ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

MIGRACIÓN HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS DEL SISTEMA DE GESTIÓN CENTRALIZADA DE LABORATORIOS DE LA DGIP

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

CHULCA QUILACHAMÍN CRISTHIAN ANDRES

cristhian.chulca@epn.edu.ec

MOLINA LOPEZ RAÚL PATRICIO

raul.molina@epn.edu.ec

DIRECTOR: PhD. FLORES NARANJO PAMELA CATHERINE

pamela.flores@epn.edu.ec

CO-DIRECTOR: MSc. EGUEZ SARZOSA VICENTE ADRIAN

vicente.egueze@epn.edu.ec

Quito, abril de 2020

DECLARACIÓN

Nosotros, Chulca Quilachamín Cristhian Andres y Molina López Raúl Patricio, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____ _____

Chulca Quilachamín Cristhian Andres **Molina López Raúl Patricio**

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Chulca Quilachamín Cristhian Andres y Molina López Raúl Patricio, bajo mi supervisión.

_____

Flores Naranjo Pamela Catherine

DIRECTOR DE PROYECTO

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Chulca Quilachamín Cristhian Andres y Molina López Raúl Patricio, bajo mi supervisión.

_____

Eguez Sarzosa Vicente Adrián

CO-DIRECTOR DE PROYECTO

AGRADECIMIENTO

A mi familia, mis padres Fabiola y Milton, mi abuelita Luz María que me dieron la oportunidad de estudiar y siempre han estado pendientes del avance de mi carrera de estudiante, a mis hermanos Ronny, Byron y su esposa Valerio por el apoyo incondicional que nos demostramos en los momentos necesarios, mis sobrinos Alann y Toño por sus travesuras y cariño que me brindan. Extiendo un fraterno agradecimiento a mis amigos que los considero como verdaderos hermanos/as: Silvana, Andrés, Johan, Raúl, Bryan, Roberto. Un cordial agradecimiento a Cesar Colcha por sus consejos entorno a una vida profesional, además a Kitu y Aldemar gracias por llegar a mi vida.

De igual manera a los profesores de quienes he recibido cátedra en toda mi vida estudiantil, con una dedicatoria especial a Pamela Flores, directora de mi proyecto de tesis, de quien puedo manifestar es una gran persona a quien admiro y le desearé siempre el mejor de los éxitos.

Cristhian Chulca

AGRADECIMIENTO

No se pueden encontrar las palabras exactas, ni este espacio es lo suficientemente grande, para agradecer y expresar, a todos quienes han sido parte y han contribuido de alguna manera en mi vida académica. Solo me queda retribuir todo el apoyo recibido, el esfuerzo, y el tiempo dedicado, con trabajo honesto, lealtad, apoyo y colaborando con quienes más lo necesiten, empleando los conocimientos y habilidades adquiridas a lo largo de este proceso.

De manera especial quiero expresar mi agradecimiento a mis padres y mi pareja quienes han estado junto a mí en todo momento, además de todo el apoyo moral y económico que me han brindado. Así mismo a la Dra. Pamela Flores y al Ing. Roberto García por su dedicación y apoyo para la realización del presente trabajo, además a mi compañero de tesis y amigos que fueron parte no solo de esta etapa final, sino a lo largo de toda la carrera compartiendo logros y venciendo dificultades.

Raúl Molina

DEDICATORIA

Dedico mi proyecto de tesis para lograr mi título de ingeniero y mis siguientes logros en mi vida profesional a mi abuelita Luz María, quien me ha enseñado a esforzarme por mis objetivos de vida, a trabajar y no declinar hasta conseguirlos. A mi familia entera por acompañarme durante esta fuerte etapa de estudios universitarios en la Poli.

Cristhian Chulca

DEDICATORIA

Dedico este trabajo a mis padres Rosa y Raúl por todo su apoyo, cariño, esfuerzo y sabios concejos a lo largo de mi vida, a mi pareja Fernanda quien llego a mi vida como una luz renovadora, quien me dio fuerzas y ánimos para seguir adelante. Así también a mis hermanas sobrinos y el resto de mi familia quienes han sido parte muy importante de mi vida.

