Universidad Técnica del Norte

Instituto de Postgrado

Maestría en Ingeniería de Software

"ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL"

Trabajo de Investigación previo a la obtención del Grado de

Magister en Ingeniería de Software

Línea de Investigación
Arquitectura de Software

AUTOR: López Hinojosa José Daniel

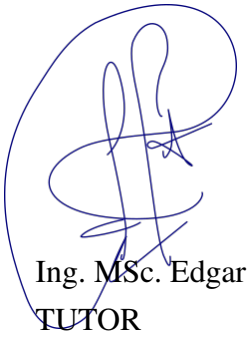
TUTOR: Ing. MSc. Edgar Maya

IBARRA, 2017

APROBACIÓN DEL TUTOR

En mi calidad de tutor del trabajo de grado: **"ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL"**, presentado por el Ing. López Hinojosa José Daniel, para optar por el grado de Magister en Ingeniería de Software, doy fe que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a presentación (pública o privada) y evaluación por parte del Jurado Examinador que se designe.

En la ciudad de Ibarra, 18 de Septiembre de 2017



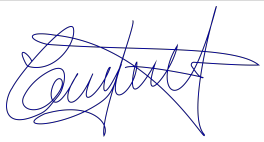
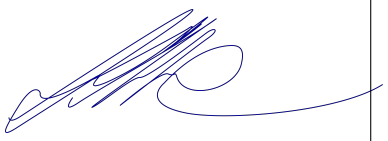
Ing. MSc. Edgar Maya
TUTOR

APROBACIÓN DEL JURADO

ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL

Autor: José Daniel López Hinojosa

Trabajo de grado de Magister, aprobado en la Universidad Técnica del Norte, por los siguientes miembros del tribunal, a los 26 días del mes de octubre de 2017.

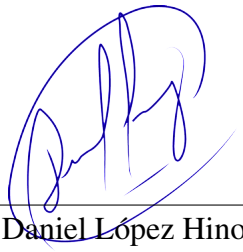
	Apellidos y Nombres	**Firma**
Miembro Tribunal 1	M.Sc. Daisy Imbaquingo	
Miembro Tribunal 2	M.Sc. Diego Trejo	
Miembro Tribunal 3	M.Sc. Mauricio Rea	

DECLARACIÓN DE RESPONSABILIDAD

Ing. José Daniel López Hinojosa

DECLARO QUE:

El proyecto de grado denominado **"ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL**", y bajo juramento que el contenido e información que se encuentra en el presente trabajo de investigación, ha sido desarrollado con base a una investigación exhaustiva y de mi autoría, respetando derechos intelectuales de terceros conforme se menciona en la sección bibliográfica de éste trabajo.



Ing. José Daniel López Hinojosa
CI.: 1002875746

AUTORIZACIÓN DE USO Y PUBLICACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información.

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1002875746		
APELLIDOS Y NOMBRES:	López Hinojosa José Daniel		
DIRECCIÓN:	Juana Atabalipa 15-03 y Hernán González de Saa		
EMAIL:	danielopezh@gmail.com		
TELÉFONO FIJO:	06 2652 192	**TELÉFONO MÓVIL**	099 341 7866

DATOS DE LA OBRA	
TÍTULO:	**ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL**
AUTOR :	José Daniel López Hinojosa
FECHA:	2017-10-26
PROGRAMA:	POSTGRADO
TÍTULO POR EL QUE OPTA:	Magister en Ingeniería de Software
ASESOR/DIRECTOR:	MSc. Edgar Maya

2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

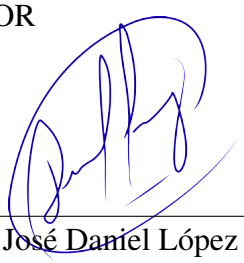
Yo, José Daniel López Hinojosa, con cédula de identidad Nro. 1002875746, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

3. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido y de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 26 días del mes de octubre de 2017

EL AUTOR



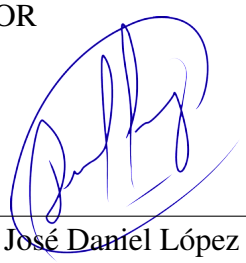
Nombre: José Daniel López Hinojosa
CI.: 1002875746

CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, José Daniel López Hinojosa, con cédula de identidad Nro. 1002875746, manifiesto de forma libre y voluntaria ceder a la Universidad Técnica del Norte los derechos de autor basados según la Ley de Propiedad Intelectual del Ecuador en sus artículos 4, 5 y 6, en calidad de autores del trabajo de grado denominado: **"ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL**", que ha sido desarrollado para optar por el título de Magister en Ingeniería de Software, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En la condición de autores nos reservamos los derechos morales de la obra antes citada. En cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

Ibarra, a los 26 días del mes de octubre de 2017.

EL AUTOR



Nombre: José Daniel López Hinojosa
CI.: 1002875746

DEDICATORIA

A mi esposa Anita y mi hijo Stephano, por todo el amor y compresión brindados día a día que fueron, son y serán siempre mi mayor fuente de inspiración y vida.

A mis padres Luis y Betty, por haberme guiado hacia el camino del conocimiento.

Daniel López Hinojosa

AGRADECIMIENTOS

A la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador, por brindarme su oportuno apoyo.

A los amigos, compañeros y colegas que he tenido la oportunidad de conocer y que con sus criterios aportaron en el desarrollo y culminación del presente trabajo de investigación.

Daniel López Hinojosa

RESUMEN

Actualmente, el proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software tradicional o monolítica que ha sido adoptada del lenguaje de programación utilizado, la plataforma o de la experiencia del personal del área de desarrollo; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas. El objetivo del presente estudio fue identificar las tecnologías, metodología y arquitectura que utiliza la CGTIC para el desarrollo de aplicaciones web y la correspondiente identificación de las tecnologías existentes para el desarrollo e implementación de microservicios, utilizando como base de la investigación un enfoque cualitativo, con un tipo de investigación descriptiva y diseño documental. Se empleó la técnica de grupo focal aplicado a los funcionarios del área de desarrollo de software de la CGTIC, revisión bibliográfica de arquitectura de microservicios y validación de la arquitectura propuesta a través del método de análisis de concesiones mutuas de arquitectura (ATAM). Como resultado de la investigación, el análisis ha permitido identificar el estado del arte respecto a microservicios y su implementación, así como la identificación de los requisitos y necesidades relativos al desarrollo de aplicaciones web y como satisfacerlas mediante el diseño de una arquitectura de software. Proporcionó un marco de trabajo así como la introducción de nuevas prácticas de desarrollo de aplicaciones, bajo un enfoque centrado en servicios con capacidades de resiliencia, escalibilidad, entre otras, que deviene de la implementación de microservicios.

Palabras Clave: Microservicios, arquitectura de software, aplicaciones web.

ABSTRACT

Currently, the software development process carried out by the General Coordination of Information and Communication Technologies (CGTIC) of the National Assembly of Ecuador (ANE) constitutes the use of a traditional or monolithic software architecture that has been adopted from the language of programming used, the platform or the experience of the personnel of the development area; because of the monolithic aspect, this of application types pack all the functionality in a single and large executable unit (a single file or application), which has caused difficulties in aspects such as maintenance, scalability and deliveries. The objective of the present study was to identify the technologies, methodology and architecture used by CGTIC for the web applications development and the corresponding identification of existing technologies to development and implementation of microservices, using a qualitative approach based on a descriptive research type and documentary design. The focal group technique was applied to CGTIC software development staff, bibliographic review of microservices architecture and validation of the proposed architecture through the Architecture Trade-off Analysis Method (ATAM). As a research result, the analysis has allowed the identification of the state of the art regarding microservices and their implementation, as well as the identification of requirements and needs related to the web application development and how to satisfy them by designing a software architecture. It provided a framework as well as the introduction of new application development practices under a service-centric approach with resiliency, scalability, and other capabilities that result from the implementation of micro-services.

Keywords: Microservices, software architecture, web applications.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

PROBLEMA DE INVESTIGACIÓN

1.1. Antecedentes

En las últimas dos décadas, las arquitecturas de software para el desarrollo de aplicaciones web de los sistemas de información han evolucionado desde los sistemas monolíticos primitivos, cliente servidor y su dificultad de administración, así como las arquitecturas orientadas a servicios (SOA), en donde sus componentes se implementan para ser reutilizables con interfaces estándar para su invocación y que si se combinan permiten crear funcionalidades más complejas.

En el Ecuador, a nivel empresarial y tanto en el sector privado como público se realiza desarrollo de software para suplir las necesidades de automatización de procesos internos, este desarrollo ha seguido las tendencias impuestas por la plataforma, lenguaje de programación o por la experiencia del área de desarrollo, lo cual deviene en la implantación de sistemas de construcción tradicional o monolítico.

Actualmente, el proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software monolítica, misma que ha sido adoptada por el uso de un lenguaje de programación específico para construcción de aplicaciones web empresariales; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento,

escalabilidad y entregas. (Carneiro y Schmelmer, 2016).

1.2. Planteamiento del problema

El presente trabajo de investigación, parte de un análisis de la problemática existente dentro de la CGTIC en el desarrollo de aplicaciones web, dicho desarrollo hace uso de una arquitectura monolítica la cual incide en diferentes aspectos tanto tecnológicos y administrativos. Las incidencias más evidentes se pueden observar al aplicar procesos de mantenimiento en sistemas complejos, sea por una petición de cambio, nueva funcionalidad o corrección de un fallo, en los cuales, la resolución de un problema o cambio simple implica el redespliegue de toda la aplicación debido a que se tiene todas las funcionalidades en un único paquete incrementando los riesgos de fallos, lo que a su vez impone resistencia al cambio por el tiempo que toma la implementación de un cambio o nueva funcionalidad, y en consecuencia las actualizaciones son menos frecuentes ya que requieren de un mayor esfuerzo y coordinación de los grupos de desarrollo y la realización de pruebas más extensas.

En aspectos de calidad, surgen complicaciones en la escalabilidad ya que puede requerirse escalar un módulo específico pero, por el aspecto monolítico es necesario escalar la aplicación en su totalidad. De igual manera sucede con la resiliencia, al ocurrir un fallo por caída o sobrecarga en una parte de la aplicación, se pierden todas las funcionalidades; si bien este último aspecto puede ser subsanado mediante replicación o clusters, también incrementa las dificultades de coordinación, configuración y eleva los costos de equipamiento y esfuerzo.

Otra problemática a resaltar, es la incidencia que produce el mantenimiento aplicaciones en el normal desenvolvimiento de las actividades de los usuarios; en este sentido, una de las

áreas más críticas es el Plenario de la ANE, el cual hace uso de software para registrar las votaciones de los legisladores en la creación o reforma de leyes de afectación nacional, por lo que resulta difícil cambiar, agregar o actualizarlo. Adicionalmente, algunos de los sistemas se encuentran impedidos de actualización que es de vital importancia por su nivel de criticidad y afectación, esto se debe una vez más por estar construida bajo el esquema monolítico.

Por los aspectos descritos se ha visto la necesidad de definir una nueva arquitectura de software con un enfoque de vanguardia que facilite el desarrollo de nuevas aplicaciones para las diferentes unidades organizacionales de la ANE. Bajo esta nueva perspectiva, se pretende obtener arquitectura de software flexible que permita el mantenimiento de las aplicaciones evitando la interrupción de actividades del personal que las usa.

1.3. Formulación del problema

Incidencia de la aplicación de una arquitectura de software monolítica para el desarrollo y mantenimiento de aplicaciones web en la continuidad de servicio de la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador .

1.4. Justificación

La presente investigación pretende proponer una nueva arquitectura de software flexible basada en microservicios que permita la construcción estándar de aplicaciones web en la CGTIC de la Asamblea Nacional con énfasis en la reutilización, escalabilidad y mantenibilidad con diversidad tecnológica e innovación.

La importancia que deviene de la aplicación de una arquitectura de software basada en microservicios radica en los beneficios que aporta en los aspectos tecnológicos, de estandarización y reducción de las incidencias que provoca el desarrollo aplicaciones monolíticas.

De continuar con el desarrollo de aplicaciones web sobre una arquitectura monolítica se continuará presentando problemas de escalabilidad, rendimiento y mantenibilidad, este último implica la interrupción de actividades críticas de las unidades organizacionales que dependen de estas y aún más, en la entrega continua de los nuevos requerimientos.

Mediante la propuesta, el personal del área de desarrollo obtendrá una guía de construcción de aplicaciones web flexible, que permitirá la realización de su mantenimiento y despliegue de una forma sencilla y sin interrupciones para los usuarios internos y externos de la Asamblea Nacional.

1.5. Objetivos de la investigación

1.5.1. Objetivo General

Proponer una arquitectura de software basada en microservicios para el desarrollo de aplicaciones web en la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador.

1.5.2. Objetivos Específicos

- Identificar las tecnologías, metodología y arquitectura usadas en la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador, para el desarrollo de aplicaciones web.

- Identificar las tecnologías existentes para el desarrollo e implementación de microservicios en la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador.

- Estructurar una arquitectura de software basada en microservicios en la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador.

- Implementar un prototipo basado en la arquitectura desarrollada en la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador.

1.5.3. Hipótesis o preguntas directrices

Preguntas directrices

1. ¿Qué arquitecturas de software basadas en microservicios existen para el desarrollo de aplicaciones web?

2. ¿Qué tecnologías existen para la aplicación de arquitecturas basada en microservicios?

3. ¿Cómo construir una arquitectura de software basada en microservicios para el desarrollo de aplicaciones web en la CGTIC de la Asamblea Nacional?

4. ¿Cómo validar una arquitectura de software basada en microservicios para el desarrollo de aplicaciones web en la CGTIC de la Asamblea Nacional?

CAPÍTULO 2

MARCO REFERENCIAL

2.1. Introducción

En este capítulo se muestra los antecedentes de la investigación y referencias bibliográficas relacionadas con la arquitectura de software de microservicios y desarrollo de software para la web.

2.2. Antecedentes investigativos

La revisión de trabajos en los repositorios de los centros de educación superior a nivel nacional no arrojaron resultados relacionados con la arquitectura de microservicios, y se procedió con los internacionales.

Claus Djernæs Nielsen (2015) en la Universidad de Aarhus de Dinamarca, evalúa diferentes tácticas de disponibilidad y mantenibilidad en la construcción de un prototipo con arquitectura de microservicios, en el proceso, concluye con una revisión de las características de los microservicios enunciadas por Fowler y Lewis (2014) en cuatro criterios principales que definen a un microservicio:

- **Enfoque en las capacidades de negocio.** El proceso de desarrollo de software se vuelve más flexible bajo el enfoque de microservicios, cada microservicio está compuesto por su propia lógica de negocios, interfaz de usuario, base de datos y funcionalidad de comunicación con otros servicios, por lo tanto, en el equipo de desarrollo que cada miembro necesita experiencia en desarrollo de backend, frontend,

y bases de datos, permitiéndoles agregar nuevas características rápidamente sin arriesgar la estabilidad y el funcionamiento de todo el sistema.

- **Independencia de los servicios autónomos.** Cada servicio contiene su propia lógica de negocio y se despliega por separado. Esto hace posible cambiar y extender gradualmente con nuevas características sin afectar el resto del sistema. La propiedad autónoma permite la flexibilidad en la selección de la tecnología más adecuada para un servicio específico.

- **Gestión descentralizada de datos.** Todos los servicios tienen su propia base de datos en esquemas pequeños y simples, que se ven por separado. Los datos están desacoplados, lo que requiere mucha gestión y pruebas para garantizar que un sistema nunca actualice o elimine los datos en un servicio sin actualizar o eliminar los datos correspondientes en otros servicios que contienen los mismos datos o conjunto de referencias.

- **Tolerancia a fallos.** Los sistemas de microservicio consisten en múltiples unidades pequeñas que pueden fallar; Estas unidades están ligeramente acopladas y pueden ser restauradas automáticamente. lo que resulta en un sistema más estable y robusto.

Por su parte Borčin (2017)en su trabajo de investigación, realiza un análisis de los atributos de calidad de la arquitectura de microservicios profundizando en los retos de su implementación:

- **Interfaces y comunicación.** Cada microservicio debe exponer alguna interface, para algunas tecnologías no equipadas con una especificación, esto puede representar

serios problemas incluso REST que es utilizado en la arquitectura de microservicio para la comunicación no tiene una interfaz bien definida.

- **Transacciones y bloqueos.** Se necesita un conjunto completamente nuevo de operaciones para una transacción que comprueba la comunicación de microservicios. se debe asegurar de que ningún microservicio utilice flujos cerrados o recursos bloqueados y que no modifique los datos que otros servicios están utilizando. Estos bloqueos podrían provocar la suspensión de todo el sistema.

- **Programación coreográfica.** El paradigma de programación coreográfica puede ayudar en la creación de sistemas libres de bloqueos, sin errores de comunicación. La programación coreográfica puede llegar a ser más importante con el desarrollo ulterior de la arquitectura del microservicio, ya que proporcionan un terreno sólido para una formalización de la comunicación.

- **Puntos de entrada.** La seguridad es un problema importante cuando se trata de la arquitectura de microservicios y los sistemas distribuidos en general. En aplicaciones monolíticas se puede asegurar fácilmente la seguridad de toda la aplicación, ya que toda la comunicación pasa a través de los puntos de entrada que posee cada módulo que lo compone. En contraste, en microservicios, garantizar la seguridad se vuelve mucho más difícil. Toda la aplicación se compone de un microservicio independiente (lenguaje y máquina) que expone sus interfaces y el núcleo de la aplicación para la comunicación exterior, proporcionando nuevos puntos de entrada para los intrusos.

- **Red compleja.** Una extensa red de comunicación puede ser fácilmente expuesta a ataques, ya que es difícil administrar cuando la aplicación consta de muchos

microservicios y esos servicios envían miles de mensajes. El monitoreo de esta compleja red es también muy difícil.

Concluye que, la arquitectura de microservicios traerá nuevos problemas y desafíos relacionados no sólo con la comunicación y su estructura distribuida. Actualmente sólo se puede decir que los microservicios todavía están en una etapa inicial y aún es desconocido para muchos desarrolladores, pero está ganando mucha atención y su sitio en el mercado, especialmente entre una medianas y pequeñas empresas.

2.3. Marco Teórico

El análisis del marco teórico, parte de la revisión de la literatura de la Ingeniería de Software en la que se encuentra inmersa la Arquitectura de Software, los modelos base para una arquitectura de microservicios y su aplicación en el desarrollo de aplicaciones web, que permite obtener la teoría más relevante sobre los conceptos claves de la investigación, los cuales serán usados para el desarrollo de la propuesta.

2.3.1. Ingeniería de Software

La IEEE Radatz, Geraci, y Katki (1990) define a la ingeniería de software como: “La aplicación de un enfoque sistemático, disciplinado y cuantitativo para el desarrollo, operación y mantenimiento de software; Es decir, la aplicación de la ingeniería al software”.

2.3.1.1. Proceso de Ingeniería de Software

La ingeniería de software define un proceso no rígido (Pressman, 2005) para la construcción de software, consta de una serie de tareas y actividades interrelacionadas

Tabla 1: *Ámbitos temáticos*

Ámbito temático	**Temas**	**Objetivo de Uso**
Ingeniería de software	Proceso de Ingeniería de Software	Identificar el proceso de ingeniería y los modelos de procesos aplicados actualmente para el desarrollo de software.
	Modelos de Procesos de Ingeniería de Software	
	Metodologías de Desarrollo	Analizar las metodologías de desarrollo tradicionales versus las ágiles.
	Ciclo de vida del Software	Describir al diseño de software y sus artefactos (documentos) en el ciclo de vida del software.
	Diseño de Software	
Arquitectura de Software	Proceso de Arquitectura de Software	Describir el proceso de arquitectura de software, sus modelos o vistas para la descripción de una arquitectura de software.
	Modelos o Vistas Arquitecturales	
Microservicios	Diseño basado en el Dominio (DDD)	Aplicar los principio de Diseño basado en el dominio en la construcción de microservicios.
	Beneficios	Conocer los beneficios de la aplicación de microservicios.
	Topologías	Analizar las principales topologías para implementación de microservicios.
	Aplicación monolítica	Describir una aplicación monolítica.
	Comparación de arquitecturas monolíticas y de microservicios	Comparar la estructura de una aplicación monolítica versus una de microservicios.
	Servicios web Integración continua Contenerización de aplicaciones Despliegue continuo Descubrimiento de servicios Balanceo de carga Monitoreo	Establecer las bases de conceptos y tecnologías relacionadas con la aplicación de una arquitectura de microservicios para el desarrollo de software.
Aplicaciones web		Identificar que es una aplicación web.

Fuente: Elaboración propia

en donde los requerimientos de entrada a través de actividades de transformación generan resultados (Bourque y Dupuis, 2014). En síntesis establece cuatro actividades fundamentales (Rajlich, 2011; Sommerville, 2010):

- Especificación del software, donde se establecen los requerimientos.
- Diseño e implementación del software, que es la construcción misma del software.
- Validación del Software, que comprueba su funcionalidad enfocada en la satisfacción de los requerimientos establecidos.
- Evolución del software, añadiendo nuevas capacidades y funcionalidades. Los cambios de software son los componentes básicos de la evolución del software.

2.3.1.2. Modelos de Proceso de Software

Cada software que se construye es único y corresponde al dominio de su aplicación, por esta razón, es necesario contar con modelos que de forma simplificada describan los procesos de software. Básicamente, un modelo describe las tareas, sus relaciones, sus resultados y a las partes interesadas (stakeholders) que participan en ellas y, como todos los modelos, presenta una abstracción y simplificación de la realidad (Rajlich, 2011).

Existen dos enfoques analizados expuestos por Bourque y Dupuis (2014) en el documento “Guide to the Software Engineering Body of Knowledge” (SWEBOK) :

- El tradicional o lineal, del cual derivan los modelos cascada, iterativo y por componentes.
- El ágil, de un modelo interativo-incremental.

2.3.1.3. Metodologías de Desarrollo Ágil

Tradicionalmente, se ha utilizado un proceso de cascada al ejecutar proyectos de software, partiendo primero de la definición de requisitos, planificación del proyecto en su totalidad, diseño del software y finalmente se construye y prueba el producto. Una gran cantidad de software se ha construido de esta manera. Pero a medida que pasaba el tiempo, las compañías seguían enfrentándose al mismo tipo de problemas lo cual lo atribuyeron al propio proceso de desarrollo en cascada. La historia del desarrollo ágil se inició cuando se buscaba una forma diferente de pensar sobre la resolución de estos problemas, iniciando con el Manifiesto para el Desarrollo de Software Ágil:

- Los individuos e interacciones por encima de los procesos y las herramientas.
- Software funcionando por encima de la documentación.
- La colaboración del cliente por encima de la negociación del contrato.
- La respuesta al cambio por encima del seguimiento de un plan. (Herrera Uribe y Valencia Ayala, 2007).

El desarrollo de software ágil es un conjunto de métodos y metodologías que ayudan a un equipo a pensar de manera más eficaz, trabajar más eficientemente y tomar mejores decisiones. Estos métodos y metodologías abordan todas las áreas de la ingeniería de software tradicional, incluyendo gestión de proyectos, diseño y arquitectura de software y mejora de procesos. Estos métodos y metodologías consisten en prácticas racionalizadas y optimizadas para que sean fáciles de adoptar. También implica un cambio de mentalidad que puede hacer una gran diferencia en la eficacia con que un equipo utiliza dichas prácticas, esta

Tabla 2: *Metodologías tradiciones vs ágiles*

Tradicionales	**Ágiles**
Predictivos	Adaptativos
Orientados a procesos	Orientados a personas
Proceso rígido	Proceso flexible
Se concibe como un proyecto	Un proyecto es subdividido en varios proyectos más pequeños
Poca comunicación con el cliente	Comunicación constante con el cliente
Entrega de software al finalizar el desarrollo	Entregas constantes de software
Documentación extensa	Poca documentación

Fuente: (Cadavid, 2013).

mentalidad ayuda al equipo a compartir información entre sí, con el fin de tomar decisiones importantes de proyectos en conjunto, en lugar de tener un gerente que tome todas las decisiones por sí solo. La mentalidad ágil consiste en abrir la planificación, el diseño y la mejora de procesos a todo el equipo el cual comparte la misma información (Stellman y Greene, 2014).

En la siguiente tabla 2.2, se presenta los aspectos más relevantes de las metodologías de desarrollo tradicional comparadas con las de desarrollo ágil:

De acuerdo con el trabajo realizado por A B M Moniruzzaman y Syed Akhter Hossain (2013), las principales razones para la adopción de enfoques ágiles se relacionan con: los cambios rápidos, la necesidad de resultados rápidos y los requerimientos emergentes; y por otro lado, estas metodologías tienen numerosas ventajas, entre ellas: adaptarse muy bien al cambio y al dinamismo; están orientadas a las personas y orientadas al valor, en lugar de orientadas a los procesos y los planes; mitigar los riesgos mediante la demostración de valores y funcionalidades en el proceso de desarrollo; proporcionar un tiempo de comercialización más rápido; mejorar la productividad (reduciendo la documentación) y si un proyecto no es factible, su falla es temprana o rápida y sin repercusiones extremas.

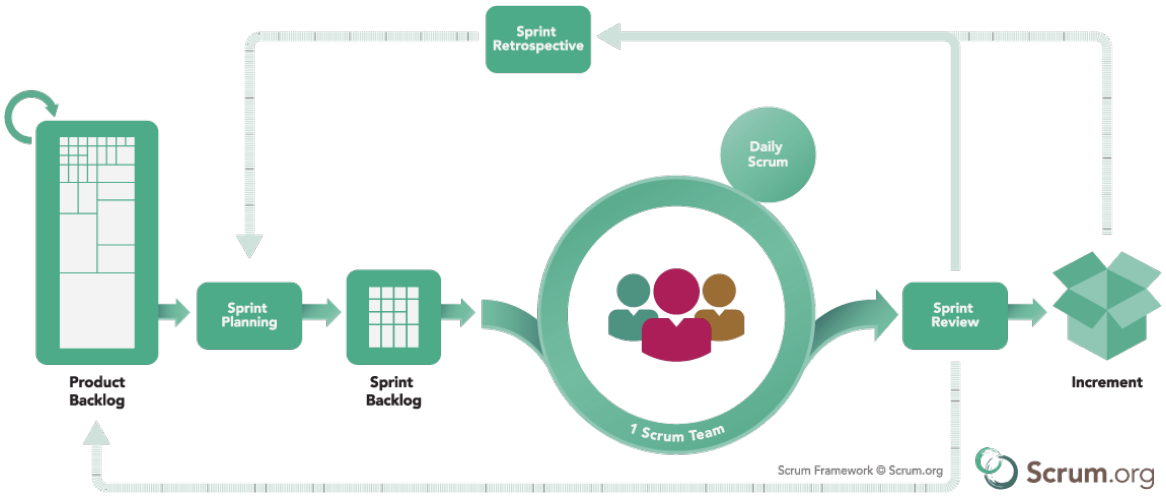


Figura 1: Marco de trabajo SCRUM
Fuente: (Scrum.org, 2016)

SCRUM. Es un macro de trabajo para gestión de proyectos de desarrollo de software, el cual se basa en la definición de roles, eventos, artefactos y reglas para la utilización conjunta de estos elementos.

Como principios SCRUM emplea la simplicidad y escalabilidad al no establecer prácticas de ingeniería del software y combinarse con otras prácticas de ingeniería, metodologías, estándares existentes dentro de la organización. Se centra a nivel de las personas y el equipo de desarrollo que es quien construye el producto, trabajando en equipo de forma eficiente para la obtención de productos complejos y sofisticados (Lina Maria Montoya Suarez, Jorge Mauricio Sepúlveda Castaño, y Luisa María Jiménez Ramos, 2017).

De acuerdo con Scrum.org (2016), el marco de trabajo lo describe de la siguiente manera:

Roles Principales

- **Product Owner,** se asegura de que el Scrum Team trabaje en perspectiva del negocio. Escribe y prioriza las historias de usuario que forman parte del Product Backlog.

- **Scrum Master (o Facilitador),** responsable de eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El Scrum Master no es el líder del equipo puesto que el equipo es auto-organizado, observa que las reglas se cumplan.

- **Scrum Team,** responsable de la entregar el producto conformado por 5 a 9 personas con habilidades transversales necesarias para el trabajo (análisis, diseño, desarrollo, pruebas, documentación, entre otras).

Eventos

- **Sprint**, período de tiempo de trabajo (duración de 2 o 3 semanas, máximo 4) ajustable al ritmo de trabajo del equipo, aunque sin relajarlo demasiado. Al finalizar un sprint, el equipo presenta los avances logrados, que como producto es potencialmente entregable hacia el cliente.

- **Daily Scrum,** se realiza una reunión rápida (15 minutos) sobre el estado del proyecto. respondiendo a las preguntas: Qué se hizo ayer? ¿Qué se hará hoy? ¿Existe algún problema que impida alcanzar un objetivo?.

- **Sprint Planning,** se realiza al inicio de cada ciclo de Sprint, con el fin de planificar el Sprint, sobre : qué se hará, preparación del Sprint Backlog detallando el tiempo y el esfuerzo del trabajo.

- **Sprint Review,** se revisa el trabajo que si o no se ha completado. Se realiza la presentación del trabajo completado a los interesados a través de una demostración.

- **Sprint Retrospective,** se realiza una retrospectiva del sprint, en la que todos dejan sus impresiones sobre el reciente sprint realizado para mejora continua del proceso.

Documentos

- **Product backlog,** es un documento de alto nivel usado en todo el proyecto. Consta de todos los requisitos de proyecto con descripciones genéricas de funcionalidades deseables y priorizadas. Es abierto administrador por el product owner.
- **Sprint backlog,** es un subconjunto de requisitos a ser desarrollados en el siguiente sprint.
- **Burn down/up chart** se representa con gráfica que mide el número de requisitos del Backlog con lo que permite ver el progreso del proyecto.

