Repaso de Django
© ADR Infor SL

eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ
eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:
Diseñar, implementar y gestionar chatbots inteligentes e interactivos en aplicaciones web utilizando Django y Python, integrando habilidades de procesamiento de lenguaje natural para mejorar la experiencia del usuario y optimizar la interacción entre usuarios y chatbots.

Resultados de Aprendizaje:

- Configurar y estructurar un proyecto Django básico para la implementación de un chatbot.
- Implementar y configurar vistas, URLs y plantillas en Django, aplicando el patrón MVC.

Repaso de Django

Para la realización del curso suponemos que tenemos los conocimientos básicos sobre Django pero no obstante vamos a realizar un repaso rápido sobre algunas de las características principales de este framework

Empezamos con: Modelos, Vistas y Templates o Plantillas, para qué sirven y cómo se utilizan.

Modelos

Los modelos son la definición de la estructura de datos y relaciones que posee Django con la base de datos.

Django se basa en un sistema denominado MTV (Model, template, View) pues la "M" de "MTV" hace referencia al "Modelo" por lo tanto una de las tres piezas clave en el funcionamiento de este framework.

Un modelo de Django es una descripción de los datos en la base de datos, representada como código de Python.

Ten en cuenta que Django fue diseñado para promover el **acoplamiento débil y la estricta separación entre las piezas de una aplicación**. Si seguimos esta filosofía, será fácil hacer cambios en un lugar particular de la aplicación sin afectar otras piezas.

En las aplicaciones web modernas, la lógica casi siempre implica interactuar con una base de datos. En el backend, un sitio web se conecta a un servidor de base de datos, recupera algunos datos necesarios y los muestra con un formato agradable en una página web. Del mismo modo, el sitio puede proporcionar funcionalidad que permita a los visitantes ingresar datos a la base de datos por su cuenta.

Django es apropiado para crear sitios web que manejen una base de datos, ya que incluye una manera fácil y muy poderosa, la realización de consultas a bases de datos utilizando Python.

Vistas

Las vistas en Django son el componente principal que **maneja la interacción entre el cliente y el servidor, procesando las solicitudes HTTP y generando las respuestas adecuadas** en función de la lógica de la aplicación.

Las vistas en Django se definen como funciones o clases dentro de los archivos de vistas de la aplicación Django(view.py) y se asocian a una URL específica mediante el enrutador de URL. Esto permite que Django dirija las solicitudes entrantes a la vista correspondiente para su procesamiento.
Algunos aspectos importantes sobre las vistas en Django incluyen:

Recepción de solicitudes HTTP

Las vistas pueden recibir diferentes tipos de solicitudes HTTP, como GET, POST, PUT, DELETE, etc. Dependiendo del tipo de solicitud y de los datos adjuntos, la vista puede realizar acciones específicas.

Generación de respuestas

Después de procesar la solicitud, la vista genera una respuesta HTTP que puede ser una página HTML renderizada, un archivo JSON, una redirección, una respuesta de error, entre otros tipos de respuestas.

Lógica de la aplicación

Las vistas contienen la lógica de la aplicación necesaria para procesar la solicitud y generar la respuesta. Esto puede incluir consultar o modificar la base de datos, interactuar con otros componentes del sistema, realizar cálculos, etc.

Renderización de plantillas

En muchos casos, las vistas renderizan plantillas HTML utilizando un sistema de plantillas como el de Django, que permite generar contenido dinámico mezclando datos y HTML.

Plantillas

Crear las vistas para nuestra web mediante código HTML dentro de funciones Python en archivo de las vistas, esto siempre es una mala idea. Por suerte, Django posee un potente motor de plantillas que nos permite separar el diseño de las páginas del código fuente.

Las plantillas en Django (la "T" de templates) son una parte fundamental para crear interfaces de usuario dinámicas en tus aplicaciones web, permiten la separación clara entre la lógica de presentación y la lógica de negocio de una aplicación web.

Problemas de codificar, mezclar e incrustar directamente el HTML en las vistas:

1

Cualquier cambio en el diseño de la página requerirá un cambio en el código de Python.

2
Escribir código Python y diseñar HTML son dos disciplinas diferentes, y la mayoría de los entornos de desarrollo web profesional dividen estas responsabilidades entre personas separadas (o incluso en departamentos separados).
3
Diseñadores y programadores HTML/CSS no deberían tener que editar código Python para conseguir hacer su trabajo; ellos deberían tratar con HTML.
4
Es más eficiente si los programadores pueden trabajar sobre el código Python y los diseñadores sobre las plantillas al mismo tiempo.

Por esta razón, resulta mucho más limpio y fácil de mantener el código al separar el diseño de la página del código Python en sí mismo. Para lograr esto, contamos con el sistema de plantillas de Django.

