Documento

Natalia Gaona Salamanca

Laura Sofia Jiménez

Pontificia Universidad Javeriana

Bogotá D.C, Colombia

2023

TABLA DE CONTENIDO

1. Definición, Historia y Evolución

1.1. Sistemas Distribuidos

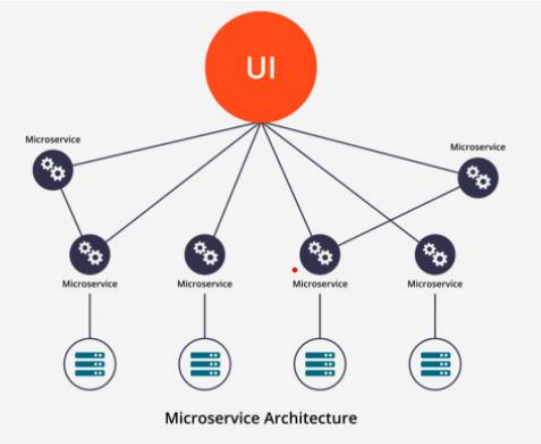
1.1.1. Definición

"Un sistema distribuido es un conjunto de programas informáticos que utilizan recursos computacionales en varios nodos de cálculo distintos para lograr un objetivo compartido común. Este tipo de sistemas, también denominados "computación distribuida" o "bases de datos distribuidas", usan nodos distintos para comunicarse y sincronizarse a través de una red común. Estos nodos suelen representar dispositivos de hardware físicos diferentes, pero también pueden representar procesos de software diferentes u otros sistemas encapsulados recursivos. La finalidad de los sistemas distribuidos es eliminar los cuellos de botella o los puntos de error centrales de un sistema".(Atlassian, s. f.)

1.1.2. Microservicios

Los microservicios son un método de desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, con operaciones bien definidas e independientes entre sí.

"Cada microservicio ejecuta su propio proceso y se encarga de implementar una funcionalidad completa del negocio. Puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos. A la hora de hacer un despliegue, cada servicio se hace de forma independiente."(Hiberus, 2021)



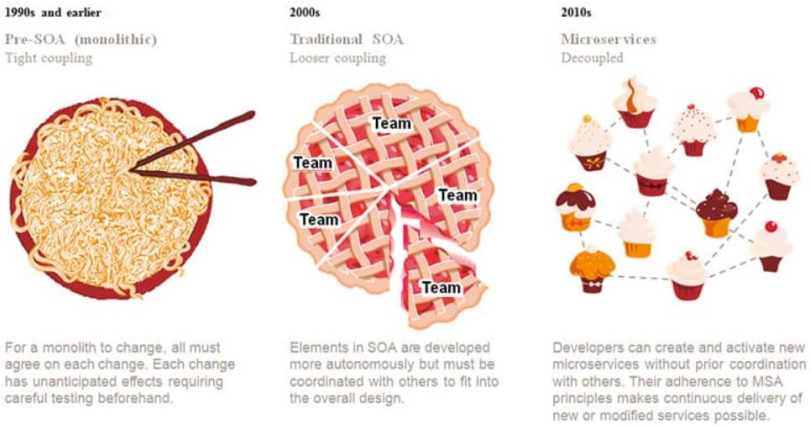
1.1.2.1. Historia y evolución

Tradicionalmente la arquitectura de software se centró en las arquitecturas monolíticas, las cuales son un solo componente que posee distintas funcionalidades. En este tipo de arquitectura el código está mezclado y es complejo de mantener debido a que cualquier modificación afecta a todo el proyecto en sí.

Durante la década de los noventa se originaron los esquemas de progreso SOA, los cuales se hicieron populares a principios del siglo XXI. En las estructuras basadas en servicios, los elementos de la aplicación se encuentran en cierta medida aislados y prestan servicios a otros componentes por medio de un protocolo de comunicaciones en red.(Martín, 2018)

El modelo SOA representa una progresión hacia arquitecturas más descentralizadas, pero dado que los componentes no son completamente autónomos, el proceso de desarrollo continúa siendo complejo y requiere que los equipos de trabajo se pongan de acuerdo en cada modificación de código. Además, estas estructuras utilizan tecnologías como los servicios web SOAP (creados durante la época de auge del XML) y los sofisticados ESB (Enterprise Service Bus), los cuales están en declive en la actualidad. El resultado es un proceso de lanzamiento muy lento y un mantenimiento tedioso. (*Idem*.).

En contraste, los microservicios llevan la idea de SOA a otro nivel. La arquitectura basada en microservicios (MSA) se enfoca en servicios que son piezas completamente autónomas, los cuales pueden ser actualizados y desplegados sin necesidad de cambiar los demás componentes. (I*dem*.).



1.2. .Net

1.2.1. Definición

.NET es una plataforma de aplicaciones que permite la creación y ejecución de servicios web y aplicaciones de Internet. En la plataforma de desarrollo se pueden utilizar una serie de lenguajes, implementaciones, herramientas y bibliotecas para el desarrollo de las aplicaciones.

En definitiva, es hoy en día la plataforma de desarrollo de software más usada para nuevos proyectos de desarrollo de software además de Java.

“Microsoft .NET es una colección de diferentes plataformas de software de Microsoft. El framework original fue desarrollado como una competencia directa a la plataforma Java. Así pues, los entornos de aplicación pueden ser desarrollados y ejecutados en base a .NET.”(aula21, 2020)

1.2.2. Historia y evolución

La historia de .Net se remonta a los años 80, cuando Anders Hejlsberg comenzó a escribir compiladores y creó Pascal y DOS en su primera empresa, PolyData. Posteriormente, se unió a Borland y contribuyó al desarrollo de Turbo Pascal y Delphi, cuyos compiladores se convirtieron en los más vendidos del mundo.(Londoño, 2017)

En 1996, Hejlsberg se unió a Microsoft y trabajó en el proyecto J++, una implementación de Java que incluía la creación de interfaces gráficas (WFC). Sin embargo, el proyecto fue demandado por Sun y no pudo continuar según lo planeado. (I*dem*.).

Basado en estas experiencias, se inició el proyecto .Net, un nuevo desafío que incluía un nuevo lenguaje de programación. Es importante mencionar a Anders porque su pasión por la programación y su compromiso en proporcionar herramientas de alta calidad a los desarrolladores han sido fundamentales para la creación de .Net. (I*dem*.).

