PONTIFICA UNIVERSIDAD JAVERIANA

TALLER 2 – PRESENTACIÓN 1

TEMA:

COMPONENTES INDEPENDIENTES, CQRS, MICROSERVICIOS, SERVICIOS, SOA

INTEGRANTES:

SARA JULIANA PEÑA

DANIEL ALZATE

IVAN MARTINEZ

MATERI: ARQUITECTURA DE SOFTWARE

PROFESOR: ANDRES SANCHEZ

2024-03

1. DEFINICIÓN
   - COMPONENTES INDEPENDIENTES
     La arquitectura de componentes es un enfoque para diseñar y construir sistemas de software que se centra en la creación de componentes reutilizables e independientes. Estos componentes representan unidades lógicas y funcionales del sistema, cada uno con su propia responsabilidad y funcionalidad especifica.
     Permite que los sistemas se construyan a partir de un conjunto de componentes interconectados que se comunican entre sí a través de interfaces bien definidas. Permitiendo mayor modularidad, flexibilidad y reutilización del código.

   - CQRS
     "Command Query Responsability Segregation". También conocido como el patrón de división de responsabilidad de consulta de comando. O conocido por la segregación de responsabilidad de comandos y consultas. Trata de separar las operaciones de lectura y actualización de un almacén de datos. Se usa el patrón para separar actualizaciones de las consultas si se tienen requisitos diferentes de rendimiento, latencia o coherencia. La implementación de CQRS en la aplicación puede maximizar el rendimiento, la escalabilidad y la seguridad.

   - MICROSERVICIOS
     Es un enfoque arquitectónico y organizativo para el desarrollo de software donde este está compuesto por pequeños servicios que se comunican a través de APIs bien definidas. Este tipo de arquitectura se considera moderna y flexible a comparación del modelo de desarrollo tradicional de la arquitectura monolítica. Ya que permite que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar, permitiendo innovación.

   - SERVICIOS – SOA
     Se define por hacer que los componentes del software sean reutilizables a través de interfaces de servicio que se comunican a través de una red con un lenguaje común. En otras palabras, esta arquitectura integra los elementos del software que se implementan y se mantienen por separado y permite que se comuniquen entre sí para que trabajen en conjunto para formar aplicaciones de software de distintos sistemas.
     Esta arquitectura se caracteriza por usar un "*ESB" o bus de servicios empresariales,* es un patrón arquitectónico mediante el cual un componente de software realiza integraciones entre aplicaciones. Este se puede encargar de integrar y transformar interfaces de servicio disponibles para su reutilización para nuevas aplicaciones.

2. CARACTERISTICAS
   - COMPONENTES INDEPENDIENTES
     - Es un estilo de diseño para aplicaciones compuestas de componentes individuales
     - Se enfatiza por la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
     - Usa componentes discretos los cuales se comunican a través de interfaces que contienen los métodos, eventos y propiedades del sistema.
     - Puede ser extensible, reemplazable, independiente. Además, permite encapsular, y reutilizar
   - CQRS

- Sistema evoluciona mejor con el tiempo, evita que los comandos de actualización provoquen conflictos de combinación en el nivel de dominio.
- Divide las operaciones de lectura (consultas) de las operaciones de escritura (comandos), lo que facilita la escalabilidad del sistema, ya que cada una puede ser escalada independientemente según sea necesario.
- Permite que cada parte del sistema evolucione de manera independiente, lo que facilita la introducción a nuevos cambios sin afectar otras partes del sistema
- Permite aplicar técnicas de optimización especificas para mejorar el rendimiento de cada parte, por lo cual permite una mejor experiencia de usuario.

- MICROSERVICIOS
  - Cada servicio componente de arquitectura de microservicios se puede desarrollar, implementar, operar y escalar sin afectar el funcionamiento de otros servicios.
  - Los servicios no necesitan compartir ninguno de sus códigos o implementaciones de otros servicios. Su comunicación se realiza a través de APIs bien definidas.
  - Cada servicio está diseñado para un conjunto de capacidades y se enfoca en resolver un problema específico.
  - Los microservicios suelen implementarse en contenedores para lograr escalabilidad y portabilidad adicionales.

- SERVICIOS – SOA
  - Se reutilizan servicios en diferentes procesos empresariales para ahorrar tiempo y dinero. Se crean aplicaciones en menos tiempo.
  - Permite un mantenimiento eficiente, ya que es más fácil crear, actualizar y corregir errores de servicios pequeños que en bloques grandes de código. Su modificación no afecta la funcionalidad general.
  - Presenta una buena adaptabilidad a los avances tecnológicos.
  - Permite la posibilidad de ejecución de varios servicios en varios lenguajes de programación, servicios y plataformas, lo cual aumenta su escalabilidad.

3. HISTORIA Y EVOLUCIÓN
   - COMPONENTES INDEPENDIENTES

     Se empieza a hablar de una arquitectura de Componentes Independientes en los años 60 y 70, cuando el concepto de modularidad comenzó a ganar importancia en el desarrollo de software. A medida que los sistemas se volvían más complejos, los desarrolladores buscaban formas de dividir aplicaciones en módulos reutilizables.
     - **Década de 1960**: Surge la modularidad como una práctica para dividir programas en piezas más manejables y reutilizables, influenciada por los primeros lenguajes de programación como **FORTRAN** y **COBOL**.
     - **Años 70 y 80**: La programación orientada a objetos (OOP) introduce la idea de encapsular datos y funciones en **clases**, un precursor de los componentes. El enfoque en el diseño modular crece con lenguajes como **Smalltalk** y **C++**.
     - **Años 90**: La **Component-Based Software Engineering (CBSE)** formaliza la idea de crear **componentes reutilizables** con interfaces definidas. Tecnologías como **CORBA**, **DCOM** y **EJB** promueven la creación de componentes distribuidos que se pueden reutilizar en diferentes aplicaciones.

- **Años 2000 en adelante**: La aparición de la arquitectura orientada a servicios (SOA) y posteriormente los **microservicios** lleva la modularidad a un nuevo nivel, permitiendo la creación de componentes totalmente independientes que pueden ser desplegados, actualizados y escalados por separado.

- CQRS

  Bertrand Meyer en su libro *Construcción de software orientado a objetos* en 1997, introdujo el termino CQS *separación de consultas y comandos* con la idea de dividir operaciones de un sistema en dos categorías: Consultas, las cuales devuelven un resultado sin cambiar el estado del sistema y Comandos, cambian el estado del sistema. Sin embargo, Greg Young introdujo el concepto *segregación de responsabilidad de consultas y comandos* en el 2000. Young fue el que formalizó el patrón y lo presentó como una solución para mejorar la capacidad de escalado y rendimiento de aplicaciones con alta concurrencia.
  En el 2010, el patrón CQRS se combinó frecuentemente con otro patrón de arquitectura: **Event Sourcing**. Es un modelo en el que los cambios en el estado del sistema se almacenan como una secuencia de eventos, en lugar de solo registrar el estado actual.
  La combinación de CQRS y Event Sourcing permitía separar de manera más efectiva las responsabilidades de consulta y comando, ya que los eventos podrían registrarse y procesarse de manera asincrónica para actualizar el estado del sistema y generar proyecciones de datos especializadas para las consultas.
  En la última década, CQRS ha sido adoptado en aplicaciones que requieren alta escalabilidad y rendimiento, especialmente en sistemas distribuidos. Con la llegada de arquitecturas modernas como microservicios, cloud computing, y plataformas como Azure, AWS, y Google Cloud, CQRS se ha vuelto un patrón común para diseñar aplicaciones que requieren flexibilidad y alta concurrencia.