Los archivos de las plantillas tienen la extensión .html y se almacenan en el directorio templates de las aplicaciones Django.

Anotación: Plantillas tienen una sintaxis particular que se incluye dentro del html

Django utiliza llaves dobles ({{ }}) para insertar variables y llaves y porcentaje ({% %}) para estructuras de control como bucles y condicionales. Por ejemplo, {{ variable }} se utiliza para mostrar el valor de una variable en la plantilla, mientras que {% if condición %} ... {% endif %} se utiliza para crear bloques condicionales.

También permite la creación de plantillas base que pueden ser extendidas por otras plantillas más específicas. Esto se logra utilizando la etiqueta {% extends "nombre_de_plantilla_base.html" %} en las plantillas secundarias y luego rellenando los bloques de contenido específico utilizando la etiqueta {% block nombre_del_bloque %} ... {% endblock %}, aunque esto lo veremos en más detalle cuando trabajemos y tengamos ejemplos concretos.

Además las plantillas en Django tienen contexto, el contexto de una plantilla es un diccionario de datos que se pasa desde la vista al renderizar la plantilla. El contexto de una plantilla es un diccionario de datos que se pasa desde la vista al renderizar la plantilla. Este contexto contiene las variables que serán utilizadas en la plantilla y puede incluir datos dinámicos recuperados de la base de datos u otras fuentes.

Por otro lado los filtros en Django permiten modificar el formato o contenido de las variables de plantilla antes de ser renderizadas. Por ejemplo, el filtro {{ variable|filtro }} puede utilizarse para formatear cadenas, fechas y otros tipos de datos antes de mostrarlos en la plantilla.

Principales características y ventajas de Django en el desarrollo web

Django posee una filosofía de programación DRY (Dont repeat yourself) o "No te repitas", algo que es muy importante en el desarrollo con esta tecnología.

Debido a que Django está hecho en Python, es multiplataforma y puede ser instalado en cualquier tipo de máquina windows, Mac o Linux con el único requisito de una instalación Python.

Las características que seguro que ya conoces para querer desarrollar con él son:

Panel de Administrador
Django cuenta con un administrador que viene activo por defecto donde se pueden con un par de líneas de código mostrar los modelos de las bases de datos y poder crear, editar, ver y eliminar registros.
Gestión de Formularios
Crear formularios en Django es muy sencillo y se pueden crear de dos formas, un formulario definiendo uno a uno los campos o usar un modelo de la base de datos y Django crea el formulario por nosotros.
Manejo de Rutas
El manejo de rutas hace que crear urls complejas sea muy sencillo de implementar.
Métodos de Autenticación
Django provee un sistema de autenticación que permite que no nos preocupemos por crear un flujo de login y registro.
Gestión de Permisos
En Django tenemos el control de los permisos a tal punto de decir que usuario puede o no crear, editar, ver y eliminar registros de un modelo específico.
Manejo de Bases de Datos
Django cuenta con un ORM que nos permite preocuparnos en la lógica de nuestra aplicación dejando al ORM la responsabilidad de la comunicación con la base de datos, es compatible con los principales motores de bases de datos como PostgreSQL, MySQL, Oracle, SQLServer entre otros.

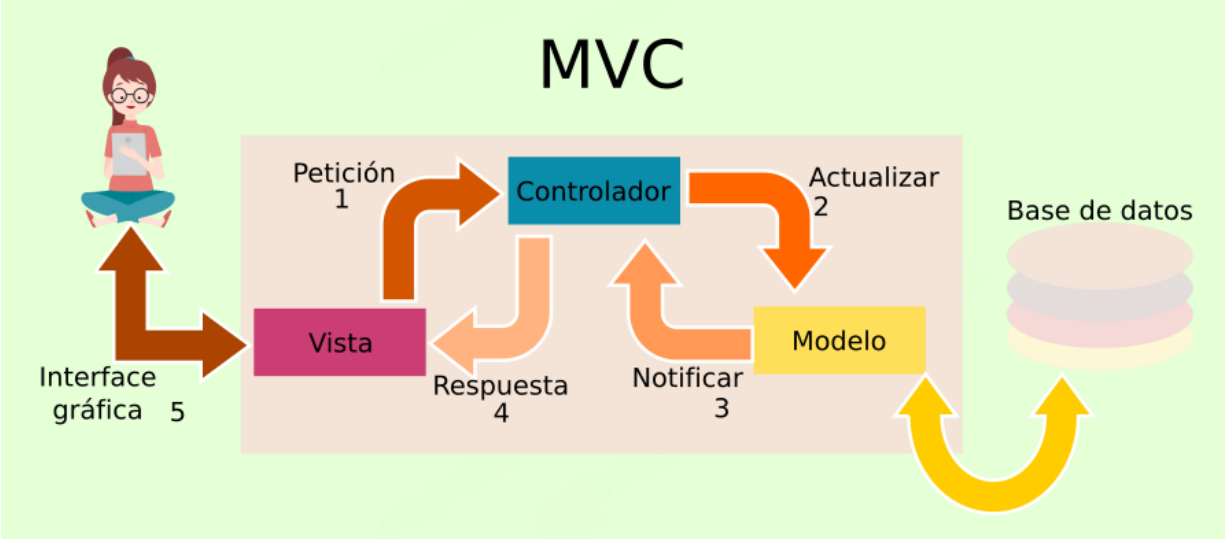
MVC (Modelo-Vista-Controlador) como arquitectura base