A finales de la década de 2000, se lanzó la primera versión beta de .Net, y en 2002 Microsoft lanzó oficialmente la versión 1.0. .Net Framework es el conjunto de elementos necesarios para desarrollar software en la plataforma Microsoft, permitiendo a los desarrolladores crear aplicaciones para Windows, la web, servicios y móviles, con una experiencia de usuario consistente en todos los tipos de aplicaciones y con un entorno que facilita su ejecución. (I*dem*.).

Entre los elementos encontramos: Herramientas de ejecución de aplicaciones (CLR) + Librerías de uso común (BCL -> FCL) + Entorno de desarrollo (Visual Studio) + Lenguajes (C#, VB.Net, F#). (I*dem*.).

1.2.2.1. CLR

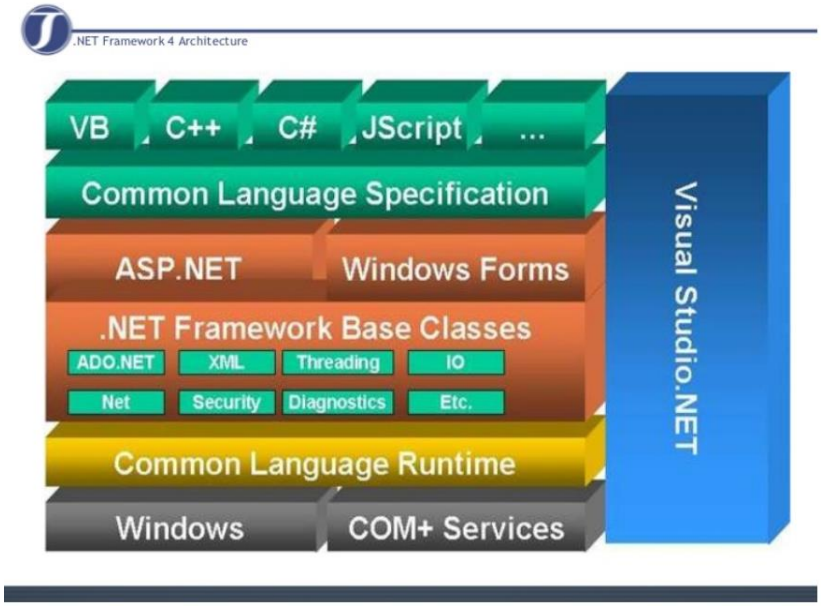
El CLR (Common Language Runtime) es el núcleo de .Net encargado de la ejecución del código, la gestión de la seguridad, manejo de excepciones, gestión de memoria y manejo de hilos, entre otros aspectos. Uno de los objetivos de .Net es permitir que cualquier lenguaje pueda utilizar su infraestructura, no solo los lenguajes de Microsoft. Para lograr esto, se definen estándares bajo los cuales los lenguajes pueden ser ejecutados por el CLR, conocidos como CLI (Common Language Infrastructure). Esta infraestructura consta de cuatro elementos: CTS (Common Type Specification) para tipos y operaciones permitidas, CLS (Common Language Specification) para reglas de acción, metadata para información sobre la estructura del programa independiente del lenguaje de programación y VES (Virtual Execution System) para cargar y ejecutar programas compatibles con la infraestructura CLI. Los programas compilados en .Net generan un código intermedio (CIL: Common Intermediate Language) que es independiente del hardware y se convierte en lenguaje de máquina durante la ejecución. (I*dem*.).

1.2.2.2. BCL->FLC

Inicialmente, solamente existía el componente BCL (Base Class Library) en .Net Framework. Posteriormente, Microsoft lanzó el componente FCL (Framework Class Library), el cual incluye más elementos que BCL. BCL es, en esencia, una parte de FCL, y ambos tienen como propósito proveer código reutilizable con características comunes que son necesarias en todas las aplicaciones, como acceso a bases de datos, archivos, redes, XML, y otros similares. (I*dem*.).

1.2.2.3. Visual Studio

El el IDE (Integrated Development Environment) o herramienta que que permite escribir aplicaciones en.Net. (I*dem*.).



1.3. Vue

1.3.1. Definición

"Vue es un framework open source de JavaScript, el cual nos permite construir interfaces de usuarios de una forma muy sencilla. La curva de aprendizaje, desde mi punto de vista, es relativamente baja, claro, debes conocer muy bien JavaScript, saber trabajar con callbacks, promesas, objetos, entre otros temas más."(*¿Qué es Vue.JS?*, s. f.)

1.3.2. Historia y evolución

Vue fue creado por Evan You después de trabajar para Google usando AngularJS en varios proyectos. Más tarde resumió su proceso de pensamiento: "Pensé, ¿qué tal si pudiera extraer la parte que realmente me gustaba de Angular y construir algo realmente liviano?". El primer código fuente comprometido con el proyecto fue fechado en julio de 2013, y Vue fue liberado por primera vez en febrero siguiente, en 2014.

1.4. Microsoft SQL Server

1.4.1. Definición

Microsoft SQL Server es una popular base de datos relacional que se utiliza en una amplia gama de aplicaciones empresariales para la inteligencia empresarial y el análisis de datos. Utiliza el lenguaje Transact-SQL y viene con una serie de extensiones de programación propias del estándar del lenguaje. SQL Server está disponible tanto en modalidad on premise como en la nube.(*¿Qué es Microsoft SQL Server y para qué sirve?*, s. f.)

1.4.2. Historia y evolución

En la década de los 90, cuando comencé a trabajar con SQL Server 6.0 y 6.5, la única alternativa para utilizarlo era en la plataforma Windows. En aquel entonces, las opciones disponibles eran Windows Server 3.5 y Windows NT 4. Me acuerdo claramente de los grandes servidores NEC y Compaq Proliant que se utilizaban para crear entornos con discos mecánicos compartidos, donde se ejecutaban las primeras versiones de Windows Clúster para lograr una mayor disponibilidad.(*Evolución de SQL Server en múltiples plataformas | CompartiMOSS*, s. f.)

Durante el 2016, la virtualización se convirtió en una realidad y muchas personas me preguntaban si era posible ejecutar SQL Server en entornos virtuales de VMWare y Hyper-V. La respuesta en ese entonces era que se debía ejecutar SQL Server en un servidor físico dedicado y hacerlo funcionar cerca de ellos para comprobar que generara suficiente calor mientras se ejecutaban nuestras cargas de trabajo transaccionales. (I*dem*.).

Otra pregunta que se solía recibir ocasionalmente era si existía la posibilidad de que SQL Server se pudiera ejecutar en entornos Linux, como lo permitía la competencia. En ese momento, la respuesta siempre fue negativa, dado que Microsoft estaba muy asociada con Windows. (I*dem*.).