- MICROSERVICIOS

  El término de "*microservicios"* comenzó a ser escuchado a mediados de la década del 2010 cuando empezó a ser popular al ser implementado en grandes empresas como Netflix, Amazon, Spotify, y Uber. Estos se enfrentaban a diversos desafíos como la escalabilidad, la frecuencia de despliegue y la necesidad de independencia entre equipos de desarrollo.
  En 2014, Martin Fowler y James Lewis publicaron un artículo titulado "*Microservices – a definition of this new architectural term*" que termino de popularizar y formalizar el termino *microservicios* como una forma de diseñar aplicaciones como un conjunto de pequeños e independientes servicios que se comunican a través de APIs ligerias.
  A lo largo de esta década, los microservicios fueron adoptados por una variedad de organizaciones que buscaban:
    - Despliegue independiente: Los microservicios permiten a los equipos desarrollar, probar y desplegar sus servicios de manera independiente, sin depender de todo el sistema.
    - Escalabilidad: Cada microservicio puede escalarse de forma independiente según las necesidades de carga.
    - Tolerancia a fallos: Si un microservicio falla, el resto del sistema puede seguir funcionando, lo que mejora la resiliencia

  Sin embargo, desde el 2015 se vio que es tipo de arquitectura implicaba una serie de desafíos en las organizaciones. Se encuentra que hay una complejidad operativa para gestionar una gran cantidad de microservicios, la cual implica manejar herramientas para la orquestación, monitoreo y administración de fallos entre los múltiples servicios. Se ha visto que el manejo de los costes de comunicación a aumentado ya que se requiere más

comunicación entre componentes a través de redes. Y se ha requerido mantener una compatibilidad entre las diferentes versiones de microservicios.
Por otro lado, hoy en día, los microservicios son ampliamente adoptados, y muchas organizaciones están construyendo nuevas aplicaciones bajo este enfoque desde el principio. Además, con el auge de tecnologías como los contenedores, Kubernets, y el movimiento hacia la nube híbrida, los microservicios continúan siendo relevantes.

- SERVICIOS – SOA
  La arquitectura orientada a servicios (SOA) surgió en los años 90 como respuesta a la necesidad de integrar sistemas complejos y promover la reutilización de componentes. Inicialmente, tecnologías como CORBA, DCOM y EJB permitieron aplicaciones distribuidas, pero presentaban desafíos de complejidad e interoperabilidad.
  A inicios de los 2000, SOA tomó forma con servicios más desacoplados y estándares como SOAP y WSDL, facilitando la comunicación entre sistemas empresariales. A pesar de su adopción, SOA se volvió complejo y pesado, lo que llevó a su evolución hacia los microservicios en la década de 2010, un enfoque más ágil y ligero que sigue principios similares, pero con servicios más pequeños y escalables de forma independiente.
  Hoy en día, SOA aún se utiliza en contextos empresariales, especialmente para integrar sistemas legados, mientras que los microservicios predominan en aplicaciones modernas por su simplicidad y flexibilidad.

4. VENTAS Y DESVENTAJAS
   - COMPONENTES INDEPENDIENTES
     - Ventajas
       - Mantenimiento: los componentes son aislados, lo que facilita la depuración y solución de problemas son más sencillas de solucionar. Reduce el riesgo de fallas de todo el sistema
       - Escalabilidad: los componentes se pueden escalar independientemente según la demanda, lo que permite una actualización más eficiente de los recursos.
       - Modularidad: cada componente es autónomo con interfaces bien definidas, lo que hace que sea más fácil de entender, desarrollar y mantener.
     - Desventajas
       - Para organizar una arquitectura basada en componentes, se deben probar todos los componentes de forma independiente. Si un componente es modificado de manera incorrecta, puede provocar fallos en el sistema.
       - A medida que aumenta la cantidad de componentes del sistema, la gestión de sus interacciones, dependencias y complejidad generar se vuelven un desafío.
       - Se requiere una alta mantenibilidad, la actualización y el mantenimiento de las dependencias puede ser complicado. Es posible que se necesiten pruebas exhaustivas y actualizaciones para garantizar la estabilidad del sistema.
   - CQRS
     - Ventajas
       - Se puede aplicar un control de acceso más detallado, lo que lleva a mejorar la seguridad del sistema.

        - Permite un modelado de dominio más exhaustivo y expresivo, ya que cada modelo de datos puede ser diseñado específicamente para satisfacer las necesidades de las operaciones que soporta.
        - Permite una separación de responsabilidades, lo que permite que el diseño del sistema sea más claro y comprensible.
    - Desventajas
        - La implementación básica de CQRS es sencilla, sin embargo, al usarlo en conjunto al patrón Event Sourcing aumenta considerablemente su complejidad.
        - Cuando se modifican estados del sistema puede tardar en reflejarse en la zona encargada de las query, lo que conlleva problemas de consistencia.
        - Comunicación entre componentes de lectura y escritura puede ser compleja y puede requerir el uso de patrones de mensajería para garantizar una comunicación efectiva y confiable.

- MICROSERVICIOS
    - Ventajas
        - Permite que cada servicio escale de forma independiente para satisfacer la demanda de la aplicación.
        - Permite que los equipos de desarrollo tengan libertad de elegir la mejor herramienta para resolver problemas específicos.
        - La independencia de los servicios aumenta la resistencia de que la aplicación genere errores.
    - Desventajas
        - La comunicación entre los microservicios puede ser compleja. Se requiere una gestión adecuada de las llamadas y sincronización entre los servicios.
        - Se necesita herramientas y estrategias adecuadas para rastrear y solucionar problemas.
        - Cada microservicio requiere su propia infraestructura. La gestión de múltiples sistemas operativos, bases de datos y entornos de ejecución puede generar una sobrecarga en la infraestructura.

- SERVICIOS – SOA
    - Ventajas
        - Permite que los desarrolladores tomen funciones de una plataforma o un entorno y la ajusten e implementen en otros.
        - Mantenimiento sencillo, ya que los servicios al ser independientes se pueden modificar y actualizar cada uno cuando sea necesario.
        - Mayor agilidad empresarial y una comercialización más rápida. Ya que facilita la integración de aplicaciones, datos y automatización de procesos de negocio. Permite a los desarrolladores enfocarse más en la entrega y mejora de sus aplicaciones.
    - Desventajas
        - Depende de una implementación con estándares. Sin estos, la comunicación entre aplicaciones requiere de tiempo y mas implementación.

- Es necesario conocer los procesos de negocio, clasificarlo y extraer funcionalidades para formar capas de servicio que van a ser requeridas para cualquier proceso de negocio.
- No es para aplicaciones de alta nivel de transferencia de datos ni para aplicaciones que tienen un corto periodo de vida.