Django se basa en la arquitectura de software Model template View (MTV) que es una variación de la Model View Controller (MVC).

Modelo-Vista-Controlador o MVC es un patrón de arquitectura de software, básicamente lo que hace es separar los datos y la lógica de negocio de una aplicación de su representación y del módulo encargado de gestionar los eventos y las comunicaciones.

MVC propone la construcción de tres componentes distintos que son el **modelo, la vista y el controlador**, es decir, **por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario**. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

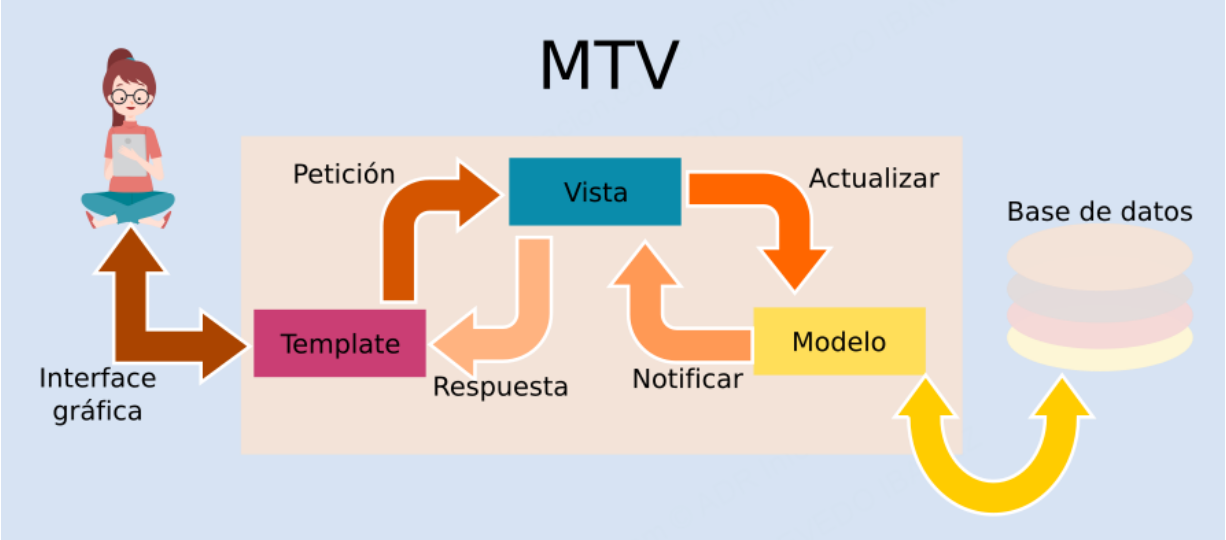
De manera genérica, los componentes de MVC se podrían definir como sigue:



Como hemos dicho, Django usa una arquitectura propia que se llama MTV, **Modelo, Vista, Template** que se basa en la MVC pero con sus propias peculiaridades.

Para entender MTV debemos fijarnos en la analogía con MVC.

- El modelo en Django sigue siendo modelo
- La vista en Django se llama Plantilla (Template)
- El controlador en Django se llama Vista



El modelo

Controla el comportamiento de los datos.

- Define los datos almacenados
- Se encuentra en forma de clases de Python (el es un objeto).
- Cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros.
- Posee métodos para la gestión de los datos.

La vista

Se presenta en forma de funciones o clases en Python,

- su propósito es determinar que datos serán visualizados serán visualizados.

El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo mas importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

Templates

Es básicamente una página HTML con algunas etiquetas extras propias de Django, no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

La plantilla(Template) recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del frontend, incluso tiene estructuras de datos como if, por si es necesaria es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python.

Administración automática de la base de datos

Django posee una administración automática de la base de datos que simplifica el proceso de manejo de la base de datos al automatizar tareas como la definición de la estructura de la base de datos, la generación de migraciones, y la creación de una interfaz de administración de datos. Esto permite a los desarrolladores centrarse en la lógica de la aplicación sin preocuparse por los detalles de la implementación de la base de datos.

