Brian Juan Carlos Oyardo Poma
Universidad La Salle
30 de Septiembre de 2023

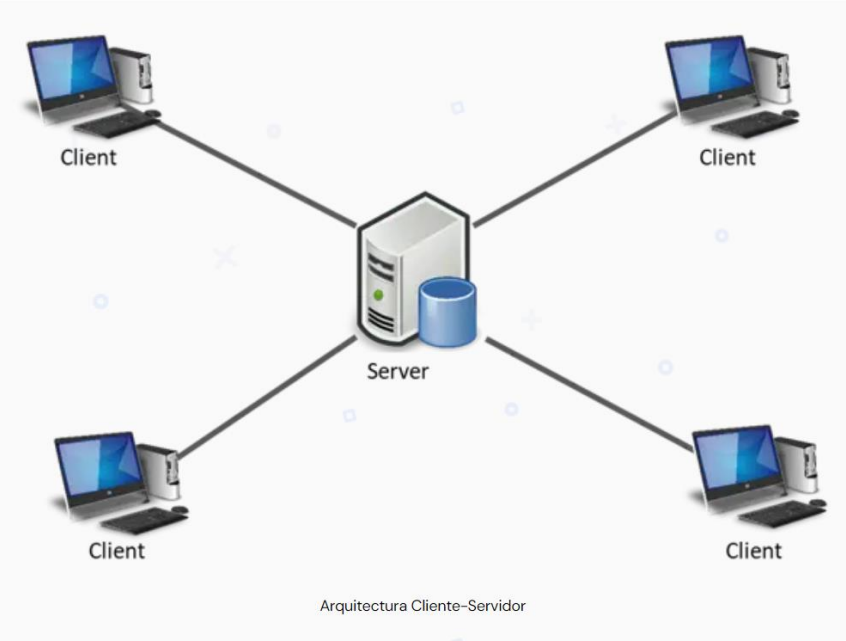
Investigación Lenguaje Audiovisual

Arquitecturas de Software mas Utilizadas en la Actualidad

Arquitectura Cliente Servidor:

Cliente-Servidor es uno de los estilos arquitectónicos distribuidos más conocidos, el cual está compuesto por dos componentes, el proveedor y el consumidor. El proveedor es un servidor que brinda una serie de servicios o recursos los cuales son consumido por el Cliente.

En una arquitectura Cliente-Servidor existe un servidor y múltiples clientes que se conectan al servidor para recuperar todos los recursos necesarios para funcionar, en este sentido, el cliente solo es una capa para representar los datos y se detonan acciones para modificar el estado del servidor, mientras que el servidor es el que hace todo el trabajo pesado.



Arquitectura Cliente-Servidor

En esta arquitectura, el servidor deberá exponer un mecanismo que permite a los clientes conectarse, que por lo general es TCP/IP, esta comunicación permitirá una comunicación continua y bidireccional, de tal forma que el cliente puede enviar y recibir datos del servidor y viceversa.

- Clientes y Servidores:

  En esta arquitectura, el sistema se divide en dos partes fundamentales: el cliente y el servidor.

El cliente es la parte de la aplicación que interactúa directamente con el usuario. Puede ser una aplicación de escritorio, una aplicación web o incluso una aplicación móvil.

El servidor, por otro lado, es el componente que proporciona los recursos y los servicios requeridos por el cliente. Estos recursos pueden ser datos, archivos, procesamiento, autenticación, etc.

- Comunicación:

  La comunicación entre el cliente y el servidor se realiza a través de una red, generalmente utilizando protocolos de comunicación estándar, como HTTP, TCP/IP o WebSocket.

  El cliente envía solicitudes al servidor para acceder a los recursos o servicios requeridos, y el servidor procesa estas solicitudes y devuelve las respuestas correspondientes.

  Separación de Responsabilidades:

  La arquitectura cliente-servidor se basa en el principio de separación de responsabilidades. El cliente es responsable de la interfaz de usuario y la interacción con el usuario, mientras que el servidor se encarga de la lógica empresarial y el almacenamiento de datos.

  Esta separación facilita el desarrollo y el mantenimiento, ya que los cambios en la interfaz de usuario (cliente) no afectan directamente a la lógica empresarial (servidor) y viceversa.

- Escalabilidad:

  La arquitectura cliente-servidor es altamente escalable. Puedes agregar más clientes sin afectar significativamente el servidor, lo que facilita la expansión de aplicaciones para atender a un número creciente de usuarios.