5. CASOS DE USO
   - COMPONENTES INDEPENDIENTES
     - Desarrollo Web y Aplicaciones Cliente-Servidor: Permite la construcción de interfaces de usuario modulares y reutilizables que pueden ser integradas fácilmente.
     - Desarrollo de Sistemas Integrados: facilita la integración de diferentes componentes de Hardware y Software en un sistema cohesivo y funcional.
     - Desarrollo de Sistemas Distribuidos: permite la construcción de sistemas modulares y escalables que pueden ser distribuidos en diferentes nodos de la red y comunicarse entre sí a través de interfaces bien definidas.
     - Desarrollo de Aplicaciones basadas en la Nube: permite la construcción de sistemas escalables y elásticos que pueden adaptarse dinámicamente a las demandas cambiantes del usuario.
   - CQRS
     - Dominios colaborativos en los que muchos usuarios acceden a los mismos datos en paralelo. Permite definir comandos con detalle para minimizar los conflictos de combinación en el nivel de dominio.
     - Escenarios en los que se espera que el sistema evolucione con el tiempo y que podrían contener varias versiones del modelo, o en los que las reglas de negocio cambian con regularidad.
     - Usar el patrón también es recomendable cuando una parte del equipo se puede hacer cargo del modelo de escritura mientras otra se dedica a la interfase de usuario y al modelo de lectura.
     - Integración con otros sistemas, especialmente en combinación con Event Sourcing, en los que el error temporal de un subsistema no debería afectar a la disponibilidad de los demás.

   - MICROSERVICIOS
     - Servicio de catálogo de productos: Maneja la lista de productos, sus descripciones, precios y disponibilidad.
     - Servicio de carrito de compras: Administra los carritos de compra de los usuarios, permitiendo agregar y eliminar productos.
     - Servicio de procesamiento de pagos: Gestiona las transacciones y la integración con pasarelas de pago.
     - Servicio de recomendación de productos: Ofrece recomendaciones personalizadas basadas en el historial de compras y navegación del usuario.
     - Servicio de envío: Calcula costos y tiempos de envío, y gestiona la logística de entrega.

- SERVICIOS – SOA
    - Integración de sistemas: Cuando una empresa tiene varios sistemas antiguos que necesitan interactuar con nuevas aplicaciones o plataformas sin reemplazar completamente la infraestructura existente.
    - Empresas grandes con múltiples aplicaciones: SOA es ideal para organizaciones grandes que manejan aplicaciones dispares (ERP, CRM, sistemas financieros) y requieren una comunicación fluida entre ellas.
    - Interoperabilidad entre plataformas heterogéneas: SOA permite que servicios desarrollados en diferentes tecnologías (Java, .NET, etc.) se comuniquen mediante estándares como SOAP o REST.
    - Automatización de procesos empresariales: Empresas que desean automatizar procesos empresariales complejos pueden utilizar SOA para orquestar servicios que colaboren entre sí, facilitando flujos de trabajo a través de diferentes sistemas.

6. CASOS DE APLICACIÓN
    - COMPONENTES INDEPENDIENTES
      Se utiliza ampliamente en varios dominios e industrias. Como:
        - Java Enterprise Edition (Java EE): ahora "*Jakarta EE*", es un marco popular para desarrollar aplicaciones empresariales utilizando un enfoque basado en componentes. Proporciona componentes como Enterprise JavaBeans (EJB), JavaServer Faces (JSF) y Contexts and Dependency Injection (CDI) para crear aplicaciones escalables y modulares.
        - .NET Framework: El .NET Framework y su sucesor, .NET Core, proporcionan una arquitectura basada en componentes para crear aplicaciones en plataformas Windows. Componentes como Windows Presentation Foundation (WPF) para aplicaciones de escritorio y ASP.NET MVC para aplicaciones web que siguen este enfoque.
        - AngularJS y Angular: son marcos de JavaScript que soportan una arquitectura basada en componentes para desarrollar aplicaciones web dinámicas. Los componentes en Angular encapsulan HTML, CSS y la lógica de JavaScript, promoviendo la reutilización y el mantenimiento.
        - Apache Hadoop: es un marco de código abierto que utiliza una arquitectura basada en componentes para el almacenamiento distribuido y el procesamiento de grandes conjuntos de datos. Componentes como el Hadoop Distributed File System (HDFS) para almacenamiento y MapReduce para procesamiento están diseñados para trabajar juntos de manera escalable y tolerante a fallos.
        - Sistemas de Planificación de Recursos Empresariales (ERP): Los sistemas ERP como SAP, Oracle ERP y Microsoft Dynamics utilizan arquitecturas basadas en componentes para gestionar diversos procesos empresariales, como finanzas, recursos humanos, cadena de suministro y gestión de relaciones con clientes (CRM). Cada módulo dentro de un sistema ERP actúa como un componente con interfaces bien definidas para la integración e interoperabilidad.
    - CQRS
        - Sistemas de Comercio Electrónico
          En plataformas como **Amazon** o **eBay**, el patrón CQRS es extremadamente útil debido a la alta concurrencia y la necesidad de manejar diferentes tipos de

operaciones con distintos requisitos de rendimiento. En el caso de que los clientes busquen productos, revisan reseñas, comparan precios, y demás, es necesario que estas operaciones requieren ser rápidas y eficientes. Además, se tienen las operaciones de actualización, como agregar productos al carrito o realizar una compra, se manejan en una arquitectura separada, que garantiza la consistencia de los datos y puede incluir validaciones más complejas.

- Sistemas Bancarios y Financieros
  En sistemas financieros y bancarios, como las **aplicaciones de banca en línea** o las **plataformas de trading**, CQRS se utiliza para separar la gestión de consultas y actualizaciones: Los usuarios realizan consultas sobre sus saldos, transacciones anteriores o estado de cuentas. Estas operaciones son frecuentes, pero no afectan el estado actual del sistema. Las transacciones financieras, como transferencias de dinero o pagos, son actualizaciones críticas que deben manejarse con gran precisión y con mecanismos que aseguren la consistencia del sistema.
- Sistemas de Logística y Gestión de Inventarios
  En empresas de logística como **FedEx**, **UPS** o en la gestión de inventarios de gigantes como **Walmart**, CQRS permite optimizar la lectura y actualización de datos: Los usuarios o sistemas realizan consultas frecuentes sobre el estado de los envíos, la ubicación de los paquetes o el nivel de inventarios. Estas consultas pueden manejarse a través de bases de datos optimizadas para lectura. Las operaciones de actualización, como registrar un nuevo envío, actualizar el estado de un paquete o modificar inventarios, son operaciones más complejas que necesitan un procesamiento cuidadoso para mantener la consistencia de los datos.
- Juegos en Línea y Sistemas de Streaming
  En **juegos masivos multijugador** o **plataformas de streaming** como **Netflix**, CQRS ayuda a gestionar la alta carga de datos: Los usuarios acceden a catálogos de series, películas o videojuegos, y estas consultas deben ser rápidas y optimizadas para manejar millones de usuarios concurrentes. Actualizaciones como guardar el progreso de un jugador o marcar contenido como visto requieren operaciones que mantengan la consistencia de los datos.