Sin embargo, la historia ha cambiado y todas estas afirmaciones del pasado han quedado obsoletas. La tecnología de virtualización ha avanzado enormemente y se ha convertido en la base de la mayoría de las implementaciones productivas, incluyendo la nube. Hoy en día, cuando se necesitaba implementar una solución SQL Server en Azure u otra opción de la nube, lo que obtiene es una implementación basada en máquinas virtuales. (I*dem*.).

La llegada de SQL Server 2017 fue una grata sorpresa para todos, ya que por primera vez Microsoft permitió usar esta versión en las distribuciones de Linux más populares, como Redhat, Suse y Ubuntu. Además, el rápido avance de la tecnología de virtualización abrió nuevas posibilidades, como la opción de usar Contenedores SQL Server "Containers". De esta manera, ya no era necesario virtualizar todo el "Host o Servidor", sino que se podía virtualizar únicamente la aplicación, es decir, nuestro motor de base de datos. (I*dem*.).

2. Relación entre C#, Vue y Microsoft SQL Server

C# es un lenguaje de programación utilizado para desarrollar aplicaciones en el entorno de Microsoft .NET. Con este lenguaje, se pueden crear microservicios, que son aplicaciones pequeñas e independientes que trabajan juntas para formar una solución de software más grande. Estos microservicios se comunican entre sí a través de interfaces de programación de aplicaciones (API).

Vue.js es un framework de JavaScript utilizado para construir interfaces de usuario en el lado del cliente. Con este framework, se pueden desarrollar aplicaciones web y aplicaciones móviles híbridas. Las aplicaciones desarrolladas con Vue.js pueden consumir servicios de microservicios a través de APIs para obtener o enviar información.

Microsoft SQL Server es un sistema de gestión de bases de datos relacionales utilizado para almacenar y manipular datos. Los microservicios pueden utilizar Microsoft SQL Server como su base de datos

para almacenar y recuperar información. Además, Vue.js puede utilizar una API de microservicios para interactuar con la base de datos de Microsoft SQL Server.

Los microservicios desarrollados con C# pueden utilizar Microsoft SQL Server como su base de datos y proporcionar una API para que Vue.js pueda consumirlos y mostrar información en una interfaz de usuario. De esta manera, los tres componentes están relacionados y pueden trabajar juntos para formar una solución completa de software.

3. Situaciones y/o problemas

3.1. En Microservicios

- Aplicaciones web escalables: Los microservicios son ideales para aplicaciones web que necesitan escalar horizontalmente, es decir, aumentar su capacidad de procesamiento mediante la adición de más servidores. En lugar de tener una sola aplicación monolítica que se ejecuta en un servidor, se pueden tener múltiples microservicios que se distribuyen en diferentes servidores y se comunican entre sí para realizar una tarea.
- Aplicaciones móviles: Las aplicaciones móviles pueden aprovechar los microservicios para separar la lógica del negocio de la interfaz de usuario. Por ejemplo, se puede tener un microservicio para autenticación de usuarios, otro para la gestión de pagos y otro para la búsqueda de productos.
- Integración de sistemas: Los microservicios pueden ayudar a integrar sistemas de diferentes proveedores y tecnologías. Cada proveedor puede proporcionar un microservicio que exponga su funcionalidad a través de una interfaz bien definida, lo que permite a otros sistemas utilizarlo sin conocer los detalles de su implementación.
- Procesamiento de datos: Los microservicios pueden utilizarse para procesar grandes cantidades de datos. Por ejemplo, se puede tener un microservicio que se encargue de la recopilación de datos, otro que se encargue de su procesamiento y otro que almacene los resultados en una base de datos.
- DevOps: Los microservicios son una herramienta valiosa para las prácticas de DevOps, ya que permiten la entrega y la integración continuas. Cada microservicio puede ser probado y desplegado de manera independiente, lo que facilita la detección de errores y la solución de problemas

3.2. En .NET

- Aplicaciones de escritorio: .NET es una excelente opción para desarrollar aplicaciones de escritorio en Windows. Al utilizar .NET, los desarrolladores pueden utilizar el lenguaje de programación C# para crear aplicaciones de escritorio altamente funcionales e intuitivas.
- Aplicaciones web: .NET también es una excelente opción para desarrollar aplicaciones web. ASP.NET, una plataforma de desarrollo web de Microsoft, es ampliamente utilizada para crear aplicaciones web empresariales escalables y seguras.
- Integración de sistemas: .NET es compatible con muchos lenguajes y plataformas, lo que lo convierte en una excelente opción para integrar sistemas existentes. Las aplicaciones .NET pueden comunicarse con otros sistemas mediante el uso de servicios web y otras tecnologías de comunicación.

- Desarrollo de servicios: .NET también es una excelente opción para desarrollar servicios y APIs que se utilizan en la comunicación entre diferentes sistemas. Las aplicaciones .NET pueden utilizar tecnologías como WCF, Web API, y otras para crear servicios altamente escalables y seguros.
- Desarrollo de bibliotecas de clases: .NET es utilizado también para crear bibliotecas de clases que se pueden utilizar en diferentes aplicaciones. Las bibliotecas de clases .NET pueden ser desarrolladas en diferentes lenguajes de programación y pueden ser compartidas entre diferentes aplicaciones.

3.3. En Vue

- Aplicaciones de una sola página (SPA): Vue es excelente para desarrollar aplicaciones de una sola página (SPA), ya que su enfoque en los componentes reutilizables y la reactividad lo hacen perfecto para la creación de interfaces de usuario altamente interactivas.
- Desarrollo de interfaces de usuario: Vue es muy adecuado para el desarrollo de interfaces de usuario (UI) debido a su capacidad para separar la lógica de presentación de la aplicación en componentes reutilizables. Esto permite una mayor eficiencia en el desarrollo de UI y una mejor mantenibilidad de código.
- Aplicaciones móviles híbridas: Vue es muy popular en el desarrollo de aplicaciones móviles híbridas, ya que proporciona una arquitectura limpia y fácil de usar para el desarrollo de interfaces de usuario interactivas y escalables.
- Integración de aplicaciones: Vue se integra bien con otras bibliotecas y frameworks de JavaScript, lo que lo hace muy adecuado para la creación de aplicaciones altamente personalizadas que requieren la integración con otros sistemas.
- Desarrollo de aplicaciones escalables: Vue se enfoca en la creación de aplicaciones escalables y mantenibles, lo que lo hace perfecto para el desarrollo de aplicaciones empresariales complejas.