  Los servidores también pueden ser escalados para gestionar una mayor carga de trabajo mediante la adición de servidores adicionales.

- Seguridad:

  La seguridad es un aspecto importante en esta arquitectura. Los servidores suelen ser responsables de la gestión de la seguridad y la autenticación de los usuarios.

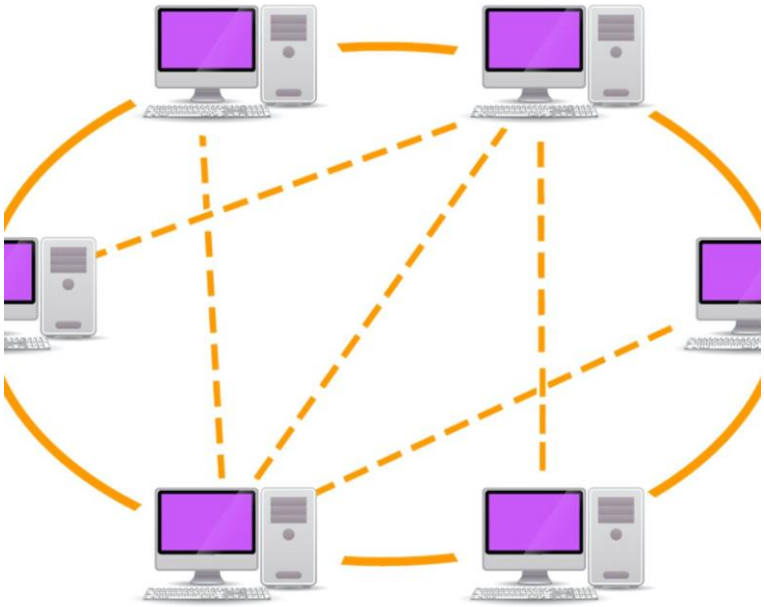
  Se pueden implementar medidas de seguridad, como cortafuegos, autenticación de usuarios y cifrado de datos, en ambos lados (cliente y servidor) para proteger la comunicación y los recursos.

- Ejemplos de Aplicaciones:

  La arquitectura cliente-servidor se utiliza en una amplia variedad de aplicaciones, desde navegadores web (donde el navegador es el cliente que se conecta a servidores web) hasta sistemas de gestión de bases de datos (donde las aplicaciones cliente acceden a datos almacenados en servidores de bases de datos).

Arquitectura Red entre pares:

Más conocida como red peer-to-peer, es una de las arquitecturas de software que permite compartir una gran cantidad de datos. A diferencia del modelo cliente-servidor, este consiste en una red descentralizada de clientes y servidores. Todas las partes consumen recursos, sin la necesidad de un servidor centralizado.



- Igualdad entre Nodos:

  En una red entre pares, todos los nodos se consideran iguales. Cada nodo puede actuar como cliente y servidor al mismo tiempo, lo que significa que pueden solicitar y proporcionar recursos.

- Descentralización:

  La arquitectura P2P elimina la dependencia de un servidor central. En cambio, los nodos se comunican directamente entre sí. Esto mejora la resistencia de la red, ya que no hay un solo punto de falla.

- Recursos Compartidos:

  Los nodos en una red P2P pueden compartir una variedad de recursos, como archivos, ancho de banda de Internet, potencia de cómputo y servicios. Un ejemplo conocido de una red P2P es BitTorrent, que se utiliza para compartir archivos entre usuarios.

- Escalabilidad:

Las redes P2P son inherentemente escalables. A medida que se agregan más nodos a la red, se incrementa la capacidad para compartir recursos y se mejora la redundancia.