- MICROSERVICIO
  - Netflix
    Esta plataforma tiene una arquitectura generalizada que desde hace ya un par de años se pasó a los microservicios para el funcionamiento de sus productos. A diario recibe una media de mil millones de llamadas a sus diferentes servicios (se dice que es responsable del 30% del tráfico de Internet) y es capaz de adaptarse a más de 800 tipos de dispositivos mediante su API de streaming de vídeo, la cual, para ofrecer un servicio más estable, por cada solicitud que le pedimos, ésta realiza cinco solicitudes a diferentes servidores para no perder nunca la continuidad de la transmisión.
  - Walmart
    Se implemento en el 2012 tras constantes problemas durante los picos de actividad de los clientes. El flujo de trabajo era imperfecto y debido a problemas con el sistema, se produjo descensos de los ingresos. Cuando la empresa realizo el cambio de sistema y de arquitectura, fue posible redistribuir las tareas y simplificar el flujo de trabajo. Algunos de los resultados fue que se hizo posible un uso completo de aplicaciones móviles, así los microservicios aumentaron la

cantidad de conversiones a través de otro flujo de compras. Y se evito el tiempo de inactividad durante las rebajas de Navidad, Black Friday y otros picos de actividad de los clientes.

- Spotify:
  Según estadísticas recientes, Spotify alberga mas de 75 millones de usuarios activos. Para ofrecer la mejor experiencia posible al usuario, la empresa a usado la arquitectura de microservicios implementando lo siguiente: Cada microservicio tiene un objetivo determinado. Por ejemplo, el análisis del comportamiento de los usuarios ayuda a autogenerar listas de reproducción.
  Todo el sistema se divide en microservicios individuales y autónomos gestionados por equipos de desarrollo independientes.
  Cualquier problema probable se encuentra en un servicio específico para que no influya en la carga de trabajo general.
  Todos los servicios están aislados y por eso son mínimamente dependientes. Debido a este hecho, se hizo posible lograr la coherencia en el servicio al cliente.

- SERVICIOS – SOA
  - **Amazon**: Inicialmente utilizó una arquitectura monolítica, pero luego implementó SOA para dividir su plataforma en servicios independientes, facilitando la integración de nuevas funcionalidades y mejorando la escalabilidad.
  - **IBM**: Como pionera en soluciones empresariales, IBM ofrece su propio middleware SOA a través de herramientas como **IBM WebSphere**, ayudando a organizaciones a integrar sistemas diversos.
  - **American Express**: Utiliza SOA para integrar sus diversos sistemas bancarios y financieros, facilitando la interacción entre múltiples plataformas y aplicaciones internas.

7. RELACIÓN ENTRE TEMAS
   - Que tan común es el stack designado
     Se combinan tecnologías de fronted y backend reconocidas con enfoques arquitectónicos modernos para el desarrollo de software.
     1. **Frontend con Gatsby**: Gatsby es una herramienta moderna basada en React para construir sitios web estáticos. Es muy popular en proyectos que buscan un rendimiento excepcional y una buena optimización de SEO, gracias a su generación de sitios pre-renderizados y su sistema de plugins extensivo.
     2. **Backend con Flask y Python**: Flask es un micro-framework de Python conocido por su simplicidad y flexibilidad. Es ideal para pequeños a medianos proyectos y prototipos rápidos, y es especialmente popular en startups y entornos donde la velocidad de desarrollo es crucial. Python es un lenguaje de programación versátil con una amplia gama de aplicaciones, desde desarrollo web hasta ciencia de datos.
     3. **Base de datos MySQL**: MySQL es uno de los sistemas de gestión de bases de datos relacionales más utilizados. Su robustez, rendimiento y compatibilidad con numerosos sistemas operativos lo hacen una elección común para una gran variedad de aplicaciones.
     4. **Arquitecturas como SOA y Microservicios**: Estas arquitecturas son fundamentales en sistemas grandes y distribuidos, permitiendo servicios escalables

y mantenibles. Microservicios en particular ha ganado una enorme popularidad como una manera de construir aplicaciones resilientes y fáciles de escalar.

5. **Tecnologías y protocolos como REST y XML**: REST es el estándar de facto para el desarrollo de APIs en la web debido a su simplicidad y flexibilidad. XML sigue siendo utilizado ampliamente en muchas aplicaciones empresariales, aunque JSON ha ganado popularidad en aplicaciones web modernas.

Este stack es bastante común en el desarrollo web y aplicaciones empresariales, especialmente en proyectos que buscan combinación en: rendimiento, escalabilidad y rapidez en el desarrollo. Algunas de estas herramientas tecnológicas impulsan a los desarrolladores a tener habilidades que son altamente demandadas en el mercado laboral actual.

- Matriz de análisis de Principios SOLID vs Temas

Temas / Principios	Single Responsability	Open - Closed	Liskov Substitution	Interface Segregation	Dependency Inversion
Componentes Independientes	Alta	Media	Alta	Alta	Alta
CQRS	Alta	Alta	Media	Alta	Alta
Microservicios	Alta	Alta	Alta	Alta	Alta
Servicios	Media	Alta	Media	Media	Alta
SOA	Media	Alta	Baja	Media	Alta

Descripción:

- Single Responsibility: Microservicios y Componentes Independientes tienen una alta adherencia por su naturaleza desacoplada.
- Open-Closed: CQRS y Microservicios facilitan la extensión de funcionalidad sin modificar componentes existentes.
- Liskov Substitution: Menos relevante para arquitecturas orientadas a servicios, pero crucial en objetos y componentes.
- Interface Segregation y Dependency Inversion: Altamente soportados en estilos que promueven la modularidad y la independencia.

- Matriz de análisis de Atributos de Calidad vs Temas

Temas / Principios	Rendimiento	Escalabilidad	Mantenibilidad	Seguridad	Flexibilidad
Componentes Independientes	Media	Alta	Alta	Media	Alta
CQRS	Alta	Alta	Media	Alta	Media
Microservicios	Alta	Alta	Alta	Alta	Alta
Servicios	Media	Alta	Media	Alta	Alta
SOA	Media	Baja	Baja	Media	Alta

Descripción:

- Rendimiento y Escalabilidad: Microservicios y CQRS son excelentes en escalabilidad y pueden ser optimizados para alto rendimiento.
- Mantenibilidad: Componentes Independientes y Microservicios sobresalen por su facilidad de mantenimiento.
- Seguridad: Debe ser incorporada en cada capa, pero SOA y Microservicios proporcionan buenas bases para políticas de seguridad robustas.

- Matriz de análisis de Tácticas vs Temas

Temas / Principios	Descomposición de servicios	Gestión de Eventos	Caching	Balanceo de Carga	Autenticación y Autorización
Componentes Independientes	Alta	Media	Media	Media	Media
CQRS	Media	Alta	Alta	Alta	Alta
Microservicios	Alta	Alta	Alta	Alta	Alta
Servicios	Alta	Media	Media	Alta	Alta
SOA	Media	Baja	Baja	Media	Alta

Descripción:

- Descomposición de Servicios: Microservicios y Componentes Independientes maximizan el beneficio de dividir funcionalidades.
- Gestión de Eventos y Caching: CQRS y Microservicios utilizan eventos y caching efectivamente para mejorar el rendimiento.
- Balanceo de Carga y Autenticación: Elementos críticos en cualquier arquitectura distribuida, con alta relevancia en Microservicios.

