PROTOTIPO DE LIBRERÍA PARA LA IMPLEMENTACIÓN DE INTEGRACIONES USANDO ARQUITECTURAS ORIENTADAS A MICROSERVICIOS

AUTORES

Juan Carlos Patiño Hernández

Alexander Tamayo Pino

Universidad Tecnológica de Pereira

Facultad de Ingenierías Física, Eléctrica, Electrónica, Sistemas y Computación

Ingeniería de Sistemas y Computación

Pereira, Risaralda

2019

PROTOTIPO DE LIBRERÍA PARA LA IMPLEMENTACIÓN DE INTEGRACIONES USANDO ARQUITECTURAS ORIENTADAS A MICROSERVICIOS

AUTORES

Juan Carlos Patiño Hernández

Alexander Tamayo Pino

DIRECTOR

Ricardo Antonio Bermúdez

Universidad Tecnológica de Pereira

Facultad de Ingenierías Física, Eléctrica, Electrónica, Sistemas y Computación

Ingeniería de Sistemas y Computación

Pereira, Risaralda

2019

CONTENIDO

1. INTRODUCCIÓN

En los últimos años el uso de arquitecturas orientadas a micro servicios ha tomado gran importancia en el desarrollo de software, esto debido a la forma en la que se concibe, permiten que una aplicación de gran tamaño se pueda distribuir en pequeños módulos, cada uno especializado en tareas específicas y puedan ser compartida entre partes interesadas con previo acuerdo de uso de información.

La implementación de la arquitectura basada en microservicios ha generado nuevos retos en el desarrollo de aplicaciones que integran diferentes fuentes de información, estas a su vez deben estar en la capacidad de poder comunicarse con otras aplicaciones o servicios, por lo general estos están construidos en diferentes lenguajes de programación y por consiguiente, lo más probable es que los protocolos de intercambio de datos que usan puedan variar según fueron implementados (SOAP, API REST).

Este proyecto aborda una investigación sobre cómo a través del tiempo, los enfoques en implementación de arquitecturas han evolucionado hasta llegar a las actuales tendencias, y como aporte, se implementa un prototipo de librerías de integración bajo la arquitectura orientada a microservicios que permita integrar dos (2) fuentes de información de dos (2) sistemas diferentes, teniendo como objetivo que puedan implementarse en un proyecto basado en microservicios.

2. RESUMEN

En este proyecto se implementará un prototipo de librería que permita hacer integraciones entre productos de software que se comuniquen a través de servicios web tales como SOAP o API REST para que ellos puedan comunicarse de una manera ágil y rápida, es el objetivo de esta investigación utilizando las tecnologías actualmente disponibles que permitan la integración entre servicios web de manera automática.

Los sistemas SOAP o API REST pueden comunicarse de manera bidireccional sin importar el tipo de servicio que se esté integrando. Este prototipo estará en la capacidad de:

- Servir como interfaz entre servicios dentro de la misma aplicación o una aplicación externa haciendo que el desarrollo se haga de forma más rápida.
- Convertirse en el primer paso para a futuro lograr que usuarios puedan integrarse con otras plataformas de manera transparente.
- Instar a que los programadores puedan centrarse en el desarrollo específicos de la aplicación web, disminuyendo tiempo al momento de realizar el mapeo entre los dos servicios.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Desarrollar un prototipo de librería para la implementación de integraciones entre dos sistemas de información, que, mediante un desarrollo que use una estrategia de arquitecturas orientadas a microservicios, permita disminuir en un 30% el tiempo dedicado al envío y recepción de información por parte de los mismos.

3.2 OBJETIVOS ESPECIFICOS

- Analizar los requerimientos funcionales y no funcionales para la implementación del prototipo de librería.

- Elaborar un análisis del prototipo de librería para la integración de servicios web.

- Diseñar la arquitectura para el prototipo de librería que integre la comunicación entre servicios web.

- Implementar la arquitectura para una versión preliminar de la librería.

- Elaborar las pruebas o correcciones necesarias, con la finalidad de verificar que el prototipo final cumpla con las expectativas deseadas.

4. DEFINICIÓN DEL PROBLEMA

4.1 Antecedentes

Nos encontramos actualmente en el auge de la era digital, en la cual, datos de cualquier tipo u origen pueden encontrarse alojados en diferentes sistemas de información, dichos sistemas deben estar en la capacidad de comunicarse al mundo exterior mediante un lenguaje común, pero con esto surge un problema : no existe un único lenguaje establecido para comunicación entre sistemas de información, el (los) desarrolladores están en la libertad de escoger un estándar para las comunicaciones, esto conlleva que a menudo encontremos dos sistemas que necesiten comunicarse, pero no hablan el mismo idioma, surge pues la necesidad de crear soluciones que permitan superar esta barrera y hacer de este problema una situación cotidiana.

4.2 Formulación del problema

1. Cuando es realizada una integración para conectar dos sistemas de información, en un porcentaje muy alto de los mismos, se incurre en tiempos adicionales de desarrollo solamente llevando a cabo la puesta a punto entre los protocolos de comunicación que manejan, teniendo que ser llevada a cabo de manera personalizada para cada situación, muchas empresas están perdiendo tiempo y dinero por carecer de una herramienta que sirva como interfaz entre los sistemas y se encargue tanto de recibir como entregar la información de forma transparente.
2. Sin importar cómo estén construidos los sistemas, si la comunicación entre los mismos se encuentra fuertemente acoplada, cualquier cambio que se requiera hacer en el tiempo ya sea en un sistema u otro requerirá realizar cambios en la estructura de ambos, encontramos la necesidad de pasarle esta responsabilidad de adaptación en cambios a la interfaz, la cual deberá estar en la capacidad de evolucionar en la medida que los protocolos de

comunicación deban ser cambiados o actualizados, con el objetivo de optimizar dichos sistemas, concentrándose en una mayor medida en el desarrollo de los mismos.

5. JUSTIFICACION

¿Por qué es importante el problema?

Este problema cobra suma importancia dado que se puede encontrar actualmente, para una gran cantidad de proyectos, la necesidad apremiante de comunicarse entre diferentes actores que realizan una tarea determinada, se sabe que hoy en día, la tendencia marca que la lógica de negocio de las aplicaciones esté lo más distribuida posible, esto ya sea para: aumentar la tolerancia a fallos, facilitar el desarrollo de nuevas funcionalidades , entre otros , todo esto implica que, entre más distribuidos estén los diferentes módulos de un sistema, mayor cantidad de comunicaciones debe realizarse, es allí donde este proyecto actuará, garantizando que, sin importar el lenguaje y/o protocolo elegidos tanto para la lógica como las comunicaciones, sea posible una interconexión ágil, mediante un intermediario que servirá, permitiendo que diferentes protocolos de comunicación puedan ser utilizados, sin tener que preocuparse más adelante por problemas de compatibilidad.

¿Por qué deben invertirse tiempo y recursos en esta investigación?

Porque después de haber invertido tiempo y recursos en este proyecto, el resultado traerá como beneficio que una gran cantidad de organizaciones puedan optimizar la comunicación entre dos o más sistemas de información, agilizando el desarrollo de estos, por consiguiente, disminuyendo el tiempo de los proyectos.

¿Por qué este proyecto debe llevarse a cabo de la forma propuesta?

Porque se ha realizado una etapa de estudio, en la cual se ha investigado, realizado pruebas y se ha valorado diferentes enfoques, se ha decantado por la opción que permitía maximizar la probabilidad de llevar a buen puerto todo el proyecto, de

manera que a futuro pueda convertirse en una solución viable para la mayor cantidad de desarrollos posibles.

6. ESTADO DEL ARTE

6.1 DEFINICION ARQUITECTURA DE SOFTWARE

Es el proceso de definir una solución estructurada que cumple con todos los requisitos técnicos y operativos, al tiempo que optimiza los atributos de calidad comunes, como el rendimiento, la seguridad y la capacidad de administración. Implica una serie de decisiones basadas en una amplia gama de factores, y cada una de estas decisiones puede tener un impacto considerable en la calidad, el rendimiento, la capacidad de mantenimiento y el éxito general de la aplicación.

La arquitectura del software abarca el conjunto de decisiones importantes sobre la organización de un sistema de software, incluida la selección de los elementos estructurales y sus interfaces mediante los cuales se compone el sistema; comportamiento como se especifica en la colaboración entre esos elementos; composición de estos elementos estructurales y de comportamiento en subsistemas más grandes; Y un estilo arquitectónico que guía a esta organización. La arquitectura del software también implica funcionalidad, facilidad de uso, resiliencia, rendimiento, reutilización, comprensibilidad, restricciones económicas y tecnológicas, compromisos y preocupaciones estéticas.

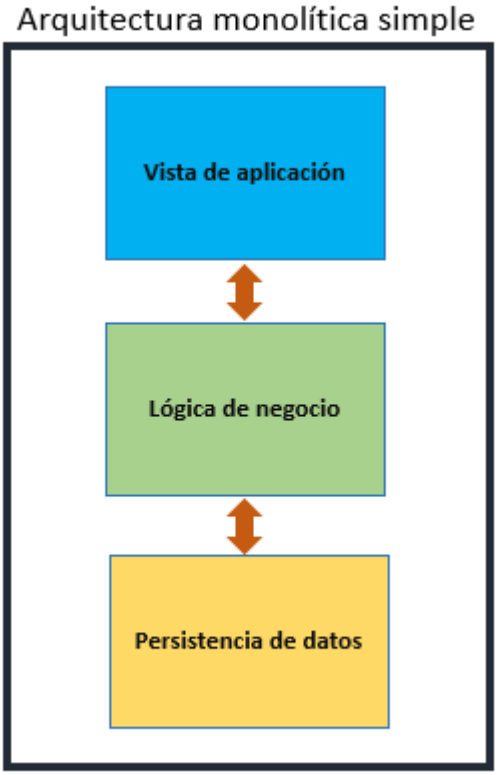
6.2 EVOLUCION ARQUITECTURAS SOFTWARE A TRAVES DEL TIEMPO

El espectro de las arquitecturas de software ha sufrido diversos cambios en las últimas tres décadas, hoy en día se está ad-portas de un cambio disruptivo en este tema, en este punto, se analizará cómo ha sido la evolución de estas y, posiblemente, qué puede esperar el futuro.

