Orquestación y Interoperabilidad

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación

CONCEPTOS

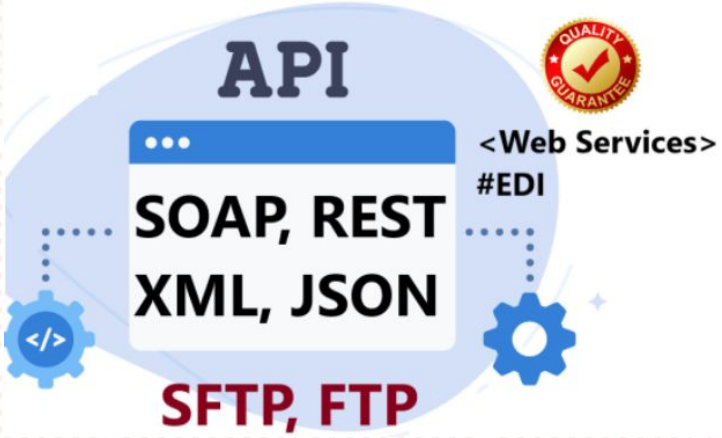
- Interoperabilidad
- Arquitectura de 3 capas
  - Capa de presentación
  - Capa de Lógica
  - Capa de Datos
- Microservicios

INTEROPERABILIDAD

En el contexto de la orquestación, la interoperabilidad es esencial para asegurar que los diferentes componentes y servicios de software se comuniquen correctamente y se integren sin problemas. Por ejemplo, si se están orquestando diferentes servicios de microservicios, es importante que cada servicio pueda comunicarse con los otros de manera efectiva para lograr una orquestación adecuada.

INTEROPERABILIDAD

Los estándares y protocolos de interoperabilidad, como REST, SOAP, JSON y XML, son importantes en la orquestación para garantizar que los diferentes componentes y servicios de software puedan comunicarse de manera efectiva y sin problemas.



Retrieved from
https://www.fiverr.com/kalimraza/integrate-any-third-party-api-restful-or-soap-based

INTEROPERABILIDAD

REST: Representational State Transfer (REST) es un estilo de arquitectura de software que se utiliza para crear servicios web que sean escalables, flexibles y fáciles de mantener. REST se basa en el protocolo HTTP y utiliza verbos HTTP como GET, POST, PUT, DELETE, etc., para realizar operaciones CRUD (crear, leer, actualizar y eliminar) en los recursos web.

INTEROPERABILIDAD

SOAP: Simple Object Access Protocol (SOAP) es un protocolo de comunicación que se utiliza para intercambiar mensajes entre aplicaciones en diferentes plataformas y lenguajes de programación. SOAP se basa en XML y utiliza un formato de mensaje estructurado para realizar operaciones en los servicios web.

INTEROPERABILIDAD

JSON: JavaScript Object Notation (JSON) es un formato de intercambio de datos que se utiliza para transmitir datos entre aplicaciones. JSON se basa en una sintaxis de objetos y valores que se puede utilizar en una variedad de lenguajes de programación y plataformas.

INTEROPERABILIDAD

XML: eXtensible Markup Language (XML) es un lenguaje de marcado que se utiliza para describir y etiquetar datos. XML se utiliza en una variedad de aplicaciones, incluyendo la descripción de datos de configuración, la creación de lenguajes de marcado personalizados y la transferencia de datos entre aplicaciones.

INTEROPERABILIDAD

REST y SOAP son protocolos utilizados en servicios web, mientras que JSON y XML son formatos de intercambio de datos utilizados para transmitir información entre aplicaciones.

Sin embargo, también hay algunas desventajas de SOAP. Debido a que utiliza XML para la serialización de mensajes, los mensajes SOAP tienden a ser más grandes que los mensajes JSON, lo que puede afectar el rendimiento en entornos con ancho de banda limitado. Además, la complejidad del protocolo SOAP puede hacer que sea más difícil de implementar y mantener que otros protocolos más simples, como REST.

ARQUITECTURA DE TRES CAPAS

La arquitectura de tres capas es un patrón de diseño de software que separa la lógica de presentación, la lógica de negocio y la lógica de datos en tres capas distintas.

- Capa de presentación
- Capa lógica
- Capa de datos

Retrieved from
https://sites.google.com/site/redesordenadoresgrupoc/home/arquitectura-cliente-servidor

Capa de presentación

La capa de presentación, también conocida como capa de interfaz de usuario o capa de presentación de la aplicación, es la capa superior en una arquitectura de tres capas. Esta capa se encarga de interactuar con el usuario final y presentar la información de una manera clara y amigable.

