Python web con Django

Módulo 1

educación IT
CODE YOUR FUTURE

Desarrollo web con Python

Desarrollo web con Python

Antes de avanzar con Django, es importante analizar cuál es el rol de Python, en general, en todo este esquema que se acaba de introducir.

Al desarrollar aplicaciones de escritorio con **Tcl/Tk,** en la programación Python, se diseña la **interfaz gráfica de usuario (GUI)** utilizando funciones del módulo estándar **tkinter**, y se asocian diversos eventos que ocurren en la ventana (por ejemplo, la presión de un botón) con funciones definidas en el código.

Esto implica que tanto la parte gráfica como la parte lógica de una aplicación de **Tcl/Tk** se desarrollan con el mismo lenguaje y las mismas herramientas (el módulo **tkinter**).

En el desarrollo web, es más o menos una regla general que la parte gráfica de una aplicación se debe desarrollar utilizando las siguientes tres tecnologías:

- El **lenguaje de etiquetas HTML,** para describir la estructura de la interfaz.
- El **lenguaje de estilos CSS,** para describir los estilos de la interfaz.
- El **lenguaje de programación JavaScript,** para ejecutar código en el navegador web que pueda realizar cambios en la interfaz.

Los códigos de estos tres lenguajes estarán almacenados en el *servidor*, pero serán interpretados o ejecutados una vez recibidos por el *cliente*. Es decir, todos los navegadores web incluyen intérpretes de HTML, CSS y JavaScript que se involucran en el proceso de transformar la respuesta del *servidor* en los sitios web que visitamos a diario.

Utilizando la opción **“Ver código de fuente”** de Firefox (otros navegadores tienen opciones similares) en el menú contextual podemos ver el contenido de la respuesta del *servidor*.

En los inicios de la web (y actualmente en sitios que no requieren de mucha interacción por parte del usuario), en la dirección de URL a la que se hace una petición se incluía el nombre del archivo HTML cuyo contenido se quiere retornar.

Por ejemplo, en una supuesta dirección:

https://ejemplo.com/contacto.html

se está indicando que se quiere obtener el contenido del archivo *contacto.html* que está alojado en el hosting al que apunta el dominio *ejemplo.com*

En ese hosting se está ejecutando el programa servidor (como Apache, NGINX, IIS, etc.) que recibe las peticiones del navegador web, lee el contenido del archivo que fue indicado en la dirección de URL y lo retorna como respuesta nuevamente
al navegador.

Cuando se visita **https://ejemplo.com/** el servidor web suele retornar por defecto el contenido del archivo ***index.html***.

Se puede imaginar un ejemplo: se está desarrollando un sitio como el de **EducaciónIT**, con una sección de cursos en donde se mostrará una lista de los que están disponibles con sus respectivas fechas de inicio.

En la década de los ‘90, se habría creado un archivo ***cursos.html***, accesible vía **https://ejemplo.com/cursos.html**y el administrador del sitio habría actualizado manualmente el contenido del archivo, cada vez que se agregaran cursos o fechas de inicio.

Con el tiempo, ante la necesidad de crear sitios web dinámicos, cuyo contenido no estuviese prefijado en un archivo HTML en el *servidor* sino que se construyese cada vez que se recibía una petición, se comenzaron a utilizar lenguajes como PHP, Perl, Java, y el mismo Python para que recibieran las peticiones y construyeran *en tiempo real* una respuesta en HTML.

Así, comenzaron a aparecer direcciones de URL como esta:

https://ejemplo.com/cursos.py

Cuando el programa servidor recibía una petición para un archivo con extensión **.py** (o .php , .pl , etc.), en lugar de retornar el contenido de ese archivo, el programa se ejecutaba con el intérprete correspondiente, y lo que se enviaba como res-
puesta era la salida de ese programa (esto es, en un script de Python, todo lo que haya sido impreso vía `print()`).

Así, este archivo cursos.py podría contener un código como el siguiente:

```
import sqlite3
print ( "<html>" )
print ( "<title>Lista de cursos</title>" )
conn = sqlite3.connect( "cursos.db" )
cursor = conn.cursor()
cursos = cursor.execute( "SELECT nombre FROM cursos" )
print ( "<ul>" )
for (nombre,) in cursos:
print ( f "<li> { nombre } </li>" )
print ( "</ul>" )
print ( "</html>" )
conn.close()
```

Si la tabla de cursos a la que se hace referencia en este código tiene cargados los cursos Python, Java y PHP, la salida del programa sería esta:

```
<html>
<title>Lista de cursos</title>
<ul>
<li>Python</li>
<li>Java</li>
<li>PHP</li>
</ul>
</html>
```

Esto es efectivamente lo que se enviaría al navegador web, que lo interpretaría y mostraría más o menos así:

Esto permitía crear sitios web con contenido diná-
mico generado en cada petición de una forma muy eficiente. Lamentablemente, cuando las aplicaciones web comenzaron a volverse más complejas durante la primera década del siglo XXI, este procedimiento de asociar una dirección de URL con un archivo ejecutable se volvió una arquitectura con muchas limitaciones.