6.2.1 Arquitectura monolítica simple

Desde hace décadas e incluso hoy en día, un punto de partida muy común es una distribución de la lógica de negocio, presentación de los datos y persistencia de

estos es denominada arquitectura monolítica o en capas. Esta manera de estructurar los proyectos de software se caracteriza por dividir el desarrollo completo en una serie de capas (comúnmente 3, aunque pueden ser n capas), que son empaquetadas y desplegadas como una simple unidad, suponen un gran punto de partida debido a que los equipos pueden crear rápidamente prototipos de la aplicación, empaquetarlos y correrlos en producción.



(KOSTIC, 2014)

Esta arquitectura es lo bastante explicita como para llegar a ser entendible por muchos desarrolladores, la separación lógica de los componentes es intuitiva y

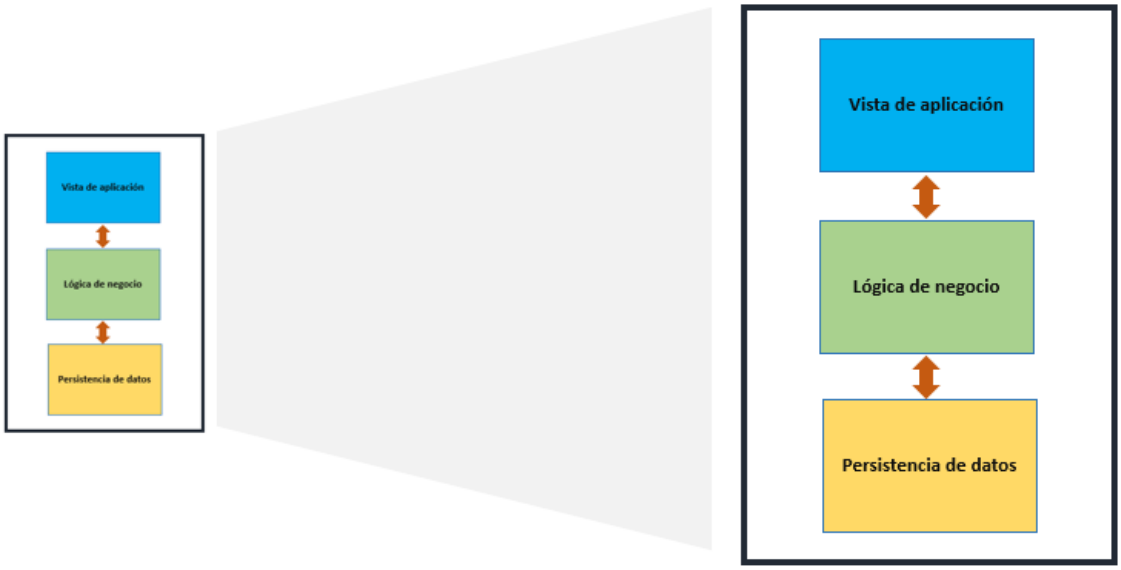
permite cierto aislamiento para que tanto desarrolladores front-end, administradores de bases de datos y desarrolladores back-end trabajen en su ámbito de especialidad.

Tan pronto como la aplicación pasa de ser un prototipo para estar en producción, los problemas empiezan a aparecer; Hay dos problemas en particular, que son constantes y frustrantes:

1- Dependencia : El sistema empieza a moverse al ritmo del miembro del equipo más lento, debido a que los mismos se convierten en dependientes uno del otro, tan pronto como son añadidas nuevas funcionalidades, la interdependencia de las mismas comienza a estar a flote, los cambios realizados en un área afectan otra área, todo el código empieza en una fase de reestructuración que nunca termina y, a medida que más desarrolladores intentan resolver los problemas, el código empieza cada vez a verse más como código espagueti, las pruebas del sistema cada vez implican mayor esfuerzo y el equipo completo comienza a mostrar los primeros signos de atraso.

2- Escalabilidad: Las aplicaciones construidas mediante este enfoque de arquitectura alcanzan su límite de escalabilidad rápidamente. Así la aplicación tenga una excelente estructuración del código, use capas de cacheo de información para incrementar la velocidad, optimice sus operaciones y su estructura en base de datos, eventualmente alcanzará su límite cuando esta aplicación es desplegada como un monolito. Según NEMANJA KOSTIC, quien es un profesional en el campo con una experiencia de más de quince años diseñando e implementando soluciones de software a nivel comercial, los primeros problemas relacionados con escalabilidad comienzan a aparecer cuando se llega a la barrera de los diez mil usuarios concurrentes, muchas aplicaciones web fallan incluso antes de llegar a este punto, pero, desarrollando apropiadamente, la marca puede ser alcanzada, recalcando, que esto depende por supuesto del tipo de aplicación web.

Si la escalabilidad comienza a ser un problema, lo primero que se puede pensar es en poner todo el proyecto en servidores más grandes (escalar hacia arriba). Cajas mucho más grandes con los recursos suficientes para atender la demanda (Ancho de banda, memoria, procesamiento, número de operaciones)



(KOSTIC, 2014)

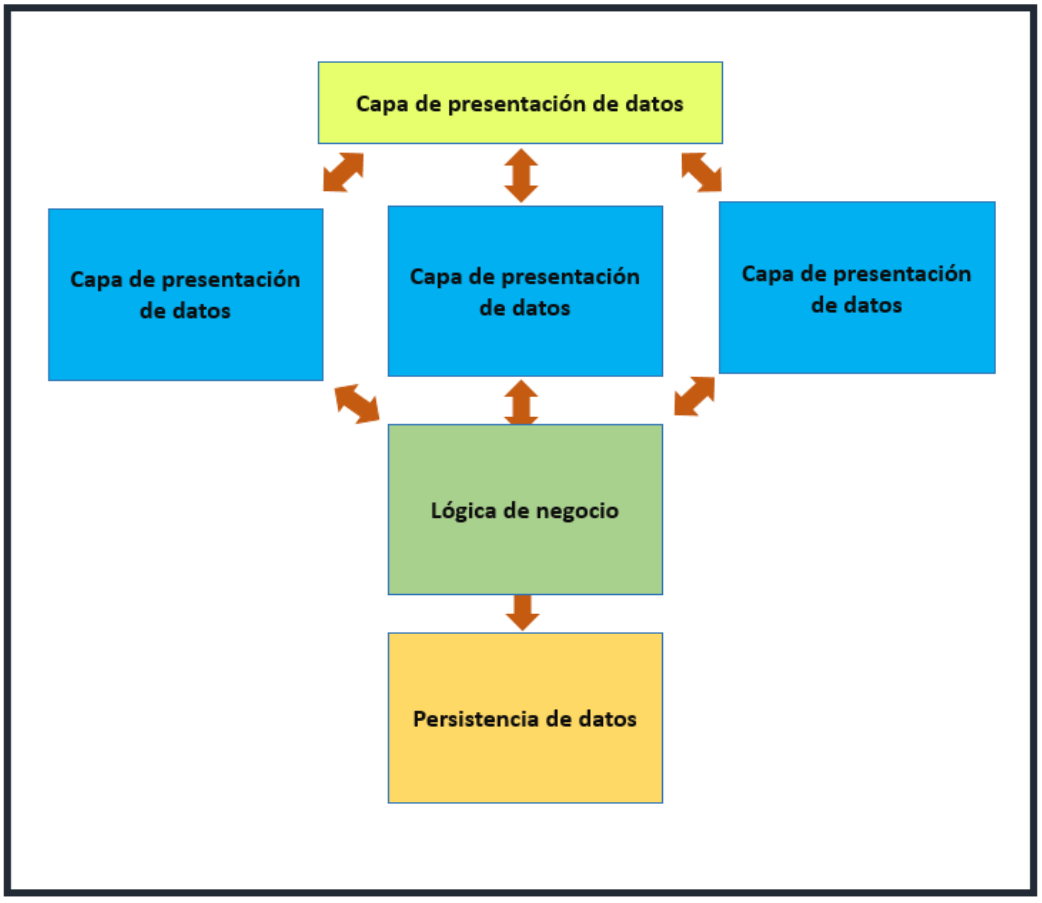
Estos cambios podrían llevar la aplicación a manejar un rango de 50 –100 mil usuarios, pero escalar con este enfoque trae consigo un aumento dramático de costos, de esta forma los servidores de gran tamaño se extienden en sus prestaciones aumentando el costo de adquisición de estos. Para hacerse a una idea de cuánto puede llegar a costar, la instancia más grande que ofrece Amazon llamada x1.32large, se compone de 1952 GB de memoria, 128 CPUs, 4 TB de almacenamiento, con un valor de $13.338 dólares la hora, es en este punto donde

se comprende de que probablemente este enfoque no sea el adecuado, y se deben buscar otras alternativas a este modelo de arquitectura.

6.2.2 Monolito Distribuido

Una vez se ha llegado a un tope con el monolito simple, usualmente los arquitectos de software comienzan a escalar horizontalmente haciendo uso de los tradicionales balanceadores de carga en cada una de las capas de la arquitectura, el primer paso es usar un balanceador en la capa de presentación de datos, logrando distribuir la carga en varios servidores web, pero los servidores de aplicaciones y bases de datos se mantendrían en su forma inicial.

