las configuraciones de tu proyecto.

Es importante que dediques un tiempo a revisar y modificar estas configuraciones para asegurarte de que tu proyecto esté optimizado para tus necesidades. Por ejemplo, puedes especificar la zona horaria de tu proyecto o configurar la conexión a una base de datos.

Ahora bien, aquí va un pequeño reto: agrega una aplicación a tu proyecto Django. ¿Cómo? Muy fácil. Utiliza el siguiente comando en tu terminal:



¡Inténtalo y verás lo fácil que es! Ahora ya sabes que tu proyecto de Django puede tener una o varias aplicaciones y esto lo vamos a ver en las siguientes lecciones.

Recuerda que siempre puedes buscar en la documentación oficial de Django si tienes dudas o necesitas más información. ¡Mucho éxito!

2.4 Estructura de un proyecto Django

Si has llegado hasta aquí, es porque quieres aprender sobre la estructura de un proyecto Django. ¡Genial! Django es un framework web muy poderoso, fácil de aprender y altamente eficiente.

Para empezar, un proyecto Django está compuesto por varios archivos y carpetas organizadas de forma jerárquica. En la carpeta principal del proyecto, tendrás un archivo llamado `manage.py`. Este archivo te permitirá ejecutar comandos desde la terminal que son esenciales para iniciar y manejar tu proyecto.

Además, encontrarás una carpeta principal del proyecto, la cual suele tener el mismo nombre que el proyecto en sí. Esta carpeta contiene otro archivo importante llamado `settings.py`, el cual es la piedra angular del proyecto. Aquí se especifican diferentes atributos que afectan el comportamiento de tu aplicación, como la configuración de la base de datos o los archivos estáticos.

Como programador de Python ya debes saber que el archivo `init.py` es un archivo especial en Python y no específico de Django que se utiliza para indicar que un

2 PRIMEROS PASOS

directorio debe tratarse como un paquete. Este archivo se coloca típicamente dentro de un directorio que contiene los archivos relacionados con un paquete de Python.

Dentro de la carpeta del proyecto, también encontrarás un archivo llamado urls.py, la cual es responsable de manejar la distribución de las URLs en tu aplicación. Aquí mapearás el patrón de URL a una vista que maneja la petición.

Otro archivo importante dentro del proyecto es el wsgi.py, el cual te permite desplegar tu aplicación usando los servidores web:

- Apache: es un software libre y de código abierto que permite a los usuarios desplegar sus sitios web en internet. Es uno de los servidores web más antiguos y confiables. Desplegar Django con Apache y mod_wsgi es una forma probada y comprobada de llevar Django a producción.
- Gunicorn (‘Green Unicorn’): un servidor WSGI escrito en Python puro para UNIX.
- uWSGI: es un servidor contenedor de aplicaciones rápido, autoreparador y amigable para desarrolladores/sysadmins, codificado en C puro.

El archivo asgi.py es un punto de entrada para el servidor ASGI (Asynchronous Server Gateway Interface) que permite manejar aplicaciones web en tiempo real y basadas en eventos, como por ejemplo, chat en línea, transmisión de video, etc. Se puede utilizar en lugar de wsgi.py y al igual que wsgi.py, asgi.py establece la configuración de la aplicación Django y carga los módulos necesarios para que el servidor ASGI pueda servir la aplicación.

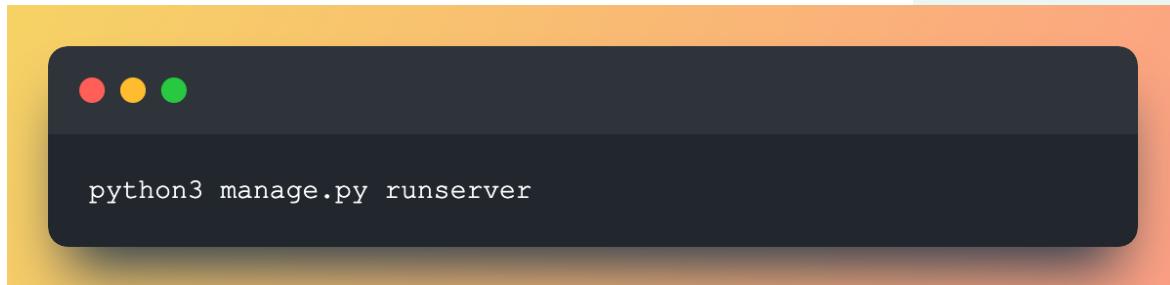
Además, en la carpeta principal del proyecto es donde albergarás tus diferentes aplicaciones que componen el proyecto (como autenticación, perfil de usuario, etc.). Cada aplicación tendrá su propio conjunto de archivos y su propia estructura, similar a la del proyecto.

2.5 El servidor de desarrollo

Si estás aquí es porque deseas saber más acerca del servidor de desarrollo en Django. Parece algo aburrido, ¿no crees? ¡Pero no lo es! ¡Es la herramienta que te permitirá tener una experiencia de desarrollo alucinante!

El servidor de desarrollo es la forma más fácil de comenzar a desarrollar en Django. Te permite probar la funcionalidad de tu aplicación sin tener que preocuparte por realizar una configuración de servidor completa. Además, está diseñado para ser fácil de usar, lo que significa que no es necesario conocer todos los detalles de la configuración del servidor para comenzar a usarlo.

Pero, ¿cómo lo usamos? Es muy sencillo. Lo único que tienes que hacer es abrir una terminal, navegar a la carpeta raíz de tu proyecto Django y escribir el siguiente comando:



¡Y eso es todo! ¡Tu servidor de desarrollo está corriendo ahora mismo en tu computadora! Ahora podrás ver tu aplicación ejecutándose en tu navegador web favorito ingresando la dirección <http://127.0.0.1:8000/>. Vas a ver algunas advertencias relacionadas con migración de base de datos que solucionaremos después.

Como reto, te propongo que intentes cambiar el puerto en el que se ejecuta el servidor. Para ello, puedes agregar un número diferente después del comando runserver. Por ejemplo: `python3 manage.py runserver 8080`.

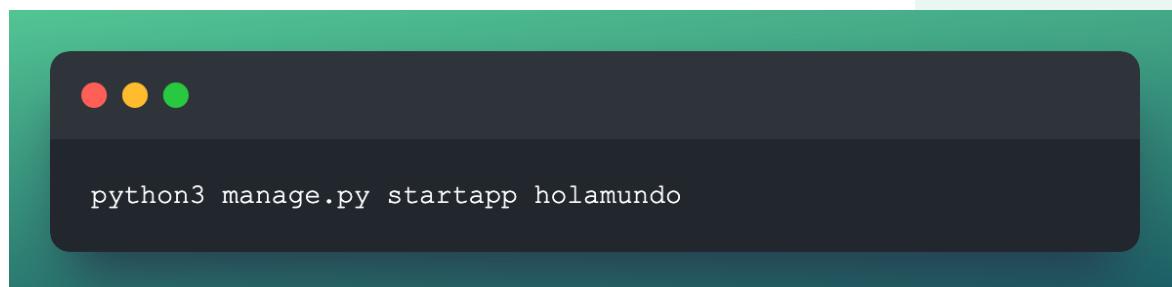
¡Manos a la obra! ¡Te deseo mucha diversión mientras desarrollas en Django!

2.6 Creación de una aplicación Django

¿Estamos listos para ingresar en el asombroso mundo de la creación de aplicaciones web con Django? ¡Pues prepárate para un viaje lleno de desafíos!

Imagínate poder crear una aplicación desde cero. Puede ser una app de blogs, una app de venta de productos, una app de chat, etc. Así es como funcionan las aplicaciones en un proyecto de Django. Django permite crear múltiples secciones modulares en un proyecto y las llama aplicaciones.

Ahora, vamos a la acción. Para comenzar, crea una aplicación dentro del proyecto. Por ejemplo, puedes crear una aplicación que se enfoque tan solo en responder nuestro primer “Hola Mundo” . Corre este comando en la terminal para crea una aplicación con el nombre “holamundo” en nuestro proyecto:



```
python3 manage.py startapp holamundo
```

Ahora, para crear una respuesta con el texto “Hola Mundo” , creamos nuestros archivos de vistas views.py y urls.py para nuestra aplicación de holamundo de esta manera:

2 PRIMEROS PASOS

```
# app/views.py
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Hola mundo!</h1>")

# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
]

# urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("holamundo/", include("holamundo.urls")),
    path("admin/", admin.site.urls),
]
```

¡Genial! Ya tenemos nuestra aplicación en Django. Ahora solo nos queda probarla corriendo el servidor y navegando a la dirección `http://127.0.0.1:8000/holamundo`. Puedes ver que nuestro servidor de Django crea un archivo de HTML y lo sirve en esta dirección.

Y para el reto, te propongo modificar el texto por “Hola universo!” en el archivo `holamundo/views.py` y refrescar el navegador para ver que los cambios se aplican sin reiniciar el programa. ¡Mucho éxito!

2.7 Estructura de una aplicación en Django

¿Quieres aprender sobre la estructura de una aplicación en Django? ¡Genial!

Imagina que una aplicación en Django es como una pizza. La pizza tiene diferentes capas, como la masa, la salsa y los ingredientes. En Django, una aplicación también tiene diferentes capas y hay varios archivos que cumplen roles específicos para diferentes aspectos de la aplicación. Aquí tienes una explicación de los archivos más comunes que puedes encontrar en una aplicación Django:

1. `__pycache__`: Esta carpeta no es específica de Django sino de Python. Contiene archivos de caché byte-compilados generados automáticamente por Python para mejorar el rendimiento de la importación de módulos. No requiere interacción directa y se puede ignorar en el control de versiones.
2. `migrations`: Esta carpeta es generada por Django y se utiliza para almacenar archivos de migración de la base de datos. Estos archivos representan los cambios en la estructura de la base de datos a medida que evoluciona la aplicación. No se deben eliminar ni modificar manualmente, a menos que se tenga un conocimiento preciso de su funcionamiento.
3. `__init__.py`: Este archivo no es específico de Django sino de Python. Este archivo es necesario para que Python trate el directorio que lo contiene como un paquete. Puede estar vacío o contener código de inicialización de la aplicación.
4. `admin.py`: Si deseas utilizar la interfaz de administración de Django para administrar los modelos de datos de tu aplicación, puedes registrar los modelos aquí. Esto te permite gestionar los registros de tu aplicación desde la interfaz de administración sin tener que escribir código adicional.
5. `apps.py`: Este es un archivo de configuración que se encuentra en cada aplicación de Django. Su propósito principal es proporcionar configuración específica de la aplicación, como su nombre legible por humanos y cualquier configu-

2 PRIMEROS PASOS

ración adicional necesaria. Es similar al archivo `settings.py` a nivel de proyecto, pero a nivel de aplicación.

6. `models.py`: Este archivo define los modelos de datos de la aplicación utilizando la API de modelos de Django. Aquí defines las clases de Python que representan las tablas de la base de datos y sus relaciones.
7. `tests.py`: Este es un archivo para escribir pruebas unitarias para tu aplicación.
8. `urls.py`: Este archivo que puedes crear manualmente y contiene las rutas URL de la aplicación. Aquí defines las correspondencias entre las URLs y las vistas que se deben invocar para manejar las solicitudes entrantes. Puedes utilizar expresiones regulares y patrones de URL para mapear las URL a las vistas adecuadas.
9. `views.py`: En este archivo se definen las vistas de la aplicación. Las vistas son funciones o clases que reciben solicitudes HTTP y devuelven respuestas. Puedes definir vistas basadas en funciones o vistas basadas en clases. Ya vimos su uso en el ejemplo de “Hola Mundo” .
10. `forms.py`: Puedes crear manualmente este archivo. En este archivo puedes definir formularios de Django que se utilizan para validar y procesar los datos enviados por los usuarios. Puedes utilizar formularios predefinidos de Django o crear tus propios formularios personalizados.

Estos son solo algunos de los archivos comunes que puedes encontrar en una aplicación de Django. Dependiendo de tus necesidades, también puedes crear las carpetas `static` y `templates` manualmente para almacenar archivos estáticos y plantillas HTML respectivamente, y crear el archivo `middleware.py` para definir middleware personalizado, entre otros.

Con esta analogía, un proyecto de Django es como una pizzería con diferentes aplicaciones, que serían como los diferentes tipos de pizza que vendes en tu pizzería.

¡Bien hecho! Ahora sabes cómo se estructura una aplicación en Django. ¡Hora de pedir una pizza como recompensa por tus logros!

3 Modelos en Django

3.1 El ORM

Ahora vamos a hablar sobre un tema muy importante en Django: ¡El ORM!

¿Qué es el ORM? Bueno, es como un políglota que hace traducciones entre nuestra base de datos y Django. Siendo más técnicos, ORM significa Mapeador de Objeto-Relacional (Object-Relational Mapping en inglés). Lo que esto significa es que se encarga de hacer la conexión entre las tablas de nuestra base de datos (la parte relacional) y los objetos en nuestro código (la parte de objetos).

¿Por qué es importante? Bueno, digamos que tienes una aplicación web que utiliza una gran cantidad de datos de usuario. Sin ORM, tendrías que escribir SQL manualmente para traer la información de la base de datos y convertirla en objetos que tu código pueda entender. Pero con el ORM, todo eso se hace por nosotros. Hace que nuestra vida sea mucho más fácil y nos permite concentrarnos en lo que realmente importa: construir una aplicación increíble.

Y como todo en Django, ¡El ORM es súper fácil de usar! Por ejemplo, para obtener una lista de todos los usuarios en nuestra base de datos, simplemente escribimos:



```
# app/models.py
from django.contrib.auth.models import User

users = User.objects.all()
print(users)
```

¿Listo? ¡Así de fácil! Y lo mejor de todo es que esto es solo la punta del iceberg; el ORM de Django ofrece muchas más funcionalidades que nos permiten hacer todo

3 MODELOS EN DJANGO

tipo de cosas geniales.

3.2 SQLite

Ahora quiero hablarte de SQLite, una base de datos súper versátil y fácil de usar. SQLite es muy popular en el mundo de la programación gracias a que no requiere un servidor para funcionar, lo que lo hace ideal para aplicaciones pequeñas o medianas.

Ahora bien, ¿puedes ver un nuevo archivo llamado db.sqlite3 en tu proyecto? Si no lo conoces, déjame decirte que ese archivo es donde SQLite guarda toda la información que se está manejando en la base de datos. Así que si estás trabajando en un proyecto de Django y ves un archivo con ese nombre, ya sabes de qué se trata.

Pero ¡eso no es todo! Si eres fan de Visual Studio Code (VSCode), te alegrará saber que hay una extensión súper útil llamada SQLite que permite visualizar información de tu base de datos de manera muy sencilla. ¿Qué tal si además de guardar información en SQLite, también la mostramos en un formato visualmente atractivo? Esta extensión te puede ayudar con eso.

Para instalar la extensión “SQLite” en Visual Studio Code, puedes seguir estos pasos:

1. Abre Visual Studio Code.
2. Ve a la pestaña “Extensiones” en la barra lateral izquierda (el ícono de cuatro cuadros superpuestos).
3. En el campo de búsqueda, escribe “SQLite” y presiona Enter.
4. Busca la extensión “SQLite” desarrollada por Alexey Sazonov y haz clic en “Instalar” .
5. Una vez instalada, puedes abrir archivos de bases de datos SQLite en Visual Studio Code y utilizar la interfaz proporcionada por la extensión para explorar y manipular los datos.

3 MODELOS EN DJANGO

Con la extensión “SQLite” instalada, puedes abrir archivos de bases de datos SQLite en Visual Studio Code y navegar por las tablas, ver registros, ejecutar consultas SQL y realizar otras operaciones comunes en una base de datos SQLite.

Con esto puedes ver el contenido de tu base de datos que en este momento debería estar vacía pero la modificaremos en las siguientes lecciones.

Recuerda que la extensión “SQLite” está diseñada específicamente para trabajar con bases de datos SQLite y no es compatible con otros tipos de bases de datos.

3.3 Configuración de base de datos

Estamos listos para configurar la base de datos SQLite en Django? ¡Genial!

Para empezar, imagina que nuestra base de datos es como un súper héroe. Y como todo súper héroe, necesita una identidad secreta, ¿verdad?

Para crear esta identidad secreta para nuestra base de datos, simplemente debemos abrir el archivo `settings.py` en nuestra aplicación de Django y buscar la sección que dice `DATABASES`. Ahí es donde podemos especificar qué base de datos queremos usar y cómo se debería nombrar nuestra ‘identidad secreta’ .

Por ejemplo, para usar la base de datos SQLite, vemos la siguiente configuración en `DATABASES`:

3 MODELOS EN DJANGO

```
from pathlib import Path
BASE_DIR = Path(__file__).resolve().parent.parent

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

De esta forma, le decimos a Django que queremos usar el motor de base de datos SQLite y que queremos que la base de datos se llame db.sqlite3 y que se encuentre en la raíz del proyecto o `BASE_DIR`.

El siguiente paso es migrar las tablas que vienen por defecto con Django. Para hacerlo, simplemente debemos correr las migraciones de Django en nuestra aplicación:

```
python3 manage.py migrate
```

Con eso, Django creará todas las tablas necesarias para nuestra aplicación. Puedes ver el contenido de tu base de datos que en este momento debería tener nuevas tablas. Si revisamos la tabla `auth_user` podemos ver que tiene varias columnas como `id` de tipo `integer`, `password` de tipo `varchar`, y `last_login` de tipo `datetime`, entre otras. Estas tablas no tienen datos por ahora pero crearemos modelos en las siguientes

3 MODELOS EN DJANGO

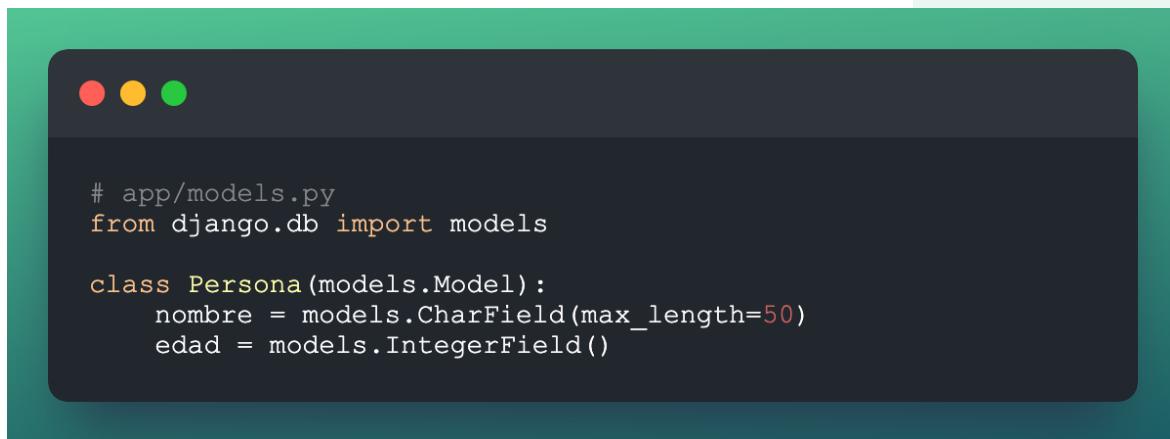
lecciones. ¿Fácil, no?

3.4 Creación de modelos

En esta ocasión, te hablaré sobre la creación de modelos en Django, que es uno de los pilares fundamentales para construir una aplicación web.

En primer lugar, ¿qué es un modelo en Django? Bueno, en términos generales, un modelo es una representación de una entidad o una tabla de una base de datos. Puede incluir atributos como nombres, fechas, direcciones, etc.

En Django, la creación de modelos es realmente fácil. Solo necesitas crear una clase y luego definir los atributos que deseas agregar a tu modelo. Algo así:



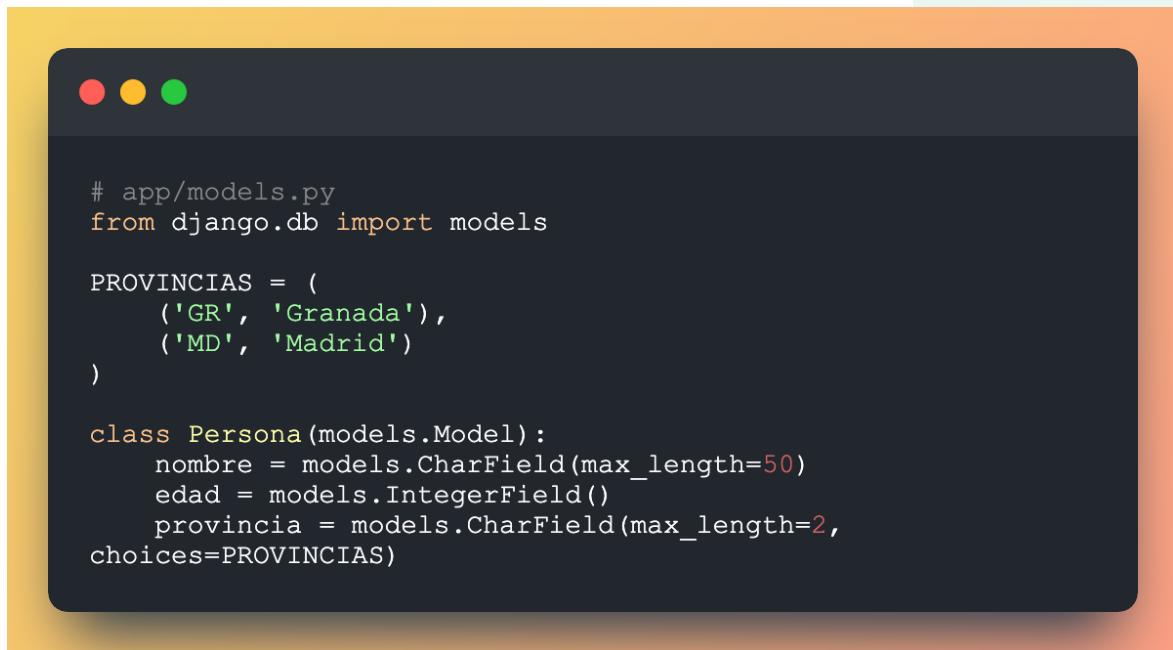
```
# app/models.py
from django.db import models

class Persona(models.Model):
    nombre = models.CharField(max_length=50)
    edad = models.IntegerField()
```

¡Listo! Acabas de crear un modelo llamado “Persona” que incluye dos atributos: nombre (cadena de caracteres) y edad (entero). Pero eso no es todo, puedes agregar mucha más funcionalidad.

Si necesitas una lista de opciones para un campo, como por ejemplo, una lista de provincias, Django tiene una opción que se llama “choices”. Verás lo fácil que es:

3 MODELOS EN DJANGO



```
# app/models.py
from django.db import models

PROVINCIAS = (
    ('GR', 'Granada'),
    ('MD', 'Madrid')
)

class Persona(models.Model):
    nombre = models.CharField(max_length=50)
    edad = models.IntegerField()
    provincia = models.CharField(max_length=2,
        choices=PROVINCIAS)
```

Aquí estamos creando un modelo “Persona” que cuenta con nombre, edad y provincia. La provincia es un campo de caracteres que solo acepta dos caracteres y que será validado si corresponde a una de las opciones incluidas en la tupla PROVINCIAS.

Ahora bien, no todo es perfecto. En la vida (y en la programación) hay retos. Por eso, aquí te propongo un reto simple: crea un modelo denominado “Libro” que incluya las siguientes características: título (cadena de caracteres), autor (cadena de caracteres) y precio (decimal). ¡Manos a la obra!

3.5 Campos de modelos

Ya vimos como crear modelos y los atributos que vimos se llaman campos.

Los campos más populares son:

- CharField: este campo es perfecto para almacenar texto corto como nombres

3 MODELOS EN DJANGO

o títulos de películas.

```
class Pelicula(models.Model):
    titulo = models.CharField(max_length=50)
```

- IntegerField: si necesitas almacenar números enteros, éste es tu campo.

```
class Producto(models.Model):
    cantidad = models.IntegerField()
```

- DateField: si necesitas almacenar fechas, este campo puede ser la solución.

```
class Publicacion(models.Model):
    fechapublicacion = models.DateField()
```

Reto:

Crea un modelo llamado “Contacto” con los siguientes campos: nombre, correo electrónico, mensaje y fecha de creación. Asegúrate de utilizar los campos más adecuados para cada tipo de información.

3.6 Migración

Ahora hablaremos sobre migraciones en Django, un tema muy importante para quienes trabajamos en el desarrollo web.

Las migraciones son como una mudanza, pero para nuestras bases de datos. Es decir, si hacemos cambios en nuestro modelo de datos, como añadir un campo o eliminar una tabla, necesitamos migrar esos cambios en nuestra base de datos para que todo siga funcionando correctamente.

Y no es algo que se deba tomar a la ligera, ya que una mala migración puede causar muchos dolores de cabeza y errores difíciles de solucionar. Así que, “mejor prevenir que lamentar”, y asegurarnos de hacer migraciones correctamente.

Para hacer una migración en Django, lo primero es crear un archivo de migración con el comando `makemigrations`. Luego, debemos aplicar esa migración a nuestra base de datos usando el comando `migrate`.

Pero ¿cómo saber si nuestras migraciones están bien hechas? Una forma es usar el comando `sqlmigrate` para ver el SQL que se generará para aplicar la migración, y comprobar si tiene sentido.

Y hablando de migraciones, ¿saben cuál es el animal que migra las distancias más largas? ¡El gavotín ártico que migra más de 80 000 km al año del ártico a la Antártida y de vuelta! Pero por suerte, en Django no necesitamos emprender largos viajes para migrar nuestras bases de datos, ¡lo podemos hacer desde la comodidad de nuestro ordenador!

Ahora, ¡vamos a practicar con este código! Creamos un modelo de datos en el archivo `models.py` en nuestra aplicación. Añadimos nuestra aplicación a la lista de `INSTALLED_APPS` en el archivo `settings.py` de nuestro proyecto para que los modelos sean reconocidos.

3 MODELOS EN DJANGO

```
# app/models.py
from django.db import models

PROVINCIAS = (
    ('GR', 'Granada'),
    ('MD', 'Madrid')
)

class Persona(models.Model):
    nombre = models.CharField(max_length=50)
    edad = models.IntegerField()
    provincia = models.CharField(max_length=2,
choices=PROVINCIAS)

# settings.py
INSTALLED_APPS = [
    'holamundo'
]
```

Hacemos la migración correspondiente con este comando:

```
python3 manage.py makemigrations
```

Esto crea el archivo migrations/0001_initial.py que tiene detalles de la nueva tabla que se creará en nuestra base de datos. Este archivo no se debe modificar manualmente.

Aplicamos la migración con este comando:

3 MODELOS EN DJANGO

```
python3 manage.py migrate
```

Y ahora al revisar nuestra base de datos podemos ver una nueva tabla <nombre de app>_persona con las propiedades que hemos creado. Aquí es importante notar que la propiedad id se crea automáticamente para cualquier modelo.

Si deseas revisar el comando de SQL que se realiza detrás de escena con esta migración puedes correr el siguiente comando:

```
# python3 manage.py sqlmigrate app 0001_initial  
python3 manage.py sqlmigrate holamundo 0001_initial
```

Y por último, un reto simple: crea un modelo de datos que incluya un campo de tipo booleano, haz la migración correspondiente y comprueba que todo funciona correctamente. ¡Adelante!

3.7 Relaciones entre modelos

¡Hablemos de relaciones en Django! En Django, podemos definir relaciones entre modelos utilizando diferentes tipos de “keys” o llaves. La más común es la ForeignKey, que permite definir una relación “muchos a uno” .

Veamos un ejemplo de cómo podríamos definir una relación ForeignKey entre dos

3 MODELOS EN DJANGO

modelos. Imagina que estamos creando una aplicación de blog, y tenemos un modelo para los blogs y otro para los autores. Queremos que cada blog tenga un autor específico, y que cada autor pueda tener varios blogs.

Empezemos creando una aplicación para nuestro blog:

```
python3 manage.py startapp blog
```

Ahora podemos crear los modelos de Autor y Blog:

```
# app/models.py
from django.db import models

class Autor(models.Model):
    nombre = models.CharField(max_length=150)
    email = models.EmailField()

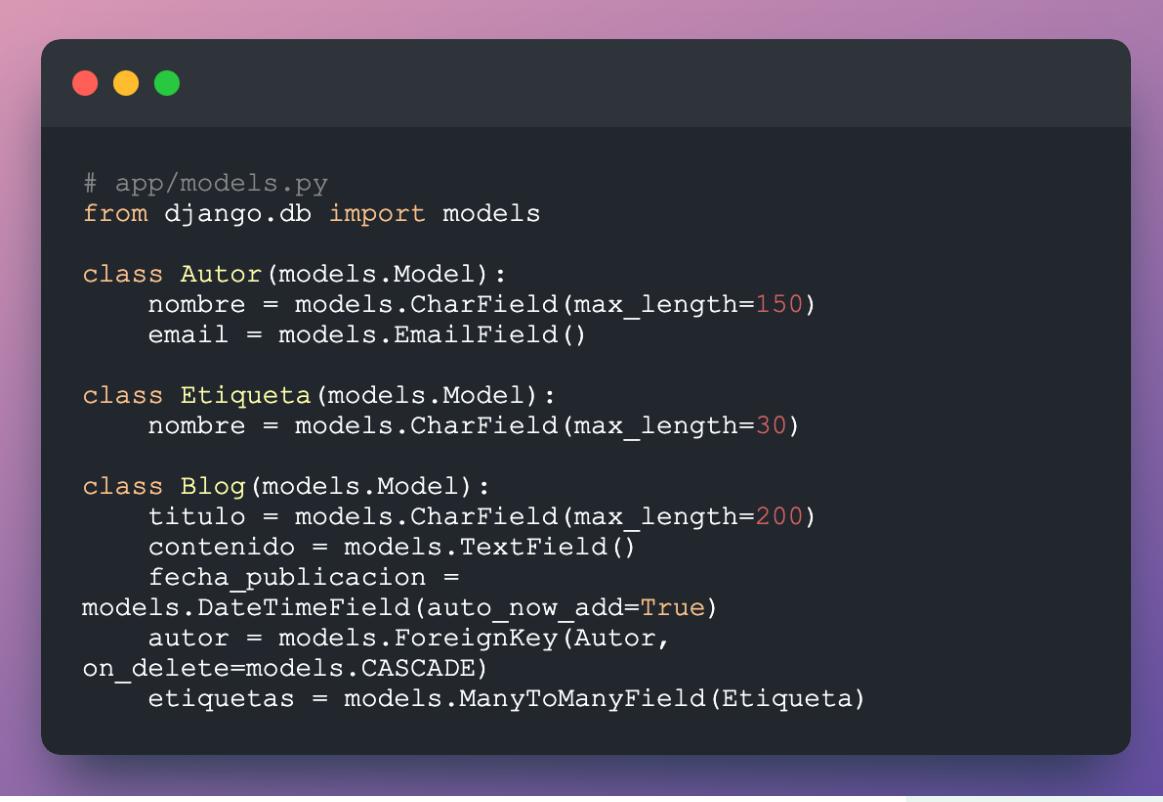
class Blog(models.Model):
    titulo = models.CharField(max_length=200)
    contenido = models.TextField()
    fecha_publicacion =
        models.DateTimeField(auto_now_add=True)
    autor = models.ForeignKey(Autor,
        on_delete=models.CASCADE)
```

En este ejemplo, el campo “autor” en el modelo de Blog es una ForeignKey que se relaciona con el modelo Autor. El parámetro “on_delete” se define con el propósito de eliminar los registros que dependen del objeto hijo cuando el padre se elimina.

3 MODELOS EN DJANGO

Esto ayuda a mantener la integridad referencial de la base de datos. ¡No queremos que nuestras relaciones se rompan y sean como un corazón roto!

Ahora imaginemos que queremos definir una relación “muchos a muchos”, porque nuestros blogs pueden tener varias etiquetas y, además, cada etiqueta puede estar en varios blogs. Podríamos entonces usar la relación ManyToManyField, que nos permitiría establecer una relación recíproca con varios objetos.



```
# app/models.py
from django.db import models

class Autor(models.Model):
    nombre = models.CharField(max_length=150)
    email = models.EmailField()

class Etiqueta(models.Model):
    nombre = models.CharField(max_length=30)

class Blog(models.Model):
    titulo = models.CharField(max_length=200)
    contenido = models.TextField()
    fecha_publicacion =
        models.DateTimeField(auto_now_add=True)
    autor = models.ForeignKey(Autor,
        on_delete=models.CASCADE)
    etiquetas = models.ManyToManyField(Etiqueta)
```

En este ejemplo, el campo “etiquetas” en el modelo de Blog es una ManyToManyField que se relaciona con el modelo Etiqueta. Eso significa que podemos tener un blog sin ninguna etiqueta, o un blog con varias etiquetas. Puedes pensar en ello como una fiesta con muchas personas, donde cada persona puede ir a varias fiestas, y en cada fiesta pueden haber muchas personas. ¡Suena como una buena fiesta, no!

3 MODELOS EN DJANGO

Finalmente podemos migrar nuestros modelos a nuestra base de datos. Añadimos nuestra aplicación a la lista de INSTALLED_APPS en el archivo settings.py de nuestro proyecto para que los modelos sean reconocidos.