Definición de la estructura de la base de datos

En Django, la estructura de la base de datos se define utilizando modelos de Python. Los desarrolladores pueden definir estos modelos de manera intuitiva utilizando clases de Python y los campos disponibles en el ORM (Mapeo Objeto-Relacional) de Django.

Generación de migraciones

Cuando se realizan cambios en los modelos de Django, como agregar un nuevo campo o modificar una relación, Django puede generar automáticamente "migraciones" que representan esos cambios en la estructura de la base de datos.

Aplicación de migraciones

Una vez que se generan las migraciones, los desarrolladores pueden aplicarlas a la base de datos utilizando comandos de administración de Django, como makemigrations y migrate.

Interfaz de administración de datos

Django también proporciona una interfaz de administración de datos que se genera automáticamente a partir de los modelos definidos en la aplicación. Esta interfaz permite a los administradores del sitio acceder y manipular los datos almacenados en la base de datos a través de una interfaz web intuitiva y fácil de usar.

Sistema de enrutamiento y gestión de URL

Es importante que sepamos que Django posee el control sobre la configuración de las rutas. Posee un mapeo de URLs que permite controlar el despliegue de las vistas apropiadas para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo, además permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

En Django, las URL se configuran mediante un mapeo de rutas en el archivo urls.py de cada aplicación. Este archivo define patrones de URL que corresponden a las vistas de la aplicación.

El mapeo de URLs en Django se realiza utilizando expresiones regulares o rutas simples. Cada patrón de URL se asigna a una vista específica que manejará la solicitud entrante. Por ejemplo, una ruta como '/productos/' podría asignarse a una vista que muestre una lista de productos, mientras que '/productos//' podría asignarse a una vista que muestre los detalles de un producto específico.

Django permite pasar variables dinámicas desde las URL a las vistas utilizando patrones de URL con parámetros. Estos parámetros pueden capturarse y pasarse como argumentos a las funciones de vista correspondientes. Por ejemplo, en la ruta '/productos//', el parámetro capturaría un entero de la URL y se pasaría a la vista para identificar el producto específico.

Django alienta a los desarrolladores a crear rutas que sean legibles y entendibles para los usuarios. Esto se logra utilizando patrones de URL descriptivos que reflejen la estructura y la funcionalidad del sitio web. Las rutas pueden incluir palabras clave que describan la acción o el contenido al que conducen, lo que facilita la navegación y comprensión del sitio.

Configuración del entorno de desarrollo

Pasada la teoría básica sobre el funcionamiento de Django, vamos a ver todos los requisitos y procesos de instalación para empezar a trabajar con este framework.

Preparar el entorno para trabajar con Django es un proceso que consta de varios pasos, debido a las múltiples partes móviles de las que constan los entornos modernos de desarrollo Web.

Django está escrito totalmente en Python, por lo tanto, lo primero que necesitamos para usarlo, es asegurarnos de que tenemos instalada una versión apropiada de Python. Para trabajar con las versiones Django 2 y siguientes es imprescindible la versión de Python 3.x por lo que debemos asegurarnos de que tenemos instalada esta versión.

Visual Studio Code

Antes de seguir con Django, debemos tener instalado un editor de código para trabajar de una forma más cómoda.

Uno de los mejores editores de código disponibles para los programadores es Visual Studio Code. Es un editor de código abierto, extensible y ligero, disponible en todas las plataformas. Son estas cualidades las que hacen que Visual Studio Code de Microsoft sea muy popular y considerado como una gran plataforma para el desarrollo de Python por lo que es una buena opción para trabajar con Django.

La importancia de un entorno virtual

- Sin entornos virtuales: directamente en el ordenador que estamos utilizando, lo que simplificaría el proceso de instalación pero limita las posibilidades,
- Con entornos virtuales: realizar una instalación a través de un entorno de trabajo virtual.

Nosotros vamos a utilizar el segundo método, **crear un entorno virtual** que en un principio tiene un poco más de trabajo pero que a medio plazo nos será muy útil para poder aislar nuestros distintos procesos de desarrollo, pudiendo instalar versiones diferentes y paquetes distintos para realizar distintos tipos de proyectos, ya que, de otra forma solo podríamos instalar una versión de Django y de Python con la que trabajar.

Veamos la diferencia entre un ordenador con o sin entornos virtuales:

Herramientas para crear entornos virtuales hay muchas pero nosotros trabajaremos con PIPENV.