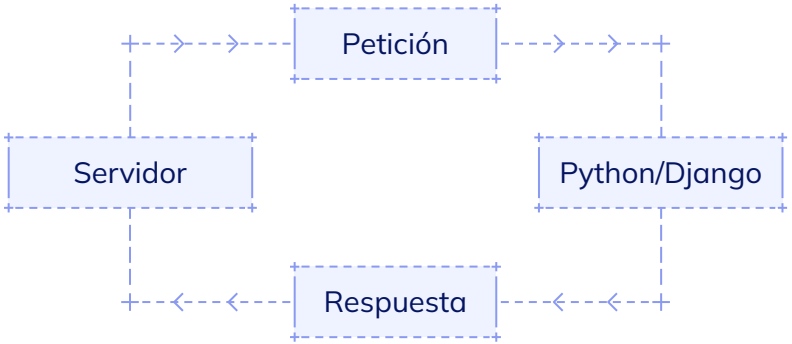
En ese momento, entran en escena muchos paquetes de Python —conocidos en la jerga como *web frameworks* — para desarrollo web, que nacieron con el objetivo de suplir las falencias del antiguo método. Django es uno de estos paquetes, cuya primera versión fue lanzada el año 2005.

En el antiguo método, el servidor web, al recibir una petición, ejecutaba un *script* determinado y devolvía la salida de ese programa al navegador. El diálogo entre el servidor web y los programas ejecutados se realizaba según las normas del protocolo CGI. Con el advenimiento de los *web frameworks* de Python se desarrolló un protocolo nuevo para la comunicación entre un servidor web y una aplicación de Python llamado WSGI.

Si en el protocolo CGI, una dirección de URL estaba asociada a un archivo ejecutable, en WSGI, por lo general, las direcciones de URL están asociadas con *funciones* de Python.

Esto quiere decir que cuando el navegador realiza una petición a una dirección de URL, el *servidor* envía, a su vez, esa petición a una función de una aplicación de Python (y Django en este caso), y éste devuelve una respuesta al *servidor*, que el *servidor* luego envía al cliente.

Se sintetiza este proceso en el siguiente diagrama:



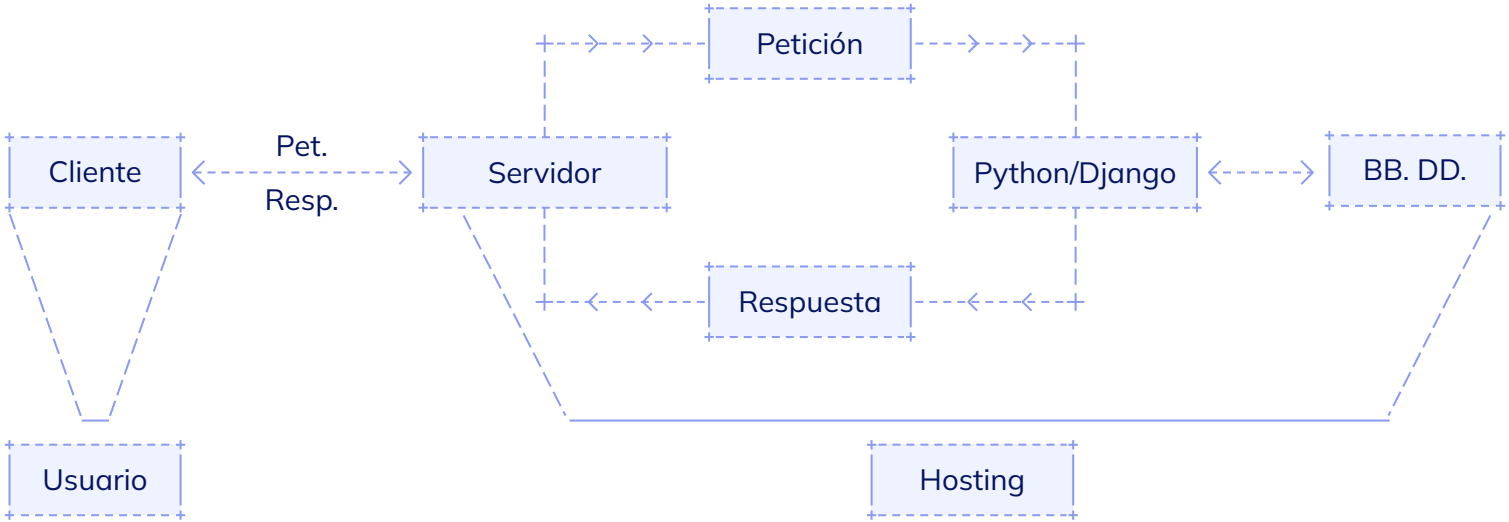
En otros términos: el *servidor* pasa el control de la petición a nuestra aplicación de Python y Django para que genere una respuesta que será enviada al cliente.

Desde nuestra aplicación podremos ejecutar cualquier código de Python, incluido todo lo visto en el curso de Python Programming. Por lo general, lo que hace una aplicación web al recibir una petición es interactuar con una base de datos y luego producir un código HTML, pero no hay ninguna restricción en ese sentido.

Algo para destacar es que el proceso que ilustra el diagrama anterior ocurre completamente en el *hosting* del sitio. Por eso todas las consultas a la base de datos, por ejemplo, se hacen desde la aplicación de Python y no vía JavaScript.

El usuario nunca podrá tener acceso al código de Python desde la opción de “Ver código fuente” mencionada anteriormente, puesto que esa información jamás es enviada al cliente, solamente la respuesta generada por dicho código.

Entonces, se agrega la posible interacción con la base de datos y se indica la distinción entre los procesos que se ejecutan en el navegador web y los que corren en el *hosting*, para ampliar el diagrama.


Petición
Cliente
Pet.
Resp.
Servidor
Python/Django
BB. DD.
Respuesta
Usuario
Hosting

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!