PONTIFICIA UNIVERSIDAD JAVERIANA

ARQUITECTURA DE SOFTWARE

TALLER 2

Presentado por:

Julián Carrillo

Juan Andrés Mejía

Fabián Zapata Bonivento

Marzo 24 de 2023

Marco Teórico

SOA (Service Oriented Architecture)

a. Historia y descripción:

El desarrollo de la arquitectura SOA nació gracias a la preocupación en la década de los 90’s por poder hacer sistemas distribuidos pues, la necesidad de tener la información en un sistema replicado en varias máquinas era muy atractivo para cualquier solución tecnológica.

Como bien fue mencionado anteriormente, la arquitectura SOA está basada en los principios de una arquitectura distribuida. Para entender un sistema distribuido es preciso ver el siguiente concepto y sus subconceptos:

Servicio

Según Thomas Erl (2005), un servicio es una "unidad lógica de funcionalidad que se expone a través de una interfaz bien definida y que se puede invocar mediante mensajes". Según Thomas Erl, la arquitectura SOA es una arquitectura de software que se basa en la idea de descomponer los sistemas de software en servicios individuales y reutilizables que se pueden utilizar en diferentes aplicaciones (Erl, 2005). En otras palabras, el estilo arquitectónico SOA permite conectar de manera distribuida e independientes a los servicios.

Erl también destaca que un servicio debe ser independiente de la plataforma y el lenguaje de programación utilizados, ser diseñado para ser escalable, confiable y seguro, cumplir con los requisitos del negocio y ser capaz de adaptarse a cambios en el entorno empresarial. Resumiendo, un servicio es un componente perteneciente a un sistema que está en capacidades de llevar a cabo una tarea.

Implementación de servicios

La implementación de servicios es la parte lógica que realiza la función específica asignada al servicio, en otras palabras, el código.

Contrato de servicio

El contrato de servicio es la definición de la naturaleza con la que el servicio se comportará y también los términos y condiciones con los que estará en juego constantemente.

Interfaz del servicio

La interfaz es la que permite a un servicio conectarse con otros servicios, en ella se define la forma en la que se puede invocar el servicio para llevar a cabo alguna tarea como transferencia de datos o cualquier actividad. Las interfaces reducen la dependencia entre los servicios.

Consumidor de servicios

El consumidor de servicios como bien dice su nombre es aquel que se comunica con los servicios y los pone a funcionar, la forma en la que este lo hace se define en el contrato del servicio.

Proveedor de servicios

El proveedor de servicios se encarga de crear y mantener los servicios que se quieren utilizar.

Principios básicos:

1. Interoperabilidad
2. Acoplamiento flexible
3. Abstracción
4. Granularidad

Funcionamiento

Según lo anterior, sabiendo que es un servicio, podemos entender finalmente cómo funciona la arquitectura orientada a servicios. Los servicios funcionan de forma independiente y son ellos, los que se encargan de realizar la transferencia de datos y las diferentes funcionalidades. Una vez procesada la información y ejecutadas las actividades, es devuelta una respuesta.

Los protocolos de comunicación utilizados dentro de la arquitectura son JMS (java message system), SOAP y REST.

Prime Faces:

a. Historia y descripción:

El framework prime faces es un framework enfocado en el desarrollo de interfaces Front-end de aplicaciones web. El framework está basado en JFS(JavaServerFaces). Fue creado y diseñado por la empresa tecnológica PrimeTek en el año 2009. EL objetivo principal del software es brindar a

los desarrolladores Java una herramienta para facilitar la tarea de creación de interfaces de usuario. Inicialmente la biblioteca contaba con pocos componentes básicos de diseño web donde se incluían diferentes botones, barras de progreso y menús.

A día de hoy la herramienta se ha posicionado bien en el mercado de herramientas de desarrollo front gracias a su enfoque en la personalización de apariencia de las páginas web.