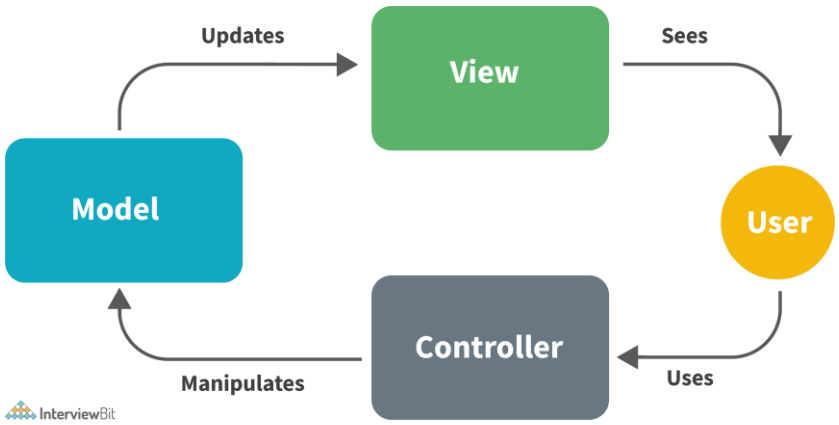
- Eficiencia:

  Debido a que los recursos se distribuyen entre los nodos, las redes P2P pueden ser eficientes en términos de ancho de banda y recursos de hardware. Los nodos pueden buscar el recurso más cercano en la red en lugar de depender de un servidor remoto.

Arquitectura Modelo Vista Controlador (MVC):

Emplea tres componentes: modelo, vista y controlador. El modelo se hace cargo de los datos, ya sean actualizaciones, búsquedas u otros. El controlador recibe las órdenes del cliente para, posteriormente, solicitar los datos al modelo y comunicar a la vista, que es la representación visual de los datos (interfaz gráfica).

En este tipo de arquitecturas de software, el proceso de desarrollo es más rápido, ya que varios desarrolladores pueden trabajar a la vez. Además, las modificaciones, de la misma forma que pasa con el patrón anterior, no afectan a la arquitectura entera.



- Modelo (Model):

  El Modelo representa la capa de datos de la aplicación. Contiene la lógica empresarial, las reglas de negocio y la gestión de datos.

  Se encarga de interactuar con la base de datos o cualquier otro sistema de almacenamiento de datos. Realiza operaciones como consultas, actualizaciones y validaciones de datos.

  El Modelo no tiene conocimiento ni dependencia de la interfaz de usuario ni de cómo se presentan los datos.

- Vista (View):

  La Vista es la capa de presentación de la aplicación. Representa la interfaz de usuario y cómo se muestran los datos al usuario.

Se encarga de la presentación visual y la interacción con el usuario, como la generación de formularios, botones y elementos gráficos.

La Vista no realiza ninguna lógica empresarial ni manipulación de datos, simplemente muestra la información proporcionada por el Modelo y permite la interacción del usuario.

- Controlador (Controller):

  El Controlador actúa como intermediario entre el Modelo y la Vista. Gestiona las solicitudes del usuario, coordina las acciones necesarias y decide cómo responder.

  Procesa las entradas del usuario y las convierte en acciones en el Modelo, como leer, escribir o actualizar datos.

  Luego, actualiza la Vista para reflejar los cambios en los datos del Modelo.

  El Controlador se encarga de mantener la lógica de flujo de la aplicación y toma decisiones basadas en la entrada del usuario y el estado del Modelo.

- Ventajas de MVC:

  Separación de preocupaciones: MVC permite una clara separación de las responsabilidades, lo que facilita el desarrollo, el mantenimiento y la colaboración entre equipos de desarrollo.

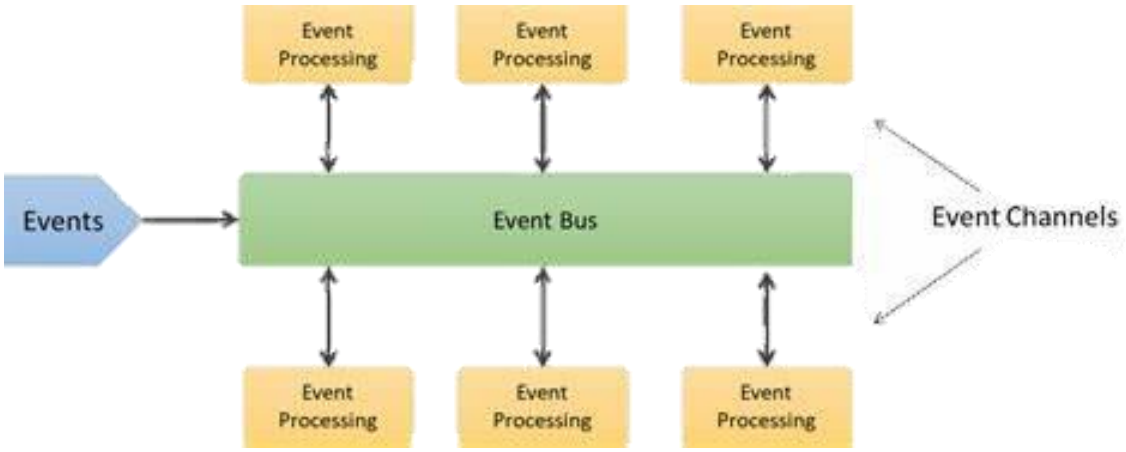
  Reutilización de componentes: Cada componente (Model, Vista y Controlador) puede reutilizarse en diferentes partes de la aplicación o incluso en diferentes proyectos.