```
# settings.py
INSTALLED_APPS = [
    'blog'
]
```

Hacemos la migración correspondiente y aplicamos la migración:

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Y ahora al revisar nuestra base de datos podemos ver nuevas tablas para cada aplicación. Nota que la relación ManyToManyField ha creado una tabla adicional que se encarga de mapear los Blogs a las Etiquetas.

Ahora, ¿estamos listos para el reto? Para ir más allá, puedes crear tus propios modelos y definir relaciones entre ellos. Por ejemplo, puedes crear un modelo de “producto” y un modelo de “pedido” y definir una relación ForeignKey entre ellos donde un pedido puede tener varios productos, pero sólo un producto puede pertenecer a un pedido.

¡Que te diviertas creando relaciones sanas y estables en Django!

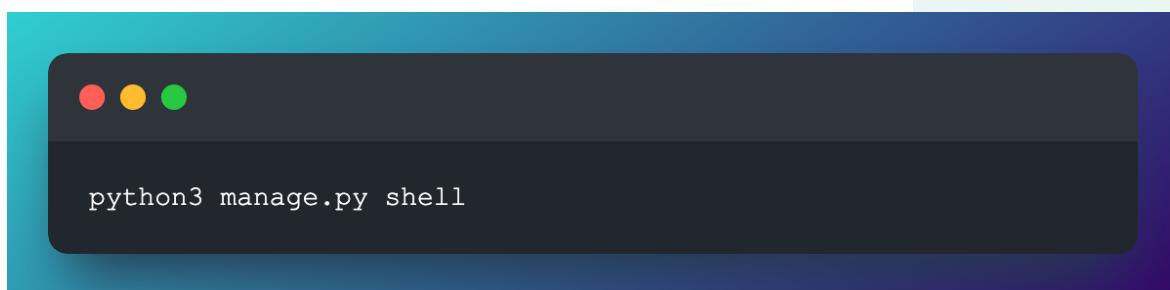
4 Shell en Django

4.1 Guardar modelo en shell

Ahora vamos a hablar sobre cómo guardar un modelo en Django desde la terminal de tu ordenador.

Ahora sí, manos a la obra. Primero, asegúrate de haber creado tu modelo en Django. Luego, abre la terminal de tu computadora y accede al directorio donde se encuentra tu proyecto de Django.

Una vez ahí, escribe el siguiente comando para acceder al shell de Django:



Después, importa tu modelo. ¡Muy bien! Ahora ya puedes crear una instancia de tu modelo y guardarla en la base de datos utilizando el siguiente código:

4 SHELL EN DJANGO

```
from blog.models import Autor

autor = Autor(nombre="Tu nombre", email="tuemail@gmail.com")
autor.save()

autor = Autor(nombre="Pedro", email="otroemail@gmail.com")
autor.save()

autor = Autor(nombre="Juan", email="juan@gmail.com")
autor.save()
```

Ahora puedes revisar los valores en tu base de datos en la tabla `blog_autor`. ¡Y listo, ¡ya has guardado tu modelo desde el shell de Django!

También puedes guardar registros relacionados:

```
from blog.models import Blog

blog = Blog(titulo="¿Qué es HTML?", contenido="Este blog
describe HTML", autor=autor[0])
blog.save()

# alternativa
autor.blog_set.create(titulo="¿Qué es CSS?", contenido="Este
blog describe CSS")
```

¡Reto! Como reto, te animo a que crees un modelo sencillo con algunos atributos y lo guardes desde la terminal de tu ordenador. ¡Atrévete a probar nuevas cosas!

4.2 Leer modelo en shell

Hoy te voy a enseñar sobre cómo leer un modelo en Django usando la shell.

Ahora sí, al grano. Vamos a la shell de Django. Una vez allí, podemos importar el modelo que queremos. Una vez importado el modelo, podemos leer los objetos que se encuentran en la base de datos usando:

```
from blog.models import Autor  
Autor.objects.all()
```

Notarás que esto nos permite obtener una lista QuerySet con cada elemento en esta tabla. Estos elementos no son fáciles de distinguir a simple vista pero podríamos obtener el valor del atributo nombre del primer elemento de esta manera:

```
Autor.objects.all()[0].nombre
```

También podemos obtener un elemento usando un valor de esta manera:

4 SHELL EN DJANGO

```
autor = Autor.objects.get(pk=1)
autor.nombre # leer atributo

# leer modelo relacionado
autor = Autor.objects.get(nombre="Juan")
autor.blog_set.all()
autor.blog_set.count()
```

Una mejor manera de distinguir estos elementos visualmente en el shell es añadiendo la función `__str__` al modelo de esta manera:

```
# app/models.py
class Autor(models.Model):
    nombre = models.CharField(max_length=150)
    email = models.EmailField()

    def __str__(self):
        return f"{self.id} {self.nombre} {self.email}"
```

Esta función retorna un f-string con el id, nombre, y el email del modelo Autor pero podríamos formatearlo a nuestra preferencia. Ahora intenta leer todos los datos en la tabla Autor a través del shell para ver la diferencia.

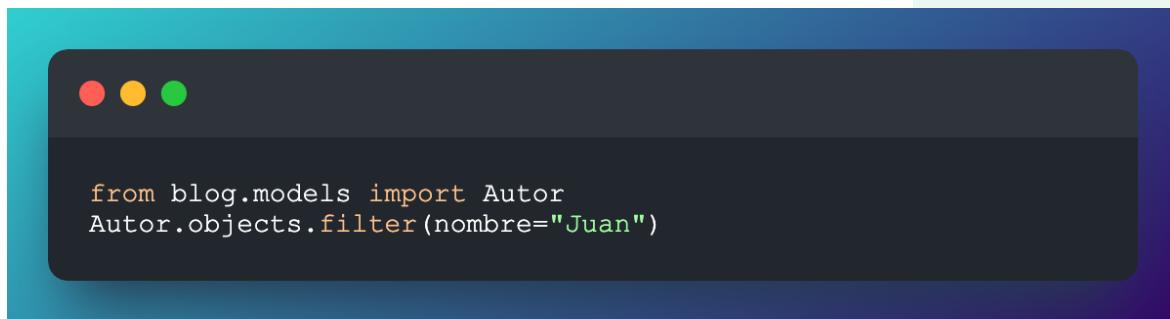
Y listo, de esta forma podemos leer nuestro modelo usando la shell. Pero, como reto, ¿puedes intentar agregar un filtro a la consulta `MiModelo.objects.all()` para que solo devuelva los objetos cuyo campo nombre sea igual a “Juan”? ¡Éxitos programando!

4.3 Filtrar modelo en shell

¿Estamos listos para aprender a filtrar modelos en Django? ¡Genial! ¡No te preocupes, aquí está Django al rescate!

Para filtrar modelos en Django, podemos usar el shell de Django. El shell de Django es como una caja de herramientas con la que podemos interactuar directamente con nuestra base de datos.

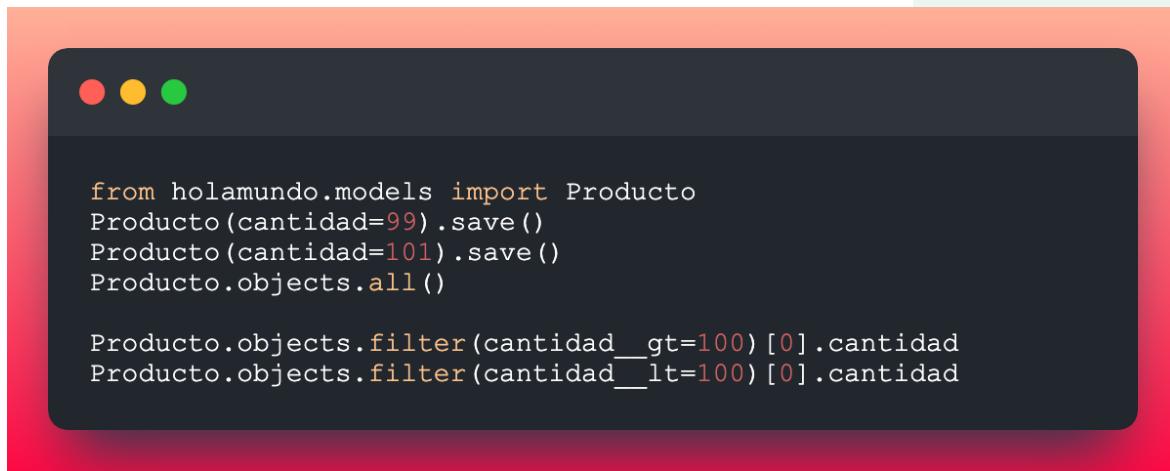
Una vez dentro del shell, podemos importar cualquier modelo del proyecto. Por ejemplo, si queremos filtrar el modelo Autor, podemos hacer lo siguiente:



```
from blog.models import Autor
Autor.objects.filter(nombre="Juan")
```

¿Fácil, no?

El ORM de Django también nos permite usar el método filter con opciones más específicas. Por ejemplo, para seleccionar solo aquellos autores que tengan un precio mayor a 100 podemos usar este código:



```
from holamundo.models import Producto
Producto(cantidad=99).save()
Producto(cantidad=101).save()
Producto.objects.all()

Producto.objects.filter(cantidad__gt=100)[0].cantidad
Producto.objects.filter(cantidad__lt=100)[0].cantidad
```

Pero espera, ¿qué significa cantidad__gt=100? ¿Es eso un error de escritura? No, no lo es. En Django, utilizamos doble guión bajo (__) para hacer referencia a las cláusulas de filtrado, en este caso, gt (greater than) que significa “mayor que” y lt (less than) significa “menor que”. Así que lo que estamos diciendo es que queremos todos los productos cuyo precio es mayor a 100.

¡Buen trabajo! Ahora tú también sabes filtrar modelos en Django. ¡Ve y hazlo! Pero antes, te reto a que pruebes filtrar otro modelo de tu proyecto. ¿Qué tal si intentas filtrar el modelo Blog y seleccionar solo aquellos blogs publicados en los últimos 7 días?

4.4 Borrar modelo en shell

Borrar instancias de modelos es algo que todos hemos tenido que hacer alguna vez. Así que, sin más preámbulos, te enseñaré cómo hacerlo. Primero, entra a la shell de Django. Luego, deberás importar el modelo que quieras borrar. Si por ejemplo, tienes un modelo llamado Blog, deberías importarlo.

¡Mantén la calma! Lo más difícil ya pasó. Ahora, sólo tienes que utilizar el método delete() para borrar el modelo que importaste. Sí, es así de simple, sólo tienes que llamar al método delete(). Así:

4 SHELL EN DJANGO

```
from autor.models import Autor

autor = Autor.objects.filter(nombre="Pedro") [0]
autor.blog_set.create(titulo="¿Qué es JavaScript?", 
contenido="Este blog describe JavaScript")
autor.blog_set.all()

autor.delete()
```

¡Listo! Ya habrás borrado tu modelo desde la shell. Nota que si tienes otros modelos relacionados con la propiedad models.CASCADE, estas instancias también se van a borrar. ¿Fácil verdad?

Este método borra todos los objetos relacionados con el modelo, así que ten cuidado. (No sea que borres a alguien importante y luego venga a acusarte)

```
Autor.objects.delete()
```

Ahora, para el reto. Te animo a que pruebes estos comandos en la shell y borres un modelo que no necesites. ¡Pero ojo! Antes de hacerlo asegúrate de que no lo necesitas, no queremos un desastre en nuestras manos. ¡Que la fuerza te acompañe!

5 Admin de Django

5.1 Acceder al panel

¿Estamos listos para convertirnos en administradores implacables? ¡Genial!

Admin de Django es una de las características más poderosas y útiles de Django. Esta es la herramienta que te permitirá crear páginas web para administrar tus bases de datos de una manera fácil y eficiente. Y creeme que es super importante manejar una buena BD, porque “más vale una buena base de datos que 1000 respaldos!”. Con Admin de Django, puedes agregar, editar y eliminar registros de tu BD con solo unos pocos clics. Admin te hace la vida más fácil, y eso es precisamente lo que queremos en esta vida, ¿verdad?

Las páginas que crees con Admin de Django tienen una apariencia profesional y elegante que seguramente impresionará a cualquier cliente o usuario final. Incluso puedes personalizar la apariencia para que se ajuste a tu marca, porque ya sabes que, “la imagen lo es todo” , ¡menos mal que no estás haciendo esto a mano!

Para empezar puedes correr el servidor de Django y navegar a la ruta del admin que está registrada en el archivo urls.py de nuestro proyecto en la url "admin/".

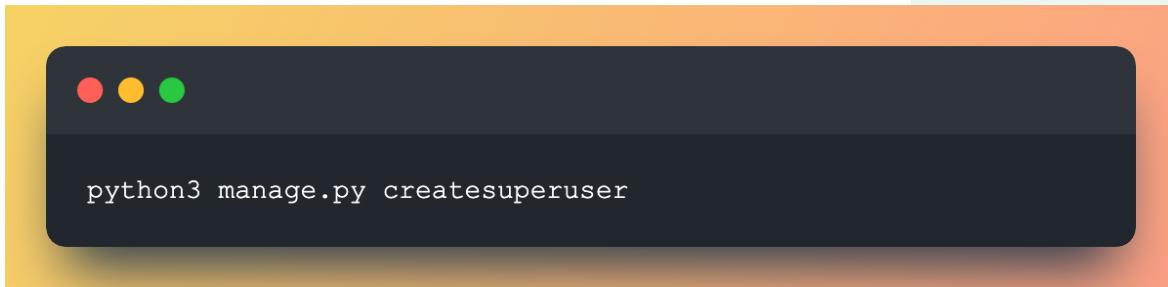
Puedes ver que necesitamos un nombre de usuario y contraseña para acceder y esto lo veremos en la siguiente lección.

5.2 Crear superusuario

¡Hola! Si estás leyendo esto, es porque estás pensando en convertirte en el superhéroe de tu aplicación Django: ¡un superusuario! No te preocupes, ¡estoy aquí para ayudarte y guiarte en este emocionante viaje!

Primero, abre tu consola y teclea el siguiente comando:

5 ADMIN DE DJANGO



Este comando nos va a pedir un nombre, un email, y una contraseña. Puedes usar admin, djangotutorial@ejemplo.com, y abcd1234! respectivamente para tu aprendizaje pero en un proyecto profesional recuerda usar un nombre y una contraseña únicos. Tampoco olvides guardar tus credenciales en un lugar muy seguro que no sea en el repositorio de tu proyecto en donde puedan verse comprometidas.

¡Con esto has creado un superusuario en Django! Ahora, puedes ingresar tu nombre de usuario y contraseña e ingresar al panel de administración.

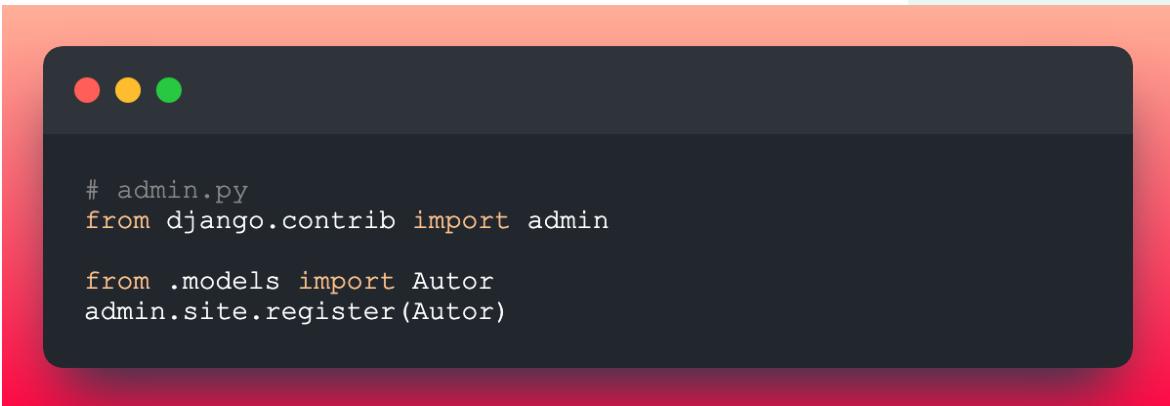
Dentro del panel puedes ver usuarios y grupos. Dentro de usuarios puedes ver el nuevo usuario que hemos creado. Pero todavía no podemos ver nuestros modelos, esto es porque debemos registrarlos y lo veremos en la siguiente lección.

¡Y ahora, para el reto! Si eres digno del título de superhéroe, crea un nuevo superusuario con el nombre de usuario “Flash”, el correo electrónico “flash@justiceleague.com” y una contraseña segura de tu elección. ¡No lo dudes, adelante y crea tu superusuario!

5.3 Registrar modelo

Si deseamos editar nuestros modelos en el admin de Django debemos registrarlos con este código sencillo. Modificamos el archivo admin.py en nuestra aplicación:

5 ADMIN DE DJANGO



```
# admin.py
from django.contrib import admin

from .models import Autor
admin.site.register(Autor)
```

¡Listo! Con solo tres líneas de código, hemos registrado nuestro modelo “Autor” con Admin de Django.

Podemos ver el nombre de nuestro registro de acuerdo a la función `__str__` que definimos en nuestro modelo.

Ahora podemos agregar, editar y eliminar registros de nuestra base de datos a través de la página web administrativa. ¡Genial, ¿no?!

Para agregar podemos dar un clic en el botón Add y llenar los datos del autor. Como nombre podemos usar Marcos y como email podemos usar marcos@ejemplo.com. Presionamos Save para guardar.

Para editar el registro puedes dar un clic en el mismo, modificarlo con el nombre de Marcos II, y guardarlo.

Para eliminar el registro puedes dar un clic en el mismo presionar DELETE.

Para el reto, te desafío a que personalices la apariencia del Admin de Django con el fin de mejorar la experiencia del usuario. ¡No te detengas aquí, sigue aprendiendo!

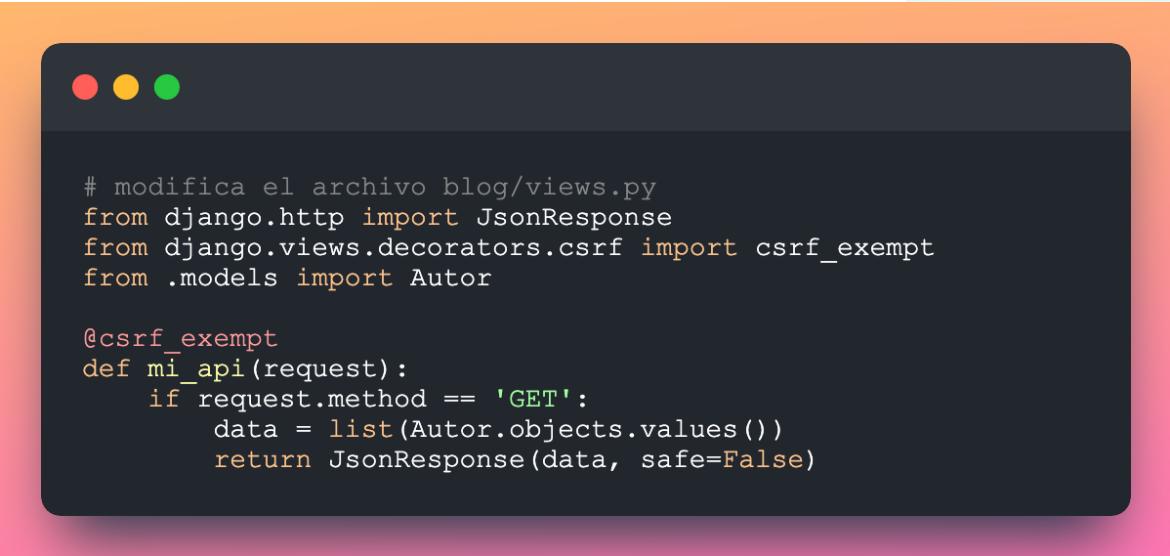
6 APIs en Django

6.1 Creación de APIs

¿Sabías que crear APIs en Django es más fácil de lo que parece? ¡Sí, lo sé!

Django tiene una arquitectura diseñada para facilitar el proceso de creación de APIs, desde la serialización hasta la autenticación y autorización de usuarios. Pero no vamos a profundizar en eso hoy, ¡vamos directo a la acción!

Para empezar, es recomendable que tengas un proyecto de Django configurado y corriendo en tu máquina. Aquí podemos continuar con nuestra app de blogs. ¿Ya tienes todo eso? ¡Genial! Ahora, vamos a crear una API REST. Esta es similar a la API que creamos en la lección de “Hola Mundo” pero en lugar de servir un archivo de HTML vamos a servir un objeto JSON que puede ser consumido por cualquier aplicación.