- Matriz de análisis de Mercado laboral vs Temas

Temas / Principios	Demanda	Disponibilidad de Recursos	Innovación	Estabilidad	Flexibilidad en la Carrera
Componentes Independientes	Media	Baja	Alta	Media	Alta
CQRS	Alta	Media	Alta	Media	Media
Microservicios	Alta	Alta	Alta	Alta	Alta
Servicios	Alta	Alta	Media	Alta	Media
SOA	Media	Media	Baja	Alta	Baja

Descripción:

- Demanda: Microservicios y servicios en general tienen una alta demanda debido a la necesidad de aplicaciones escalables y mantenibles.
- Disponibilidad de Recursos: Hay una gran cantidad de recursos y profesionales formados en Microservicios y tecnologías de servicio.
- Innovación: Componentes Independientes y CQRS se consideran más innovadores debido a su adaptabilidad y capacidad para manejar complejidad.
- Estabilidad y Flexibilidad en la Carrera: SOA, siendo una tecnología más madura, ofrece estabilidad, mientras que los Microservicios ofrecen altas oportunidades de flexibilidad en la carrera debido a su prevalencia en diversas aplicaciones.

Análisis Estadístico de Tecnologías:

A continuación, se va a analizar el mercado laboral con las herramientas tecnológicas escogidas del presente año:

- Python: Se encuentra dentro del top 3 de herramientas mas usadas. Sin embargo, es la primera opción para aprender a programar.

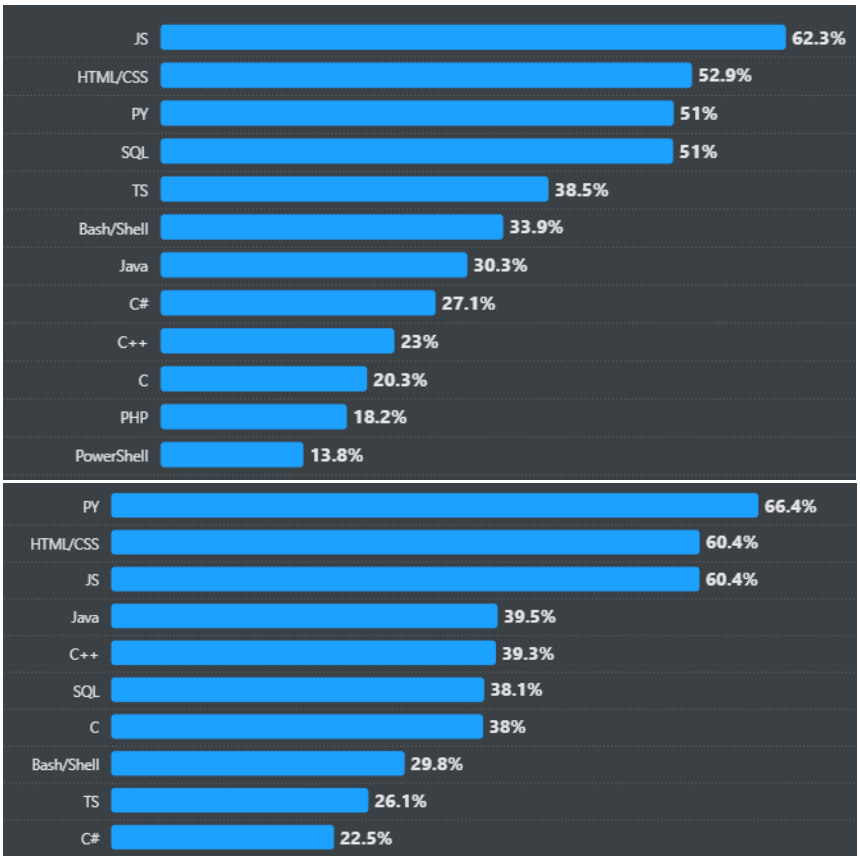


*Imagen 1-2. "**Programming, scripting, and markup languages**"Obtenido de: https://survey.stackoverflow.co/2024/technology*

- SQL: Se encuentra dentro del top 5 de herramientas más usadas. Sin embargo, es la segunda opción más usada por desarrolladores profesionales.

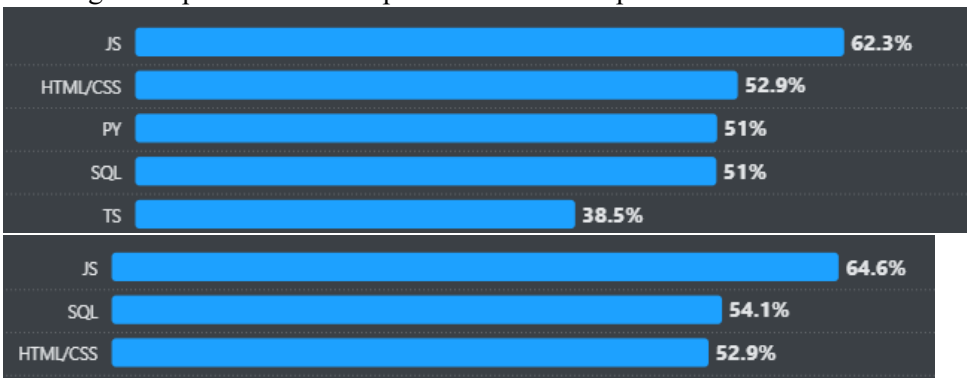


*Imagen 3-4. "**Programming, scripting, and markup languages**"Obtenido de: https://survey.stackoverflow.co/2024/technology*

- MySQL: Es la segunda herramienta más usada por desarrolladores profesionales. Y la primera para aprender a programar.

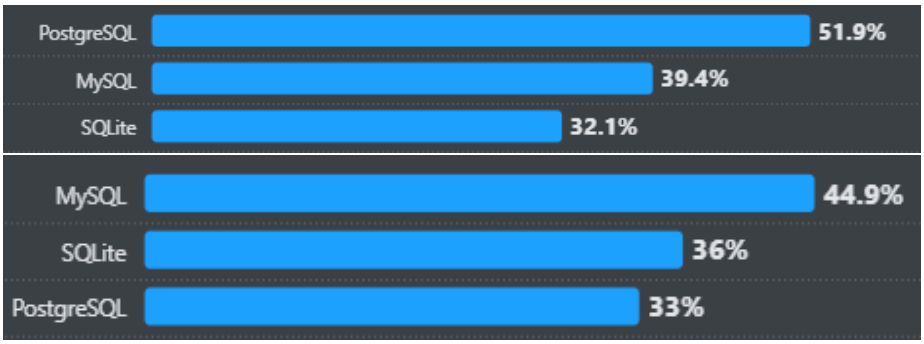


*Imagen 5-6. "**Databases**"Obtenido de: https://survey.stackoverflow.co/2024/technology*

- Flask: Se encuentra entre el top 10 de frameworks orientados a web.