Pipenv

Como decíamos, para crear entornos virtuales existen muchas herramientas diferentes, una de ellas es virtualenv que tiene el propósito de generar dichos entornos aislados. El problema de este método es que obliga a crear los entornos uno a uno y a identificarlos con un nombre y una versión de Python. Ese es un problema común también en el gestor de paquetes conda, que es una alternativa a virtualenv que permite elegir la versión de Python, pero que también requiere otorgar nombres a los entornos.

Pipenv soluciona estos dos problemas, permite crear un entorno virtual individual para cada proyecto sin tener que darle un nombre. Solo hay que crear el entorno en el directorio del proyecto y manejarlo desde ahí.

La limitación que tiene este método es que está ligado a las versiones de Python que tengamos instaladas. Es decir, puedes crear entornos con otras versiones de Python diferentes a la que tienes por defecto, pero las necesitas previamente descargadas en la máquina y establecer las rutas durante la creación del entorno, verás que es bastante sencillo.

Instalación y uso de Pipenv

Para poder utilizar Pipenv, lo primero que hay que hacer es instalarlo.

Vamos a realizar esta operación a través del instalador pip, en este caso Python3 porque queremos que lo instale para Python3.

pip3 install pipenv

Si tienes errores con su utilización prueba a instalarlo de la siguiente forma:
sudo -H pip3 install pipenv
Esto funciona cuando quiero instalar un paquete de Python globalmente en /usr/local/bin/.

Una vez instalado, ya está listo para usar, si queremos crear un nuevo entorno virtual, navegamos con la terminal al directorio donde se encuentra nuestro proyecto, y ahí, creamos el entorno de la siguiente forma:

pipenv shell

Esto no sólo lo creará, también lo activará. Lo podemos comprobar porque en la parte izquierda de la terminal aparecerá el nombre del proyecto entre paréntesis:

(proyecto)

Para salir de él usamos el comando exit:

(proyecto) exit

Y para volver a activarlo, simplemente situándonos de nuevo en la carpeta del proyecto hacemos de nuevo:

pipenv shell

Siempre que tengamos la shell activa, el intérprete python hará referencia al del entorno virtual, sin embargo no es obligatorio tener la shell activada para seguir interactuando con el entorno haciendo uso de Pipenv.

Por ejemplo para instalar un paquete la lógica es simple, como si usáramos pip pero con Pipenv:

pipenv install

Y para desinstalar un paquete:

pipenv uninstall

O si queremos ver la lista de paquetes instalados en el entorno organizado por dependencias, podemos hacerlo con:

pipenv graph

Mantenimiento del entorno virtual

Vamos a ver cómo gestionar nuestro entorno, vamos a importar y exportar entornos para el desarrollo.

Supón que ha venido alguien nuevo al equipo de desarrolladores y necesitas clonar el entorno de trabajo en su máquina.
Para esto debe acceder a su espacio de trabajo y crear un entorno virtual en el directorio donde va a desarrollar el proyecto, una vez hecho esto, abrir el entorno de trabajo anterior a través del navegador de archivos y abrir el archivo pipfile.lock, copia el contenido que hay dentro de *packages y dev-packages* para pegarlo en el nuevo pipfile.lock del nuevo entorno.

Una vez hecho esto teclearemos lo siguiente.

pipenv install -ignore-pipfile

Esto instalará todos los paquetes y dependencias de nuestro entorno, podemos comprobarlo tecleando:

pipenv graph

Una vez clonado nuestro entorno y comprobado que sea así, simplemente hay que copiar la carpeta proyecto Django a nuestro nuevo clon del entorno y ya podríamos trabajar con él de forma como lo haríamos normalmente.

Otra opción posible es la necesidad de importar nuestro proyecto a un entono que no posea Pipenv y que necesite un archivo requirements.txt para instalar las dependencias, así que debemos ser capaces de crear dicho archivo.

Esto lo haremos tecleando

pipenv lock -r

y veremos como aparecen las dependencias en el formato requirements.txt, pero no se han guardado en ningún archivo, para guardarlo teclearemos:

pipenv lock -r > requirements.txt

Podremos ver que se ha creado si entramos en nuestro entorno virtual.

Por último vamos a tener la posibilidad del caso contrario, es decir, traer a un entorno Pipenv un entorno virtual creado a través de un archivo requirements.txt.

Para esto, una vez creado nuestro entorno vacío con Pipenv tecleamos:

pipenv install -r requirements.txt

De esta forma hemos visto como trasladar entornos virtuales de una forma fácil y sencilla.

