UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN CAMPUS 1

ACTIVIDAD. Act.1.2 Arquitectura Orientada a Servicios

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

Materia: Taller De Desarrollo 4

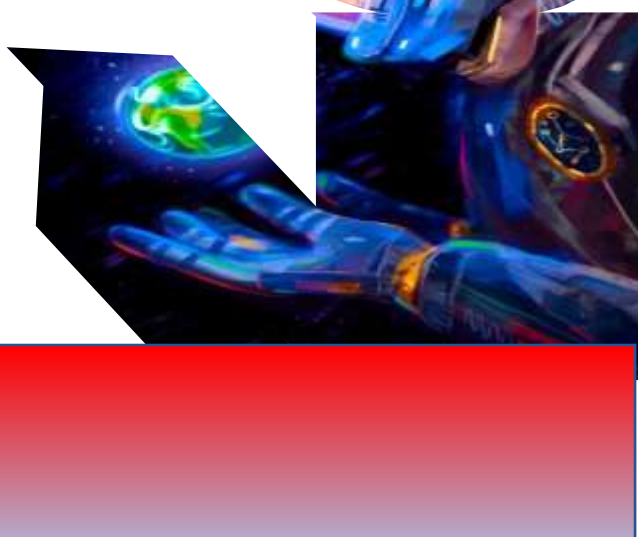
Grado y grupo: 6M

Matricula: A211120

ALUMNO: Gabriel Omar Fuentes Chacon.

DOCENTE: Mtro. Luis Gutiérrez Alfaro.

Fecha de entrega: 19/08/2023



INDICE

Introducción

Fomentando la Modularidad y la Eficiencia en el Desarrollo de Aplicaciones, en el vertiginoso mundo del desarrollo de aplicaciones, la arquitectura de microservicios se ha erigido como un paradigma que revoluciona la forma en que concebimos y construimos sistemas de software.

La clave de su éxito radica en su habilidad para comunicarse con otros microservicios mediante interfaces claras y bien definidas, a menudo a través de APIs, esta arquitectura es su capacidad para potenciar la colaboración y acelerar el desarrollo, en el ámbito de la programación, los equipos de desarrolladores pueden trabajar en sus propios microservicios sin interrumpir a los demás, facilitando un proceso de desarrollo rápido y fluido, se caracteriza por las claves de la arquitectura de Microservicios, como la descentralización, la esencia de la arquitectura de microservicios reside en la construcción de una aplicación como un conjunto de microservicios independientes. Cada uno de estos microservicios opera como una entidad autónoma, responsable de una funcionalidad específica. Esta descentralización posibilita que los equipos de desarrollo trabajen en diferentes áreas de la aplicación sin afectar a los demás microservicios, agilizando el proceso de desarrollo y mejorando la escalabilidad, la comunicación entre microservicios se lleva a cabo a través de interfaces de programación de aplicaciones (APIs) claramente definidas, además de diversidad Tecnológica, cada microservicio puede ser construido utilizando las tecnologías más apropiadas para su función específica o la escalabilidad Individual, los microservicios pueden ser escalados de manera independiente en función de su carga de trabajo y demanda y contamos resiliencia y aislamiento, la independencia de los microservicios contribuye a la resiliencia general del sistema, los benéficos que obtienes de la arquitectura de microservicios aporta una serie de ventajas que revolucionan la forma en que desarrollamos, escalamos y fortalecemos las aplicaciones como su diseño modular y la comunicación a través de interfaces bien definidas se adaptan a las cambiantes y exigentes demandas del desarrollo de aplicaciones modernas. Entre los beneficios más notables se encuentran, desarrollo y Despliegue equipos de desarrollo pueden trabajar de manera independiente en microservicios específicos, lo que agiliza el proceso de desarrollo. Los cambios y actualizaciones pueden implementarse más rápidamente, ya que solo afectarán al microservicio relevante en lugar de a toda la aplicación, además de la escalabilidad Eficiente, la capacidad de escalar microservicios individualmente según la demanda optimiza el uso de recursos y también durante esta investigación que se realizó podremos encontrar la Flexibilidad en las Tecnologías, que cada microservicio puede utilizar las tecnologías más adecuadas para su función, lo que optimiza el rendimiento y la eficiencia, este tipo de arquitectura de microservicios redefine la forma en que creamos aplicaciones, como aplicando una pieza individual de un rompecabezas de manera colectiva teniendo así aplicación en partes autónomas y comunicativas, esta arquitectura nos brinda la flexibilidad, escalabilidad y resiliencia necesarias para afrontar los retos del desarrollo de aplicaciones en la era moderna.

