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

Django es un *framework* de alto nivel en python, permitiendo desarrollar aplicaciones *web* de manera rápida, segura y con buena estructura. Su diseño está realizado con la intención de automatizar tareas comunes, tales como el manejo de formularios, la conexión con bases de datos, el sistema de datos, entre otras. Sigue el principio *"Don´t Repeat Yourself",* permitiendo que sea posible escribir menos código y mantenerlo de forma limpia, facilitando su lectura.

Existen diversas razones por las cuales se decidiría utilizar Django entre las cuales las principales serían la gran cantidad de de funcionalidades con las que viene (admin, autenticación, ORM), el hecho de que está muy bien documentado, su seguridad y su función de evitar errores comunes y su fácil mantenimiento y escalabilidad en los proyectos.

Fuente:

Django Software Foundation. (2024). *Django documentation*.
https://docs.djangoproject.com/en/stable/misc/design-philosophies/

2. ¿Qué es el patrón MTV en Django? Comparalo con MVC.

Al hablar de patrones nos referimos a la forma en la que es organizada el código de una aplicación. El patrón MTV (*Model-Template-View*) es una variación del MVC (*Model-View-Controller*) en la cual *Model* refiere a la estructura de datos y lógica de negocio (está relacionado de forma directa con el ORM antes mencionado de Django); *Template* define como es mostrada la información, es decir, el HTML, CSS, etc; por último *View* contiene la lógica con lo que se conecta todo, recibiendo las peticiones, consultando los modelos y enviando datos a los templates.

Este patrón se diferencia de su predecesor el MVC o *Model-View-Controller* principalmente por la forma en la que son nombrados los componentes ya que en Django es la *View* quién maneja la lógica en lugar del *Controller* en MVC, y el *Template* cumple la función de manejar la vista visual que realiza el *View* en MCV. Sin lugar a dudas realizan tareas muy similares y lo que varía es que están organizados de forma distinta según el *framework.*

Fuentes:

Django Software Foundation. (2024). *Django overview*.
https://docs.djangoproject.com/en/stable/misc/design-philosophies/
Mozilla Developer Network. (s.f.). *Django introduction*.
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

3. ¿Qué entendemos por app en Django?

En Django, por app entendemos una unidad modular de funcionalidad, esto significa que es una parte del proyecto encargada de cumplir una tarea específica. Cada proyecto de Django suele contener gran cantidad de apps y estas pueden ser reutilizadas en varios proyectos distintos.
Fuente:
Django Software Foundation. (2024). *Writing your first Django app, part 1*.
https://docs.djangoproject.com/en/stable/intro/tutorial01/

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

El flujo *request-response* en Django refiere al proceso en el que se hace una solicitud (*request*) que es respondida por Django por medio de una *response* con una página web. Este proceso puede ser dividido en cuatro partes, siendo la primera cuando un usuario entra a una URL que es interpretada por Django usando su sistema de rutas (*URL dispatcher*), la segunda es cuando se llama a la *view* o vista correspondiente, luego en el tercer paso la vista por medio de consultas de datos y templates genera una respuesta que generalmente es en html, y por último Django envía esta respuesta al navegador. En caso de que la URL no sea encontrada se muestra la vista del error correspondiente.
fuente:
Django Software Foundation. (2024). *How Django processes a request*.
https://docs.djangoproject.com/en/stable/topics/http/urls/

5. ¿Qué es el concepto de ORM?

El ORM (*Object-Relational Mapping*) es una herramienta clave de Django que hace posible la interacción con bases de datos relacionales (como SQLite o MySQL) utilizando código Python evitando tener que trabajar con SQL. De esta forma permite que los desarrolladores puedan crear, leer, actualizar y eliminar datos en la base directamente desde Python.
Esta herramienta hace que el código sea más seguro evitando errores comunes como inyecciones SQL, más fácil ya que al ser limpio y legible es intuitivo y mejora su portabilidad entre distintas bases de datos. Se basa en mapear modelos de Python con tablas de la base de datos haciendo que cada clase representa una tabla, cada atributo una columna y cada instancia un registro.
fuente:
Django Software Foundation. (2024). *Models*.
https://docs.djangoproject.com/en/stable/topics/db/models/

6. ¿Qué son los templates en Django?

Los templates en Django son archivos de texto generalmente realizados en HTML usados con el fin de definir la presentación visual de la información y al combinarlos con datos enviados de las *views* pueden hacer contenido dinámico. Siguen el principio de separación de responsabilidades para separar la lógica de la aplicación de la interfaz visual. Esto significa que se encarga de mostrar los datos de forma ordenada mientras que la *view* los prepara.
Estos utilizan el lenguaje de plantillas propio de Django que posee una estructura especial permitiendo insertar variables, estructuras de control, filtros y etiquetas. De esta forma cuando una vista genera una respuesta, se renderiza un template y se pasa un diccionario de datos. Por último el template reemplaza las variables con esos datos generando el HTML final que se ve.
Los templates pueden mostrar listas de datos, contenido adicional y heredar estructura base evitando repetir HTML. Por esto los principales beneficios de utilizarlos son separar el diseño visual de la lógica del backend (facilitando el trabajo en equipo), reutilizar código y facilitar la creación de interfaces dinámicas.
fuentes:
Django Software Foundation. (2024). *The Django template language: For Python programmers*. https://docs.djangoproject.com/en/stable/ref/templates/language/

Mozilla Developer Network. (s.f.). *Django templates*.
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Templates