Raúl Molina

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

A lo largo del ciclo de vida del software y en casi todas las metodologías se contempla una etapa de diseño de arquitectura de software, la misma que en poca o gran medida ha sido muchas veces solapada por la etapa de implementación del software. Producto de este solapamiento se han creado e implementado arquitecturas monolíticas incapaces de crecer, emergiendo de esta manera la necesidad de migrar a otra arquitectura que le otorgue al sistema la capacidad de flexibilidad, modificabilidad y comprensibilidad del software; beneficios de los que carecen las arquitecturas monolíticas. Una de las arquitecturas que se caracteriza por su escalabilidad es la arquitectura de Microservicios. El buen diseño de una arquitectura de microservicios, representa un conjunto de servicios que son desplegables de forma independiente y que ofrece un alto grado de resiliencia y escalabilidad para la aplicación.

Sin embargo, el proceso de transformar una arquitectura a otra requiere una gran inversión en un plan de ejecución para lograr una migración exitosa, este reto es abordado para el presente trabajo de titulación donde se plantea la implementación de una arquitectura de microservicios basado en una propuesta metodológica de migración de arquitecturas monolíticas hacia microservicios. La implementación aquí presentada contempla el estudio de herramientas disponibles para el desarrollo de microservicios, la selección de la metodología de desarrollo ágil, la adquisición de conocimiento previo de la funcionalidad de la aplicación a migrar, para luego llevar a cabo un completo análisis del código fuente y finalmente aplicar un trabajo de ingeniería inversa sobre la base de datos para comprender los procesos de negocio de la aplicación. La implementación de la migración presentada en este trabajo de titulación se ha llevado a cabo sobre el Sistema de Gestión Centralizada de Laboratorios (SISLAB), perteneciente al Sistema Integrado de Información (SII) de la Escuela Politécnica Nacional del Ecuador.

Palabras clave: Desarrollo de Software, Diseño de Arquitectura, Migración, Arquitectura de Microservicios.

ABSTRACT

Throughout the software life cycle and a stage of software architecture design is contemplate by virtually all the methodologies, usually this stage is overlapped by implementation stage. As a result of this overlapping, monolithic architectures unable to grow functionally have been created and implemented, in this way, appear the need to migrate into another architecture that gives flexibility, modifiability and understandability of the software to system, these benefits are not present in monolithic architectures. One of the architectures that is characterized by its scalability is the microservice architecture. The correct design of a microservice architecture, represents a set of services that are deployable independently and it offers a high degree of resilience and scalability to application.

However, the process of transforming one architecture to another, requires a large investment in an execution plan to achieve a successful migration, this challenge is addressed for this degree work, where the implementation of a microservice architecture based on a methodological proposal of migration of monolithic architectures to microservices is proposed. The implementation presented here contemplates the study of available tools for the development of microservices, the selection of the agile development methodology, the acquisition of prior knowledge of the functionality of the application to migrate, and then to carry out a complete analysis of the source code and finally apply a reverse engineering work on the database to understand the business processes of the application. The implementation of the migration presented in this degree work has been carried out on the “Sistema de Gestión Centralizada de Laboratorios” (SISLAB), belonging to the “Sistema Integrado de Información” (SII) of the National Polytechnic School of Ecuador.

Keywords: Software Development, Desing Arquitecture, Migration, Microservices Arquitecture.

INTRODUCCIÓN

El Sistema de Gestión Centralizada de Laboratorios (SISLAB), de la Escuela Politécnica Nacional, nace con el objetivo de llevar un control centralizado de la facturación e inventario de los laboratorios que brindan servicios, tanto a la comunidad politécnica como al público en general. Desde su creación, el sistema se encuentra en un cambio continuo, producto de nuevas funcionalidades requeridas y el aumento de características del sistema, además de las ya implementadas, asimismo debido a su naturaleza monolítica y su base de tecnología tradicional, han provocado problemas en aspectos como escalabilidad y mantenibilidad. Asimismo, la alta demanda de estabilidad y resiliencia del sistema requiere reforzar su estructura, así como también su arquitectura.