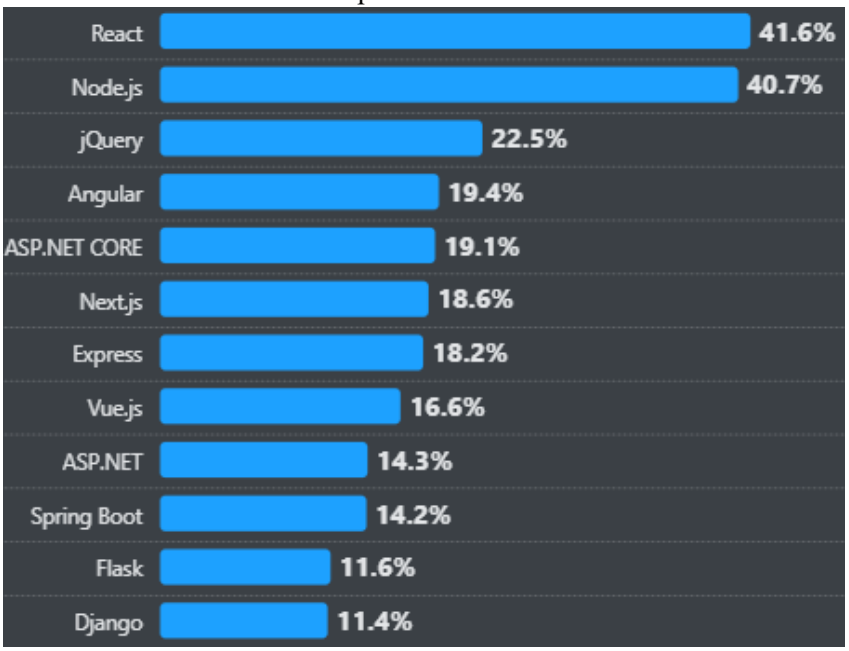


*Imagen 7. "**Web frameworks and technologies**"Obtenido de: https://survey.stackoverflow.co/2024/technology*

- Gatsby: Es de las herramientas menos usadas actualmente. Se encuentra en los últimos puestos por desarrolladores profesionales.

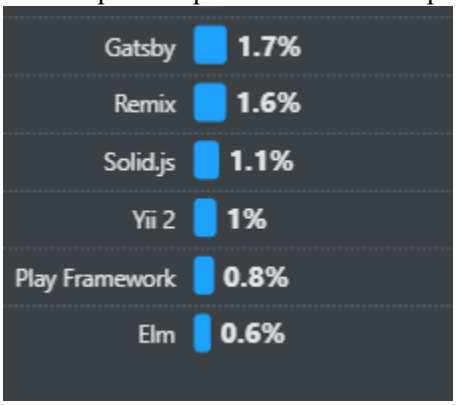


*Imagen 8. "**Web frameworks and technologies**"Obtenido de: https://survey.stackoverflow.co/2024/technology*

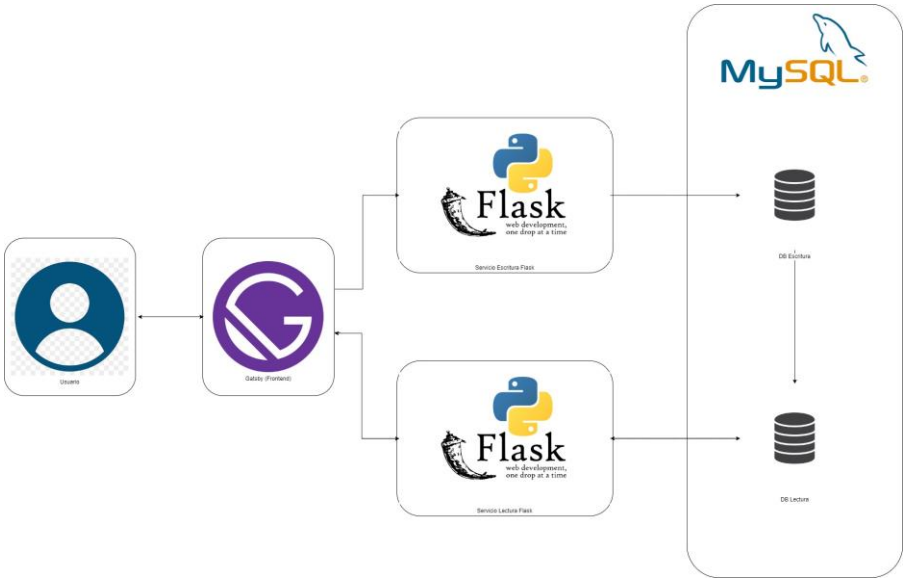
- Matriz de análisis de Patrones laboral vs Temas

Temas / Principios	Trabajo en Equipo	Trabajo Remoto	Desarrollo Ágil	Escalabilidad de Proyecto	Requerimientos Cambiantes
Componentes Independientes	Alta	Alta	Alta	Alta	Alta
CQRS	Alta	Media	Alta	Alta	Alta
Microservicios	Alta	Alta	Alta	Alta	Alta
Servicios	Alta	Alta	Alta	Alta	Media
SOA	Media	Media	Baja	Media	Baja

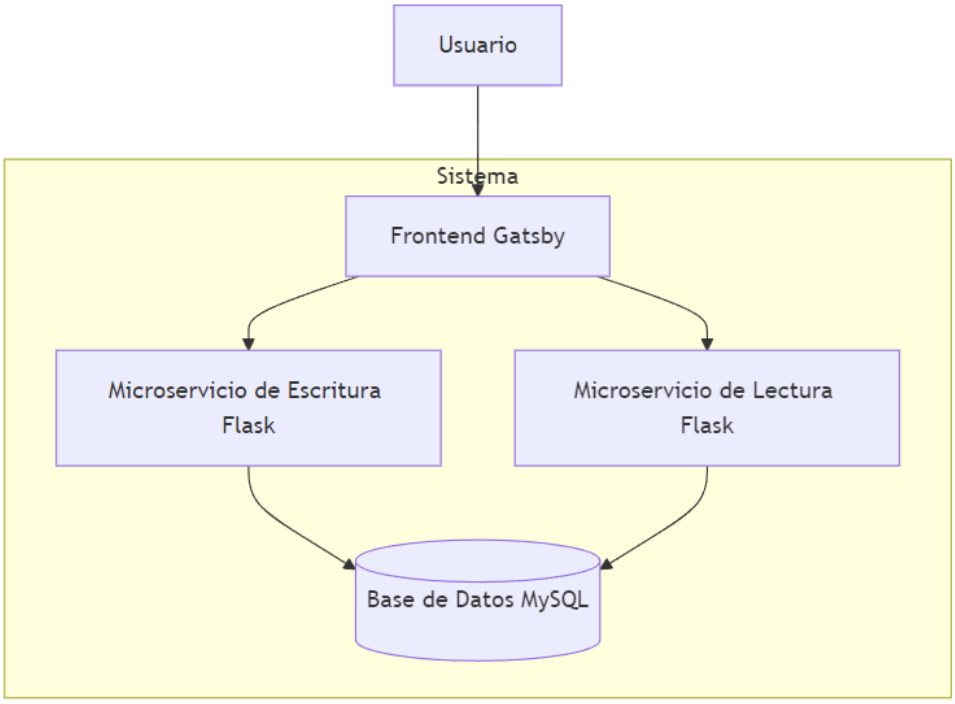
Descripción:

- Trabajo en Equipo y Trabajo Remoto: La mayoría de estos estilos arquitectónicos facilitan el trabajo en equipo y remoto debido a su naturaleza modular y desacoplada.
- Desarrollo Ágil: Estilos como Microservicios y CQRS son ideales para entornos ágiles donde los requisitos pueden cambiar rápidamente y se necesita iterar rápidamente.
- Escalabilidad de Proyectos y Requerimientos Cambiantes: Microservicios ofrecen la mejor escalabilidad y adaptabilidad a cambios, haciéndolos populares en proyectos grandes y dinámicos.