3.4. En Microsoft SQL Server

- Aplicaciones empresariales: Microsoft SQL Server se utiliza comúnmente en aplicaciones empresariales para el almacenamiento y gestión de grandes cantidades de datos empresariales. Con su alta escalabilidad y capacidad para manejar grandes conjuntos de datos, Microsoft SQL Server es una excelente opción para aplicaciones empresariales que necesitan un almacenamiento seguro y eficiente de datos.
- Análisis de datos: Microsoft SQL Server se puede utilizar para análisis de datos avanzados, como minería de datos, análisis de series temporales y análisis predictivo. También cuenta con herramientas de Business Intelligence (BI) integradas que permiten a los usuarios analizar los datos en tiempo real y obtener información valiosa para la toma de decisiones empresariales.
- Aplicaciones web: Microsoft SQL Server se puede integrar fácilmente con aplicaciones web para proporcionar un almacenamiento seguro y escalable de datos. Es compatible con diferentes plataformas de desarrollo web, como ASP.NET y PHP, lo que lo hace muy adecuado para el desarrollo de aplicaciones web empresariales y de comercio electrónico.

- Aplicaciones de escritorio: Microsoft SQL Server también se puede utilizar en aplicaciones de escritorio para el almacenamiento y gestión de datos. Esto es especialmente útil para aplicaciones de escritorio empresariales que requieren el acceso a grandes cantidades de datos.
- Sistemas de gestión de contenido: Microsoft SQL Server se puede utilizar en sistemas de gestión de contenido (CMS) para el almacenamiento y gestión de contenido. Algunos ejemplos de CMS que utilizan Microsoft SQL Server son SharePoint y DotNetNuke.

4. Estilos y patrones

4.1. Microservicios

- Arquitectura basada en API: Este estilo se enfoca en la exposición de interfaces de programación de aplicaciones (API) para los microservicios, permitiendo una mayor flexibilidad y desacoplamiento entre los diferentes componentes de la arquitectura.
- Patrón Gateway: En este patrón, se utiliza un servicio de gateway para dirigir las solicitudes de los clientes a los microservicios adecuados. El gateway también puede proporcionar características de seguridad y autenticación.
- Patrón de balanceo de carga: En este patrón, se utiliza un balanceador de carga para distribuir la carga de trabajo entre varias instancias de un mismo microservicio.
- Patrón de descubrimiento de servicios: Este patrón se utiliza para localizar servicios en una arquitectura de microservicios. Un servicio de descubrimiento de servicios mantiene un registro de los servicios disponibles y su ubicación.
- Patrón de agregación de servicios: En este patrón, un servicio de agregación combina varios microservicios para proporcionar una funcionalidad de nivel superior.
- Patrón de mensajería: Este patrón utiliza sistemas de mensajería para comunicarse entre microservicios. Los mensajes se envían de manera asíncrona, lo que reduce la dependencia entre los servicios y mejora la escalabilidad.
- Patrón de contenedorización: En este patrón, se utilizan contenedores para encapsular los microservicios y sus dependencias. Esto permite una mayor portabilidad y escalabilidad de los servicios.

4.2. .NET

- Arquitectura basada en capas: Esta arquitectura organiza el código en diferentes capas, como la capa de presentación, la capa de negocio y la capa de acceso a datos, para proporcionar una separación clara de responsabilidades y facilitar el mantenimiento y la escalabilidad.
- Patrón MVC (Modelo-Vista-Controlador): Este patrón separa la lógica de negocio de la interfaz de usuario, permitiendo una mayor modularidad y reutilización de código.
- Patrón Repositorio: Este patrón separa la lógica de acceso a datos del resto de la aplicación, proporcionando una capa de abstracción que facilita el cambio de la fuente de datos sin afectar el resto de la aplicación.
- Patrón Inyección de Dependencias: Este patrón permite la creación de componentes de manera más flexible y mantenible, separando la creación de objetos de su uso y permitiendo un mayor modularidad y testabilidad.

- Patrón Unit of Work: Este patrón se utiliza para manejar transacciones y cambios de estado en la base de datos, proporcionando una capa de abstracción sobre el contexto de la base de datos y simplificando el código de acceso a datos.
- Patrón de mensajes: Este patrón utiliza un sistema de mensajería para comunicar diferentes componentes de la aplicación, permitiendo una mayor escalabilidad y desacoplamiento entre los diferentes componentes.
- Patrón de microservicios: Este patrón arquitectónico se basa en la creación de servicios pequeños y autónomos que se comunican entre sí a través de API, permitiendo una mayor flexibilidad y escalabilidad en el desarrollo de aplicaciones.

4.3. Vue

- Patrón de componente: Este patrón se basa en la creación de componentes reutilizables que se pueden utilizar en diferentes partes de la aplicación. Los componentes pueden encapsular lógica de presentación y de interacción, y facilitan la modularidad y el mantenimiento del código.
- Patrón de estado centralizado: Este patrón utiliza una sola fuente de verdad para el estado de la aplicación, lo que facilita la gestión y el acceso al estado de la aplicación. Vue ofrece Vuex, una biblioteca para la gestión del estado de la aplicación que implementa este patrón.
- Patrón de eventos y emisiones: Este patrón utiliza eventos y emisiones para comunicar diferentes componentes de la aplicación, lo que facilita el acoplamiento entre los componentes y mejora la reutilización de código.
- Patrón de inyección de dependencias: Este patrón permite la creación de componentes de manera más flexible y mantenible, separando la creación de objetos de su uso y permitiendo un mayor modularidad y testabilidad.
- Patrón de renderizado condicional: Este patrón se utiliza para renderizar componentes de manera condicional en función de una serie de condiciones. Vue proporciona directivas para el renderizado condicional, como v-if y v-show, que simplifican la implementación de este patrón.
- Patrón de renderizado dinámico: Este patrón se utiliza para renderizar componentes de manera dinámica en función de una serie de datos. Vue proporciona la directiva v-for para la implementación de este patrón.
- Patrón de animaciones: Este patrón se utiliza para agregar animaciones a la interfaz de usuario de la aplicación. Vue proporciona la biblioteca de animaciones anim.js para la implementación de este patrón.

