INFORMATORIO

Taller de Programación Web

PATRONES WEB

INFORMATORIO

Patrones Web

¿Qué es un Patrón de Arquitectura de Software?

La disciplina encargada de crear las estructuras fundamentales de un sistema de software se conoce como **arquitectura de software**. Las estructuras de software están formadas por elementos de software y sus relaciones que funcionan como un modelo, estableciendo el patrón de tareas a ejecutar. Los equipos de diseño de software dependen en gran medida de estos **patrones arquitectónicos de software**.

Cabe señalar que la arquitectura de software debe elegirse sabiamente porque una vez implementada no es fácil cambiarla.

Cuando nos iniciamos en el Desarrollo de Software es muy importante conocer respecto de los patrones de software:

- **Patrones de diseño** (singleton, repository, factory, builder, decorator, etc) que nos brindan una solución comprobada a problemas existentes y recurrentes
- **Patrones de desarrollo** (de los cuales vamos a dar más detalle a continuación) que son útiles para el diseño general de una aplicación o aplicaciones ya que abordan cuestiones específicas.

  Para seleccionar un patrón de desarrollo debemos considerar las siguientes características:

  - Costo
  - Número de usuarios (actuales y a futuro)
  - Nivel de aislamiento (es decir: integración con otros sistemas / plataformas)

Patrones de Desarrollo de Software

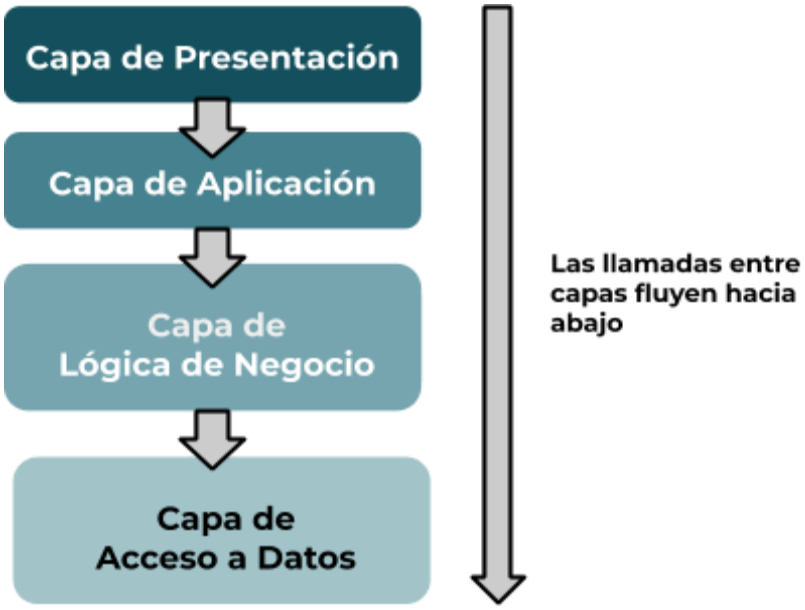
Estos son los patrones de Desarrollo de Software más utilizados.

Programación por capas

Este patrón puede usarse para estructurar programas que pueden descomponerse en grupos de subtareas, cada uno de los cuales se encuentra en un nivel particular de abstracción. Cada capa proporciona servicios a la siguiente capa superior.
Aquí están las capas más comunes:

- ➔ Capa de presentación
- ➔ Capa de aplicación
- ➔ Capa de lógica de negocio
- ➔ Capa de acceso a datos

