Trabajo Práctico N°3

Vigna Luca

Ins. Ind. Luis A. Huergo

Curso: 4°AO

Laboratorio de Algoritmos y Estructuras de Datos

Profesor: Ignacio Miguel García

10 de julio de 2025

1. ¿Qué es Django y por qué lo usaríamos?

Django es un framework web de alto nivel, escrito en Python, que permite desarrollar aplicaciones y sitios web de forma rápida y segura. Su principal ventaja es que viene con "baterías incluidas", lo que significa que ofrece muchas herramientas y funcionalidades ya integradas para tareas comunes, como la autenticación de usuarios, la gestión de bases de datos y un panel de administración.

Lo usaríamos por su velocidad de desarrollo, ya que permite pasar del concepto a la producción de manera eficiente. También destaca por su seguridad incorporada, que ayuda a proteger contra vulnerabilidades comunes como CSRF, XSS y SQL injection. Además, es escalable, lo que lo hace adecuado para proyectos de gran tamaño y tráfico , y cuenta con una comunidad activa y una amplia documentación.

2. ¿Qué es el patrón MTV (Model-Template-View) en Django? (Simplificado de MVC). Compará MTV con MVC.

El patrón MTV (Model-Template-View) es la arquitectura que utiliza Django, una adaptación del patrón MVC (Model-View-Controller). Cada componente en MTV tiene un rol específico:

Model: Representa la estructura de los datos y la lógica del negocio. En Django, son clases Python que se mapean a tablas en la base de datos.

Template: Se encarga de la presentación de la información al usuario. Son archivos, usualmente HTML, que contienen marcadores de posición para datos dinámicos.

View: Recibe las peticiones del usuario, interactúa con el Model para obtener o modificar datos, y luego pasa estos datos al Template para generar la respuesta final.

Por otro lado, el patrón MVC, que surgió en la década de 1970, también divide una aplicación en tres componentes para mejorar el mantenimiento:

Model: Similar al MTV, gestiona los datos y la lógica del negocio.

View: Es la interfaz de usuario que muestra los datos y captura las interacciones del usuario.

Controller: Recibe las entradas del usuario (a través de la View), procesa la lógica, actualiza el Model y decide qué View mostrar.

La principal diferencia radica en cómo se distribuyen las responsabilidades. En Django, la "View" del patrón MVC es manejada por el "Template" en MTV, que se enfoca puramente en la presentación. Por otro lado, la "View" en Django asume el rol del "Controller" de MVC, manejando las peticiones, la lógica y la interacción con el Model, para luego seleccionar el Template adecuado. El "Controller" en el MVC tradicional está implícito en la forma en que Django enruta las URLs y maneja las peticiones.

3. ¿Qué entendemos por app en Django?

En Django, una "app" es un módulo o paquete de Python autocontenido que proporciona una funcionalidad específica y reutilizable dentro de un proyecto. Puede incluir sus propios modelos, vistas, plantillas, URLs, archivos estáticos y pruebas. La idea es que cada app se centre en una tarea bien definida, como un sistema de blog o una aplicación de encuestas. Esto promueve la modularidad y la organización del código, haciendo que los proyectos sean más fáciles de mantener y escalar. Una app bien diseñada puede incluso ser reutilizada en distintos proyectos de Django sin necesidad de modificaciones significativas.

4. ¿Qué es el flujo request-response en Django?

El flujo request-response describe la secuencia de pasos que Django sigue desde que recibe una petición HTTP de un cliente (como un navegador web) hasta que le devuelve una respuesta apropiada. Este ciclo involucra varias etapas clave:

Petición HTTP: El navegador envía una petición HTTP al servidor web (ej., Apache, Nginx), que luego la delega a Django a través de una interfaz como WSGI.

Objeto HttpRequest: Django crea un objeto HttpRequest que contiene toda la información de la petición, como el método, la URL y las cabeceras.

Middlewares (fase de request): La petición pasa por una serie de middlewares configurados en Django. Estos pueden modificar la petición o incluso generar una respuesta directamente sin que la petición llegue a la vista.

URL Dispatcher: Django usa la URL de la petición para encontrar la vista (View) que le corresponde, basándose en las configuraciones de URL del proyecto.

Middlewares (fase process_view): Antes de ejecutar la vista, pueden aplicarse middlewares adicionales que realicen validaciones o comprobaciones.

Ejecución de la View: La vista correspondiente se ejecuta, recibe el objeto HttpRequest, interactúa con los modelos (a través del ORM) para acceder o procesar datos, y prepara el contexto para la plantilla.

Renderizado del Template: La vista pasa los datos a un template, que el motor de plantillas de Django utiliza para generar la respuesta final, generalmente HTML.

Middlewares (fase de response): La respuesta generada pasa nuevamente por los middlewares, esta vez en orden inverso, permitiéndoles modificar la respuesta antes de ser enviada.

Envío de respuesta: Finalmente, la respuesta HTTP es enviada de vuelta al servidor WSGI, que la transmite al servidor web, y este la envía al navegador del cliente.

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

ORM(Object-Relational Mapping) es una técnica que permite a los desarrolladores interactuar con bases de datos relacionales utilizando objetos de un lenguaje de programación, en lugar de escribir código SQL directamente. Un ORM actúa como una capa de abstracción que traduce las operaciones realizadas sobre objetos (como la creación o modificación de atributos) a consultas SQL optimizadas para la base de datos subyacente.

En Django, el ORM es una parte fundamental. Cada "modelo" que defines en Django es una clase Python que hereda de models.Model, y cada atributo de esta clase representa una columna en una tabla de la base de datos. El ORM de Django permite realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar) sobre estos modelos utilizando métodos intuitivos como .create(), .filter(), .get(), .update() y .delete().

6. ¿Qué son los templates en Django?

Los templates en Django son archivos de texto, comúnmente HTML, que utilizan el lenguaje de plantillas de Django (Django Template Language) para generar contenido dinámico. Su objetivo principal es separar la lógica de negocio de la presentación, lo que significa que las vistas (Views) de Django se encargan de preparar los datos, mientras que los templates se concentran en cómo se muestran esos datos al usuario.

El lenguaje de plantillas de Django ofrece una sintaxis sencilla para incluir contenido dinámico y lógica básica de presentación:

Variables: Se utilizan con doble llave {{ variable }} para insertar valores del contexto enviado desde la vista.

Tags: Son estructuras de control que permiten realizar operaciones como bucles ({% for %}), condicionales ({% if %}) o incluir otros archivos ({% include %}).

Filters: Se aplican a las variables para transformar o formatear su valor, por ejemplo, {{ value|date:"Y-m-d" }} para formatear una fecha.

Comments: Son secciones de código que no se renderizan en la salida final, útiles para explicaciones internas {# ... #}.

Un aspecto importante de los templates en Django es la herencia de plantillas, que permite definir una estructura básica en un archivo (base.html) con bloques que pueden ser sobrescritos o extendidos por otros templates. Esto ayuda a evitar la duplicación de código y mantener un diseño coherente en todo el sitio. Los templates se organizan típicamente dentro de una carpeta templates/ en cada aplicación o en una carpeta central a nivel de proyecto.

Referencias

https://en.wikipedia.org/wiki/Model–view–controller

https://en.wikipedia.org/wiki/Django_(web_framework)

https://www.almabetter.com/bytes/tutorials/django/django-orm

https://forum.djangoproject.com/t/mvc-or-mvt-architecture/3496

https://esketchers.com/9-reasons-why-use-django-framework/