5. Ventajas y Desventajas

5.1. Ventajas y Desventajas de un sistema distribuido

Un sistema distribuido es un conjunto de computadoras interconectadas y coordinadas entre sí para trabajar juntas como si fueran una sola. Hay varias ventajas y desventajas asociadas con la implementación de un sistema distribuido:

5.1.1. Ventajas

- Escalabilidad: Un sistema distribuido puede ser escalado fácilmente agregando más nodos al sistema para manejar un mayor volumen de trabajo.
- Disponibilidad: Si un nodo falla, otros nodos pueden continuar trabajando sin interrupción.
- Rendimiento: Un sistema distribuido puede realizar tareas en paralelo, lo que permite una mayor velocidad de procesamiento.
- Tolerancia a fallos: Los sistemas distribuidos están diseñados para continuar funcionando incluso si uno o varios nodos fallan.
- Distribución geográfica: Los nodos pueden estar ubicados en diferentes lugares, lo que permite el acceso remoto a los recursos del sistema.
- Procesadores más poderosos y a menos costos.
- Desarrollo de Estaciones con más capacidades.
- Avances en la Tecnología de Comunicaciones.
- Disponibilidad de elementos de Comunicación.
- Desarrollo de nuevas técnicas.
- Compartición de Recursos.
- Respuesta Rápida.
- Ejecución Concurrente de procesos (En varias computadoras).
- Empleo de técnicas de procesamiento distribuido
- Disponibilidad y Confiabilidad.
- Mayores servicios que elevan la funcionalidad (Monitoreo, Telecontrol, Correo Eléctrico, Etc.).
- Crecimiento Modular.
- Es inherente al crecimiento.
- Inclusión rápida de nuevos recursos.
- Los recursos actuales no afectan.

5.1.2. Desventajas

- Complejidad: La coordinación de los nodos en un sistema distribuido puede ser compleja, lo que aumenta la complejidad del diseño y la implementación.
- Comunicación: La comunicación entre los nodos puede ser lenta y costosa, lo que puede afectar el rendimiento del sistema.
- Seguridad: La seguridad en un sistema distribuido puede ser difícil de lograr, ya que hay múltiples puntos de acceso.
- Mantenimiento: El mantenimiento de un sistema distribuido puede ser más complicado debido a la naturaleza distribuida del sistema.
- Fallos: La presencia de múltiples nodos puede aumentar la probabilidad de fallas en el sistema, lo que puede ser difícil de detectar y corregir.

- Complejidad de programación: La programación de aplicaciones para sistemas distribuidos puede ser compleja debido a la necesidad de coordinar los nodos y la comunicación entre ellos.
- Requerimientos de mayores controles de procesamiento.
- Administración más compleja.
- Costos.

(*Ventajas y Desventajas de un Sistema Distribuido - SISTEMAS DISTRIBUIDOS & CLUSTERS*, s. f.)

5.2. Ventajas y Desventajas de un Vue

Vue.js es un framework de JavaScript para construir aplicaciones web de una sola página. A continuación, se presentan algunas de sus ventajas y desventajas:

5.2.1. Ventajas

- Es fácil de aprender y de utilizar, incluso para aquellos que no tienen mucha experiencia en programación.
- Tiene una documentación muy detallada y una gran comunidad de desarrolladores que ofrecen soporte y tutoriales en línea.
- Ofrece una arquitectura basada en componentes, lo que hace que sea fácil de mantener y escalar.
- Proporciona una excelente experiencia de usuario gracias a su capacidad para renderizar cambios en tiempo real.
- Se integra fácilmente con otras bibliotecas y marcos de trabajo, lo que lo hace muy versátil.
- Es muy ligero y rápido, lo que se traduce en una mejor experiencia del usuario.
- Su pequeño tamaño: puede que esto no sea una gran característica, pero los 18 KB que pesa este framework lo hacen ideal para una descarga rápida y poder almacenarlo en equipos de baja memoria, impactando de manera positiva tu SEO y UX.
- Desglosar los componentes en archivos individuales: cuando se crea una aplicación o página web con VueJS, cada pieza de esta se divide en componentes individuales, representados como elementos encapsulados en su interfaz. Estos componentes se pueden escribir en HTML, CCS y JavaScript, siendo esta otra de las ventajas.
- Esta característica es ideal cuando se realicen pruebas unitarias para comprobar si funcionan por sí solas hasta las partes más pequeñas de la aplicación o la página web a crear.
- Fácil de aprender y utilizar: VueJS es uno de los frameworks más amigables a la hora de ser utilizado por desarrolladores que van comenzando o personas que tienen entusiasmo por aprender lo básico de este campo. Cuando se comienza a codificar no es necesario conocer JSX y TypeScript, elementos que sí se utilizan en otras tecnologías de front-end.

5.2.2. Desventajas

- Aunque es fácil de aprender, Vue.js puede ser difícil de dominar debido a su enfoque en la programación reactiva.
- A pesar de que tiene una comunidad muy activa, no es tan grande como la de algunos de sus competidores, como React o Angular.
- No es tan maduro como algunos otros frameworks, lo que puede hacer que sea menos estable en algunas situaciones.
- Al ser una tecnología relativamente nueva, es posible que algunas empresas no estén familiarizadas con ella o no estén dispuestas a adoptarla.
- La dependencia de los componentes puede aumentar la complejidad de las aplicaciones y generar un código más difícil de mantener.
- La barrera del lenguaje: el masivo uso de VueJS por parte de empresas chinas como Xiaomi y AliBaba ha motivado a que los programadores de dicho país se conviertan en expertos en este framework y publiquen mucha información en la web sobre el mismo, estando la mayoría escrita en chino. A la hora de acceder a ella desde occidente, el hecho de no saber hablar este idioma puede ser un impedimento para conocer sobre actualizaciones y mejoras.
- No cuenta con apoyo para grandes proyectos: debido a ser un framework relativamente joven, VueJS no cuenta con el apoyo de un equipo tan extenso como otros en el campo, por ejemplo Angular. Esto lo aleja de los grandes proyectos y por lo general se le utiliza para tareas de menor envergadura, a pesar de estar presente en empresas como IBM y Adobe.
- También, por ser un framework nuevo y con pocos años en el mercado, hay pocos desarrolladores expertos en VueJS, llevando esto a ser uno de los menos utilizados en el mercado laboral y, por su parte, generando poco interés a la hora de aprender sobre él. (s. f.)

5.3. Ventajas y Desventajas .NET

.NET es un framework de desarrollo de software que permite la creación de aplicaciones para Windows, web y dispositivos móviles. A continuación, se describen algunas de las ventajas y desventajas de utilizar .NET:

5.3.1. Ventajas

- Amplia comunidad de desarrolladores: La comunidad de desarrolladores de .NET es grande y activa, lo que significa que hay una gran cantidad de recursos disponibles en línea para aprender y solucionar problemas.
- Multiplataforma: .NET es multiplataforma, lo que significa que puedes desarrollar aplicaciones para Windows, macOS y Linux. También puedes utilizar .NET para desarrollar aplicaciones móviles para Android y iOS utilizando Xamarin.