En el presente trabajo se realizó la migración del sistema SISLAB para solventar los problemas anteriormente mencionados, basado en la propuesta metodológica presentada en el Proyecto de Titulación de Arboleda Cola Carlos Augusto, Propuesta Metodológica para Migración de Sistemas Web con Arquitectura Monolítica hacia una Arquitectura Basada en Microservicios.

En las primeras etapas de la implementación es fundamental obtener un completo dominio de la aplicación a migrar, por lo que de manera inicial se realiza un estudio de la actualidad del sistema, así como también los cambios históricos desde su creación. De la misma manera se realiza un estudio breve de la propuesta metodológica presentada por Arboleda Cola Carlos Augusto, ya que, al ser un estudio teórico, existen nuevas implicaciones a considerar al llevar la propuesta teórica a un caso práctico, además del análisis de aplicabilidad de la propuesta dentro del contexto del SISLAB.

Una vez realizado este análisis inicial, se escoge las herramientas para el desarrollo de los microservicios, tomando en consideración las características del monolito, así como la disponibilidad de las mismas. De la misma forma se escoge la metodología de desarrollo para la implementación, en concordancia con las metodologías utilizadas en la Dirección de Gestión de la Información y Procesos (DGIP) donde SISLAB es parte de su gama de sistemas en producción. El proyecto tendrá como guía la Metodología de Desarrollo Ágil Scrum, mediante la cual se definirá una adecuada planificación y asignación de tareas para el equipo de trabajo. Dentro de la planificación se puede evidenciar cada etapa del desarrollo, así como también los problemas y dificultades que se presentaron dentro del mismo.

Finalmente se muestra a detalle la arquitectura resultante de la implementación realizada, además se lleva a cabo un proceso de validación basado en el checklist de Susan Fowler, para comprobar que la nueva arquitectura y el sistema migrado solucionan los problemas planteados inicialmente.

CAPITULO 1 – MARCO TEÓRICO

1.1. Arquitectura de Microservicios

"El término Arquitectura de Microservicios ha surgido en los últimos años para describir una forma particular de diseñar aplicaciones de software como conjuntos de servicios desplegables de forma independiente. Si bien no existe una definición precisa de este estilo arquitectónico, existen ciertas características comunes en torno a la organización y la capacidad empresarial, la implementación automatizada, la inteligencia en los puntos finales y el control descentralizado de lenguajes y datos" [2], es la definición de Martin Fowler en su página web y una representación gráfica de su forma de escalar se describe en la Fig. 1.

.

