Módulo 6, Aprendizaje Esperado 6: Ejercicio individual

Nombre: Cristian Aranda Bórquez

1. Investigación Guiada:

    - ¿Qué es Django y por qué se usa?

Django es un **framework** web de código abierto y escrito en Python. Se utiliza para el desarrollo rápido de páginas y aplicaciones web complejas, destacando por su filosofía "battery *included*"(incluye muchas funcionalidades por defecto) y su seguridad integrada.

    - Diferencias entre el entorno de desarrollo y producción en Django.

El entorno de desarrollo es el espacio de trabajo local donde se escribe, prueba y depura el código. Éste se configura para mostrar errores detallados, usar un servidor liviano (como el de desarrollo de Django) y una base de datos local.

Por otro lado, entorno de producción es el ambiente real y público donde los usuarios acceden a la aplicación. A diferencia del de desarrollo, está optimizado para la seguridad, el rendimiento y la estabilidad. Esta vendría siendo ya la página subida con la que interactúan los usuarios.

    - Comparación entre Django y Python: ¿cómo se relacionan?

Python es un lenguaje de programación, mientras que **Django** es un framework que utiliza el lenguaje de **Python** para facilitar específicamente el desarrollo de aplicaciones y sitios web. Se relacionan en que, como se mencionó anteriormente, **Django** esta construido con el lenguaje de **Python** y utiliza su sintaxis, bibliotecas y ecosistema para funcionar.

    - ¿Por qué Django facilita el desarrollo de aplicaciones web?

Django facilita el desarrollo debido a que proporciona diferentes componentes preconstruidos y convenciones, como un sistema de ORM (Mapeo Objeto-Relacional), un panel de administración automático, un sistema de plantillas y manejo de URLs. Estos componentes simplifican el desarrollo y permiten que los desarrolladores puedan centrarse en la **lógica de negocio** en lugar de en la implementación de funcionalidades comunes.

    - ¿Qué bases de datos soporta Django?

Django, de forma nativa, soporta **PostgreSQL**, **MySQL**, **SQLite** y **Oracle**. Dentro de estos, la que se utiliza por defecto en el framework es **SQLite** y se recomienda para aplicaciones pequeñas o para el desarrollo (Y en producción se utiliza una base de datos más robusta). A través de terceros Django soporta diversas bases de datos como **MongoDB** a través de ***Djongo*.**

- ¿Qué es un entorno virtual en Python y por qué es útil?

Un entorno virtual es un directorio aislado que contiene una instalación separada de Python y sus propias bibliotecas y dependencias. Esto es útil y necesario debido a que aísla diferentes proyectos con sus propias dependencias, evitando que se generen conflictos con otros proyectos debido a diferencias de versiones y/o bibliotecas.

- ¿Cómo se crea y se usa un entorno virtual en Python?

Para crear un entorno virtual, se ejecuta un comando dentro de la terminal:

```
python -m venv nombre_del_entorno
```

Y para poder utilizarlo es necesario activar el entorno virtual con el comando:

```
nombre_del_entorno\Scripts\activate
```

Cabe destacar, que se debe mantener la estructura con los backslash '\' para que pueda funcionar.

2. **Análisis de la Estructura de Django:**
    - Explica en un párrafo la estructura general de un proyecto Django y su enfoque MVC.

Un proyecto Django se compone de una configuración general y una o más aplicaciones ("apps") que son módulos independientes y reutilizables de funcionalidad. Aunque Django usa la convención **MTV** (Modelo-Template-Vista), se alinea con el patrón **MVC** (Modelo-Vista-Controlador). El **Modelo** maneja la interacción con la base de datos, la **Vista** maneja la lógica de negocio y decide qué información enviar, y el **Template** (parte del "**Controlador**" de la arquitectura tradicional) se encarga de la presentación final de los datos al usuario. El "**Controlador**" implícito de Django es el framework mismo, que dirige las peticiones a la **Vista** apropiada y esta renderiza a través de las **URLs** a las **Templates**.

    - Describe cómo Django maneja el enrutamiento de URLs.

Django maneja el enrutamiento de **URLs** a través de un archivo principal de configuración de **URLs** (urls.py) en el proyecto, que mapea patrones de **URL** a funciones de vista específicas. Cuando llega una solicitud, Django busca una coincidencia para la **URL** en ese mapeo. Es común que el archivo principal apunte a archivos **urls.py** secundarios dentro de cada aplicación, permitiendo un enrutamiento modular y una mejor organización.

3. **Reflexión sobre las Buenas Prácticas:**

    - Explica el principio **DRY (Don't Repeat Yourself)** y cómo se aplica en Django.

En palabras simples, como lo dicta su nombre, es el no repetirse a uno mismo, evitando redundancia en el código y optimizando la estructura de este.

Bajo esta filosofía se entiende que cada pieza de conocimiento o lógica de negocio debe tener una representación única, inequívoca y autorizada dentro de un sistema. En Django, esto se aplica a través del **ORM** (evitando escribir SQL repetidamente), la herencia de **templates** (reutilizando estructuras de HTML comunes) y la definición de clases base y **mixins** para las Vistas (reutilizando lógica común).

    - ¿Qué ventajas tiene la herencia de componentes en Django?

La herencia de componentes en Django, especialmente la herencia de **templates**, ofrece las ventajas de reutilización de código, coherencia visual y funcional, y fácil mantenimiento. Permite definir un diseño base una sola vez y luego que las páginas específicas solo definan las partes que cambian, asegurando que la estructura común del sitio (como el encabezado, el pie de página o la navegación) se mantenga uniforme en todo momento.

4. **Exploración de los Templates en Django:**

    - Define qué son los templates en Django y su función en el desarrollo web.

Los **templates** en Django son archivos que contienen **código HTML estático** mezclado con el **lenguaje de plantillas** de Django (variables y etiquetas especiales). Su función es separar la presentación de la lógica de negocio, permitiendo definir cómo se mostrarán los datos al usuario en un navegador. Esto, sumado a archivos “static” permite generar un front-end completo utilizando CSS, JavaScript e imágenes.

    - Explica cómo Django renderiza las vistas con templates.

La función de **Vista** en Django es recibir la solicitud del usuario, ejecutar la lógica necesaria, y recupera los datos requeridos. Luego, la vista llama a la función de renderizado (render), pasando el nombre del **template** a usar y un diccionario de contexto que contiene los datos que se deben inyectar. El sistema de templates de Django combina el **template** con los datos de contexto, genera el HTML final, y la vista lo devuelve como la respuesta HTTP al navegador del usuario.