2.3.1.4. Ciclo de Vida del Software

El ciclo de vida de software (SLC), proporciona una secuencia de actividades a seguir para el desarrollo o mantenimiento de software (Mishra y Dubey, 2013). Incluye varias fases, que parten desde el análisis preliminar hasta la prueba y evaluación post-desarrollo del software. Adicionalmente, consiste en modelos y metodologías que se utilizan para desarrollar los sistemas de software por parte del equipo, usando las metodologías como un marco para planificar y controlar todo el proceso de desarrollo (Leau, Loo, Tham, y Tan, 2012).

El estándar ISO/IEC/IEEE (2008)y Bourque y Dupuis (2014), define grupos de procesos del contexto del sistema y los específicos de software; en este último se define el proceso de implementación de software, que consta de:

- **Proceso de Análisis de Requisitos de Software.** Establece los requisitos de los elementos de software del sistema.

- **Proceso de diseño:**
  - **Arquitectónico de software.** Proporcionar un diseño para el software que implementa y se puede verificar en función de los requisitos.
  - **Detallado de software.** Proporciona un diseño para el software que implementa y se puede verificar en función de los requisitos y la arquitectura del software y es suficientemente detallado para permitir la codificación y las pruebas.
- **Proceso de construcción de software.** Produce unidades de software ejecutables que reflejen adecuadamente el diseño del software.
- **Proceso de Integración de Software.** Combina las unidades de software y los componentes de software, produciendo elementos de software integrados, acordes al diseño del software, demostrando que los requisitos de software funcionales y no funcionales se satisfacen en una plataforma operacional equivalente o completa.
- **Proceso de Pruebas de Calificación de Software.** Confirma que el producto de software integrado cumple con sus requisitos definidos.

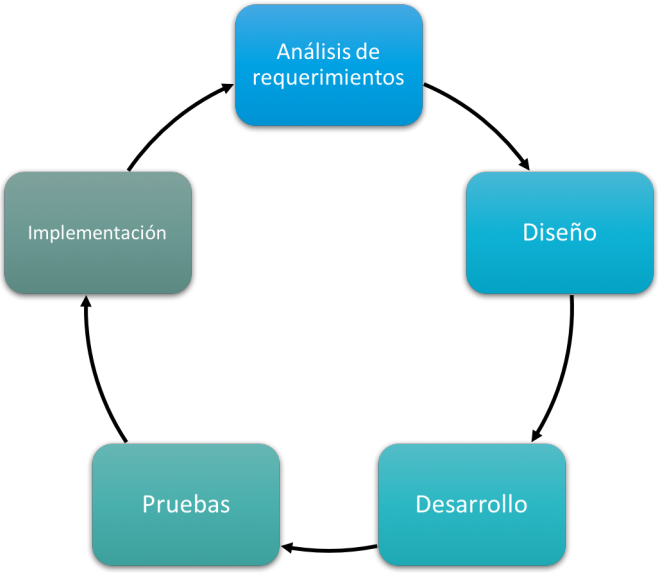


Figura 2: Ciclo de vida del Software

Fuente: (Mishra y Dubey, 2013).

Cada sistema de software tiene un proceso de ciclo de vida, este proceso describe todas las tareas en un proyecto de ingeniería de software de principio a fin y las mejores prácticas a implementar para asegurar que el proyecto se está ejecutando sin problemas, cumplir los plazos que se requieren y con código de calidad. (Foster, 2014; Stephens, 2015)

Existen varios modelos de ciclo de vida que se han desarrollado, estos modelos se han categorizado bajo dos enfoques: el tradicional y ágil, este último ha ganado gran aceptación en desarrollo de software por su diseño para soportar aplicaciones que cambian rápidamente durante el proceso de desarrollo, con el fin de entregar el software de forma rápida a los clientes que pueden verlo y utilizarlo y a la vez proponer cambios para su inclusión en las iteraciones posteriores del sistema (Narang, 2015).

Una característica distintiva de los diferentes modelos de ciclo de vida de desarrollo de software es la forma en que se gestionan los requisitos de software. Los modelos de

desarrollo lineales desarrollan típicamente un conjunto completo y controlado de requisitos de software, en la medida de lo posible, durante etapa inicial del proyecto (Bourque y Dupuis, 2014).

En un modelo incremental, se produce incrementos sucesivos de trabajo y entregas de software basado en la partición de los requisitos de software a implementar en cada ciclo a medida que el producto de software evoluciona. Los modelos ágiles pueden definir el alcance del producto y las características de alto nivel inicialmente; Sin embargo, los modelos ágiles están diseñados para facilitar la evolución de los requisitos de software durante el desarrollo del proyecto (Strydom, 2017).

2.3.1.5. Diseño de Software

Como parte de ciclo de vida del software, el diseño es una representación o modelo del software que se va a construir. Su propósito es permitir a los programadores implementar los requisitos, designando las partes proyectadas de la implementación. Durante el proceso, se obtiene un conjunto de documentos que contienen texto descriptivo y diagramas que sirven de base para que una aplicación pueda ser completamente programada. Un diseño de software completo debería ser tan explícito que un programador podría codificar la aplicación sin necesidad de otros documentos. Pueden entenderse en dos partes: diseño de alto nivel, a menudo denominado "arquitectura de software", que es generalmente indispensable, y todo otro diseño, denominado "diseño detallado". Puede ser beneficioso hacer diseños muy detallados, a falta de ser código real (Braude y Bernstein, 2016).

En un principio, el diseño es desordenado incluso si se entienden completamente los requisitos del problema. Normalmente se plantea muchas alternativas cuando se está

diseñando una solución de software, a la vez, suele cometerse muchos errores antes de llegar a una solución, mientras que el diseño cambiará a medida que se comprenda mejor el problema con el tiempo.

Se tiene que dar prioridad a ciertos requisitos e intercambiar un subconjunto por otro por la limitación de tiempo que la mayoría de los proyectos de software tiene, en consecuencia, el diseño se compone de intercambios y prioridades.

Para la mayoría de los proyectos no hay un conjunto de reglas que indiquen el diseño de un componente “X” bajo una técnica “Y”, el diseño de software se realiza utilizando un conjunto de heurísticas (reglas empíricas) y patrones que permiten obtener rápidamente los elementos fáciles de un diseño y llegar al núcleo del problema.

Finalmente, para cualquier problema, sea fácil o complejo, se establece que los requisitos cambiarán con el tiempo, lo que produce un efecto cascada en el diseño, por lo que este evolucionará con el tiempo, por lo que es importante en esta etapa crear una arquitectura de software que sea susceptible de cambiar y que tenga poco impacto en el diseño y código descendente. (Dooley, 2011)

2.3.2. Arquitectura de software

“El estándar IEEE 1471 define la arquitectura de software como la organización fundamental de un sistema incorporado en sus componentes, sus relaciones entre sí y con el medio ambiente y los principios rectores de su diseño y evolución”(Sherland, Wade, Emery, y Hilliard, 2000)La arquitectura se ocupa fundamentalmente en la organización de un sistema en sus elementos constitutivos y sus interrelaciones para lograr un propósito determinado (Sangwan, 2014). Bajo esta perspectiva, se puede afirmar que, cada sistema

tiene una arquitectura, pero una adecuada arquitectura que permite a un sistema logre la finalidad para la que fue creado. Por otra parte los sistemas no solo se limitan al hardware, sin que, también incluyen a personas, instalaciones, software, políticas y documentos, ya que toso estos elementos son requeridos para producir el resultado esperado.

2.3.2.1. Proceso de Arquitectura de Software

El proceso de diseño de la arquitectura es una extensión del proceso general de diseño de ingeniería. El diseño de la arquitectura se centra en la descomposición de un sistema en componentes y las interacciones entre los componentes para satisfacer los requisitos funcionales y no funcionales. Un sistema de software puede ser visto como una jerarquía de las decisiones de diseño (reglas también llamado diseño o contratos). Cada nivel de la jerarquía tiene un conjunto de reglas de diseño que de alguna manera se une o conecta los componentes en ese nivel. La salida principal del proceso de diseño de la arquitectura es una descripción arquitectónica. El proceso describe los siguientes pasos generales:

- **El establecimiento de requisitos,** mediante el análisis de los controladores del negocio, el contexto del sistema, y los factores que los interesados del sistema estimen crítico para el éxito.
- **La definición de una arquitectura,** mediante el desarrollo de estructuras arquitectónicas y estrategias de coordinación que satisfagan los requisitos.
- **La evaluación de la arquitectura,** mediante la determinación de cuándo y qué métodos de evaluación de la arquitectura son apropiados, la realización de las evaluaciones, y la aplicación de los resultados para mejorar el desarrollo de la

arquitectura.

- **La documentación de la arquitectura,** con el suficiente detalle y en una forma fácilmente accesible para los desarrolladores y otras partes interesadas.
- **El análisis de la arquitectura,** para el rendimiento del sistema, la seguridad.
- **Implementación,** de la arquitectura definida a través del desarrollo de un prototipo.

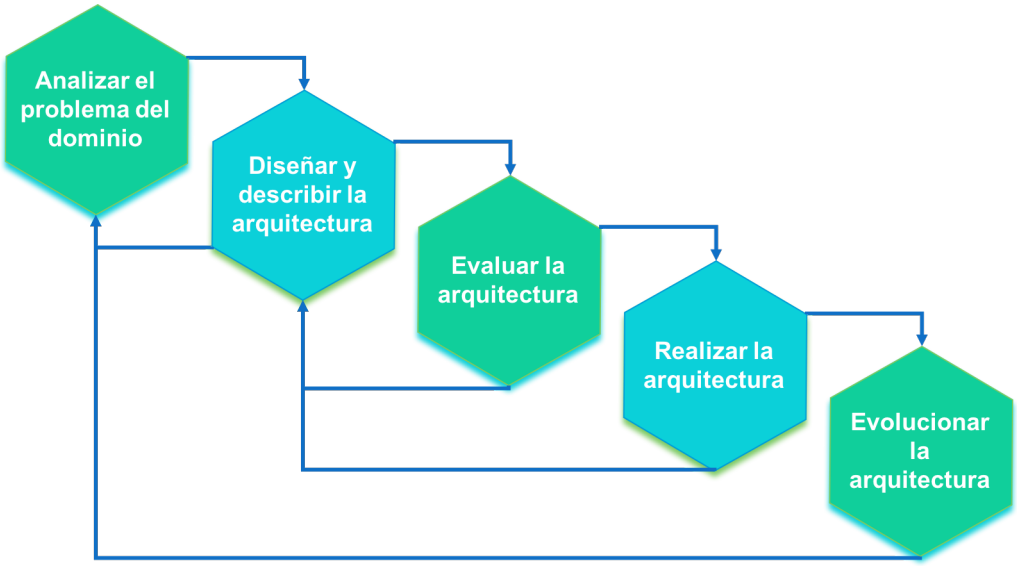


Figura 3: Modelo del proceso de arquitectura de software

Fuente: (Babar, Brown, y Mistrík, 2013).

2.3.2.2. Modelos o Vistas Arquitecturales

La documentación de una arquitectura deben plasmarse de tal forma que sea entendible tanto por el equipo de desarrollo así como por el resto de participantes del proyecto, incluidos los clientes. Existen muchas formas de describir una arquitectura, por ejemplo mediante ADL (Architecture Description Language), UML, Agile Modeling, IEEE 1471 o

el modelo 4+1. El modelo 4+1 describe una arquitectura software mediante 4 vistas (de la Torre Llorente, Castro, Barros, y Nelson, 2010):

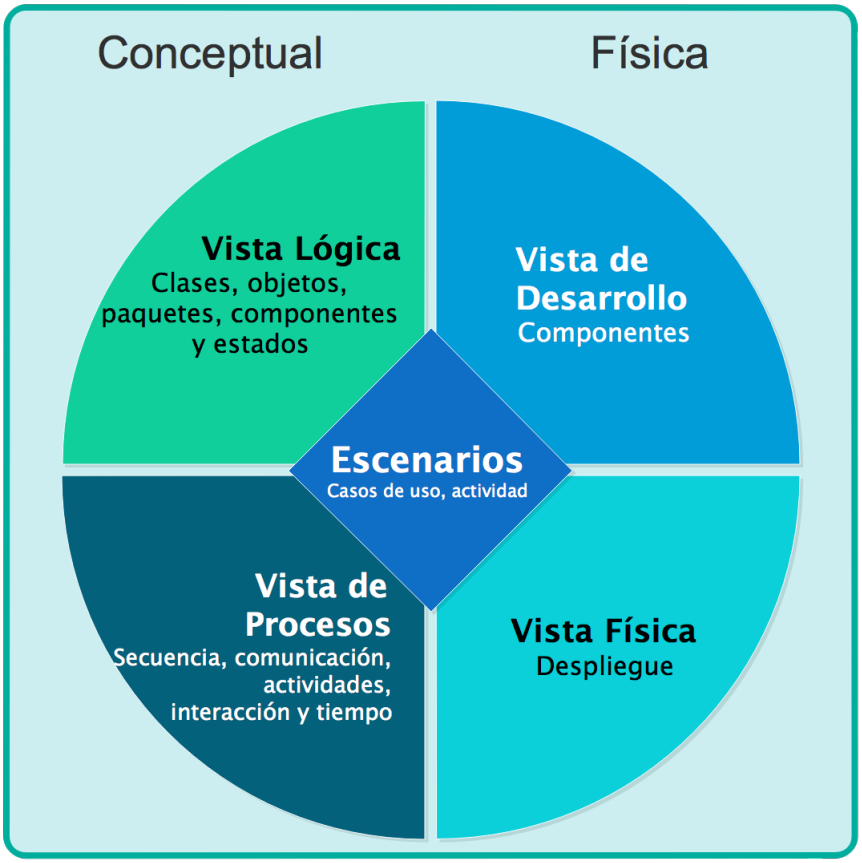


Figura 4: Modelo de 4+1.

Fuente: (de la Torre Llorente et al., 2010).

- **Vista lógica:** muestra la funcionalidad del sistema hacia el usuario final (diagramas de clases, de comunicación y de secuencia).

- **Vista del proceso:** muestra el comportamiento del sistema en ejecución, sus procesos activos y la comunicación entre ellos. Se observa la concurrencia, escalabilidad, rendimiento, (cualquier característica dinámica del sistema).

- **Vista física:** muestra la distribución de los componentes de software en los nodos

físicos de infraestructura y su comunicación (diagramas de despliegue).

- **Vista de desarrollo:** muestra el sistema desde el punto de vista de los desarrolladores, centrandose en el mantenimiento del software (diagramas de componentes y de paquetes).

- **Escenarios:** describen secuencias de interacciones entre objetos y procesos y son usados para identificar los elementos arquitecturales y validar el diseño.

2.3.2.3. Diseño basado en el dominio (DDD)

DDD subraya la necesidad de centrarse en el dominio por encima de cualquier otra cosa cuando se trabaja en la creación de software complejo. Los expertos en el dominio del problema trabajan con el equipo de desarrollo para centrarse en las áreas del dominio que son útiles para poder producir software valioso. Por ejemplo, al crear un software para la industria de la salud para registrar el tratamiento del paciente, no es importante aprender a ser médico, lo que es importante entender es la terminología de la industria de la salud, cómo los diferentes departamentos ven a los pacientes y la atención, qué información recolectan los médicos y qué hacen con ella (Millett, 2015).

Para el diseño basado en el dominio, el núcleo radica en la comprensión del dominio en el que se coloca el software, ya que los desarrolladores de ese software no son expertos en se dominio, por lo que es necesario recabar información de otras personas que son expertos. Esta realización significa que, es necesario optimizar el proceso de desarrollo para recopilar e incorporar este aporte (Fehre, 2015).

DDD proporciona los conceptos clave requeridos y determina el factor clave de éxito en la construcción de aplicaciones basadas en microservicio a través de la representación del

mundo real en la arquitectura. El diseño basado en el dominio ofrece principios, patrones, actividades y ejemplos de cómo construir un modelo de dominio, pero no proporciona un proceso de desarrollo detallado y sistemático para aplicar estos principios y patrones ni los clasifica en el campo de la ingeniería de software (Steinegger, Giessler, Hippchen, y Abeck, 2017).

De acuerdo con el estudio de (Landre, Wesenberg, y Rønneberg, 2006; Vernon, 2016), básicamente, el diseño basado en el dominio se divide en tres áreas:

- **Bloques de construcción básicos:** trata de cómo el dominio está separado de la tecnología mediante el uso de una arquitectura en capas, combinada con patrones prácticos de diseño orientado a objetos.

- **Modelos sofisticados:** aborda cómo el software está alineado con el pensamiento del experto del dominio, los conceptos del dominio se hacen explícitos en el código y la refactorización del código es conducida por la penetración del dominio.

- **Diseño estratégico:** aborda la integridad del modelo y la gestión de la complejidad en sistemas grandes. El diseño estratégico proporciona tres bloques básicos:

    - Mapeo contextual, es una gráfica que documenta los contextos de modelado y sus relaciones. Los sistemas grandes contienen múltiples contextos de modelado, por lo que se representa el contexto de modelaje de intereses, no las aplicaciones o sistemas de información que implementan los diferentes contextos.

    - Destilación , consiste en separar lo importante de lo menos importante. Idealmente, debería ser posible identificar el área problemática que motiva el

desarrollo real del software. Se conoce como el dominio principal, separando algunas funciones las cuales deben ser movidas fuera, lo que permite dejar que la gente más especializada se centre en lo principal.

- Estructuras a gran escala, en sistemas a gran escala son difícil coordinación, aún más difícil es llegar a un tema unificador que haga que todos tengan la misma idea. Cuando este tema unificador falta, los desarrolladores encuentran difícil comunicarse y entender el sistema, por lo que se necesita de un conjunto de reglas o patrones que pueden abarcar todo el sistema y así permitir a los desarrolladores captar el sistema como un todo.

2.3.3. Arquitectura Hexagonal

La arquitectura hexagonal conocida también como de puertos y adaptadores, en donde, una aplicación en su conjunto se ve como un hexágono, con el dominio de negocios ubicado en el interior. Mientras que el dominio empresarial sólo se preocupa por su propio lenguaje y conceptos, utiliza puertos para comunicarse hacia el exterior. Los puertos son las interfaces con el mundo exterior y establecen un API que establece lo que se necesita y cómo se debe proporcionar. Por otro lado, hay adaptadores, es decir, elementos que proporcionan la API. Esto le da mucha flexibilidad, y no sólo le permite intercambiar adaptadores, por ejemplo durante una prueba o en el caso de probar diferentes tecnologías rápidamente para encontrar la mejor, para no adivinar, sino probarlo con la aplicación (Fehre, 2015).

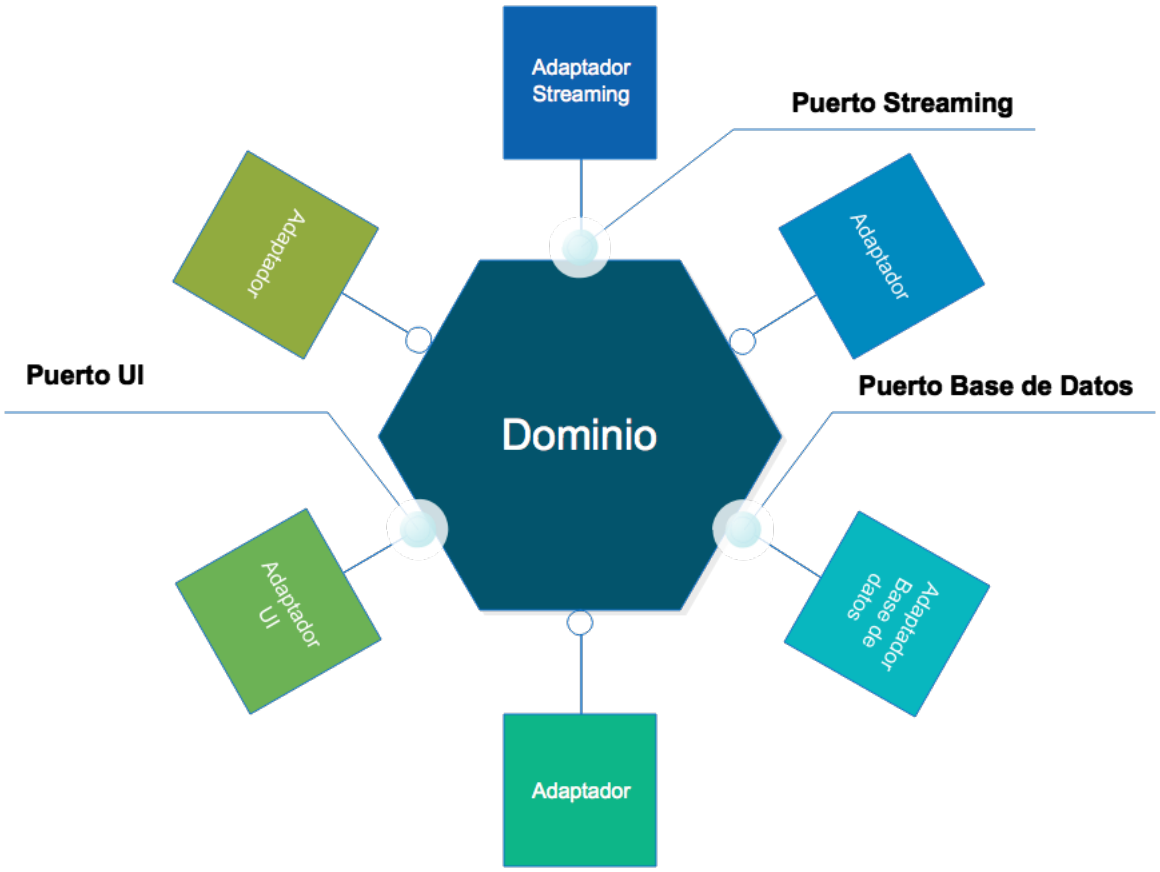


Figura 5: Ejemplo de una arquitectura hexagonal

Fuente: Adaptado de del libro Javascript DDD (Fehre, 2015)

Actualmente los puertos ya han sido implementados por lo que sólo se hacen uso de ellos, se puede pensar que un puerto es un protocolo como el HTTP, y que el adaptador es una clase de tipo Servlet de Java que recibe llamadas desde un contenedor, en este caso de tipo JEE o por una implementación de un framework de servicios web RES.

El diseño usando "hexágonos" requiere que se tenga en cuenta los casos de uso y no el número de clientes soportados, ya que cualquier número y tipo de cliente puede ser requerido a través de varios puertos y adaptadores, pero cada adaptador delega a la aplicación a través de la misma API.

El hexágono, representa la limitación de un caso de uso (o una historia de usuario), lo que quiere decir que, se debe crear casos de uso basados en los requerimientos funcionales de la aplicación, mas no en el número de clientes o los mecanismos de salida (Vernon, 2016).

2.3.4. Microservicios

Es un enfoque para el desarrollo de una aplicación única como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y mecanismos ligeros de comunicación, a menudo un recurso de una interfaz de programación de aplicaciones (API) sobre protocolo de transferencia de hipertexto (HTTP). Estos servicios están construidos alrededor de las capacidades del negocio y con independencia de despliegue e implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, los que pueden estar escritos en lenguajes de programación diferentes y utilizar diferentes tecnologías de almacenamiento de datos (Fowler y Lewis, 2014).

El término microservicios no es relativamente nuevo, este estilo arquitectural fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una definición en concreto para microservicio, sin embargo una aproximación que la realiza (Newman, 2015) lo define como: “Pequeños servicios autónomos que trabajan juntos”.

Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto-contenidas, lo cual difiere de la forma tradicional o monolítico (Wolff, 2016).

Una de las ventajas de utilizar microservicios es la capacidad de publicar una aplicación grande como un conjunto de pequeñas aplicaciones (microservicios) que se

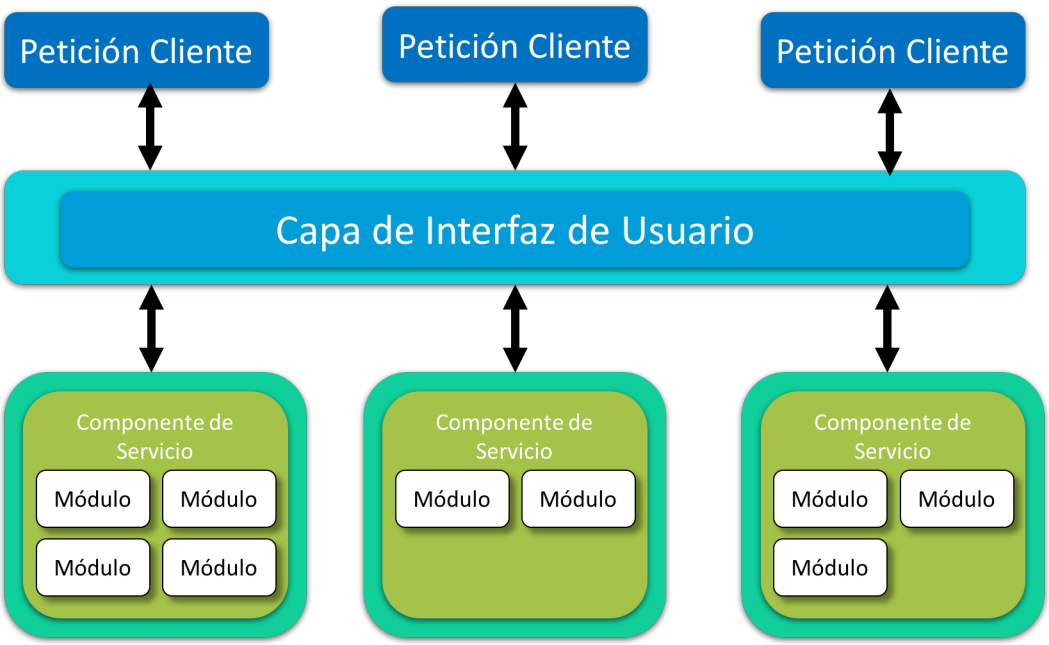


Figura 6: Patrón básico de arquitectura de Microservicios
Fuente: (Mark Richards, 2015).

pueden desarrollar, desplegar, escalar, manejar y visualizar de forma independiente. Los microservicios permiten a las empresas gestionar las aplicaciones de código base grande usando una metodología más práctica donde las mejoras incrementales son ejecutadas por pequeños equipos en bases de código y despliegues independientes. La agilidad, reducción de costes y la escalabilidad granular, traen algunos retos de los sistemas distribuidos y las prácticas de gestión de los equipos de desarrollo que deben ser considerados.(Villamizar et al., 2015).

2.3.4.1. Beneficios

Un enfoque de microservicios provee de una serie de beneficios tales como:

Intensa modularización. Básicamente, la arquitectura de microservicios consiste en dividir una aplicación o sistema en partes más pequeñas. La modularización facilita

la automatización y proporciona un medio concreto de abstracción. Los servicios modularizados son desplegables de forma independiente y ayudan en la velocidad de la que se entrega del software (Nadareishvili, Mitra, McLarty, y Amundsen, 2016).

Intercambiabilidad. Los microservicios se pueden reemplazar más fácilmente que los módulos en un sistema monolítico. Si un nuevo servicio ofrece la misma interfaz, puede reemplazar el microservicio existente, el cual puede utilizar una base de código diferente e incluso diferentes tecnologías, siempre y cuando presente la misma interfaz, lo que en un sistema monolítico puede ser difícil o imposible de lograr. La fácil sustitución de los microservicios reduce los costes de decisiones incorrectas, si un microservicio fue construido bajo la decisión de una tecnología o enfoque, este puede ser completamente reescrito si surge la necesidad. Esta necesidad de reemplazar el código en el futuro, frecuentemente se descuida durante el desarrollo de los sistemas de software (Wolff, 2016).

Desarrollo sostenible. El desarrollo de software sostenible se debe gracias a la intensa modularización y la fácil sustitución o intercambiabillidad ya que los microservicios no están vinculados a una tecnología específica, lo que permite la utilización de nuevas tecnologías apropiadas para cada problema, evitando su caducidad o deterioro por la acción de su mantenimiento correctivo, evolutivo o adaptativo (dependiendo de qué se quiere perfeccionar).

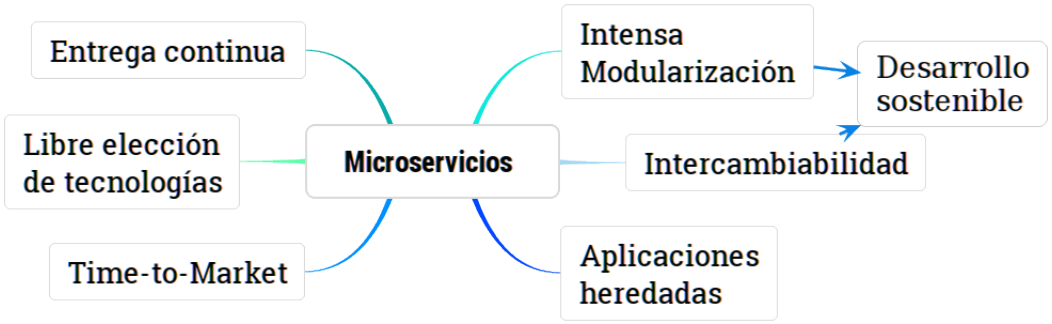


Figura 7: Beneficios de los microservicios

Fuente: (Wolff, 2016)

Entrega continua. Lo que hace especial la modularización, es que cada microservicio puede ser desplegado de manera independiente. Concretamente, los microservicios pueden ser, por ejemplo: contenedores (Docker) en los que está instalado el software que constituye el microservicio que forma parte de un sistema de software general, siendo alternativamente, partes de una página HTML con llamadas JavaScript ó pueden ofrecer servicios a través de REST, que pueden ser utilizados por otros microservicios u otros sistemas externos o clientes móviles. Estos microservicios pueden ofrecer una interfaz de usuario, o llamar a otros que implementan la lógica de negocio (Wolff, 2017).

