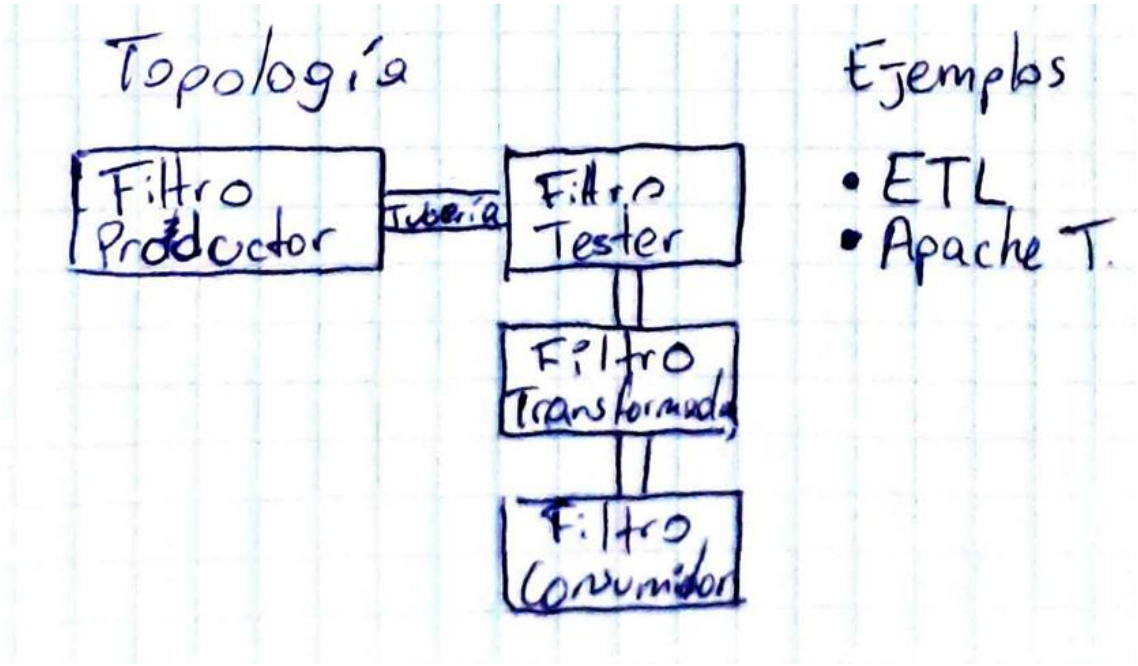
Que es la arquitectura de software

La arquitectura de software es la definición, estructura y relaciones de componentes lógicos, su forma de definirlo y la relación entre cada uno de ellos, además del comportamiento que van a tomar a lo largo de la vida del software para solventar los requerimientos de la empresa y sus activos.

Estilo arquitectura tubería y filtros

Tubería: Forman el canal de comunicación entre filtros, a través de endpoints.
Filtros: autónomos y sin estado, cumplen una sola tarea se dividen en:

1. Productor, punto de partida origen.
2. Transformador, recibe entrada, transforma uno o mas datos y emite salida.
3. Comprobador, recibe entrada ejecuta prueba en base a uno o mas criterios, emite salida.
4. Consumidor, punto de terminación, puede almacenar el resultado final de proceso de canalización en una base de datos o exponerlo en interfaz usuario.