Algunos de los atributos de calidad que refleja PrimeFaces son:

- Usabilidad: La usabilidad se ve reflejada en la amplia variedad de componentes de UI que mejoran la experiencia del usuario
- Eficiencia: La eficiencia se da dado a que Primefaces utiliza tecnologías y metodologías de optimización de rendimiento para garantizar comunicaciones y transferencia de información de manera ágil.
- Mantenibilidad: Gracias a su modularidad el software es fácil de actualizar o mantener.
- Flexibilidad: La amplia gama de componentes permiten el uso de tecnologías y personalización de los diseños.
- Integración: PrimeFaces es integrable fácilmente con muchas tecnologías y frameworks como Java EE, JSF, Spring y demás.

Jakarta

a. Historia y descripción:

El framework Jakarta fue un desarrollo de tipo open source que inicialmente hacia parte del proyecto de desarrollo Apache en el año 1999. Con el progreso de los años, el proyecto Jakarta integró los proveedores de serverlets y paginas JavaServer Pages (JSP) y, cambo su nombre a Apache Tomcat mientras que el resto de los proyectos resultantes migraron a un apartado llamado Jakarta EE.

Java EE provee APIs estándar para facilitar el desarrollo de aplicaciones empresariales donde se acoplan tecnologías de serverlets, Beans y Java Message.

Algunos de los atributos de calidad que refleja Jakarta son:

- Seguridad: La tecnología permite la implementación de varios filtros y sistemas que garantizan la seguridad como autenticación, cifrados y hasta gestión de contraseñas.
- Fiabilidad: Mecanismos de tolerancia a fallos y gestión de transacciones hacen de Jakarta un sistema con alta fiabilidad
- Mantenibilidad: La modularidad del software permite el mantenimiento de la misma
- Interoperabilidad: Jakarta utiliza protocolos de comunicación comunes que garantizan la interoperabilidad entre sistemas.

Payara

a. Historia y descripción

Payara Server es un servidor de aplicaciones Java de código abierto que se basa en el código del servidor GlassFish de Oracle. En 2015, Oracle anunció que dejaría de ofrecer soporte comercial para GlassFish, lo que llevó a C2B2 Consulting a crear una bifurcación del código y renombrarlo como Payara Server.

La primera versión de Payara Server se lanzó en 2016, y desde entonces ha sido mejorada por la comunidad de Payara, agregando nuevas características y corrigiendo errores. Payara Server se ha convertido en una alternativa popular a GlassFish, especialmente en entornos empresariales que necesitan soporte comercial y actualizaciones regulares.

En la actualidad, Payara Server es utilizado por una variedad de empresas y organizaciones en todo el mundo, y Payara también ofrece herramientas adicionales, como Payara Micro y Payara Enterprise, para satisfacer las necesidades específicas de los clientes. La historia de Payara Server destaca la importancia del soporte comercial y la comunidad de desarrolladores en la creación y mantenimiento de servidores de aplicaciones Java de código abierto de calidad empresarial.

MySQL

a. Historia y descripción:

MySQL es un motor y sistema de bases de datos relacionales de código abierto que fue desarrollado al principio en la década de los 90s, el proyecto desde un inicio tuvo la iniciativa de

ser open source. El proyecto fue llevado de la mano por la empresa sueca MySQL AB. La idea desde un comienzo era crear un sistema de base de datos con alto rendimiento y escalabilidad. Mas adelante el proyecto fue adquirido por la empresa Sun Microsystems para finalmente acabar en manos del magnate tecnológico Oracle.

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) utilizado para almacenar y administrar datos en grandes cantidades. El sistema está basado en el lenguaje de consulta SQL donde se interactúa con la base de datos. Es integrable con una amplia variedad de lenguajes de programación y tecnologías web.