Desarrollo

Arquitectura Orientada a Servicios.

Arquitectura de Microservicios

La arquitectura de microservicios es como armar una aplicación de computadora en piezas pequeñas y autónomas. Cada una de estas piezas es como un mini programa llamado microservicio. Cada microservicio se concentra en hacer una sola cosa y se comunica con otros usando reglas claras, generalmente a través de APIs.

Los beneficios de esto son geniales. Por un lado, los equipos de programadores pueden trabajar en su propio microservicio sin interrumpir a los demás. Imagina si cada chef en un restaurante pudiera cocinar su plato sin molestar a los demás. Esto hace que el desarrollo sea mucho más rápido y suave.

Características clave de la arquitectura de microservicios:

Descentralización:

La arquitectura de microservicios promueve la construcción de una aplicación como un conjunto de microservicios independientes. Cada microservicio se encarga de una funcionalidad específica y opera como una entidad autónoma. Esto significa que cada microservicio puede ser desarrollado, desplegado y gestionado de forma separada. Esta descentralización permite a los equipos de desarrollo trabajar en áreas diferentes de la aplicación sin afectar a otros microservicios, lo que agiliza el proceso de desarrollo y mejora la escalabilidad.

En conjunto, estas características definen la arquitectura de microservicios como un enfoque que promueve la construcción de aplicaciones modulares, escalables y resilientes al dividir la funcionalidad en unidades autónomas y comunicarse a través de interfaces bien definidas. Cada microservicio es como una pieza individual en un rompecabezas más grande, y juntos forman una aplicación completa y colaborativa.

Comunicación por Interfaces Definidas (APIs):

Los microservicios interactúan entre sí mediante interfaces de programación de aplicaciones (APIs) claramente definidas. Estas APIs actúan como reglas de comunicación que permiten a los microservicios intercambiar información de manera estructurada y controlada. Esto posibilita que los componentes de la aplicación se conecten sin depender de su implementación interna, lo que facilita la colaboración y la integración.

Diversidad Tecnológica:

Cada microservicio puede ser construido utilizando las tecnologías más apropiadas para su función específica. En lugar de estar limitados por una sola tecnología o lenguaje de programación, los

equipos pueden elegir las herramientas que mejor se adapten a las necesidades del microservicio. Esto permite optimizar el rendimiento y la eficiencia de cada componente, ya que no hay una restricción de tecnología uniforme para toda la aplicación.

Escalabilidad Individual:

Los microservicios pueden ser escalados de forma independiente según su carga de trabajo y demanda. Esto significa que los microservicios que experimentan un aumento en la carga pueden ser escalados para manejar esa demanda específica, sin afectar a los otros componentes. Esta escalabilidad granular mejora la utilización de recursos y la capacidad de respuesta de la aplicación en su conjunto.

Resiliencia y Aislamiento:

La independencia de los microservicios contribuye a la resiliencia general del sistema. Si uno de los microservicios falla, no afectará a otros, ya que están aislados y trabajan de manera independiente. Esto limita la propagación de problemas y mejora la disponibilidad de la aplicación en general, ya que los fallos en un microservicio no tendrán un impacto dominó en el resto.

Beneficios.

La arquitectura de microservicios trae consigo ventajas que mejoran la forma en que desarrollamos, escalamos y hacemos que las aplicaciones sean fuertes, esta forma de diseñar se ajusta muy bien a las demandas cambiantes y difíciles del desarrollo de aplicaciones en la actualidad, ofrece una serie de beneficios que mejoran el proceso de desarrollo, la escalabilidad, la resiliencia y la colaboración, permiten la construcción modular y la comunicación a través de interfaces definidas, esta arquitectura se adapta bien a las necesidades cambiantes y desafiantes del desarrollo de aplicaciones modernas, los beneficios que se pueden encontrar se mencionan continuación:

Desarrollo y Despliegue Ágil:

La arquitectura de microservicios permite a los equipos de desarrollo trabajar de manera independiente en microservicios específicos. Esto acelera el proceso de desarrollo ya que los equipos pueden trabajar en paralelo sin depender unos de otros. Además, los cambios y actualizaciones pueden ser implementados de manera más rápida, ya que solo afectarán al microservicio relevante en lugar de a toda la aplicación. Esto fomenta la agilidad y la capacidad de respuesta a las demandas cambiantes del mercado.