```
# modifica el archivo blog/views.py
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from .models import Autor

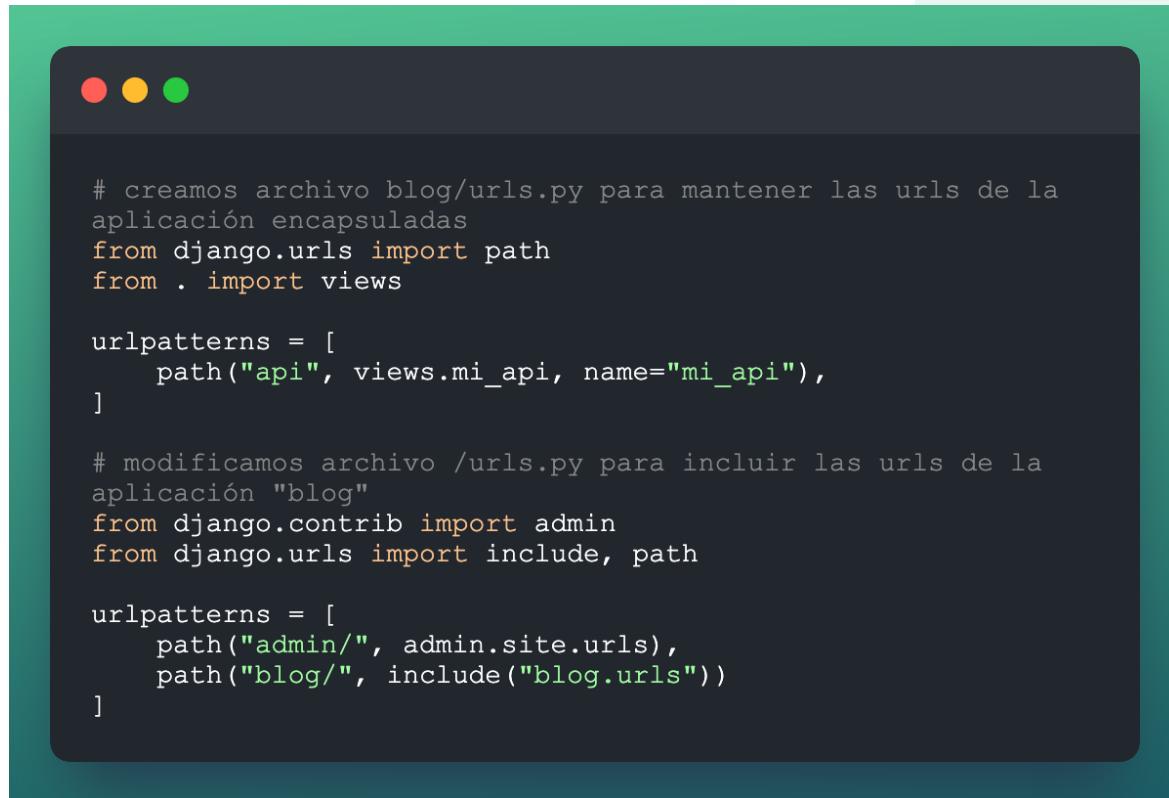
@csrf_exempt
def mi_api(request):
    if request.method == 'GET':
        data = list(Autor.objects.values())
        return JsonResponse(data, safe=False)
```

Este código define una vista `mi_api` que toma todos los objetos de `Autor` y los convierte en JSON. La etiqueta `@csrf_exempt` es necesaria si planeas permitir solicitudes de métodos no seguros (como POST, PUT, DELETE, etc.) desde tu API. Esto solo lo

6 APIS EN DJANGO

habilitamos para evitar ver un error de CSRF si lo pruebas con métodos no seguros para este ejemplo.

Ahora registramos la función en una URL en nuestro proyecto.



```
# creamos archivo blog/urls.py para mantener las urls de la
# aplicación encapsuladas
from django.urls import path
from . import views

urlpatterns = [
    path("api/", views.mi_api, name="mi_api"),
]

# modificamos archivo /urls.py para incluir las urls de la
# aplicación "blog"
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("admin/", admin.site.urls),
    path("blog/", include("blog.urls"))
]
```

Ahora puedes correr el servidor de Django y visitar la URL /blog/api para ver que recibes un objeto json con todos los datos de la tabla de Autores.

Eso es todo lo que necesitas para crear tu propia API de ejemplo, ¿qué más puedes pedir? Pero no te rindas todavía, ¡tenemos un reto! Intenta mejorar este código agregando alguna funcionalidad adicional.

¡Recuerda, siempre puedes contar con Django para impulsar tus proyectos web!

6.2 Serialización de datos

Ahora vamos a hablar sobre serialización de datos en Django. ¿Deseas dominar la serialización? ¡Estoy seguro que si!

Primero, ¿qué es la serialización de datos? Básicamente es el proceso de convertir un objeto de Python en un formato que pueda ser almacenado o enviado a través de una red. Piénsalo como empaquetar un regalo para enviarle a alguien. Tienes que ponerlo en una caja y envolverlo para que llegue a su destino seguro y en buenas condiciones. Lo mismo sucede con los objetos de Python.

Django tiene una herramienta llamada serializers que nos permite crear una representación en formato JSON o XML de los objetos de nuestra base de datos. ¿Quieres ver cómo funciona? ¡Veamos!

```
# Creamos una nueva api para obtener un solo autor

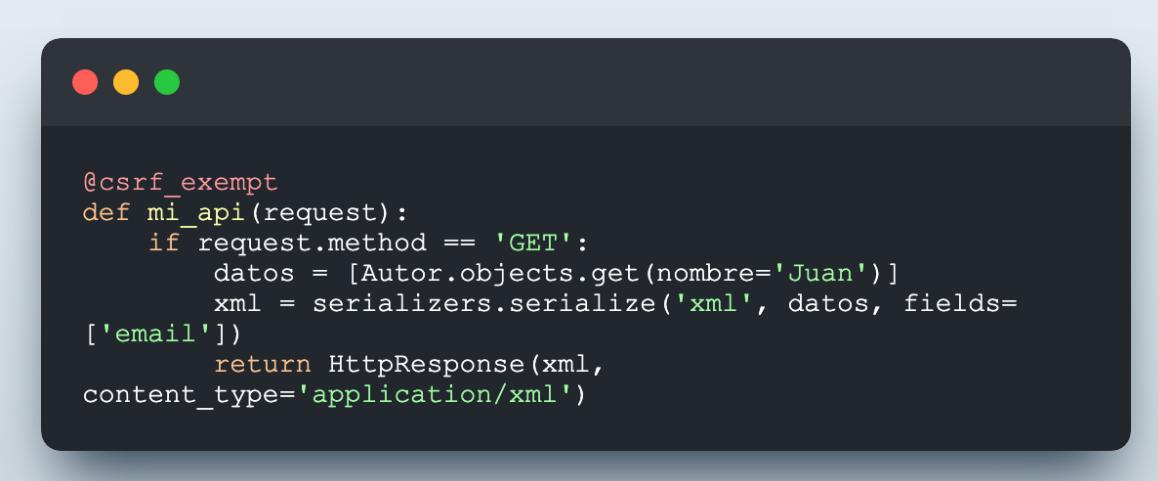
from django.core import serializers
from .models import Autor

@csrf_exempt
def mi_api(request):
    if request.method == 'GET':
        datos = [Autor.objects.get(nombre='Juan')]
        json = serializers.serialize('json', datos)
        return JsonResponse(json,
                           content_type='application/json')
```

¡Listo! Acabamos de serializar nuestra instancia de la clase Autor en formato JSON. Fácil, ¿verdad? Puedes ver que el método serialize requiere los datos en forma de lista. Ok, veo que ahora envío tan solo un registro en esta lista y me pregunto ¿no puedo hacer esto ya con JsonResponse?. JsonResponse se enfoca en convertir

6 APIS EN DJANGO

una lista o diccionario y responder automáticamente en formato JSON, mientras que los serializers no están limitados al formato JSON. Puedes serializar a los formatos JSON, XML, y YAML. Adicionalmente, los serializers se usan específicamente para modelos de Django y permiten usar filtros. Veamos otro ejemplo:



```
@csrf_exempt
def mi_api(request):
    if request.method == 'GET':
        datos = [Autor.objects.get(nombre='Juan')]
        xml = serializers.serialize('xml', datos, fields=['email'])
        return HttpResponse(xml,
content_type='application/xml')
```

¡Increíble! Ahora solo estamos serializando el nombre del autor y retornando tan solo el campo email. Ahora puedes revisar la respuesta del servidor en el inspector de tu navegador.

Para desafiar tus habilidades de serialización, aquí te dejo un reto: serializa una instancia de tu modelo favorito utilizando el formato XML en lugar de JSON. ¡Sal de tu zona de confort y hazlo! ¡Ánimo!

6.3 Parámetros en url (str, int, slug, uuid, path)

¿Te imaginas ser capaz de recibir diferentes tipos de valores en tus URLs sin tener que hacer malabares para procesarlos después? ¡Pues no busques más! Django nos ofrece una gran solución: ¡los parámetros en la URL!

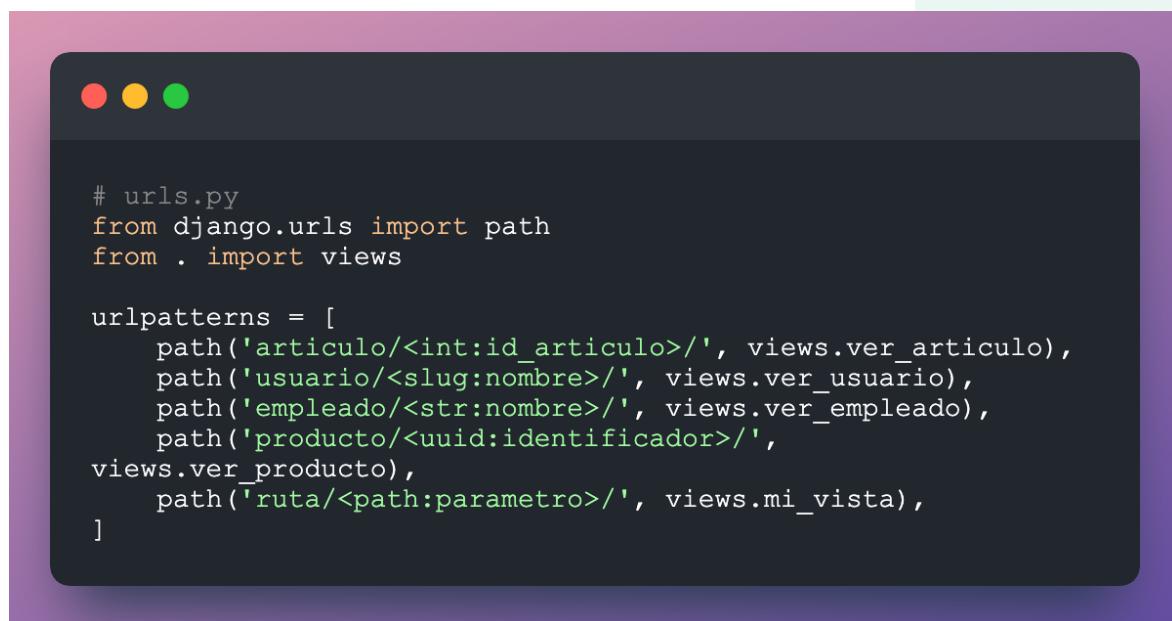
Con Django podemos establecer diferentes tipos de parámetros en la URL, como strings (str), enteros (int), slug, uuid y path. Así, podremos utilizar nuestros URLs pa-

6 APIS EN DJANGO

ra enviarle a nuestra app toda la información que necesitamos para construir nuestras páginas de una forma organizada y estructurada. Además, esto también nos permite construir URLs efectivas y fácilmente reconocibles para los usuarios.

Ahora bien, ¡presta atención! Si estás acostumbrado a definir URLs como `www.ejemplo.com/usuario/01`, ya no tendrás que romperte la cabeza para procesar ese número 01 que se te presenta en tu URL. Simplemente tendrás que declarar el tipo de parámetro en tu `urls.py` y Django se encarga del resto.

Por ejemplo, si queremos que uno de nuestros parámetros sea un número entero, podemos utilizar `int`. Si queremos que contenga solamente caracteres alfanuméricos y guiones, podemos utilizar `slug`. Si queremos asegurarnos de que sea un identificador único, podemos utilizar `uuid`. Y si queremos aceptar cualquier tipo de valor como parámetro en nuestra URL, utilizamos `path` ¡todo depende de lo que necesitemos!