  Facilita las pruebas: Debido a la separación de responsabilidades, es más sencillo realizar pruebas unitarias en cada componente.

<u>Arquitectura Orientada a Eventos:</u>

Se diferencia de otras arquitecturas de software porque es asíncrona y distribuida, utilizada principalmente para la creación de aplicaciones escalables. Sus componentes no se comunican de forma síncrona. El resultado esperable es que las aplicaciones envíen eventos y otros componentes reaccionen a ellos, procesándolos y generando nuevos eventos.

Entre sus ventajas, la arquitectura orientada a eventos se caracteriza por su flexibilidad. Debido a que los componentes procesadores de eventos tienen solamente una responsabilidad, si acontece un cambio se aísla un solo componente, manteniendo el resto inalterado.



- Eventos y Mensajes:

  En esta arquitectura, los componentes de la aplicación se comunican entre sí a través de eventos o mensajes. Un evento puede ser cualquier tipo de notificación, cambio de estado o acción que ocurra en un componente y que sea relevante para otros componentes.

- Desacoplamiento:

  La arquitectura orientada a eventos fomenta el desacoplamiento entre componentes. Esto significa que los componentes no necesitan conocer la estructura interna de los demás ni estar directamente conectados. En cambio, interactúan a través de eventos, lo que permite una mayor flexibilidad y reutilización de componentes.

- Publicación y Suscripción:

  Un patrón común en esta arquitectura es el patrón de publicación y suscripción. Los componentes (suscriptores) se registran para recibir notificaciones de eventos específicos. Cuando un evento relevante se produce, se envía a todos los suscriptores interesados, lo que les permite responder de acuerdo a sus necesidades.

- Flujo de Control Reactivo:

  La arquitectura orientada a eventos se presta bien a la programación reactiva, donde los componentes reaccionan a los eventos a medida que ocurren. Esto es útil en aplicaciones en tiempo real y en escenarios donde las acciones de un componente pueden desencadenar respuestas en otros componentes.

- Escalabilidad:

  Esta arquitectura es escalable y flexible, ya que es relativamente fácil agregar nuevos componentes o modificar la lógica de respuesta a eventos sin afectar el sistema en su conjunto.

- Ejemplos de Aplicación:

La arquitectura orientada a eventos se encuentra en una variedad de aplicaciones, como aplicaciones web en tiempo real, sistemas de control industrial, sistemas de notificación, juegos y aplicaciones de mensajería instantánea.

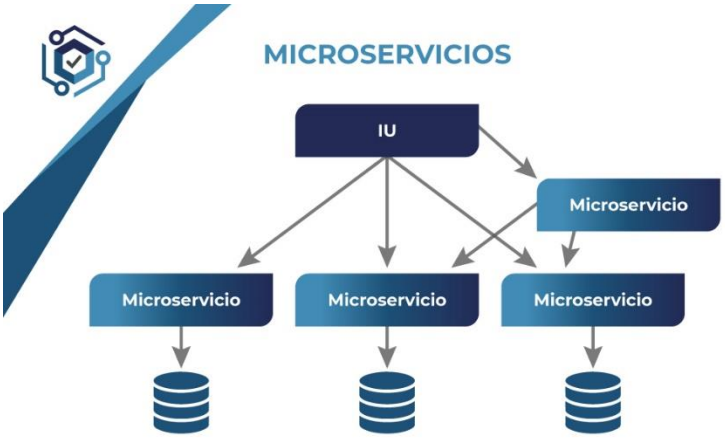
- Desafíos:

  La gestión de eventos y la coordinación de múltiples componentes pueden complicarse en aplicaciones complejas. La depuración y el seguimiento de eventos a través de múltiples componentes también pueden ser desafiantes.