Escalabilidad Eficiente:

La capacidad de escalar microservicios individualmente en función de la demanda optimiza el uso de recursos. Los microservicios que requieren más capacidad pueden ser escalados sin afectar a los que no lo necesitan. Esto asegura que los recursos se asignen de manera eficiente, evitando el

sobredimensionamiento o la subutilización. Como resultado, la aplicación puede manejar picos de carga sin problemas de rendimiento.

Resiliencia y Disponibilidad:

Debido a la naturaleza independiente de los microservicios, los fallos en un microservicio no afectan a los demás. Esto mejora la resiliencia del sistema, ya que los problemas aislados no propagarán fallas a través de la aplicación. La disponibilidad también se beneficia, ya que la caída de un microservicio no se traduce en una caída completa de la aplicación. Los otros microservicios pueden seguir funcionando sin problemas.

Tecnologías Adecuadas:

Cada microservicio puede elegir las tecnologías más adecuadas para su función específica. Esto permite a los equipos utilizar las herramientas más eficientes y apropiadas para cada microservicio. No hay una restricción para adoptar una única tecnología en toda la aplicación. Esto mejora el rendimiento y la eficiencia al utilizar lo mejor de cada tecnología.

Colaboración y Especialización:

Los equipos de desarrollo pueden centrarse en microservicios específicos, lo que fomenta la colaboración y la especialización. Cada equipo puede convertirse en un experto en su microservicio, lo que resulta en una mayor calidad y eficiencia. Además, diferentes equipos pueden trabajar en paralelo sin interferir entre sí, lo que agiliza el desarrollo y promueve la innovación.

Técnicas de integración de microservicios

Las Interfaces de Programación de Aplicaciones (APIs) son reglas que permiten una comunicación organizada entre microservicios. Cada microservicio tiene su propia API que otros utilizan para intercambiar información, posibilitando la integración sin conocer detalles internos. APIs bien definidas garantizan una comunicación coherente, en el enfoque de Eventos y Mensajería, los microservicios intercambian eventos o mensajes a través de sistemas como Kafka o RabbitMQ. Un microservicio envía eventos cuando algo relevante ocurre, y otros responden según su interés. Esto brinda comunicación asincrónica, adecuada para situaciones menos urgentes, en términos de Orquestación y Coreografía, la orquestación implica un control central para dirigir el flujo entre microservicios, mientras que la coreografía sugiere colaboración sin control central. Ambos enfoques manejan flujos complejos.

El Gateway de API actúa como intermediario para las solicitudes de clientes, centralizando el enrutamiento y ofreciendo autenticación y autorización. Sirve como punto de entrada único, una Malla de Servicio es una capa entre microservicios que brinda seguridad, monitorización, equilibrio de carga y detección de fallos en la comunicación. Mejora el control y la visibilidad de las interacciones.

Balanceo de carga

la Arquitectura Orientada a Servicios de Balanceo de Carga es una estrategia inteligente para asegurar el funcionamiento confiable y suave de servicios en un entorno distribuido. Consiste en distribuir de manera eficiente las tareas entre diferentes servicios para mejorar el rendimiento y la disponibilidad del sistema.

En esta arquitectura, los servicios se organizan en una red interconectada, donde un componente central, conocido como "balanceador de carga", juega un papel clave. El balanceador de carga se encuentra entre los clientes (que pueden ser usuarios o aplicaciones) y los servicios que proporcionan la funcionalidad. Su función principal es recibir las solicitudes entrantes y decidir a qué instancia del servicio dirigir cada solicitud, con el propósito de evitar la sobrecarga de un servicio específico y garantizar una distribución uniforme de la carga.

La escalabilidad es un aspecto fundamental para adaptarse a las demandas cambiantes. La arquitectura permite agregar o reducir instancias de servicios según sea necesario, lo que optimiza el uso de los recursos disponibles y garantiza un rendimiento óptimo en momentos de alta demanda.

La gestión de tráfico es otra característica esencial de esta arquitectura. Se pueden aplicar estrategias como el enrutamiento basado en el peso de las instancias o la evaluación de la salud de los servicios. Estas estrategias permiten optimizar cómo se manejan las solicitudes, asegurando una distribución equitativa y eficiente de la carga de trabajo entre los diferentes servicios.