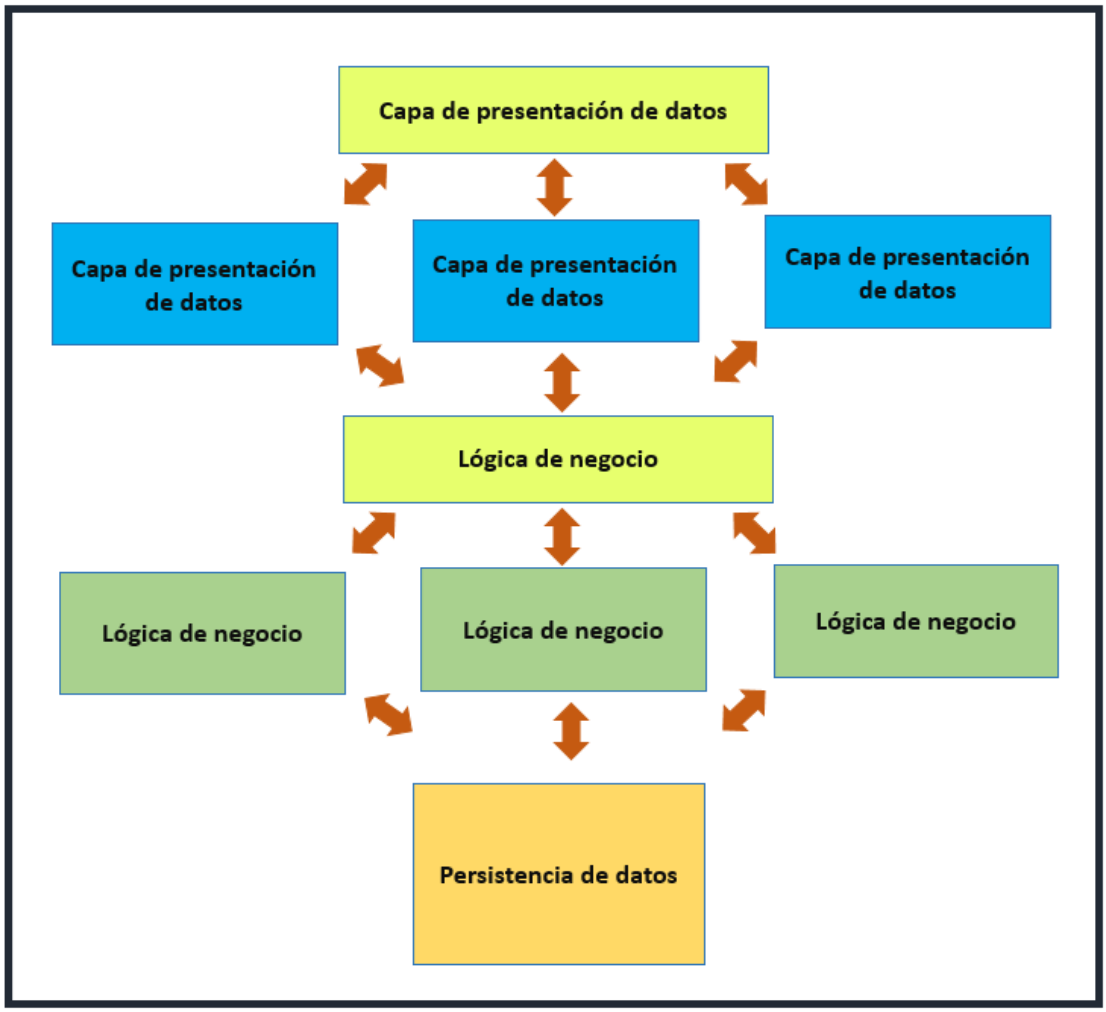
Arquitectura monolítica distribuida



(KOSTIC, 2014)

Una vez que la capa de aplicación empieza a quedarse rezagada, la solución es implementar otro balanceador de carga justo en la capa en la cual se maneja la lógica de negocio. En este punto, probablemente se necesiten ciertos cambios en la lógica de la aplicación. Si las capas de presentación y de negocios no poseen estados, los balanceadores de carga pueden lanzar fácilmente usuarios a cualquiera de los servidores web y de aplicaciones menos ocupados. En caso de que alguna de estas capas tenga estado, se deben usar configuraciones adicionales en los balanceadores, para garantizar la integridad de la sesión.

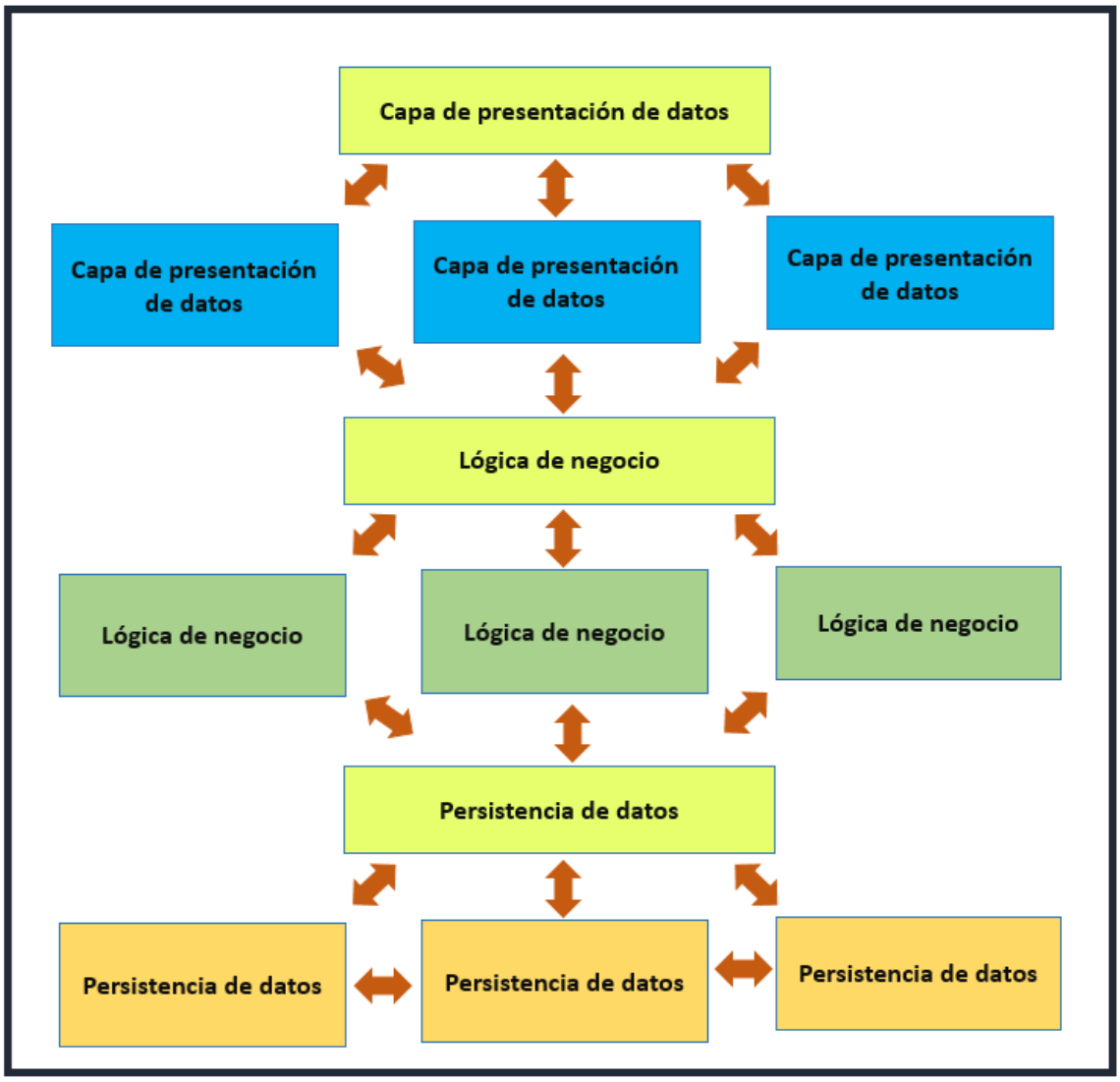
Arquitectura monolítica distribuida



(KOSTIC, 2014)

Para mover aún más la arquitectura hacia un grado de escalabilidad mucho mayor, se puede introducir un tercer balanceador de carga, este es el que distribuye la carga sobre varios servidores de bases de datos. Las bases de datos con balanceadores de carga no se usan a menudo ya que, cuantas más bases de datos se encuentren en el clúster, se requerirá un mayor tiempo para crear replicación entre las bases de datos para preservar la coherencia de estos.