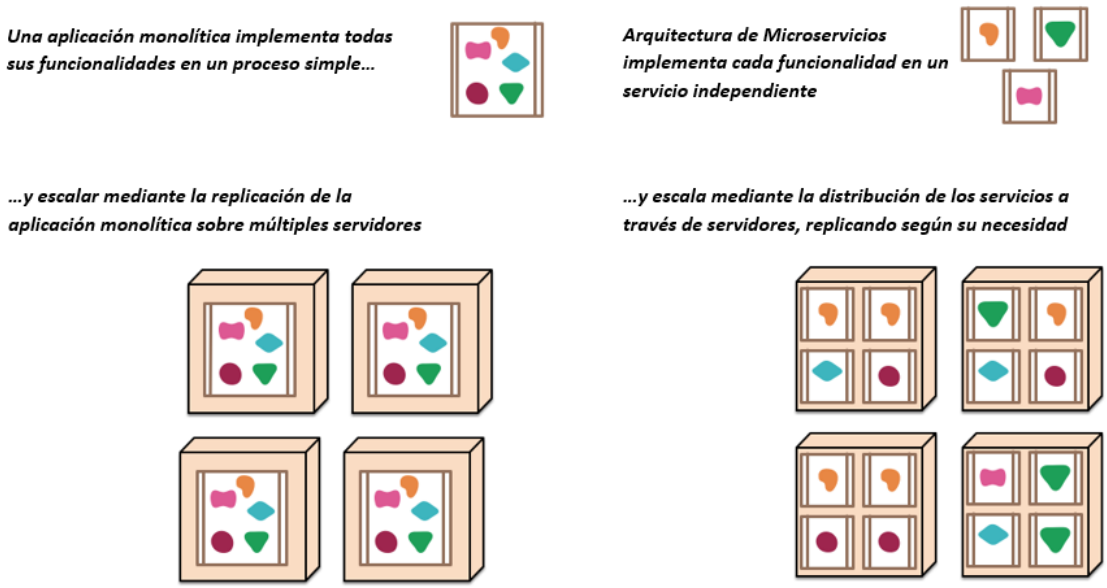


Fig. 1: Monolítico y Microservicios [2]

Existen más conceptos para describir microservicios, Sam Newman consultor independiente en su libro *Building Microservices* los describe como servicios autónomos pequeños construidos con los siguientes principios: modelo entorno a conceptos de negocio, adoptan una cultura de automatización, detalles de implementación interna ocultos, componentes descentralizados, aislamiento de fallos; servicios desarrollados: con despliegue independiente y altamente observable [3].

Amazon Web Services es una plataforma que ofrece distintos tipos de servicios en la nube, entre ellos una plataforma para los componentes de una arquitectura de Microservicios, en su página oficial define el siguiente concepto:

"Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes. Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más

rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características" [4].

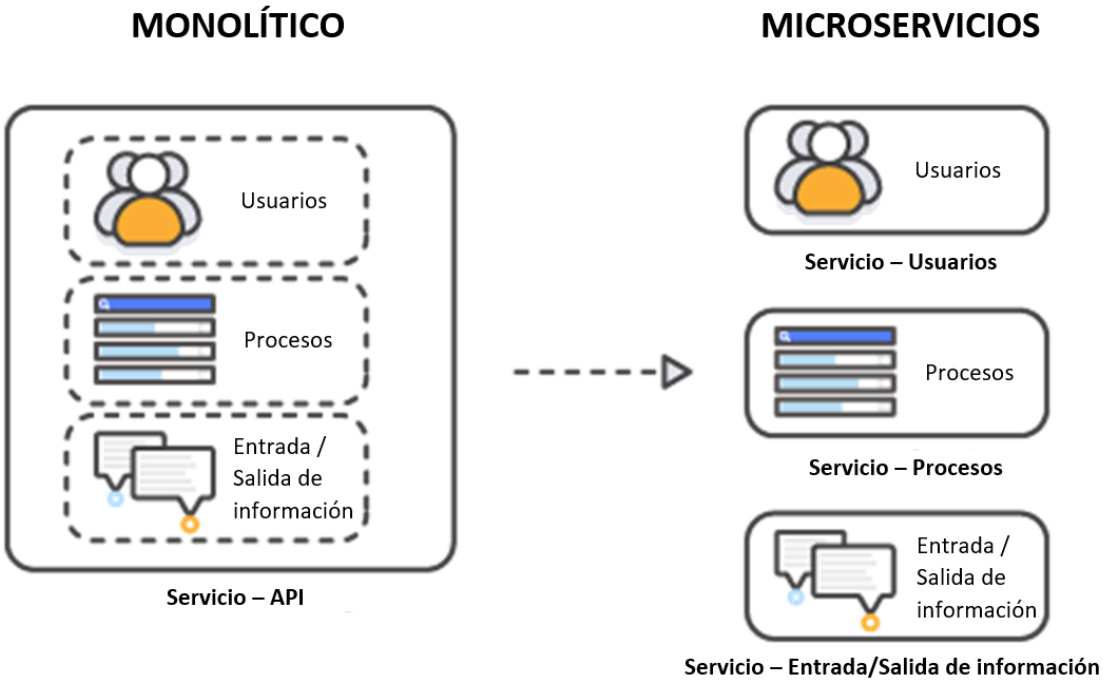


Fig. 2: Ejemplo de Monolítico a Microservicios [4]