Por último nos queda eliminar el entorno una vez hemos terminado de trabajar con él. Entra en la carpeta donde está el entorno creado (sin el entorno activo) y teclea:

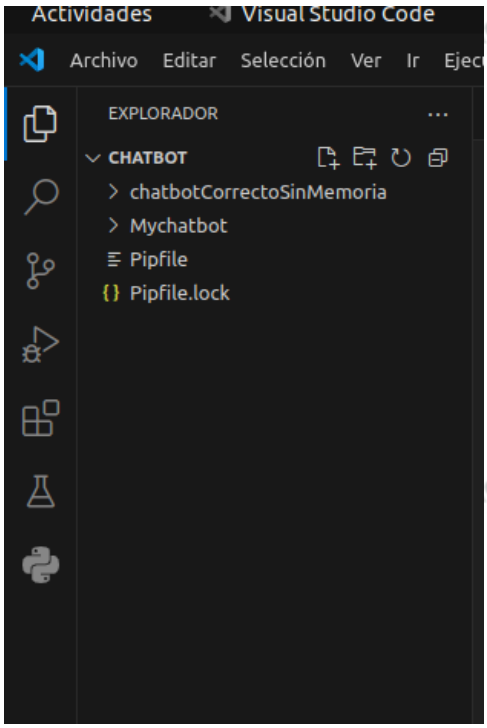
pipenv -rm

Esto eliminará el entorno virtual pero no borrará los archivos necesarios para restablecerlo, pudiendo hacer uso de ellos si fuera necesario recrear el entorno de trabajo eliminado.

Configurando Pipenv en Visual studio Code

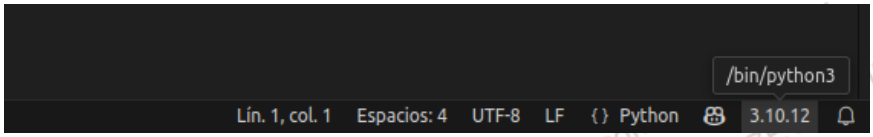
Antes de seguir trabajando vamos a configurar nuestro entorno de trabajo para pipenv en Visual studio Code. Como ya hemos dicho, Pipenv es una de las formas más modernas de gestionar las dependencias del proyecto en Python. Sin embargo, si quieres poder usar el plugin Python de Visual Studio Code en tu proyecto con entorno virtual, necesitas decirle dónde puede encontrar tu entorno virtual.

Abre la carpeta de tu entorno virtual



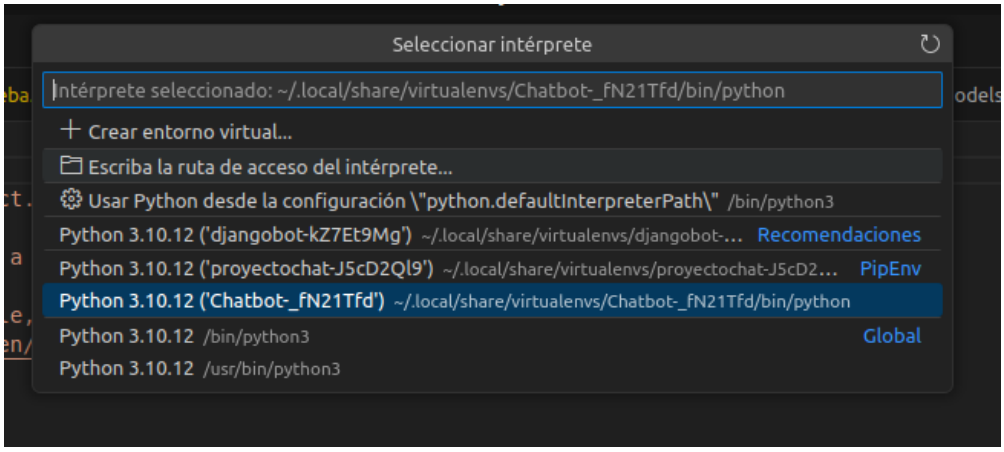
Una vez abierta la carpeta de tu entorno virtual donde vas a trabajar.

En la parte inferior derecha, verás una versión de python que es la que en estos momentos estás utilizando para esta carpeta



Se despliega una lista con todas las versiones de Python instaladas y cada uno de los entornos creado, ahora simplemente.

Seleccionas de entre todos los que aparecen el intérprete o la versión que pertenece a este entorno.



Ahora sí, estamos preparados para empezar a trabajar con Django de la manera más profesional

Estructura del proyecto Django

Una vez instalado Python y Django en el entorno virtual podemos recordar los primeros pasos en el desarrollo de aplicaciones creando un proyecto.

Un proyecto es una colección de configuraciones para una instancia de Django, incluyendo configuración de base de datos, opciones específicas de Django y configuraciones específicas de aplicaciones.

Crear el primer proyecto Django

Comenzar un proyecto

Debemos tener cuidado con algunas configuraciones iniciales.