Por lo general es utilizado para aplicaciones de escritorio o de e-commerce

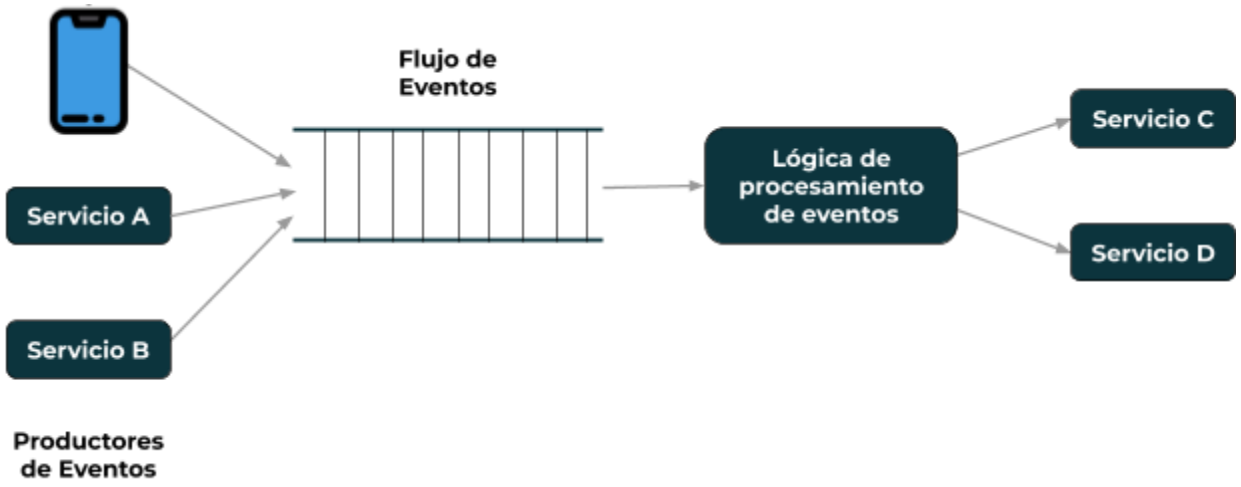


Dirigido por eventos

Las arquitecturas basadas en eventos son bastante populares en el desarrollo moderno de aplicaciones web. Son capaces de manejar una gran cantidad de conexiones concurrentes con un consumo mínimo de recursos.

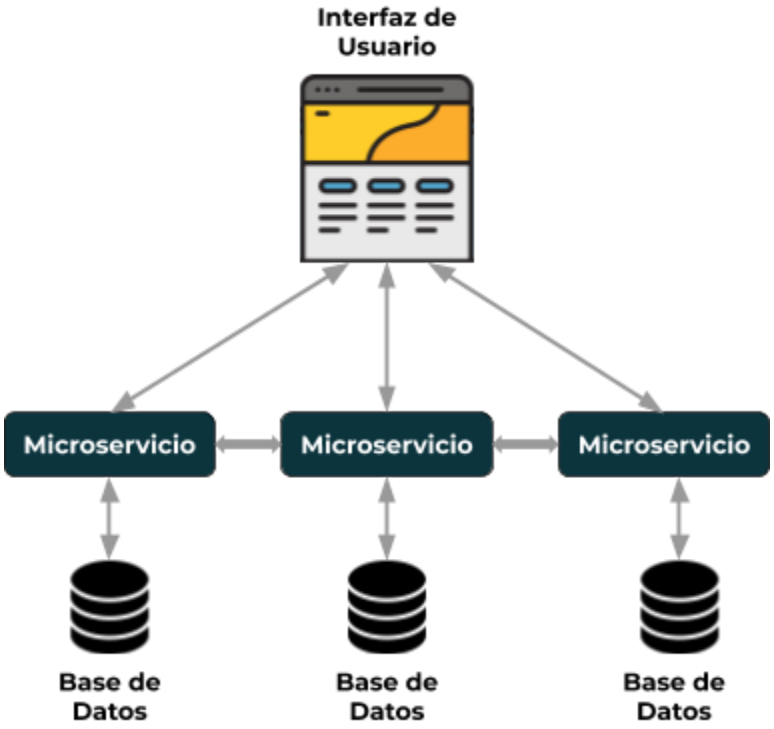
Las aplicaciones modernas necesitan un modelo completamente asíncrono para escalar. Estos marcos web modernos proporcionan un comportamiento más confiable en un entorno distribuido.

Por lo general es utilizado para Desarrollo en Android o servicios de notificaciones.