Libre elección de tecnologías. A diferencia de una aplicación monolítica en donde todos necesitan estar de acuerdo en un lenguaje de programación, frameworks o librerías, incluso de una versión específica de dichos frameworks, con un enfoque de microservicios la aplicación puede estar compuesta de servicios independientes que aprovechan diferentes frameworks, versiones de bibliotecas e incluso plataformas de sistema operativo completamente diferentes (Scholl, Swanson, y Fernandez, 2016).

Time-to-market. Los microservicios reducen el tiempo de lanzamiento al mercado (time-to-market) pudiendo permitir realizar cambios sólo en aquellos que requieren de un

cambio, cada equipo a cargo del desarrollo de uno o varios microservicios puede desarrollar e implementar características sin la coordinación de tiempo con el resto de equipos, lo que permite trabajar en paralelo y poner en producción las nuevas funcionalidades en corto tiempo.(Wolff, 2017)

2.3.4.2. Estrategias de descomposición de aplicaciones (microservicios)

Existen cuatro criterios genéricos para la descomposición en componentes de servicio (microservicios) que los expone Carneiro y Schmelmer (2016):

Descomposición por funcionalidades: se aplica cuando se divide una aplicación de software analizando las funcionalidades que se pretende proporcionar. Una vez identificado las piezas individuales de funcionalidad que todo el sistema debe proporcionar, se comienza por agruparlas en paquetes que encajan lógicamente. Algunos ejemplos de funcionalidades que pueden ser candidatos a ser servicio independientes son el correo electrónico, monitoreo de recursos, o la conexión y el consumo de fuentes de datos externos a través de API's como el web scraping de un sitio web, entre otros.

Descomposición por madurez: Otro método para identificar partes de una aplicación es buscar partes que han alcanzado un nivel similar de madurez. Normalmente, la organización de una empresa pasa por varias iteraciones en la definición de objetivos y especificaciones, mismos que se reflejan en las funcionalidades y comportamientos de partes de la aplicación en cuestión. Las partes cuyos requerimientos funcionales y no funcionales se entienden bien deben agruparse, mientras que las cosas que son menos estables y sujetas a iteraciones más y más rápidas podrían pasar a otros servicios separados y versionados (o no se dividen en servicios separados en absoluto, inicialmente). Por ejemplo, una aplicación de lectura de

noticias (social feed), en donde su función principal es leer y actualizar las noticias el cual es independiente de cualquier otra función, mientras que los componentes de comentarios y valoraciones se repiten constantemente por lo que no deben formar parte del mismo servicios principal (administrador de noticias).

Descomposición por el patrón de acceso a datos: Se centra en la recuperación de datos y la eficacia de la persistencia. Aquí se analizar el acceso de almacenamiento esperado para las distintas funcionalidades y se clasifican en "lectura intensiva", "escritura intensiva" y "balanceo de lectura y escritura". La ventaja de este enfoque es que los esquemas de un servicio, las consultas de acceso, o incluso el motor de almacén de datos (RDBMS, almacén de valores clave, basado en documentos u otro método) pueden optimizarse para el acceso de lectura o escritura intensiva en aislamiento. Siguiendo con el ejemplo del lector de noticias: los datos de perfil de usuario y la lista de lectura de un usuario están estrechamente relacionados, pero los datos de perfil se leen a menudo, mientras que la lista de lecturas se actualiza mucho más de lo que se accede para mostrar (se actualiza con tanta frecuencia como se lee), por lo que es necesario un balanceo de escritura, lectura.

Descomposición por Contexto: se basa en que, en varias ocasiones tiene sentido implementar múltiples servicios que parecen centrarse exactamente en la misma entidad. Esto es apropiado si representan significados distintos en contextos separados, este es uno de los conceptos clave de un enfoque de diseño arquitectónico denominado diseño dirigido por el dominio (DDD). Martin Fowler(2014), explica que, por ejemplo, un cliente puede ser polisémico. Un cliente puede significar dos cosas diferentes en diferentes contextos de negocio; los atributos y la lógica de negocio asociados con un cliente diferirán grandemente para el departamento de asistencia al cliente de una organización de los atributos y de

las asociaciones en el contexto del negocio del departamento de ventas. Tal separación de intereses (separation of concerns - SoC) podría ser tan extensa que los únicos datos compartidos entre tales servicios centrados en el cliente serían un identificador de cliente único y común (UID). Ambos servicios de cliente específicos de dominio también podrían hacer referencia cruzada a un tercer servicio de cliente de nivel inferior para datos compartidos(la dirección del cliente por ejemplo).

2.3.4.3. Topologías

Según Mark Richards (2015), existen muchas maneras de implementar un patrón de arquitectura de microservicios, pero se destacan tres topologías principales que son las más comunes y populares: basada en API REST Transferencia de Estado Representacional (del inglés Representational State Transfer), basada en aplicaciones REST y la centralizada de mensajería.

- La topología basada en API REST es útil para sitios web que exponen servicios individuales pequeños y autónomos a través de algún tipo de API. Consta de componentes de servicio de grano fino (de ahí el nombre microservicios) que contienen uno o dos módulos que realizan funciones empresariales específicas independientes del resto de los servicios. En esta topología, estos componentes de servicio de grano fino se acceden normalmente mediante una interfaz basada en REST implementada a través de una capa de API basada en la Web desplegada separadamente. Algunos ejemplos de esta topología incluyen algunos de los servicios web REST basados en la nube como los usados por Yahoo, Google y Amazon.

- La topología basada en aplicaciones REST difiere del enfoque basado en API REST

en que las solicitudes de cliente se reciben a través de pantallas de aplicaciones empresariales tradicionales basadas en web o en clientes pesados (fat-client) en lugar de una simple capa de API. La capa de interfaz de usuario de la aplicación se despliega como una aplicación web separada que accede de forma remota a componentes de servicio desplegados por separado (funcionalidad empresarial) a través de simples interfaces basadas en REST. Otra diferencia se encuentra en que los componentes de servicio tienden a ser más grandes, de grano más grueso y representan una pequeña porción de la aplicación empresarial general en lugar de granos finos, servicios de acción simple. Esta topología es común para las aplicaciones empresariales pequeñas y medianas que tienen un grado relativamente bajo de complejidad.

- La topología de mensajería centralizada, es similar a la basada en REST, excepto que en lugar de usar REST para acceso remoto, esta utiliza un intermediario de mensajes centralizado ligero (por ejemplo, ActiveMQ, HornetQ, etc.). Es importante considerar que no se debe confundir con el patrón SOA o considerarlo "SOA-Lite". El agente de mensajes ligeros que se encuentra en esta topología no realiza ninguna orquestación, transformación o enrutamiento complejo, es sólo un transporte ligero para acceder a los componentes de servicio remotos. Esta topología se encuentra frecuentemente en aplicaciones de negocios más grandes o aplicaciones que requieren un control más sofisticado sobre la capa de transporte entre la interfaz de usuario y los componentes de servicio. Entre los beneficios que aporta frente a las anteriores es que son mecanismos avanzados de colas, mensajería asíncrona, monitoreo, manejo de errores y mejor balanceo de carga y escalabilidad. El único punto de fallo y los problemas de cuello de botella arquitectónico generalmente asociados con un

intermediario centralizado se abordan a través de la agrupación de intermediarios y de la federación de intermediarios (dividir una única instancia de intermediario en múltiples instancias de intermediario para dividir la carga de procesamiento de mensajes en función de las áreas funcionales del sistema).

2.3.4.4. Aplicación monolítica

En una aplicación monolítica o bajo una arquitectura monolítica, toda la lógica se ejecuta en un único servidor de aplicaciones o un único programa que lo hace todo. Las aplicaciones monolíticas típicas son grandes y construidas por múltiples equipos, requiriendo una orquestación cuidadosa del despliegue para cada cambio. También se consideran aplicaciones monolíticas cuando existen múltiples servicios de API que proporcionan la lógica de negocio, toda la capa de presentación es una sola aplicación web grande, es decir la aplicación hace todo(Stephens, 2015). En ambos casos, la arquitectura de microservicios puede proporcionar una alternativa.

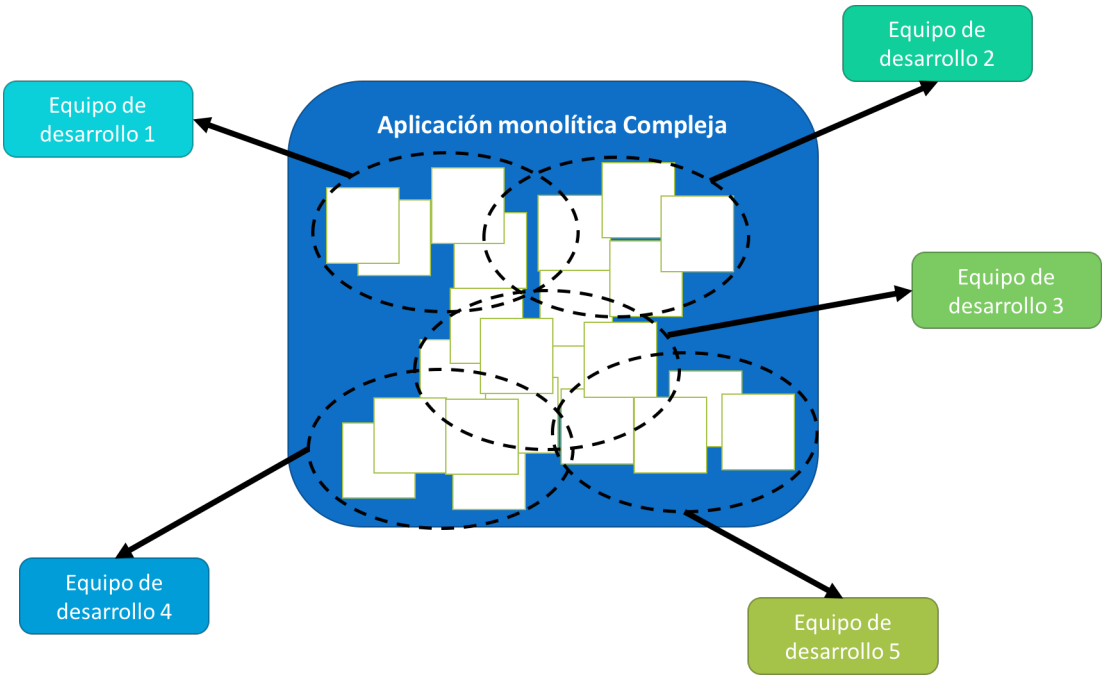


Figura 8: Aplicación monolítica compleja

Fuente: (Atchison, 2016)

Tradicionalmente, las aplicaciones aparecen como monolitos únicos, grandes y distintos. Para la implementación de una funcionalidad de negocio o mejora, un desarrollador debe realizar cambios dentro de la misma aplicación y todos los desarrolladores que realicen cambios deben integrarlos en la misma aplicación única. Los desarrolladores de otros equipos pueden fácilmente afectar el trabajo de los otros, creando conflictos que resultan en problemas. En la figura anterior, se visualiza cinco equipos de desarrollo independientes que trabajan en las áreas que se solapan de la aplicación, por lo que dificulta saber quién está trabajando en qué pieza de la aplicación en cualquier momento en el tiempo lo que afecta la calidad de código y por lo tanto la calidad de la aplicación y la disponibilidad. Adicionalmente, cada vez es más difícil para los equipos de desarrollo individuales hacer cambios sin afectar a otros equipos con cambios incompatibles (Atchison, 2016).

Tabla 3: *Arquitecturas monolíticas vs. microservicios*

Categoría	Arquitectura Monolítica	Arquitectura de microservicios
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Idioma	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera aunque los cuellos de botella estén localizados.	Le permite escalar servicios con cuello de botella sin escalar la aplicación completa.

Fuente: (Shahir Daya et al., 2015).

2.3.4.5. Comparación de arquitecturas monolíticas y de microservicios

2.3.4.6. Servicios web

Los servicios Web son componentes de software ligeramente acoplados entregados a través de tecnologías estándar de Internet. Es decir, los servicios Web son aplicaciones de negocio autodescriptivas y modulares que exponen la lógica empresarial como servicios a través de Internet a través de la interfaz programable y donde el protocolo de Internet (IP) puede ser utilizado para encontrar e invocar esos servicios.

Un servicio web es el elemento clave en la integración de diferentes sistemas de información, ya que los sistemas de información pueden basarse en diferentes plataformas, lenguajes de programación y tecnologías (Wagh y Thool, 2012).

SOAP. La implementación de servicios web por protocolo simple de acceso a objetos

(SOAP) se desarrolló como una alternativa al estándar CORBA (Common Object Request Broker Architecture). Para garantizar el transporte de datos en SOAP, se utilizan protocolos como HTTP, SMTP, entre otros, en formato XML. En este enfoque, un proveedor de servicios publica una descripción del servicio o una interfaz para el registro de servicios, por lo que el solicitante del servicio puede encontrar una instancia de servicio correcta y utilizarla. Algunos problemas de rendimiento en SOAP se producen al formar el mensaje SOAP ya que agrega un encabezado adicional y partes al cuerpo al mensaje. Los servicios Web basados en SOAP incluyen una variedad de estándares, tales como WSDL, WSBPEL, WS-Security, WS-Addressing Estas normas fueron desarrolladas por organizaciones de normalización, como W3C y OASIS (Tihomirovs y Grabis, 2016).

REST. Un servicio REST se define como una agregación de diferentes recursos que pueden ser alcanzados desde un identificador universal de recurso (URI) base. Un recurso representa a una entidad del mundo real cuyo estado está expuesto y puede cambiarse accediendo a un URI. Una Representación es la descripción de los mensajes enviados o recibidos de un Recurso en términos de un lenguaje tecnológico. Actualmente XML y JSON son los idiomas más populares para describir estos mensajes (Valverde y Pastor, 2009).

2.3.4.7. Integración continua

La integración continua (CI) es una práctica en el desarrollo de software en la que los desarrolladores hacen integraciones automáticas de un proyecto lo más a menudo posible (generalmente a diario) con el propósito de detectar fallos lo antes posible. Esta practica de integración comprende la compilación y ejecución de pruebas de todo un proyecto.

Prácticas de integración continua. Para la aplicación de prácticas de CI se requiere

que cada vez que se agregue una nueva parte al sistema, también se creen casos de prueba automáticos para cubrir todo el sistema incluyendo las partes recién agregadas. De igual forma, se requiere que el software sea probado y construido automáticamente y una retroalimentación inmediata sobre los nuevos códigos integrados hacia los desarrolladores. De acuerdo con Hamdan y Alramouni (2015), estas practicas son:

- Integrar el código con frecuencia. Este núcleo de la integración continua. No se debe esperar más de un día para enviar código al repositorio de código compartido.
- No integrar código “roto”. Código roto es un código que contiene cualquier tipo de error cuando se incluye en una construcción de CI.
- Corregir las compilaciones no funcionales de inmediato. Una compilación no funcionales puede ser un error en la compilación, en la base de datos o en la implementación. Es cualquier cosa que impide que la compilación de informes de éxito.
- Escribir pruebas automatizadas. Las pruebas deben ser automatizadas para que funcionen en un sistema de CI. También debe cubrir todo el código fuente.
- Todas las pruebas e inspecciones deben aprobarse. Para que una compilación se apruebe, el 100 % de las pruebas automatizadas deben pasar con éxito. Este es el criterio más importante de CI en cuanto a la calidad del software.
- Ejecutar compilaciones privadas. Las herramientas de CI permiten a los desarrolladores tener una copia del software del repositorio de código compartido localmente en sus estaciones de trabajo. Esto les proporciona la capacidad de tener

una compilación de integración reciente localmente antes de integrarla al servidor de generación de integración principal para asegurar que no falle.

2.3.4.8. Contenerización de aplicaciones

Un contenedor de aplicaciones se basa en la virtualización de sistemas operativos usando los contenedores Linux (Kratzke, 2015), precisamente es un motor de contenedor de código abierto, que automatiza el empacado, entrega y despliegue de cualquier aplicación de software, se presenta como contenedores livianos, portátiles y autosuficientes, que se ejecutarán prácticamente en cualquier lugar. Un contenedor es un cubo de software que comprende todo lo necesario para ejecutar el software de forma independiente. Puede haber varios contenedores en una sola máquina y los contenedores están completamente aislados entre sí y también de la máquina anfitriona (Raj, Chelladhurai, y Singh, 2015).

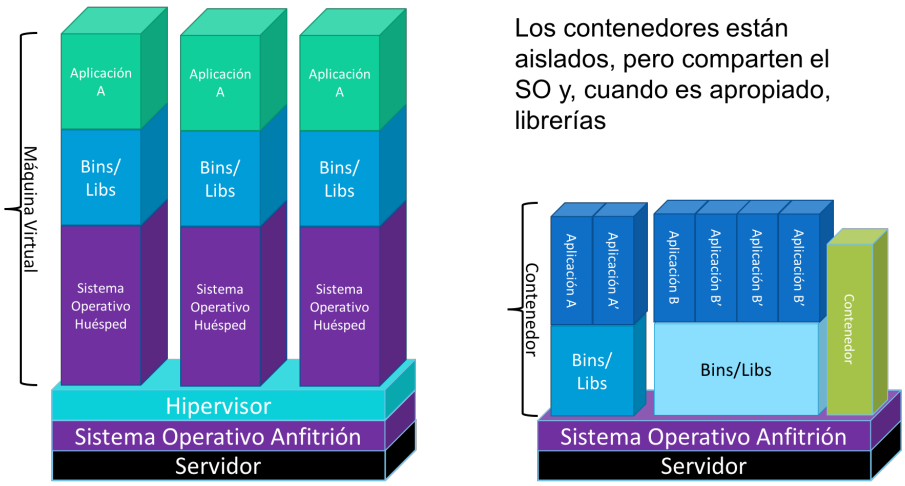


Figura 9: Comparación de utilización de recursos de máquinas virtuales y contenedores

Fuente: (Viktor Farcic, 2016)

La palabra contenedor se define como "un objeto para contener/resguardar o el transporte de algo". La idea detrás de los contenedores de "software" es similar. Son imágenes aisladas e inmutables que proporcionan funcionalidad diseñada en la mayoría de los casos para ser accesibles sólo a través de sus APIs. Es una solución para hacer que el software funcione de forma fiable y en (casi) cualquier entorno. No importa dónde se estén ejecutando (computador portátil, servidor de pruebas o de producción, centro de datos, etc.), el resultado siempre debe ser el mismo (Viktor Farcic, 2016).

Iniciar un contenedor toma alrededor de un segundo, mientras que las máquinas virtuales toman entre decenas de segundos a varios minutos, debido a que inicializan un sistema operativo completo. Los contenedores permiten a las aplicaciones tener autonomía al empaquetar junto on ellas las librerías y otros binarios de los que dependen, evitando conflictos entre aplicaciones que de otra manera dependen de componentes principales del sistema operativo anfitrión. Los contenedores no contienen un kernel de sistema operativo, haciéndolos más rápidos y ágiles que las máquinas virtuales. Un contenedor tiene la capacidad de ejecutar cada aplicación con múltiples conjuntos aislados de procesos bajo una única instancia del kernel, este aislamiento posibilita guardar el estado completo de un contenedor y reiniciarlo más tarde en el mismo o en un diferente sistema anfitrión, de una forma totalmente transparente (Rusek, Dwornicki, y Orłowski, 2017).

2.3.4.9. Despliegue continuo

El despliegue continuo hace referencia al proceso por el cual se implementa un aplicación en producción tan rápidamente como se ha realizado su registro, empaquetado, prueba y validación. Generalmente las organizaciones utilizan herramientas de integración

continua tales como Jenkins, Hudson, Bamboo entre otros, para automatizar el despliegue de sus aplicaciones. Estas herramientas en conjunto con adecuados casos de prueba que permitan una correcta validación a través de procesos de pruebas automatizadas, pueden permitir el despliegue de aplicaciones de manera segura (llevando a cabo procesos de pruebas y validación para producción). En la actualidad las herramientas mencionadas permiten automatizar las implementaciones y despliegues de manera mucho más fácil (Murugesan, 2017).

La mayoría de las empresas que despliegan aplicaciones en la nube, también deben ser capaces de innovar lo más rápido posible debido a que la competencia puede perjudicar su negocio, si no se innovan con mejores experiencias de producto y nuevas características. Esta es una de las razones por la que la entrega continua a tomado mayor protagonismo durante los últimos años, principalmente en empresas emergentes, grandes empresas de Internet y proveedores de Software como un Servicio (SaaS). La entrega continua permite a las empresas cambiar y actualizar continuamente sus aplicaciones en producción utilizando ciclos y metodologías de desarrollo ágil (Villamizar et al., 2015).

Los sistemas basados en microservicio requieren una infraestructura más sofisticada como DevOps, por lo general requieren la construcción de una cadena de implementación de servicios. El uso de tecnologías de nube (cloud) y contenedores permite la construcción de tales cadenas, que impulsan la adopción de estos procesos hiperactivos y ligeros (O'Connor, Elger, y Clarke, 2016).

2.3.4.10. Descubrimiento de servicio

Según Aroraa, Kale, y Kanwar (2017), para poder determinar la dirección de un microservicio simplemente se debe conocer la dirección IP y el puerto donde este se ejecuta. Sin embargo, se debe considerar que, cuando existen muchos microservicios que están configurados dinámicamente en tiempo de ejecución, la ubicación de un servicio es difícil de determinar.

Para resolver problema es necesario tener en cuneta 2 aspectos:

Registro del servicio: es el proceso de registrar el servicio dentro de un registro central donde se almacenan todos los metadatos a nivel de servicio, lista de hosts, puertos, entre otros.

Descubrimiento de servicios: El descubrimiento es el establecimiento de comunicación en tiempo de ejecución con una dependencia a través de un componente de registro centralizado.

Para considerar una solución de registro y descubrimiento de servicios se debe tener las siguientes características:

- Alta disponibilidad del registro centralizado de servicios.
- Un microservicio específico en ejecución, debe permitir la recepción automáticamente de peticiones.
- Capacidades de balanceo de carga inteligente y dinámica.
- Capacidad de monitoreo del estado del servicio y la carga a la que está sometido.
- Capacidad de desviar el tráfico a otros nodos en caso de presentarse nodos defectuosos

sin ningún impacto en sus consumidores (transparente).

- Si un servicio cambia de ubicación o se realizan cambios en los metadatos, la solución de descubrimiento del servicio poseer la capacidad de aplicar los cambios sin afectar las instancias de tráfico o servicio existentes.

Ejemplos de este tipo de soluciones son Zookeeper, Consul, Etcd, entre otros.

2.3.4.11. Balanceo de carga

El balanceo de carga se hace necesario para maximizar la velocidad y capacidad de utilización de atención de las solicitudes, asegurando que ningún servidor esté sobrecargado de solicitudes. Adicionalmente, tienen la capacidad de redireccionar las solicitudes a los servidores restantes si uno de ellos queda fuera de linea o servicio. En la arquitectura de microservicios, un microservicio puede responder a solicitudes internas o externas, por lo que, se puede tener dos tipos de balanceo de carga: del lado del cliente y del servidor.

Balanceo de carga del lado del cliente. Para comunicarse entre si los microservicios necesitan una comunicación entre procesos. Un balanceador de carga del lado del cliente desempeña este papel crítico y puede manejar tanto protocolos HTTP como TCP. La desventaja del balanceo de carga del lado del cliente es que agrega complejidad a la aplicación y es a menudo específica del idioma de programación utilizado sin embargo agrega mayor sofisticación al balanceo. En la mayoría de los casos, es preferible utilizar el balanceo de carga de servicios integrados en la tecnología como lo es Kubernetes (Posta, 2016; Sharma, 2016).

Balanceo de carga del lado del servidor. Esta relacionado con tecnologías que integran componentes de balanceo de carga sobre contenerización, muchos de ellos implementan

algoritmos tales como Round Robin, entre otros, para garantizar la distribución de las peticiones que llegan hacia los microservicios.

2.3.4.12. Monitoreo

La supervisión de microservicios se vuelve más difícil conforme crece su número y se hace necesario conocer su funcionamiento para determinar si alguna parte de la plataforma no funciona como se espera mediante mecanismos de notificación. El monitoreo es un aspecto importante de los microservicios por lo que se debe realizar el monitoreo de las métricas correctas. Conforme se emplea tiempo en la implementación de microservicios, también se ejecutan actividades tales como despliegues, integración continua, monitoreo y registro, por lo que es importante saber dónde guardar dichas métricas. Para resolver este problema se usan bases de datos de serie de tiempo como Graphite, Prometheus, Riak, combinadas con herramientas para visualizar estas métricas en forma de gráficos como Kibana. El monitoreo de microservicios debe ser un mecanismo fuerte no solo a nivel de infraestructura sino a nivel de servicios y de extremo a extremo a fin de detectar de forma temprana fallos, mismos que pueden producirse en cascada (Nadareishvili et al., 2016; Rajesh, 2016; Wolff, 2016).

En el contexto de microservicios, cada uno puede tener su propia instalación de monitoreo independiente del equipo de operaciones, esto permite obtener información independiente del desempeño por cada servicio al equipo que lo esta desarrollando. Los desarrolladores puede adoptar modelos de desempeño paramétrico adecuados para estimar el desempeño del sistema de extremo a extremo o facilitar análisis de estado del servicios, el cual ayuda al equipo a renovar o evolucionar la arquitectura para eliminar problemas

de rendimiento o detectar anomalías utilizando un modelo estadístico utilizando datos de monitoreo en situaciones normales y calculando los valores para cada nuevo punto de datos de monitoreo entrante comparándolos con los principales para detectar valores atípicos (Balalaie, Heydarnoori, y Jamshidi, 2016).

2.3.5. Aplicaciones web

El término aplicación web hace referencia a cualquier tipo de aplicación que se visualiza a través de un navegador, utilizando una conexión a Internet para comunicarse con otras aplicaciones, tales como servidores u otros similares. Las aplicaciones Web se distinguen de los sitios web por la interacción avanzada que ofrecen ya que permiten al usuario interactuar y cambiar el contenido, haciendo que este sea dinámico.

La definición de aplicación web también está relacionada con el modelo de negocio de Software como Servicio (del inglés Software as a Service - SaaS), ya que las aplicaciones SaaS a menudo se entregan como aplicaciones in-browser a los clientes. El denominador común de los sitios web dinámicos que usan viewports (espacio útil de la página donde se desarrolla las actividades del usuario) basados en páginas web, como aplicaciones de comercio electrónico, también pueden ser vistos como aplicaciones web. Una diferencia importante entre una aplicación web simple y uno avanzado puede determinarse por el uso de tecnologías de comunicación asíncronas. Estas tecnologías permiten a la aplicación web comunicarse con el servidor de forma asíncrona o "en segundo plano", sin limitarse a la cadena de respuesta de acción de los sitios web típicos. Una implementación de comunicación asíncrona utiliza la funcionalidad Javascript del navegador para comunicarse con la aplicación de servidor a través de métodos HTTP y JavaScript Object Notation

(JSON) o Extensible Markup Language (XML) para pasar los datos.

La interfaz gráfica de usuario también puede diferir de los sitios web típicos, a menudo una aplicación web imita la interfaz de usuario de aplicaciones de escritorio. En este sentido, el enfoque más directo se toma en que la aplicación se ve gráficamente y es reconocible a la tarea que está realizando, por ejemplo el editor de documentos de Google o el reproductor de música web de Spotify imitan a sus respectivas versiones de escritorio (propias o de otro fabricante) (Oksa, 2016).

2.3.5.1. Patrones de diseño

"Un patrón de diseño es una buena práctica o núcleo documentado de una solución que se ha aplicado con éxito en varios entornos para resolver un problema que se produce en un conjunto específico de situaciones."(Kuchana, 2004), en otras palabras, un patrón es una manera comprobada de abordar la solución para un tipo de problema conocido. Un problema dado se puede resolver de muchas maneras. El enfoque que proporciona la mejor solución y que ya ha sido probado por otros constituye un patrón. Los patrones proporcionan soluciones listas para un tipo conocido de problema, básicamente reutilizan los esfuerzos humanos y la inteligencia que ya están invertidos por otros (Baig, 2016).

2.4. Marco legal

La investigación se fundamenta en leyes, reglamentos y normas aplicables a las entidades públicas y, por el hecho de haber aplicado la investigación en la Asamblea Nacional del Ecuador, se ha tomado en cuenta la normativa vigente y aplicable la función legislativa, que se detalla a continuación:

2.4.1. Reglamento Orgánico Funcional de la Asamblea Nacional

Como parte fundamental del accionar de la Coordinación General de Tecnologías de la Información y Comunicación, en relación con los productos y servicio informáticos y específicamente del desarrollo de aplicaciones se puede mencionar:

Artículo 3.- Objetivos estratégicos, numeral 4: *"Implementar un modelo de gestión administrativa sobre la base del talento humano capacitado y un sistema tecnológico automatizado que optimice los recursos"*.

Artículo 30, Gestión de Tecnologías de la Información y Comunicación, en su apartado de Atribuciones y responsabilidades, los literales:

"g) Asesorar en el análisis, desarrollo e implantación de las aplicaciones informáticas".

"h) Desarrollar, administrar y mejorar los sistemas informáticos propios y adquiridos por la institución" (Asamblea Nacional del Ecuador, 2016).

