**Actividad: Investigación de arquitecturas**

Investigar y desarrollar la siguiente actividad sobre diferentes arquitecturas de software, donde deberás encontrar y escribir sus definiciones, empresas que las utilizan, y casos de estudio relevantes.

Una vez investigado comentar con tus compañeros.

**Completar lo siguiente:**

* ¿Qué es?: Definición y explicación con lo que entiendes de la arquitectura.
* Beneficios y desafíos enfrentados (ventajas y desventajas).
* Empresas que la utilizan: Ejemplos de empresas que han adoptado esta arquitectura o técnica y cómo la implementan.

**Temas a Investigar:**

1. Arquitectura Monolítica

Los sistemas monolíticos son aplicaciones diseñadas como una única unidad indivisible, donde todos los componentes—como la interfaz de usuario, la lógica de negocio y la capa de acceso a datos—están integrados y funcionan como una entidad única. Esta arquitectura contrasta con enfoques más modulares o basados en microservicios, donde los componentes son más independientes y pueden operar de manera autónoma.

Ventajas:

* Simplicidad: Las arquitecturas monolíticas suelen ser más fáciles de desarrollar inicialmente debido a su estructura sencilla.
* Rendimiento: La comunicación interna entre procesos es más rápida, ya que todos los componentes comparten el mismo espacio de memoria.
* Facilidad de prueba: La integración de todos los componentes permite realizar pruebas end-to-end de manera más directa.

Desventajas:

* Desafíos de escalabilidad: Escalar componentes específicos de manera independiente es complejo, ya que toda la aplicación debe escalarse conjuntamente.
* Complejidad en el mantenimiento: A medida que el código base crece, entender y modificar el sistema puede volverse más difícil.

Limitaciones en el despliegue: Cualquier cambio requiere el redespliegue completo de la aplicación, lo que puede ser ineficiente y arriesgado.

Empresas que lo Utilizan:

* GitHub: Mucha de sus funcionalidades principales utilizan este tipo de arquitectura.
* Atlassian: Sus aplicaciones como Jira y Confluence fueron construidos en este tipo de arquitectura.

1. Arquitectura de Capas (N-tier)

La Arquitectura de Capas o Arquitectura de estilo N-Tier es un patrón de arquitectura de software cliente-servidor donde diferentes funciones, como la presentación, el procesamiento de aplicaciones y la gestión de datos, se separan físicamente en capas o niveles distintos. Esta separación mejora el modularidad, la escalabilidad y la facilidad de mantenimiento, al permitir a los desarrolladores modificar o añadir niveles específicos sin tener que rediseñar toda la aplicación.

Ventajas:

* Separación de responsabilidades: Cada capa gestiona una responsabilidad específica, lo que hace que el sistema sea modular y más fácil de mantener.
* Escalabilidad: Las capas se pueden escalar de forma independiente, lo que permite una mejor distribución de recursos.
* Flexibilidad y seguridad: Se pueden utilizar diferentes tecnologías para cada capa y la información confidencial se puede proteger mejor aislando las capas.

Desventajas:

* Complejidad: La gestión de varias capas puede aumentar la complejidad del sistema y la sobrecarga de coordinación.
* Sobrecarga de rendimiento: Una mayor comunicación entre capas puede generar latencia y afectar al rendimiento.
* Mayor coste y pruebas: Mayores costes de desarrollo y operativos, y pruebas más complejas debido a las interacciones entre capas.

1. Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (SOA) es un patrón de diseño donde el software se estructura como un conjunto de servicios reutilizables y poco acoplados que se comunican a través de una red. Estos servicios suelen gestionar funciones empresariales más amplias y pueden ser accedidos por múltiples aplicaciones. La SOA suele utilizar un Bus de Servicios Empresariales (ESB) para la comunicación e integración entre servicios.

Ventajas:

* Reutilización: Los servicios se comparten y pueden reutilizarse en diferentes aplicaciones, lo que reduce la redundancia.
* Escalabilidad: Los servicios pueden escalarse de forma independiente según la demanda.
* Integración: Facilita la integración con diversos sistemas y tecnologías dentro de una organización.