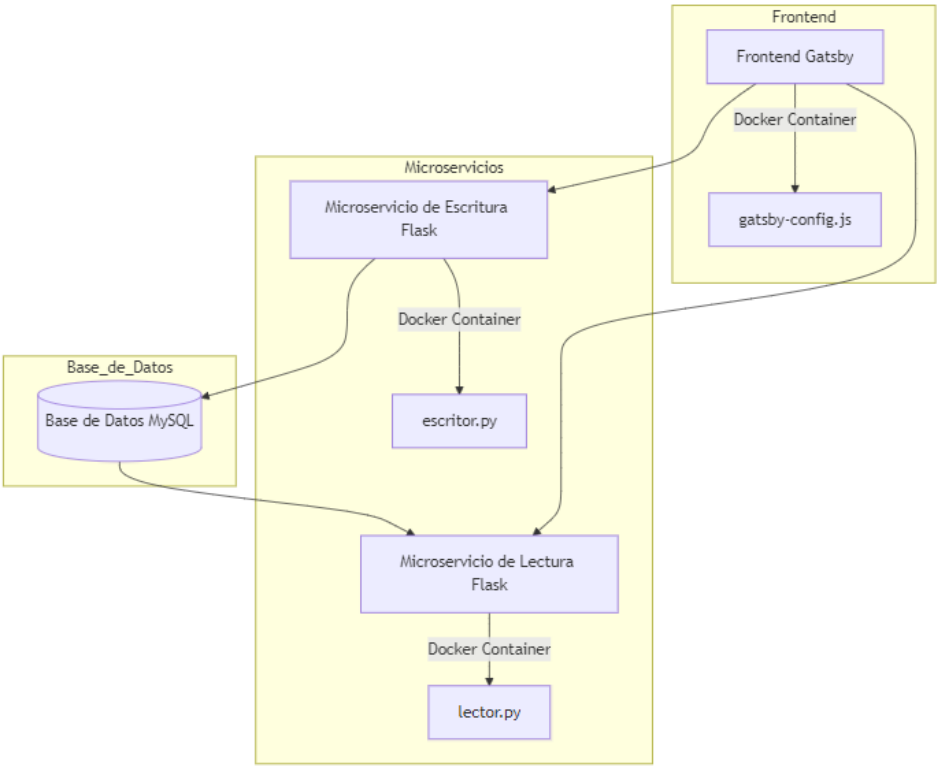
8. EJEMPLO PRÁCTICO
   - Diagrama de Alto Nivel