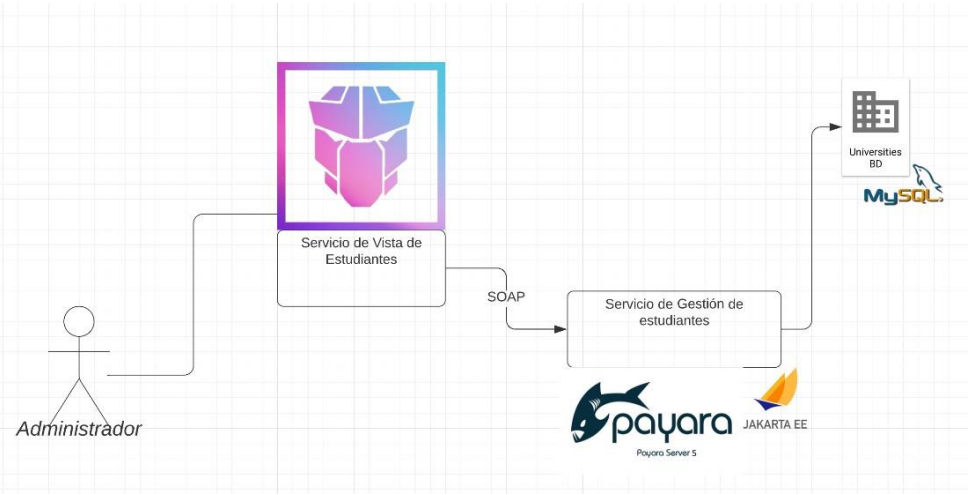
Algunos de los atributos de calidad que MySQL proporciona son:

- Fiabilidad: Al ser reconocido como una plataforma estable y confiable, esta cuenta con una alta capacidad de recuperación después de una falla del sistema y, además, maneja con mucha eficacia grandes volúmenes de datos.
- Escalabilidad: Plataforma altamente escalable, es decir puede crecer y adaptarse a las necesidades dependiendo la organización y como estas van cambiado en el transcurso del tiempo. Asimismo, es capaz de manejar grandes cantidades de datos y alta concurrencia en materia de conectividad, lo que resulta ideal para las aplicaciones de alta carga.
- Rendimiento: La tecnología se destaca por su alto rendimiento, particularmente en el caso de aplicaciones web y aplicaciones que requieren rápido acceso a grandes cantidades de datos. Así, esta es capaz de manejar consultas complejas y acceso a datos de manera eficiente.
- Seguridad: ofrece opciones de seguridad sólidas, entre las cuales se encuentran la autenticación de usuario y la encriptación de datos. Adicionalmente, permite que los administradores de bases de datos controlen el acceso a los datos y logren proteger la información confidencial.
- Facilidad de uso: Se trata de una herramienta fácil de instalar y configurar, lo que lo hace accesible a usuarios novatos. Igualmente, cuenta con una interfaz de usuario amigable y una amplia oferta de documentación disponible en línea.