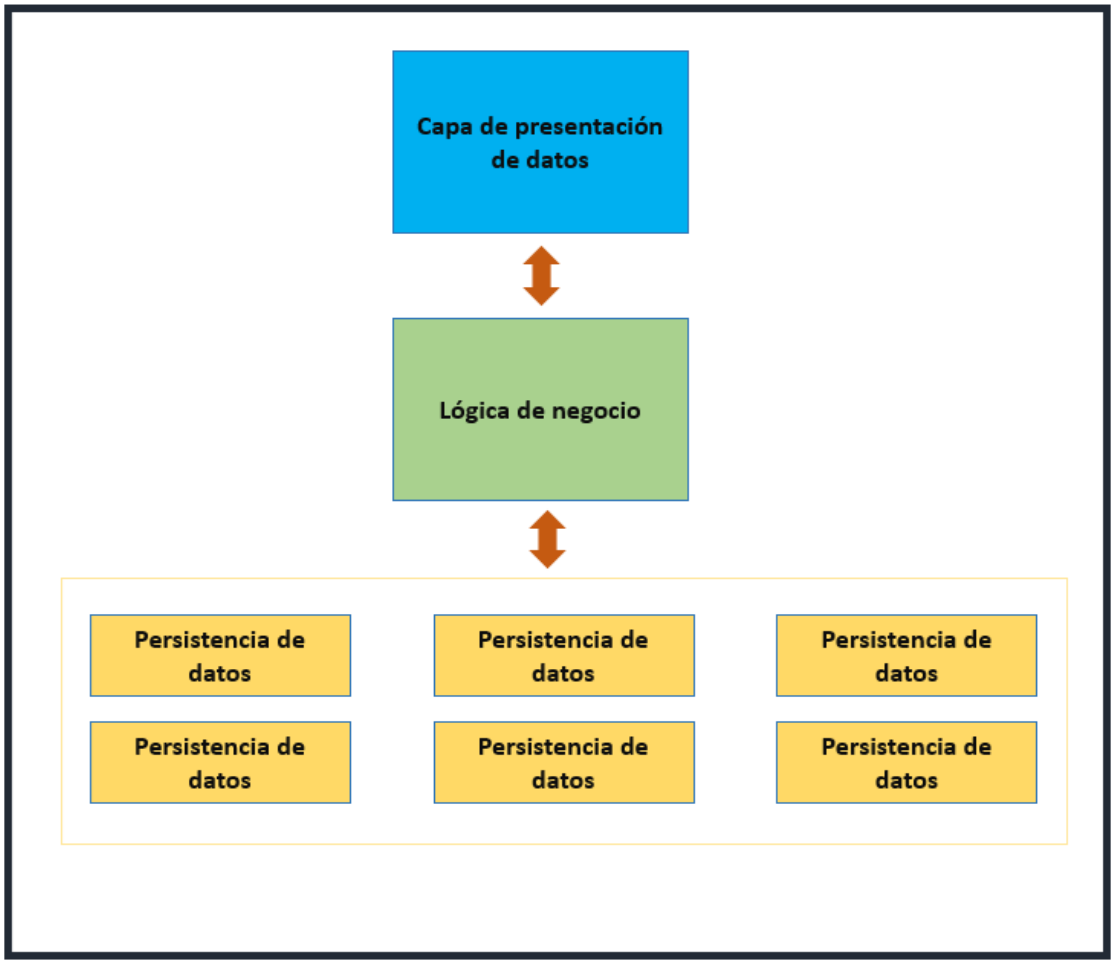
Arquitectura monolítica distribuida



(KOSTIC, 2014)

Ahora, tal vez la mejor manera de aumentar la escalabilidad a nivel de bases de datos es mediante la fragmentación de datos, ello consiste en realizar la partición de la base de datos por alguna separación lógica de un dominio, mediante este enfoque, la carga en las bases de datos se puede distribuir entre muchos servidores y al mismo tiempo mantener la coherencia, así como el modelo transaccional simple.

Arquitectura monolítica distribuida



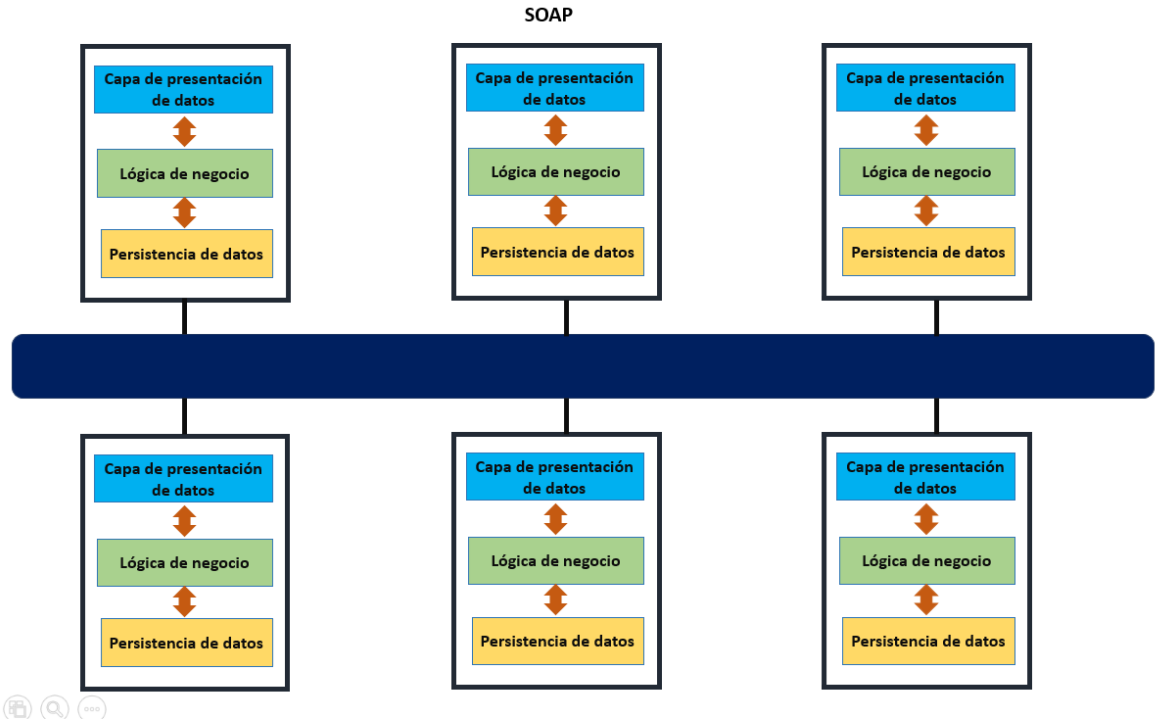
(KOSTIC, 2014)

Todos los escenarios de escalabilidad anteriores son soluciones alternativas para la arquitectura monolítica, el monolito todavía se encuentra presente pero se distribuye total o parcialmente en más servidores, aquí se ve una realidad latente, cuanto más se escala o más se particionan sus datos en fragmentos, más difícil será mantener el sistema completo, es en este punto donde un nuevo enfoque aparece, denominado SOA.

6.2.3 SOA

Como se hacía mención anteriormente, los principales inconvenientes con los monolitos son las restricciones de escalabilidad y dependencia y, aunque la escalabilidad se resolvió hasta cierto punto, la dependencia sigue siendo un gran problema. A medida que las aplicaciones crecen, se ven en la necesidad de intercambiar datos con las aplicaciones circundantes. El panorama general se volvía cada vez más complejo, es en este punto donde una nueva arquitectura surgió como una posible solución a este problema: La arquitectura orientada a servicios

SOA es, a grandes rasgos, una forma de conectar diferentes monolitos de manera consistente.



(KOSTIC, 2014)

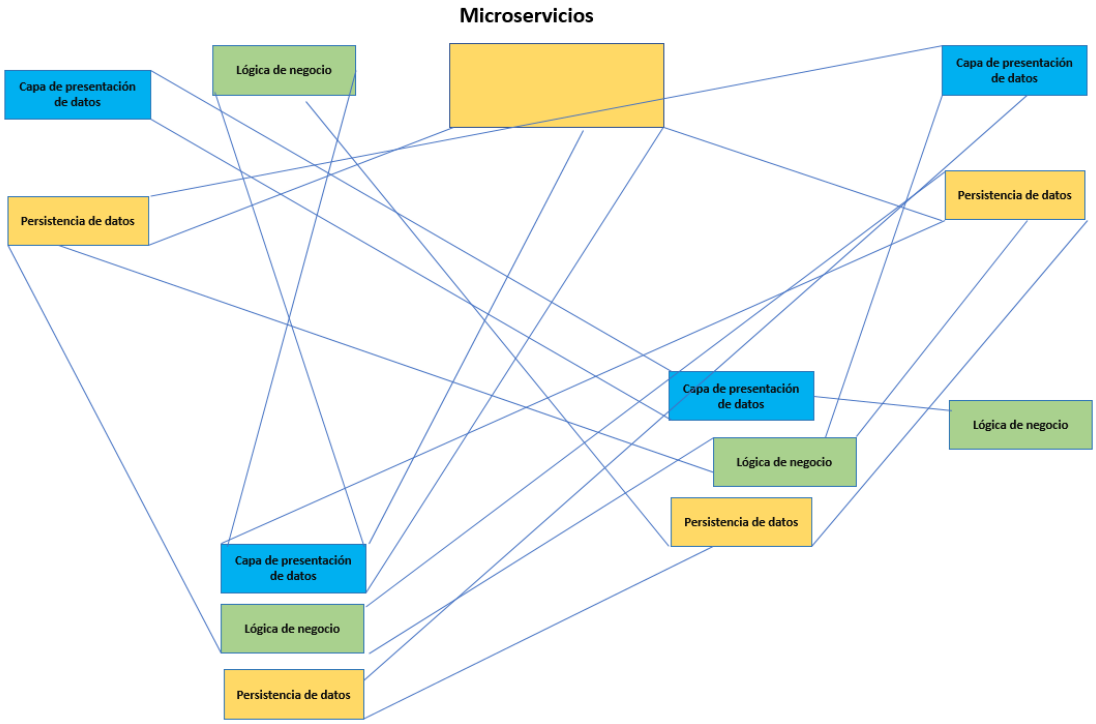
En el ámbito empresarial, SOA tuvo una buena acogida, sin embargo, tuvo sus detractores, quienes exponían que su componente principal, el Enterprise Service Bus (ESB), se convirtió en otro monolito complejo

SOA no representó una gran evolución en términos de escalabilidad en las aplicaciones, pero el concepto introdujo una importante solución de integración que ayudo a las aplicaciones a comunicarse entre sí.

La implementación tradicional de SOA, ofrecida por grandes compañías especializadas, no favoreció el estilo de la arquitectura de integración, pero allanó el camino hacia los servicios más livianos.