- Gran biblioteca de clases: .NET cuenta con una gran biblioteca de clases que se pueden utilizar para la creación de aplicaciones. Esto significa que hay una gran cantidad de código reutilizable disponible que puede acelerar el proceso de desarrollo.
- Lenguajes de programación: .NET soporta una variedad de lenguajes de programación, como C#, VB.NET, F# y más. Esto permite a los desarrolladores trabajar con el lenguaje que mejor se adapte a sus necesidades.
- Seguridad: .NET tiene características de seguridad integradas, como el control de acceso, la autenticación y la encriptación, lo que permite crear aplicaciones seguras y proteger los datos del usuario.

5.3.2. Desventajas

- Curva de aprendizaje: .NET puede tener una curva de aprendizaje empinada para los desarrolladores que no están familiarizados con él. Los desarrolladores nuevos en .NET pueden tener que pasar tiempo aprendiendo las características y funcionalidades del framework.
- Licenciamiento: Aunque .NET es un framework gratuito y de código abierto, algunas de sus características, como la edición empresarial, requieren una licencia.
- Rendimiento: Aunque .NET es rápido y eficiente, puede ser más lento que otros frameworks de programación como C++ o Rust. Esto se debe a que .NET utiliza una máquina virtual para ejecutar el código.
- Dependencia de Microsoft: .NET es desarrollado y mantenido por Microsoft, lo que significa que existe una dependencia en la empresa para mantener y actualizar el framework. Esto puede ser una desventaja para aquellos que prefieren trabajar con frameworks independientes.
- Tamaño de la aplicación: Debido a la gran biblioteca de clases que .NET tiene disponible, las aplicaciones pueden tener un tamaño considerablemente grande. Esto puede ser un problema para aplicaciones móviles que tienen limitaciones de almacenamiento.

(*Human Verification*, s. f.)

5.4. Ventajas y Desventajas de un Microsoft SQL Server

5.4.1. Ventajas

- Es un sistema de gestión de base de datos.
- Es útil para manejar y obtener datos de la red de redes.
- Nos permite olvidarnos de los ficheros que forman la base de datos.
- Si trabajamos en una red social nos permite agregar otros servidores de SQL Server. Por ejemplo, dos personas que trabajan con SQL Server, uno de ellos se puede conectar al servidor de su otro compañero y así se puede ver las bases de datos del otro compañero con SQL Server.

- SQL permite administrar permisos a todo. También permite que alguien conecte su SQLO al nuestro pero sin embargo podemos decirle que no puede ver esta base de datos pero otro sí.

5.4.2. Desventajas

- Utiliza mucho la memoria RAM para las instalaciones y utilización de software.
- No se puede utilizar como practicas porque se prohíben muchas cosas, tiene restricciones en lo particular.
- La relación, calidad y el precio está muy debajo comparado con oracle.
- Tiene muchos bloqueos a nivel de página, un tamaño de página fijo y demasiado pequeño, una pésima implementación de los tipos de datos variables.

(*Ventajas y desventajas*, s. f.)

6. Principios SOLID

6.1. SOLID en microservicios

Los principios SOLID son un conjunto de principios de diseño orientados a objetos que se utilizan para crear sistemas de software más mantenibles, escalables y fáciles de entender. Aunque estos principios fueron desarrollados originalmente para sistemas orientados a objetos, pueden aplicarse a sistemas distribuidos para mejorar su diseño y rendimiento. A continuación, se describen algunos de los principios SOLID y cómo se aplican a sistemas distribuidos:

- Principio de responsabilidad única (SRP): Este principio establece que cada clase debe tener una única responsabilidad, lo que significa que debe tener solo una razón para cambiar. En sistemas distribuidos, esto puede aplicarse a cada componente o microservicio que forma parte del sistema. Cada componente debe tener una única responsabilidad y ser independiente de los demás componentes. Esto facilitará la escalabilidad del sistema y su mantenimiento a medida que se agreguen nuevos componentes o servicios.
- Principio de abierto/cerrado (OCP): Este principio establece que las clases deben estar abiertas para la extensión, pero cerradas para la modificación. En sistemas distribuidos, esto se puede aplicar a la arquitectura del sistema. El sistema debe estar diseñado de tal manera que pueda adaptarse a nuevos requisitos o funcionalidades sin necesidad de realizar cambios importantes en el código fuente existente. Esto se puede lograr mediante el uso de interfaces y abstracciones en lugar de implementaciones concretas.
- Principio de sustitución de Liskov (LSP): Este principio establece que las clases derivadas deben ser sustituibles por sus clases base. En sistemas distribuidos, esto se puede aplicar a la interacción entre diferentes componentes o servicios.

Cada componente o servicio debe ser diseñado de tal manera que pueda ser sustituido por otro componente o servicio con funcionalidades similares sin causar problemas en el resto del sistema.

- Principio de segregación de interfaces (ISP): Este principio establece que las interfaces deben ser lo suficientemente pequeñas y específicas para las necesidades de sus clientes. En sistemas distribuidos, esto se puede aplicar a las API o interfaces que utilizan diferentes componentes o servicios. Cada componente o servicio debe exponer solo las interfaces necesarias para su uso, en lugar de exponer una interfaz monolítica que puede contener funcionalidades innecesarias.
- Principio de inversión de dependencia (DIP): Este principio establece que las clases de nivel superior no deben depender de las clases de nivel inferior, sino de las abstracciones. En sistemas distribuidos, esto se puede aplicar a la arquitectura del sistema y la forma en que los diferentes componentes interactúan entre sí. Cada componente debe depender de una abstracción en lugar de una implementación concreta, lo que facilitará el reemplazo o la actualización de diferentes componentes del sistema.