Representación del sistema

Arquitectura de alto nivel

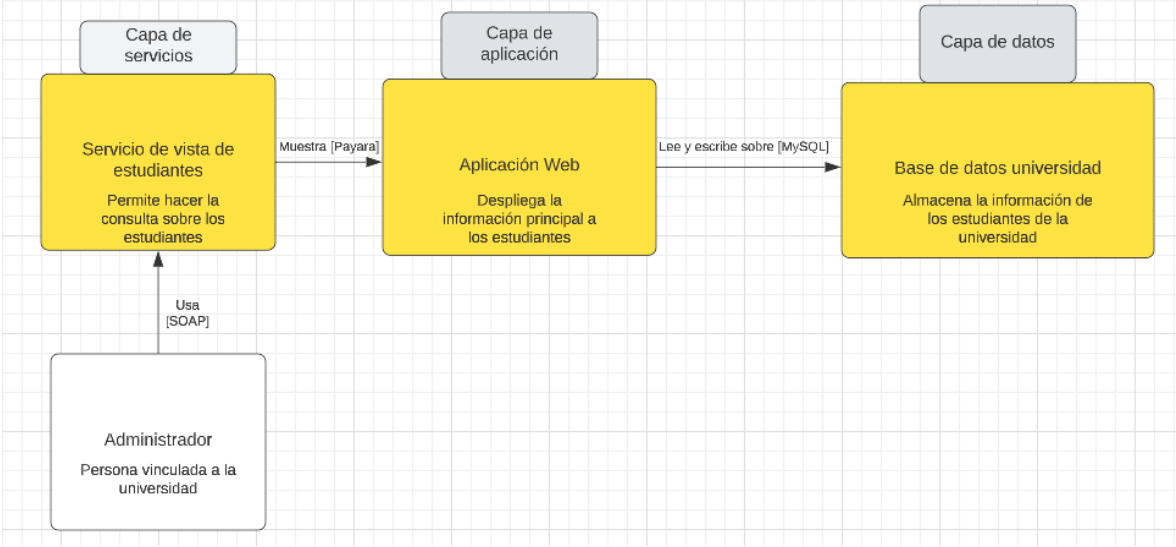
El diagrama de alto nivel busca dar un entendimiento general de la arquitectura, ilustrando y denotando las tecnologías utilizadas. Como podemos ver, se encuentran principalmente 3 componentes atados a un actor "Administrador", En la parte izquierda vemos al actor conectado al "Servicio de vista de estudiantes" este, es un Client Web Service que se conecta por medio del protocolo SOAP. Unido a él, el servicio a consumir, Servicio de gestión de estudiantes. Este servicio Jakarta EE esta soportado por un servidor de aplicaciones Payara. Dentro del servicio se encuentra el CRUD que posibilita la extracción de la información con la base de datos MySQL a la que se encuentra atado.



Arquitectura de alto nivel

Arquitectura bajo nivel

En la siguiente arquitectura se puede apreciar la composición que tiene la plataforma de gestión de estudiantes, inicialmente proporciona los servicios con los que va a contar, los componentes y la funcionalidad básica con la que se va a regir el sistema, adicionalmente facilita la vista frente a la distribución de sus componentes, así como, las capas por las cuales están definido el sistema. A partir de lo anterior se tiene un administrador que actúa como un Client Web Service que accede a la información de los estudiantes por medio del protocolo SOAP y prontamente es desplegado por el Servidor contenido en Payara para luego acceder a los datos en MySQL.



Arquitectura de bajo nivel

Diagrama de clases

En el siguiente diagrama se muestran las clases utilizadas en el sistema, desde la clase entidad *Students* al *StudentsWebService* que contiene las operaciones contenidas en el CRUD. Los patrones del diseño de la arquitectura SOA exigen la instanciación de las clases facade en donde implementan aquellas funciones de ETL, o sea abstracción e inserción de información de la base de datos.

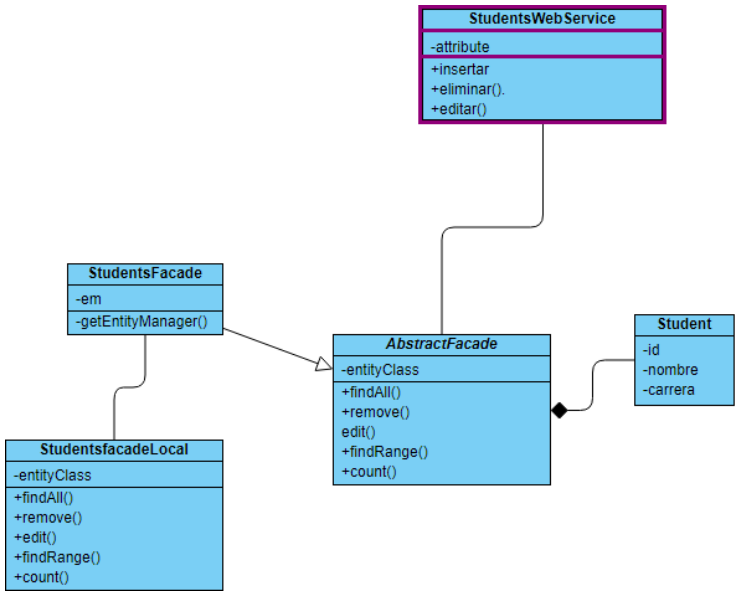


Diagrama de clases del sistema

Desventajas de las tecnologías

Jakarta EE: Puede tener problemas de escalabilidad sino se presenta la configuración adecuada para el servidor de aplicaciones, lo que puede llegar a afectar el rendimiento y puede representar problemas de vulnerabilidad. Igualmente, puede presentar problemas de seguridad sino se aplican los parches o se establecen unos parámetros que permitan fortalecer su componente de seguridad.