Si antes has trabajado con PHP, seguramente pondrías el proyecto dentro de la carpeta raíz del servidor Web (en lugares como /var/www). Con Django, no debes que hacer esto. No es una buena idea poner cualquier código Python en la carpeta raíz del servidor Web, porque al hacerlo te arriesgas a que la gente sea capaz de ver el código de la Web. Esto no es bueno para la seguridad.

Cámbiate al directorio que acabas de crear dentro del entorno virtual donde instalaste Django, entra en la carpeta del proyecto, activa el entorno(si no lo tienes activo) y ejecuta el siguiente comando:

Django-admin startproject misitioweb

Este comando creará un directorio llamado ***misitioweb*** en el directorio donde está nuestro entorno virtual, creando una carpeta de trabajo que contiene varios archivos:

misitioweb/
manage.py
misitioweb/
__init__.py
asgi.py
settings.py
urls.py
wsgi.py

Estos archivos son los siguientes:

misitioweb/
El directorio de trabajo misitioweb/, es solo un contenedor, es decir una carpeta que contiene nuestro proyecto. Por lo que se le puede cambiar el nombre en cualquier momento sin afectar el proyecto en sí.
manage.py
Una utilidad de línea de comandos que te permite interactuar con un proyecto Django de varias formas. Es el archivo que nos va a ayudar gestionar todo nuestro proyecto y creación de aplicaciones. Usa manage.py help para ver lo que puede hacer. No deberías editar este archivo, ya que este es creado en el directorio convenientemente para manejar el proyecto.

misitioweb/misitioweb/
El directorio interno misitioweb/ contiene el paquete Python para el proyecto. El nombre de este paquete Python se usará para importar cualquier cosa dentro de él. (Por ejemplo import misitioweb.settings).
__init__.py
Un archivo requerido para que Python trate el directorio misitioweb como un paquete o como un grupo de módulos. Es un archivo vacío y generalmente no necesitarás agregarle nada.
settings.py
Las opciones/configuraciones para nuestro proyecto Django.
urls.py
Declaración de las URLs para este proyecto de Django. Piensa que es como una ""tabla de contenidos"" de tu sitio hecho con Django.
wsgi.py
El punto de entrada WSGI para el servidor Web, encargado de servir nuestro proyecto.
asgi.py
ASGI (Asynchronous Server Gateway Interface) es el sucesor de WSGI (Web Server Gateway Interface). El último lanzamiento de Django incluye soporte para ASGI.

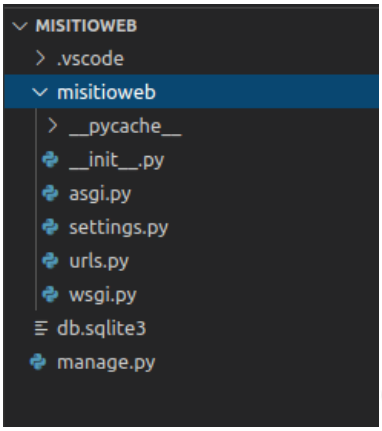
Todos estos pequeños archivos constituyen un proyecto Django que puede albergar múltiples aplicaciones.

Con esto ya tenemos nuestro primer proyecto creado con Django, ahora vamos a ver de forma más detallada algunas cosas importantes para nuestro desarrollo.

Probando el proyecto

Una vez tenemos nuestro primer proyecto vamos a abrir el directorio en Visual Studio Code desde Archivo > Abrir carpeta o arrastrándolo al programa.

Recuerda realizar el paso para indicarle a VSC donde encontrar el pythonpath para tu entorno virtual.



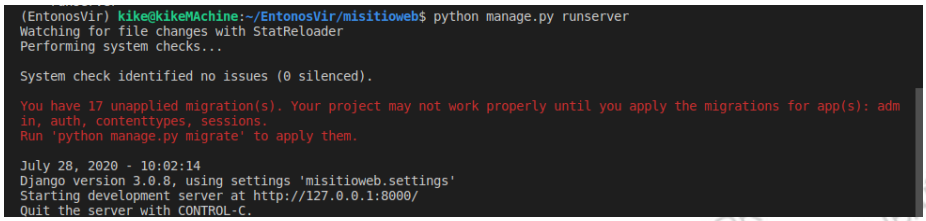
Lo primero que haremos será activar el entorno para poder trabajar, para esto pulsa con el botón derecho encima del archivo manage.py y haz clic en ***ejecutar archivo Python en la terminal***, esto hará que se active el entorno virtual pipenv dentro del programa VSC.

Si no te gusta este sistema o prefieres trabajar con la consola del sistema, no te preocupes, no es problema.