La arquitectura de microservicios está conformada por un grupo de servicios autónomos e independientes, encargados de una funcionalidad de negocio en particular. Un microservicio puede ser implementado y desarrollado de manera independiente de esta manera puede ser también actualizado en la etapa de producción sin la necesidad de generar e instalar una aplicación entera. La programación y estructura interna de un microservicio es completamente transparente para los otros microservicios y el sistema o componente encargado de su orquestación, de igual manera la tecnología, arquitectura interna, lenguaje de programación, etc., pueden ser totalmente diferentes entre cada microservicio.

1.1.1. Componentes en Arquitectura de Microservicios

En la Arquitectura de Microservicios los componentes están definidos por el sistema en desarrollo y sus requerimientos, su implementación no es totalmente requerida sin embargo para poder cumplir con la orquestación y comunicación de los microservicios y componentes del sistema, existen algunos requeridos, se los detalla a continuación:

- *Discovery Server (Servidor de descubrimiento de servicios)*

El componente está encargado del registro y direccionamiento de las API expuestas por los microservicios a ser consumidos, tiene conocimiento de las instancias de microservicios disponibles en tiempo de ejecución del sistema. En caso de existir dependencias de ejecución

entre microservicios puede tener la responsabilidad de su comunicación, esto porque la comunicación puede ser asignada en otro nivel o componente dentro del ecosistema de microservicios [5].

- *Dashboard (Tablero)*

En la Arquitectura de Microservicios el ecosistema está constituido de varios componentes que necesitan trabajar coordinados, previamente monitoreados para conocer su estado y funcionalidad. El componente Dashboard tiene un fácil acople con el Discovery Server en la arquitectura, además puede ser accedido y entendido por personas que no sean específicamente desarrolladores [5].

- *Load Balancing (Balanceador de carga)*

En aplicaciones con distintas arquitecturas trabaja asignando las solicitudes a las instancias de los servidores disponibles para mantener un equilibrio en el consumo de los servicios, en la Arquitectura de Microservicios su trabajo se ejecuta específicamente con solicitudes entre el API Gateway y las instancias de cada microservicio. Los microservicios tienen asignado diferentes funcionalidades de negocio que varían en la cantidad de peticiones por parte del cliente y su trabajo es individual, el balanceador de carga permite que la cantidad de instancias de los microservicios no se vean afectadas entre sí [5].

- *Circuit Breaker (Cortacircuitos)*

La arquitectura de Microservicios es dependiente de una orquestación de todos sus componentes, esta es propensa a fallos por distintos motivos y si el error es con un microservicio el componente cortacircuitos tiene la responsabilidad de evitar que el error escale a un mayor nivel en otros componentes, mediante el impedimento de envíos de solicitudes al microservicio con error. Para este componente los microservicios se manejan con los siguientes estados:

  - Cerrado: Ocurre cuando el micro servicio funciona en total normalidad.
  - Abierto: Ocurre al momento de presentarse algún tipo de fallo.
  - Semi Abierto: Ocurre cada determinado tiempo para determinar si el servicio ya se encuentra disponible para habilitarlo o sigue caído para mantenerlo aislado [6].

- *Edge Server (Servidor Perimetral – API Gateway)*

Este servicio de borde o perimetral en la mayoría de casos es un API Gateway en el que se exponen los servicios a consumir. Su principal función es convertirse en un punto de entrada único que permita manejar las solicitudes de los clientes, independientemente del medio por el cual accedan (aplicación móvil o aplicación web), o inclusive, aunque sea más complejo se puede crear varios API dependiendo los requerimientos de cada grupo de consumidores. De esta forma

aísla a los clientes de cómo se divide la aplicación en microservicios y todo el tráfico del exterior es encaminado a los servicios internos correspondientes [7].