Primefaces: puede presentar problemas de compatibilidad con diferentes navegadores web lo que puede afectar su rendimiento y también su visualización en los componentes del servicio web. Si se aplican funcionalidades que sobrecarguen la capacidad del sistema, afecta de manera considerable el despliegue de la página.

MySQL: Si se utilizan consultas complejas en la base de datos puede llegar a tener tiempos de respuesta bastante lentos que afecten a la integración y las consultas que sean realizadas en la base de datos. Por lo tanto, es buena práctica mantener bases de datos no tan grandes y que puedan estar distribuidas, para que las consultas realizadas no afecten el rendimiento, ya que esta es una principal característica que posee MySQL.

Principios Solid

Los principios Solid como conjuntos de principios de diseño de software son aplicables en diferentes niveles de las arquitecturas, incluyendo a las arquitecturas SOA, por lo que se pueden aplicar dichos principios de la siguiente manera:

- Principio de responsabilidad única (SRP): este principio establece que una clase o componente debe tener una única responsabilidad o motivo de cambio. En el contexto de la arquitectura SOA, esto significa que los servicios deben tener una sola responsabilidad y no deben contener funcionalidades ajenas a su propósito principal. Por ejemplo, el servicio de gestión de estudiantes no debe ser responsable de procesar los pagos.

- Principio abierto/cerrado (OCP): este principio establece que las unidades de software (clases, módulos, funciones, etc.) están abiertas a la extensión, pero cerradas a la modificación. En el contexto de la arquitectura SOA, esto significa que los servicios deben

ser fáciles de extender a nuevas funciones o adaptarse a nuevos requisitos sin cambiar el servicio original. En el caso del sistema propuesto, el servicio de estudiantes debe estar en capacidad de extender la información de estos en caso de ser necesario.

- Principio de sustitución de Liskov (LSP): este principio establece que una subclase debe poder ser reemplazada por su superclase sin afectar la integridad del sistema. En la arquitectura SOA, esto significa que un servicio basado en otro servicio debe poder reemplazarlo sin interrumpir el servicio que lo utiliza. Aplicándolo al sistema de gestión de estudiantes, en caso de que se tuviese un servicio extra y este servicio dependiera de la gestión de estudiantes, este servicio adicional podría ser reemplazado sin que afecte el funcionamiento del sistema.

- Principio de aislamiento de interfaz (ISP): este principio establece que las interfaces de un componente deben estar aisladas para que los clientes dependan solo de las interfaces que necesitan. En la arquitectura SOA, esto significa que los servicios deben tener sus propias interfaces para cada función que brindan, lo que facilita la elección de los servicios necesarios para construir la solución general. El servicio de gestión de estudiantes cuenta con su propia interfaz en donde se va a desplegar la información solicitada.

- 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 que ambos deben depender de abstracciones. En la arquitectura SOA, esto significa que los servicios deben depender de abstracciones (como interfaces) de implementaciones concretas de otros servicios. Esto permite la independencia del servicio y mejora la escalabilidad del sistema. El servicio de gestión de estudiantes no depende de un servicio de bajo nivel para acceder a la base de datos, pero sí de la abstracción del Web Cliente SOAP que tiene definida una interfaz para acceder a los datos

Arquitectura

La arquitectura orientada a servicios tiene fortalezas y debilidades, a continuación, una tabla de los atributos de calidad que más refleja.