Responsabilidades de la Capa de presentación

- Recopilar información de entrada del usuario, como formularios o botones.
- Validar y procesar la información de entrada para garantizar que sea coherente y precisa.
- Obtener información de la capa de negocio y presentarla al usuario de manera clara y concisa.
- Permitir al usuario interactuar con la información, por ejemplo, mediante la selección de opciones o la realización de búsquedas.
- Proporcionar una experiencia de usuario atractiva y fácil de usar, que fomente la adopción y el uso continuo de la aplicación.

Herramientas capa de presentación

- HTML, CSS y JavaScript: HTML proporciona la estructura básica de la página web, CSS se utiliza para diseñar y dar estilo a la página, y JavaScript se utiliza para agregar interactividad a la página web.
- React: es una biblioteca de JavaScript popular para construir interfaces de usuario reutilizables.
- Angular: es un framework de JavaScript que se utiliza para construir aplicaciones web de una sola página.
- Vue.js: es un framework de JavaScript que se utiliza para construir aplicaciones web de una sola página y aplicaciones móviles.
- Bootstrap: es un framework de CSS que se utiliza para construir interfaces de usuario receptivas y móviles
- jQuery: es una biblioteca de JavaScript popular que se utiliza para simplificar la manipulación del DOM y el manejo de eventos.
- Materialize: es un framework de CSS que se utiliza para crear interfaces de usuario en el estilo de Material Design de Google. Materialize proporciona muchos estilos y componentes preconstruidos que facilitan la creación de interfaces de usuario.

Capa Lógica

La capa de negocio es responsable de la lógica de negocios y procesamiento de datos de una aplicación. Esta capa recibe solicitudes de la capa de presentación (capa frontal) y se comunica con la capa de datos (capa de acceso a datos) para obtener y actualizar información en la base de datos.

Responsabilidades Capa lógica (negocio)

- **Validación de datos**: La capa de negocio verifica si los datos ingresados por el usuario son válidos antes de procesarlos. También puede realizar validaciones adicionales como la verificación de permisos y restricciones de acceso.
- **Lógica de negocio**: La capa de negocio implementa la lógica de negocio de la aplicación. Esto incluye cualquier proceso de negocio que debe realizarse antes de que los datos sean almacenados en la base de datos. Por ejemplo, si una aplicación de comercio electrónico recibe una solicitud para comprar un producto, la capa de negocio verifica si hay suficiente inventario disponible antes de procesar la transacción.

Procesamiento de datos: La capa de negocio se encarga de procesar los datos ingresados por el usuario. Esto puede incluir la creación de nuevas entradas en la base de datos, la actualización de registros existentes o la eliminación de datos.

Gestión de transacciones: La capa de negocio también es responsable de administrar transacciones. Las transacciones son operaciones que se deben completar como una unidad completa, lo que significa que si alguna parte de la transacción falla, la transacción completa se cancela.

Herramientas capa lógica

- Java: es un lenguaje de programación popular para construir aplicaciones de software empresarial. Java es conocido por su capacidad para manejar grandes cargas de trabajo y por su seguridad y confiabilidad.
- Spring Framework: es un framework de Java para construir aplicaciones empresariales. Spring proporciona muchas características, como inyección de dependencias, **ORM (Mapeo de Objetos-Relacionales)**, seguridad y pruebas unitarias.
- Node.js: es un entorno de tiempo de ejecución de JavaScript que se utiliza para construir aplicaciones escalables en el lado del servidor. Node.js proporciona muchas características, como E/S sin bloqueo, procesamiento de datos en tiempo real y manejo de múltiples conexiones.
- Express: es un framework de Node.js para construir aplicaciones web escalables y flexibles. Express proporciona muchas características, como enrutamiento, middleware, gestión de sesiones y soporte para múltiples vistas.
- Django: es un framework de Python para construir aplicaciones web. Django proporciona muchas características, como un ORM, autenticación, seguridad y soporte para plantillas.

Capa de datos

Almacenar y gestionar los datos pertenecientes al sistema de software.

- Proporcionar los mecanismos de acceso necesarios para que la capa lógica pueda usar los datos.