2.4.2. Ley de Propiedad Intelectual

Al ser el medio que el Estado Ecuatoriano reconoce, regula y garantiza la propiedad adquirida de obras protegidas entre las que se encuentran programas de ordenador, podemos citar el Art.1 de esta normativa *"El Estado reconoce, regula y garantiza la propiedad intelectual adquirida de conformidad con la ley, las Decisiones de la Comisión de la Comunidad Andina y los convenios internacionales vigentes en el Ecuador. . . "*

CAPÍTULO 3

MARCO METODOLÓGICO

3.1. Introducción

En este capítulo, se hace referencia al aspecto investigativo y las herramientas utilizadas que validan método científico del proyecto. Haciendo uso la información recabada en el marco referencial se desarrolla la propuesta de solución al problema planteado, describiendo de manera sistemática la implementación de la propuesta bajo la metodología y la tecnologías adecuadas y que son de acorde a los intereses institucionales.

3.2. Materiales y métodos

3.2.1. Investigación bibliográfica

Partiendo de la necesidad de obtener los recursos teóricos necesarios para el desarrollo de la investigación, mismos que son obtenidos de las diferentes fuentes de información, como: artículos de bases de datos indexadas, libros y otros materiales informativos, la presente investigación requiere de un componente bibliográfico que permita recabar principalmente sobre temas tales como: arquitectura de software, microservicios y aplicaciones web.

3.2.2. Investigación de campo

Debido a que la recolección de datos realizada bajo un proceso sistemático y racional se la obtiene de forma directa del objeto de estudio, la cual está representada por el equipo

de desarrollo de software de la CGTIC, esta investigación tiene una modalidad de campo, con la finalidad de conocer de primera mano la problemática plateada y así desarrollar una solución a corto plazo.

3.2.3. Nivel o tipo de investigación

3.2.3.1. Exploratoria

Con base en la revisión de antecedentes y el análisis bibliográfico, se profundizó en el problema de investigación, en donde se evidenció que a nivel nacional este tipo de estilos y patrones arquitecturales y su aplicación práctica en el desarrollo de aplicaciones web son escaso o inexistentes por lo que la investigación se enmarca dentro de la investigación exploratoria.

3.2.4. Técnicas de recolección de datos

Las herramientas de recolección de datos que permitieron el desarrollo de la investigación fueron las siguientes: Grupo focal, la entrevista y la observación.

3.2.5. Población y muestra

El estudio se realizará considerando al personal de la CGTIC de la Asamblea Nacional el cual está distribuido de la siguiente manera:

Tabla 4: *Población directamente beneficiada*

Población	Número	Porcentaje
Coordinación	2	7 %
Infraestructura	5	17 %
Soporte tecnológico	9	30 %
Desarrollo de Software	4	13 %
Seguridad Informática	2	7 %
Proyectos	3	10 %
Audio y Video	5	17 %
Total:	30	100 %

Fuente: Elaboración propia

Se considera el trabajo con todo el universo misma que al ser menor a 100 no se aplica la toma de una muestra.

3.3. Institución y unidad ejecutora

La Asamblea Nacional de la República del Ecuador es el órgano que ejerce el poder legislativo de la República del Ecuador. Es un parlamento unicameral, formada por 137 asambleístas, repartidos en 12 comisiones permanentes.

- **Nombre de la institución:** Asamblea Nacional del Ecuador
- **Tipo:** Gubernamental - Legislativo
- **Dirección:** Av. 6 de Diciembre y Piedrahita
- **Provincia:** Pichincha
- **Cantón:** Quito
- **Ciudad:** Quito

- **Teléfono:** (+593) 2399 1000

3.3.1. Misión

Legislar y fiscalizar con oportunidad, excelencia y transparencia para aportar al desarrollo integral del Estado, en un marco de democracia y participación (Asamblea Nacional del Ecuador, 2016)

3.3.2. Visión

La Asamblea Nacional será una Institución referente en el contexto regional para ejercer sus atribuciones constitucionales de manera participativa con producción legislativa de calidad y fiscalización de excelencia para el Buen Vivir. Asamblea Nacional del Ecuador (2016)

3.3.3. Objetivos estratégicos

1. Cumplir con una agenda legislativa que priorice aspectos que contribuyan al Buen Vivir.

2. Fortalecer la gestión fiscalizadora de la Asamblea Nacional en la administración pública.

3. Intensificar la labor de la participación ciudadana, a nivel nacional, en los procesos de legislación y fiscalización a través del fortalecimiento de la comunicación y la difusión de actividades.

4. Implementar un modelo de gestión administrativa sobre a base del talento humano y un sistema tecnológico automatizado que optimice lo recursos.

3.3.4. Organigrama

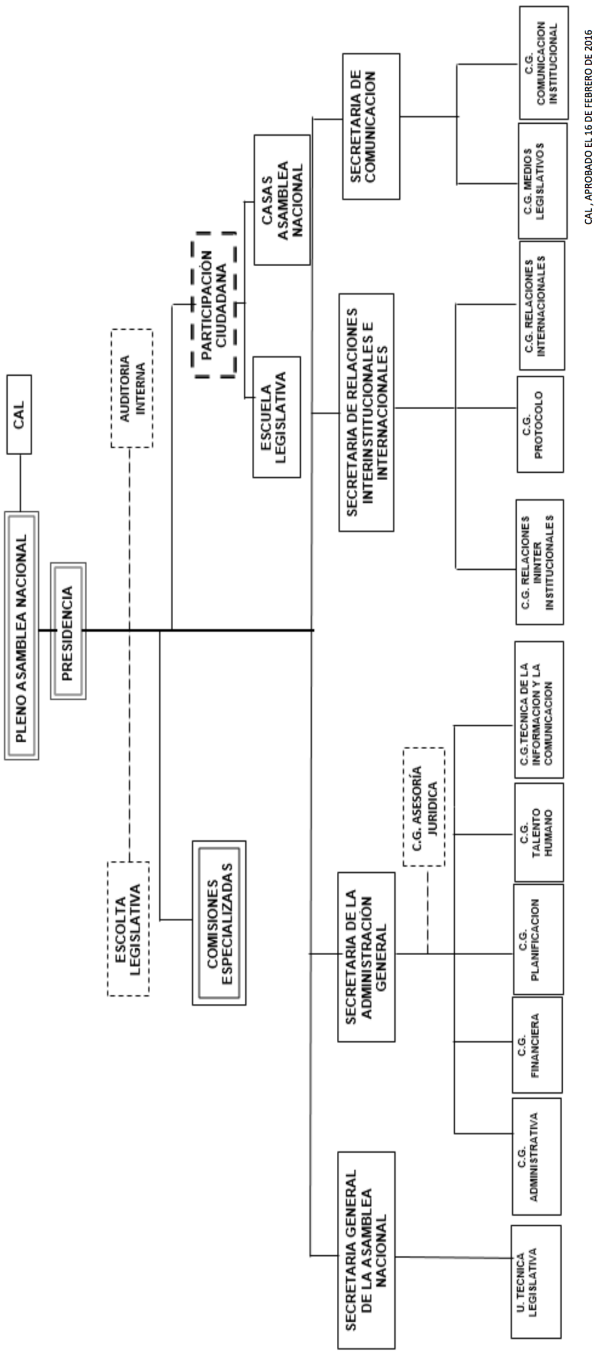


Figura 10: Estructura orgánica funcional de la Asamblea Nacional

Fuente: (Asamblea Nacional del Ecuador, 2016).

3.3.5. Ubicación

La investigación se realizó en la Asamblea Nacional del Ecuador ubicada en la ciudad de Quito, provincia de Pichincha.

Figura 11: Ubicación de la Asamblea Nacional del Ecuador

Fuente: Elaboración propia.

3.3.6. Área o departamento de ejecución

El área de ejecución del proyecto de investigación es la Coordinación General de Tecnologías de la Información y Comunicación de la Asamblea Nacional del Ecuador.

3.3.6.1. Misión

Proveer servicios integrales de tecnologías de la información y comunicación para el procesamiento de datos y acceso a la información, mediante la implantación eficaz de infraestructura tecnológica, reduciendo los riesgos inherentes a los procesos de servicios tecnológicos y aplicando estándares para la gestión y seguridad de TI que coadyuven con el

desarrollo integral de la Asamblea Nacional (Asamblea Nacional del Ecuador, 2016).

3.3.6.2. Atribuciones y responsabilidades

De acuerdo con el artículo 31 de la Codificación del Reglamento Orgánico Funcional(2016):

- Formular y coordinar la ejecución del Plan Informático alineado a la planificación institucional.
- Asesorar y brindar apoyo en tecnologías de la información en todas las instancias de la Asamblea Nacional.
- Administrar, asegurar y mantener operativa la infraestructura de tecnologías de la Información de la Asamblea Nacional.
- Emitir informes técnicos de la adquisición de software y hardware, requeridas por las distintas áreas de la Asamblea Nacional.
- Mantener y garantizar la disponibilidad , confiabilidad e integridad de la información almacenada en los equipos tecnológicos de la Asamblea Nacional.
- Custodiar los originales de los programas y licencias adquiridas.
- Asesorar en el análisis, desarrollo e implantación de las aplicaciones informáticas
- Desarrollar, administrar y mejorar los sistemas informáticos propios y adquiridos por la institución.
- Elaborar y establecer políticas, normativas y estándares para el uso y mantenimiento de tecnologías de la información y comunicación de la Asamblea Nacional

- Elaborar y ejecutar el Plan de Seguridad Informática de la Asamblea Nacional.
- Brindar soporte tecnológico a los requerimientos realizados por los usuarios de los sistemas informáticos, utilizados en la institución y garantizar la operatividad de los mismos.
- Registrar y actualizar el inventario de los equipos de cómputo asignados a los usuarios.
- Brindar servicios tecnológicos especializados en audio, video y videoconferencia en todas las instancias de la Asamblea Nacional.
- Administrar, asegurar y mantener operativa la infraestructura del servicio de audio, video, videoconferencia y voto electrónico de la Asamblea Nacional.
- Supervisar y garantizar el cumplimiento de los niveles de acuerdo de servicios establecidos con los proveedores de tecnología.
- Coordinar la implementación de los protocolos y medidas de seguridad establecidas en la Política y Plan de Seguridad.
- Colaborar con la Mesa de Servicios en la Gestión de Incidentes para el tratamiento y resolución de casos relacionados con la seguridad.
- Instalar y mantener las herramientas de hardware y software necesarias para garantizar la seguridad informática.
- Colaborar con la Gestión de Cambios y Versiones para minimizar el riego de vulnerabilidades en los sistemas en producción o entornos de pruebas.

- Proponer requerimientos de cambio con el fin de aumentar los niveles de seguridad de las aplicaciones informáticas.
- Colaborar con al Gestión de Continuidad del Servicio para asegurar la integridad y confidencialidad de los datos en caso de desastre.
- Establecer las políticas y protocolos de acceso a la información.
- Monitorizar las redes y servicios en red para detectar intrusiones y ataques informáticos.
- Capacitar a los usuarios en la operación y temas relacionados con el uso de herramientas tecnológicas, y
- Cumplir con las demás que establezca la normativa vigente y las disposiciones y delegaciones que le confiera la autoridad competente.

3.3.6.3. Tamaño de área o departamento

3.3.6.4. Organigrama

El Reglamento Orgánico Funcional de la Asamblea Nacional establece una estructura organizacional por procesos en donde ubica a la CGTIC bajo los procesos habilitantes de apoyo con la siguiente estructura interna:

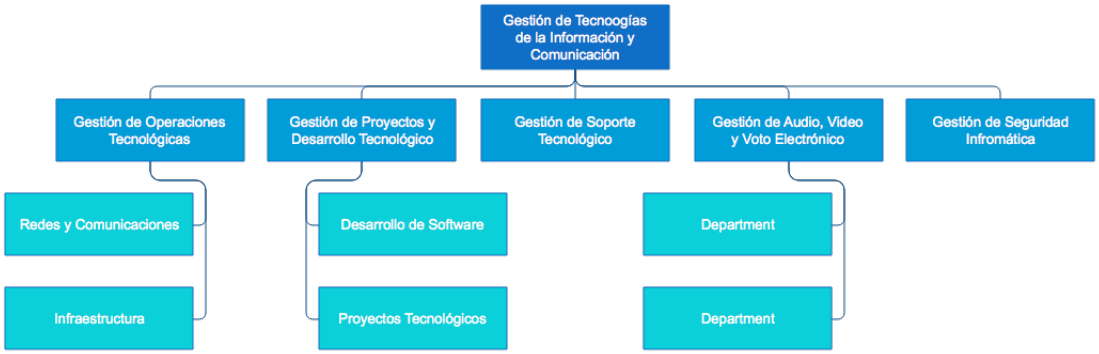


Figura 12: Organigrama CGTIC

Fuente: (Asamblea Nacional del Ecuador, 2016).

3.3.7. La arquitectura

Actualmente el proceso de desarrollo de software, específicamente aplicaciones web se lo realiza bajo la arquitectura establecida por la plataforma Java Virtual Machine (JVM) bajo un esquema monolítico, con una implementación de patrones de arquitectura, entre los cuales están: model view controller (MVC) para web y cliente servidor (Escritorio). Cada una de las aplicaciones construidas lleva una base en el lenguaje de programación java implementando uno o más patrones de diseño antes mencionados.

Como resultado de la aplicación de los lineamientos del lenguaje de programación utilizado, sus patrones y tecnologías (de cada versión del lenguaje o marco de trabajo - framework) las aplicaciones adquirían características de estos y en algunos casos pasaban a convertirse a librerías reutilizadables como servicios web SOAP o como librerías compartidas locales o remotas.

3.3.8. Problema de la arquitectura actual

Los sistemas monolíticos construidos hasta ahora han permitido solventar las diferentes necesidades de automatización de los procesos identificados en las unidades organizacionales requirentes, a pesar de ello, los sistemas se han quedado relegados sea por su atadura a ciertas tecnologías, patrones de diseño o versiones de plataforma con lo que se han pasado de ser una solución a un problema continuo de mantenimiento. Al no contar con una arquitectura definida, las aplicaciones se crean bajo el criterio del equipo de desarrollo de software basados en lineamientos generales y su propia experiencia (empirismo).

3.3.9. Delimitación y alcance de la arquitectura

El diseño de una arquitectura basada en microservicios para el desarrollo de aplicaciones web está limitada a la propuesta del diseño y su validación a través de un prototipo de aplicación bajo dicha arquitectura, mas no en su implementación formal en el Área de Desarrollo de Software de la Coordinación General de Tecnologías de la Información y Comunicación puesto que la decisión de implementación depende de la autoridad de la coordinación y de la planificación estratégica que este establezca. De igual manera, el prototipo se limita a la demostración del diseño bajo tecnologías de implementación propuestas, esto debido a que para cada aspecto de la solución existe en el mercado productos que cumplen en menor o mayor grado dicho requerimiento y no se realiza un análisis comparativo de cada tecnología sino que se limita al uso de las que la comunidad o los propios autores de la bibliografía revisada sugieren como una alternativa de implementación y las que se ajusten al requerimiento del área de ejecución.

La arquitectura diseñada no esta sujeta a un producto de software concreto ya que se

pretende modelar una arquitectura orientada al desarrollo de aplicaciones web, para lo cual se propone su desarrollo en un caso de studio.

Los aspectos avanzados de diseño e implementación de patrones adicionales por cada capa, tales como seguridades o persistencia de datos e integración, así como prácticas previas, durante y post desarrollo de la arquitectura (Ej.: DevOps), no forman parte de esta propuesta ya que se encuentran fuera del alcance del estudio por su extensión y complejidad.

El aspecto evolutivo/incremental de la arquitectura para este estudio, se limita a la definición de una arquitectura base y la incorporación de cambios o ajustes que se hayan definido hasta la primera entrega del diseño arquitectónico, dejando a posterior la incorporación de nuevos requerimientos bajo la responsabilidad a la equipo de Desarrollo de la CGTIC de la Asamblea Nacional.

3.3.10. Involucrados

Beneficiarios

Directos, El área de desarrollo de software.

Indirectos, La Coordinación General de Servicios Tecnológicos, usuarios internos y externos de los sistemas de la Asamblea Nacional del Ecuador.

3.3.11. Equipo técnico responsable

Tabla 5: *Equipo técnico responsable*

Participante	Cargo	Área	Experiencia
Ing. López Hinojosa José Daniel	Especialista en Desarrollo de Software	Gestión de Proyectos y Desarrollo Tecnológico	2 años

Fuente: Elaboración propia.

3.3.12. Factibilidad técnica

Para la realización del presente estudio, existe el conocimiento necesario así como la información y recursos disponibles.

- La CGTIC actualmente ha aprobado su plan operativo en cual se incluye el desarrollo de aplicaciones bajo una nueva tecnología, así como la integración de las aplicaciones actuales por lo que es factible la realización del presente proyecto y existe la predisposición y apoyo en el diseño y propuesta de una nueva arquitectura de software acorde a los nuevos requerimientos y objetivos planteados.

- En la actualidad existe una marcada tendencia hacia el desarrollo ágil, así como la utilización de nuevas arquitecturas con enfoques en la eficiencia, escalabilidad y participación de todo el equipo técnico humano involucrado en el proceso de desarrollo de software por lo que el presente trabajo es de utilidad actual.

- Existen modelos arquitectónicos, patrones de diseño, tecnologías y metodologías que permiten la implementación de una arquitectura de microservicios, además de contar con la información y recursos disponibles.

- La asamblea cuenta con los recursos tecnológicos de infraestructura y personal técnico capacitado que permite el desarrollo y propuesta de la arquitectura planteada.

3.3.13. Factibilidad operativa

La Asamblea Nacional del Ecuador a través de la Coordinación General de Tecnologías de la Información y Comunicación brindan las facilidades operativas que son requeridas

para el desarrollo del proyecto y conseguir los objetivos planteados. La CGTIC firmemente promueve proyectos que coadyuven en la implementación de soluciones que permitan solventar las necesidades tecnológicas de la Asamblea Nacional. Cabe mencionar que en el análisis preliminar (anteproyecto) se estableció la necesidad de una solución de arquitectura de software, ya que el Área de Desarrollo no cuenta con una o usa una aproximación a la proporcionada por la plataforma con la que desarrolla sus sistemas.

3.4. Desarrollo de la propuesta

En esta sección se presenta el diseño de arquitectura de acuerdo con el modelo de proceso de arquitectura de software planteado en el marco de referencia así como aspecto técnicos y tecnológicos para su desarrollo y validación.

3.4.1. Arquitectura y proceso de desarrollo

Para el diseño, inicialmente se presenta el conocimiento básico sobre el proceso de desarrollo iterativo/incremental que se centra en un desarrollo paso a paso de los sistemas de software que se ejecuta a través de ciclos múltiples. En este contexto, el término "proceso de desarrollo incremental iterativo" se ha generalizado. Todo el proceso de desarrollo se divide en pasos individuales de desarrollo secuencial que se construyen unos sobre otros, lo que se conoce como iteraciones. Lo especial en este proceso es que todas las disciplinas y actividades típicas de un desarrollo de software tienen lugar dentro de cada iteración, lo que significa que en cada iteración, se analiza parte de la tarea general, se crea un diseño para lo que se ha analizado y se implementa el diseño. Por lo tanto, al final de una iteración, se ha avanzado un “paso” en la tarea general y se ha añadido otra pieza a la solución. De

hecho, una de estas piezas que componen la solución se llama "incremento", y representa un sistema con algunas partes que ya están funcionando. De esta manera, se llega al sistema final, paso a paso y pieza por pieza, de forma iterativa e incremental.

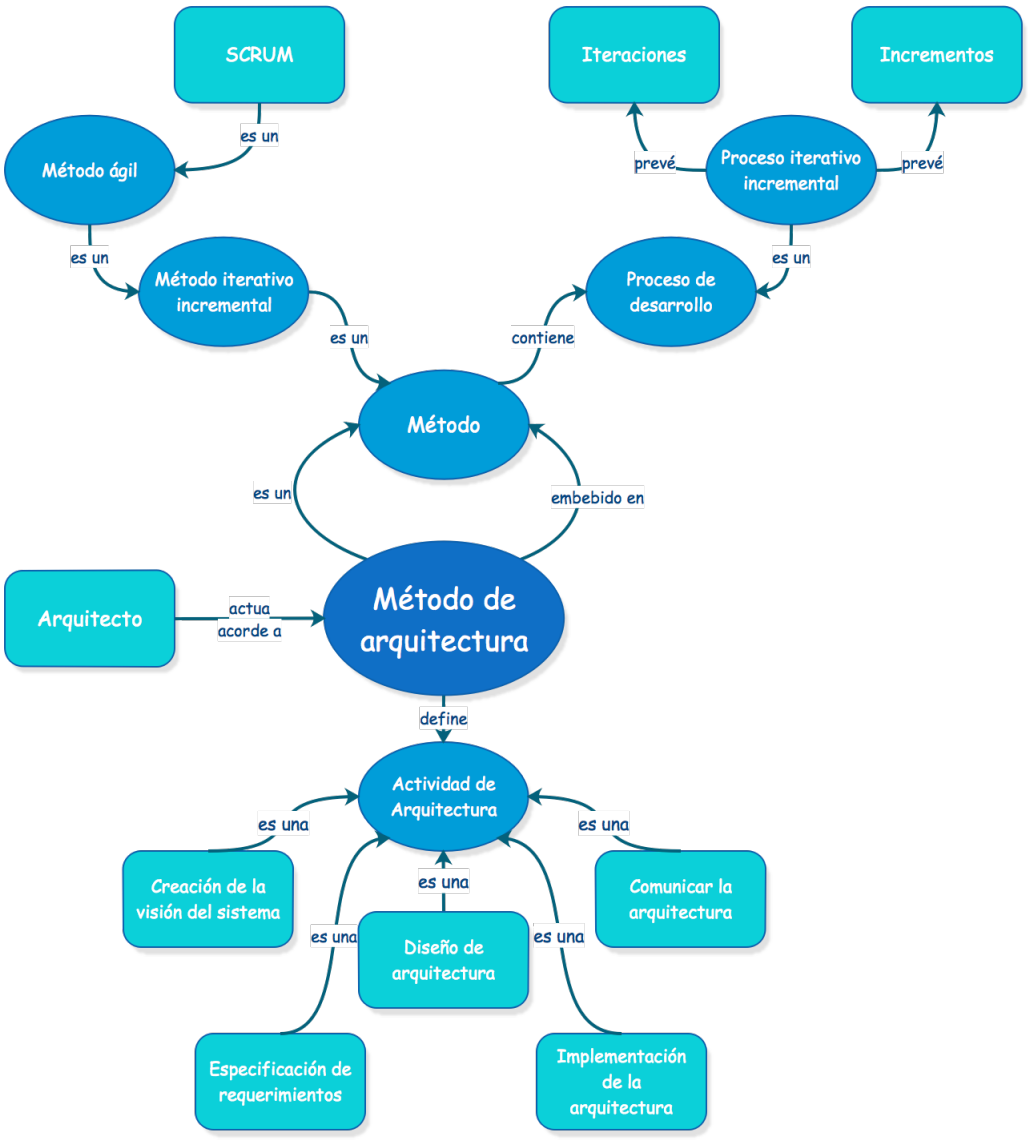


Figura 13: Mapa conceptual del diseño de arquitectura, actividades y el método/proceso desarrollo de software

Fuente: Adaptado de (Vogel, Arnold, Chughtai, y Kehrer, 2011).

Las actividades de arquitectura se realizan continuamente en un proceso de desarrollo incremental iterativo. Por cada iteración, se combinan las actividades de arquitectura en una proporción individual que cambia de iteración a iteración. Inicialmente, todo se centra en las actividades "creación de la visión del sistema" y "especificación de requerimientos", luego, durante el desarrollo, el enfoque pasa a las actividades de "diseño de arquitectura", "implementación de la arquitectura" y "comunicar la arquitectura ". Por lo tanto, las actividades de arquitectura se encuentran inmersas en la estructura del proceso de desarrollo incremental iterativo, permitiendo adaptar el trabajo en las actividades de arquitectura individuales con los requisitos cambiantes, a la vez que se adapta el proceso de desarrollo.

3.4.2. Arquitectura genérica de sistemas basados en microservicios

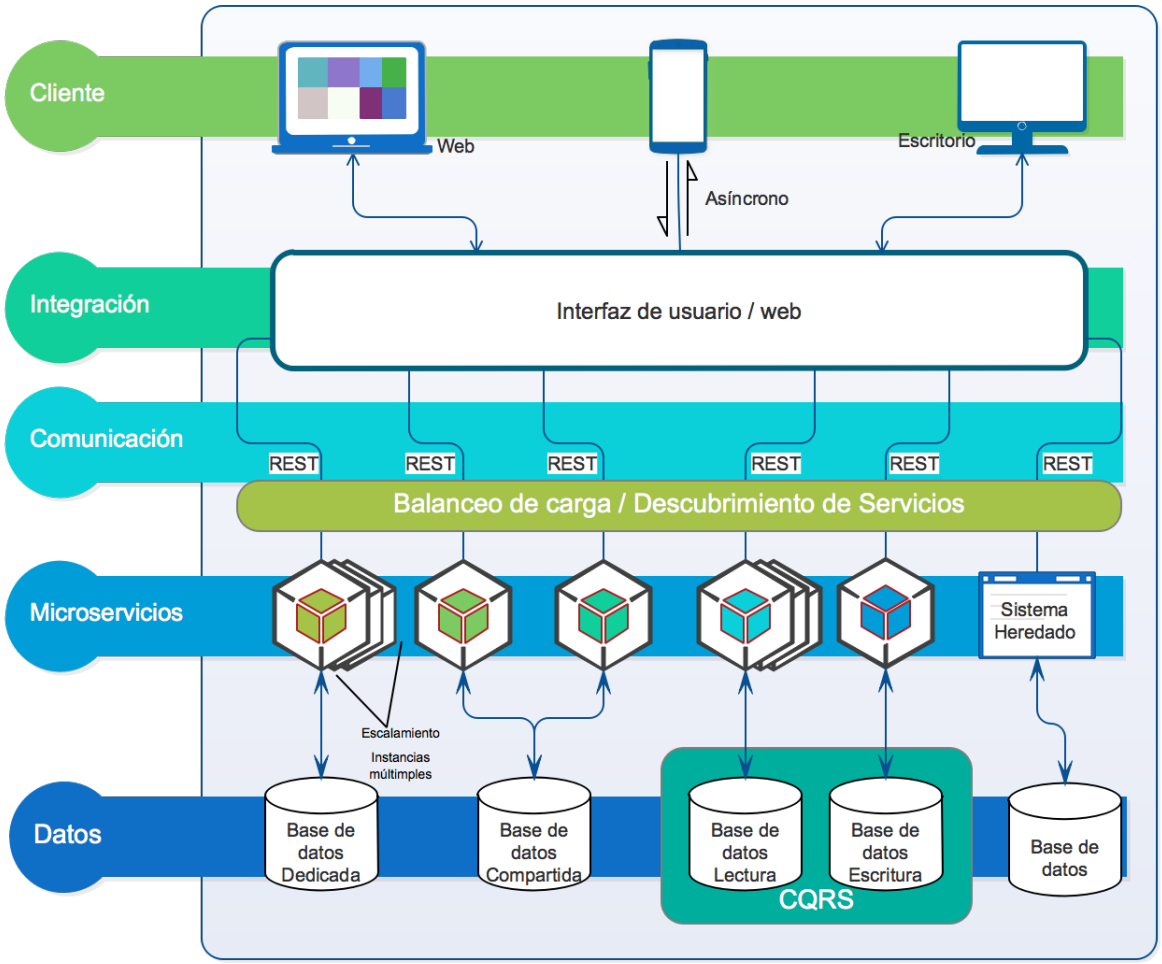


Figura 14: Arquitectura genérica de aplicaciones web bajo microservicios

Fuente: Elaboración propia.

La descripción de tecnologías que se presenta a continuación, sigue los lineamientos tecnológicos generales con los cuales trabaja el área de ejecución.

3.4.3. Descripción de la arquitectura genérica

3.4.3.1. Base de datos

Los motores de base de datos a implementar, son PostgreSQL 9.x ó MySQL 5.x en sus versiones más recientes estables bajo los siguientes patrones:

- Dedicada, en la cual se aloja los datos exclusivos de cada microservicio.
- Compartida, almacena y sirve los datos para uno o más microservicios.
- Segregación de responsabilidad de consulta de comandos (del inglés Command query responsibility segregation - CQRS), este patrón permite separar las consultas de lectura de las de escritura a fin de proporcionar mayor desempeño en consulta de datos exhaustiva.
- Las bases de datos de sistemas heredados no se alteran, sin embargo, pueden ser reemplazadas en cuanto se creen nuevos microservicios para dichos sistemas y datos.

3.4.3.2. Microservicios

Las tecnologías para su implementación son diversas pero cabe mencionar que, uno de los beneficios de los microservicios es la libertad en las tecnologías y lenguaje para su desarrollo e implementación por lo que se deja abierta la decisión de las tecnologías a usar de acuerdo al análisis que realice el área de desarrollo para cada caso y se sugiere el empleo de las tecnologías y lenguajes de mayor uso por parte del Área de Desarrollo de la CGTIC, entre las cuales:

- Plataforma Empresarial JAVA (JEE 1.8.x), como plataforma y lenguaje base de los desarrollos de microservicios.
- Node.js Como alternativa base de ejecución de microservicios.

3.4.3.3. Comunicación

La comunicación que prevalece entre los diferentes componentes es a través de servicios web REST, y para el caso de aplicaciones heredadas las que cada sistema defina como:

- Servicios web basados en SOAP,
- Invocación de métodos remotos (RMI)
- Conectividad abierta a base de datos ODBC/JDBC, entre otros.

3.4.3.4. Integración

Los microservicios pueden ser integrados a través de la delegación de la secuenciación de los microservicios existentes a un tercer microservicio que tendrá la función de llamar al resto y coordinar una respuesta al usuario.