Desventajas:

* Complejidad: Requiere una infraestructura compleja, especialmente al utilizar un ESB, lo que puede aumentar la sobrecarga.
* Sobrecarga de Rendimiento: La comunicación entre servicios a través de un ESB o protocolos pesados ​​como SOAP puede causar latencia.
* Consistencia de Datos: Gestionar la consistencia entre servicios y sistemas puede ser difícil debido a la gestión centralizada de datos.

1. Arquitectura de Microservicios

La arquitectura de microservicios es un estilo en el que una aplicación se construye como un conjunto de pequeños servicios que se implementan de forma independiente, cada uno centrado en una capacidad empresarial específica. Estos servicios están débilmente acoplados y se comunican mediante protocolos ligeros como HTTP/REST. Los microservicios promueven la flexibilidad, la escalabilidad y la autonomía de los equipos de desarrollo.

Ventajas:

* Independencia: Cada servicio puede desarrollarse, implementarse y escalarse de forma independiente, lo que ofrece flexibilidad y resiliencia.
* Escalabilidad: Los servicios pueden escalarse individualmente según la demanda, optimizando así los recursos.
* Resiliencia: El fallo de un servicio no afecta a todo el sistema, lo que mejora su fiabilidad.

Desventajas:

* Complejidad: La gestión y coordinación de muchos servicios independientes puede generar complejidad en el sistema.
* Gestión de datos: Cada servicio puede tener su propia base de datos, lo que dificulta la consistencia y la gestión de los datos.
* Infraestructura: Requiere una infraestructura más sofisticada, que incluye el descubrimiento de servicios, la monitorización y la comunicación entre servicios.

1. Arquitectura Multitenant

La arquitectura Multitenant es un modelo de diseño de software donde una sola instancia de una aplicación sirve a múltiples clientes, conocidos como inquilinos. Los datos y las configuraciones de cada inquilino están aislados, lo que garantiza la privacidad y la personalización dentro de un entorno compartido.

Ventajas:

* Rentabilidad: Al compartir recursos, la arquitectura Multitenant reduce los costos de infraestructura y mantenimiento, ya que no se necesitan instancias separadas para cada inquilino.
* Actualizaciones simplificadas: Con una sola instancia de aplicación, las actualizaciones y el mantenimiento se agilizan, ya que los cambios se aplican universalmente sin necesidad de múltiples implementaciones.

Desventajas:

* Problemas de rendimiento: Los recursos compartidos pueden generar problemas de rendimiento si el uso de un inquilino afecta a otros, lo que requiere una gestión cuidadosa de los recursos.
* Seguridad de los datos: Garantizar el aislamiento y la seguridad de los datos es crucial, ya que las vulnerabilidades podrían exponer los datos de un inquilino a otros.
* Limitaciones de personalización: Si bien la personalización es posible, puede ser limitada en comparación con las instancias dedicadas, lo que podría afectar los requisitos específicos de cada inquilino.

1. Arquitectura Event-Driven
2. Arquitectura Micro Frontends

**Recomendaciones para la Investigación:**

* Uso de Herramientas de Colaboración: Utilizar plataformas como Google Docs, Miro y Trello para coordinar y compartir información.
* Asignación de Tareas: Dividir los temas entre los miembros del grupo de manera equitativa.

**Desarrollo:**

**Arquitectura Monolítica**

**Definición:**

La arquitectura monolítica es un modelo de diseño de software donde todos los componentes y funcionalidades de una aplicación se integran en un único bloque o unidad. Este enfoque centraliza el desarrollo, despliegue y mantenimiento del software.

**Explicación con tus palabras:**

Es como tener todo el código de una aplicación en un solo proyecto grande. Cada funcionalidad, desde el manejo de usuarios hasta el procesamiento de pagos, está en un solo lugar.

**Beneficios**:

* Desarrollo y despliegue sencillos.
* Facilidad para la depuración y las pruebas.
* Menor latencia de comunicación entre componentes.

**Desafíos**:

* Dificultad para escalar horizontalmente.
* Mantenimiento complejo a medida que la aplicación crece.
* Implementación de cambios y actualizaciones más riesgosa.

**Empresas que la Utilizan**:

* Algunas startups y pequeñas empresas que buscan una solución rápida y sencilla.
* Aplicaciones tradicionales y heredadas de empresas más grandes.

🤖🦝🤖