Herramientas capa datos

- **MySQL**: es un sistema de gestión de bases de datos relacional de código abierto que se utiliza para almacenar y recuperar datos. MySQL es popular por su escalabilidad, velocidad y fiabilidad.
- **PostgreSQL:** es otro sistema de gestión de bases de datos relacional de código abierto que se utiliza para almacenar y recuperar datos. PostgreSQL es conocido por su robustez, seguridad y capacidad de gestión de datos geoespaciales.
- **MongoDB:** es un sistema de gestión de bases de datos NoSQL que se utiliza para almacenar y recuperar datos en formato JSON. MongoDB es popular por su capacidad de escalabilidad horizontal, su flexibilidad y su facilidad de uso.
- **Oracle Database:** es un sistema de gestión de bases de datos relacional que se utiliza para almacenar y recuperar datos. Oracle es popular por su escalabilidad, rendimiento y seguridad.
- **Firebase:** es una plataforma de Google que proporciona una base de datos en tiempo real y herramientas de backend para aplicaciones web y móviles. Firebase es popular por su facilidad de uso, escalabilidad y capacidad de integración con otras herramientas de Google.

MICROSERVICIOS

Los microservicios son una arquitectura de software en la que una aplicación se divide en pequeños servicios independientes, cada uno de los cuales realiza una tarea específica y se comunica con otros servicios mediante una interfaz bien definida. Cada servicio se ejecuta en su propio proceso y puede implementarse, actualizarse y escalar de forma independiente. Esta arquitectura promueve la flexibilidad, la escalabilidad y la capacidad de respuesta, lo que la hace popular para aplicaciones modernas basadas en la nube y altamente distribuidas. Además, los microservicios permiten una mayor agilidad en el desarrollo de software y facilitan la integración continua y la entrega continua (CI/CD).

EJEMPLO

- **Autenticación:** Este servicio se encargaría de la autenticación de los usuarios y la generación de tokens de autenticación para su uso en la aplicación.
- **Catálogo de productos:** Este servicio almacenaría y recuperaría información sobre los productos que se venden en la plataforma.
- **Carrito de compras:** Este servicio permitiría a los usuarios agregar y eliminar productos de su carrito de compras y calcular el precio total de los productos seleccionados.
- **Gestión de pedidos:** Este servicio se encargaría de procesar los pedidos de los usuarios y coordinar el envío de los productos.
- **Pago:** Este servicio manejaría los pagos en línea y verificaría la autenticidad de las transacciones.
- **Comentarios y calificaciones**: Este servicio permitiría a los usuarios dejar comentarios y calificaciones sobre los productos que han comprado.

CONSIDERACIONES

- **División de responsabilidades:** Es importante identificar las responsabilidades de cada microservicio de forma clara y concisa. Cada microservicio debería ser responsable de una tarea específica y no debería tener una funcionalidad superpuesta con otros microservicios.

- **Comunicación entre microservicios:** La comunicación entre microservicios debe ser cuidadosamente diseñada para asegurarse de que los datos se transmitan de forma segura y eficiente. Es importante definir un protocolo de comunicación entre los microservicios y establecer un mecanismo de control de errores.

- **Escalabilidad:** Es importante que cada microservicio pueda escalar de forma independiente para manejar aumentos en la carga de trabajo. Se debe tener en cuenta la escalabilidad tanto vertical (aumento de recursos) como horizontal (adición de instancias)