La disponibilidad también está asegurada a través del balanceador de carga. Si un servicio experimenta una falla, el balanceador redirige automáticamente las solicitudes a instancias saludables, minimizando cualquier impacto negativo en los usuarios y asegurando una experiencia continua.

Finalmente, la distribución de la carga tiene un impacto directo en el rendimiento general del sistema. Al evitar que un servicio se convierta en un cuello de botella debido a una carga desproporcionada, la arquitectura garantiza una mejor velocidad y tiempos de respuesta para los usuarios, lo que resulta en una experiencia más fluida y satisfactoria.

Despliegue

La Arquitectura Orientada a Servicios de Despliegue (AOSD) es una manera inteligente de construir y organizar aplicaciones de software. Imagina que estás creando un equipo de superhéroes para salvar el día.

En la AOSD, en lugar de hacer un solo programa grande que hace muchas cosas diferentes, divides el programa en partes más pequeñas llamadas "servicios". Cada servicio es como un superhéroe que realiza una tarea específica, como volar o disparar rayos láser.

La ventaja principal de esta arquitectura es que, si necesitas hacer cambios en un servicio, no tienes que rehacer todo el programa.

Un ejemplo de esto sería imaginar que estás construyendo un edificio de apartamentos. En lugar de hacer un solo edificio enorme con todas las funciones en un solo lugar, decides construir pequeños apartamentos separados. Cada apartamento hace una cosa específica, como cocinar o dormir.

Cada apartamento es como un "servicio". Cada uno hace su trabajo y puede hablar con los otros apartamentos cuando sea necesario. Por ejemplo, si necesitas algo de comida para tu fiesta en tu apartamento, puedes pedir a tu vecino de al lado que te preste algunos ingredientes.

La idea inteligente aquí es que, si necesitas hacer cambios en tu apartamento, no afectará a los otros. Puedes redecorar tu espacio sin tener que preocuparte por cómo se ve o funciona el apartamento de tu vecino.

En definitiva, el Despliegue es como construir un complejo de apartamentos donde cada uno tiene su propósito y puede trabajar juntos de manera eficiente. Esto hace que las aplicaciones sean más fáciles de mejorar y mantener a lo largo del tiempo.

Arquitectura monolítica vs arquitectura de microservicios

La arquitectura monolítica es como una casa grande con todas las habitaciones juntas, mientras que la arquitectura de microservicios es como un vecindario de casitas separadas.

Cada enfoque tiene sus ventajas y desafíos, y la elección depende de la complejidad y necesidades de la aplicación que estés construyendo.

Arquitectura de Microservicios:

Imagina un vecindario lleno de casitas independientes. En una arquitectura de microservicios, la aplicación se divide en "casas" separadas llamadas microservicios. Cada microservicio es como una casita que tiene una función específica. Puede haber una casita para manejar pagos, otra para gestionar usuarios y otra para mostrar contenido, por ejemplo.

Ventajas:

Cambiar o mejorar partes individuales de la aplicación es más sencillo, ya que cada microservicio es independiente.

La escalabilidad y flexibilidad son mejoradas, porque puedes mejorar o añadir nuevos microservicios según lo necesites.

Escalabilidad Precisa: Puedes escalar solo las partes de la aplicación que necesitas, lo que mejora la eficiencia.

Flexibilidad en las Tecnologías: Cada microservicio puede usar diferentes tecnologías según sus necesidades, lo que puede ser beneficioso.

Desarrollo Independiente: Diferentes equipos pueden trabajar en diferentes microservicios sin interrumpir a los demás, permitiendo un desarrollo más rápido.

Desventajas:

Gestionar muchos microservicios puede volverse complicado y requerir una buena planificación.

Los microservicios necesitan comunicarse entre sí para trabajar juntos, por lo que una buena coordinación es esencial.

Mayor Complejidad Inicial: Coordinar y gestionar múltiples microservicios puede ser más complicado al principio.

Comunicación Requerida: Los microservicios necesitan comunicarse entre sí, lo que puede aumentar la complejidad de la arquitectura.

Posibles Dificultades de Coherencia: Mantener la coherencia entre los datos almacenados en diferentes microservicios puede ser un desafío.