Atributos de calidad	**Descripción**
Modularidad	Una arquitectura SOA se basa en la creación de servicios modulares y reutilizables que pueden ser utilizados por varios sistemas y aplicaciones. Esto puede aumentar la modularidad del sistema y facilitar el mantenimiento y la escalabilidad.
Flexibilidad	Los servicios de una arquitectura SOA se pueden cambiar y actualizar sin afectar a otros servicios. Esto permite que el sistema sea flexible y adaptable a cambios en los requisitos del negocio.
Interoperabilidad	La arquitectura SOA se basa en el uso de estándares abiertos y protocolos comunes para la comunicación entre servicios. Esto puede mejorar la interoperabilidad entre sistemas heterogéneos y reducir la complejidad de la integración de sistemas
Escalabilidad	Una arquitectura SOA puede permitir la escalabilidad horizontal y vertical de los servicios. Los servicios pueden ser escalados de forma independiente para satisfacer las demandas de carga.

Reutilización	Puede utilizar servicios que ya hayan sido implementados en otras aplicaciones, de esta forma se puede garantizar su reducción de tiempo de desarrollo en nuevos servicios.

Casos de estudio

Para este ítem, la arquitectura SOA puede ser aplicable bajo diferentes contextos, a continuación, se presentan diferentes casos de estudios

- Integración de sistemas comerciales: SOA se usa a menudo para integrar sistemas comerciales existentes, como la gestión de relaciones con el cliente (CRM), la gestión de la cadena de suministro (SCM), los recursos humanos (HR) y los sistemas de TI, y la gestión de inventario. La arquitectura SOA permite que estos sistemas se integren más fácilmente, lo que ayuda a mejorar la eficiencia y la productividad empresarial.

- Servicios Web: Los servicios Web son una de las aplicaciones más importantes de SOA. Las empresas utilizan los servicios en línea para ofrecer una amplia gama de actividades en Internet, incluidos pagos en línea, reservas de vuelos, reservas de habitaciones de hotel, compras en línea y más.

- Dispositivos móviles: SOA también se utiliza para crear aplicaciones móviles que se conectan a servicios empresariales en la nube. Las aplicaciones móviles pueden usar servicios web para acceder a datos comerciales, como datos de inventario, datos de clientes y otros recursos.

- Sistemas de administración de contenido: SOA se utiliza para crear sistemas de administración de contenido que permiten a las empresas administrar y compartir

información, como documentos, imágenes, videos y otros recursos digitales. La arquitectura SOA permite a los usuarios acceder a estos recursos de manera más fácil y rápida. Integración de

- Servicios en la nube: SOA también se utiliza para integrar servicios en la nube, como almacenamiento en la nube, procesamiento en la nube, análisis de datos en la nube y otros servicios empresariales en la nube.

Casos de Éxito de uso de SOA

SOA es una arquitectura que hoy en día es utilizada en diferentes ambientes empresariales, donde la implementación es esencial para las empresas líderes a lo largo de la industria, Algunos ejemplos de empresas que han adoptado la arquitectura orientada a servicios (SOA) son Cisco, IBM, Amazon, entre otros. Cisco, por ejemplo, implementó SOA para garantizar que la experiencia de pedidos de productos fuera uniforme en todos los canales, exponiendo los procesos de pedidos como servicios que las divisiones, adquisiciones y socios de negocios de Cisco podrían incorporar a sus sitios web. Esto les permitió integrar los servicios de pedidos en su infraestructura existente, mejorar la eficiencia y reducir costos.

Otro ejemplo de aplicación exitosa de la arquitectura orientada a servicios (SOA) es Independence Blue Cross (IBC) de Filadelfia. IBC implementó SOA para garantizar que los diferentes componentes que gestionan los datos de los pacientes, como los agentes del servicio al cliente,

consultas médicas y usuarios del sitio web de IBC, trabajaran con la misma fuente de datos, lo que se conoce como una "única fuente de datos". Esto permitió a IBC mejorar la calidad y consistencia de los datos de los pacientes, aumentar la eficiencia y reducir los costos operativos. Además, la arquitectura SOA permitió una mayor flexibilidad y capacidad de integración, lo que les permitió adaptarse a los cambios en el mercado de la atención médica y brindar un mejor servicio al cliente.