- *Logging (Servicio de monitorización)*

En este punto tratamos la monitorización, y como uno de sus ejes principales un registro de logs de toda la información relevante, lo que permite a los desarrolladores comprender el estado del microservicio con un histórico de información para poder prevenir problemas futuros. Una vez que ha madurado nuestro registro de logs se puede proponer un componente adicional que es el uso de *Alertas* para notificar cuando un microservicio está teniendo una conducta diferente a la normal. Por lo general este tipo de alertas se las programa en base a un registro del comportamiento que ha tenido un servicio ya sea en estado normal, peligroso o crítico, y que posteriormente nos permitirá establecer reglas de comportamiento para identificar daños mínimos que puedan evolucionar en catástrofes [7].

1.2. Desarrollo de Arquitectura Monolítica

La implementación de una aplicación con Arquitectura Monolítica tiene como principal ventaja un desarrollo menos complejo, porque no existe necesidad de orquestar una comunicación interna, los módulos en una aplicación o tablas en de una base datos pueden o no tener dependencia entre sí.

El escalamiento representa una menor complejidad pues si se tienen tiempos de respuesta lentos tiene como soluciones: optimizar los algoritmos que se usen dentro de la aplicación o aumentar la capacidad de hardware del equipo sobre el cual está corriendo el servidor desplegado, en ocasiones el nivel de complejidad alto limita el poder experimentar nuevas opciones de implementación durante la etapa de producción.