Arquitectura Monolítica:

Puedes imaginar una arquitectura monolítica como una gran casa de varios pisos. En esta casa, todas las habitaciones están en el mismo edificio. Esto significa que todas las partes de una aplicación, como la cocina, el dormitorio y la sala de estar, están ubicadas dentro de ese edificio gigante. Es como si todas las funciones de una aplicación vivieran bajo el mismo techo.

Ventajas:

Al principio, es más fácil de construir y desarrollar, porque todo está en un solo lugar.

Para aplicaciones pequeñas o simples, la complejidad puede ser manejable.

Sencillez Inicial: La construcción y el desarrollo inicial son más simples, ya que todo está en un solo lugar.

Menos Comunicación Compleja: No necesitas establecer una comunicación compleja entre diferentes partes, ya que todo está en el mismo código.

Menor Sobrecarga de Comunicación: No hay necesidad de preocuparse por la comunicación entre diferentes componentes, lo que puede ser más eficiente en aplicaciones pequeñas.

Desventajas:

Cambiar o mejorar partes específicas de la aplicación puede ser complicado, ya que todo está entrelazado.

Si algo sale mal en una parte, podría afectar toda la aplicación, como si un problema en una habitación afectara toda la casa.

Dificultad en la Escalabilidad: A medida que la aplicación crece, puede volverse más difícil de escalar horizontalmente.

Actualizaciones Complejas: Cambiar una parte de la aplicación podría afectar otras áreas, lo que hace que las actualizaciones sean más riesgosas y complicadas.

Acoplamiento Fuerte: Las partes de la aplicación pueden estar muy interconectadas, lo que hace que sea difícil cambiar o reemplazar componentes individuales.

Buenas prácticas para diseñar arquitecturas de microservicio

El diseño de arquitecturas de microservicios se centra en el modularidad, la independencia y la eficiencia. Cada punto que mencionaste contribuye a un sistema que es más fácil de construir, mantener y escalar, al tiempo que permite una comunicación efectiva entre los microservicios y el cliente.

División de Responsabilidades**,** divide la funcionalidad en microservicios basados en dominios o tareas específicas, cada microservicio debe tener una responsabilidad única y estar centrado en una tarea específica.

Comunicación entre Microservicios, Utiliza APIs claras y documentadas para que los microservicios se comuniquen entre sí, Prefiere la comunicación a través de protocolos ligeros como HTTP/REST o protocolos de mensajería como AMQP.

Autonomía y Escalabilidad, Cada microservicio debe ser independiente y autónomo, lo que permite escalabilidad individual, la escala los microservicios según la demanda de cada uno, en lugar de toda la aplicación.

Persistencia Independiente, Cada microservicio puede elegir la tecnología de persistencia más adecuada para su tarea, evita compartir bases de datos entre microservicios para evitar acoplamiento.

Gestión de Errores y Resiliencia**,** diseña microservicios para manejar fallas y errores de manera individual, utiliza técnicas como Circuit Breaker y fallbacks para garantizar la resiliencia del sistema.

Registro y Descubrimiento**,** Implementa un registro y descubrimiento de servicios para que los microservicios puedan encontrar y comunicarse entre sí

Monitorización y Telemetría, Implementa sistemas de monitorización y telemetría para rastrear el rendimiento y la salud de cada microservicio.

Despliegue y Automatización**,** utiliza herramientas de automatización para el despliegue y gestión de microservicios, emplea prácticas como CI/CD para facilitar actualizaciones y despliegues frecuentes.

Versiona miento de API, siempre que realices cambios en las APIs de los microservicios, mantén una forma de versionamiento para evitar problemas de compatibilidad.

Seguridad y Autenticación**,** implementa medidas de seguridad, autenticación y autorización en cada microservicio, Considera el uso de tokens y protocolos de autenticación estándar.

Diseño y Patrones de Mensajería, Utiliza patrones de mensajería como el patrón "cola" o "publicar/suscribir" para comunicación asíncrona y desacoplada.

Pruebas Unitarias y de Integración**,** realiza pruebas unitarias y pruebas de integración para cada microservicio para garantizar su funcionamiento correcto.

Aplicación con el cliente:

Mantén la interfaz de usuario (cliente) lo más ligera posible: Esto significa que la mayoría de la lógica y el procesamiento de datos deben realizarse en los microservicios en lugar de en el cliente. El cliente debe centrarse en la presentación y la interacción con el usuario.