6.2.4 Microservicios

En la actualidad, las grandes compañías invierten grandes esfuerzos en la arquitectura de microservicios. Se puede pensar los microservicios como una colección de monolitos pequeños que están diseñados para ser completamente independientes pero juntos forman una sola aplicación. Son independientes, de manera que pueden evolucionar y desplegarse por separado. Los equipos que trabajan con microservicios se comunican entre sí mediante APIs muy bien estructuradas. Cada microservicio tiene su propio ciclo de vida y tecnología asociada, mediante este enfoque se crea independencia y aislamiento entre microservicios, cada microservicio posee su propio contexto y, a menudo, se utiliza Docker como una forma de empaquetar todas las dependencias en una unidad desplegable.



(KOSTIC, 2014)

Las compañías líderes en tecnología como (Amazon, Netflix, Google, Facebook) dependen en gran medida de la arquitectura de microservicios para permitirles impulsar nuevas funciones a producción de manera ágil, incluso en más de una ocasión diaria, sin embargo, es importante no apresurarse a los microservicios solo porque los grandes jugadores los estén usando. Siempre debe realizarse un análisis y preguntarse realmente por qué necesita microservicios, ya que, aunque pueden conducir un proyecto a un nivel de escalabilidad aun mayor, los microservicios vienen con muchas responsabilidades. Es necesario dominar una serie de técnicas para tener éxito con los microservicios:

- Los equipos necesitan usar DevOps al máximo.

- Las herramientas para la orquestación, descubrimiento de servicios y registro deben ser dominadas.

Al emplear una combinación de arquitecturas anteriores, casi que podríamos hablar de escalabilidad infinita, pero ninguna aplicación puede escalar infinitamente a menos que la infraestructura subyacente pueda escalar infinitamente, de esta forma, Amazon desarrolló esta solución hace un par de años siendo pionero que, no solo introdujo servicios gestionados para todas las aplicaciones y componentes de red (bases de datos, DNS, balanceadores de carga, almacenamiento, computación, etc.) sino que fueron más allá e inventaron algo completamente revolucionario : la arquitectura sin servidor

6.2.5 Serverless

Los servicios en nube introdujeron la posibilidad de aprovisionar nuevos servidores en segundos y pagar solo por lo que se utiliza; En un principio Amazon ofreció algo incluso mejor.

Hace apenas cinco años Amazon introdujo Lambdas, el cual es denominado “Función como servicio”, es simplemente una función que se activa cuando ocurre un evento determinado, eventos como recibir una solicitud HTTP, escribir algo en una base de datos, crear archivos, etc.

Los desarrolladores solo deben ocuparse de escribir el cuerpo de esa función de Lambda, dónde se ejecuta esa función y como se escala? Ya no es su problema. Lambdas pueden escalar de uno a mil millones de usuarios y solo se paga por milisegundos de ejecución de lambdas, otros proveedores de nube tienen el mismo concepto, pero con un nombre más descriptivo, en Google y Azure se les denomina Funciones.

Este es realmente un enfoque revolucionario, ya que potencialmente podría poner fin al monolito. Las capas empresariales y de datos ahora son pequeños fragmentos interconectados de funciones y bases de datos administradas, no se sabe dónde y cómo se almacena o ejecuta ni como se escala, simplemente lo hace, automáticamente.

Muchas de las cosas que se solían trabajar antes pueden desaparecer pronto. Todos los servidos cuya configuración se extendía durante horas ya no representan retrasos para los desarrolladores, simplemente se escribe una función y se ejecuta en algún lugar en la nube.

Como se puede observar, se han analizado las posibilidades de cada enfoque para escoger la arquitectura mediante la cual se implementará el desarrollo del proyecto, finalmente se decantado por una alternativa: La arquitectura orientada a microservicios.

6.3 ARQUITECTURA ELEGIDA PARA EL PROTOTIPO

En este punto se expondrá el por qué detrás de esta elección, además de entrar en detalles sobre las características principales que debe poseer un desarrollo orientado mediante este enfoque.

6.3.1 Ventajas arquitectura orientada a microservicios

Se ha elegido esta arquitectura en particular pensando no solo en el presente (el cual es desarrollar el prototipo) sino que también en el futuro, con el ánimo de que luego pueda ser usado de una manera ágil y eficaz, al no encontrarse atado o acoplado al núcleo de la aplicación. Por lo cual se agregan razones puntuales, las cuales se listan a continuación:

- Desacoplamiento: Los servicios dentro de un sistema orientado a microservicios se desacoplan en gran medida. Por lo tanto, la aplicación en su conjunto se puede construir, modificar y escalar fácilmente.
- Componentes: Los microservicios se tratan como componentes independientes que se pueden reemplazar y actualizar fácilmente.
- Capacidades empresariales: Los microservicios son muy simples y se centran en una sola capacidad.
- Autonomía: Los desarrolladores y los equipos pueden trabajar de forma independiente, lo que aumenta la velocidad.
- Entrega continua: Permite lanzamientos frecuentes de software, a través de la automatización sistemática de la creación, prueba y aprobación del software.
- Responsabilidad: Los microservicios no se centran en aplicaciones como proyectos. En cambio, tratan las aplicaciones como productos de los que son responsables
- Gobernanza descentralizada: El enfoque está en usar la herramienta adecuada para el trabajo correcto. Eso significa que no hay un patrón estandarizado o ningún patrón tecnológico. Los desarrolladores tienen la libertad de elegir las mejores herramientas útiles para resolver sus problemas.
- Agilidad: Los microservicios apoyan el desarrollo ágil. Cualquier nueva característica puede ser desarrollada rápidamente y descartada nuevamente.

6.3.2 Ejemplo práctico (caso de estudio)

Como caso de estudio se recurre a la transformación que ha sufrido la arquitectura de Netflix en la última década, pasando de ser monolítica, a una arquitectura orientada a microservicios.

Netflix actualmente posee una infraestructura enorme, han llevado la granularidad de su lógica a tal punto que, hoy en día poseen en producción un número cercano a los mil microservicios, cada uno cumple una tarea específica, lo que les permite tener grandes ventajas:

En primer lugar, la flexibilidad. Esto significa que la arquitectura de microservicios que han construido a través del tiempo es mucho más fácil de mantener y soportar que un monolito. Al constar todo el ecosistema de muchos módulos independientes, los ingenieros son libres de modificar, arreglar o eliminar cada uno de ellos de manera separada. Surge una ventaja adicional: cuando se necesita actualizar un módulo, no existe el miedo generalizado de tener un tiempo de inactividad que afecte a todo el sistema, simplemente se anuncia que una parte del sitio web pasará por mantenimiento programado y su sistema seguirá con el flujo normal.

Segundo y no menos importante, la escalabilidad. Cuando se trata de la actualización de hardware de un sistema con arquitectura de microservicios, su nivel de complejidad baja drásticamente. Están en la capacidad de escalar gradualmente módulos separados, no es necesario actualizar todo el sistema al mismo tiempo.

Ahora bien, a grandes rasgos, la infraestructura que esta compañía ha construido para manejar los diferentes microservicios y garantizar la entrega continua de contenido se divide en:

- Servicios de descubrimiento (Service Discovery): Se encarga específicamente de la detección automática de servicios ofrecidos dentro del sistema; tiene como objetivo reducir los esfuerzos de configuración entre las instancias del sistema
- Balanceadores de carga (Load balancing): Es el encargado de optimizar la distribución de las cargas de trabajo entre los diferentes servicios, tiene como

objetivo optimizar el uso de los recursos, maximizar el rendimiento, minimizar el tiempo de respuesta y evitar la sobrecarga de cualquier recurso único.

- Tolerancia a fallos (Fault tolerance): Se encarga principalmente de garantizar que el ecosistema continúe funcionando correctamente en caso de que se produzcan fallos en uno o más componentes del sistema, es una propiedad vital en esta compañía cuyo producto debe estar disponible el mayor tiempo posible.
- Agregación de datos (Data aggregation): Por ultimo y no menos importante, se encuentra el servicio encargado de procesar los datos en crudo y transformarlos en información, en forma resumida y concisa para su posterior análisis estadístico.

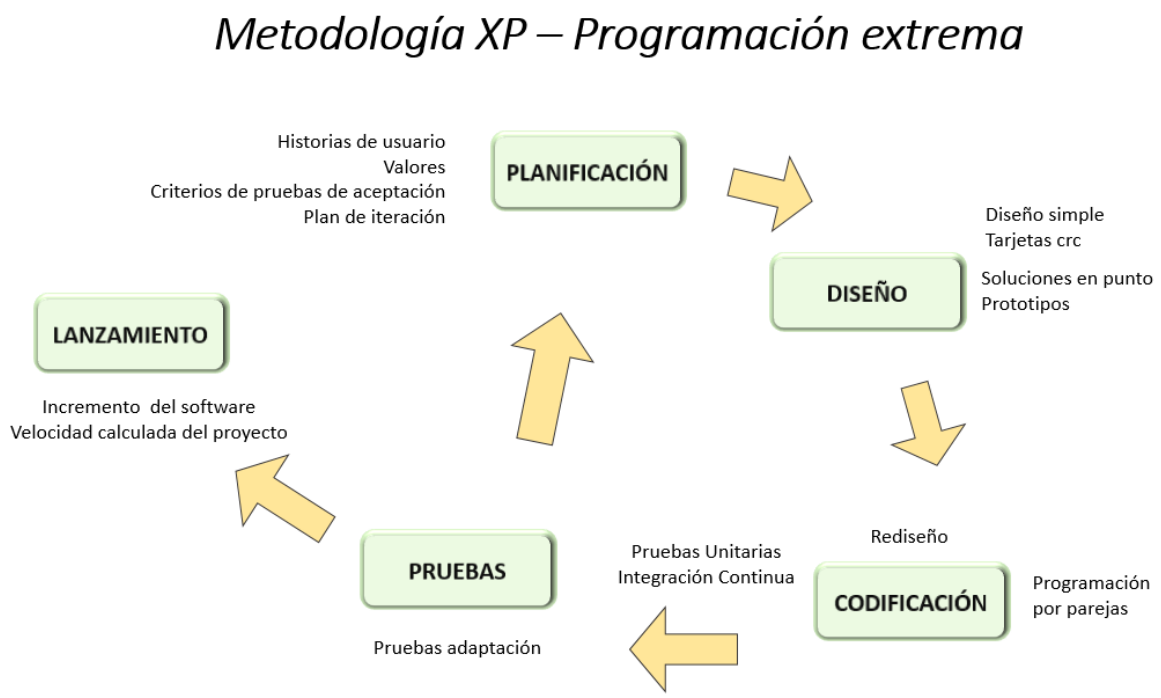
https://www.oreilly.com/ideas/the-evolution-of-software-architecture