En aspectos mas avanzados se plantea el uso de tecnologías de mensajería asíncrono utilizando un intermediario de mensajes centralizado ligero (por ejemplo, ActiveMQ, HornetQ, entre otros). Este intermediario no ejecuta las siguientes acciones:

- Ninguna orquestación,
- Transformación de mensajes
- Enrutamiento complejo, es sólo un transporte ligero para acceder a los componentes de servicio remotos

3.4.3.5. Cliente

Los clientes pueden ser de cualquier tipo, debido a que la capa de presentación es construida para integrar los microservicios creados bajo un esquema web de diseño adaptativo, sin embargo, los servicios expuestos pueden ser consumidos por cualquier cliente a desarrollar como:

- Escritorio (Cliente Rico)
- Móvil nativo o híbrido
- Otros servicios o sistemas

3.4.3.6. Seguridad

Se debe considerar que, para los aspectos de seguridad de las aplicaciones construidas siguiendo los lineamientos de esta arquitectura se plantea el aseguramiento del canal de comunicación mediante protocolo HTTP seguro (HTTPS).

3.4.4. Arquitectura de un microservicio

Utilizando el modelo hexagonal se expone la estructura de cada microservicio a construir, bajo la siguiente composición y tomando en cuenta las siguientes restricciones arquitecturales:

- Funcionalidad que cumpla el principio de responsabilidad única.
- Expone su función a través de servicios web REST en formatos XML y/o REST.
- La API de servicios REST se documentan y exponen como HTML.

- Opcionalmente, expone su función a través de una interfaz de usuario (UI) HTML.
- Se conecta a servicios de acceso a datos a través de JDBC.
- Incluye autoregistro en el servicio de registro y descubrimiento para el balanceo de carga.
- Incluye prevención de fallas en cascada (circuit braker).

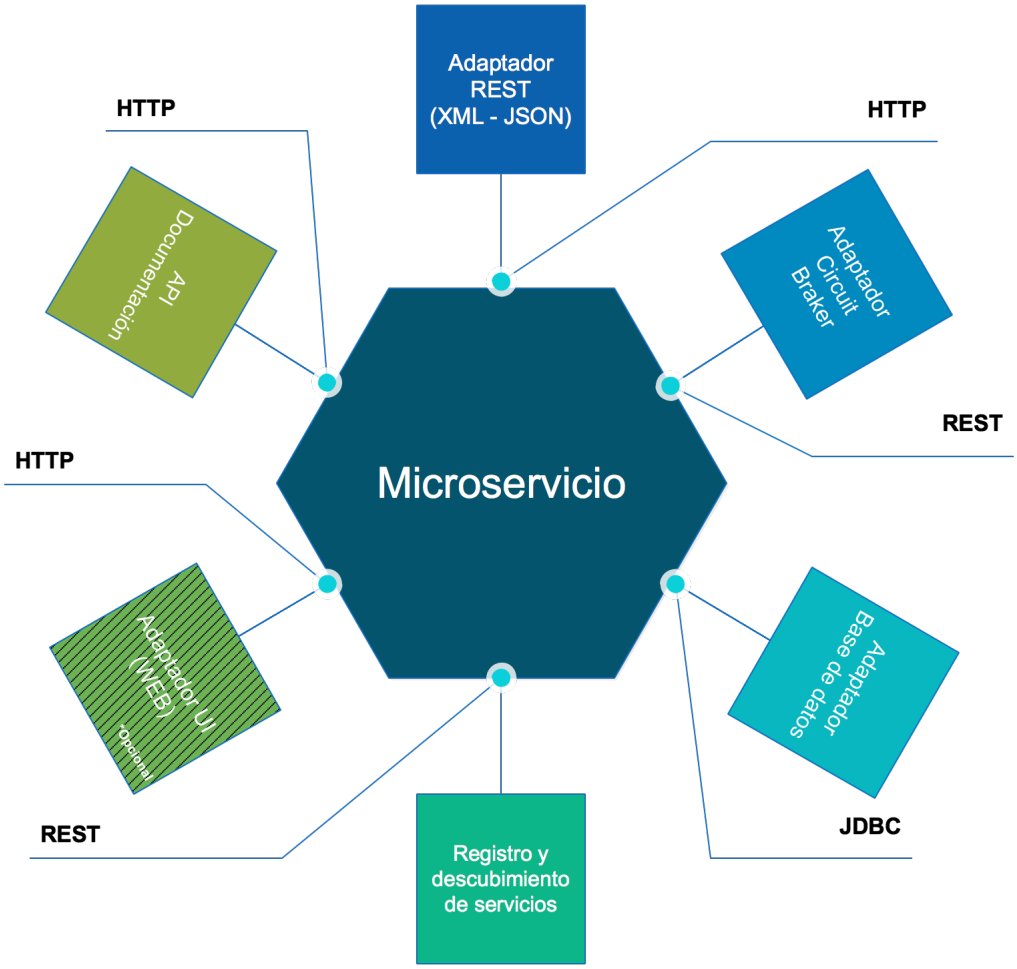


Figura 15: Arquitectura de un microservicio

Fuente: Elaboración propia.

3.4.5. Descripción de la arquitectura de un microservicio

3.4.5.1. Microservicio:

Describe el dominio a ser implementado (funcionalidad). La implementación del microservicio y sus capas internas dependen de la funcionalidad y tecnología (lenguaje) que se desarrollo y emplee respectivamente.

3.4.5.2. Adaptadores y puertos

La descripción de los puertos y adaptadores que componen la arquitectura individual de un microservicio, se presentan en la siguiente tabla:

Tabla 6: *Descripción de puertos y adaptadores de un microservicio (modelo hexagonal)*

Adaptador	Puerto	Descripción de uso
REST (servicios web)	HTTP	Exposición de servicios web en formato JSON o XML
Circuit Breaker	REST	Integración al servidor (microservicio) de control de fallas en cascada
Base de datos	JDBC	Acceso a datos 3.4.3.1
Registro y descubrimiento de servicios	REST	Integración al servidor (microservicio) de registro y descubrimiento de servicios web
UI web	HTTP	Interfaz de usuario web (opcional)
API Documentación	HTTP	Interfaz web de documentación de la API REST del microservicio

Fuente: Elaboración propia.

3.4.6. Validación de la arquitectura

Para la validación de la arquitectura se empleó el análisis basado en escenarios, el cual se basa en los atributos de calidad de tiempo de desarrollo (como mantenimiento y facilidad de uso, entre otros) y por su flexibilidad de aplicación (no lineal) (Babar et al., 2013). El método

utilizado es: análisis de concesiones mutuas de arquitectura (ATAM del inglés Architecture Trade-off Analysis Method) desarrollada por el Instituto de Ingeniería de Software (SEI del inglés Software Engineering Institute).

El método especifica 2 fases:

- Fase 1: Presentación, investigación y análisis
    - Esta fase se desarrolla en las etapas de identificación de necesidades, el marco referencial y la propuesta de la arquitectura desarrollada en el presente trabajo de investigación.
- Fase 2: Pruebas e Informe
    - Se genera el árbol/matriz de utilidad
    - El informe corresponde a los resultados obtenidos de la aplicación en el prototipo desarrollado.

Tabla 7: *Árbol/matriz de utilidad (ATAM)*

Categoría	Refinamiento (atributo)	Escenario
Funcionalidad	Interoperabilidad	Proporcionar funciones de integración entre sistemas nuevos y legados.
Mantenibilidad	N/A	Susceptible de modificación específica y estabilidad sin afectación en otros sistemas o subsistemas. Facilidad de extender
Modificabilidad	N/A	Soporte de cambio y versionado de funcionalidades
Portabilidad	Instalación	Facilidad de instalación en ambientes específicos
	Reemplazo	Capacidad de transferencia entre ambientes con facilidad de instalación, coexistencia y de reemplazo
Fiabilidad	Tolerancia a fallos	Mantener un nivel aceptable de desempeño en caso de falla.
	Recuperabilidad	Recuperación después de un fallo
Reusabilidad	N/A	Componentes reusables.
Disponibilidad	N/A	Operación continua
Seguridad	N/A	Aseguramiento de servicios expuestos y su comunicación

Fuente: Elaboración propia.

3.4.7. Caso de estudio

3.4.7.1. Curul electrónica - eCurul

Del artículo 3, sección 1, capítulo II del Reglamento de Funcionamiento de la Curul Electrónica (2012), El sistema eCurul es una herramienta informática que administra las sesiones del Pleno de la Asamblea Nacional y facilita el cumplimiento de las obligaciones legales de la o el Presidente de la Asamblea Nacional; la o el Secretario General de la Asamblea Nacional; las y los asambleístas principales, suplentes y quienes se principalicen de acuerdo con las disposiciones previstas en la Ley Orgánica de la Función Legislativa y en la Ley Orgánica Electoral y de Organizaciones Políticas de la República del Ecuador,

Código de la Democracia.

3.4.7.2. Arquitectura actual eCurul v1.0

El sistema eCurul está construido bajo un esquema multiplataforma distribuida y de tiempo real, que permite realizar la gestión de votación electrónica para parlamentos y cuerpos colegiados interactuando con cualquier equipo de escritorio (PC) y/o terminal GONSIN(tm) para el registro de asistencias y votaciones de forma electrónica. Sistema Open Source licenciado bajo la licencia GPL versión 3 (Asamblea Nacional del Ecuador, 2017).

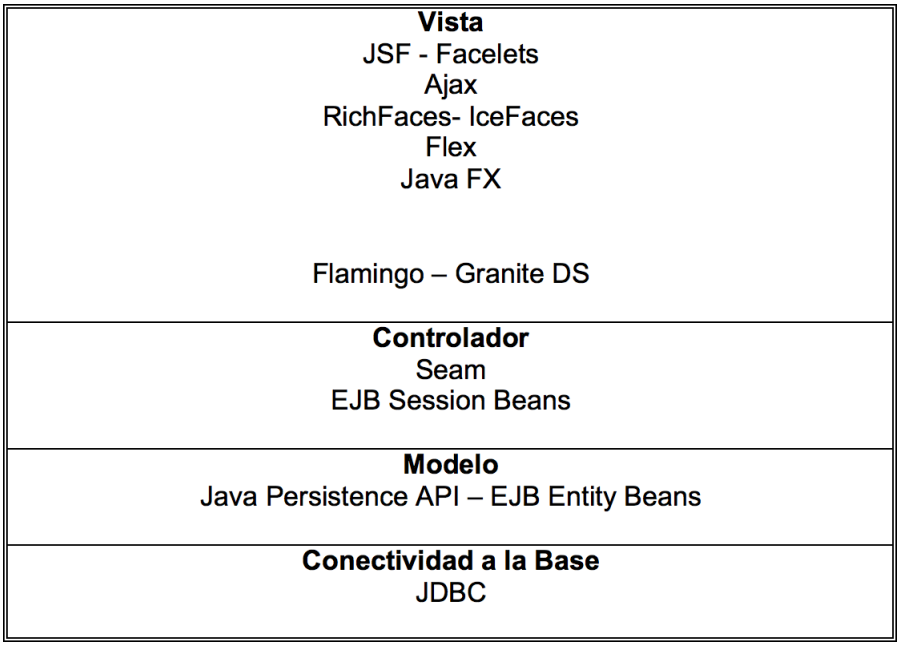


Figura 16: Arquitectura en capas eCurul v1.0

Fuente: (Asamblea Nacional del Ecuador, 2017).

La arquitectura del sistema se basa en el patrón Modelo Vista Controlador (MVC) bajo la plataforma Java EE (v1.5), con componentes visuales enriquecidos (Rich Internet Application - RIA).

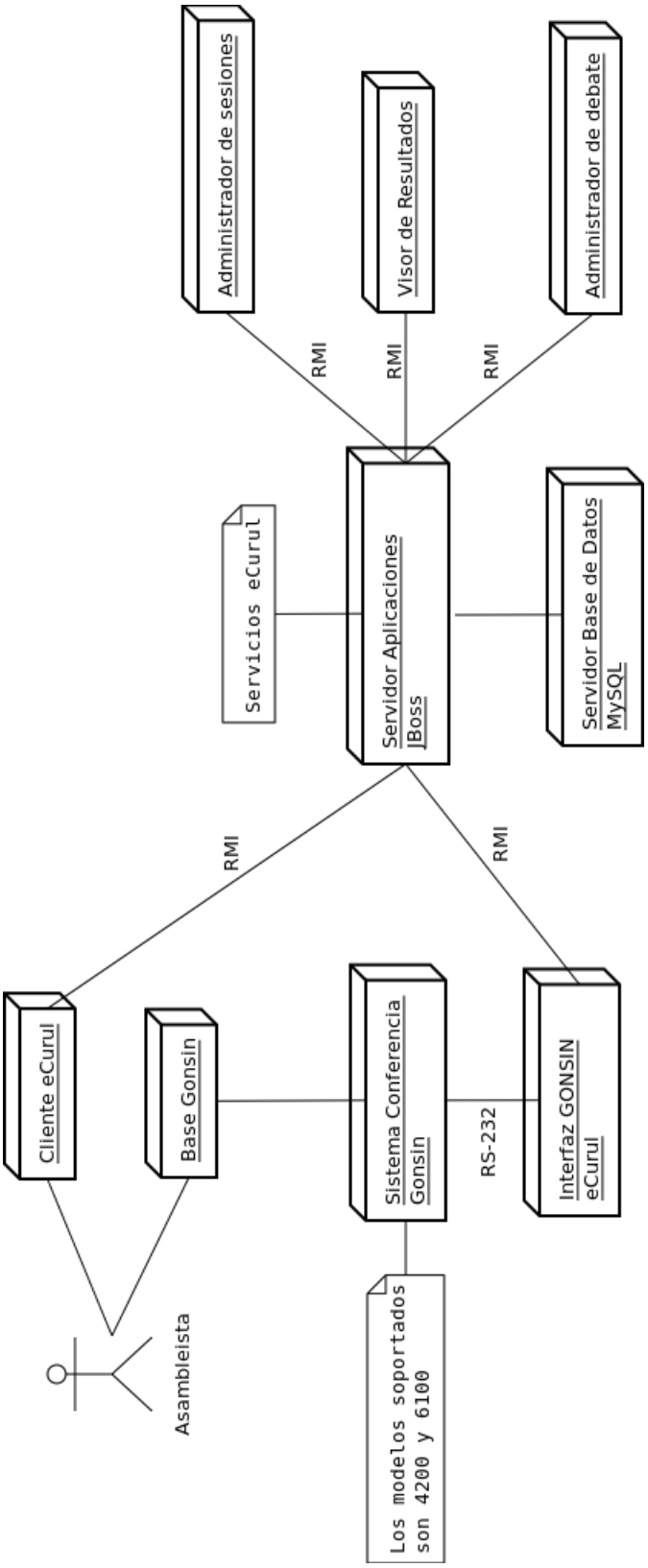


Figura 17: Diagrama de componentes

Fuente: (Asamblea Nacional del Ecuador, 2017)

3.4.8. Obtención de requerimientos

3.4.8.1. Identificación de partes interesadas (Stakeholders)

Tabla 8: *Partes interesadas (Stakeholders)*

Nº	Identificación	Tipo	Descripción
1	Asambleísta	Interno	Usuario del sistema en el proceso de registro de asistencia (quórum) y votaciones para aprobación de leyes
2	Secretario	Interno	Administración de solicitudes de intervención sobre un tema específico a tratar en el pleno de la Asamblea
3	Administrador	Interno	Administrador del proceso de registro de usuarios, asistencia (quórum) y votaciones para aprobación de leyes
4	Operador	Interno	Administra el sistema operacionalmente (iniciar, detener, actualizar recursos)
5	Ciudadano	Externo	Visualiza los resultados de votaciones

Fuente: Elaboración propia.

3.4.8.2. Requerimientos funcionales

De acuerdo con el artículo 5, sección I y II del Reglamento de Funcionamiento de la Curul Electrónica (2012):

- Registro y verificación de quórum (presencia de las y los asambleístas en el Pleno de la Asamblea Nacional);
- Registro de la votación de las y los asambleístas, en las formas previstas en la Ley Orgánica de Función Legislativa;
- Distribución digital de documentación;
- Registro y verificación de la petición de la palabra de cada asambleísta

- Registro y verificación de la petición de punto de información, punto de orden del día y alusión, requerida por cada asambleísta;
- Registro que detalla el tiempo de la intervención de cada asambleísta.
- Autenticación (en esa manera y orden):
  - Lector de huella dactilar;
  - Lector de tarjeta de banda magnética;
  - Usuario y contraseña del correo electrónico institucional; y,
  - Firma electrónica

3.4.8.3. Requerimientos no funcionales

- **Rendimiento:** El rendimiento del sistema no puede ser superior al funcionamiento normal del sistema Gonsin original. El sistema debe asegurar que la respuesta de los procesos de votación y registro de asistencia tenga iguales o menores tiempos de respuesta que el sistema actual.
- **Confiabilidad:** Debe ser muy confiable, dado que, los valores de los votos pueden afectar un resultado completo. Para ello debe disponer de una alta tolerancia a fallos y control de errores adecuado, además de la posibilidad de dejar un registro (log) de cada acción realizada.
- **Escalabilidad:** La naturaleza del sistema es transaccional, aunque la cantidad de usuarios concurrentes en el sistema para la alimentación de la información no será excesiva.

- **Tolerancia a fallas:** El sistema cliente debe garantizar que todas las señales generadas para votos o asistencias sean entregadas para su procesamiento. El riesgo mayor es el control de concurrencia que debe tener la aplicación.

- **Disponibilidad:** El módulo de gestión debe garantizar su disponibilidad, para ello, en ambientes en donde se requiera una disponibilidad total.

3.4.9. Especificación de requerimientos

Los requerimientos funcionales obtenidos de la documentación oficial del sistema actual se especificaron y plantearon tomando en cuenta las limitaciones impuestas de su implementación como una aplicación web.

Tabla 9: *Especificación de requerimientos funcionales*

Código	Requerimiento	Descripción
RF1	Autenticación	Los usuarios deben autenticarse con un usuario y contraseña para tener acceso al sistema
RF2	Quórum	Registro y verificación de quórum (presencia de las y los asambleístas en el Pleno de la Asamblea Nacional)
RF3	Votación	Registro de la votación de los asambleístas, en las formas previstas en la Ley Orgánica de Función Legislativa
RF4	Intervenciones	Registro y verificación de la petición de la palabra de cada asambleísta y su tiempo de intervención
RF5	Alusión	Registro y verificación de la petición de punto de información, punto de orden del día y alusión, solicitada por cada asambleísta
RF6	Reporte	Reporte visual e impreso de resultados de votaciones

Fuente: Elaboración propia.

Los requerimientos no funcionales, de igual manera se especifican por la razón expuesta anteriormente en esta sección y se toman de los expresados en el estándar ISO/IEC 9126-1

(2001):

Tabla 10: *Especificación de requerimientos no funcionales*

Código	**Requerimiento**	**Descripción**
RNF1	Funcionalidad	Proporcionar funciones para satisfacer los requerimientos. Se requiere precisión, interoperabilidad y conformidad
RNF2	Confiabilidad	Debe mantener un nivel adecuado de ejecución. Se requiere tolerancia ante fallos y facilidad de restablecimiento.
RNF3	Usabilidad	Facilidad, simplicidad de uso. Se requiere facilidad de comprensión y operabilidad.
RNF4	Eficiencia	Ejecución sobre recursos determinados. Se requiere utilización de recursos apropiados.
RNF5	Mantenibilidad	Susceptible de modificación. Se requiere mutabilidad en una modificación específica y estabilidad.
RNF6	Portabilidad	Capacidad de transferencia entre ambientes. Se requiere facilidad de instalación, coexistencia y facilidad de reemplazo

Fuente: Elaboración propia.

3.4.10. Arquitectura propuesta

Con base en la documentación recabada en el marco referencial, para la implementación de microservicios es necesario realizar la separación o división del sistema monolítico una vez identificados los requerimientos, sus funcionalidades y módulos, siguiendo la metodología para el diseño de arquitectura.

3.4.10.1. Contexto general del sistema

Contiene los actores (roles). Estos fueron derivados de las partes interesadas (stakeholders) y la relación/dependencia con sistemas periféricos.

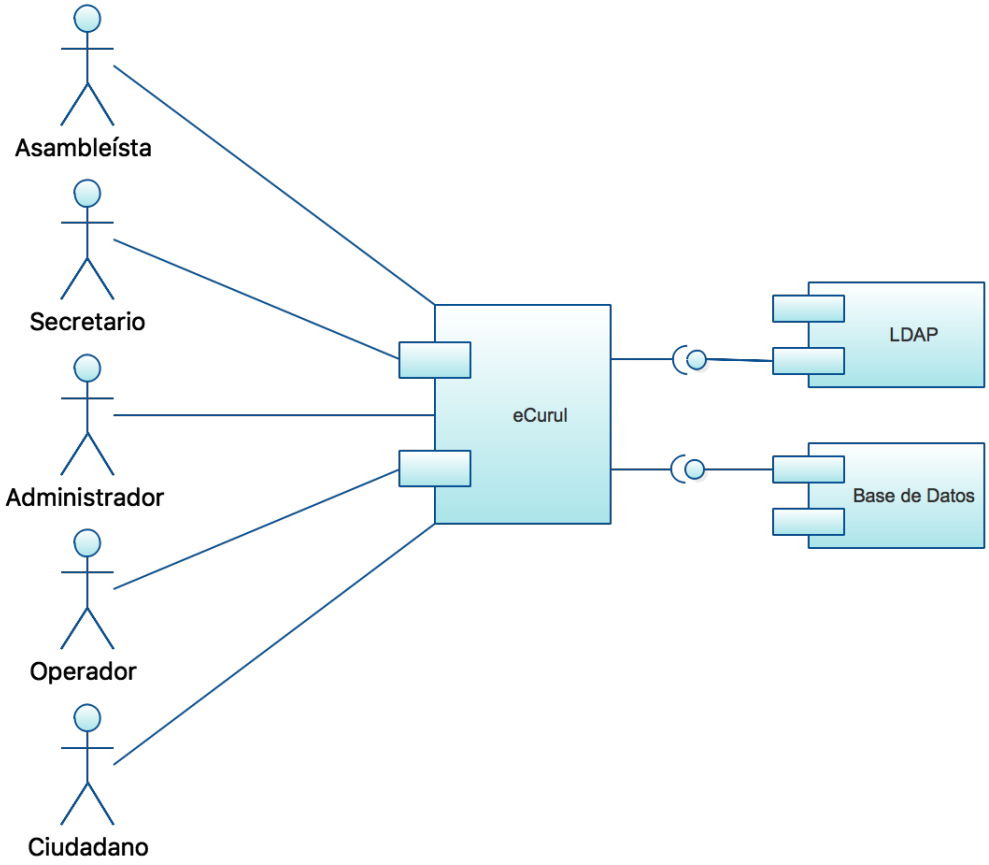


Figura 18: Contexto general del sistema

Fuente: Elaboración propia.

Los actores relacionados con el sistema eCurul interactúan con las funcionalidades acordes a su rol, el sistema depende de un directorio activo (LDAP) para obtener los datos de los usuarios que acceden al sistema. La base de datos almacena los dato de sesiones, ordenes del día (temas a tratar), registros de asistencia, quórum y votaciones de propuestas de ley.

3.4.10.2. Visión general de la arquitectura

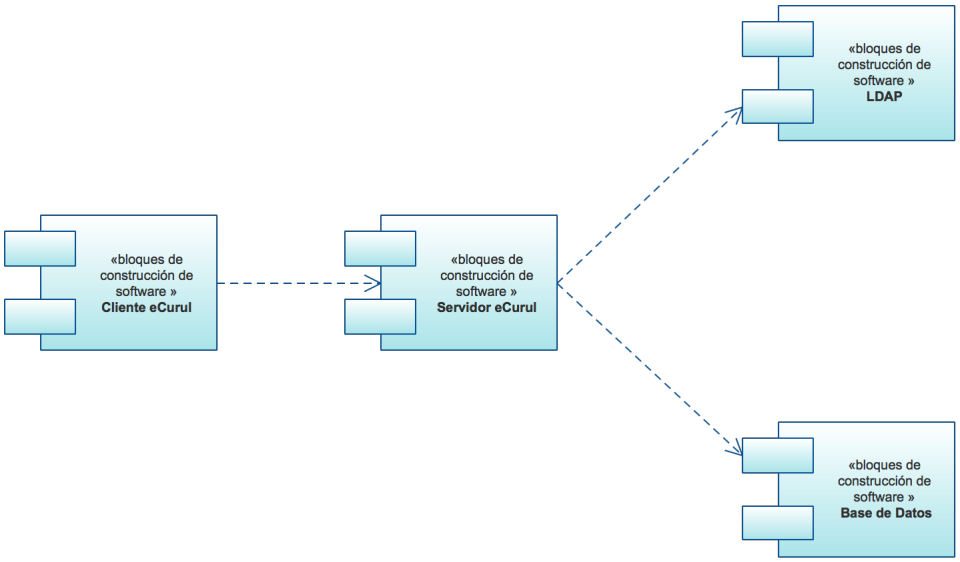


Figura 19: Visión inicial general de la Arquitectura

Fuente: Elaboración propia.

Cliente eCurul: El sistema se construye como una aplicación web de acceso a través de la intranet, siendo este cliente de tipo:

- Navegador de internet
- Aplicación móvil
- Cliente de escritorio (un cliente rico)

3.4.10.3. Priorización de requerimientos

Tabla 11: *Priorización de requerimientos funcionales*

Código	Requerimiento	Beneficio	Riesgo	Prioridad
		(1=alto, 3=bajo)		
RF1	Autenticación	2	1	1,5
RF2	Quórum	1	2	1,5
RF3	Votación	1	1	1
RF4	Intervenciones	2	3	2,5
RF5	Alusión	3	3	3
RF6	Reporte	2	2	2

Fuente: Elaboración propia.

Tabla 12: *Priorización de requerimientos no funcionales*

Código	Requerimiento	Beneficio	Riesgo	Prioridad
		(1=alto, 3=bajo)		
RNF1	Funcionalidad	1	1	1
RNF2	Confiabilidad	1	1	1
RNF3	Usabilidad	2	2	2
RNF4	Eficiencia	2	3	2,5
RNF5	Mantenibilidad	2	3	2,5
RNF6	Portabilidad	3	3	3

Fuente: Elaboración propia.

La priorización es el resultado de la clasificación de los requisitos según el beneficio y el riesgo. Se considera satisfacer los requisitos de un alto beneficio y riesgo tan pronto como sea posible. La prioridad se determina utilizando la siguiente fórmula: Prioridad = Promedio(Beneficio + Riesgo).

3.4.10.4. Vista de casos de uso

A continuación se describen los casos de uso identificados para cada actor del sistema.

Tabla 13: *Casos de uso - actor asambleísta*

Código	Caso de uso	Descripción
CU1	Registrar quórum	El usuario registra su asistencia ante un solicitud de constatación de quórum
CU2	Consignar voto	El usuario consigna su voto ante una petición de votación para aprobación de una moción o ley
CU3	Solicitar intervención	El usuario registra una solicitud de intervención ante un tema activo en debate.
CU4	Solicitar alusión	El usuario registra una solicitud de punto de información, punto de orden del día o alusión

Fuente: Elaboración propia.

Tabla 14: *Caso de uso - actor secretario*

Código	Caso de uso	Descripción
CU5	Habilitar/Deshabilitar temas	El usuario habilita o deshabilita un tema específico para debate y registro de intervenciones de los asambleístas.
CU6	Administrar intervenciones	El usuario administra (concede) las solicitudes de intervención de los asambleístas sobre un tema activo para debate

Fuente: Elaboración propia.

Tabla 15: *Caso de uso - actor administrador*

Código	Caso de uso	Descripción
CU7	Administrar usuarios	El usuario registra a los asambleístas a participar en una sesión plenaria
CU8	Solicitar quórum	El usuario registra una solicitud de quórum
CU9	Solicitar votación	El usuario registra una solicitud de votación para aprobación de leyes

Fuente: Elaboración propia.

Tabla 16: *Caso de uso - actor ciudadano*

Código	Caso de uso	Descripción
CU10	Visualizar resultado	El usuario visualiza los resultados de votación de manera gráfica a través de un medio digital (monitor) o impreso.

Fuente: Elaboración propia.

Tabla 17: *Caso de uso - actor usuario*

Código	Caso de uso	Descripción
CU11	Autenticación	El usuario inicia una sesión en el sistema.

Fuente: Elaboración propia.