Microservicios

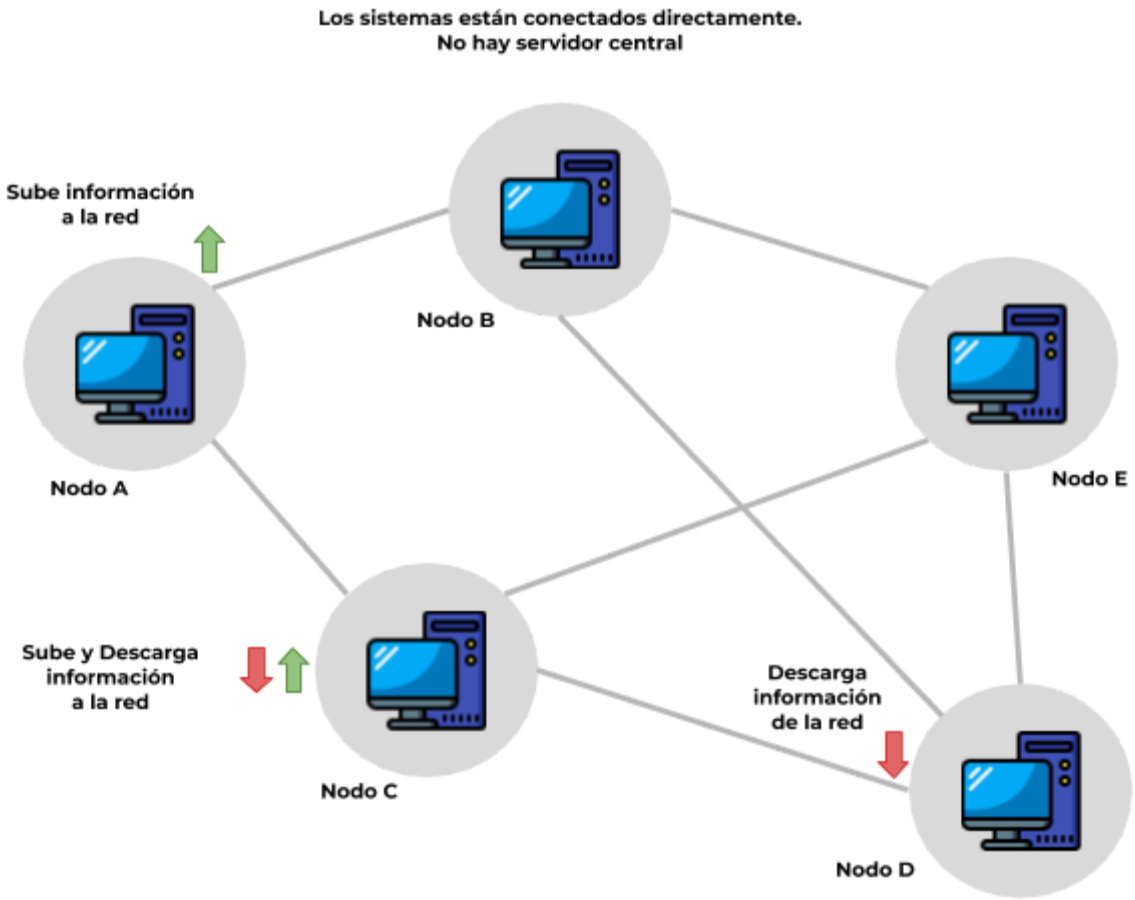
En una arquitectura de microservicios, las diferentes funciones/tareas se dividen en módulos/bases de código respectivos que funcionan en conjunto, formando un gran servicio en su conjunto. Esta arquitectura particular facilita el mantenimiento de aplicaciones más fácil y limpio, el desarrollo de características, las pruebas y la implementación en comparación con una arquitectura monolítica.



De igual a igual (Peer-to-peer)

Este patrón hace referencia a un tipo de arquitectura para la comunicación entre aplicaciones que permite a individuos comunicarse y compartir información con otros individuos sin necesidad de un servidor central que facilite la comunicación. Una red P2P es una red en la cual las computadoras también conocidas como nodos pueden comunicarse entre sí sin la necesidad de un servidor central. La ausencia de un servidor central descarta la posibilidad de un único punto de falla. Todas las computadoras en la red tienen los mismos derechos. Un nodo actúa como sembrador y leecher al mismo tiempo. Entonces, incluso si algunas de las computadoras / nodos se caen, la red y la comunicación aún están activas.

P2P es la base de la tecnología blockchain.



Modelo Vista - Controlador (MVC)

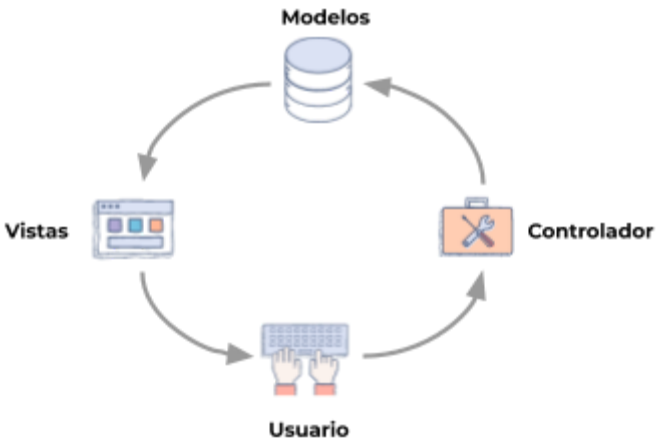
Es un patrón de software en el que la lógica de la aplicación se divide en tres componentes en función de la funcionalidad. Estos componentes se denominan:

- *Modelos*: representan cómo se almacenan los datos en las
- *Vistas* de la base de datos: los componentes que son visibles para el usuario, como una salida o un
- *Controlador* GUI (Graphic User Interface): los componentes que actúan como una interfaz entre modelos y vistas

La arquitectura MVC se usa no solo para aplicaciones de escritorio, sino también para aplicaciones móviles y web. Resumiendo

➔ Modelo: Contiene la funcionalidad y los datos básicos.
➔ Vista: Muestra la información al usuario (se puede definir más de una vista).
➔ Controlador: Maneja la entrada del usuario.

Esto se hace para separar las representaciones internas de información de las formas en que se presenta y acepta la información del usuario. Desacopla los componentes y permite la reutilización eficiente del código.



¿Modelo MVC o MVT en Django?

Veamos cómo funcionaría el Modelo MVC si lo aplicamos a Django:

➔ Modelo: El Modelo es la parte de la aplicación web que actúa como mediador entre la interfaz del sitio web y la base de datos, por lo tanto es el que contiene la lógica de negocio en la arquitectura de Django.

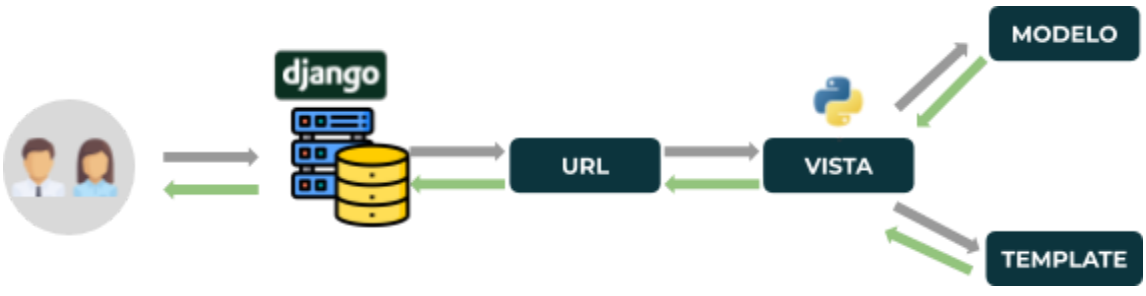
Por ejemplo: Si nos registramos en una tienda online en realidad estamos enviando información al controlador, que luego la transfiere esta información a los modelos, que a su vez aplican la lógica comercial y almacenan en la base de datos.

➔ Vista: Es en realidad la interfaz de usuario de la aplicación web y contiene partes como HTML, CSS, JS y otras tecnologías Frontend. En general, se crea a partir del componente Modelo, es decir, el contenido proviene del componente Modelo.

Por ejemplo: Cuando hacemos clic en cualquier enlace o interactuamos con los componentes del sitio web, las nuevas páginas web que genera ese sitio web son en realidad las vistas específicas que almacena y genera cuando interactuamos con los componentes específicos.

➔ Controlador: Es el componente de control principal, maneja la interacción del usuario y selecciona una vista de acuerdo con el modelo.
La tarea principal del controlador es seleccionar un componente de vista de acuerdo con la interacción del usuario y también aplicar el componente del modelo.

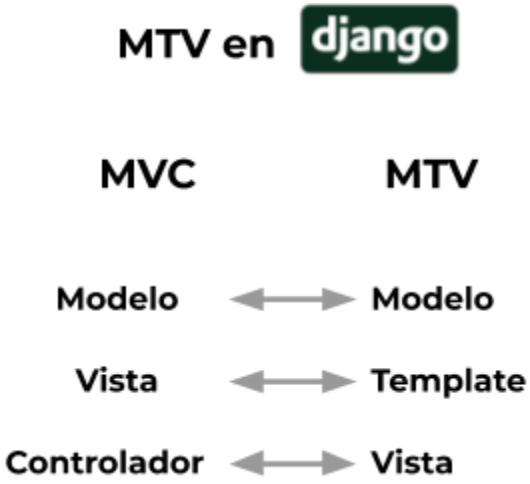
Por ejemplo: Cuando combinamos los dos ejemplos anteriores, podemos ver muy claramente que el componente que en realidad selecciona diferentes vistas y transfiere los datos al componente del modelo es el controlador.