Conclusión

Finalmente, después de haber analizado los componentes de cada una de las tecnologías y la arquitectura se puede establecer que la arquitectura SOA es una metodología de diseño de sistemas empresariales que promueve la modularidad, reutilización y flexibilidad de los servicios. Servicios en los que se pueden integrar tecnologías como Primefaces, Jakarta EE, y MySQL. En donde cada una de estas herramientas proporciona distintas capacidades y variedades para implementar servicios web, que al mismo tiempo se puede evidenciar la escalabilidad, compatibilidad y seguridad al momento de querer gestionar información o interfaces, según lo requiera y según la tecnología que se esté empleando. Sin embargo, es importante considerar cuidadosamente la configuración y el diseño de cada una de estas tecnologías para evitar problemas de rendimiento, seguridad y complejidad.

Lecciones aprendidas

- La implementación de la arquitectura SOA facilita la distribución de servicios y ayuda a que puedan ser reutilizados, mejorando el rendimiento y la eficiencia que puede tener el código, así como su extensión.
- La implementación de los Frameworks mencionados en el documento pueden llegar a representar cierto grado de complejidad debido a problemas de versiones que puede haber entre sí, generando problemas de compatibilidad, prolongando su despliegue. Esto se debe a la antigüedad de la arquitectura y los componentes utilizados en ella.
- Es importante separar las responsabilidades de cada uno de estos ya que es un factor demandante de la arquitectura SOA, en que se deba centrar PrimeFaces, Jakarta EE y MySQL para el montaje efectivo.
- Es importante escoger las tecnologías necesarias para poder hacer la integración que se necesita para el despliegue final que requiere el sistema, ya que existen tecnologías que facilitan su funcionamiento.

TOKEN caso práctico:

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

Bibliografía:

- Erl, T. (2005). Service-oriented architecture: concepts, technology, and design. Prentice Hall PTR.
- Sprott, D., & Wilkes, L. (2004). Understanding service-oriented architecture. *The Architecture Journal*, *1*(1), 10-17.
- Jakarta EE (S.f). *JAKARTA EE: AN OPEN SOURCE FRAMEWORK FOR DEVELOPING CLOUD NATIVE JAVA APPLICATIONS recuperado de: https://jakarta.ee/about/jakarta-ee/*
- MySQL (S.f). *About MySQL* recuperado de: https://www.mysql.com/about/
- MySQL (S.f). *About Documentation* recuperado de: https://dev.mysql.com/doc/
- Cuervo, V. (2021, March 2). *¿Qué es jakarta ee?* Arquitecto IT. Retrieved March 24, 2023, from https://www.arquitectoit.com/java/que-es-jakarta-ee/
- *Primefaces configuration - javatpoint.* www.javatpoint.com. (n.d.). Retrieved March 24, 2023, from https://www.javatpoint.com/primefaces-configuration
- Chakray. (2021, February 1). *Comparativa ESB open source: ¿Cuál es mejor?* Chakray. Retrieved March 24, 2023, from https://www.chakray.com/es/comparativa-esb-open-source/
- IBM. (s.f.). Arquitectura Orientada a Servicios (SOA). Recuperado el 2 de abril de 2023, de https://www.ibm.com/mx-es/topics/soa#:~:text=Mayor%20agilidad%20empresarial%3B%20tiempo%20de,r%C3%A1pido%20en%20respuesta%20a%20nuevas.
- Vivanco, J. (2019) *El Modelo C4 de Documentación para la Arquitectura de software*, *Medium*. Medium. Available at: https://medium.com/@javiervivanco/el-modelo-c4-de-documentaci%C3%B3n-para-la-arquitectura-de-software-424704528390 (Accessed: March 24, 2023).

- Adam Bien, (2023, 2 de abril). GlassFish Became the Killer AppServer [Mensaje de blog]. Recuperado de https://www.adam-bien.com/roller/abien/entry/glassfish_became_the_killer_appserver