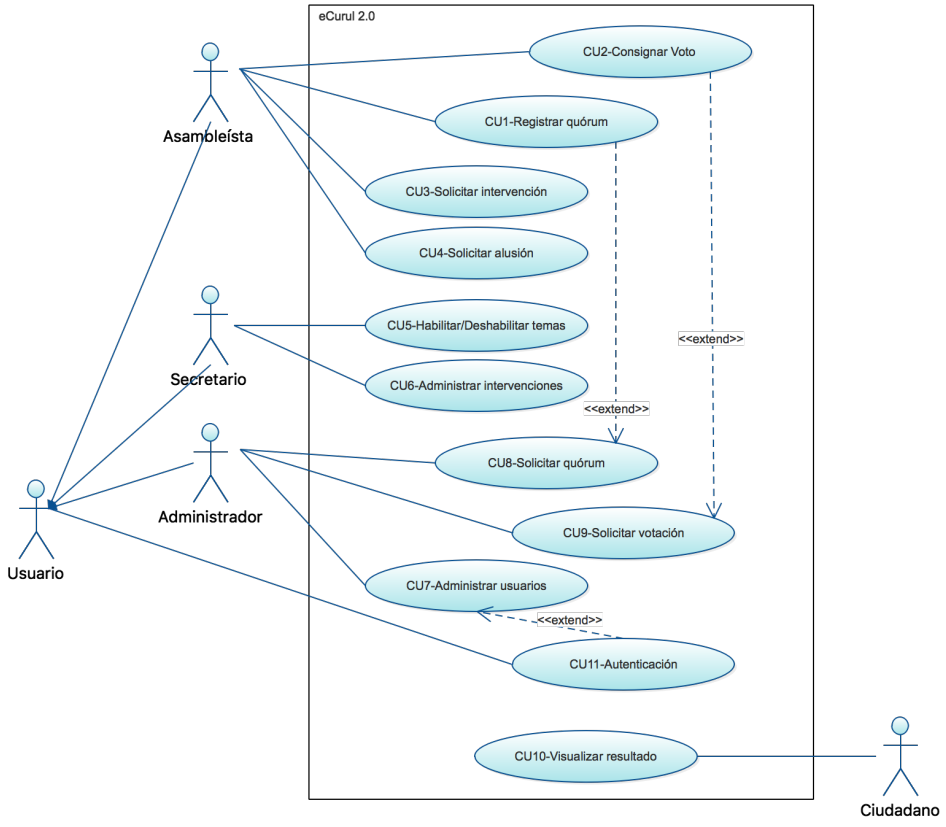


Figura 20: Casos de uso

Fuente: Elaboración propia.

3.4.10.5. Restricciones

Para la construcción del sistema se consideraron las siguientes restricciones:

Tabla 18: *Restricciones*

Código	Descripción
R1	Las cuentas de usuario son manejadas por un servidor de directorio existente mismo que es usado por diferentes aplicaciones en la Asamblea Nacional
R2	La comunicación con la fuente de datos debe ser por JDBC.
R3	El acceso al sistema debe ser a través de un navegador de internet y estará disponible sólo en la intranet de la Asamblea Nacional
R4	Los resultados que se produzcan las actividades de quórum y votación serán visualizados a través de un reporte gráfico en pantalla.

Fuente: Elaboración propia.

3.4.10.6. Consideraciones arquitecturales

En la siguiente tabla se presenta las consideraciones arquitecturales que inicialmente se consideran para una primera iteración de diseño del sistema.

Tabla 19: *Consideraciones arquitecturales*

Código	Descripción
CA1	El sistema debe usar como lenguaje base Java y sus tecnologías, especificaciones y frameworks relacionados a fin de aprovechas los conocimientos y experiencia del equipo de desarrollo en este lenguaje.
CA2	La implementación de microservicios debe ser progresiva, y aplicada sobre las funcionalidades del sistema que lo ameriten.
CA3	Inicialmente se considera el uso de tecnología de contenerización sin replicación o cluster.

Fuente: Elaboración propia.

3.4.10.7. Vista de Módulos

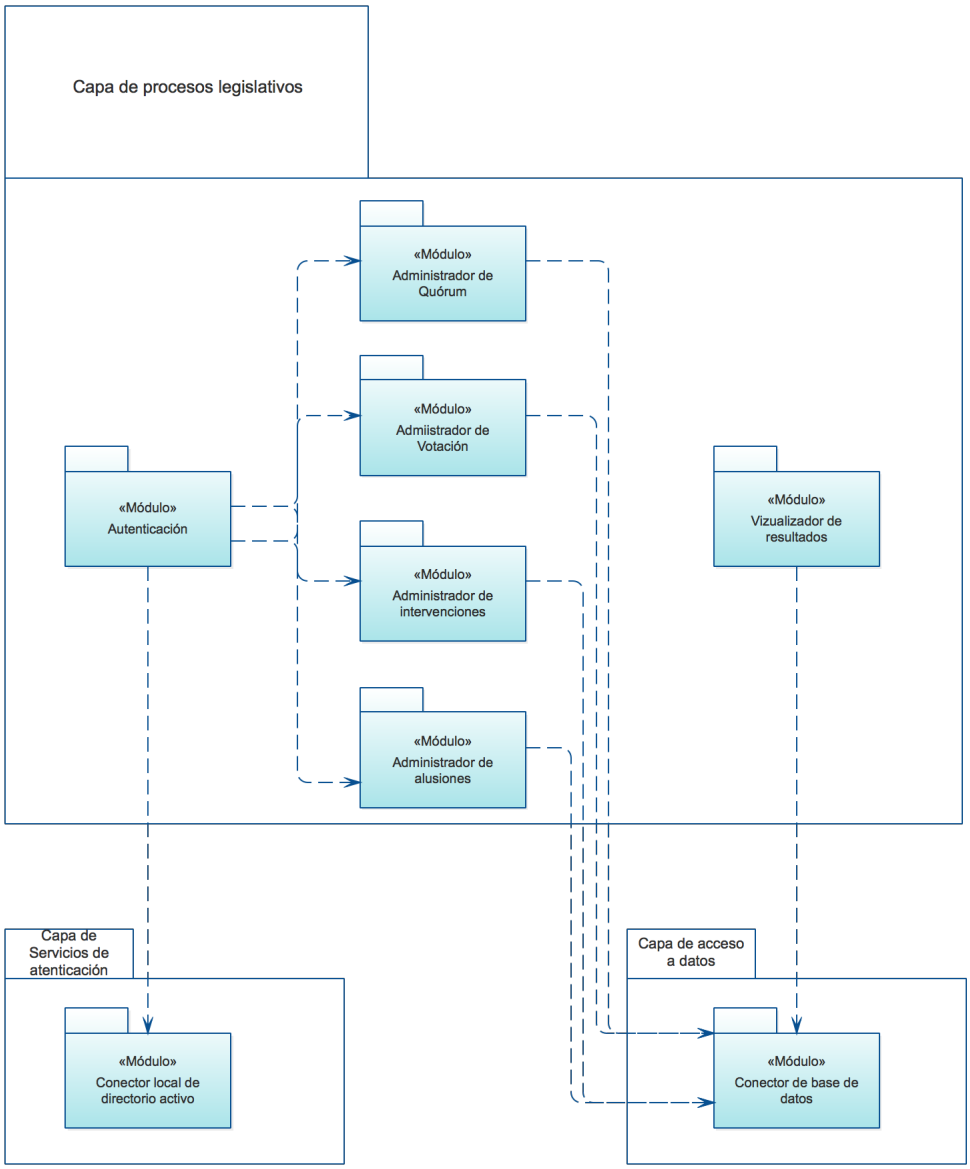


Figura 21: Vista de módulos del sistema eCurul v2.0

Fuente: Elaboración propia.

Las responsabilidades de cada elemento son:

Capa de servicios de autenticación: contiene módulos que proporcionan los datos de usuarios y su validación para acceso al sistema.

Capa de acceso a datos: contiene módulos que almacenan los datos generados por los módulos de la capa de procesos legislativos.

Capa de procesos legislativos: contiene módulos para efectuar procesos legislativos.

Conector local de directorio activo: el módulo se encarga de proporcionar acceso al directorio activo institucional para validar usuarios con acceso al sistema.

Conector de base de datos: el módulo proporciona acceso a la base de datos para el registro/consulta de datos generados por los módulos de procesos legislativo.

Autenticación: módulo de administración de accesos de usuario al sistema. Interactúa con los datos de directorio activo.

Administrador de quórum: módulo de administración de asistencia/quórum de asambleístas. Interactúa con el módulo de autenticación para su validación, registrando los datos del proceso en la base de datos local.

Administrador de votación: módulo de administración de consignación de votos de asambleístas en sobre aprobación de una moción de proyecto de ley y los almacena en la base de datos local.

Administrador de intervenciones: módulo de administración de intervenciones sobre un tema activo para debate por parte de los asambleístas y los almacena en la base de datos local.

Administrador de alusiones: módulo de administración de alusiones sobre un tema activo para debate por parte de los asambleístas y los almacena en la base de datos local.

3.4.10.8. Vista lógica

El propósito del diagrama de clases es mostrar los tipos que están siendo modelados dentro del sistema. que incluyen: una clase, una interfaz, un tipo de dato, un componente.

Figura 22: Diagrama de clases eCurul v2.0

Fuente: Elaboración propia.

Descripción de clases (entidades) del sistema eCurul v2.0.

Tabla 20: *Descripción de clases eCurul v2.0*

Clase	**Nombre descriptivo**	**Descripción**
session	Sesión	Datos relacionados con una las sesiones plenarias de la Asamblea Nacional.
diary	Diario	Datos del orden del día a tratar en una sesión plenaria.
proposal	Propuestas	Datos de las propuestas de ley para sometimiento a votación.
quorumrequest	Solicitud de quórum	Datos de solicitudes de constatación de quórum para tratamiento de propuestas.
quorumregistration	Registro de quórum	Datos relacionado de una solicitud de quórum y su resultado
vote	Votos	Datos de votaciones de los asambleístas sobre una propuesta de ley específica
assemblyman	Asambleísta	Datos de los asambleístas que participan en una sesión plenaria.
role	Rol	Roles de usuarios del sistema (Asambleísta/Autoridad).
interventionrequest	Solicitud de intervención	Datos de solicitudes de intervención, punto de orden del día, punto de información que solicitan los asambleístas.

Fuente: Elaboración propia.

3.4.10.9. Vista de procesos

En esta vista se presenta las actividades que realiza el sistema, el flujo de trabajo y las rutas alternativas.

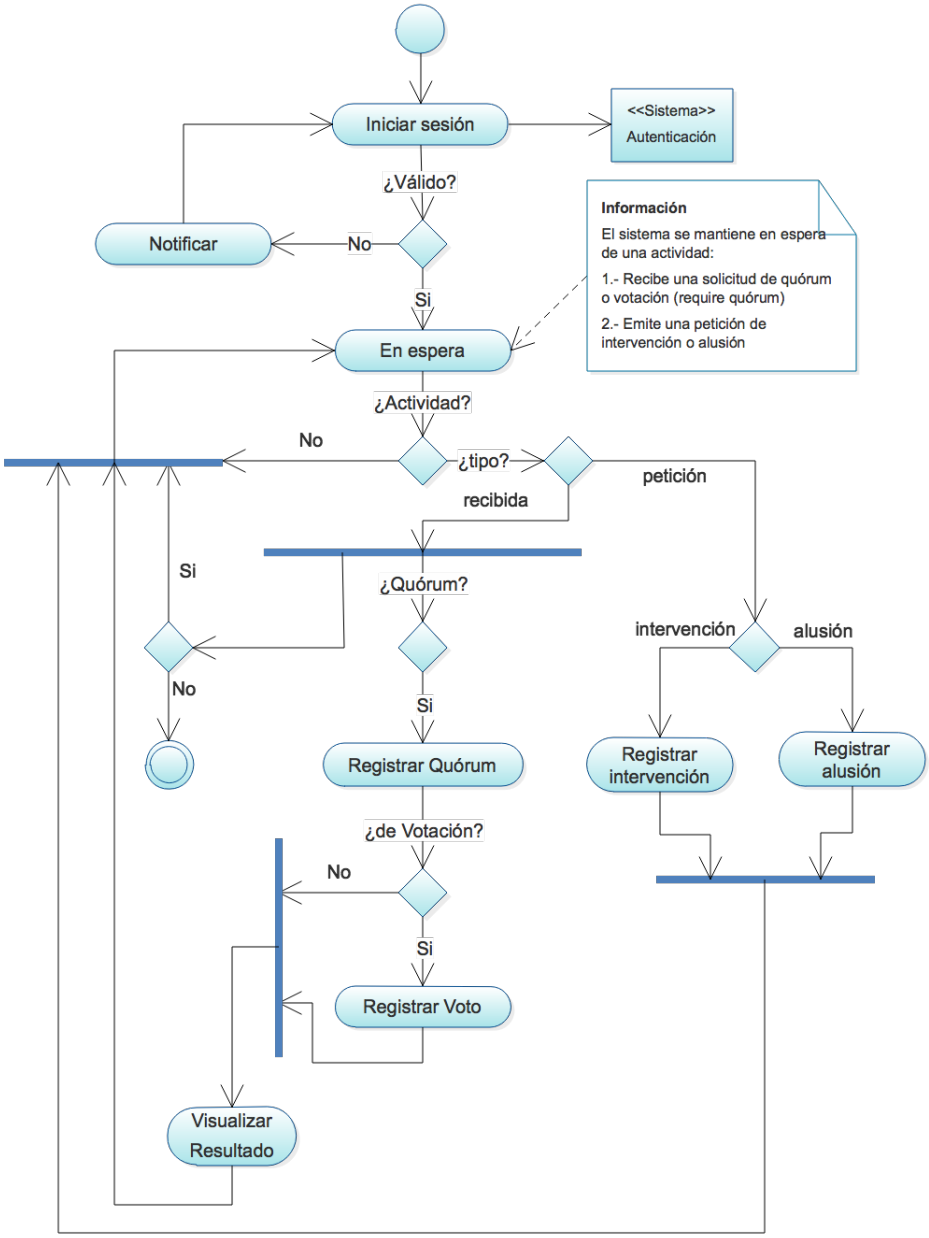


Figura 23: Diagrama de actividades eCurul v2.0

Fuente: Elaboración propia.

3.4.10.10. Vista de desarrollo

En esta vista se presenta los componentes con el propósito principal de mostrar las relaciones estructurales entre los componentes de un sistema.

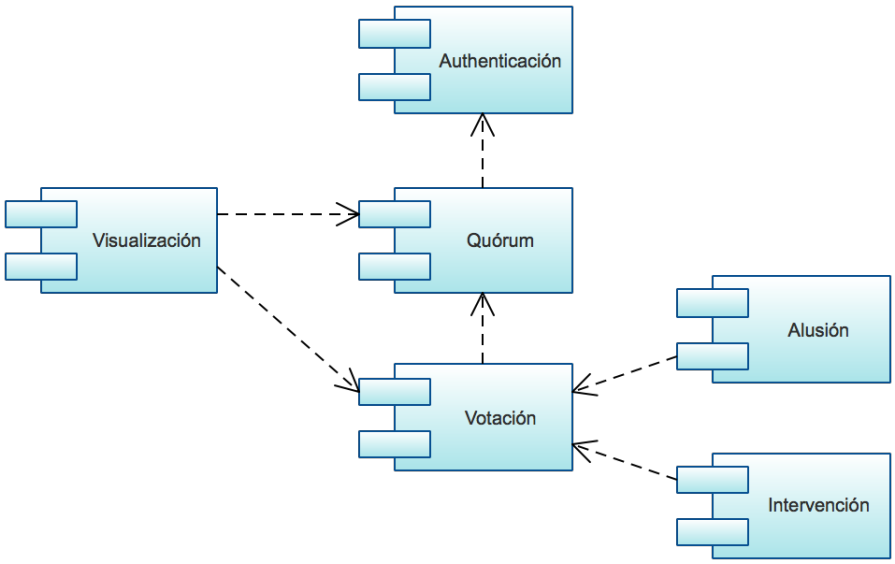


Figura 24: Diagrama de componentes eCurul v2.0

Fuente: Elaboración propia.

3.4.10.11. Vista física

En esta vista se presenta cada componente con el objetivo principal de brindar una guía al momento de desplegar la aplicación dentro de la infraestructura institucional.

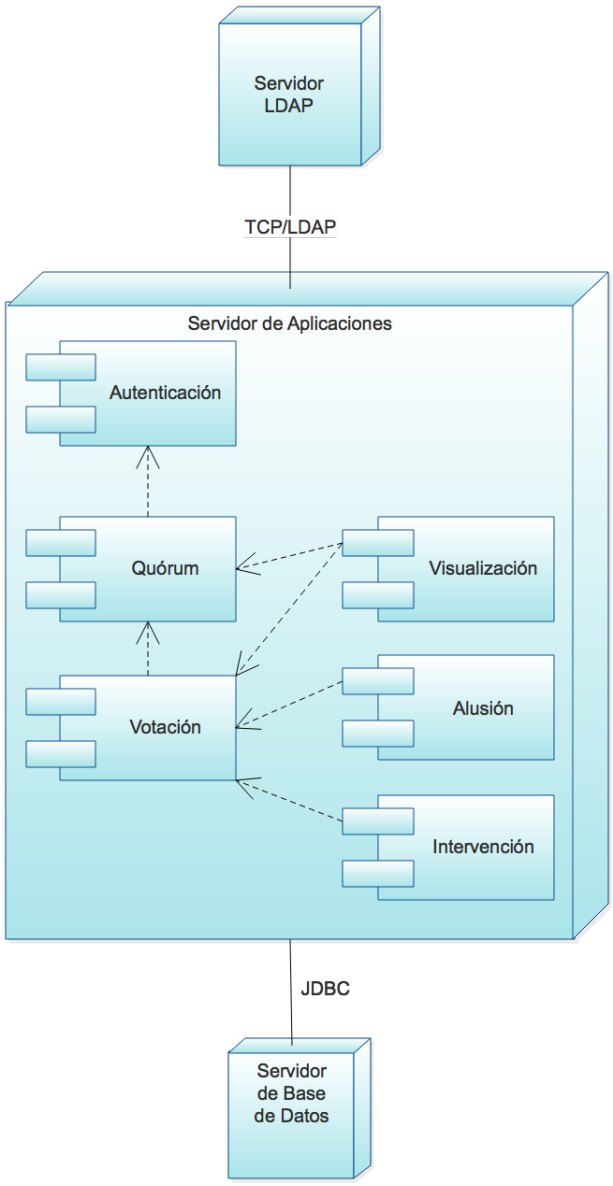


Figura 25: Diagrama de despliegue eCurul v2.0

Fuente: Elaboración propia.

3.4.10.12. Diseño de microservicios

Tabla 21: *Matriz de dependencias*

Módulo	Autenticación	Quórum	Votación	Intervención	Alusión	Visualización
Autenticación	-					
Quórum	X	-				
Votación	X	X	-			
Intervención	X			-		
Alusión	X				-	
Visualización		X	X			-

Nota. Lectura horizontal, "El módulo X, depende del módulo Y".
Fuente: Elaboración propia.

El resultado del análisis de la matriz de dependencias, sugiere que el módulo de autenticación es requerido por la mayoría de módulos, lo cual sugiere considerar balanceo de carga, adicionalmente de la vista de módulos del sistema, se evidencia que el módulo de autenticación requiere de una interfaz de comunicación local hacia el directorio activo. Cada funcionalidad se implementa como un microservicio.

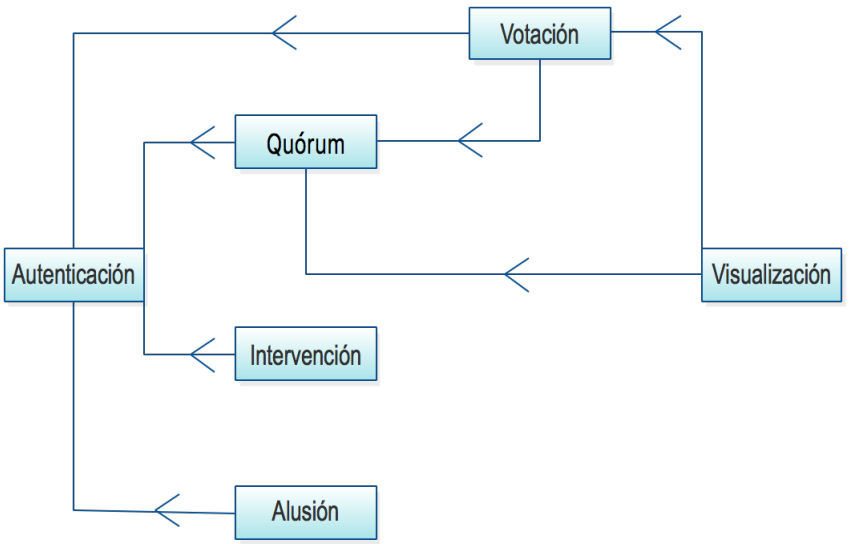


Figura 26: Diagrama de dependencias eCurul v2.0

Fuente: Elaboración propia.

El sistema se expresa como un conjunto de microservicios, cada uno define sus propias actividades y flujo. Para la construcción individual de microservicios se emplea la estructura definida en la sección 3.4.4.

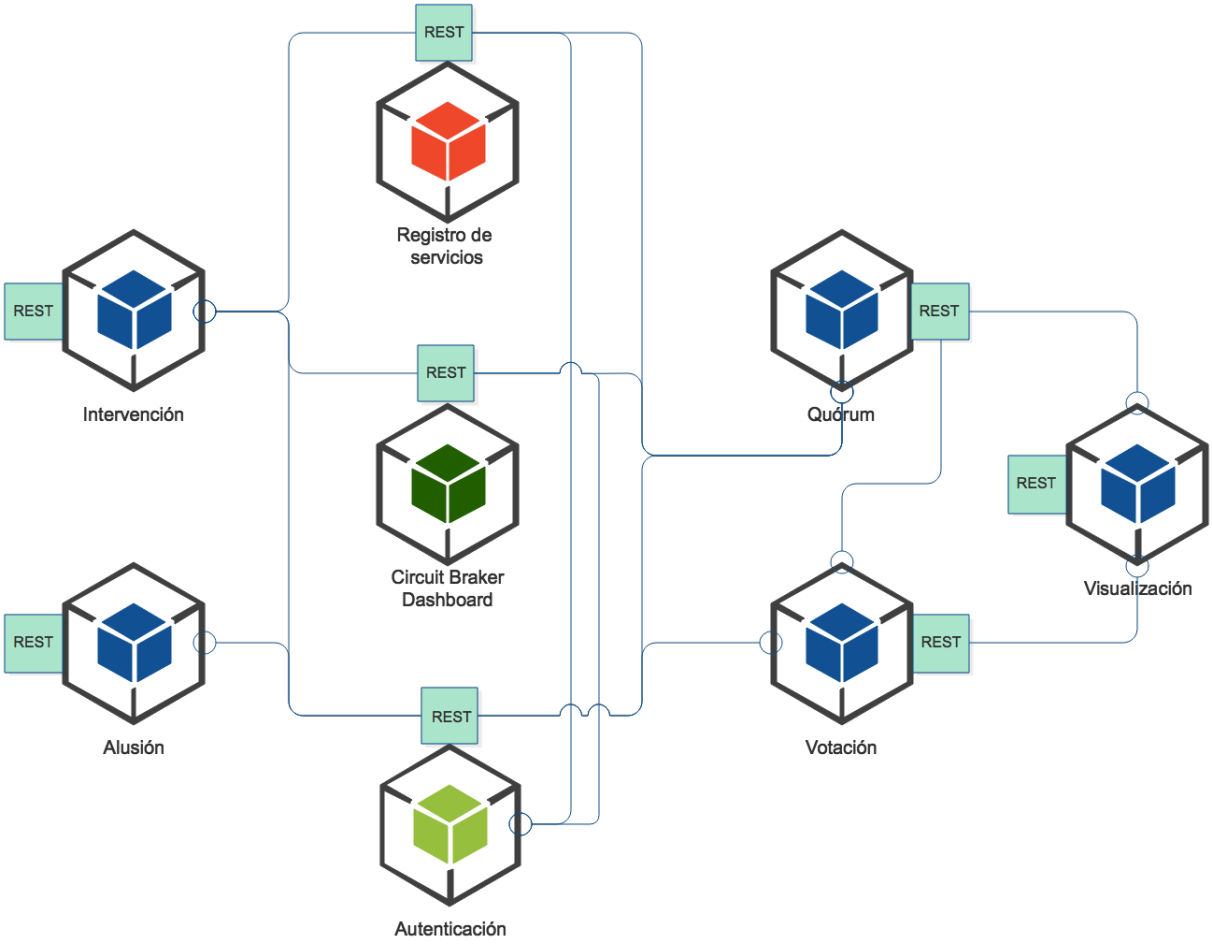


Figura 27: Arquitectura de microservicio - eCurul v2.0

Fuente: Elaboración propia.

3.4.10.13. Tecnologías

Tabla 22: *Tecnologías para implementación*

Tecnología	**Descripción de uso**
JEE7	Base de código de construcción de las soluciones
Jenkins, Gradle, Git, Jenkins, SonarQube	Integración continua
Ansible	Despliegue continuo
Spring Boot	Construcción de microservicios
Spring Cloud	Integración de microservicios
Spring Security	Seguridad en servicios REST
Netflix Hystrix	Implementación de Circuit Breaker
Netflix Eureka	Registro y descubrimiento de servicios
Netflix Ribbon	Balanceo de carga
Swagger	Documentación REST API
Docker	Contenerización de aplicaciones

Nota. Las tecnologías referenciadas se basan en las necesidades identificadas, sin embargo, pueden usarse tecnologías que se consideren necesarias y adecuadas para cada caso específico de microservicio y su funcionalidad.
Fuente: Elaboración propia.

CAPÍTULO 4

RESULTADOS CONCLUSIONES Y RECOMENDACIONES

4.1. Introducción

En este capítulo se presentan los resultados obtenidos del caso de estudio desarrollado aplicando el diseño arquitectural basado en microservicios propuesto. Se evaluó los resultados obtenidos al aplicar el diseño propuesto a nivel de desarrollo de aplicaciones, así como el nivel operacional (infraestructura).

Se adjunta las conclusiones y recomendaciones de acuerdo a los resultados de evaluaciones que evidencian los objetivos alcanzados así como las respectivas de la presente investigación.

4.2. Presentación de resultados

4.2.1. Validación de la arquitectura

Mediante el uso de ATAM, se obtuvieron los siguientes resultados de cumplimiento planteados en la propuesta de arquitectura:

4.2.1.1. Resultados

La arquitectura propuesta basada en microservicios cumple con los criterios de evaluación planteados y seleccionados a través de la aplicación del método ATAM y su verificación mediante el uso de los elementos arquitecturales que la componen, así como de su implementación tecnológica conforme a las tecnologías seleccionadas.

Tabla 23: *Validación del Árbol/matriz de utilidad (ATAM)*

Atributo	Medio de cumplimiento (componente arquitectural)	Tecnología usada	Cumple
Interoperabilidad	Microservicios, Integración (REST)	Spring Boot e Integration	Si
Mantenibilidad	Microservicios	Spring Boot	Si
Modificabilidad	Microservicios, Integración continua, Despliegue continuo	Spring Boot, Gradle, Git, Jenkins, SonarQube, Ansible	Si
Instalación	Contenerización de aplicaciones	Docker	Si
Reemplazo	Replicación de microservicios, contenerización de apicaciones	Docker	Si
Tolerancia a fallos	Patrón circuit breaker	Netflix Hystrix, Ribbon	Si
Recuperabilidad	Replicación de microservicios, patrón circuit braker	Netflix Hystrix	Si
Reusabilidad	Microservicios	Docker	Si
Disponibilidad	Replicación de microservicios	Netflix Eureka	Si
Seguridad	Aseguramiento de servicios y canal de comunicación	Spring Security, SSL	Si

Fuente: Elaboración propia.

Cabe recalcar que, todas los elementos arquitecturales y sus respectivas tecnologías forman parte de una implementación del estilo de arquitectura de microservicios por lo que, hace disponible los beneficios descritos en el correspondiente sección (2.3.4.1) del marco de referencia del presente trabajo de investigación.

En consecuencia, la aplicación del diseño de arquitectura bajo microservicios es adecuada y cumple a cabalidad los requerimientos institucionales y tecnológicos de la Coordinación General de Tecnologías de la Información y Comunicacion de la Asamblea Nacional del Ecuador en comparación con el diseño monolítico utilizado en el desarrollo de aplicaciones web.

4.2.2. Análisis cualitativo

El instrumento de reunión de expertos fue desarrollado tomando en cuenta los antecedentes del entorno, tales como: Tecnología en uso, documentación disponible, experiencia del personal.

La reunión de expertos tuvo dos enfoque principales:

1. Desarrollo de aplicaciones

2. Arquitectura de software

Tabla 24: *Ficha técnica*

Metodología	Se utilizó la técnica de "Focus Group", que consiste en una dinámica de grupo efectuadas con personas con características homogéneas, dirigidas por un moderador el cual realiza preguntas y genera la discusión en torno al tema tratado.
	Las sesiones se desarrollaron sobre la base de una guía de pautas pre-elaboradas conjuntamente con el líder del área de desarrollo.
Universo	Talento humano del área de desarrollo de software (5 personas)
Realización	Sala de sesiones de la Coordinación de Tecnologías de la Información y Comunicación.

Fuente: Elaboración propia.

4.2.2.1. Análisis e interpretación de resultados del instrumento aplicado

Luego de aplicado el instrumentos de recolección de la información, se procedió a realizar su análisis y de acuerdo con los enfoques empleados, se selecciona las tres preguntas más relevantes que se relacionan con el problema de investigación.

Tabla 25: *Preguntas relevantes para el análisis cualitativo*

Variable	**Preguntas seleccionadas**
Desarrollo de aplicaciones web	¿Qué procesos de desarrollo de aplicaciones se usa habitualmente? ¿Cuántas y de qué tipo de aplicaciones se han desarrollado?. ¿Cuáles son planes futuros?. ¿Existen servicios alternativos a los principales? En caso afirmativo: ¿Qué tipo de soluciones?
Arquitectura de software	¿El software producido es modular funcionalidad? En caso afirmativo: ¿Qué tecnologías se usaron?. En caso negativo: ¿Por qué no se lo ha hecho? ¿El software producido separa sus funcionalidades o estas están bien definidas? En caso afirmativo: ¿Cuáles son? En caso negativo: ¿Por qué no se lo ha hecho? ¿Para usted, qué grado de importancia tiene el empleo de una arquitectura de software y por qué?

Fuente: Elaboración propia.

