Engineer

Django

D framework de Python para crear aplicaciones web

Qué es

Django es un framework web escrito en Python: llega una solicitud, la configuración de URL la dirige a una función o clase llamada vista y esta devuelve una respuesta, normalmente HTML generado a partir de una plantilla o JSON. Alrededor de ese núcleo incluye las piezas que necesitan la mayoría de los sitios respaldados por una base de datos: un mapeador objeto-relacional, migraciones de esquema, formularios, autenticación con permisos y una interfaz de administración generada automáticamente, que su documentación considera una de sus funciones más populares. El proyecto se escribe con D mayúscula; el paquete que se instala e importa es `django`, en minúsculas.

Nació en una redacción. En otoño de 2003, Adrian Holovaty y Simon Willison, desarrolladores de World Online, el departamento web de un periódico de Lawrence, Kansas, pasaron de PHP a Python para los sitios que creaban con los plazos del periodismo. Mientras construían sitios como Lawrence.com, extrajeron del trabajo un framework genérico, lo perfeccionaron durante dos años y, en verano de 2005, World Online lo publicó como código abierto. Lleva el nombre de Django Reinhardt, guitarrista de jazz manouche; se pronuncia JANG-oh, con la D muda, y se distribuye bajo la licencia BSD de 3 cláusulas.

Con la numeración A.B, las versiones con nuevas funciones llegaban aproximadamente cada ocho meses y solo las designadas de soporte a largo plazo recibían tres años de soporte; la versión 5.2 LTS tiene soporte hasta abril de 2028. Django 6.2, una versión LTS prevista para abril de 2027, es la última bajo esa política. A partir de Django 2028, las versiones con nuevas funciones llevan el año en su nombre, llegan cada enero y todas reciben el mismo período de soporte de tres años.

Las preguntas frecuentes del proyecto proponen llamar a Django un framework MTV, por model, template y view (modelo, plantilla y vista): el modelo define los datos, la vista decide qué datos ve el usuario y la plantilla decide su aspecto. En esa interpretación, el controlador es el propio framework, la maquinaria que envía cada solicitud a la vista correcta. Los modelos son clases de Python y las consultas sobre ellos son QuerySets, que se evalúan de forma diferida: se puede construir, filtrar y pasar un QuerySet sin tocar la base de datos, que solo se consulta cuando algo lo evalúa, como un bucle. Los cambios en los modelos se convierten en migraciones: `makemigrations` empaqueta cada cambio en un archivo que la documentación compara con un commit en un sistema de control de versiones para el esquema, y `migrate` lo aplica. Los objetos relacionados se cargan bajo demanda, una consulta cada vez, a menos que el QuerySet los solicite de antemano.

```
# Book has a ForeignKey to Author.
# 1 query for the books, then 1 per book for its author.
for book in Book.objects.all():
    print(book.author.name)

# 1 query: an SQL join loads each author with its book.
for book in Book.objects.select_related("author"):
    print(book.author.name)

# 2 queries: the authors, then all their books in one batch.
for author in Author.objects.prefetch_related("book_set"):
    print(author.name, [b.title for b in author.book_set.all()])
```

Utilice el concepto en una decisión

Aplique «¿Qué es Django?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Django importa por lo que establece como comportamiento predeterminado. Sus QuerySets construyen SQL manteniendo separados los parámetros y el texto de la consulta, de modo que el controlador de la base de datos escapa los valores aportados por el usuario. Sus plantillas escapan los caracteres más peligrosos en

HTML. Su protección CSRF comprueba un secreto en cada POST. Django almacena las contraseñas con PBKDF2 sobre SHA-256, una técnica de estiramiento de claves que, según su documentación, recomienda el NIST. Un equipo que escribe código Django corriente obtiene esas protecciones sin tener que solicitarlas.

Los mismos comportamientos predeterminados delimitan el riesgo. Una sola línea puede anular cualquiera de ellos: una consulta SQL directa, un `mark_safe`, un `csrf_exempt` o un `DEBUG = True` que se deja activado en el servidor en servicio. Cada una de esas líneas parece inocua en una revisión. El ORM plantea un intercambio parecido en rendimiento: es fácil escribir una vista cuyo número de consultas crezca con la longitud de la página, y el código Python se lee igual en ambos casos.

Recomendamos elegir Django cuando los datos determinan la forma de la aplicación: registros que las personas crean y editan, cuentas con permisos y personal que necesita una pantalla de administración para todo ello. Mantenga cada proyecto en una versión con soporte, elija una LTS mientras continúen las versiones A.B y anote la fecha de fin de soporte junto a la versión. Ejecute `manage.py check --deploy` con los ajustes de producción dentro del flujo de integración, mantenga la clave secreta y la contraseña de la base de datos fuera del repositorio y lea el SQL que escribe el ORM para cada página importante. Si el cliente va a gestionar el sitio principalmente editando páginas, un sistema de gestión de contenidos o unas páginas estáticas encajan mejor que un framework.

Cómo se aplica en la práctica

Entre 2015 y 2022, el trabajo como desarrolladora web incluyó aplicaciones web Django en torno a una plataforma IPTV: las partes en las que la gente realmente hacía clic dentro de un servicio que funcionaba las veinticuatro horas. El equipo se ocupaba de nuevas funciones, correcciones de errores y trabajo continuo de rendimiento y estabilidad. En un servicio así, explica la página, la estabilidad es el límite que deben respetar las funciones; las aplicaciones siguieron ganando capacidades y mantuvieron su fiabilidad para los equipos internos y los clientes.

Antes, entre 2003 y 2014, en el trabajo como desarrolladora web full stack, se crearon cien aplicaciones web para clientes que buscaban cosas muy distintas. Cuando necesitaban algo a medida, las herramientas eran HTML/HTML5, CSS/SCSS y Django; cuando necesitaban algo más estándar que también pudieran gestionar por sí mismos, WordPress o Joomla eran la respuesta más rápida y

sensata. La página sitúa la habilidad en distinguir un caso del otro, incluido convencer al cliente de que no necesitaba un desarrollo a medida o de que sí lo necesitaba. Es la misma distinción que traza nuestra recomendación.

La página de tecnologías conserva ambas vertientes. Su capa web da prioridad a lo estático, con Django y PHP cuando hay que servir una aplicación. Su lista de lenguajes trata SQL como un lenguaje de pleno derecho, en lugar de algo que genera un ORM, porque ahí reside la mayor parte del trabajo de rendimiento para el que nos solicitan. En una aplicación Django, eso significa que el ORM escribe las consultas y alguien sigue leyéndolas.

Dónde surgen los problemas

- **Un bucle ejecuta una consulta por fila.** En el modo de obtención predeterminado, leer una clave foránea que el QuerySet no cargó ejecuta una consulta para ese objeto. Una lista de cien libros con sus autores cuesta, por tanto, 101 consultas. Las plantillas dificultan verlo porque llaman a métodos como `all` sin paréntesis. Utilice `select_related()` para claves foráneas y campos uno a uno, y `prefetch_related()` para campos muchos a muchos y claves foráneas inversas. En Django 6.1 y posteriores, el modo `FETCH_PEERS` agrupa esas búsquedas en una consulta adicional y `FETCH_RAISE` convierte una consulta no prevista en una excepción en el código que debe mantenerse rápido.
- **El modo de depuración llega a producción.** La lista de comprobación del despliegue indica que nunca se debe activar `DEBUG` en producción, porque sus páginas de error filtran fragmentos de código fuente, variables locales, ajustes y las bibliotecas utilizadas. Con `DEBUG = False`, Django no atiende ninguna solicitud hasta que `ALLOWED_HOSTS` tenga un valor adecuado; un comodín en ese ajuste obliga a validar la cabecera Host por cuenta propia. Ejecute `manage.py check --deploy` con el archivo de ajustes de producción: automatiza parte de esa lista.
- **La clave secreta está en el repositorio.** `SECRET_KEY` protege las sesiones con la mayoría de sus mecanismos de almacenamiento, todos los tokens de restablecimiento de contraseñas y todas las firmas criptográficas que no indiquen otra clave. La referencia de ajustes advierte de que una clave conocida puede permitir la escalada de privilegios y la ejecución remota de código. Cárguela desde el entorno o un archivo, manténgala fuera del control de versiones, rótela mediante `SECRET_KEY_FALLBACKS` y retire pronto las claves antiguas de esa lista.

- **Se desactiva una protección predeterminada para una vista.** Los QuerySets están parametrizados, pero las consultas directas, `extra()` y `RawSQL` dejan el escape al desarrollador. Las plantillas escapan HTML, pero `mark_safe`, el filtro `safe` y la desactivación del escape automático también devuelven esa tarea al desarrollador. `csrf_exempt` elimina por completo la comprobación de una vista. Cada caso merece un motivo explícito en la revisión, y el HTML que almacenan los usuarios debería sanearse tanto a la entrada como a la salida.
- **Una migración se detiene a medias.** En PostgreSQL y SQLite, todas las operaciones de una migración se ejecutan dentro de una sola transacción de forma predeterminada. MySQL, en cambio, no engloba los cambios de esquema en transacciones, por lo que una migración que falla a mitad deja cambios que hay que deshacer a mano. Lea qué cree `makemigrations` que ha cambiado, compruebe el SQL con `sqlmigrate` antes de que llegue a producción y registre cada migración en el mismo commit que el cambio de su modelo.

Lecturas adicionales

- Documentación de Django — La documentación de la versión estable actual, desde el tutorial hasta la referencia.
- Django: lista de comprobación del despliegue — Los ajustes que hay que cambiar antes de entrar en servicio, una parte de los cuales automatiza `check --deploy`.
- Django: optimización del acceso a la base de datos — La evaluación de QuerySets, la obtención de objetos relacionados y las prácticas que mantienen constante el número de consultas.
- Django: migraciones — El flujo de migraciones, `sqlmigrate` y cómo maneja cada motor de base de datos los cambios de esquema.
- Seguridad en Django — Las amenazas frente a las que el framework protege de forma predeterminada y dónde termina esa protección.
- Descargar Django — Las series de versiones con soporte y sus fechas de soporte principal y extendido.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es Django?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/django/