```
# urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('articulo/<int:id_articulo>', views.ver_articulo),
    path('usuario/<slug:nombre>', views.ver_usuario),
    path('empleado/<str:nombre>', views.ver_empleado),
    path('producto/<uuid:identificador>',
        views.ver_producto),
    path('ruta/<path:parametro>', views.mi_vista),
]
```

Ahora tenemos que crear las funciones para recibir estos parámetros.

6 APIS EN DJANGO

```
# views.py

# visita articulo/2
def ver_articulo(request, id_articulo):
    return HttpResponseRedirect(f"Estás viendo el artículo {id_articulo}")

# visita usuario/marcos
def ver_usuario(request, nombre):
    return HttpResponseRedirect(f"Estás viendo el usuario {nombre}")

# visita empleado/marcos perez
def ver_empleado(request, nombre):
    return HttpResponseRedirect(f"Estás viendo el empleado {nombre}")

# visita producto/550e8400-e29b-41d4-a716-446655440000
def ver_producto(request, identificador):
    return HttpResponseRedirect(f"Estás viendo el producto {identificador}")

# visita ruta/cualquier_ruta
def mi_vista(request, parametro):
    return HttpResponseRedirect(f"Estás viendo la ruta con el parámetro: {parametro}")
```

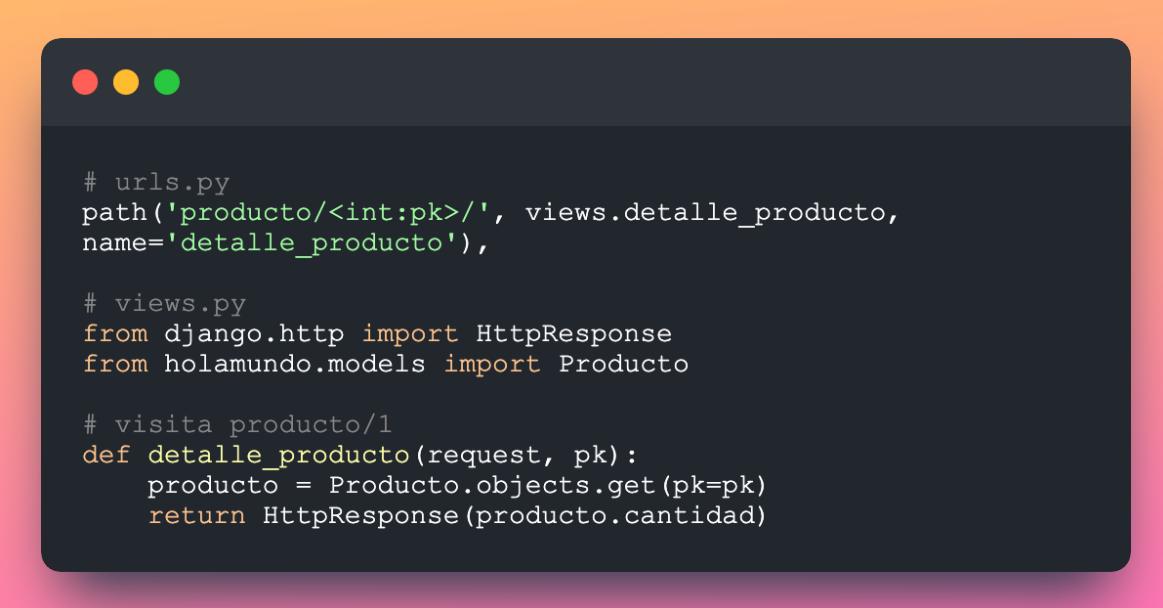
Ahora ¡Fácil, no? Simplemente utilizando los parámetros en nuestra URL, podemos hacer que nuestras vistas sean más eficientes, nuestros URLs más sencillos y nuestras aplicaciones más elegantes.

¡Reto! Agrega una nueva vista en tu app que reciba un parámetro slug en la URL y muestra un mensaje de bienvenida personalizado para el usuario. ¡Ponte a prueba y deja a tus usuarios impresionados con URLs distintivas y efectivas!

6.4 Parámetros en url para consultar modelos

Aprendamos sobre los parámetros en las URL. Con esta funcionalidad, podemos crear páginas dinámicas y personalizadas según los datos enviados directamente en la URL con una sintaxis sencilla y fácil de entender.

Por ejemplo, si queremos mostrar información de un modelo específico en Django, podemos usar parámetros para hacer una consulta personalizada de la siguiente manera:



```
# urls.py
path('producto/<int:pk>', views.detalle_producto,
      name='detalle_producto'),

# views.py
from django.http import HttpResponse
from holamundo.models import Producto

# visita producto/1
def detalle_producto(request, pk):
    producto = Producto.objects.get(pk=pk)
    return HttpResponse(producto.cantidad)
```

En el ejemplo anterior, usamos `<int:pk>` en nuestra URL que permite recibir un número entero y agregarlo como parámetro de nuestra vista `detalle_producto()`. Luego, hacemos una consulta al modelo `Producto` usando el número entero recibido para recuperar la información del producto deseado, imprimiendo un atributo del producto.

Ahora, ¡vamos al reto! En el código anterior, debes agregar una redirección personalizada a una página de error si un producto no se encuentra en tu base de datos. ¿Te atreves a intentarlo?

6 APIS EN DJANGO

¡Ánimo! No tengas miedo de intentar nuevas cosas en Django. Cada vez que lo haces, ¡aprendes algo nuevo!

6.5 Parámetros en peticiones

En el mundo de la programación, es importante tener un buen manejo de los parámetros en peticiones, y en Django no es la excepción. Un parámetro es un valor o variable que se pasa a una función para que ésta lo utilice en su ejecución.

En Django, los parámetros en peticiones suelen ser utilizados para enviar información adicional al servidor, como por ejemplo, en una petición GET, donde el usuario puede enviar información a través de la URL.

Pero, ¡ojo! Antes de utilizar los parámetros en peticiones, debemos asegurarnos de validarlos para evitar cualquier vulnerabilidad de seguridad.

En Django, es muy sencillo trabajar con los parámetros en peticiones. Para acceder a ellos, podemos hacer uso del objeto “request” y utilizar sus distintos métodos como “GET” o “POST” .

Por ejemplo, si deseamos acceder a un parámetro en una petición GET llamado “nombre” , podemos hacer lo siguiente:

```
def mi_funcion(request):
    nombre = request.GET.get('nombre')
    return nombre
```

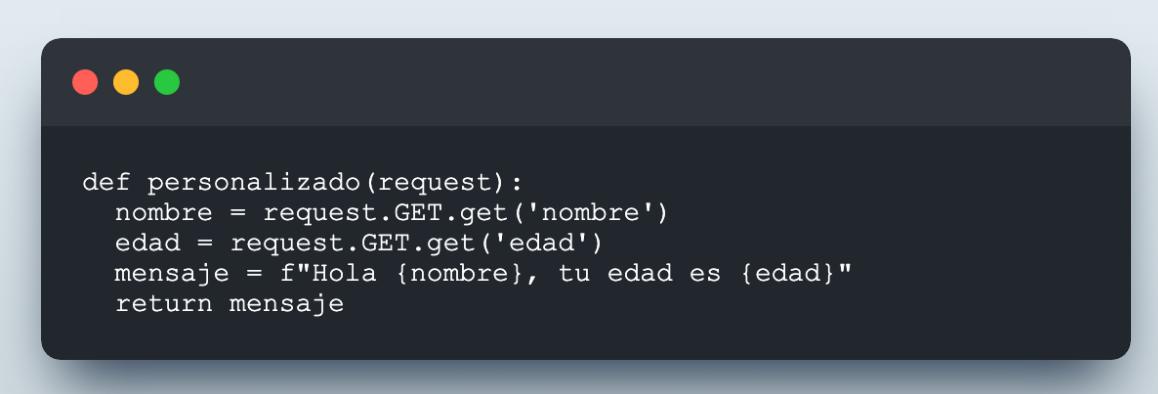
Así de fácil es obtener un parámetro en peticiones en Django.

¿Qué tal si ponemos en práctica lo aprendido?

6 APIS EN DJANGO

Reto:

Crea una vista en Django que reciba dos parámetros por GET, “nombre” y “edad”, y devuelva un mensaje personalizado que incluya ambos parámetros. ¡Anímate, puedes hacerlo!



```
def personalizado(request):
    nombre = request.GET.get('nombre')
    edad = request.GET.get('edad')
    mensaje = f"Hola {nombre}, tu edad es {edad}"
    return mensaje
```

¡Manos a la obra, programadorxs!

6.6 Funciones de atajo

6.6.1 redirect()

Las funciones de atajo son funciones comunes que se usan en el desarrollo de apps en Django. La función render() es la más común pero la dejaremos para su propia lección más adelante. Por ahora veamos redirect().

Imagínate que eres una persona súper animada que va a una fiesta, pero al llegar te das cuenta que te has equivocado de dirección. Tu instinto es ir rápidamente al lugar correcto, ¿verdad? Bueno, pues redirect() es exactamente eso, te permite redirigir a tus usuarios a la página correcta en tu sitio web.

¿Pero cómo funciona? ¡Es súper fácil! Solo necesitas indicar a qué dirección quieres que vaya la persona, y redirect() lo hace por ti. Además, también puedes personalizar un mensaje de error en caso de que algo salga mal en el proceso.

```
from django.shortcuts import redirect
def detalle_producto(request, pk):
    if pk == 1:
        return redirect('/admin/')
    producto = Producto.objects.get(pk=pk)
    return HttpResponse(producto.cantidad)
```

En una situación más práctica, si un usuario intenta acceder a una página que requiere de autenticación y no ha iniciado sesión, entonces puedes usar este atajo para redirigirlos al formulario de inicio de sesión.

¡Ahora es tu turno! ¿Te atreves a intentar este sencillo reto? Crea una función para una URL que redirija al usuario a la página de inicio.

6.6.2 get_object_or_404()

Imagina que tienes una vista que busca un objeto en la base de datos y lo muestra en pantalla. ¿Qué pasa si ese objeto no existe? Django te entregará un error 500 y tu usuario no entenderá qué ha pasado. Ahí es donde entra `get_object_or_404()`.

Con esta función podrás buscar el objeto solicitado, y si no lo encuentra, recibirás un error 404. Así le das una experiencia más amigable a tu usuario y evitas que quieran quemar tu sitio web.

Para utilizarla, simplemente importa la función de la librería:

6 APIS EN DJANGO

```
from django.shortcuts import get_object_or_404
```

Y en tu vista, reemplaza el típico método filter() por get_object_or_404():

```
def detalle_producto(request, pk):
    if pk == 1:
        return redirect('/admin/')
    producto =
    get_object_or_404(Producto.objects.filter(pk=pk))
    return HttpResponseRedirect(producto.cantidad)
```

Con esto, la función buscará el objeto Productos con el id solicitado. Si no lo encuentra, lanzará un error 404 que podrás personalizar a tu antojo.

Ahora que dominas get_object_or_404(), ve y utilízala en todas tus vistas. Tu usuario te lo agradecerá. ¡Hasta la próxima!

Reto:

¿Te gustaría seguir practicando con Django? Crea una vista que busque un objeto por otro campo que no sea el id. Usa get_object_or_404() para manejar los errores. ¡Éxito!

6.6.3 get_list_or_404()

Hoy vamos a hablar sobre una función muy útil en Django, se trata de `get_list_or_404()`. Esta función es como Batman, siempre aparece cuando más lo necesitamos y nos salva de los errores.

¿Alguna vez te ha pasado que intentas acceder a un objeto en Django pero no existe en la base de datos? Es muy probable que hayas encontrado el famoso error 404, que significa “No encontrado”. Pero no te preocupes, aquí es donde entra en acción `get_list_or_404()`. Esta función nos permite obtener una lista de objetos o devolver un error 404 si no se encuentran.

¿Quieres saber cómo trabajar con ella? No te preocupes, es muy fácil. Lo primero que tienes que hacer es importarla:

```
from django.shortcuts import get_list_or_404
```

Después, podrás utilizarla en tus vistas. Aquí te muestro un ejemplo práctico:

```
from django.shortcuts import get_list_or_404
from .models import Producto

def productos(request):
    productos = get_list_or_404(Producto)
    return JsonResponse(productos)
```

7 VISTAS EN DJANGO

En este ejemplo, utilizamos `get_list_or_404()` para obtener todos los objetos de Producto. Si no existen objetos en la base de datos, se mostrará un error 404 al usuario.

Puedes probarlo borrando todos los productos usando la shell de Django.



```
from holamundo.models import Producto
Producto.objects.all().delete()
```

¡Y eso es todo por hoy! Ahora ya sabes cómo utilizar `get_list_or_404()` para evitar los errores 404 en Django. ¿Te atreves a utilizarlo en tu próximo proyecto?

Reto:

Crea una vista en Django que utilice `get_list_or_404()` para mostrar todos los usuarios registrados en tu aplicación. ¡Anímate a probarlo!

7 Vistas en Django

7.1 Creación de vistas

Crear vistas en Django es realmente sencillo y ya lo hemos visto un par de veces. Aquí crearemos una nueva vista que vamos a utilizar para las siguientes lecciones. Solo necesitas escribir una función en Python que tome como argumento una solicitud (`request`) y devuelva una respuesta (`response`). Esta respuesta puede ser la página web que deseas mostrar o cualquier otro contenido que desees.

Por ejemplo, vamos a crear una vista de bienvenida para tu sitio web. En tu archivo

7 VISTAS EN DJANGO

views.py, puedes definir la función bienvenida que tome como argumento la solicitud y devuelva una respuesta que diga “¡Hola, bienvenid@ a mi sitio web!” .

```
# app/views.py
from django.http import HttpResponse

def bienvenida(request):
    return HttpResponse('¡Hola, bienvenid@ a mi sitio web!')
```

¡Y eso es todo! Ahora solo necesitas configurar tus URL para que apunten a esta vista. En tu archivo urls.py, puedes importar la vista bienvenida que creaste en el paso anterior y asignarle una URL.

```
# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('bienvenida/', views.bienvenida)
]
```

Aquí hemos asignado la URL /bienvenida/ para que muestre la vista bienvenida. Ya sabes que puedes elegir cualquier URL que deseas y asignarla a la vista correspondiente.

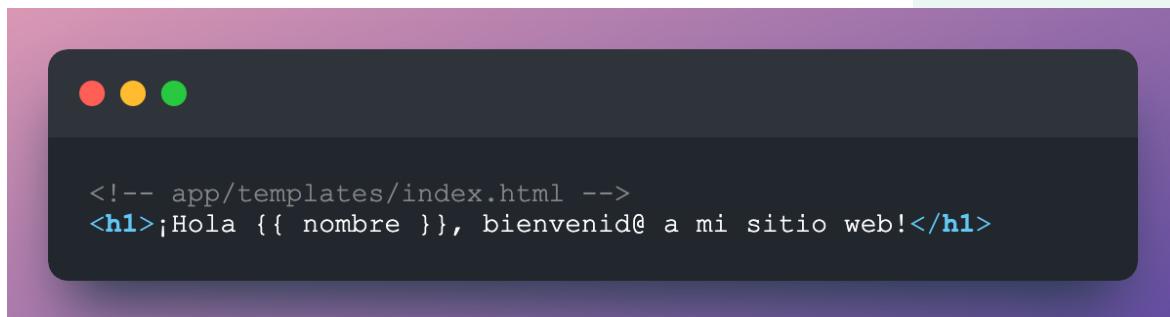
¡Y eso es todo! Ahora, cuando un usuario acceda a la URL /bienvenida/ en tu sitio web, verá la página de bienvenida que creaste.

7.2 Renderizado de plantillas

Ahora vamos a hablar sobre el renderizado de plantillas en Django.

Imagina que estás construyendo un sitio web y quieres saludar a tus usuarios de una manera personalizada. En Django, puedes crear una plantilla HTML donde podrás mostrar contenido dinámico. Imagina que tienes una plantilla llamada index.html y quieres agregar un saludo personalizado. Usaremos el formato {{ variable }} para insertar contenido dinámico en la plantilla.

Aquí tienes un ejemplo de cómo podrías hacerlo:



En este ejemplo, hemos creado un saludo que se mostrará en la página. La variable nombre se define en la función de vista y se pasa como contexto al renderizar la plantilla.

Ahora, en el archivo urls.py, debes definir la ruta que mostrará esta plantilla. Puedes hacerlo de la siguiente manera:

7 VISTAS EN DJANGO

```
# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('bienvenida/', views.bienvenida)
]
```

En este caso, tenemos la ruta llamada “bienvenida” que estará asociada a la función de vista bienvenida.

Y finalmente, en el archivo views.py, debes definir la función de vista que renderizará la plantilla y pasará la variable nombre.

```
# app/views.py
from django.shortcuts import render

def bienvenida(request):
    return render(request, 'index.html', {'nombre':
    'colega'})
```

En esta función, utilizamos el módulo render de Django para renderizar la plantilla index.html y pasar el contexto con la variable nombre que queremos mostrar.

¡Y eso es todo! Ahora, cuando un usuario visite la ruta /bienvenida/ en tu sitio web, verá el saludo personalizado basado en la variable nombre.

Reto:

7 VISTAS EN DJANGO

Ahora te reto a que intentes agregar más contenido a la plantilla index.html. Puedes agregar imágenes, enlaces o incluso darle tu propio estilo con CSS. ¡Diviértete explorando y creando tu sitio web personalizado!

7.3 Manejo de formularios

Ahora vamos a hablar sobre el manejo de formularios en Django. Los formularios nos permiten recopilar información de los usuarios y procesarla en nuestra aplicación web. Django nos proporciona una forma sencilla y estructurada de trabajar con formularios.

Para empezar, necesitamos crear un formulario en Django. Podemos hacer esto creando una clase que herede de forms.Form y añadiendo los campos que queremos recopilar. Por ejemplo, si queremos recopilar el nombre, el email y un mensaje, podemos crear un formulario llamado NombreForm de la siguiente manera:

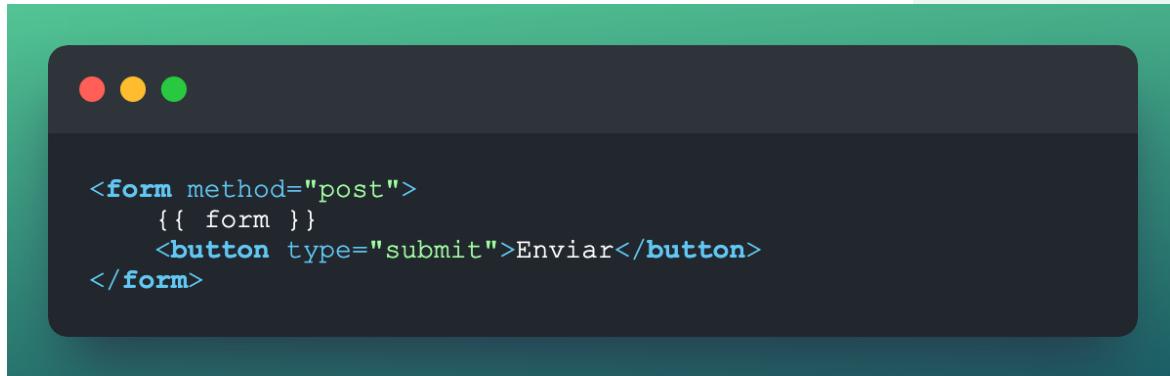
```
from django import forms

class NombreForm(forms.Form):
    nombre = forms.CharField(label='Nombre', max_length=50)
    email = forms.EmailField(label='Email', max_length=100)
    mensaje = forms.CharField(label='Mensaje',
        widget=forms.Textarea)
```

En este ejemplo, hemos añadido tres campos al formulario: nombre, email y mensaje. Cada campo tiene un tipo específico, como CharField para campos de texto y EmailField para campos de email. También hemos añadido etiquetas a los campos para que sean más fácilmente identificables.

7 VISTAS EN DJANGO

Una vez que tengamos nuestro formulario, necesitamos crear una plantilla en la que mostrarlo. Podemos hacer esto creando un archivo HTML y utilizando el formulario en él. Por ejemplo, podríamos crear un archivo llamado nombre.html con el siguiente contenido:



En esta plantilla, hemos utilizado la variable form para mostrar el formulario. La etiqueta {{ form }} se encarga de renderizar los campos del formulario.

Ahora que tenemos nuestro formulario y nuestra plantilla, necesitamos crear una vista en Django que maneje el envío del formulario. Podemos hacer esto creando una función de vista y utilizando el formulario en ella. Por ejemplo, podríamos tener una función llamada obtener_nombre en nuestro archivo views.py:

7 VISTAS EN DJANGO