Con respecto al desarrollo de software y con base en la información recolectada de la

aplicación del instrumento de investigación se concluye que:

- El proceso de desarrollo de software utiliza una metodología ágil debido al carácter cambiante de los sistemas en ejecución, estas a su vez, constituyen en su mayoría los orientados a la web, con una clara tendencia al desarrollo en entornos móviles. Existe un uso mayoritario de aplicaciones a nivel interno con alta criticidad mismas que no cuentan con servicios alternativos en presencia de fallos o eventos fortuitos que permitan la continuidad del servicio.

- El producto de software que se genera carece de principios de modularidad debido al uso de la arquitectura implícita de la plataforma de desarrollo utilizada. Como resultado las aplicaciones adquieren un carácter monolítico embebiendo en una sola unidad ejecutable todas sus funcionalidades lo que incide tanto en su desarrollo como mantenimiento.

- De forma unánime se concuerda que es de vital importancia la aplicación de una arquitectura de software porque permite aumentar la calidad del producto debido a la estandarización de construcción de aplicaciones.

En consecuencia y bajo el criterio unificado obtenido, se concluye que es de alta importancia emplear una arquitectura de software para el desarrollo de software que permita una estandarización de las aplicaciones y brinde una mejor calidad a los productos de software.

4.3. Conclusiones

Con la realización del trabajo de investigación, se desarrolló y cumplió con cada objetivo planteado a través de los cuales se ha dado una solución al problema identificado.

Debido a la naturaleza monolítica (arquitectura) de los proyectos desarrollados en la CGTIC bajo su metodología personalizada (adaptada del marco de trabajo SCRUM), genera complicaciones en el proceso de mantenimiento o actualización de las funcionalidades ya que esta no se adapta a los cambios.

La arquitectura genérica propuesta permite al equipo de desarrollo y operaciones tomar las mejores desiciones a la hora de implementar o migrar una funcionalidad para una aplicación web. Si bien, los microservicios son mayormente utilizados en una migración de un sistema monolítico que en la creación de nuevos, el diseño no limita su aplicación para este último debido a que puede construirse microservicios de "grano grueso" (con mas de una funcionalidad) para luego, conforme avanza el desarrollo del proyecto volver a dividirlo hasta llegar a un tamaño adecuado para cada microservicio.

A nivel de infraestructura y operaciones, se ha evidenciado que la aplicación del diseño propuesto plantea una nueva organización de la misma, esto implica la organización tanto de la infraestructura física, como lógica y de comunicaciones para dar soporte a la solución bajo esta arquitectura, todo esto por el uso de tecnologías como la contenerización, balanceo de carga y monitoreo, los cuales difieren del soporte necesario para un sistema monolítico.

Este tipo de diseños de arquitectura bajo microservicios no solo trae consigo una serie de beneficios, sino que también plantea retos en desarrollo de software, operaciones y cultura organizacional ya que platea una nueva forma de construcción y soporte de sistemas de información, estos retos no solo son de tipo tecnológico sino organizacionales.

La arquitectura propuesta permitió realizar un desarrollo enfocado en las funcionalidad del sistema al dividirlo en pequeñas partes manejables en equipos independientes (trabajo en paralelo).

En el proceso de diseño y desarrollo de un sistema bajo el enfoque de microservicios, lograr la separación de funcionalidades es realmente dificultoso si no se cuenta con un conocimiento previo de los procesos que componen el sistema, así como en el empleo de estrategias que permitan aplicar la separación adecuada, sin embargo se sigue los lineamientos que se detallan en la bibliografía consultada, misma que determina una implementación incremental de microservicios, hasta llegar a cumplir con un nivel de granularidad adecuado y cumplir con el principio de responsabilidad única.

4.4. Recomendaciones

La construcción de microservicios requiere de un conocimiento o dominio de negocio amplio, por lo que es recomendable su uso en la migración de sistemas con un nivel de madurez considerable, esto no quiere decir que no se puede usar este tipo de arquitectura para sistemas nuevos, pero si facilita la descomposición del sistema monolítico y su correspondiente desarrollo.

La adopción o implementación de una arquitectura de microservicios implica el cumplimiento de ciertos requerimientos previos, entre los cuales se puede mencionar:

- Despliegue y Aseguramiento de calidad, se requiere desplegar y probar lo más rápidamente posible.
- Plataforma de colaboración para el equipo de desarrollo y operaciones, incorporar las prácticas de DevOps, las que influirán en la forma en que organiza equipos, crea sistemas e incluso la estructura de los sistemas que crea.
- Monitoreo, se necesita monitorear el funcionamiento y la salud de todo el sistema a

fin de detectar problemas.

Con base en lo anterior, es recomendable pero no obligatorio, cumplir previamente estos requerimientos para facilitar su implementación.

Por la naturaleza atómica de los microservicios, sus características y tecnologías que implica, se hace necesario contar con el equipo de desarrollo adecuado, el cual debe tener un conocimiento no solo de un lenguaje de programación sino de otros aspectos tales como: de diseño de interfaces, web, móvil (front-end), lenguajes y frameworks para desarrollo de lógica de negocios (core), bases de datos, repositorios documentales, entre otros; es decir que es deseable que los perfiles de los desarrolladores se orienten a un tipo Full Stack, para facilitar el desarrollo del sistema bajo este enfoque.

La libertad de elección de tecnologías de desarrollo que ofrecen los microservicios trae como consecuencia que el sistema incremente su grado de complejidad al punto de volverse desconocido para un desarrollador o equipo, por tal motivo es recomendable el establecimiento de una política de uso de tecnologías patrones y estilos arquitecturales al memento de su implementación a fin de evitar elevar la comprensibilidad del sistema en su conjunto, sin embargo, las tecnologías base no limita el uso de otras en casos específicos para los cuales dichas tecnologías sean mas apropiadas que las preestablecidas.

A nivel de infraestructura y operaciones, la implementación de microservicios usualmente requiere de su ejecución en un servidor propio, lo que incrementa el uso de máquinas virtuales o contenedores; esto sugiere que debe emplearse automatización de la infraestructura involucrada, la cual permita una correcta organización y despliegue de los microservicios desarrollados, adicionalmente, a mayor número de microservicios, mayor infraestructura es requerida.

La comunicación entre los equipos de desarrollo y operaciones es fundamental a la hora de implementar esta arquitectura, una comunicación fácil permite reducir la necesidad de coordinación del despliegue y operación del sistema, razón por la cual se recomienda adoptar las mejores prácticas impulsadas por el enfoque de DevOps.

REFERENCIAS

A B M Moniruzzaman, y Syed Akhter Hossain. (2013, julio). Comparative Study on Agile software development methodologies. *Global Journal of Computer Science and Technolog*, *XIII*(VII).

Aroraa, G., Kale, L., y Kanwar, M. (2017). *Building Microservices with .Net Core*. Packt Publishing, Limited.

Asamblea Nacional del Ecuador. (2016, marzo). *Codificación del Reglamento Orgánico Funcional.*

Asamblea Nacional del Ecuador. (2017, mayo). *Descripción y arquitectura eCurul v1.0.*

Atchison, L. (2016). *Architecting for Scale: High Availability for Your Growing Applications* (1st ed.). O'Reilly Media, Inc.

Babar, M. A., Brown, A. W., y Mistrík, I. (2013). *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes.

Baig, M. Y. A. (2016). *Build Web Applications with Java: Learn Every Aspect to Build Web Applications from Scratch* (1st ed.). USA: CreateSpace Independent Publishing Platform.

Balalaie, A., Heydarnoori, A., y Jamshidi, P. (2016, mayo). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, *33*(3), 42–52. doi: 10.1109/MS.2016.64

Borčin, T. (2017). *Monitorování aktivity služeb v platformě SilverWare* (Tesis Doctoral no publicada). Masarykova univerzita, Fakulta informatiky.

Bourque, P., y Dupuis, R. (2014). Swebok Version 3.0. *IEEE, ISBN-10: 0-7695-5166-1*.

Braude, E. J., y Bernstein, M. E. (2016). *Software engineering: Modern approaches*. Waveland Press.

Cadavid, A. N. (2013, septiembre). Revisión de metodologías ágiles para el desarrollo de software. *Prospectiva*, *11*(2), 30. doi: 10.15665/rp.v11i2.36

Carneiro, C., y Schmelmer, T. (2016). *Microservices From Day One*. Berkeley, CA: Apress.

Claus Djernæs Nielsen. (2015). *Investigate availability and maintainability within a microservice architecture* (Tesis Doctoral no publicada). Master's thesis, Aarhus University.

de la Torre Llorente, C., Castro, U. Z., Barros, M. A. R., y Nelson, J. C. (2010). Guía de Arquitectura N-Capas orientada al Dominio con .NET.

Dooley, J. (2011). *Software development and professional practice*. Apress.

Fehre, P. (2015). *JavaScript Domain-Driven Design*. Packt Publishing.

Foster, E. (2014). *Software Engineering: A Methodical Approach*. Apress.

Fowler, M., y Lewis, J. (2014). Microservices. *Viittattu*, *28*, 2015.

Hamdan, S., y Alramouni, S. (2015). A Quality Framework for Software Continuous Integration. *Procedia Manufacturing*, *3*, 2019–2025. doi: 10.1016/j.promfg.2015.07.249

Herrera Uribe, E., y Valencia Ayala, L. E. (2007). Del manifiesto ágil sus valores y principios. *Scientia et technica*, *13*(34).

ISO/IEC. (2001). *ISO/IEC 9126. Software engineering – Product quality*. Autor.

ISO/IEC/IEEE. (2008). ISO/IEC/IEEE Standard for Systems and Software Engineering - Software Life Cycle Processes.

(OCLC: 812596257)

Kratzke, N. (2015). About microservices, containers and their underestimated impact on network performance. *Proceedings of CLOUD COMPUTING*, *2015*, 165–169.

Kuchana, P. (2004). *Software architecture design patterns in Java*. CRC Press.

Landre, E., Wesenberg, H., y Rønneberg, H. (2006). Architectural improvement by use of strategic level domain-driven design. En (p. 809). ACM Press. doi: 10.1145/1176617.1176728

Leau, Y. B., Loo, W. K., Tham, W. Y., y Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. En *International Conference on Information and Network Technology* (Vol. 37, pp. 162–167).

Lina Maria Montoya Suarez, Jorge Mauricio Sepúlveda Castaño, y Luisa María Jiménez Ramos. (2017, junio). Análisis comparativo de las metodologías ágiles en el desarrollo de software aplicadas en Colombia. *CIMTED Corporación*, 14.

Mark Richards. (2015). *Software Architecture Patterns* (n.º 9781491924242). O'Reilly Media, Inc.

Millett, S. (2015). *Patterns, Principles and Practices of Domain-Driven Design*. Wiley.

Mishra, A., y Dubey, D. (2013). Suitability Analysis of Various Software Development Life Cycle Models.

Murugesan, V. (2017). *Microservices Deployment Cookbook*. Packt Publishing, Limited.

Nacional, A. (2012, febrero). *Reglamento de funcionamiento de la curul electrónica.* Asamblea Nacional.

Nadareishvili, I., Mitra, R., McLarty, M., y Amundsen, M. (2016). *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Inc.

Narang, R. (2015). *Software Engineering—Principles and Practices*. McGraw-Hill

Education.

Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc.

O'Connor, R. V., Elger, P., y Clarke, P. M. (2016). Exploring the impact of situational context: A case study of a software development process for a microservices architecture. En (pp. 6–10). ACM Press. doi: 10.1145/2904354.2904368

Oksa, M. (2016). *Web API development and integration for microservice functionality in web applications* (Tesis Doctoral no publicada). UNIVERSITY OF JYVÄSKYLÄ.

Posta, C. (2016). *Microservices for Java Developers - O'Reilly Media*. O'Reilly Media, Inc.

Pressman, R. S. (2005). *Software engineering: A practitioner's approach*. Palgrave Macmillan.

Radatz, J., Geraci, A., y Katki, F. (1990). IEEE standard glossary of software engineering terminology. *IEEE Std*, *610121990*(121990), 3.

Raj, P., Chelladhurai, J. S., y Singh, V. (2015). *Learning Docker: Optimize the power of Docker to run your applications quickly and easily*. Birmingham Mumbai: Packt Publishing. (OCLC: 944442093)

Rajesh, R. (2016). *Spring Microservices*. Packt Publishing Ltd.

Rajlich, V. (2011). *Software engineering: The current practice*. CRC Press.

Rusek, M., Dwornicki, G., y Orłowski, A. (2017). A Decentralized System for Load Balancing of Containerized Microservices in the Cloud. En J. Świątek y J. M. Tomczak (Eds.), *Advances in Systems Science* (Vol. 539, pp. 142–152). Cham: Springer International Publishing. doi: 10.1007/978-3-319-48944-5_14

Sangwan, R. S. (2014). *Software and systems architecture in action*. CRC Press.

Scholl, B., Swanson, T., y Fernandez, D. (2016). *Microservices with Docker on Microsoft Azure*. Boston, MA: Addison-Wesley.

Scrum.org. (2016, julio). *What is Scrum?* `http://www.scrum.org/resources/what-is-scrum`.

Shahir Daya, Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, ... Ramratan Vennam (2015). *Microservices fron Theory to Practice* (First Edition ed.). IBM Corporation.

Sharma, S. (2016). *Mastering Microservices with Java*. Packt Publishing Ltd.

Sherland, B., Wade, R., Emery, D., y Hilliard, R. (2000). *IEEE recommended practice for architectural description for software-intensive systems*. New York: Institute of Electrical and Electronics Engineers.

Sommerville, I. (2010). *Software engineering*. Pearson.

Steinegger, R. H., Giessler, P., Hippchen, B., y Abeck, S. (2017, abril). Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications.

Stellman, A., y Greene, J. (2014). *Learning agile: Understanding scrum, XP, lean, and kanban*. O'Reilly Media, Inc.

Stephens, R. (2015). *Beginning software engineering*. John Wiley & Sons.

Strydom, M. (2017). *Software Engineering 021*. KWL.

Tihomirovs, J., y Grabis, J. (2016, enero). Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. *Information Technology and Management Science*, *19*(1). doi: 10.1515/itms-2016-0017

Valverde, F., y Pastor, O. (2009). Dealing with REST services in model-driven web engineering methods. *V Jornadas Científico-Técnicas en Servicios Web y SOA,*

JSWEB, 243–250.

Vernon, V. (2016). *Domain-Driven Design Distilled*. Pearson Education, Limited. (LCCB: 2016936587)

Viktor Farcic. (2016). *The DevOps 2.0 Toolkit*. packtpub.

Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., y Gil, S. (2015, septiembre). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. En *Computing Colombian Conference (10CCC), 2015 10th* (pp. 583–590). doi: 10.1109/ColumbianCC.2015.7333476

Vogel, O., Arnold, I., Chughtai, A., y Kehrer, T. (2011). *Software architecture: A comprehensive framework and guide for practitioners*. Springer Science & Business Media.

Wagh, K., y Thool, R. (2012). A comparative study of soap vs rest web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications*, *2*(5), 12–16.

Wolff, E. (2016). *Microservices: Flexible Software Architectures*. CreateSpace Independent Publishing Platform.

Wolff, E. (2017). *A Practical Guide to Continuous Delivery*. Addison-Wesley Professional.

ANEXO A

GUÍA DEL GRUPO FOCAL REALIZADO CON EL EQUIPO DE DESARROLLO Y AUTORIDADES DE LA CGTIC.

A.0.1. Desarrollo de aplicaciones

1. ¿Qué procesos de desarrollo de aplicaciones se usa habitualmente?
   - Tradicional (Espiral, Cascada, RUP).
   - Ágil (XP, SCRUM).
2. ¿Se cumple todas las fases del ciclo de vida de desarrollo?
   - Explicación o porcentaje de cumplimiento.
3. ¿Qué dificultades se han presentado al desarrollar un nuevo sistema?
   - Explicación
   - Opciones:
     - Toma de requerimientos
     - Definición de tecnologías Integración con otros aplicativos
     - Otro (Usuarios/Requirentes)
4. ¿Cuál es el uso de las aplicaciones web? ¿Cuáles son las de mayor incidencia?
   - Interna
   - Externa
   - Híbrido
5. ¿Cuántas y de qué tipo de aplicaciones se han desarrollado?. ¿Cuáles son planes futuros?.
   - Web
   - Escritorio (Desktop)
   - Móvil
6. ¿Cuál es el promedio de usuarios por tipo de aplicación?
   - Web
   - Escritorio (Desktop)
   - Móvil - aun no

7. ¿En forma descendente, cuantas aplicaciones tienen criticidad?
   - Alta
   - Media resto
   - Baja directorio
8. ¿Qué problemas presenta el mantenimiento de aplicaciones y qué medidas se han tomado al respecto?
   - Tiempo
   - Integración
   - Tecnología usada
   - Otros Personal
9. ¿Durante el mantenimiento o en un evento de falla de un aplicativo crítico, que tiempo toma la solución?
   - Minutos
   - Horas
   - Días
10. ¿Existen servicios alternativos a los principales? En caso afirmativo: ¿Qué tipo de soluciones?
    - Si
    - No

A.0.2. Arquitectura y tecnología

1. ¿El área de desarrollo se tiene definida una arquitectura para todas o para cada tipo de aplicaciones? En caso afirmativo ¿Cuál es? En caso negativo
   - Si
   - No
2. ¿El desarrollo de aplicaciones usa tecnología en versiones actuales? En caso afirmativo ¿En que tipo y que nivel de criticidad?. En caso negativo ¿Por qué no se ha actualizado?
   - Si
   - No
3. ¿Liste las principales tecnologías en uso de las aplicaciones implementadas y en desarrollo?

- Sistemas operativos: Linux/Windows/Mac
- Lenguajes de programación: .Net, JAVA, JavaScript, PHP, Ruby/Python/Perl
- Frameworks: JAVA, JavaScript, PHP, Ruby/Python/Perl
- Base de datos: PostgreSQL, MySQL, MS-SQL, No-SQL

4. ¿El software producido es modular funcionalidad? En caso afirmativo: ¿Qué tecnologías se usaron?. En caso negativo: ¿Por qué no se lo ha hecho?

   - Si
   - No

5. ¿El software producido separa sus funcionalidades o estas están bien definidas? En caso afirmativo: ¿Cuáles son? En caso negativo: ¿Por qué no se lo ha hecho?

   - Si
   - No

6. ¿Cuál es el tipo de integración entre aplicaciones? En caso afirmativo: ¿Cuáles métodos se usa?

7. ¿El software producido expone servicios web? En caso afirmativo: ¿Está estandarizado en cuanto a formato, frameworks, versiones?

   - Si
   - No

8. ¿Cuál es la frecuencia promedio de actualización/cambio en las aplicaciones?

   - Semanal
   - Mensual
   - Trimestral
   - Semestral
   - Anual
   - Otro

9. ¿Las aplicaciones cuentan con documentación suficiente? En caso afirmativo: listar la documentación. En caso negativo: ¿ Por qué no se lo ha hecho?

   - Manual de usuario
   - Manual técnico
   - Diagramas UML
   - En el código

10. ¿Se usan estándares de calidad para el desarrollo de software? En caso afirmativo: ¿Qué se aplica y qué documentación existe? En caso negativo: ¿Existen planes de aplicación y cuales serian?

    - Si
    - No

11. ¿Cuáles son los planes a futuro en el desarrollo de aplicaciones respecto a las tecnologías usadas y las emergentes? Y porqué?

    - Mantener la tecnología actual
    - Actualizar todas o parcialmente las aplicaciones
    - Migrar todas o parcialmente las aplicaciones a nuevas y distintas tecnologías
    - Nuevos desarrollos basados en nuevas tecnologías e integrar los actuales

12. ¿Para usted, qué grado de importancia tiene el empleo de una arquitectura de software y por qué?

    - Alto
    - Medio
    - Bajo

ANEXO B

CARTA DE ACEPTACIÓN DEL TRABAJO DE INVESTIGACIÓN EN EL EVENTO TICAL 2017

Santiago de Chile, 22 de mayo de 2017

Srs.
Daniel López, Edgar Maya
*Pre*sente

Estimados:

Tenemos el agrado de comunicarles que el trabajo presentado por usted, **Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web**, ha sido aceptado para ser parte del programa de la Sexta Conferencia TICAL 2017, a realizarse entre el 3 y el 5 de julio en San José, Costa Rica, bajo el eje temático **Infraestructura y desarrollo de software.**

A continuación, usted encontrará las instrucciones para su participación en la conferencia y para gestionar su viaje.

A) Presentación en la Conferencia:

- ✓ El autor deberá registrarse al evento en http://tical2017.redclara.net/. El costo de la inscripción es:

	Autores en General	Autores pertenecientes a instituciones socias
Hasta el 31 de mayo del 2017	USD 320	USD 256
Desde el 1 de junio del 2017	USD 400	USD 320

- ✓ La presentación será de 20 minutos y 10 minutos para preguntas (30 minutos en total).
- ✓ Enviar una foto del autor que hará la presentación en la conferencia y una presentación de no más de 3 líneas. La foto debe tener las siguientes características: Color, 100 x 120 pixeles (100 pixeles de ancho por 120 de largo), fotografía de rostro y resolución mínima 72 DPI.
- ✓ Enviar el trabajo en formato Word actualizado con las correcciones que el comité ha indicado. Si las hubiera, por favor revise los comentarios que se indican a continuación.
- ✓ La foto y el trabajo actualizado tendrá deberá ser enviado a mi cuenta de correo antes del 20 de agosto.

Comentarios Comité:
Sin Comentarios

B) Viaje y estadía

RedCLARA becará la participación de uno de los autores con el costo de un pasaje de ida y vuelta a la ciudad del evento. Por ello, necesitamos a la brevedad los datos del participante que utilizará el boleto, tales como: nombre como aparece en el pasaporte, número de pasaporte e itinerario preferido. Esta información debe ser enviada a la cuenta de correo secretaria.santiago@redclara.net, con copia a mi cuenta de correo (marcela.larenas@redclara.net), antes del **día 16 de junio 2017.**

La gestión de los pasajes es realizada por la agencia de viajes **Jetmar** de Uruguay. La empresa

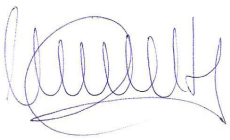
asignará a un ejecutivo que tomará contacto con la persona designada para gestionar su viaje.

Es importante que usted tome en consideración los siguientes aspectos al coordinar su vuelo con la agencia **Jetmar:**

- ✓ El ejecutivo le solicitará información para emitir su boleto: su nombre tal como aparece en el pasaporte, número de pasaporte, país de emisión e itinerario preferido. Por favor asegúrese de ser riguroso con esta información.
- ✓ Por favor haga saber al ejecutivo si ha decidido utilizar el pasaje aéreo para quedarse más tiempo en el lugar del evento. De esta forma se podrá gestionar de la manera más adecuada las fechas de su viaje ya que una vez confirmado el boleto.
- ✓ Una vez emitido el pasaje, en caso que el autor solicite un cambio, los costos de dicho cambio correrán por cuenta del autor.
- ✓ El pasaje contemplado para su viaje será un pasaje en categoría económica y sera desde su ciudad origen ida y vuelta. En caso que desee viajar en una categoría superior o un itinerario diferente, los costos extras correrán por cuenta del autor.
- ✓ En el caso que usted decida viajar acompañado, puede adquirir los pasajes con la misma agencia y el pago del pasaje o los pasajes adicionales deben ser cancelados directamente por usted a la agencia de viajes.
- ✓ Asimismo, si por cualquier causa usted decide no utilizar el boleto de avión después de su confirmación, RedCLARA le pedirá que reembolse el costo del pasaje no utilizado.
- ✓ Las gestiones con la agencia deberán ser realizadas a la brevedad posible, con fecha límite del **16 de junio.** Si usted tiene dificultades para cumplir con esta fecha, por favor hágamelo saber a la brevedad.

Finalmente, es importante señalar que RedCLARA ha publicado en el sitio web de la Conferencia las referencias de hoteles con valores preferenciales, pero no realizará reservas para usted o pagará su estancia.

Agradeciendo su pronta respuesta,



Marcela Larenas
Gerente de Relaciones Académicas
RedCLARA

ANEXO C

CERTIFICADO DE PARTICIPACIÓN COMO AUTOR Y PONENTE EN EL EVENTO TICAL 2017

TICAL

CONFERENCIA 2017

SAN JOSÉ - COSTA RICA - JULIO

CERTIFICADO DE PARTICIPACIÓN

Se otorga el presente reconocimiento a:

José Daniel López H.

procedente de Ecuador, por su participación en calidad de Autor en la séptima Conferencia de Directores de Tecnologías de Información y Comunicación de Instituciones de Educación Superior Latinoamericanas, TICAL2017, llevada a cabo los días 3 al 5 de julio de 2017 en la ciudad de San José, Costa Rica.

CARMEN GLORIA LABBÉ
Gerente General Adjunta
RedCLARA

COOPERACION LATINOAMERICANA DE REDES AVANZADAS
CLARA

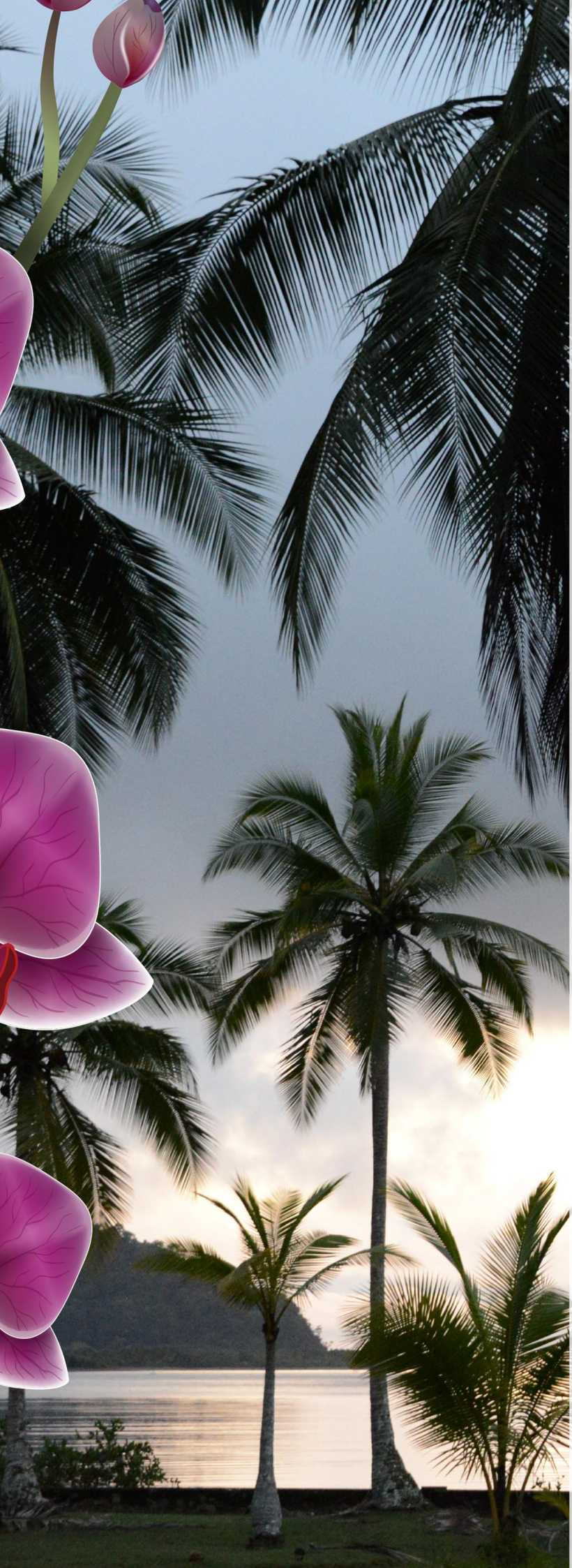
CONSEJO NACIONAL DE RECTORES

RedCLARA
+ Red + Ciencia

RED CONARE

ANEXO D

LIBRO DE ACTAS TICAL 2017



ACTAS

TICAL

CONFERENCIA 2017

SAN JOSÉ - COSTA RICA - JULIO

San José, Costa Rica
Julio | Julho | July 3 - 5, 2017

ACTAS TICAL 2017

Hotel Real Intercontinental, San José, Costa Rica
3 - 5 de julio de 2017

Comité de Programa:

Alonso Castro Mattei, Costa Rica
Presidente del Comité de Programa TICAL2017
Ernesto Chinkes, Argentina
Presidente Honorario TICAL
Jussara Issa Musse, Brasil
Carlos Gabriel Córdova Erreis, Ecuador
Mario Rafael Ruiz Vargas, El Salvador

RedCLARA (http://www.redclara.net)
Fecha en que se terminó la presente edición: 01-09-2017
ISBN: 978-956-9390-08-1

Copyright de la presente edición:

ACTAS TICAL 2017
San José, Costa Rica.
3 - 5 de julio de 2017

ACTAS TICAL 2017

Hotel Real Intercontinental, San José, Costa Rica
3 - 5 de julio de 2017

Índice

PRESENTACIÓN

San José de Costa Rica, en el mes de julio de 2017, fue la sede de la séptima Conferencia TICAL, que permitió debatir sobre las Tecnologías de Información y Comunicaciones (TIC) y su aporte desde distinta áreas y visiones. Una vez más, las experiencias y proyectos muestran un avance sustantivo en la incorporación de las TIC en universidades de la región. Esta publicación presenta 33 trabajos seleccionados y presentados en la Conferencia, sobre un total de 159, que fueron evaluados.