7. METODO O ESTRUCTURA DE LA UNIDAD DE ANALISIS

7.1 METODOLOGIA XP PARA DESARROLLO DE LIBRERIA:

La metodología para implementar el prototipo para proyecto de grado será una metodología de desarrollo de software denominada XP (Extreme programming), la cual tiene como objetivo mejorar la calidad del software y la capacidad de respuesta a los cambios en los requisitos del cliente. Se enfoca en los ciclos de desarrollo cortos, que tienen como objetivo mejorar la productividad e introducir puntos de control en los que se pueden adoptar nuevos requisitos de los clientes. (Beck, 1999)

Tiene un esquema que permitirá avanzar de forma rápida:



(Calvo, 2019)

Esta metodología permitirá:

- Programar en pares o realizar ágilmente revisiones del código.
- Evitar la programación de las funciones hasta que sean realmente necesarias.
- Una estructura de administración plana, simple y claridad del código.
- Capacidad de adaptarse a cambios de los requisitos por parte del cliente.

7.2 METODOLOGIA DE INVESTIGACIÓN EXPLORATORIA

Por ser un tema bastante amplio, través de esta investigación el objetivo será ofrecer un primer acercamiento al problema que se debe abordar, por ello se trata de delimitar cumpliendo puntos fundamentales de investigación.

Este método de investigación permitirá “familiarizarnos” y aumentar el conocimiento en áreas específicas de arquitectura de software.

Como resultado de esta investigación se planteará una hipótesis inicial para ser abordada más adelante (Universia, 2017)

8. RECURSOS Y PRESUPUESTO

8.1 RECURSOS FÍSICOS

- Portátiles y computadoras de escritorio.
- Contenedor en la nube.

8.2 Recurso humano e institucional

- 2 desarrolladores

TABLA DE RECURSOS

RECURSOS HUMANOS	**$/HR**	**NUM. HORAS**	**TOTAL**	**FUENTE FINANCIACIÓN**
Alexander Tamayo Pino	10000	8	80000	Propia
Juan Carlos Patiño	10000	8	80000	Propia
Ricardo	50000	1	50000	Universidad Tecnológica
TOTAL, TALENTO HUMANO	70000	17	1190000	
COMPRA O ALQUILER DE MAQUINARIA Y EQUIPOS	**COSTO UNITARIO**	**CANTIDAD**	**TOTAL**	**FUENTE FINANCIACION**
Computador personal	1500000	1	1500000	Propia
Papelería y otros	50000	1	50000	Propia
Software libre	0	0	0	Propia
Servicios públicos	30000	4	120000	Propia
Viajes	8000	16	128000	Propia
Gasto de representación	0	0	0	Propia

Arrendamientos local	0	0	0	Propia
TOTAL, TALENTO HUMANO	4648000	27	5038000	

FUENTE	**COSTO A CARGO**	**%**
Universidad	120000	2.290950745
Propia	5118000	97.70904926
Costo total proyecto	5238000	100

Tabla 1. Tabla de recursos humanos y financieros

CRONOGRAMA AÑO 2018-2019																								
ACTIVIDADES	**Agosto**				**Septiembre**				**Octubre**				**Noviembre**				**Diciembre**				**Enero**			
	Semanas																							
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Elaboración del anteproyecto																								
Entrega anteproyecto																								
Corrección anteproyecto																								
Fundamentación y análisis																								
Implementación del sistema																								
Pruebas funcionales 1																								
Pruebas funcionales 2																								
Compilación de librería y despliegue en contenedor																								
Fase de pruebas																								

CRONOGRAMA AÑO 2019								
ACTIVIDADES	Febrero				Marzo			
	Semanas							
	1	2	3	4	1	2	3	4
Fase de pruebas								
Ajustes y entrega del producto final								
Sustentación								

Tabla 2. Tabla de cronograma

9. DESARROLLO

9.1 ANALISIS DE REQUISITOS

9.1.1Requerimientos funcionales

Los requerimientos funcionales de la librería son los siguientes:

RF-01	**Entrada en XML, genera salida en formato JSON**	
TIPO	Esencial	
ACTORES	Programador	
PROPOSITO		
RESUMEN	La librería debe estar en la capacidad de conectar un servicio que exponga una entidad de datos de tipo XML en la entrada y debe proporcionar una salida en formato JSON	
PRECONDICIÓN	N/A	
SECUENCIA NORMAL	**Acción de los actores**	
	1. Un servicio realiza una petición a la librería exponiendo un XML	**2** la librería recibe el XML, lo procesa de acuerdo con los estándares
	3. La librería entrega una salida en formato JSON.	
	Curso alterno	
	Paso	
	2	En caso de recibir un XML inválido, se emitirá un mensaje de error como respuesta.
POST CONDICIONES	N/A	
COMENTARIOS	N/A	

RF-02	**Entrada en XML, genera salida en formato XML**
TIPO	Esencial

ACTORES	Programador	
PROPOSITO		
RESUMEN	La librería debe estar en la capacidad de conectar un servicio que exponga una entidad de datos de tipo XML en la entrada y debe proporcionar una salida en formato XML	
PRECONDICIÓN	N/A	
SECUENCIA NORMAL	**Acción de los actores**	
	1. Un servicio realiza una petición a la librería exponiendo un XML	**2** la librería recibe el XML, lo procesa de acuerdo con los estándares
	3. La librería entrega una salida en formato XML.	
	Curso alterno	
	Paso	
	2	En caso de recibir un XML inválido, se emitirá un mensaje de error como respuesta.
POST CONDICIONES	N/A	
COMENTARIOS	N/A	

RF-03	**Entrada en JSON, genera salida en formato XML**	
TIPO	Esencial	
ACTORES	Programador	
PROPOSITO		
RESUMEN	La librería debe estar en la capacidad de conectar un servicio que exponga una entidad de datos de tipo JSON en la entrada y debe proporcionar una salida en formato XML	
PRECONDICIÓN	N/A	
SECUENCIA NORMAL	**Acción de los actores**	
	1. Un servicio realiza una petición a la librería exponiendo un JSON	**2** la librería recibe el JSON, lo procesa de acuerdo con los estándares
	3. La librería entrega una salida en formato XML.	
	Curso alterno	
	Paso	

	2	En caso de recibir un JSON inválido, se emitirá un mensaje de error como respuesta.
POST CONDICIONES	N/A	
COMENTARIOS	N/A	

RF-04	**Entrada en JSON, genera salida en formato JSON**	
TIPO	Esencial	
ACTORES	Programador	
PROPOSITO		
RESUMEN	La librería debe estar en la capacidad de conectar un servicio que exponga una entidad de datos de tipo JSON en la entrada y debe proporcionar una salida en formato JSON	
PRECONDICIÓN	N/A	
SECUENCIA NORMAL	**Acción de los actores**	
	1. Un servicio realiza una petición a la librería exponiendo un JSON	**2** la librería recibe el JSON, lo procesa de acuerdo con los estándares
	3. La librería entrega una salida en formato JSON.	
	Curso alterno	
	Paso	
	2	En caso de recibir un JSON inválido, se emitirá un mensaje de error como respuesta.
POST CONDICIONES	N/A	
COMENTARIOS	N/A	

9.1.2 Requerimientos no funcionales

Los requerimientos funcionales de la librería son los siguientes:

Requerimiento no funcional 1

Identificador	Descripción
Código	RNF – 01
Tipo	Eficiencia
Nombre	Eficiente con el consumo de memoria
Resumen	El consumo de memoria al estarse ejecutando la aplicación no debe exceder el 30% de capacidad del contenedor en el cual se esté ejecutando.

Requerimiento no funcional 2

Identificador	Descripción
Código	RNF – 02
Tipo	Usabilidad
Nombre	Implementación y uso de forma transparente
Resumen	La implementación y el uso de la librería debe ser transparente para el desarrollador, sin importar el lenguaje que se encuentre utilizando.

Requerimiento no funcional 3

Identificador	Descripción
Código	RNF – 03
Tipo	Seguridad
Nombre	Evitar que se desarrollen métodos adicionales
Resumen	El sistema solo debe exponer las urls respectivas para procesar las entidades de datos, esto para evitar cambios en el código que comprometa la integridad de la información.