Usa APIs claras y bien documentadas: Las interfaces de programación de aplicaciones (API) son como puertas de entrada a tus microservicios. Asegúrate de que estén documentadas de manera clara y detallada para que los desarrolladores del cliente puedan entender cómo interactuar con los servicios.

Utiliza patrones de diseño en el cliente, como el patrón "renderless": El patrón "renderless" implica separar la lógica de presentación de la lógica de negocio en el cliente. Esto significa que tu interfaz de usuario se encarga de mostrar información y capturar interacciones, mientras que la lógica compleja se maneja en otros lugares (como los microservicios) y se comunica con el cliente a través de APIs.

Servicios:

Divide los servicios basándote en la funcionalidad específica: Cada microservicio debe tener una tarea clara y específica, como autenticación, gestión de usuarios, procesamiento de pedidos, etc.

Esto permite que los equipos se enfoquen en áreas específicas y mantengan los servicios pequeños y manejables.

Mantén cada servicio autónomo y enfocado: Cada servicio debe poder funcionar por sí mismo sin depender en gran medida de otros servicios. Esto hace que sea más fácil de desarrollar, probar y mantener, ya que cada equipo puede trabajar de manera independiente en su propio servicio.

Utiliza la tecnología más adecuada: Puedes elegir diferentes tecnologías para cada microservicio según sus necesidades. Por ejemplo, un servicio puede usar un lenguaje de programación específico o una base de datos que se adapte mejor a su funcionalidad.

Directorio de Servicio:

Implementa un registro centralizado: Esto permite que los microservicios se registren y compartan información sobre su ubicación y capacidades. Esto facilita la comunicación y el descubrimiento entre los servicios.

Utiliza tecnologías como Consul o Eureka: Estas herramientas son ejemplos de soluciones de registro y descubrimiento de servicios. Ayudan a mantener un registro actualizado de los servicios disponibles y permiten que los microservicios encuentren y se comuniquen entre sí.

Bus de Servicio:

Emplea un bus de servicios: Un bus de servicios como RabbitMQ o Kafka permite la comunicación asíncrona entre los microservicios. Esto es especialmente útil cuando los servicios necesitan comunicarse sin bloquear el flujo de trabajo, utiliza patrones de mensajería comunicación de manera confiable, escalable y flexible. Garantizan que los mensajes se entreguen y procesen correctamente incluso en situaciones de alta demanda.

CONCLUSION

Como conclusión llegamos que, al tener un dinámico universo del desarrollo de aplicaciones, la arquitectura de microservicios aparece como un paradigma transformador que remodela la manera en que concebimos y construimos sistemas de software, como la creación de algo tan complejo mediante la unión de piezas individuales, esta metodología puede fragmentar una aplicación en unidades autónomas y diminutas conocidas como microservicios.

Además, pudimos aprender sobre que son una esencia de mini programas, cada uno dedicado a ejecutar una tarea específica, su gran éxito y triunfo reside en su habilidad para comunicarse con otros microservicios a través de interfaces, bien definidas, típicamente a través de APIs.

Se podría decir que es una de las mejores cosas que podremos encontrar en la programación arquitectónica es su aptitud para catalizar la colaboración y acelerar el desarrollo,

Para finalizar se puede saber que de manera análoga, en el dominio de la programación, los equipos de desarrolladores pueden trabajar en sus propios microservicios sin perturbar, siendo muy útil, veloz y sobre todo tener una mejor manera de poder trabajar con servicios demasiado amplios

BIBLIOGRAFIA

Salinas, E., Cerpa, N., & Rojas, P. (2011). Arquitectura orientada a servicios para software de apoyo para el proceso personal de software. *Ingeniare. Revista chilena de ingeniería*, *19*(1), 40-52.

Contreras, D. A. B. (2018). Arquitectura de microservicios. *Tecnología Investigación y Academia*, *6*(1), 36-46.

López, D., & Maya, E. (2017). Arquitectura de software basada en microservicios para desarrollo de aplicaciones web.

Vera-Rivera, F. H., Astudillo, H., & Gaona, C. (2019). Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación. *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, (E23 (2019)), 107-120.

Velepucha, V., Flores, P., & Torres, J. (2019). MOMMIV: Modelo para descomposición de una arquitectura monolítica hacia una arquitectura de microservicios bajo el Principio de Ocultación de Información. *Revista Ibérica de Sistemas e Tecnologias de Información*, (E17), 1000-1009.

ANEXOS