6.2. SOLID en Vue

Los sistemas distribuidos tienen requisitos de calidad específicos que deben ser considerados para garantizar su correcto funcionamiento. A continuación, se describen algunos de los atributos de calidad clave para los sistemas distribuidos:

- Escalabilidad: La escalabilidad es la capacidad de un sistema para manejar un mayor volumen de trabajo a medida que aumenta la cantidad de usuarios o la carga de trabajo. Los sistemas distribuidos deben ser escalables para garantizar que puedan manejar un aumento en el número de usuarios o la cantidad de datos procesados sin afectar su rendimiento o disponibilidad.
- Disponibilidad: La disponibilidad se refiere a la capacidad de un sistema para estar disponible y accesible para los usuarios en todo momento. Los sistemas distribuidos deben garantizar una alta disponibilidad para evitar interrupciones en el servicio y garantizar que los usuarios puedan acceder a los datos y servicios en todo momento.
- Rendimiento: El rendimiento se refiere a la capacidad de un sistema para procesar y responder rápidamente a las solicitudes de los usuarios. Los sistemas distribuidos deben ser diseñados para garantizar un alto rendimiento y minimizar la latencia de red y la sobrecarga de procesamiento.
- Fiabilidad: La fiabilidad se refiere a la capacidad de un sistema para mantener la integridad y consistencia de los datos y servicios en caso de fallos o errores en el sistema. Los sistemas distribuidos deben ser diseñados para garantizar la

fiabilidad de los datos y servicios, lo que incluye la implementación de mecanismos de recuperación y tolerancia a fallos.

- Seguridad: La seguridad es fundamental en los sistemas distribuidos, ya que deben garantizar la protección de los datos y servicios contra posibles amenazas externas. Los sistemas distribuidos deben incluir mecanismos de autenticación, autorización y encriptación de datos para garantizar la seguridad y privacidad de los datos y servicios.
- Mantenibilidad: La mantenibilidad se refiere a la facilidad con la que se puede mantener y actualizar un sistema a lo largo del tiempo. Los sistemas distribuidos deben ser diseñados para facilitar el mantenimiento y la actualización, lo que incluye la documentación detallada, el uso de patrones de diseño y la modularidad de los componentes.

6.3. SOLID en .NET

Los principios SOLID son un conjunto de reglas de diseño de software que ayudan a los desarrolladores a crear sistemas más mantenibles y escalables. A continuación, se describen cómo se aplican estos principios en el desarrollo con .NET:

- Principio de Responsabilidad Única (SRP): Este principio establece que una clase debe tener una sola responsabilidad. En .NET, esto se traduce en dividir las funcionalidades de una clase en múltiples clases más pequeñas y especializadas.
- Principio de Abierto/Cerrado (OCP): Este principio establece que una clase debe estar abierta para su extensión, pero cerrada para su modificación. En .NET, esto se puede lograr utilizando interfaces y clases abstractas que permitan la extensión de una clase sin tener que modificar su código fuente.
- Principio de Sustitución de Liskov (LSP): Este principio establece que las subclases deben ser sustituibles por sus clases base. En .NET, esto significa que las clases derivadas deben poder ser utilizadas en lugar de las clases base sin alterar el comportamiento del sistema.
- Principio de Segregación de Interfaz (ISP): Este principio establece que una clase no debe ser forzada a implementar interfaces que no necesita. En .NET, esto significa crear interfaces específicas para cada funcionalidad que una clase necesita, en lugar de una sola interfaz grande que obligue a implementar funcionalidades innecesarias.
- Principio de Inversión de Dependencia (DIP): Este principio establece que los módulos de alto nivel no deben depender de los módulos de bajo nivel, sino de abstracciones. En .NET, esto significa que las clases de alto nivel deben depender de interfaces y no de implementaciones concretas, lo que permite la flexibilidad y el intercambio de implementaciones en tiempo de ejecución.

En resumen, la aplicación de los principios SOLID en .NET puede ayudar a los desarrolladores a crear sistemas más mantenibles, escalables y flexibles.

6.4. SOLID en SQL Server

Los principios SOLID son un conjunto de principios de diseño de software que se utilizan para crear sistemas de software escalables, flexibles y mantenibles. Aunque estos principios fueron originalmente desarrollados para la programación orientada a objetos, muchos de ellos también se aplican a otros tipos de sistemas, como los sistemas de gestión de bases de datos. A continuación, se detallan cómo se pueden aplicar los principios SOLID en SQL Server:

- Principio de Responsabilidad Única (SRP): Este principio establece que una clase debe tener solo una razón para cambiar. En el caso de SQL Server, esto significa que cada tabla debe tener una única responsabilidad y propósito. Si una tabla tiene múltiples responsabilidades, es probable que se vuelva difícil de mantener y modificar.
- Principio de Abierto/Cerrado (OCP): Este principio establece que una clase debe ser abierta para la extensión, pero cerrada para la modificación. En el caso de SQL Server, esto significa que los cambios en la estructura de la base de datos deben hacerse a través de la adición de nuevas tablas o la modificación de las existentes, sin afectar las aplicaciones que utilizan la base de datos.
- Principio de Sustitución de Liskov (LSP): Este principio establece que una clase derivada debe ser sustituible por su clase base sin cambiar el comportamiento del sistema. En el caso de SQL Server, esto significa que las

7. Atributos de Calidad

7.1. Atributos de Calidad para microservicios

Los sistemas distribuidos tienen requisitos de calidad específicos que deben ser considerados para garantizar su correcto funcionamiento. A continuación, se describen algunos de los atributos de calidad clave para los sistemas distribuidos:

- Escalabilidad: La escalabilidad es la capacidad de un sistema para manejar un mayor volumen de trabajo a medida que aumenta la cantidad de usuarios o la carga de trabajo. Los sistemas distribuidos deben ser escalables para garantizar que puedan manejar un aumento en el número de usuarios o la cantidad de datos procesados sin afectar su rendimiento o disponibilidad.
- Disponibilidad: La disponibilidad se refiere a la capacidad de un sistema para estar disponible y accesible para los usuarios en todo momento. Los sistemas distribuidos deben garantizar una alta disponibilidad para evitar interrupciones en el servicio y garantizar que los usuarios puedan acceder a los datos y servicios en todo momento.

- Rendimiento: El rendimiento se refiere a la capacidad de un sistema para procesar y responder rápidamente a las solicitudes de los usuarios. Los sistemas distribuidos deben ser diseñados para garantizar un alto rendimiento y minimizar la latencia de red y la sobrecarga de procesamiento.
- Fiabilidad: La fiabilidad se refiere a la capacidad de un sistema para mantener la integridad y consistencia de los datos y servicios en caso de fallos o errores en el sistema. Los sistemas distribuidos deben ser diseñados para garantizar la fiabilidad de los datos y servicios, lo que incluye la implementación de mecanismos de recuperación y tolerancia a fallos.
- Seguridad: La seguridad es fundamental en los sistemas distribuidos, ya que deben garantizar la protección de los datos y servicios contra posibles amenazas externas. Los sistemas distribuidos deben incluir mecanismos de autenticación, autorización y encriptación de datos para garantizar la seguridad y privacidad de los datos y servicios.
- Mantenibilidad: La mantenibilidad se refiere a la facilidad con la que se puede mantener y actualizar un sistema a lo largo del tiempo. Los sistemas distribuidos deben ser diseñados para facilitar el mantenimiento y la actualización, lo que incluye la documentación detallada, el uso de patrones de diseño y el modularidad de los componentes.