```
from django.http import HttpResponse
from django.shortcuts import render
from django.views.decorators.csrf import csrf_exempt

from .forms import NombreForm

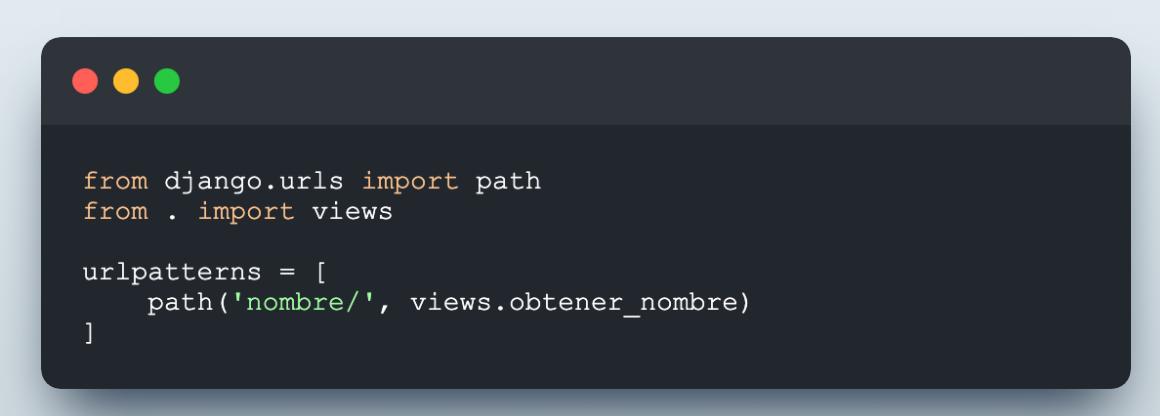
@csrf_exempt
def obtener_nombre(request):
    if request.method == 'POST':
        form = NombreForm(request.POST)
        if form.is_valid():
            nombre = form.cleaned_data['nombre']
            email = form.cleaned_data['email']
            mensaje = form.cleaned_data['mensaje']
            # Hacer algo con la información recopilada
            print(nombre, email, mensaje)
            return HttpResponse('Recibido!')
    else:
        form = NombreForm()

    return render(request, 'nombre.html', {'form': form})
```

En esta función de vista, primero comprobamos si el método de la solicitud es ‘POST’ , lo que significa que el formulario se ha enviado. Si es así, creamos una instancia del formulario con los datos enviados y comprobamos si es válido. Si el formulario es válido, podemos acceder a los datos recopilados utilizando la propiedad `cleaned_data`. En este ejemplo, simplemente imprimimos los datos por la consola.

Si el método de la solicitud no es ‘POST’ , simplemente creamos una instancia del formulario vacío.

Por último, necesitamos definir una URL en nuestro archivo `urls.py` para que podamos acceder a nuestra vista. Por ejemplo:



```
from django.urls import path
from . import views

urlpatterns = [
    path('nombre/', views.obtener_nombre)
]
```

Aquí estamos definiendo una URL llamada ‘nombre/’ que apunta a la función de vista obtener_nombre.

¡Y eso es todo! Ahora tenemos un formulario funcional en nuestra aplicación Django. Puedes probarlo abriendo la URL ‘nombre/’ en tu navegador y rellenando los campos.

¡Espero que esto te haya sido útil!

Reto:

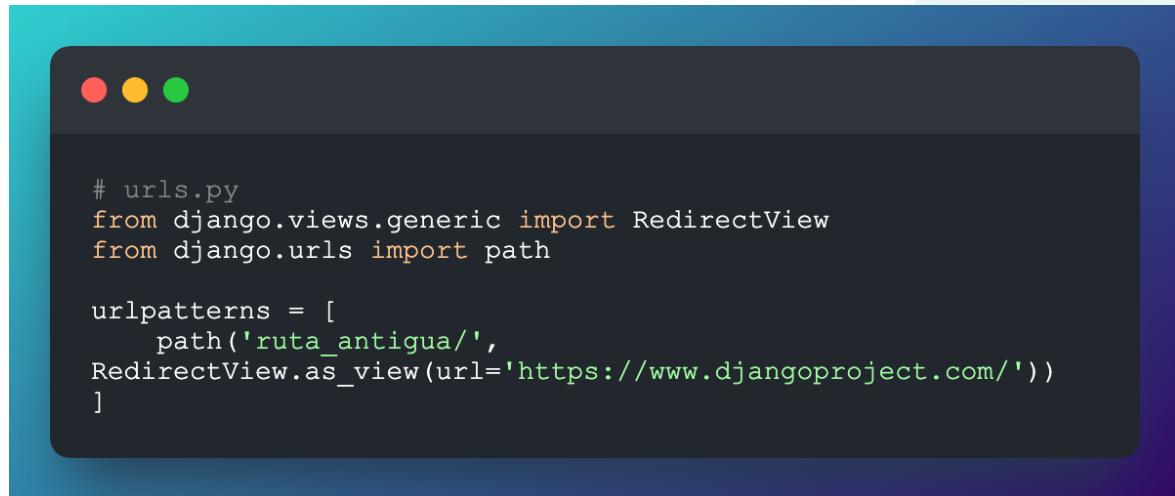
Para poner en práctica lo aprendido, te reto a añadir un campo adicional al formulario, por ejemplo, un campo de teléfono. ¡No olvides modificar la plantilla y la vista para incluir este nuevo campo!

7.4 Redirecciones y URLs

Ahora vamos a hablar sobre redirecciones y URLs en Django. Las redirecciones son una forma útil de dirigir a los usuarios hacia una nueva página cuando intentan acceder a una URL específica. En Django, podemos implementar redirecciones utilizando la clase `RedirectView` y configurando la URL hacia la cual queremos redirigir.

7 VISTAS EN DJANGO

Aquí tienes un ejemplo de cómo implementar una redirección en Django:



```
# urls.py
from django.views.generic import RedirectView
from django.urls import path

urlpatterns = [
    path('ruta_antigua/', RedirectView.as_view(url='https://www.djangoproject.com/'))
]
```

En este ejemplo, creamos una nueva ruta llamada 'ruta_antigua/' que apunta a una instancia de RedirectView. Pasamos la URL de destino que queremos que el usuario sea redirigido como argumento en el parámetro url. En este caso, estamos redirigiendo a los usuarios hacia la página principal de Django.

¡Genial! Ahora, cada vez que un usuario intente acceder a la ruta antigua, será redirigido automáticamente a la nueva URL.

Y ahora, ¡aquí va un reto para tí!

Reto:

Implementa una redirección en Django que lleve a los usuarios a la página de documentación oficial de Python.

¡Buenas habilidades!

7.5 Archivos estáticos

Los archivos estáticos son los archivos que no cambian, como imágenes, hojas de estilo CSS o archivos JavaScript. En Django, los archivos estáticos se almacenan

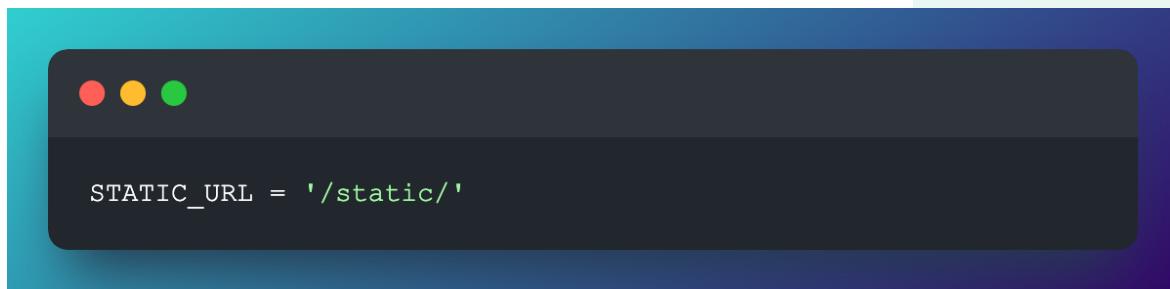
7 VISTAS EN DJANGO

en la carpeta “static” dentro de cada aplicación.

Aquí tienes un ejemplo de cómo trabajar con archivos estáticos en Django:

Primero, asegúrate de tener instalada la aplicación ‘django.contrib.staticfiles’ en tu archivo “settings.py” dentro de la lista “INSTALLED_APPS” . Esto te permitirá acceder a los archivos estáticos en tus vistas.

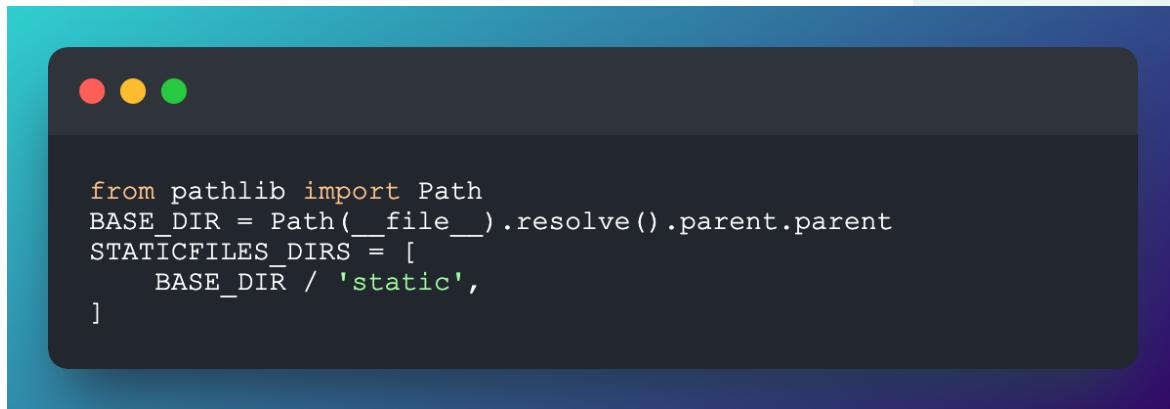
Luego, configura la URL base para servir los archivos estáticos. Puedes hacerlo agregando la siguiente línea en tu archivo “settings.py” :



```
STATIC_URL = '/static/'
```

Esta configuración te permitirá acceder a los archivos estáticos a través de la URL “/static/” .

A continuación, debes especificar la ubicación de tus archivos estáticos en tu aplicación. Puedes hacerlo agregando la siguiente línea en tu archivo “settings.py” :

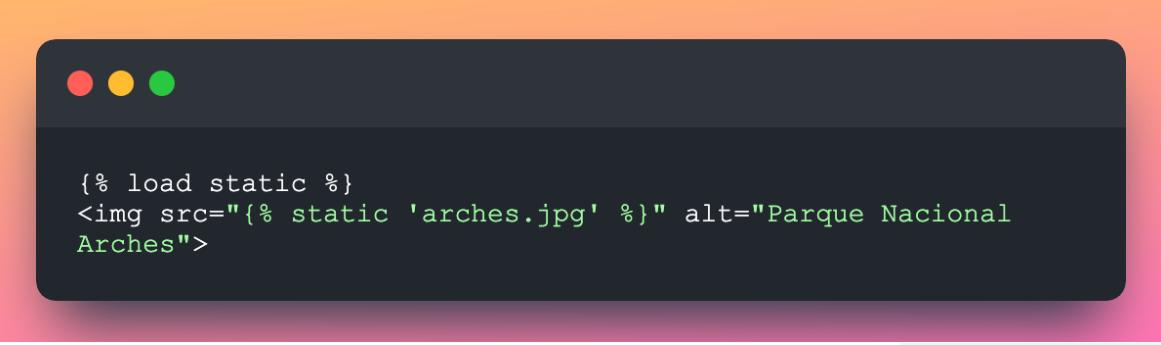


```
from pathlib import Path
BASE_DIR = Path(__file__).resolve().parent.parent
STATICFILES_DIRS = [
    BASE_DIR / 'static',
]
```

Esto le dirá a Django que busque archivos estáticos en la carpeta “static” dentro de tu aplicación.

7 VISTAS EN DJANGO

Ahora, supongamos que tienes un archivo de imagen llamado “arches.jpg” en la carpeta “static” de tu aplicación. Puedes mostrar esta imagen en una vista utilizando el siguiente código en tu archivo “imagen.html” :



```
{% load static %}  

```

¡Y eso es todo! Ahora, cuando accedas a la vista correspondiente a “imagen.html”, verás la imagen “arches.jpg” del Parque Nacional Arches.

¡Espero que esto te haya sido útil!

Reto:

Inténtalo tú mismo. Crea una nueva vista en tu archivo “urls.py”, haz referencia a ella en tu archivo “views.py” y muestra un archivo estático en tu nuevo template.

7.6 Manejo de errores

El manejo de errores es una parte crucial en el desarrollo de aplicaciones web, ya que te permite controlar y solucionar cualquier problema que pueda surgir en tu código. Pero no te preocupes, ¡no es tan complicado como parece!

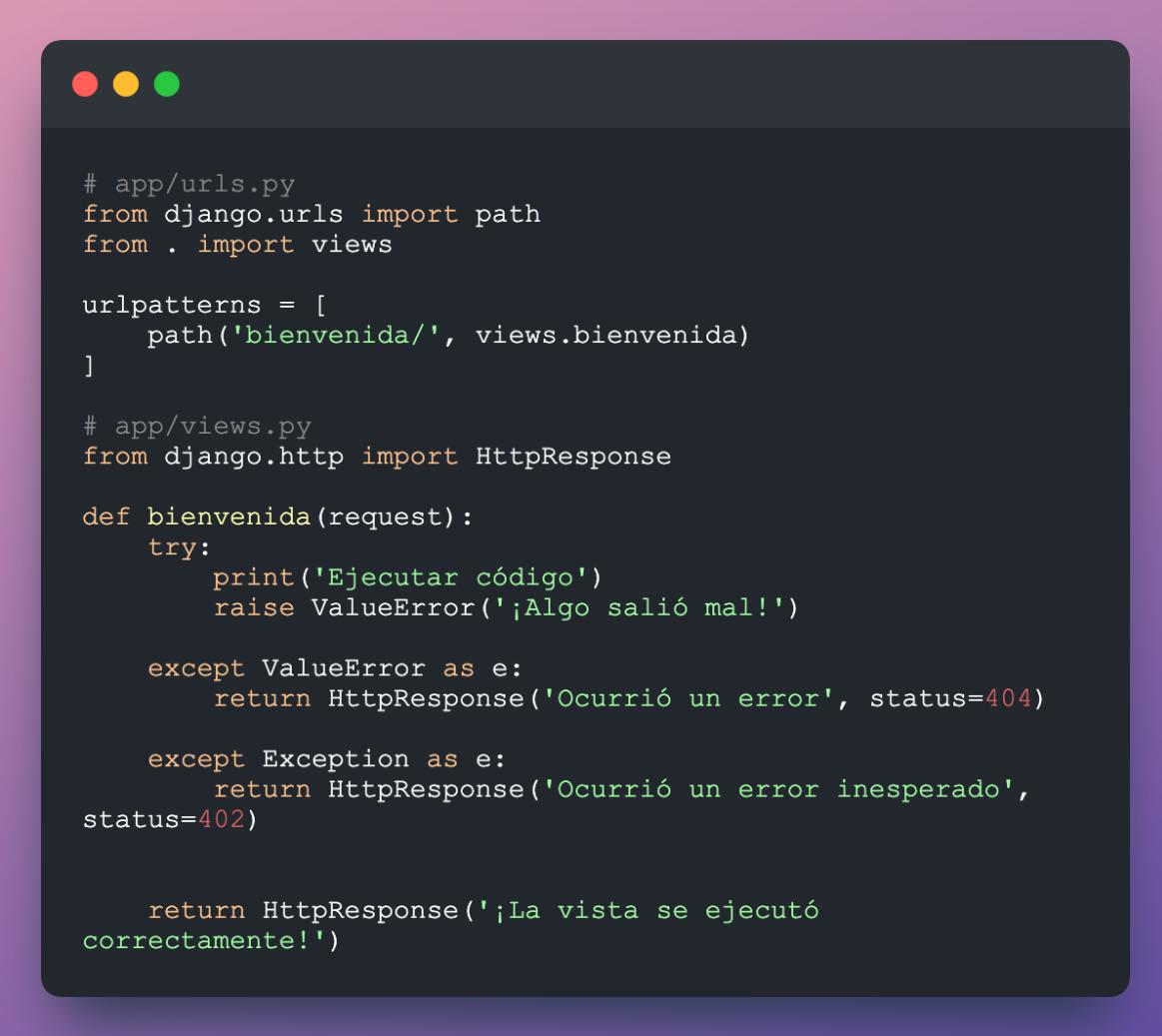
Imagina que estás construyendo una aplicación y te encuentras con un error. No te desanimes, ¡es normal que los errores ocurran! Pero lo importante es cómo los manejes.

En Django, puedes utilizar bloques try-except para capturar y controlar los errores que puedan ocurrir durante la ejecución de tu código. Te recomendaría siempre utili-

7 VISTAS EN DJANGO

zar bloques try-except alrededor de cualquier sección de código que pueda generar un error.

Aquí tienes un ejemplo de cómo manejar errores en Django:



```
# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('bienvenida/', views.bienvenida)
]

# app/views.py
from django.http import HttpResponse

def bienvenida(request):
    try:
        print('Ejecutar código')
        raise ValueError(';Algo salió mal!')

    except ValueError as e:
        return HttpResponse('Ocurrió un error', status=404)

    except Exception as e:
        return HttpResponse('Ocurrió un error inesperado',
status=402)

    return HttpResponse(';La vista se ejecutó
correctamente!')
```

En este ejemplo, definimos una vista “bienvenida” que ejecuta código dentro de un bloque try. Dentro de ese bloque, levantamos intencionalmente un error utilizando la instrucción raise. Luego, capturamos ese error utilizando un bloque except y devolvemos una respuesta personalizada.

8 TEMPLATES EN DJANGO

Si ocurre un `ValueError`, devolvemos una respuesta con el mensaje “Ocurrió un error” y un código de estado 404. Si ocurre cualquier otro tipo de error, devolvemos una respuesta con el mensaje “Ocurrió un error inesperado” y un código de estado 402.

Recuerda que también puedes utilizar bloques `finally` para ejecutar código que siempre debe ser ejecutado, independientemente de si se produjo un error o no.

¡Ahora te reto a que intentes manejar tus propios errores en Django! Empieza por identificar una parte de tu código que pueda arrojar un error y utiliza bloques `try-except` para manejarlo de forma adecuada. ¡No tengas miedo de cometer errores, es parte del proceso de aprendizaje!

Recuerda, ¡los errores son oportunidades de aprendizaje! No te desanimes y sigue adelante. ¡Siempre habrá una forma de solucionar los problemas y mejorar tu código! ¡Buena suerte!

8 Templates en Django

8.1 Variables en plantillas

Hoy vamos a hablar sobre las variables en las plantillas de Django. ¡Pero no te preocupes, no es tan complicado como parece!

En Django, podemos pasar variables desde nuestras vistas a las plantillas para mostrar información dinámica en nuestro sitio web. Esto nos permite personalizar y mostrar datos relevantes para cada usuario.

Veamos un ejemplo que te ayudará a comprenderlo mejor:

8 TEMPLATES EN DJANGO

```
# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('usuario/', views.get_usuario)
]
```

En este ejemplo, estamos configurando una URL para la vista que mostrará la información de un usuario.

```
<!-- app/templates/usuario.html -->
#{ comentario #}
<ul>
    <li>Nombre: {{ usuario.nombre }} {{ usuario.apellido }}</li>
    <li>Correo electrónico: {{ usuario.email }}</li>
    <li>Edad: {{ usuario.perfil.edad }}</li>
    <li>Ciudad: {{ usuario.perfil.ciudad }}</li>
</ul>
```

En la plantilla, utilizamos las llaves dobles {{ }} para indicar que queremos mostrar una variable. En este caso, estamos accediendo a las propiedades de la variable usuario, como el nombre, el apellido, el correo electrónico, la edad y la ciudad.

8 TEMPLATES EN DJANGO

```
# app/views.py
from django.shortcuts import render

def get_usuario(request):

    usuario = {
        'nombre': 'Gustavo',
        'apellido': 'Ulloa',
        'email': 'guga@example.com',
        'perfil': {
            'edad': 30,
            'ciudad': 'Sangolquí'
        }
    }

    return render(request, 'usuario.html', {'usuario': usuario})
```

En la vista `get_usuario`, creamos un diccionario llamado `usuario` con la información del usuario. Luego, utilizamos la función `render` para enviar la plantilla `usuario.html` y la variable `usuario` a la plantilla.

¡Y eso es todo! Ahora, cuando accedas a la URL `/usuario/`, verás la información del usuario en la plantilla.

Ahora, aquí está el reto simple para que practiques:

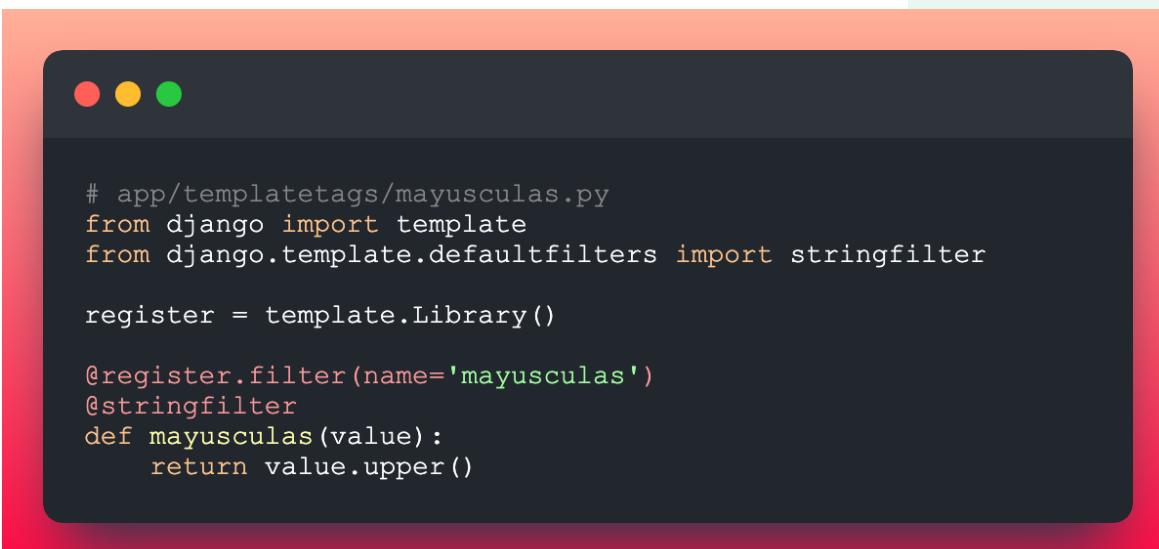
Intenta crear una vista en Django que pase una variable y muestre su contenido en una plantilla. Puedes utilizar el ejemplo anterior como referencia para guiarte.

¡Ánimo, tú puedes hacerlo!

8.2 Filtros en plantillas

Los filtros son una forma sencilla y poderosa de realizar transformaciones en los datos que se muestran en las plantillas. Son como pequeñas funciones que toman un valor de entrada, lo procesan y devuelven un valor transformado.

Imagínate que tienes un texto en minúsculas y quieres mostrarlo todo en mayúsculas en tu plantilla. Puedes crear un filtro personalizado utilizando el decorador `@register.filter` y `@stringfilter`. Por ejemplo:



```
# app/templatetags/mayusculas.py
from django import template
from django.template.defaultfilters import stringfilter

register = template.Library()

@register.filter(name='mayusculas')
@stringfilter
def mayusculas(value):
    return value.upper()
```

En este ejemplo, hemos creado un filtro llamado `mayusculas` que toma un valor y lo convierte en mayúsculas utilizando el método `upper()` de Python.

Luego, en nuestra plantilla, podemos cargar el filtro utilizando el tag `{% load %}` y utilizarlo en cualquier variable. Por ejemplo:

8 TEMPLATES EN DJANGO

```
<!-- app/templates/texto.html -->
<p>Tamaño de texto: {{ mi_texto|length }}</p>
{%
    load mayusculas %
}
<p>Texto en mayúsculas: {{ mi_texto|mayusculas }}</p>
```

En este caso, estamos utilizando el filtro `mayusculas` para transformar la variable `mi_texto` en mayúsculas antes de mostrarla en la plantilla.

Para probarlo, necesitamos crear una vista en nuestro archivo `views.py` que renderice esta plantilla. Por ejemplo:

```
# app/views.py
from django.shortcuts import render

def get_texto(request):
    return render(request, 'texto.html', { 'mi_texto': 'python' })
```

Finalmente, necesitamos agregar una nueva URL en nuestro archivo `urls.py` que mapee a esta vista. Por ejemplo:

```
# app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('texto/', views.get_texto)
]
```

Ahora, podemos acceder a nuestra página en <http://localhost:8000/texto/> y veremos el resultado.

¡Y ahí lo tienes! Ahora sabes cómo utilizar filtros en plantillas en Django para hacer transformaciones en los datos. Recuerda que puedes crear tus propios filtros según tus necesidades. ¡Diviértete y sigue explorando Django!

Reto:

¡Tu reto es crear un filtro personalizado que tome un número y devuelva su valor al cuadrado! Puedes utilizar la función `math.pow()` de Python para calcularlo.

8.3 Tags de plantillas

¡Presta atención porque esto te ayudará a crear páginas web increíbles!

Los Tags de plantillas son pequeñas piezas de código en Django que nos permiten realizar diferentes acciones dentro de nuestras plantillas. Podemos pensar en ellos como comandos que le indican a Django qué hacer en una determinada parte de la página.

Un ejemplo de Tag de plantilla en Django es el `{% load static %}`. Este tag nos permite cargar archivos estáticos, como imágenes, CSS o JavaScript, en nuestras plantillas.

8 TEMPLATES EN DJANGO

Por ejemplo, si queremos mostrar una imagen en nuestra página, podemos utilizar el tag `{% static %}` para indicar la URL de la imagen.



A screenshot of a code editor window titled 'app/templates/imagen.html'. The code inside the editor is:

```
<!-- app/templates/imagen.html -->
<!-- app/static/arches.jpg -->
{% load static %}
![Parque Nacional Arches]({% static 'arches.jpg' %})
```

En este ejemplo, hemos cargado la imagen 'arches.jpg' ubicada en la carpeta 'app/static' utilizando el tag `{% static %}`. Luego, la mostramos utilizando el tag de HTML.

¡Pero espera, hay más! Los Tags de plantillas en Django también nos permiten realizar bucles, condicionales, incluir otras plantillas y mucho más. Son una herramienta muy poderosa que nos ayuda a crear páginas web dinámicas y personalizadas.

Ahora, ¿estás listo para el reto? ¡Aquí va!

Reto:

Crea una nueva plantilla en Django y utiliza un tag de plantilla para mostrar una lista de nombres de personas. Puedes utilizar un bucle para recorrer la lista y mostrar cada nombre en un elemento

de HTML. ¡Anímate a probarlo!

¡Buena suerte y que el código esté contigo!

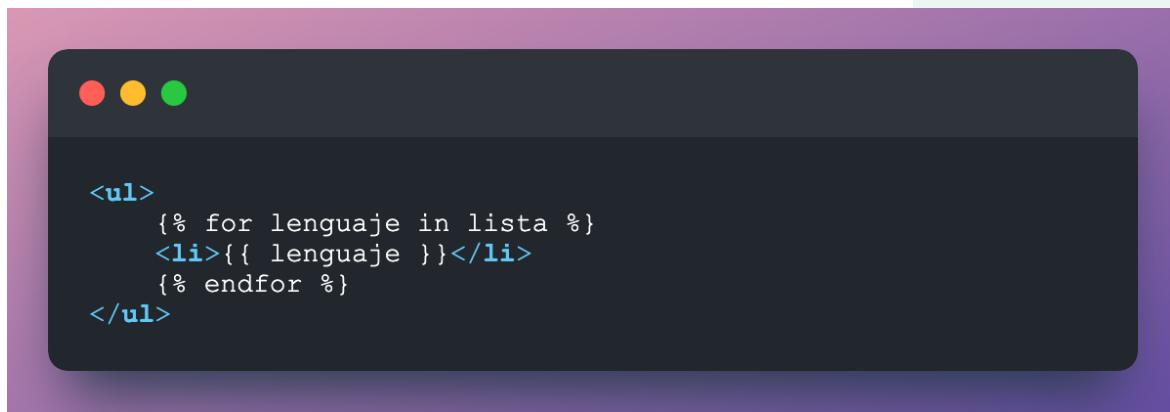
8.4 Iteración en plantillas

¿Estás listo para adentrarte en el increíble mundo de la iteración en plantillas en Django? ¡Claro que estás listo! Permíteme guiarte a través de este emocionante concepto.

Imagínate que tienes una lista de lenguajes de programación que deseas mostrar en una página web. En lugar de escribir manualmente cada lenguaje en tu plantilla, puedes aprovechar la potencia de la iteración en plantillas de Django.

En tu archivo de plantilla `plantilla.html`, puedes utilizar un bucle `for` para recorrer tu lista de lenguajes. Dentro de este bucle, puedes mostrar cada lenguaje utilizando la sintaxis `{{ lenguaje }}`. ¿No es genial?

Volvamos al ejemplo. Imagina que tienes una lista llamada `lista` que contiene los lenguajes de programación: `html`, `css` y `javascript`. Puedes hacer lo siguiente en tu archivo `plantilla.html`:

A screenshot of a terminal window with a dark background and light-colored text. It shows a portion of a Django template file. The code includes an opening `` tag, a `{% for lenguaje in lista %}` tag, an `{{ lenguaje }}` tag, a `{% endfor %}` tag, and a closing `` tag.

```
<ul>
    {% for lenguaje in lista %}
        <li>{{ lenguaje }}</li>
    {% endfor %}
</ul>
```

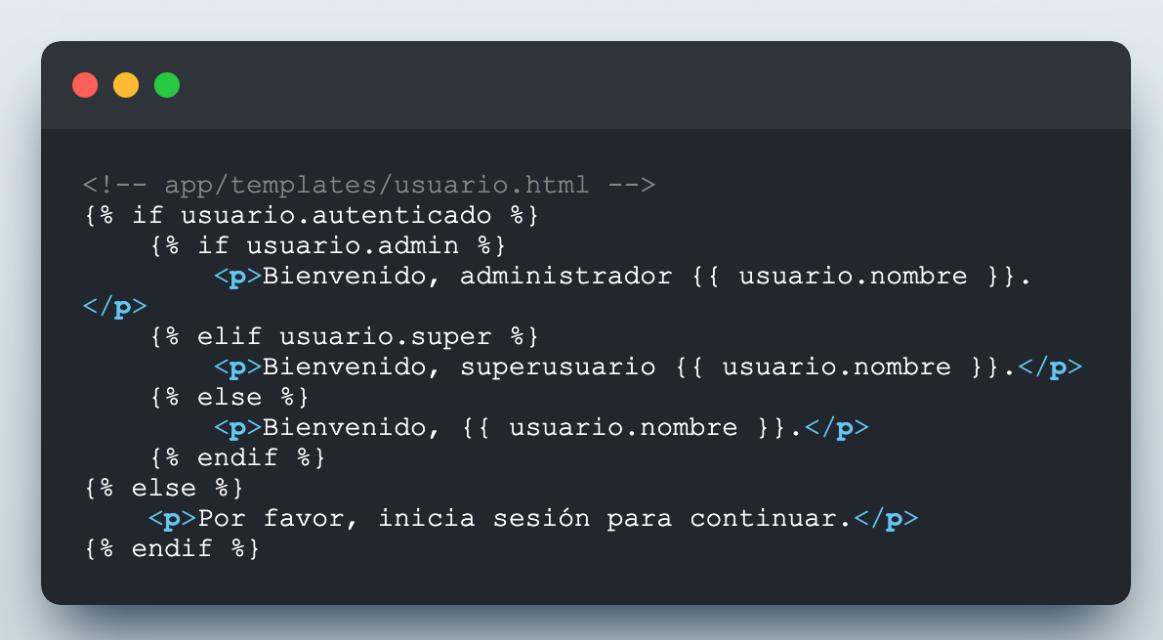
Este código creará una lista desordenada en HTML, y dentro de ese bucle `for`, cada lenguaje de tu lista será representado por un elemento ``. Así de simple y poderoso es utilizar la iteración en plantillas de Django.

Puedes asociar esta plantilla a una URL en tu archivo `urls.py` y utilizar la función `render()` en tu archivo `views.py` para mostrar esta plantilla en una página web. ¿Ve cómo todo encaja perfectamente?

¿Quieres hacer un reto? ¡Perfecto! Aquí tienes un desafío simple para ti: Intenta agregar un cuarto lenguaje a la lista, como por ejemplo “Python” . Prueba tu código y observa cómo se actualiza automáticamente en tu página web. ¡Diviértete experimentando!

8.5 Condicionales en plantillas

Imagina que tenemos un sitio web donde los usuarios pueden iniciar sesión y tienen diferentes roles, como administrador o superusuario. Queremos mostrar un mensaje personalizado de bienvenida según el rol del usuario.



The screenshot shows a terminal window with three colored dots (red, yellow, green) at the top. The main area contains the following Django template code:

```
<!-- app/templates/usuario.html -->
{% if usuario.autenticado %}
    {% if usuario.admin %}
        <p>Bienvenido, administrador {{ usuario.nombre }}.</p>
    {% elif usuario.super %}
        <p>Bienvenido, superusuario {{ usuario.nombre }}.</p>
    {% else %}
        <p>Bienvenido, {{ usuario.nombre }}.</p>
    {% endif %}
    {% else %}
        <p>Por favor, inicia sesión para continuar.</p>
    {% endif %}
```

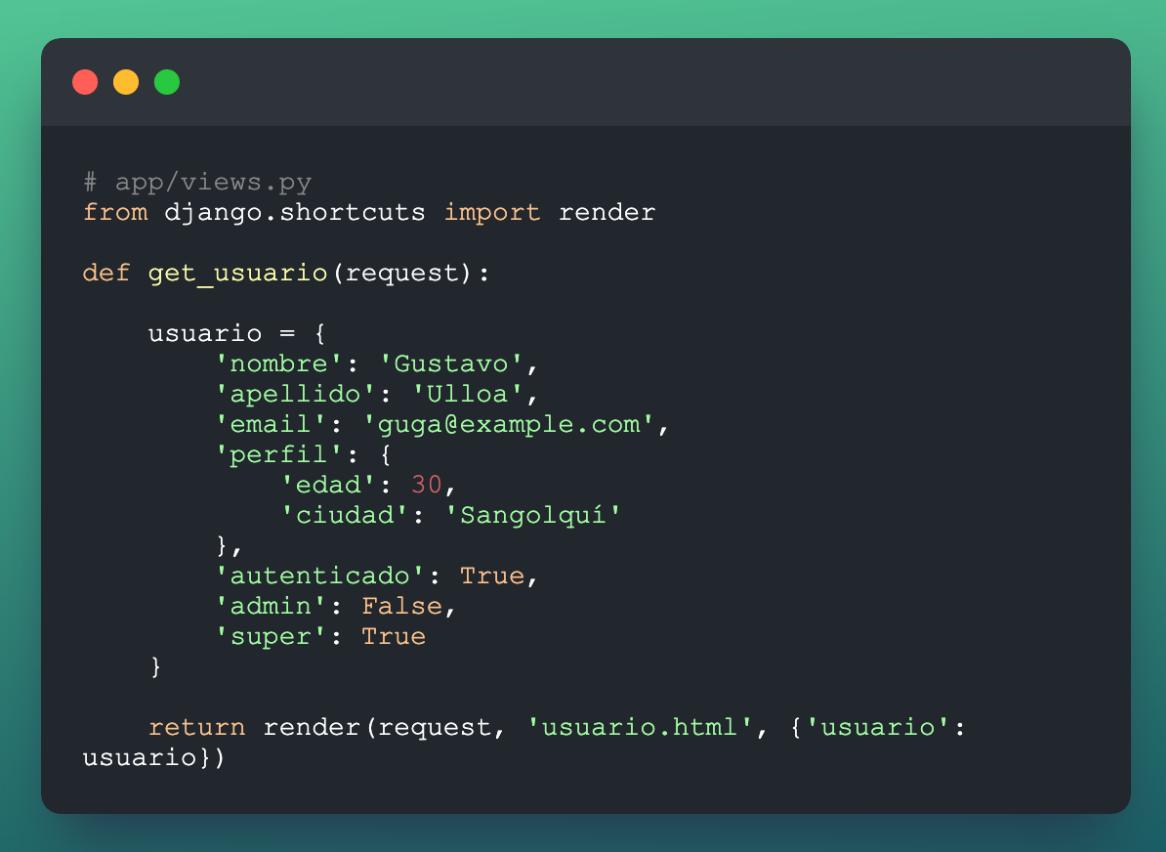
En el ejemplo anterior, estamos utilizando los condicionales de Django en una plantilla. Primero, verificamos si el usuario ha iniciado sesión usando `{% if usuario.autenticado %}`. Dentro de ese bloque condicional, verificamos si el usuario es un administrador con `{% if usuario.admin %}`. Si es cierto, mostramos un mensaje personalizado de bienvenida. Si no es un administrador, verificamos si

8 TEMPLATES EN DJANGO

es un superusuario con `{% elif usuario.super %}`. Si también es cierto, mostramos otro mensaje personalizado de bienvenida. Si no es ni un administrador ni un superusuario, simplemente mostramos un mensaje de bienvenida genérico con `{{ usuario.nombre }}`.

Si el usuario no ha iniciado sesión, mostramos un mensaje que les pide que inicien sesión para continuar.

Pero, ¿cómo obtenemos los datos del usuario para pasarlo a la plantilla? No te preocupes, tengo otro ejemplo para ti.



```
# app/views.py
from django.shortcuts import render

def get_usuario(request):

    usuario = {
        'nombre': 'Gustavo',
        'apellido': 'Ulloa',
        'email': 'guga@example.com',
        'perfil': {
            'edad': 30,
            'ciudad': 'Sangolquí'
        },
        'autenticado': True,
        'admin': False,
        'super': True
    }

    return render(request, 'usuario.html', {'usuario': usuario})
```

En este ejemplo, tenemos una vista llamada `get_usuario` que simplemente devuelve los datos de un usuario en forma de diccionario. Estos datos se pasan a la plantilla

'usuario.html' utilizando el diccionario 'usuario' como contexto.

Y eso es todo, ¡así es como puedes usar condicionales en plantillas en Django para mostrar mensajes personalizados según los roles de los usuarios! Recuerda, ¡Django te facilita la vida!

Ahora, aquí viene el reto: ¿puedes probar a agregar una condición adicional para mostrar un mensaje especial si el usuario tiene una edad mayor de 18 años?

8.6 Herencia de plantillas

Imagina que estás construyendo un sitio web y tienes muchas páginas que comparten elementos comunes, como la estructura básica, la barra de navegación o el pie de página. Sería un gran dolor de cabeza tener que duplicar todo el código en cada una de las páginas, ¿verdad?

Aquí es donde entra en juego la herencia de plantillas en Django. Puedes crear una plantilla base con todos los elementos comunes y luego heredar esta plantilla en cada una de las páginas que necesites.

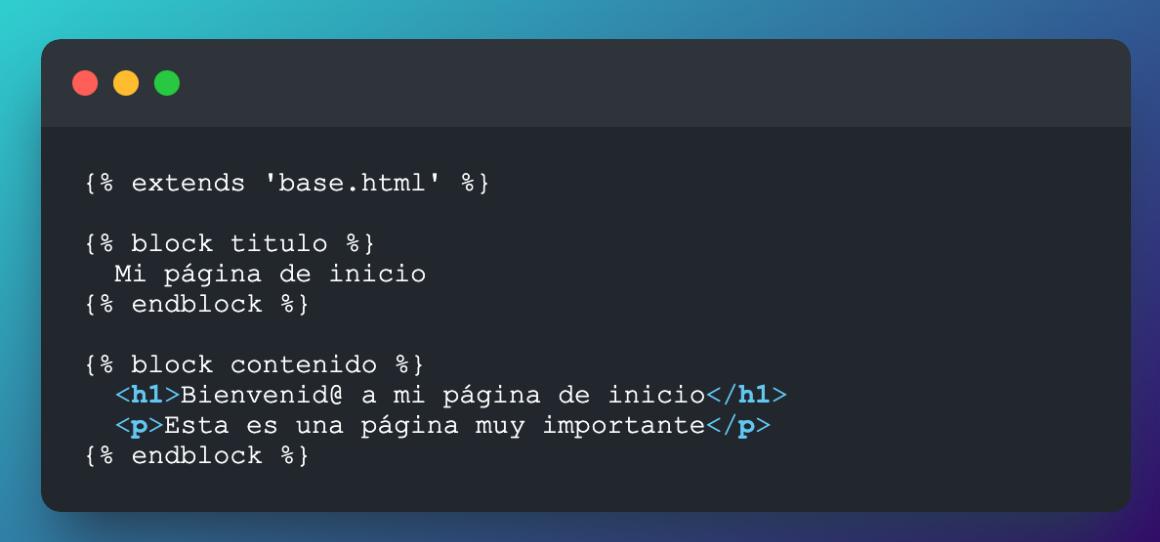
Por ejemplo, imagina que tienes una plantilla base llamada `base.html`. En esta plantilla, tienes el encabezado con el título de la página en un bloque llamado `titulo` y el contenido principal en un bloque llamado `contenido`.

8 TEMPLATES EN DJANGO



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block titulo %} {% endblock %}</title>
</head>
<body>
    {% block contenido %}
        <h1>Bienvenido a mi sitio web</h1>
    {% endblock %}
</body>
</html>
```

Ahora, en tu página de inicio, puedes extender esta plantilla base usando la etiqueta `{% extends 'base.html' %}`. Esto significa que toda la estructura y el contenido de la plantilla base se heredarán en tu página de inicio.



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

{% block titulo %}
    Mi página de inicio
{% endblock %}

{% block contenido %}
    <h1>Bienvenid@ a mi página de inicio</h1>
    <p>Esta es una página muy importante</p>
{% endblock %}
```

Y eso es todo. ¡Ahora tienes una página de inicio con todos los elementos comunes

9 SIGUIENTES PASOS

de la plantilla base!

Recuerda que puedes heredar la plantilla base en todas las páginas que necesites y sobreescibir los bloques según tus necesidades. Esto te permitirá mantener un código limpio y organizado, ¡sin tener que duplicar todo el contenido una y otra vez!

¡Desafío para tí!

Ahora te desafío a que crees una nueva página en tu sitio web, heredando la plantilla base y agregando tu propio contenido. Puedes personalizar el título, el contenido o incluso agregar nuevos bloques si lo deseas. ¡Diviértete y experimenta con la herencia de plantillas en Django!

9 Siguientes pasos

9.1 Herramientas

Aquí te presento algunas de las herramientas más utilizadas con Django:

1. Django REST Framework - Es una potente biblioteca que permite construir y diseñar APIs web de manera rápida y sencilla. Proporciona una gran cantidad de funciones que facilitan el proceso de desarrollo de APIs RESTful. Puedes obtener más información en su [sitio oficial](#).

2. Django Debug Toolbar - Esta herramienta es muy útil durante el desarrollo, ya que proporciona información detallada sobre las consultas a la base de datos, tiempos de carga y otros datos relevantes para identificar cuellos de botella en el rendimiento de tu aplicación Django. Puedes encontrar más sobre ella en el [repositorio oficial](#).

3. Celery - Es una herramienta de programación en segundo plano que se integra muy bien con Django. Permite ejecutar tareas asíncronas, programadas o periódicas en tu aplicación web. Puedes obtener más información en la [documentación](#)

9 SIGUIENTES PASOS

oficial.

4. Django Crispy Forms - Es una biblioteca que te permite generar formularios HTML con una apariencia elegante y personalizable. Proporciona una sintaxis sencilla para definir los campos de tu formulario y también te permite personalizar su apariencia. Puedes consultar más información en el [sitio oficial](#).

5. Django Cors Headers - Esta herramienta es útil si necesitas habilitar solicitudes HTTP desde dominios diferentes al tuyo. Te permite añadir encabezados de Autorización CORS a tus respuestas HTTP. Puedes obtener más información en su [repositorio oficial](#).

Estas son solo algunas de las muchas herramientas disponibles para Django. Espero que esta información te sea útil. ¡Buena suerte con tu aprendizaje de Django!

9.2 Recursos

Aquí tienes algunos de los recursos en línea más utilizados para aprender y trabajar con Django:

1. **Documentación oficial de Django** - Esta es la mejor fuente de información para aprender Django desde cero. Proporciona una guía completa sobre cómo empezar, conceptos fundamentales, referencia de API y ejemplos prácticos.
[Enlace a la documentación oficial de Django](#)
2. **DjangoGirls Tutorial** - Este tutorial interactivo está dirigido a principiantes y cubre los conceptos básicos de Django. Proporciona instrucciones paso a paso para construir un blog simple. [Enlace al tutorial de DjangoGirls](#)
3. **Real Python Django Learning Path** - Real Python ofrece una serie de artículos, tutoriales y cursos en línea que cubren Django. Su Learning Path para Django es una excelente manera de aprender el framework desde cero hasta niveles más avanzados. [Enlace a la página de Real Python Django Learning Path](#)

9 SIGUIENTES PASOS

4. **Django for Beginners** - Este libro escrito por William S. Vincent es una excelente guía para principiantes que desean aprender Django. Cubre los conceptos básicos, proporciona ejemplos prácticos y explica las mejores prácticas. [Enlace a Django for Beginners](#)
5. **Django Reddit** - Esta comunidad en línea es un gran recurso para obtener ayuda y discutir temas relacionados con Django. Los usuarios publican preguntas y problemas, y otros miembros de la comunidad brindan soluciones y consejos útiles. [Enlace a Django Reddit](#)

Estos son solo algunos de los recursos en línea más populares para Django. ¡Espero que te sean útiles!

9.3 ¿Que viene después?

Después de aprender Django, hay varios pasos que puedes tomar para seguir trabajando en tu desarrollo como desarrollador web. Algunas opciones incluyen:

1. Profundizar en Django: Puedes seguir explorando y aprendiendo más sobre Django, explorando sus características avanzadas y descubriendo nuevas formas de optimizar y mejorar tus proyectos.
2. Aprender otro framework: Si ya te sientes cómodo con Django, puedes considerar aprender otro framework web, como Flask, Ruby on Rails o Laravel. Esto ampliará tu conocimiento y te permitirá trabajar con diferentes tecnologías en el futuro.
3. Estudiar HTML, CSS y JavaScript: Si aún no lo has hecho, es muy recomendable tener conocimientos sólidos en HTML, CSS y JavaScript. Estas son las bases del desarrollo web y te permitirán crear interfaces atractivas y dinámicas para tus aplicaciones.
4. Familiarizarte con bases de datos: Aprender sobre bases de datos relacionales y cómo interactuar con ellas es esencial para desarrollar aplicaciones web

9 SIGUIENTES PASOS

completas. Puedes explorar tecnologías como MySQL, PostgreSQL o MongoDB para adquirir experiencia en la gestión de datos.

5. Aprender a usar herramientas de control de versiones: El uso de herramientas como Git te permitirá trabajar de manera colaborativa y gestionar eficientemente el código de tus proyectos. Aprende a usar Git y familiarízate con los conceptos de ramas, fusiones y repositorios para mejorar tu flujo de trabajo.
6. Desarrollar proyectos personales: La mejor forma de seguir mejorando tus habilidades es desarrollando proyectos personales. Estos te permitirán aplicar lo que has aprendido y enfrentarte a desafíos reales. Además, podrás construir tu propio portafolio y mostrar tus habilidades a futuros empleadores.
7. Mantenerse actualizado: La tecnología web está en constante evolución, por lo que es importante mantenerse actualizado sobre las últimas tendencias y avances en el campo. Sigue blogs, participa en comunidades en línea y asiste a conferencias para estar al tanto de los últimos avances y oportunidades en tu área.

Recuerda que el aprendizaje continuo es clave en el desarrollo web. Mantén una mentalidad abierta y nunca dejes de buscar nuevas formas de mejorar tus habilidades y conocimientos. ¡Buena suerte en tu camino de desarrollo!

9.4 Preguntas de entrevista

Aquí te presento una lista con preguntas comunes en una entrevista de Django, junto con sus respuestas correspondientes:

1. ¿Qué es Django y para qué se utiliza? Django es un framework de desarrollo web de alto nivel que se utiliza para crear aplicaciones web de manera rápida y eficiente.
2. ¿Cuál es la diferencia entre una clase y una función basada en vistas en Django? Una clase basada en vistas es un enfoque orientado a objetos para mane-

9 SIGUIENTES PASOS

jar las solicitudes web en Django, mientras que una función basada en vistas es un enfoque más simple y tradicional.

3. ¿Qué es una migración en Django? Una migración en Django es un archivo que contiene instrucciones para modificar la estructura de la base de datos.
4. ¿Cuál es la principal ventaja de utilizar Django ORM? La principal ventaja de utilizar Django ORM es que nos permite interactuar con la base de datos de manera intuitiva y segura, sin tener que escribir consultas SQL directamente.
5. ¿Qué es una “queryset” en Django? Un queryset en Django es una representación de una consulta a la base de datos, que permite recuperar, filtrar y ordenar datos de manera eficiente.
6. ¿Cómo se crea un nuevo proyecto en Django? Para crear un nuevo proyecto en Django, se utiliza el comando `django-admin startproject nombre_proyecto`.
7. ¿Cuál es la función de los archivos `settings.py` en Django? Los archivos `settings.py` en Django contienen la configuración del proyecto, como la base de datos, las claves secretas, las aplicaciones instaladas, entre otros.
8. ¿Qué es el archivo `urls.py` en Django? El archivo `urls.py` en Django contiene las rutas URL del proyecto y las vistas asociadas a cada ruta.
9. ¿Cómo se crea una nueva aplicación en Django? Para crear una nueva aplicación en Django, se utiliza el comando `python manage.py startapp nombre_app`.
10. ¿Cuál es la diferencia entre `HttpRequest` y `HttpResponse` en Django? `HttpRequest` es un objeto que representa una solicitud HTTP, mientras que `HttpResponse` es un objeto que representa la respuesta HTTP enviada al usuario.

Espero que esta lista te sea de ayuda para prepararte para tus entrevistas. ¡Buenas habilidades!