Tp Django

Alumno: Fabrizio Giordanelli

Colegio: Instituto Luis A. Huergo

Entrega:10-07-2025

Materia: Lab de Algoritmos

¿Qué es Django y por qué lo usaríamos?

Django es un framework web de alto nivel escrito en Python, diseñado para fomentar el desarrollo rápido, limpio y pragmático de aplicaciones web. Al ser de "alto nivel", abstrae muchas de las complejidades del desarrollo web, permitiendo que los desarrolladores se centren en la lógica de la aplicación. Su filosofía se basa en el principio "Don't Repeat Yourself" (DRY), que busca reducir la repetición de código mediante una arquitectura que promueve la reutilización de componentes. Además, sigue la filosofía de "baterías incluidas", lo que significa que viene con una vasta biblioteca de herramientas integradas listas para usar, como un sistema de autenticación, un panel de administración, un ORM y un motor de plantillas. Las razones para usarlo son variadas y de gran peso: su enfoque en la eficiencia permite pasar del concepto a la aplicación finalizada en menos tiempo; está diseñado para ser seguro por defecto, proporcionando protección integrada contra vulnerabilidades comunes como inyección SQL o XSS; su arquitectura basada en componentes le otorga una gran escalabilidad para crecer junto con el proyecto; y, al ser de código abierto, cuenta con una comunidad muy activa, excelente documentación y un enorme ecosistema de paquetes que extienden sus funcionalidades.

¿Qué es el patrón MTV (Model-Template-View) en Django?

Comparación con MVC

El patrón Modelo-Plantilla-Vista (MVT) es la arquitectura de software que Django utiliza para organizar la lógica de una aplicación web, con el objetivo principal de lograr una

clara separación de intereses. Este patrón se compone de tres elementos interconectados. Primero, el Modelo (Model), que define la estructura de los datos, siendo la fuente única y definitiva de información que se corresponde con las tablas de la base de datos. Segundo, la Plantilla (Template), que constituye la capa de presentación y define, generalmente en un archivo HTML, cómo se mostrarán los datos al usuario utilizando el Lenguaje de Plantillas de Django. Finalmente, la Vista (View) actúa como la capa de lógica de negocio, recibiendo las peticiones del usuario, interactuando con el Modelo para acceder a los datos y pasando esa información a la Plantilla para su renderización. Al comparar MVT con el patrón tradicional Modelo-Vista-Controlador (MVC), ambos buscan separar la lógica de la presentación. La diferencia fundamental radica en el rol del "Controlador". Mientras que en MVC el Controlador es una pieza explícita que recibe la entrada y coordina el Modelo y la Vista, en MVT esta responsabilidad es gestionada implícitamente por el propio framework de Django, que a través de su sistema de enrutamiento de URLs dirige cada petición a la Vista adecuada, siendo esta última la que orquesta la interacción entre el Modelo y la Plantilla.

¿Qué entendemos por app en Django?

En el ecosistema de Django, una app (aplicación) es un módulo de software autocontenido y reutilizable que cumple una función específica dentro de un proyecto web global. Un proyecto Django puede ser visto como una colección de configuraciones y diversas apps que, en conjunto, conforman un sitio web completo. La filosofía detrás de esto es dividir un proyecto grande en piezas más pequeñas y manejables para mejorar la

organización y el mantenimiento. Por ejemplo, un sitio de comercio electrónico podría tener apps separadas para la gestión de productos, el carrito de compras y la autenticación de usuarios. Cada una de estas apps encapsula su propia lógica, conteniendo archivos como `models.py` para sus estructuras de datos, `views.py` para manejar la lógica de negocio, `urls.py` para sus rutas específicas y un directorio `templates/` para su presentación. Esta modularidad no solo organiza el código, sino que fomenta la reutilización, ya que una app bien diseñada podría ser integrada en otros proyectos con mínimas modificaciones.

¿Qué es el flujo request-response en Django?

El ciclo de petición-respuesta (request-response) es el proceso central mediante el cual Django gestiona toda interacción con el usuario. Este flujo comienza cuando un usuario realiza una acción en su navegador, lo que genera una petición HTTP que es enviada al servidor. Al recibirla, Django la encapsula en un objeto HttpRequest, que contiene toda la información relevante de la solicitud. A continuación, el enrutador de URLs del framework examina la ruta solicitada y la compara con los patrones definidos en los archivos urls.py para determinar qué vista es la encargada de manejarla. Una vez encontrada la coincidencia, Django invoca a la vista correspondiente, pasándole el objeto HttpRequest como argumento. La vista entonces ejecuta la lógica de negocio necesaria, que puede incluir interacciones con la base de datos a través de los modelos. Finalmente, la vista debe devolver un objeto HttpResponse, que contiene la respuesta que se enviará

de vuelta al navegador del usuario, ya sea una página HTML renderizada, una redirección, un archivo JSON o cualquier otro tipo de respuesta HTTP.