7.2. Atributos de Calidad para Vue

Vue.js es un popular framework de JavaScript utilizado para la creación de interfaces de usuario y aplicaciones web. Los atributos de calidad de Vue.js son los siguientes:

- Usabilidad: Vue.js es conocido por su facilidad de uso y simplicidad de aprendizaje, lo que lo hace muy amigable para los desarrolladores principiantes. Además, Vue.js ofrece una excelente experiencia de usuario gracias a su capacidad de crear interfaces de usuario interactivas y altamente responsivas.
- Mantenibilidad: Con Vue.js, es fácil mantener y actualizar el código debido a su estructura modular y componentes reutilizables. Además, su arquitectura basada en componentes facilita la identificación y corrección de errores.
- Rendimiento: Vue.js es muy eficiente en cuanto a la carga y la velocidad de ejecución de las aplicaciones. Utiliza una arquitectura de renderizado basada en componentes, lo que permite una carga más rápida y una menor sobrecarga de recursos.
- Flexibilidad: Vue.js es altamente personalizable, lo que permite a los desarrolladores ajustar el comportamiento de la aplicación de acuerdo a sus necesidades específicas. Además, Vue.js puede ser fácilmente integrado con otras bibliotecas y frameworks.
- Comunidad: Vue.js tiene una comunidad de desarrolladores activa y comprometida, lo que garantiza el soporte y la disponibilidad de recursos en línea, como documentación, ejemplos de código y tutoriales.

En resumen, los atributos de calidad de Vue.js incluyen su usabilidad, mantenibilidad, rendimiento, flexibilidad y comunidad. Estos atributos hacen que Vue.js sea una excelente opción para el desarrollo de aplicaciones web modernas y altamente interactivas.

7.3. Atributos de Calidad para .NET

.NET es un framework de desarrollo de software desarrollado por Microsoft que se utiliza principalmente para la creación de aplicaciones de escritorio, aplicaciones web y servicios web. Algunos de los atributos de calidad de .NET son:

- Rendimiento: .NET es conocido por su alto rendimiento y velocidad de ejecución de aplicaciones. Utiliza la compilación Just-In-Time (JIT) para generar código de máquina nativo en tiempo de ejecución, lo que mejora el rendimiento en comparación con otros frameworks.
- Seguridad: .NET ofrece un alto nivel de seguridad para las aplicaciones. Incluye características de seguridad integradas, como la autenticación y la autorización, así como herramientas para el cifrado de datos y la prevención de ataques de seguridad.
- Escalabilidad: Las aplicaciones desarrolladas con .NET son escalables y pueden manejar grandes cantidades de tráfico. Además, .NET admite la creación de aplicaciones distribuidas y de alta disponibilidad.
- Mantenibilidad: .NET es un framework altamente estructurado y modular, lo que hace que el mantenimiento y la actualización de las aplicaciones sean más fáciles. También tiene una gran comunidad de desarrolladores y recursos en línea disponibles.
- Interoperabilidad: .NET es compatible con varios lenguajes de programación y sistemas operativos, lo que permite la integración de aplicaciones y sistemas existentes.
- Productividad: .NET es conocido por su facilidad de uso y productividad. Incluye características como IntelliSense, depuración y herramientas de prueba integradas, que ayudan a los desarrolladores a escribir código de manera más rápida y efectiva.

En resumen, los atributos de calidad de .NET incluyen su rendimiento, seguridad, escalabilidad, mantenibilidad, interoperabilidad y productividad. Estos atributos lo hacen un framework popular y eficiente para el desarrollo de aplicaciones de software.

7.4. Atributos de Calidad para SQLServer

Microsoft SQL Server es un sistema de gestión de bases de datos relacionales que ofrece una amplia variedad de atributos de calidad que lo hacen una opción confiable y de alta calidad para el almacenamiento y gestión de datos. Algunos de estos atributos incluyen:

- Confiabilidad: SQL Server ofrece una alta disponibilidad y confiabilidad, con características como la replicación de datos, la recuperación ante desastres y la capacidad de realizar copias de seguridad y restauración de datos.
- Rendimiento: SQL Server está diseñado para ofrecer un alto rendimiento, permitiendo una rápida respuesta a las consultas y transacciones de la base de datos.
- Escalabilidad: SQL Server es altamente escalable, lo que significa que puede manejar grandes volúmenes de datos y un alto tráfico de usuarios.
- Seguridad: SQL Server ofrece una amplia variedad de características de seguridad para proteger los datos almacenados, incluyendo autenticación y autorización, encriptación de datos y seguimiento de auditoría.
- Integración: SQL Server está diseñado para integrarse con otros productos y tecnologías de Microsoft, lo que permite una mayor flexibilidad y una mejor gestión de los datos.
- Gestión de recursos: SQL Server ofrece una eficiente gestión de recursos, lo que permite una mejor gestión de la memoria, el almacenamiento y el procesamiento de datos.
- Administración y monitorización: SQL Server proporciona herramientas de administración y monitorización para facilitar la gestión de la base de datos, incluyendo el uso de alertas y notificaciones para detectar problemas y tomar medidas preventivas.

8. Casos de estudios relevantes:

8. Casos de estudios relevantes

8.1. Microservicios

- La arquitectura de Netflix está compuesta por más de 500 microservicios, y cuentan con más de 50 millones de suscriptores que realizan unos 2.000 millones de peticiones al día. Plaza (s. f.)

Referencias:

Ventajas y Desventajas de un Sistema Distribuido - SISTEMAS DISTRIBUIDOS & CLUSTERS. (s. f.). https://sites.google.com/site/sdistribuidoscluster/ventajas-y-desventajas-de-un-sistema-distribuido

R. (s. f.). VueJS: Ventajas y desventajas de este framework. Rootstack. https://rootstack.com/es/blog/vuejs-ventajas-desventajas

Ventajas y desventajas. (s. f.). SQL Server. https://sqlserver4b.weebly.com/ventajas-y-desventajas.html

Human Verification. (s. f.). https://www.cisin.com/coffee-break/es/enterprise/the-good-and-the-bad-of-net-framework-development.html

Plaza, S. B. (s. f.). ECOSISTEMA DE PROYECTOS NETFLIX PARA MICROSERVICIOS. prezi.com. https://prezi.com/p/ydozozb46-wg/ecosistema-de-proyectos-netflix-para-microservicios/