Falacias de la arquitectura distribuida

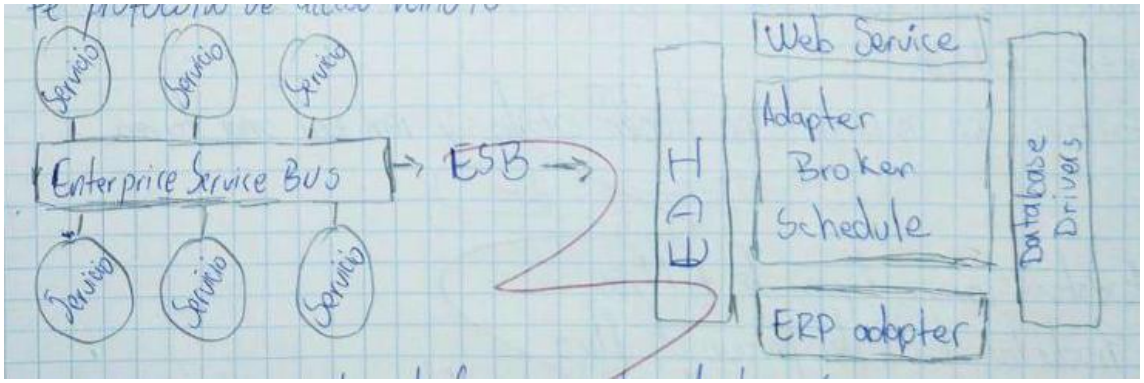
1. **La latencia es cero**, no siempre esta latencia va a ser de cero así el tiempo de respuesta no se note al usuario siempre habrá unos microsegundos de diferencia entre la petición y la respuesta.
2. **El ancho de banda es infinito**, esto depende sobre todo en el ancho de banda de comunicación entre las peticiones si el usuario maneja un ancho

de banda bajo y el servidor un ancho de banda alto se notará la diferencia y la petición trabajara a ese ancho de banda.

3. **La red es confiable**, todos los desarrolladores como arquitectos suponen que la red es confiable pero no lo es.
4. **La red es segura**, la seguridad se vuelve mucho mas desafiante en una arquitectura distribuida.
5. **La tipología nunca cambia**, se asume erróneamente que la red es estática, pero pequeños cambios en la infraestructura pueden causar fallos inesperados como timeouts o errores de latencia.

Arquitectura Orientada a microservicios

Promueve la reutilización de competentes de la empresa y expone en forma comercial mediante servicios independientes para que puedan ser usados mediante protocolos de acceso remoto.



Servicio es una parte de funcionamiento de la aplicación.

ESB es la parte del SOA, promueve o contiene enrutamiento y transformación de datos es la interconexión entre los servicios y que apunte o manejen bien los servicios adecuados a los activos empresariales.

Base de datos documentales vs Base de datos relacionales

✅ **Similitud**

- **Persistencia y consulta de datos:** Ambas almacenan datos de forma estructurada y permiten realizar operaciones como inserción, actualización, eliminación y consultas. Es decir, ambas sirven como sistemas de gestión de datos y pueden usarse en aplicaciones reales.