A lo largo de estos siete años, la Conferencia ha ido ampliando los temas abordados con el fin de acoger el avance permanente de las instituciones en la incorporación de TIC y, para ello, cada convocatoria actualiza los ejes temáticos que se privilegian en la Conferencia. En TICAL2016 se definió por primera vez un eje temático exclusivo para recibir las experiencias de las TIC como apoyo a la investigación, a partir de esa experiencia y motivados por los objetivos del Proyecto BELLA (Construyendo el enlace de Europa con América Latina), este año se decidió hacer en forma conjunta con TICAL, el Primer Encuentro Latinoamericano de e-Ciencia, generándose un nuevo espacio para el intercambio de conocimientos y buenas prácticas en el uso de las TIC en las labores de investigación, con vistas a contribuir a la mejora y optimización de la gestión y quehacer de los grupos científicos de la América Latina.
El Encuentro agregó 18 presentaciones en ámbitos relevantes de investigación regional, como son Biodiversidad, e-Salud, Medio Ambiente, Astronomía, Arte y Cultura en Red, y Física de Altas Energías, donde se expusieron experiencias de las diversas formas en las que las TIC son empleadas en el desarrollo de la investigación, siendo un aporte al cumplimiento del objetivo por ella perseguido.

La realización de ambos eventos en un espacio compartido, buscaba fortalecer la colaboración e intercambio de conocimientos entre los grupos de investigación convocados al Encuentro de e-Ciencia, la comunidad de Directores de Tecnología y las Redes Nacionales socias y RedCLARA.

Y, como si lo anterior no fuera suficiente para hacer de esta conferencia el espacio de conocimiento e intercambio de experiencias de las TIC de la región, Internet Society (ISOC), realizó también, de manera exitosa, durante la mañana del 3 de julio,el evento

ION Costa Rica, donde los ingenieros de red y líderes de la industria, revisaron diversos temas relacionados con Internet del futuro y los avances en tecnología, y los asistentes obtuvieron respuestas para implementar nuevos estándares y tecnologías en sus propias redes.

El 2017, por primera vez desde su creación, la Conferencia TICAL se transformó así en un escenario único donde todos los actores relevantes en TIC de la región se convocaron durante tres días para colaborar, intercambiar experiencias, ampliar la red de contactos y reconocer problemáticas comunes, buscando responder mejor el desafío de las TIC en nuestras instituciones, países y en la región.

En estas Actas encontrará las experiencias TIC de distintas instituciones de Argentina, Brasil, Colombia, Costa Rica, Ecuador, Japón, México, Países Bajos, y Uruguay, que fueron seleccionadas por el Comité de Programa de TICAL2017 como las más interesantes e innovadoras para ser presentadas en la Conferencia de este año.

Marcela Larenas Clerc
Gerente de Relaciones Académicas
RedCLARA

ANEXO E

ARTÍCULO

Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de las TICs para la Investigación y la Colaboración, San José, del XX al XX de julio de 2017

Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web

Daniel López[a], Edgar Maya[b]

[a] Asamblea Nacional del Ecuador, Coordinación General de Tecnologías de la Información y Comunicación, Piedrahita 5-21 y Av. Gran Colombia, Pichincha, Ecuador
daniel.lopez@asambleanacional.gob.ec

[b] Universidad Técnica del Norte, Instituto de Posgrados, Av. 17 de Julio 5-21 Gral. José María Cordova, Imbabura, Ecuador
eamaya@utn.edu.ec

Resumen. Actualmente, el proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software tradicional o monolítica que ha sido adoptada del lenguaje de programación utilizado, la plataforma o de la experiencia del personal del área de desarrollo; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas. El objetivo del presente estudio fue identificar las tecnologías, metodología y arquitectura que utiliza la CGTIC para el desarrollo de aplicaciones web y la correspondiente identificación de las tecnologías existentes para el desarrollo e implementación de microservicios, utilizando como base de la investigación un enfoque cualitativo, con un tipo de investigación descriptiva y diseño documental. Se empleó la técnica de grupo focal aplicado a los funcionarios del área de desarrollo de software de la CGTIC, revisión bibliográfica de arquitectura de microservicios. Como avance de la investigación, el análisis ha permitido identificar el estado del arte respecto a microservicios y su implementación así como la identificación de los requisitos y necesidades relativos al desarrollo de aplicaciones web y como satisfacerlas mediante el diseño de una arquitectura de software.

Palabras Clave: Microservicios, arquitectura de software, aplicaciones web.

Eje temático: Infraestructura y desarrollo de software.

1 Introducción

En la actualidad, a nivel empresarial y tanto en el sector privado como público se realiza desarrollo de software para suplir las necesidades de automatización de procesos internos, este desarrollo ha seguido las tendencias impuestas por la plataforma, lenguaje de programación o por la experiencia del área de desarrollo, lo cual deviene en la implantación de sistemas de construcción tradicional o monolítico.

El proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software monolítica,

misma que ha sido adoptada por el uso de un lenguaje de programación específico para construcción de aplicaciones web empresariales; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas [1]

1.1 Problema

El presente trabajo de investigación, parte de un análisis de la problemática existente con el desarrollo de aplicaciones web de la CGTIC, dicho desarrollo hace uso de una arquitectura monolítica la cual incide en diferentes aspectos tanto tecnológicos y administrativos. Las incidencias más evidentes se pueden observar al aplicar procesos de mantenimiento en sistemas complejos, sea por una petición de cambio, nueva funcionalidad o corrección de un fallo, en los cuales, la resolución de un problema o cambio simple implica el redespliegue de toda la aplicación debido a que se tiene todas las funcionalidades en un único paquete incrementando los riesgos de fallos. De igual forma, por el tiempo que toma la implementación de un cambio o nueva funcionalidad, se presenta resistencia al cambio en el usuario final, y en consecuencia las actualizaciones son menos frecuentes porque requieren de un mayor esfuerzo y coordinación de los grupos de desarrollo y la realización de pruebas más extensas.

En aspectos de calidad, surgen complicaciones en la escalabilidad ya que puede requerirse escalar un módulo específico, pero, por el aspecto monolítico es necesario escalar la aplicación en su totalidad. De igual manera sucede con la resiliencia, al ocurrir un fallo por caída o sobrecarga en una parte de la aplicación, se pierden todas sus funcionalidades; si bien este último aspecto puede ser subsanado mediante replicación o clusters, también incrementa las dificultades de coordinación, configuración y eleva los costos de equipamiento y esfuerzo.

Otra problemática a resaltar, es la incidencia que produce el mantenimiento aplicaciones en el normal desenvolvimiento de las actividades de los usuarios; en este sentido, una de las áreas más críticas es el Plenario de la ANE, el cual hace uso de software para registrar las votaciones de los legisladores en la creación o reforma de leyes de afectación nacional, por lo que resulta difícil cambiar, agregar o actualizarlo. Adicionalmente, algunos de los sistemas se encuentran impedidos de actualización que es de vital importancia por su nivel de criticidad y afectación, esto se debe una vez más por estar construida bajo el esquema monolítico.

Por los aspectos descritos se concluye que existe la necesidad de definir una nueva arquitectura de software con un enfoque de vanguardia que facilite el desarrollo de nuevas aplicaciones para las diferentes unidades organizacionales de la ANE. Bajo esta nueva perspectiva, se pretende obtener arquitectura de software flexible que permita el mantenimiento de las aplicaciones evitando la interrupción de actividades del personal que las usa.

2 Antecedentes

En la Universidad de Aarhus de Dinamarca [2], evalúa diferentes tácticas de disponibilidad y mantenibilidad en la construcción de un prototipo con arquitectura de microservicios, en el proceso, concluye con una revisión de las características de los microservicios enunciadas por Fowler y Lewis [3]en cuatro criterios principales que definen a un microservicio:

Enfoque en las capacidades de negocio. El proceso de desarrollo de software se vuelve más flexible bajo el enfoque de microservicios, cada microservicio está compuesto por su propia lógica de negocios, interfaz de usuario, base de datos y funcionalidad de comunicación con otros servicios, por lo tanto, en el equipo de desarrollo que cada miembro necesita experiencia en desarrollo de backend, frontend, y bases de datos, permitiéndoles agregar nuevas características rápidamente sin arriesgar la estabilidad y el funcionamiento de todo el sistema.

Independencia de los servicios autónomos. Cada servicio contiene su propia lógica de negocio y se despliega por separado. Esto hace posible cambiar y extender gradualmente con nuevas características sin afectar el resto del sistema. La propiedad autónoma permite la flexibilidad en la selección de la tecnología más adecuada para un servicio específico.

Gestión descentralizada de datos. Todos los servicios tienen su propia base de datos en esquemas pequeños y simples, que se ven por separado. Los datos están desacoplados, lo que requiere mucha gestión y pruebas para garantizar que un sistema nunca actualice o elimine los datos en un servicio sin actualizar o eliminar los datos correspondientes en otros servicios que contienen los mismos datos o conjunto de referencias.

Tolerancia a fallos. Los sistemas de microservicio consisten en múltiples unidades pequeñas que pueden fallar; Estas unidades están ligeramente acopladas y pueden ser restauradas automáticamente. lo que resulta en un sistema más estable y robusto.

Por su parte Borčin [4] en su trabajo de investigación, realiza un análisis de los atributos de calidad de la arquitectura de microservicios profundizando en los retos de su implementación:

Interfaces y comunicación. Cada microservicio debe exponer alguna interfase, para algunas tecnologías no equipadas con una especificación, esto puede representar serios problemas incluso REST que es utilizado en la arquitectura de microservicio para la comunicación no tiene una interfaz bien definida.

Transacciones y bloqueos. Se necesita un conjunto completamente nuevo de operaciones para una transacción que comprueba la comunicación de microservicios.

se debe asegurar de que ningún microservicio utilice flujos cerrados o recursos bloqueados y que no modifique los datos que otros servicios están utilizando. Estos bloqueos podrían provocar la suspensión de todo el sistema.

Programación coreográfica. El paradigma de programación coreográfica puede ayudar en la creación de sistemas libres de bloqueos, sin errores de comunicación. La programación coreográfica puede llegar a ser más importante con el desarrollo ulterior de la arquitectura del microservicio, ya que proporcionan un terreno sólido para una formalización de la comunicación.

Puntos de entrada. La seguridad es un problema importante cuando se trata de la arquitectura de microservicios y los sistemas distribuidos en general. En aplicaciones monolíticas se puede asegurar fácilmente la seguridad de toda la aplicación, ya que toda la comunicación pasa a través de los puntos de entrada que posee cada módulo que lo compone. En contraste, en microservicios, garantizar la seguridad se vuelve mucho más difícil. Toda la aplicación se compone de un microservicio independiente (lenguaje y máquina) que expone sus interfaces y el núcleo de la aplicación para la comunicación exterior, proporcionando nuevos puntos de entrada para los intrusos.

Red compleja. Una extensa red de comunicación puede ser fácilmente expuesta a ataques, ya que es difícil administrar cuando la aplicación consta de muchos microservicios y esos servicios envían miles de mensajes. El monitoreo de esta compleja red es también muy difícil.

Concluye que, la arquitectura de microservicios traerá nuevos problemas y desafíos relacionados no sólo con la comunicación y su estructura distribuida. Actualmente sólo se puede decir que los microservicios todavía están en una etapa inicial y aún es desconocido para muchos desarrolladores, pero están ganando mucha atención y un sitio en el mercado, especialmente entre una medianas y pequeñas empresas.

2.1 Microservicios

Es un enfoque para el desarrollo de una aplicación única como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y mecanismos ligeros de comunicación, a menudo un recurso de una interfaz de programación de aplicaciones (API) sobre protocolo de transferencia de hipertexto (HTTP). Estos servicios están construidos alrededor de las capacidades del negocio y con independencia de despliegue e implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, los que pueden estar escritos en lenguajes de programación diferentes y utilizar diferentes tecnologías de almacenamiento de datos [3].

El término microservicios no es relativamente nuevo, este estilo arquitectural fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una

definición en concreto para microservicio, sin embargo, una aproximación que la realiza [5] lo define como: "Pequeños servicios autónomos que trabajan juntos".

Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto-contenidas, lo cual difiere de la forma tradicional o monolítico [6].

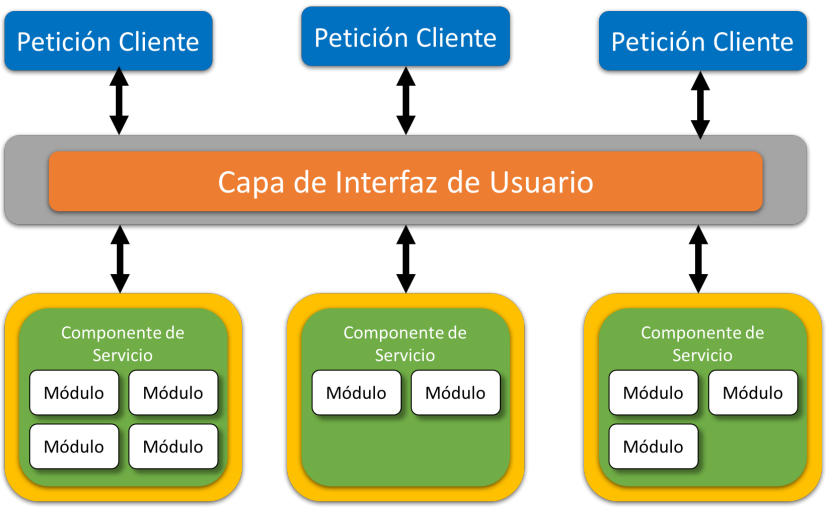


Fig. 1. Patrón básico de arquitectura de Microservicios [7]

Una de las ventajas de utilizar microservicios es la capacidad de publicar una aplicación grande como un conjunto de pequeñas aplicaciones (microservicios) que se pueden desarrollar, desplegar, escalar, manejar y visualizar de forma independiente. Los microservicios permiten a las empresas gestionar las aplicaciones de código base grande usando una metodología más práctica donde las mejoras incrementales son ejecutadas por pequeños equipos en bases de código y despliegues independientes. La agilidad, reducción de costes y la escalabilidad granular, traen algunos retos de los sistemas distribuidos y las prácticas de gestión de los equipos de desarrollo que deben ser considerados. [8]

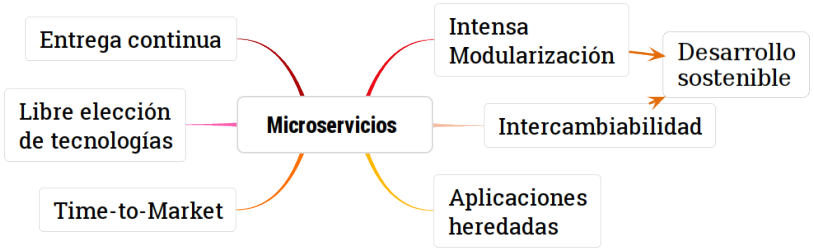


Fig. 1. Beneficios de los microservicios

Topologías. Según Richards [7], existen muchas maneras de implementar un patrón de arquitectura de microservicios, pero se destacan tres topologías principales que son las más comunes y populares: basada en API REST (Transferencia de Estado Representacional del inglés Representational State Transfer), basada en aplicaciones REST y la centralizada de mensajería.

- Topología basada en API REST es útil para sitios web que exponen servicios individuales pequeños y autónomos a través de algún tipo de API. Consta de componentes de servicio de grano fino (de ahí el nombre microservicios) que contienen uno o dos módulos que realizan funciones empresariales específicas independientes del resto de los servicios. En esta topología, estos componentes de servicio de grano fino se acceden normalmente mediante una interfaz basada en REST implementada a través de una capa de API basada en la Web desplegada separadamente. Algunos ejemplos de esta topología incluyen algunos de los servicios web REST basados en la nube como los usados por Yahoo, Google y Amazon.
- Topología basada en aplicaciones REST difiere del enfoque basado en API REST en que las solicitudes de cliente se reciben a través de pantallas de aplicaciones empresariales tradicionales basadas en web o en clientes pesados (fat-client) en lugar de una simple capa de API. La capa de interfaz de usuario de la aplicación se despliega como una aplicación web separada que accede de forma remota a componentes de servicio desplegados por separado (funcionalidad empresarial) a través de simples interfaces basadas en REST. Otra diferencia se encuentra en que los componentes de servicio tienden a ser más grandes, de grano más grueso y representan una pequeña porción de la aplicación empresarial general en lugar de granos finos, servicios de acción simple. Esta topología es común para las aplicaciones empresariales pequeñas y medianas que tienen un grado relativamente bajo de complejidad.
- La topología de mensajería centralizada, es similar a la basada en REST, excepto que en lugar de usar REST para acceso remoto, esta utiliza un intermediario de mensajes centralizado ligero (por ejemplo, ActiveMQ, HornetQ, etc.). Es importante considerar que no se debe confundir con el patrón SOA o considerarlo "SOA-Lite". El agente de mensajes ligeros que se encuentra en esta topología no realiza ninguna orquestación, transformación o enrutamiento complejo, es sólo un transporte ligero para acceder a los componentes de servicio remotos. Esta topología se encuentra frecuentemente en aplicaciones de negocios más grandes o aplicaciones que requieren un control más sofisticado sobre la capa de transporte entre la interfaz de usuario y los componentes de servicio. Entre los beneficios que aporta frente a las anteriores es que son mecanismos avanzados de colas, mensajería asíncrona, monitoreo, manejo de errores y mejor balanceo de carga y escalabilidad. El único punto de fallo y los problemas de cuello de botella arquitectónico generalmente asociados con un intermediario centralizado se abordan a través de la agrupación de intermediarios y de la federación de intermediarios (dividir una única instancia de intermediario en múltiples instancias de intermediario para dividir la carga de procesamiento de mensajes en función de las áreas funcionales del sistema).

Aplicación monolítica. En una aplicación monolítica toda la lógica se ejecuta en un único servidor de aplicaciones. Las aplicaciones monolíticas típicas son grandes y construidas por múltiples equipos, requiriendo una orquestación cuidadosa del despliegue para cada cambio. También se consideran aplicaciones monolíticas cuando existen múltiples servicios de API que proporcionan la lógica de negocio, toda la capa de presentación es una sola aplicación web grande. En ambos casos, la arquitectura de microservicios puede proporcionar una alternativa.

Tabla 1. Arquitecturas monolíticas vs. microservicios [9]

Categoría	**Arquitectura Monolítica**	**Arquitectura de microservicios**
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Lenguaje	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera, aunque los cuellos de botella estén localizados.	Permite escalar servicios con cuello de botella sin escalar la aplicación completa.

2.2 Servicios web.

Los servicios Web son componentes de software ligeramente acoplados entregados a través de tecnologías estándar de Internet. Es decir, los servicios Web son aplicaciones de negocio autodescriptivas y modulares que exponen la lógica empresarial como servicios a través de Internet a través de la interfaz programable y donde el protocolo de Internet (IP) puede ser utilizado para encontrar e invocar esos servicios. Un servicio web es el elemento clave en la integración de diferentes sistemas de información, ya que los sistemas de información pueden basarse en diferentes plataformas, lenguajes de programación y tecnologías [10].

SOAP. La implementación de servicios web por protocolo simple de acceso a objetos (SOAP) se desarrolló como una alternativa al estándar CORBA (Common Object Request Broker Architecture). Para garantizar el transporte de datos en SOAP, se utilizan protocolos como HTTP, SMTP, entre otros, en formato XML. En este enfoque, un proveedor de servicios publica una descripción del servicio o una interfaz para el registro de servicios, por lo que el solicitante del servicio puede encontrar una instancia de servicio correcta y utilizarla. Algunos problemas de rendimiento en SOAP se producen al formar el mensaje SOAP ya que agrega un encabezado adicional y partes al cuerpo al mensaje. Los servicios Web basados en SOAP incluyen una variedad de estándares, tales como WSDL, WSBPEL, WS-Security, WS-Addressing Estas normas fueron desarrolladas por organizaciones de normalización, como W3C y OASIS [11]

REST. Un servicio REST se define como una agregación de diferentes recursos que pueden ser alcanzados desde un identificador universal de recurso (URI) base. Un recurso representa a una entidad del mundo real cuyo estado está expuesto y puede cambiarse accediendo a un URI. Una Representación es la descripción de los mensajes enviados o recibidos de un Recurso en términos de un lenguaje tecnológico. Actualmente XML y JSON son los idiomas más populares para describir estos mensajes [12]

2.3 Integración continua

La integración continua (CI) es una práctica en el desarrollo de software en la que los desarrolladores hacen integraciones automáticas de un proyecto lo más a menudo posible (generalmente a diario) con el propósito de detectar fallos lo antes posible. Esta práctica de integración comprende la compilación y ejecución de pruebas de todo un proyecto.

Prácticas de integración continua. Para la aplicación de prácticas de CI se requiere que cada vez que se agregue una nueva parte al sistema, también se creen casos de prueba automáticos para cubrir todo el sistema incluyendo las partes recién agregadas. De igual forma, se requiere que el software sea probado y construido automáticamente y una retroalimentación inmediata sobre los nuevos códigos integrados hacia los desarrolladores. De acuerdo con Hamdan y Alramouni [13], estas prácticas son:

- Integrar el código con frecuencia. Este núcleo de la integración continua. No se debe esperar más de un día para enviar código al repositorio de código compartido.
- No integrar código “roto”. Código roto es un código que contiene cualquier tipo de error cuando se incluye en una construcción de CI.
- Corregir las compilaciones no funcionales de inmediato. Una compilación no funcional puede ser un error en la compilación, en la base de datos o en la implementación. Es cualquier cosa que impide que la compilación de informes de éxito.
- Escribir pruebas automatizadas. Las pruebas deben ser automatizadas para que funcionen en un sistema de CI. También debe cubrir todo el código fuente.

- Todas las pruebas e inspecciones deben aprobarse. Para que una compilación se apruebe, el 100% de las pruebas automatizadas deben pasar con éxito. Este es el criterio más importante de CI en cuanto a la calidad del software.
- Ejecutar compilaciones privadas. Las herramientas de CI permiten a los desarrolladores tener una copia del software del repositorio de código compartido localmente en sus estaciones de trabajo. Esto les proporciona la capacidad de tener una compilación de integración reciente localmente antes de integrarla al servidor de generación de integración principal para asegurar que no falle.

2.4 Contenedor de aplicaciones

Un contenedor de aplicaciones se basa en la virtualización de sistemas operativos usando los contenedores Linux [14], precisamente es un motor de contenedor de código abierto, que automatiza el empacado, entrega y despliegue de cualquier aplicación de software, se presenta como contenedores livianos, portátiles y autosuficientes, que se ejecutarán prácticamente en cualquier lugar. Un contenedor es un cubo de software que comprende todo lo necesario para ejecutar el software de forma independiente. Puede haber varios contenedores en una sola máquina y los contenedores están completamente aislados entre sí y también de la máquina anfitriona [15]

La palabra contenedor se define como "un objeto para contener/resguardar o el transporte de algo". La idea detrás de los contenedores de "software" es similar. Son imágenes aisladas e inmutables que proporcionan funcionalidad diseñada en la mayoría de los casos para ser accesibles sólo a través de sus APIs. Es una solución para hacer que el software funcione de forma fiable y en (casi) cualquier entorno. No importa dónde se estén ejecutando (computador portátil, servidor de pruebas o de producción, centro de datos, etc.), el resultado siempre debe ser el mismo [16].

3 Metodología

Como base de la investigación se usó un enfoque cualitativo que permitió descubrir o afinar las preguntas de investigación sobre las necesidades del diseño de una arquitectura de software. Se aplica un tipo de investigación descriptiva [17] para la contextualización y descripción del entorno de aplicación de una arquitectura en el desarrollo de aplicaciones web. El diseño documental [18] utilizado para la revisión bibliográfica sobre arquitectura de microservicios. Se empleó la técnica de grupo focal [19] aplicado a los funcionarios del área de desarrollo de software de la CGTIC para determinar el estado de situación actual sobre proceso de software aplicado.

3.1 Análisis Cualitativo

Recolección de datos. El instrumento de reunión de expertos fue desarrollado tomando en cuenta los antecedentes del entorno, tales como: Tecnología en uso, documentación disponible, experiencia del personal.

La reunión de expertos tuvo dos enfoques principales:

- Desarrollo de aplicaciones
- Arquitectura de software

Por cada pregunta se definió posibles respuestas esperadas como guía del conductor de acuerdo su ámbito relacionado.

Resultados. Con respecto al desarrollo de software y con base en la información recolectada de la aplicación del instrumento de investigación se concluye que:

El proceso de desarrollo de software utiliza una metodología ágil debido al carácter cambiante de los sistemas en ejecución, estas a su vez, constituyen en su mayoría los orientados a la web, con una clara tendencia al desarrollo en entornos móviles. Existe un uso mayoritario de aplicaciones a nivel interno con alta criticidad mismas que no cuentan con servicios alternativos en presencia de fallos o eventos fortuitos que permitan la continuidad del servicio.

El producto de software que se genera carece de principios de modularidad debido al uso de la arquitectura implícita de la plataforma de desarrollo utilizada. Como resultado las aplicaciones adquieren un carácter monolítico embebiendo en una sola unidad ejecutable todas sus funcionalidades lo que incide tanto en su desarrollo como mantenimiento.

En consecuencia y bajo el criterio unificado obtenido, se concluye que es de alta importancia emplear una arquitectura de software para el desarrollo de software que permita una estandarización de las aplicaciones y brinde una mejor calidad a los productos de software.

3.2 Proceso de arquitectura de software

El proceso de diseño de la arquitectura es una extensión del proceso general de diseño de ingeniería. El diseño de la arquitectura se centra en la descomposición de un sistema en componentes y las interacciones entre los componentes para satisfacer los requisitos funcionales y no funcionales. Un sistema de software puede ser visto como una jerarquía de las decisiones de diseño (reglas también llamado diseño o contratos). Cada nivel de la jerarquía tiene un conjunto de reglas de diseño que de alguna manera se une o conecta los componentes en ese nivel. La salida principal del proceso de diseño de la arquitectura es una descripción arquitectónica. El proceso describe los siguientes pasos generales [20]:

- El establecimiento de requisitos, mediante el análisis de los controladores del negocio, el contexto del sistema, y los factores que los interesados del sistema estimen crítico para el éxito.
- La definición de una arquitectura, mediante el desarrollo de estructuras arquitectónicas y estrategias de coordinación que satisfagan los requisitos.

- La evaluación de la arquitectura, mediante la determinación de cuándo y qué métodos de evaluación de la arquitectura son apropiados, la realización de las evaluaciones, y la aplicación de los resultados para mejorar el desarrollo de la arquitectura.
- La documentación de la arquitectura, con el suficiente detalle y en una forma fácilmente accesible para los desarrolladores y otras partes interesadas.
- El análisis de la arquitectura, para el rendimiento del sistema, la seguridad.
- Implementación, de la arquitectura definida a través del desarrollo de un prototipo.

4 Resultados

La investigación se encuentra en etapa de desarrollo de la propuesta y posteriormente su aplicación mediante la implementación de un prototipo bajo la arquitectura propuesta que demuestre su uso, por lo que aún no se ha obtenido resultados.

5 Conclusiones

- El actual desarrollo de software bajo la metodología aplicada se contrapone a la arquitectura utilizada, lo cual dificulta su correcta aplicación.
- El mantenimiento o cambio en las funcionalidades de las aplicaciones representa un problema debido a su naturaleza monolítica.
- Existe una clara intención de establecer una nueva arquitectura de desarrollo de aplicaciones acorde a la utilización de nuevas tecnologías.

Agradecimientos

Este trabajo forma parte de la Tesis Arquitectura de Software Basada en Microservicios para Desarrollo de Aplicaciones Web de la Asamblea Nacional.

Referencias

1. Carneiro, C.; Schmelmer, T.: Microservices From Day One. Apress. Berkeley, CA. (2016).
2. Nielsen, D. Investigate availability and maintainability within a microservice architecture (Tesis Doctoral). Master's thesis, Aarhus University. (2015).
3. Fowler, M.; Lewis, J. Microservices. Viittattu, (2014)
4. Borčin T. Service activity monitoring for SilverWare. Masaryk University. (2017)
5. Newman, S. Building Microservices. O'Reilly Media, Inc. (2015).
6. Wolff, E. Microservices: Flexible Software Architectures. CreateSpace Independent Publishing Platform. (2016).
7. Richards, M. Software Architecture Patterns. O'Reilly Media, Inc. (2015).

8. Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., y Gil, S. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. Computing Colombian Conference (10CCC), 2015 10th (pp. 583–590). (2015).
9. Daya, S., Van Duy, N., Eati, K., Ferreira, C., Glozic D., Gucer, V., Vennam R. Microservices fron Theory to Practice IBM Corporation.(2015).
10. Wagh, K., Thool, R. A comparative study of soap vs rest web services provisioning techniques for mobile host. Journal of Information Engineering and Applications, 2(5), 12–16. (2012).
11. Tihomirovs, J., Grabis, J. Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. Information Technology and Management Science, 19(1). (2016)
12. Valverde, F., Pastor, O. Dealing with REST services in model-driven web engineering methods. V Jornadas Científico -Técnicas en Servicios Web y SOA, JSWEB, 243–250. (2009).
13. Hamdan, S., Alramouni, S. A Quality Framework for Software Continuous Integration. Procedia Manufacturing, 3. (2015).
14. Kratzke, N. About microservices, containers and their underestimated impact on network performance. Proceedings of Cloud Computing, 2015, 165–169. (2015).
15. Raj, P., Chelladhurai, J. S., Singh, V. Learning Docker: Optimize the power of Docker to run your applications quickly and easily. Birmingham Mumbai: Packt Publishing. (2015).
16. Farcic, V., The DevOps 2.0 Toolkit. packtpub. (2016).
17. Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P.. Metodología de la investigación. 5ta Edición La Habana: Editorial Félix Varela, 2. (2010)
18. Galeano, M. E. Diseño de proyectos en la investigación cualitativa. Universidad Eafit. (2004).
19. Álvarez-Gayou, J. L. Cómo hacer investigación cualitativa. Fundamentos y metodología. Colección Paidós Educador. México: Paidós Mexicana. (2003).
20. Babar, M., Brown, A., Mistrík, I., Agile Software Architecture: Aligning Agile Processes and Software Architectures. Newnes, (2013).