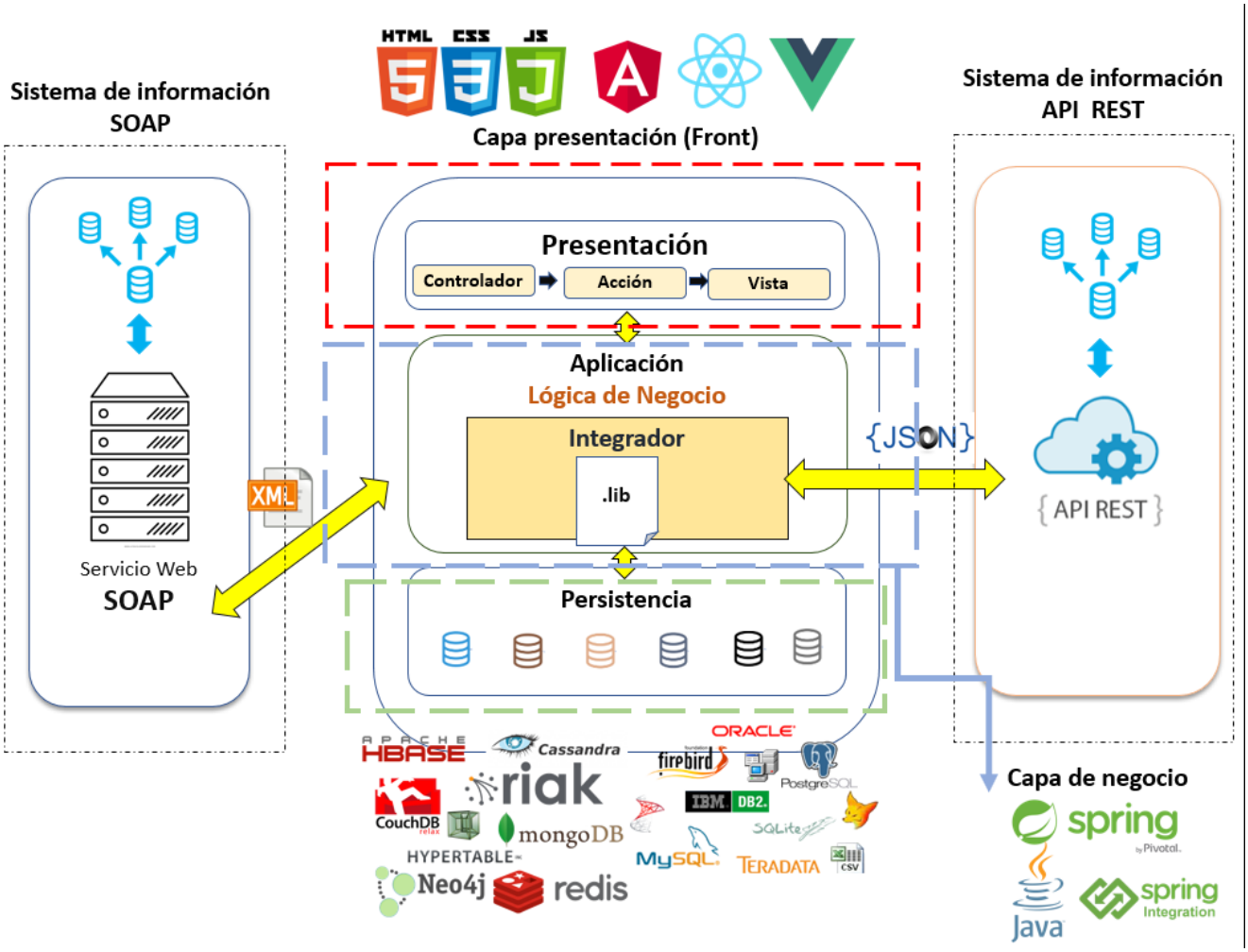
Requerimiento no funcional 4

Identificador	Descripción
Código	RNF – 04
Tipo	Integridad de datos
Nombre	Preservar integridad de datos
Resumen	El sistema realizará validaciones al momento de recibir y entregar los datos con el objetivo de preservar la integridad de estos.