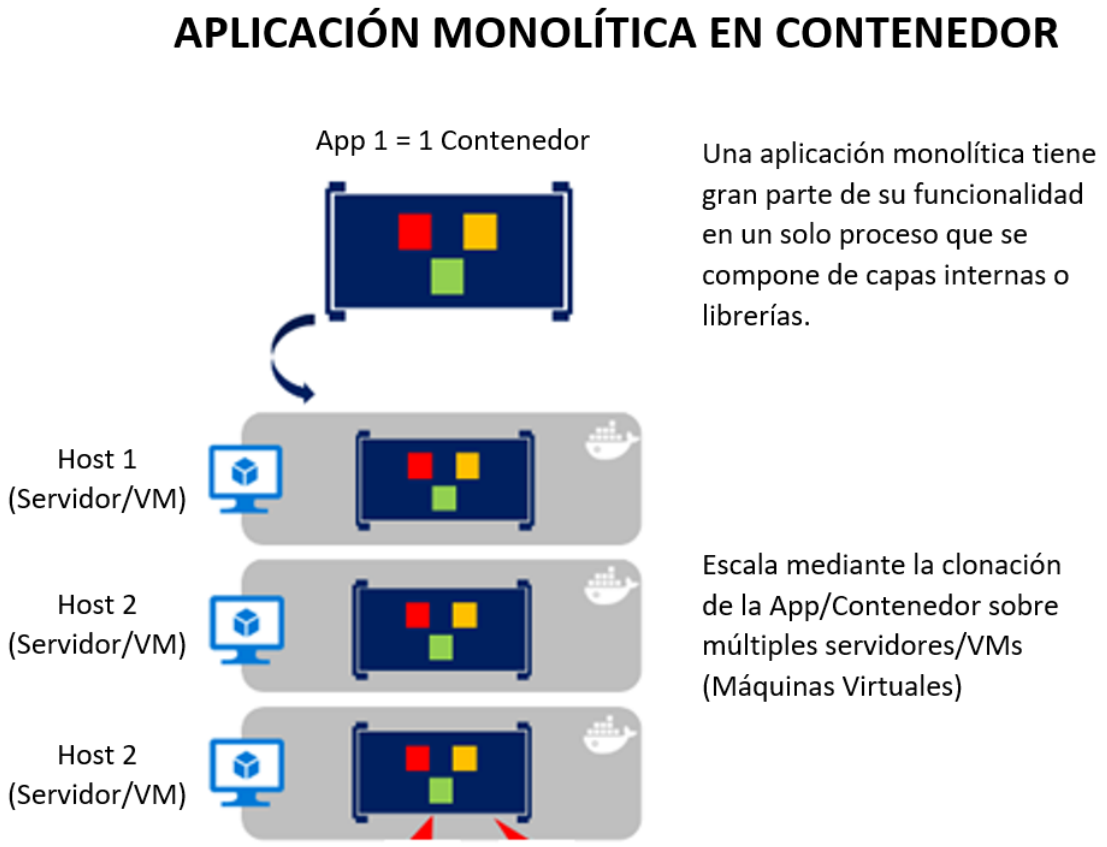


Fig. 3: Sistema Monolítico en Producción [8]

1.2.1. Desarrollo de Arquitectura de Microservicios

Dado que cada microservicio tiene una lógica de negocio exacta, el desarrollo representa menor complejidad pues se lo divide en proyectos menos extensos con menos requerimientos, el mantenimiento de cada microservicio de igual forma es más simple porque al ser un proyecto más pequeño con una funcionalidad bien definida un desarrollador puede tener un mejor entendimiento del código.

Los microservicios son de multi lenguaje, se puede tener una diversidad tecnológica debido a los proyectos pueden tener cada uno su lenguaje de programación y pueden estar acoplados. El desarrollo en paralelo también representa una ventaja al tener cada proyecto sus propios requerimientos y lógica definida.

Una desventaja está representada por el alto nivel de complejidad en la configuración, dentro del ecosistema de microservicios debe haber comunicación en caso de que existan dependencias entre los mismos y debe existir un proyecto mediante el cual se pueda conocer su estado en producción.

La creación de un proyecto Gateway que funcione como proxy entre la petición del cliente y el microservicio que procesa dicha petición, es incluida como un tema más de discusión en el momento de desarrollo de la arquitectura de microservicios. En el ámbito de la seguridad ya no se tiene que proteger solo un sistema sino se deben proteger cada uno de los microservicios

incluido la comunicación entre los mismos. La gestión de errores se extiende para cada proyecto, su comunicación y la conexión entre el Gateway que orquesta las peticiones. Para el desarrollo del Gateway se debe controlar que el enrutamiento no debe tener ningún tipo de conocimiento sobre la lógica de negocio de cada microservicio [9].

Los errores deben ser manejados de tal manera que si existe un error dentro de un microservicio no se reproduzca en cascada, esto se lo puede realizar mediante estrategias de resistencia, en el caso de existir un error únicamente la funcionalidad del microservicio se ve afectada sin bloquear toda la aplicación.

CAPITULO 2 – ESTADO DEL ARTE

El desarrollo de sistemas con Arquitectura de Microservicios se ha implementado en aplicaciones que se encuentran en producción, tomando en cuenta que para la migración de arquitectura se pueden tener o no nuevos requerimientos funcionales. Dwy Bagus Cahyono et al. [10], describen un ejemplo sobre un sistema para el registro y manipulación de la actividad estadística a nivel empresarial, los autores realizan un estudio inicial de los servicios expuestos por la aplicación antes de la migración, y analizan las diferencias entre arquitectura SOA y microservicios, posteriormente describen la metodología que van a utilizar. Mediante BPMs se describen los servicios a ser expuestos por el sistema final con arquitectura de Microservicios. En el desarrollo de sistemas con Microservicios existen temas de discusión para su desarrollo, uno de ellos es el versionamiento de código y APIs publicadas, Akhan Akbulut y Harry G. Perros [11] proponen el control de las versiones publicadas de un sistema a través del patrón API Gateway. Las diferentes versiones de los microservicios son publicadas y expuestas por el componente Gateway y mediante su configuración en ambiente de desarrollo o para la ejecución de pruebas las peticiones son redireccionadas a la versión indicada de cada microservicio.