Arquitectura de Microservicios:

La arquitectura de microservicios es una de las más buscadas en la actualidad. Consiste en la creación de componentes de software que se dedican a realizar una única tarea y son autosuficientes, por lo que evolucionan de forma independiente.

Vale aclarar también que, cuando nos referimos a microservicios, hablamos de pequeños programas (aplicaciones) que brindan servicios con el fin de resolver determinadas tareas. Como gran ventaja, la arquitectura de microservicios destaca por sus componentes encapsulados, ya que pueden evolucionar a la velocidad requerida y cada microservicio se puede desarrollar con distintas tecnologías (bases de datos).



- Descomposición en Servicios:

  En una arquitectura de microservicios, la aplicación se divide en múltiples microservicios, cada uno de los cuales realiza una tarea específica. Por ejemplo, un servicio puede encargarse de la autenticación de usuarios, otro de la gestión de pedidos y otro de la generación de informes.

  Cada microservicio es independiente y puede desarrollarse, desplegarse y escalar de forma separada.

- Independencia y Escalabilidad:

  Cada microservicio es independiente en términos de tecnología, base de datos y desarrollo. Esto permite que los equipos se centren en un solo servicio y tomen decisiones tecnológicas que sean apropiadas para ese servicio en particular.

  La arquitectura de microservicios facilita la escalabilidad, ya que puedes escalar solo los servicios que requieren más recursos sin afectar a otros.

- Comunicación entre Microservicios:

  Los microservicios se comunican entre sí a través de interfaces bien definidas, como API REST o mensajería. Esto permite que los servicios interactúen de manera eficiente y se mantengan desacoplados.

  Las llamadas a través de la red son comunes en esta arquitectura, lo que significa que se debe prestar atención a la latencia y la disponibilidad de la red.

- Facilita la Implementación Continua:

  La arquitectura de microservicios facilita la implementación continua (CI/CD) ya que cada servicio se puede implementar de forma independiente, lo que reduce el impacto de los cambios en otras partes de la aplicación.

- Tolerancia a Fallos:

  Dado que los microservicios son independientes, un fallo en uno de ellos no afecta necesariamente a los demás. Esto hace que el sistema en su conjunto sea más tolerante a fallos.

- Gestión de Datos:

  La gestión de datos puede ser un desafío en una arquitectura de microservicios, ya que cada servicio puede utilizar su propia base de datos. Se requiere coordinación y soluciones como bases de datos políglotas o bases de datos en memoria compartida.

Arquitectura Orientada a Servicios (SOA):

La Arquitectura Orientada a Servicios (SOA) es un enfoque de diseño de software que se centra en la creación y el uso de servicios independientes y reutilizables. En SOA, los servicios representan unidades funcionales de software que se comunican entre sí a través de estándares de comunicación, como el Protocolo de Acceso a Objetos (SOAP) o el Protocolo de Transferencia de Hipertexto Simple (HTTP).

- Servicios:

  Los servicios en SOA son componentes de software independientes que encapsulan una funcionalidad específica. Pueden ser pequeños módulos que realizan tareas como procesamiento de pedidos, autenticación de usuarios o generación de informes.

  Los servicios son autónomos y se comunican a través de interfaces bien definidas. Pueden ser reutilizados en diferentes aplicaciones y contextos.

- Desacoplamiento:

  SOA promueve el desacoplamiento entre los servicios. Esto significa que un servicio no necesita conocer los detalles internos de otro servicio con el que se comunica. Se centra en la definición de contratos claros (interfaces) para la comunicación.

- Reutilización:

  Uno de los objetivos principales de SOA es fomentar la reutilización de servicios. Al definir y crear servicios independientes y bien encapsulados, puedes utilizarlos en diferentes aplicaciones y sistemas, lo que ahorra tiempo y esfuerzo de desarrollo.

- Estandarización:

  SOA utiliza estándares de comunicación, como XML y HTTP, para garantizar que los servicios puedan comunicarse de manera efectiva, independientemente de las tecnologías utilizadas en su implementación.

  Los protocolos de descripción de servicios, como WSDL (Web Services Description Language), se utilizan para definir las interfaces de los servicios.

- Gestión de Servicios:

  En SOA, la gestión de servicios es fundamental. Esto incluye la publicación, el descubrimiento, la orquestación y la supervisión de servicios. Los registros de servicios (registros UDDI) se utilizan para el descubrimiento de servicios.