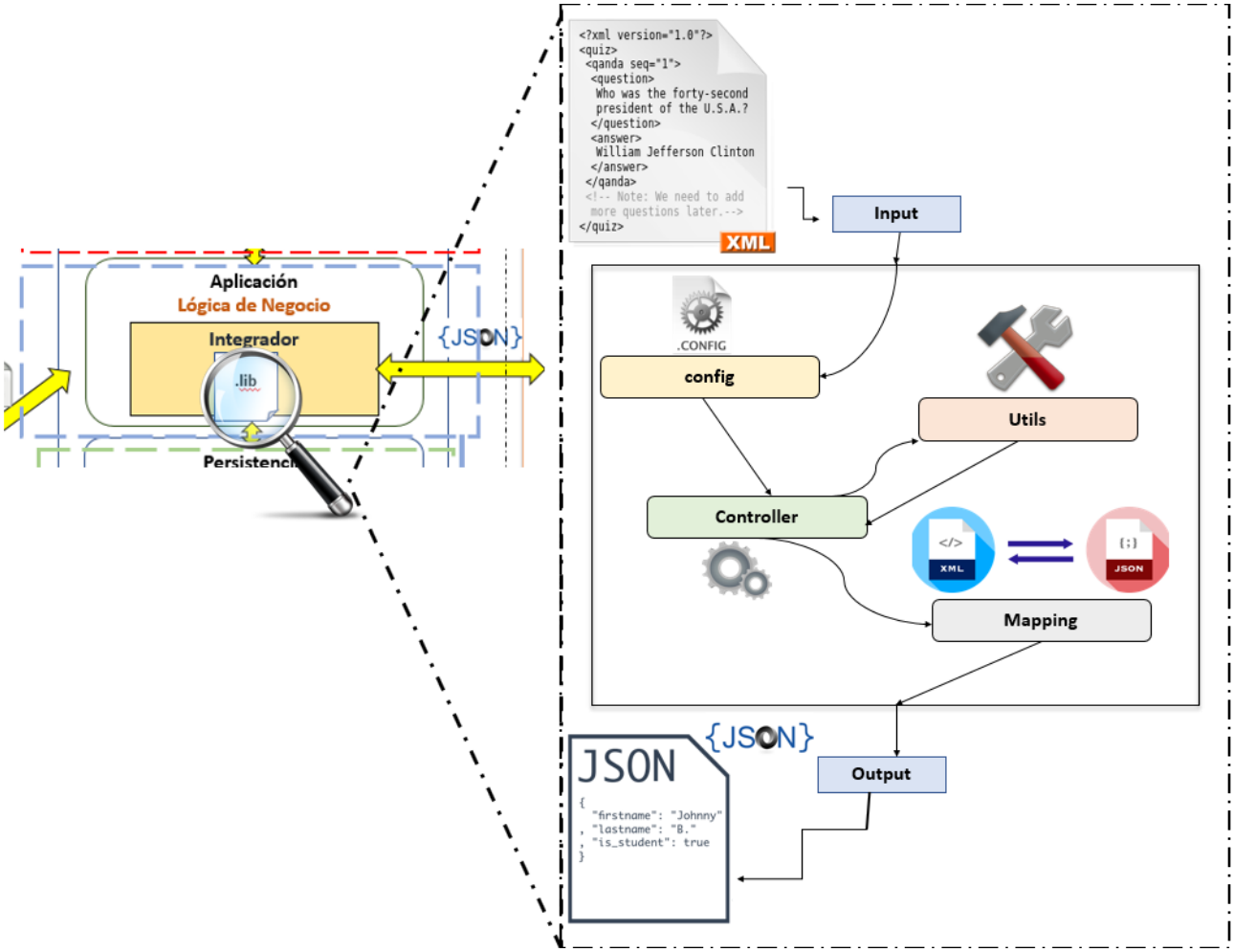
9.2 ARQUITECTURA DEL SISTEMA

9.2.1 Diagrama de la arquitectura

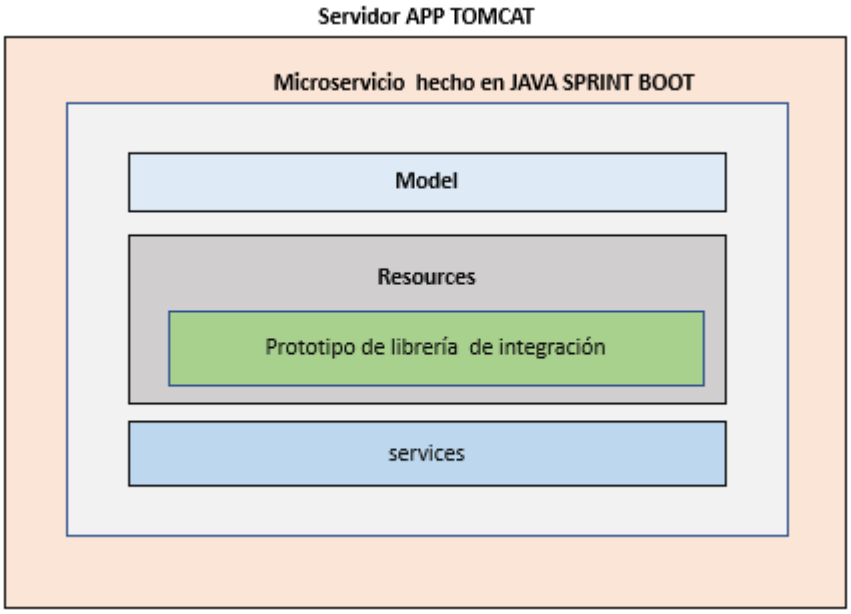
9.2.1.1 Capas de la arquitectura



9.2.1.2 Diagrama del prototipo de librería

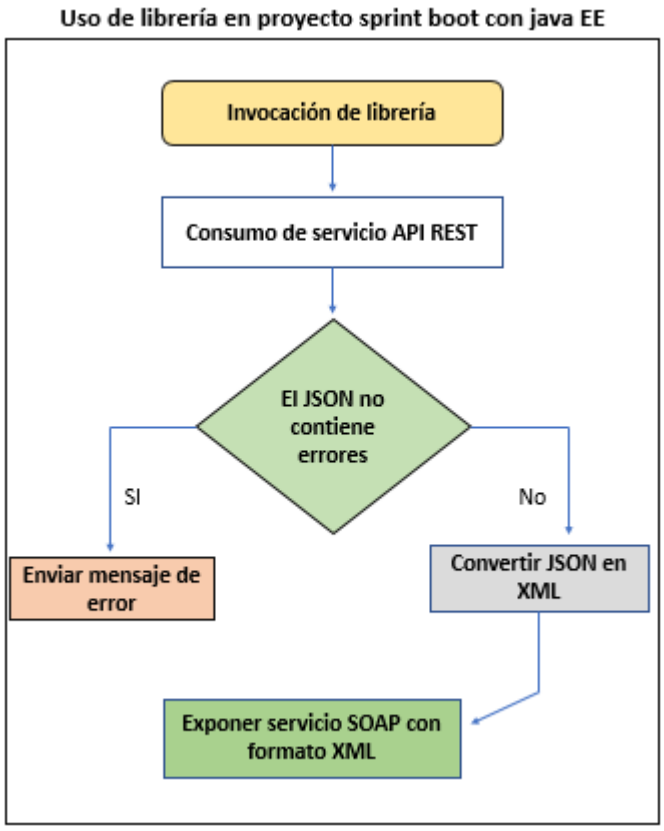


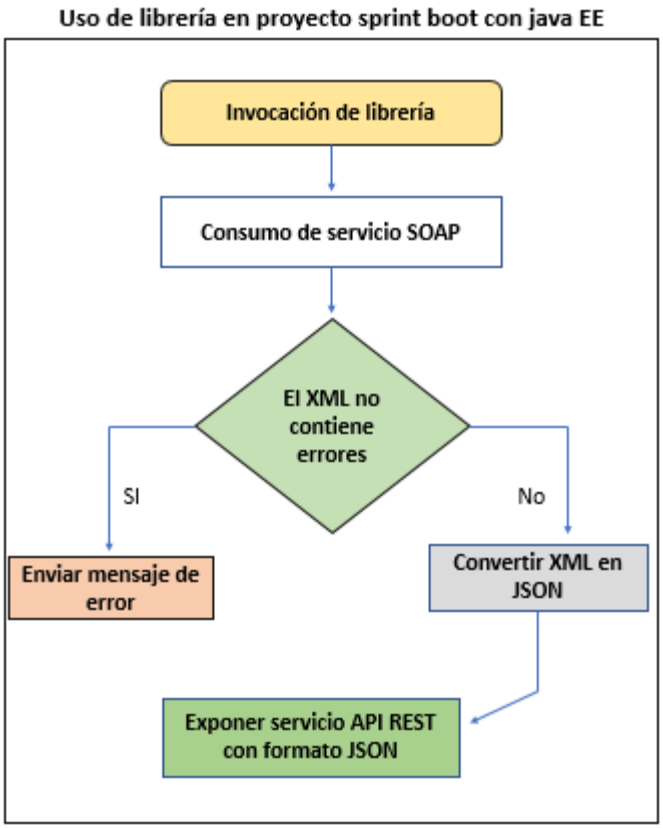
9.2.2.1 Diagrama del despliegue



9.3 DOCUMENTO DE DISEÑO

9.3.1 Diagramas de flujo





9.3 Proceso de desarrollo

Como se hablaba en apartes anteriores del proyecto, se decidió por una metodología de desarrollo ágil llamada XP (extreme programming), la cual tiene como objetivo mejorar la calidad del software y la capacidad de respuesta a los cambios en los requisitos del cliente.

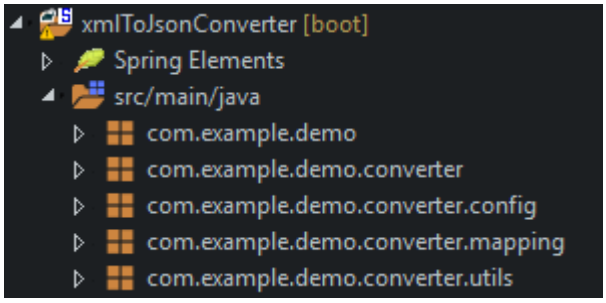
Por otra parte, después de analizar teóricamente las arquitecturas con sus ventajas y desventajas, se ha elegido la arquitectura orientada a microservicios, en este orden de ideas, y después de haberlo analizado cuidadosamente, se elige el lenguaje y las herramientas específicas para realizar el desarrollo, listadas a continuación:

- **JAVA EE**: es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una especificación. Similar a otras especificaciones del Java Community Process (JCP), Java EE es también considerado informalmente como un estándar debido a que los proveedores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes a Java EE; estandarizado por JCP.
- **Spring Boot**: El cual es un framework basado en JAVA EE para el desarrollo de aplicaciones, esta elección se hizo debido a que es una de las primeras plataformas listas para el enfoque de arquitectura orientada a microservicios, y el consenso, hoy en día es la más completa y la mayor difusión.
- **MAVEN:** El cual es una herramienta de software para la gestión y construcción de proyectos en JAVA.
- **Apache Tomcat:** es un contenedor de servlets que se puede usar para compilar y ejecutar aplicaciones web realizadas en Java. Implementa y da soporte tanto a servlets como a páginas JSP (Java Server Pages) o Java Sockets. Además, Tomcat es compatible con otras tecnologías como Java Expression Language y Java WebSocket, del ecosistema Java. Tomcat puede funcionar de manera autónoma como motor de aplicaciones web desarrolladas con Java, aunque habitualmente se usa en combinación con

otros productos como el servidor web Apache, para dar un mayor soporte a tecnologías y aumentar sus características.

```java
package com.example.demo.converter;

import java.io.File;

public class Converter {

    public void xmlFileToJson() {
        String data = "";
        try {
            data = FileUtils.readFileToString(new File("D:\\Work\\laboratory\\java\\testXml.xml"), "UTF-8");
            XmlMapper xmlMapper = new XmlMapper();
            JsonNode jsonNode = xmlMapper.readTree(data.getBytes());
            ObjectMapper objectMapper = new ObjectMapper();
            String value = objectMapper.writeValueAsString(jsonNode);

            System.out.println("*** Converting XML to JSON ***");
            System.out.println(value);

        } catch (JsonParseException e)
        {
            e.printStackTrace();
        } catch (JsonMappingException e)
        {
            e.printStackTrace();
        } catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

Clase que convierte de XML a JSON



Estructura de paquetes del proyecto

Converter.java | XmlToJsonConverterApplication.java

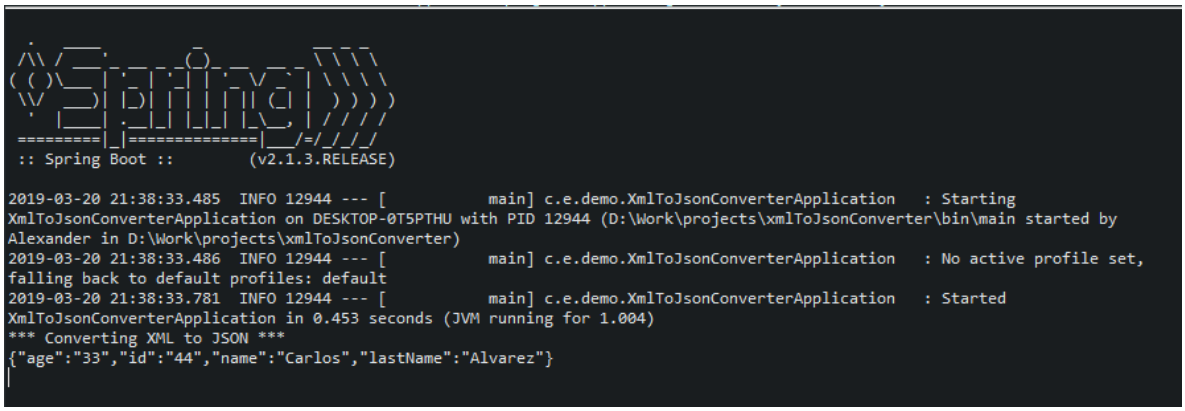
```java
package com.example.demo;

import org.springframework.boot.SpringApplication;


@SpringBootApplication
public class XmlToJsonConverterApplication {

    public static void main(String[] args) {
        SpringApplication.run(XmlToJsonConverterApplication.class, args);
        Converter converter = new Converter();
        converter.xmlFileToJson();
    }
}

```

Punto de entrada del prototipo

testXml.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<student>
    <age>33</age>
    <id>44</id>
    <name>Carlos</name>
    <lastName>Alvarez</lastName>
</student>
```

XML de entrada (prueba)



Salida en formato JSON (prueba)

10. CONCLUSIONES

Una vez finalizado el proyecto, se pueden analizar los siguientes resultados:

- Este prototipo es la piedra angular, el paso inicial para, fases posteriores o en entornos productivos, aumentar la productividad de los equipos de trabajo, permitiendo a los desarrolladores enfocarse en resolver sus problemas de dominio especifico, sin tener que cargar con la responsabilidad de crear interfaces para conectar sus desarrollos.

- Usando las herramientas propuestas, fue posible cumplir a cabalidad con el desarrollo de la aplicación, obteniendo como resultado un prototipo, el cual es funcional en su totalidad.

- A nivel de arquitectura, se cumplió con el objetivo, el prototipo cumple con la estructura necesaria para, en un entorno en el cual necesite escalarse, desplegarse como un microservicio.

- A nivel de desarrollo, el objetivo fue alcanzado, el prototipo fue desarrollado con las herramientas propuestas, en el tiempo estipulado.

- Los requisitos no funcionales se encuentran presentes en el desarrollo, cada uno actúa de forma directa o indirecta en las diferentes fases del ciclo de vida del desarrollo.

BIBLIOGRAFIA

Palmero, M. A. S., Martínez, N. S., & Rojas, O. Y. (2019). Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software Review of conceptual elements at representation of software reference architectures. Revista Cubana de Ciencias Informáticas, 13(1).

http://knowing-microservicios.blogspot.com/2018/01/estado-del-arte.html

https://documentos.redclara.net/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

https://documentos.redclara.net/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

https://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/

https://documentos.redclara.net/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

1.