Lo hagas desde donde lo hagas, una vez tenemos nuestro entorno activado ***entra en la carpeta del proyecto y teclea***:

python manage.py runserver

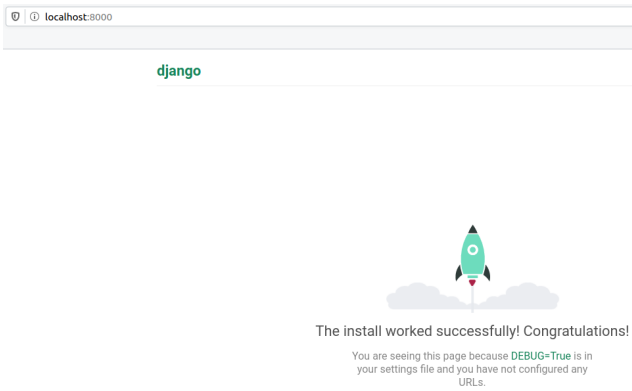
Vamos a analizar la salida de la consola.



Como puedes observar salta un error en rojo con una notificación avisando de que no hemos creado las migraciones, esto quiere decir que no existe la base de datos.

Aun así, esto activará el servidor virtual que Django trae por defecto para desarrollar las aplicaciones.

Ahora entrando en el navegador y tecleando localhost:8000 o 127.0.0.1:8000 aparecerá una pantalla avisándonos de que la instalación y la activación del servidor han sido un éxito.



De momento Django lo dejamos aquí, ya volveremos en unas unidades más adelante.

Resumen

- Django utiliza el patrón de diseño MVC, lo que facilita la organización del código mediante la separación de responsabilidades entre los modelos, vistas y controladores. Esto permite que los componentes de la aplicación trabajen de manera independiente y más eficiente.
- La administración automática de la base de datos en Django proporciona una interfaz administrativa lista para usar que refleja automáticamente los modelos de la base de datos. Por ejemplo, una vez definido un modelo de usuario, Django genera formularios para crear, editar y eliminar usuarios sin necesidad de configuración adicional.
- El sistema de enrutamiento y gestión de URL en Django proporciona un mapeo sencillo y poderoso de direcciones URL a vistas, permitiendo una navegación clara y lógica dentro de la aplicación web.
- Configurar un entorno de desarrollo adecuado es crucial, y Visual Studio Code es una herramienta recomendada por su versatilidad y soporte para Python y Django.
- Un entorno virtual es fundamental para aislar las dependencias de los proyectos y evitar conflictos. Pipenv es una herramienta que facilita este proceso en Django, y su integración en Visual Studio Code es sencilla.
- La estructura de un proyecto Django incluye directorios para manejar modelos, vistas, plantillas, archivos estáticos y otros componentes necesarios para crear aplicaciones web robustas y escalables.

Actividades prácticas

Creando el proyecto Django

En esta actividad práctica, aprenderás a crear un proyecto Django desde cero. A través de los pasos descritos, configurarás un entorno virtual usando Pipenv, iniciarás un proyecto Django y confirmarás que tu instalación ha sido exitosa ejecutando el servidor de desarrollo. Al completar esta actividad, habrás sentado las bases necesarias para desarrollar aplicaciones web con Django dentro de un entorno correctamente aislado.

Instrucciones:

1. Instala Pipenv si aún no lo has hecho, usando el comando `pip3 install pipenv`.
2. Crea un nuevo directorio para tu proyecto y navega hasta él en la terminal.
3. Dentro de este directorio, utiliza Pipenv para crear y activar un entorno virtual: `pipenv shell`.
4. Dentro del entorno virtual, instala Django utilizando: `pipenv install django`.
5. Inicia un nuevo proyecto Django con el comando `django-admin startproject misitioweb`.
6. Abrir el proyecto en Visual Studio Code asegurándote de configurar el entorno virtual en las preferencias del editor, tal como se mencionó en la teoría.
7. En la terminal de Visual Studio Code, activa el entorno si no está activado y ejecuta el comando `python manage.py runserver` para iniciar el servidor de desarrollo.
8. Accede a http://localhost:8000 en tu navegador para verificar que se ha iniciado correctamente el proyecto.

Envía capturas de pantalla de tu terminal al ejecutar cada uno de los pasos y del navegador mostrando la página de inicio de Django.

1. ¿Cómo se asegura que cada proyecto Django tenga su propio entorno virtual independiente con Pipenv?

2. ¿Por qué es importante no colocar el código Python en la carpeta raíz del servidor web al trabajar con Django?

3. ¿Cuál es el archivo que permite interactuar con un proyecto Django mediante la línea de comandos y para qué se utiliza?

4. ¿Qué mensaje debería aparecer si el servidor de desarrollo de Django se inició con éxito y estás accediendo a la URL correcta?

Procesando respuesta, no cierres el navegador, este proceso podría tardar unos segundos