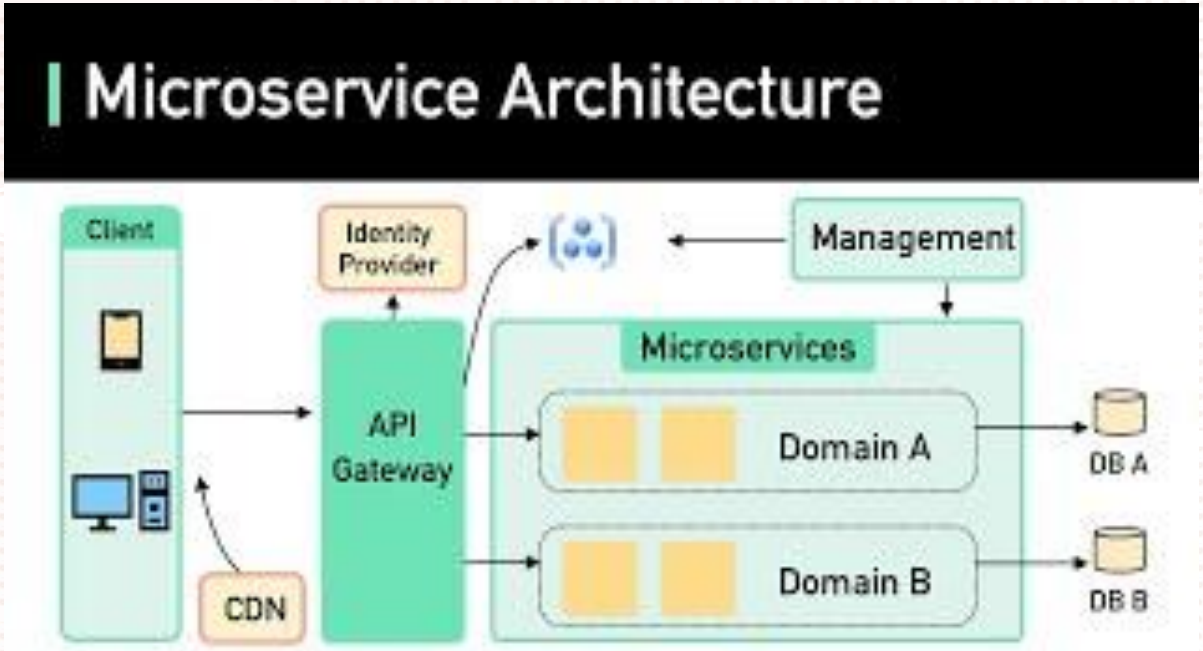
- **Tolerancia a fallos**: La arquitectura de microservicios debe estar diseñada para ser tolerante a fallos. Si un microservicio falla, no debería afectar el funcionamiento de los demás microservicios.

- **Monitoreo y registro:** Es importante tener un sistema de monitoreo y registro para detectar problemas de rendimiento, errores y otros problemas en la arquitectura de microservicios. El monitoreo debería incluir tanto el rendimiento individual de cada microservicio como la interacción entre ellos.

HERRAMIENTAS

- **Docker Compose**: de orquestación de contenedores de Docker que permite definir y ejecutar aplicaciones multicontenedor. Con Docker Compose, puedes definir la configuración para cada contenedor y definir cómo se comunican entre sí, lo que facilita la orquestación de los servicios y su integración en una aplicación completa. Con un archivo YAML, puedes definir los servicios de tu aplicación y ejecutarlos todos juntos con un solo comando.
- **Kubernetes:** plataforma de orquestación de contenedores de código abierto que automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores. Con Kubernetes, los desarrolladores pueden definir y desplegar sus aplicaciones en contenedores, y luego Kubernetes se encarga de administrar y orquestar la ejecución de esos contenedores en un clúster. Kubernetes proporciona características avanzadas como el autoescalado, la tolerancia a fallos y la gestión de configuración, lo que hace que sea más fácil para los desarrolladores crear y administrar aplicaciones distribuidas en un entorno de nube.
- **Docker Swarm:** Docker Swarm es una herramienta de orquestación de contenedores que se integra en la plataforma Docker. Swarm permite la creación y gestión de clústeres de nodos Docker, lo que facilita el despliegue, la escalabilidad y la gestión de aplicaciones distribuidas en contenedores.
- **Funciones Lambda:** Forma popular de implementar microservicios sin tener que preocuparse por la infraestructura subyacente. Con las funciones Lambda, los desarrolladores pueden escribir pequeñas piezas de código que se ejecutan de manera independiente y se escalan automáticamente según la demanda.

MICROSERVICES



REFERENCES

- Al-Masri, E., & Mahmoud, Q. H. (2016). RESTful web services APIs for IoT interoperability: A review. Journal of Network and Computer Applications, 66, 212-222. doi: 10.1016/j.jnca.2016.04.015
- Rossini, A., Panzieri, F., Rizzi, D., & Mirandola, R. (2017). A microservices-based architecture for interoperability in smart cities. IEEE Transactions on Services Computing, 10(1), 1-14. doi: 10.1109/TSC.2016.2539023
- Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley Professional.
- Pressman, R. S., & Maxim, B. R. (2014). Software engineering: A practitioner's approach. McGraw-Hill Education.
- O'Reilly, T. (2015). What are microservices? Why should we use them? Retrieved from https://www.oreilly.com/radar/what-are-microservices-why-should-we-use-them/
- Dragoni, N., Giallorenzo, S., Lafuente, A. L., & Mazzara, M. (2017). Microservices: Yesterday, today, and tomorrow. Communications of the ACM, 60(6), 85-93. doi: 10.1145/3085563