Como podemos observar en el gráfico y teniendo en cuenta el detalle que dimos anteriormente de cómo funciona el Modelo MVC originalmente, podemos decir que Django utiliza los Templates como Vistas y las Vistas solo para los Controladores.

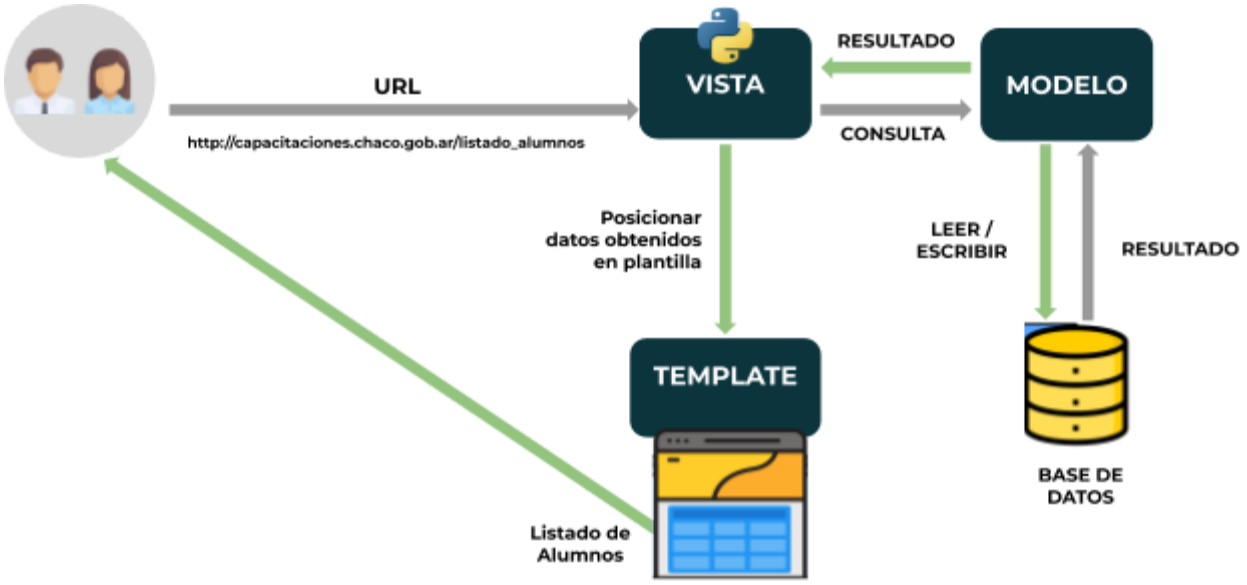
De esta manera entonces la plantilla se relaciona con la Vista en el patrón MVC, ya que se refiere a la capa de presentación que administra la lógica de presentación en el framework y esencialmente controla el contenido a mostrar y cómo mostrarlo para el usuario.

Por lo tanto, nuestro código Python estará en vistas y modelos, y el código HTML estará en plantillas.



Entonces en el Modelo MVT que es el que utiliza Django tendremos los siguientes componentes:

- Modelo: Es la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- Vista: Es la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelos y las plantillas.
- Template: Es la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento. Controla la interacción del usuario. Las plantillas se utilizan para especificar una estructura para la salida. Define cómo se presentan los datos.



Beneficios del modelo utilizado por Django

- **Desarrollo rápido**: Al separar los diferentes componentes facilita que múltiples desarrolladores trabajen en diferentes aspectos de la misma aplicación simultáneamente. Esa es también una de las características de Django.
- **Débilmente acoplado:** La arquitectura de Django tiene diferentes componentes que se requieren entre sí en ciertas partes de la aplicación, en cada instante, lo que aumenta la seguridad del sitio web en general. Como el archivo del modelo ahora solo se guardará en nuestro servidor en lugar de guardar en la página web.
- **Facilidad de modificación:** Este es un aspecto importante del desarrollo ya que hay diferentes componentes en la arquitectura de Django. Si hay un cambio en diferentes componentes, no tenemos que cambiarlo en otros componentes.

  Esta es en realidad una de las características especiales de Django, ya que aquí nos proporciona mucha más adaptabilidad de nuestro sitio web que otros frameworks.