¿Qué es el concepto de ORM (Object-Relational Mapping)?

El Mapeo Objeto-Relacional (ORM) es una poderosa técnica de programación que actúa como un traductor o puente entre un lenguaje orientado a objetos, como Python, y una base de datos relacional, como PostgreSQL. El ORM de Django permite a los desarrolladores interactuar con la base de datos de una manera mucho más intuitiva y pitónica, utilizando clases y objetos en lugar de escribir consultas SQL. En la práctica, una tabla de la base de datos se representa como una clase de Python (un Modelo), cada fila de esa tabla es un objeto o instancia de esa clase, y las columnas son los atributos del objeto. Por ejemplo, en lugar de escribir INSERT INTO..., un programador simplemente crea una instancia de la clase y llama al método .save(). Las ventajas de este enfoque son significativas, ya que proporciona una mayor abstracción y productividad al desarrollador, ofrece portabilidad del código entre diferentes sistemas de bases de datos con solo cambiar la configuración, y mejora la seguridad al parametrizar las consultas de forma segura, ayudando a prevenir ataques de inyección SQL.

¿Qué son los templates en Django?

En Django, un template (plantilla) es un archivo de texto, comúnmente HTML, que se utiliza para separar la lógica de negocio de la capa de presentación. Su principal objetivo es definir la estructura y el diseño visual de la información que se mostrará al usuario, permitiendo que los programadores se enfoquen en la lógica en Python (vistas) y los diseñadores en el marcado y estilo (HTML/CSS). Las plantillas son dinámicas gracias al Lenguaje de Plantillas de Django (DTL), que enriquece el HTML con una sintaxis especial. Este lenguaje permite insertar contenido dinámico a través de variables (envueltas en `{{ }}`), que son reemplazadas por los datos pasados desde la vista. También incluye etiquetas (envueltas en `{% %}`), que proporcionan lógica de programación como bucles para iterar sobre listas de datos o condicionales para mostrar contenido de forma selectiva. Además, una de sus características más potentes es la herencia de plantillas, que permite crear una plantilla base con la estructura común de un sitio para luego extenderla en otras plantillas, redefiniendo solo los bloques necesarios. Esto reduce drásticamente la duplicación de código y facilita el mantenimiento del frontend.

¿Cómo se lo instala?

Para instalar django se necesita tener previamente instalado el python con el pip. Luego de esto debes entrar a la consola de comandos, cmd o mismo la consola en visual studio y escribir la siguiente oración “pip install django”. Luego iniciara la instalación y

una vez terminada ya podremos usar django con python para los proyectos que querramos

Referencias

APIDog. (s.f.). *Django GET Request.* https://apidog.com/es/blog/django-get-request-3/

Amazon Web Services. (s.f.). *¿Qué es Django?* https://aws.amazon.com/es/what-is/django/

CertiDevs. (s.f.). *Tutorial Django MVC: Vistas funciones.* https://certidevs.com/tutorial-django-mvc-vistas-funciones

Chirilov, A. (s.f.). *Apps in Django: Concept & Free Samples.* https://www.codementor.io/@chirilovadrian360/apps-in-django-concept-free-samples-294vudyim5

Código Facilito. (s.f.). *MVC: Model View Controller explicado.* https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

Django Software Foundation. (s.f.). *Request and response objects.* https://docs.djangoproject.com/en/5.2/ref/request-response/

Ekezie, J. (s.f.). *MVC vs MVT: What You Need To Know About Them.* https://medium.com/@ekezieju/mvc-vs-mvt-what-you-need-to-know-about-them-5511f561d593

EspiFreelancer. (s.f.). *MTV Django.* https://espifreelancer.com/mtv-django.html

IBM. (s.f.). *Django.* https://www.ibm.com/es-es/topics/django

MDN Web Docs. (s.f.). *Introducción a Django.* https://developer.mozilla.org/es/docs/Learn_web_development/Extensions/Server-side/Django/Introduction

W3Schools. (s.f.). *Django Create App.* https://www.w3schools.com/django/django_create_app.php

Wikipedia. (s.f.). *Modelo–vista–controlador.* https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador