Introducción a Django

En este primer apartado responderemos la pregunta ¿Qué es Django? y daremos una visión general de lo que hace que este framework sea tan especial. Vamos a delinear las características principales. También mostraremos algunos de los componentes principales de una aplicación de Django.

¿Qué es Django?

Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto, tiene una comunidad próspera y activa, una gran documentación y muchas opciones de soporte gratuito y de pago.

Django te ayuda a escribir software que es:

Completo

Django sigue la filosofía "Baterías incluidas" y provee casi todo lo que los desarrolladores quisieran que tenga "de fábrica". Porque todo lo que necesitas es parte de un único "producto", todo funciona a la perfección, sigue principios de diseño consistentes y tiene una amplia y actualizada documentación (https://docs.djangoproject.com/en/3.2/).

Versátil

Django puede ser (y ha sido) usado para construir casi cualquier tipo de sitio web — desde sistemas manejadores de contenidos y wikis, hasta redes sociales y sitios de noticias. Puede funcionar con cualquier framework en el lado del cliente, y puede devolver contenido en casi cualquier formato (incluyendo HTML, RSS feeds, JSON, XML, etc.).

Internamente, mientras ofrece opciones para casi cualquier funcionalidad que desees, también puede ser extendido para usar otros componentes si es necesario.

Seguro

Django ayuda a los desarrolladores evitar varios errores comunes de seguridad al proveer un framework que ha sido diseñado para "hacer lo correcto" para proteger el sitio web automáticamente. Por ejemplo, Django, proporciona una manera segura de administrar cuentas de usuario y contraseñas, evitando así errores comunes como colocar informaciones de sesión en cookies donde es vulnerable (en lugar de eso las cookies solo contienen una clave y los datos se almacenan en la base de datos) o se almacenan directamente las contraseñas en un hash de contraseñas.
Django permite protección contra algunas vulnerabilidades de forma predeterminada, incluida la inyección SQL, scripts entre sitios, falsificación de solicitudes entre sitios y clickjacking.

Escalable

Django usa un componente basado en la arquitectura “shared-nothing” (cada parte de la arquitectura es independiente de las otras, y por lo tanto puede ser reemplazado o cambiado si es necesario). Teniendo en cuenta una clara separación entre las diferentes partes significa que puede escalar para aumentar el tráfico al agregar hardware en cualquier nivel: servidores de cache, servidores de bases de datos o

servidores de aplicación. Algunos de los sitios más concurridos han escalado a Django para satisfacer sus demandas (por ejemplo, Instagram y Disqus, por nombrar solo dos).

Mantenible

El código de Django está escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable. En particular, utiliza el principio No te repitas "Don't Repeat Yourself" (DRY) para que no exista una duplicación innecesaria, reduciendo la cantidad de código. Django también promueve la agrupación de la funcionalidad relacionada en "aplicaciones" reutilizables y en un nivel más bajo, agrupa código relacionado en módulos (siguiendo el patrón MVT; similar al patrón MVC, Model View Controller).

Portable

Django está escrito en Python, el cual se ejecuta en muchas plataformas. Lo que significa que no está sujeto a ninguna plataforma en particular, y puede ejecutar sus aplicaciones en muchas distribuciones de Linux, Windows y Mac OS X. Además, Django cuenta con el respaldo de muchos proveedores de alojamiento web, y que a menudo proporcionan una infraestructura específica y documentación para el alojamiento de sitios de Django.

Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

¿Qué tan popular es Django?

No hay una medida de popularidad definitiva. Una pregunta mejor sería si Django es lo "suficientemente popular" para evitar los problemas de las plataformas menos populares. ¿Continúa evolucionando? ¿Puedes conseguir la ayuda que necesitas? ¿Hay alguna posibilidad de que consigas un trabajo si aprendes Django?

De acuerdo con el número de sitios que usan Django, el número de gente que contribuye al código base, y el número de gente que proporciona soporte tanto libre como pagado, podemos entonces decir que sí, Django es un framework “popular”.