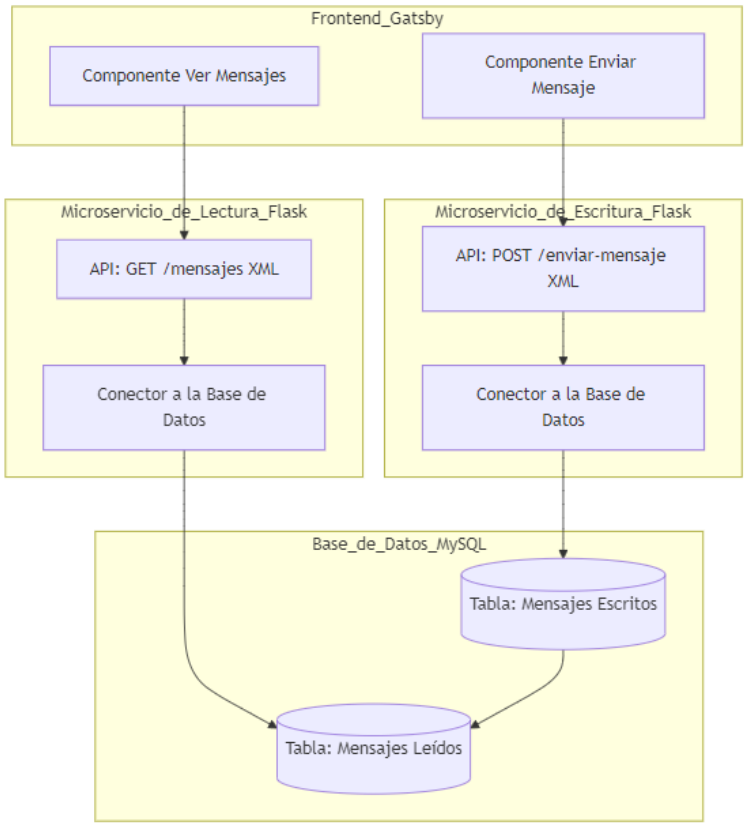
   - Diagramas C4Model
     - Contexto



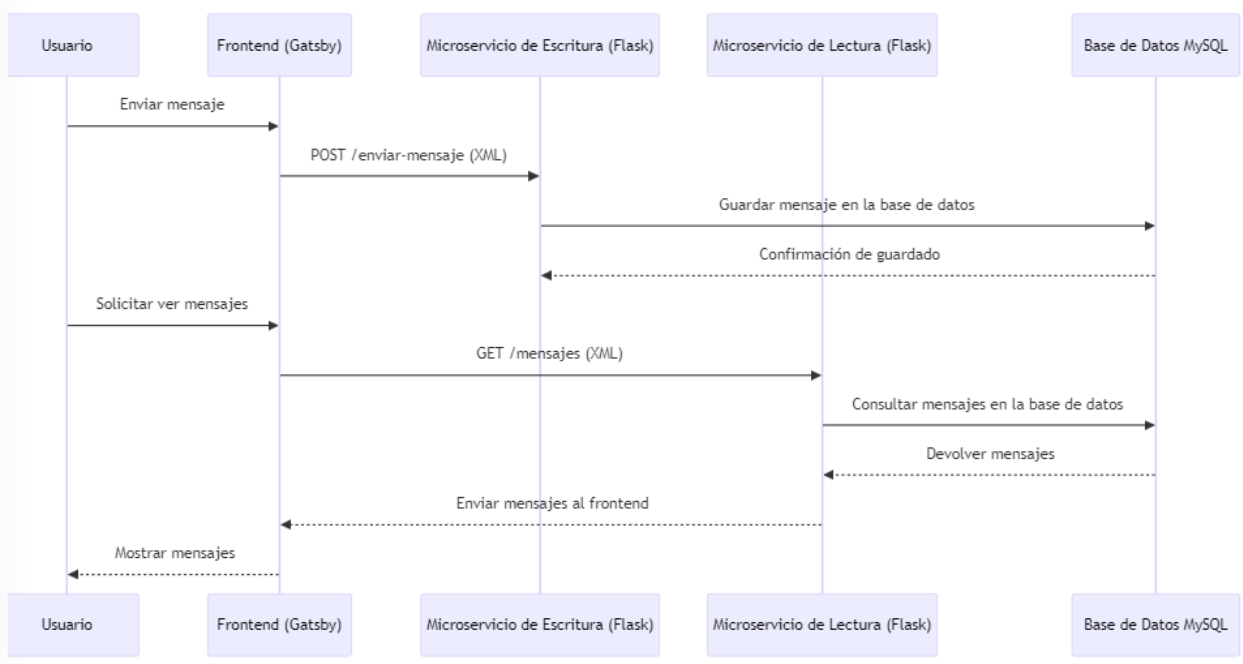
- Contenedores



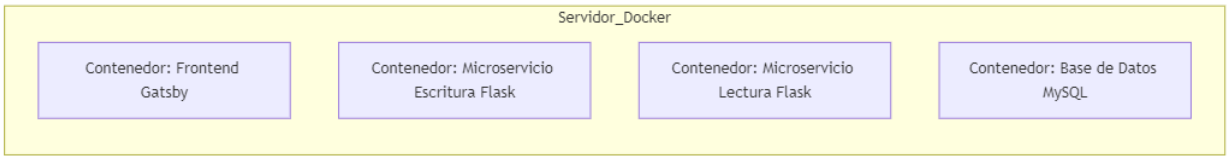
- Componentes



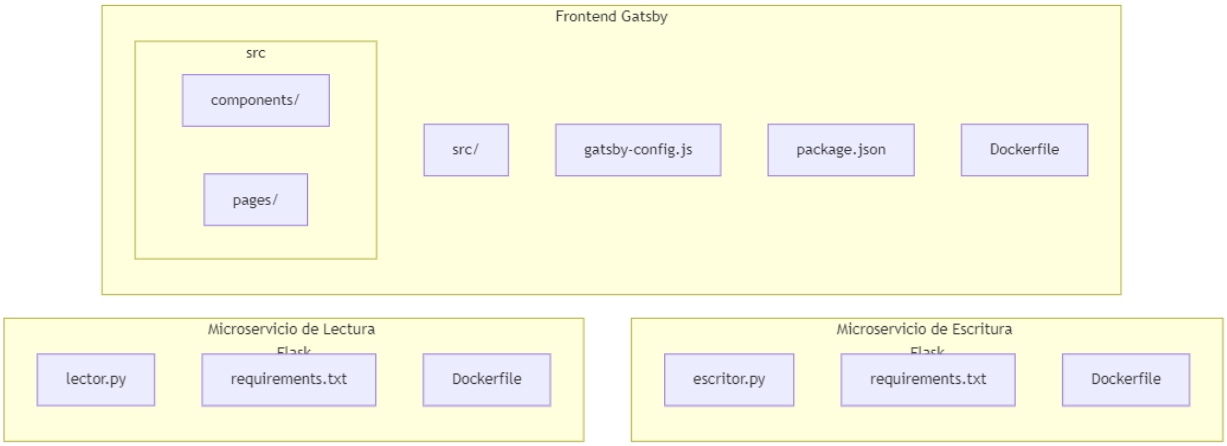
- Diagrama Dynamic C4



- Diagrama Despliegue C4



- Diagrama de Paquetes



9. REFERENCIAS

1. Aristizabal, V. (2024, marzo 1). *Arquitectura de componentes*. DEV Community. https://dev.to/vanessamarely/arquitectura-de-componentes-283p
2. *Arquitectura basada en Componentes*. (2009, abril 18). Geeks.ms. https://geeks.ms/jkpelaez/2009/04/18/arquitectura-basada-en-componentes/
3. navlaniwesr Follow Improve. (2024, junio 28). *Component-based architecture - system design*. GeeksforGeeks. https://www.geeksforgeeks.org/component-based-architecture-system-design/
4. RobBagby. (s/f). *Patrón CQRS*. Microsoft.com. Recuperado el 20 de septiembre de 2024, de https://learn.microsoft.com/es-es/azure/architecture/patterns/cqrs
5. Camargo, J. R. (24 de julio de 2023). Patrón CQRS: qué es y cómo implementarlo dentro del framework Axon. Pragma.co. https://www.pragma.co/es/blog/patron-cqrs-que-es-y-como-implementarlo-dentro-del-framework-axon
6. Ibáñez, Y. G. (2024, mayo 6). *Patrones de arquitectura de microservicios: comunicación y…*. Paradigmadigital.com. https://www.paradigmadigital.com/dev/patrones-arquitectura-microservicios-comunicacion-coordinacion-cqrs-bff-outbox/
7. Mairaw, J. G. E. (2023, marzo 23). *Aplicar los patrones CQRS y DDD simplificados en un microservicio - .NET*. Microsoft.com. https://learn.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/apply-simplified-microservice-cqrs-ddd-patterns
8. *Microservicios*. (n.d.). Amazon.com. Retrieved September 20, 2024, from https://aws.amazon.com/es/microservices/
9. *Microservicios y arquitectura de microservicios*. (n.d.). Intel.La. Retrieved September 20, 2024, from https://www.intel.la/content/www/xl/es/cloud-computing/microservices.html
10. ¿Qué son los microservicios? (2023, July 17). *Ibm.com*. https://www.ibm.com/mx-es/topics/microservices
11. Marta. (2023, June 9). Pros y contras de usar microservicios. *ACKstorm*. https://www.ackstorm.com/blog/pros-y-contras-para-usar-microservicios/

12. *Arquitectura de Microservicios: Características, ventajas y ejemplos reales*. (2024, February 21). HostZealot. https://es.hostzealot.com/blog/about-solutions/arquitectura-de-microservicios-caracteristicas-ventajas-y-ejemplos-reales
13. Nicolás, J. (13 de junio de 2024). *Microservicios 101. Qué Son, Ventajas, Desventajas y Comparativa con Arquitecturas Monolíticas*. Juannicolas.Eu. https://www.juannicolas.eu/microservicios-101-que-son-ventajas-desventajas/
14. Nuñez, E. A. (2016, April 19). Qué son Microservicios y ejemplos reales de uso. *Openwebinars.net*. https://openwebinars.net/blog/microservicios-que-son/
15. Fowler, M. (2015, July 1). *Microservice Trade-Offs*. Martinfowler.com. https://martinfowler.com/articles/microservice-trade-offs.html
16. *Microservicios: ¿Qué son y qué tipos existen? ¿Cómo funciona su arquitectura?* (2023, August 22). Whitestack. https://whitestack.com/es/blog/microservicios/
17. *¿Qué es la arquitectura orientada a servicios (SOA)?* (n.d.). Amazon.com. Retrieved September 20, 2024, from https://aws.amazon.com/es/what-is/service-oriented-architecture/
18. *¿Qué es la arquitectura orientada a los servicios (SOA)?* (4 de agosto de 2023). Redhat.com. https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture
19. ¿Qué es la SOA (arquitectura orientada a servicios)? (2023, April 28). *Ibm.com*. https://www.ibm.com/mx-es/topics/soa
20. Soafumc, P. (2016, August 21). *VENTAJAS Y DESVENTAJAS DEL*. SOA. https://yamilpo.wordpress.com/2016/08/21/ventajas-y-desventajas-del-soa/
21. Ejemplos de arquitectura SOA. (2023, June 26). *Revista Cloud*. https://revistacloud.com/ejemplos-de-arquitectura-soa/
22. Beservices, E. P. (2021, December 13). ¿Qué es la arquitectura orientada a servicios (SOA)? ¿Cuáles son los beneficios en la empresa? *Beservices.es*. https://blog.beservices.es/blog/que-es-la-arquitectura-orientada-servicios-soa-cuales-son-los-beneficios-en-la-empresa
23. *Historical Background*. (n.d.). Soa.org. Retrieved September 20, 2024, from https://www.soa.org/about/historical-background/
24. *Technology*. (s/f). Stackoverflow.Co. https://survey.stackoverflow.co/2024/technology