🔁 **Diferencias**

1. **Estructura de los datos:**

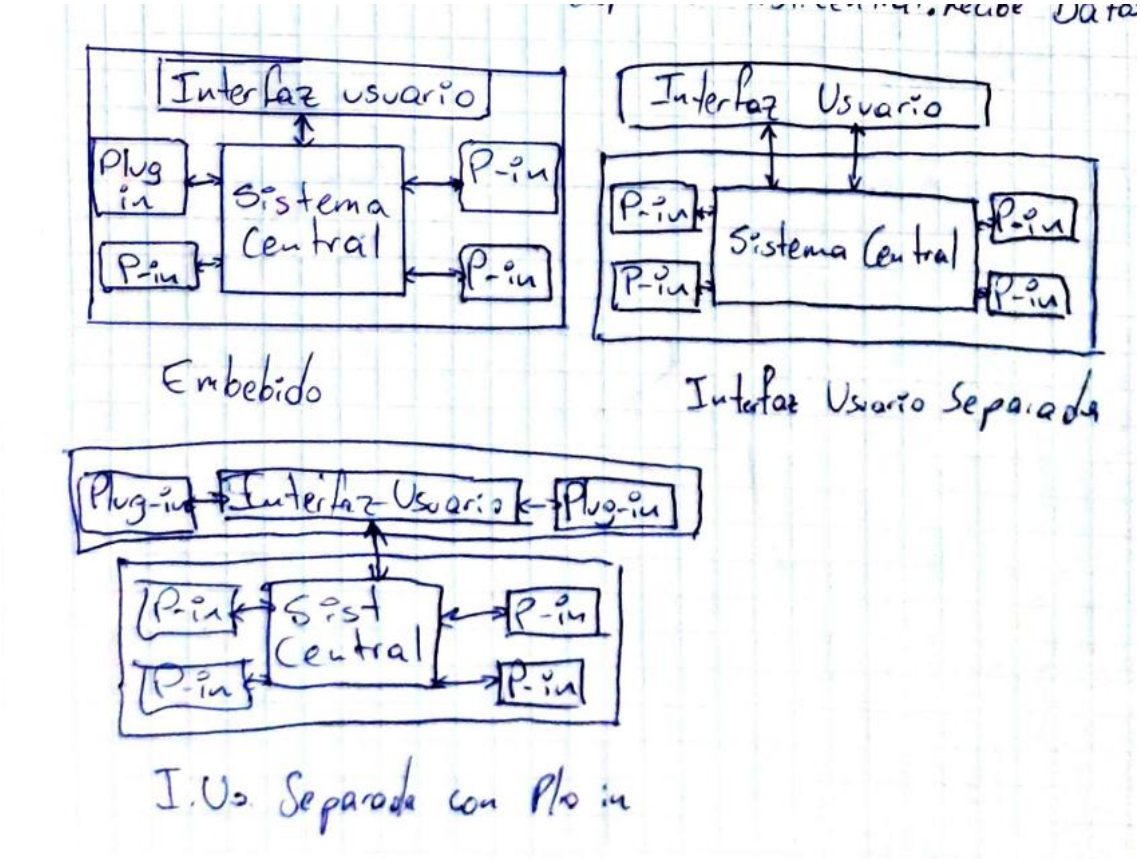
- **Relacional:** Usa tablas con filas y columnas. Cada fila representa un registro, y cada columna un atributo con un tipo de dato definido.
- **Documental:** Usa documentos generalmente en formato JSON o BSON, donde cada documento puede tener una estructura flexible y diferente a los demás.

2. **Modelo de esquemas:**
    - **Relacional:** Tiene un esquema rígido. Es obligatorio definir previamente la estructura de las tablas.
    - **Documental:** Tiene un esquema flexible o incluso sin esquema (schema-less). Los documentos pueden tener campos diferentes entre sí y no requieren una estructura predefinida.

Micronucleo

Arquitectura monolítica que divide en:

1. Sistema Central: Funcionalidad mínima para la ejecución del sistema.
2. Plu-in: Componente independiente con funcionalidad especifica que se comunica a través de endpoint al sistema central recibe datos necesarios del sistema central.



Principios de api REST

1. Client-server Separation, el servidor solo provee servicios al cliente pero los dos están separados.
2. Stateless, el servidor no recuerda nada sobre las solicitudes anteriores realizadas por el cliente, cada solicitud contiene toda la información.
3. Cacheable, puede almacenar la respuesta a una solicitud y utilizarla para responder a solicitudes futuras.
4. Layared System, una api restfull puede tener varias capas y cada capa proporcionar una función.
5. Uniform interface, una api restfull debe tener una interfaz consistente que todos los clientes puedan entender.
6. Codeo n Demand, algunas api restfull pueden proporcionar código ejecutable al cliente, como una función de javascritp.

Buenas prácticas de diseño de api

1. Utilice URI de recursos significativos.
2. Utilice los métodos HTTP correctamente.
3. Utilice los códigos de respuesta HTTP correctamente.
4. Utilice sustantivos en lugar de verbos en URI.
5. Utilice convenciones de nomenclatura coherentes.

6. Proporcione documentación completa.
7. Control de versiones en la API para garantizar que los cambios en la API no interrumpan la funcionalidad.
8. Maneje los errores con elegancia.
9. Utilice medidas de seguridad adecuadas.