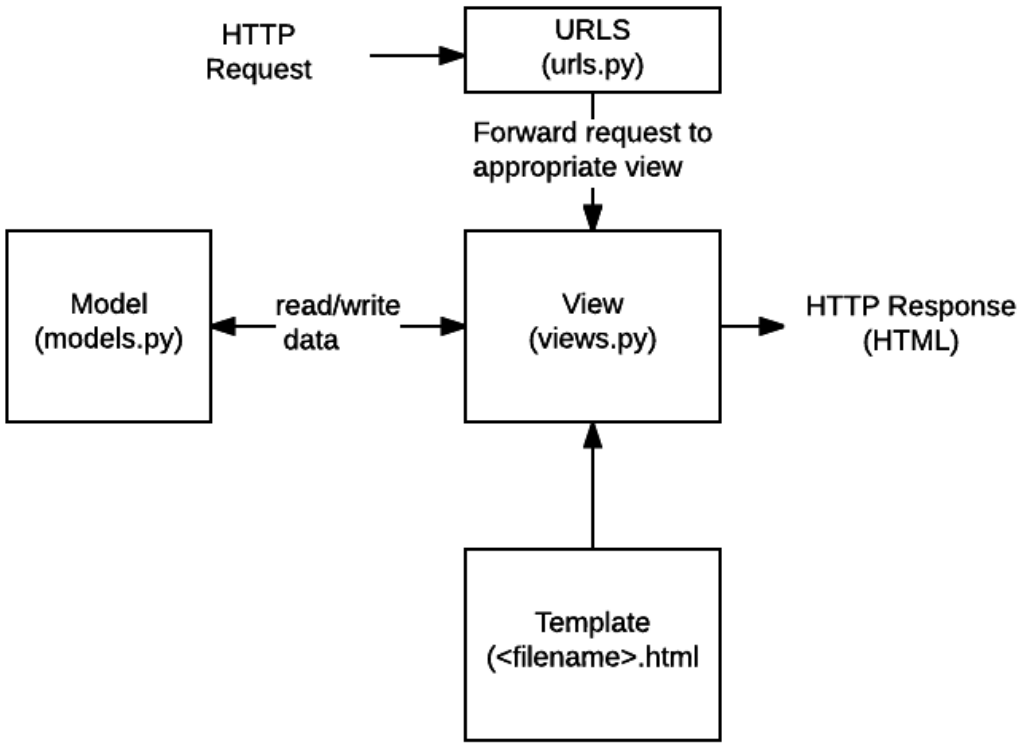
Los sitios que actualmente usan Django son: Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest, etc.

Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

¿Qué aspecto tiene el código de Django?

En un sitio web tradicional basado en datos, una aplicación web espera peticiones HTTP del explorador web (o de otro cliente). Cuando se recibe una petición la aplicación elabora lo que se necesita basándose en la URL y posiblemente en la información incluida en los datos POST o GET. Dependiendo de qué se necesita quizás pueda entonces leer o escribir información desde una base de datos o realizar otras tareas requeridas para satisfacer la petición. La aplicación devolverá a continuación una respuesta al explorador web, con frecuencia creando dinámicamente una página HTML para que el explorador la presente insertando los datos recuperados en marcadores de posición dentro de una plantilla HTML.

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados:



URLs: aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso. Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos.

Vista (View): una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de *modelos*, y delegan el formateo de la respuesta a las *plantillas* (*"templates"*).

Modelos (Models): los Modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.

Plantillas (Templates): una plantilla (template) es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una *vista* puede crear dinámicamente una página usando una plantilla, rellenándola con datos de un *modelo*. Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero; aunque no tiene porqué ser HTML.

Nota: *Django se refiere a este tipo de organización como arquitectura Modelo Vista Plantilla "Model View Template (MVT)". Tiene muchas similaridades con la arquitectura más familiar Model View Controller.*

Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

¿Qué se puede hacer con Django?

Las principales características que podrás usar con Django en casi todas las aplicaciones web son: mapeo de URLs, vistas, modelos y plantillas. Sin embargo, Django ofrece también:

Formularios: los formularios HTML se usan para recolectar datos de los usuarios para su procesamiento en el servidor. Django simplifica la creación, validación y procesamiento de los formularios.

Autenticación y permisos de los usuarios: Django incluye un sistema robusto de autenticación y permisos que ha sido construido con la seguridad en mente.

Cacheo: la creación dinámica de contenido es mucho más intensiva computacionalmente (y lenta) que un servicio de contenido estático. Django proporciona un cacheo flexible de forma que puedes almacenar todo o parte de una página renderizada para que no sea re-renderizada nada más que cuando sea necesario.

Sitio de Administración: el sitio de administración de Django está incluido por defecto cuando creas una app usando el esqueleto básico. Esto hace que sea trivialmente fácil proporcionar una página de administración para que los administradores puedan crear, editar y visualizar cualquiera de los modelos de datos de su sitio.

Serialización de datos: Django hace fácil el serializar y servir tus datos como XML o JSON. Esto puede ser útil cuando se está creando un servicio web (un sitio web que sólo sirve datos para ser consumidos por otras aplicaciones o sitios, y que no presenta en pantalla nada por sí mismo), o cuando se crea un sitio web en el que el código del lado cliente maneja toda la renderización de los datos.

Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

Próximos pasos

En el Tutorial “Desarrollando una aplicación web con Python y Django” podrán encontrar una guía que los acompañará en sus primeros pasos con Django.