La seguridad en ecosistemas de microservicios es tratada por Tetiana Yarygina y Anya Helene Bagge [3] en su publicación, ellos describen que existen dos posibles ideas: a) Solución multicapas, b) Una solución más robusta y menos compleja la cual afectaría el acoplamiento, aislamiento y tiempos de respuesta en la arquitectura. Para abordar la falta de pautas de seguridad, en el estudio se describe la implementación y el diseño de un marco de seguridad simple para el desarrollo de microservicios, realiza un estudio de problemas de seguridad en las distintas capas de la arquitectura de microservicios: hardware, virtualización, procesos a nivel de nube, comunicación, aplicación, orquestación. La seguridad perimetral toma un nuevo significado en la propuesta (Ver Fig. 4), proponiendo una defensa en profundidad y excluyendo a la infraestructura, API Gateway y servicio de monitoreo. La propuesta desarrolla un framework con el nombre de MiSSFire, mediante el uso de varias herramientas entre ellas Tokens.

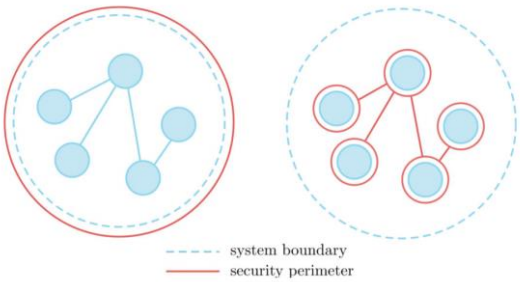


Fig. 4: Redefinición de perímetro de seguridad en Microservicios [3]

El desarrollo de aplicaciones con Arquitectura de Microservicios se ha llevado a cabo también en sistemas aproximadamente nuevos, porque su idea de negocio no es nueva pero el sistema con requerimientos funcionales se ha empezado a desarrollar desde cero. En el documento de Ratthida Wongsakthaworn y Yachai Limpiyakorn [12] se detalla el desarrollo de una aplicación de mesa de ayuda para TI, existe un capítulo de implementación en el cual se detallan las funciones de los 4 microservicios del diseño de arquitectura del sistema.

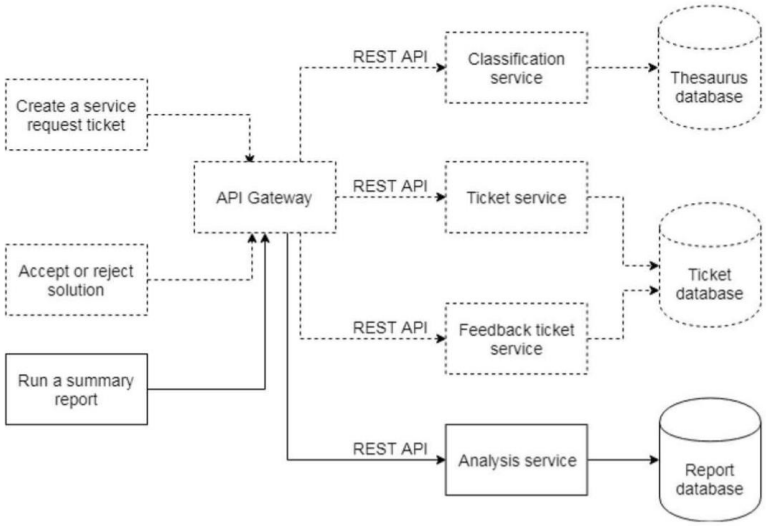


Fig. 5: Diseño de Microservicios para IT Helpdesk [12]

La publicación no especifica las herramientas de desarrollo utilizadas, pero en la descripción de cada microservicio demuestra mediante imágenes las ejecuciones de los mismos y al final del documento una comparación de los tiempos esperados con los tiempos obtenidos.

Santonu Sarkar et al. [13], estudia la migración de arquitectura de un sistema de automatización industrial, la versión “as-is” maneja un sistema distribuido mediante contenedores de sus módulos como se muestra en Fig. 6, el estudio detalla los cambios concretos para migrar a la arquitectura de Microservicios.

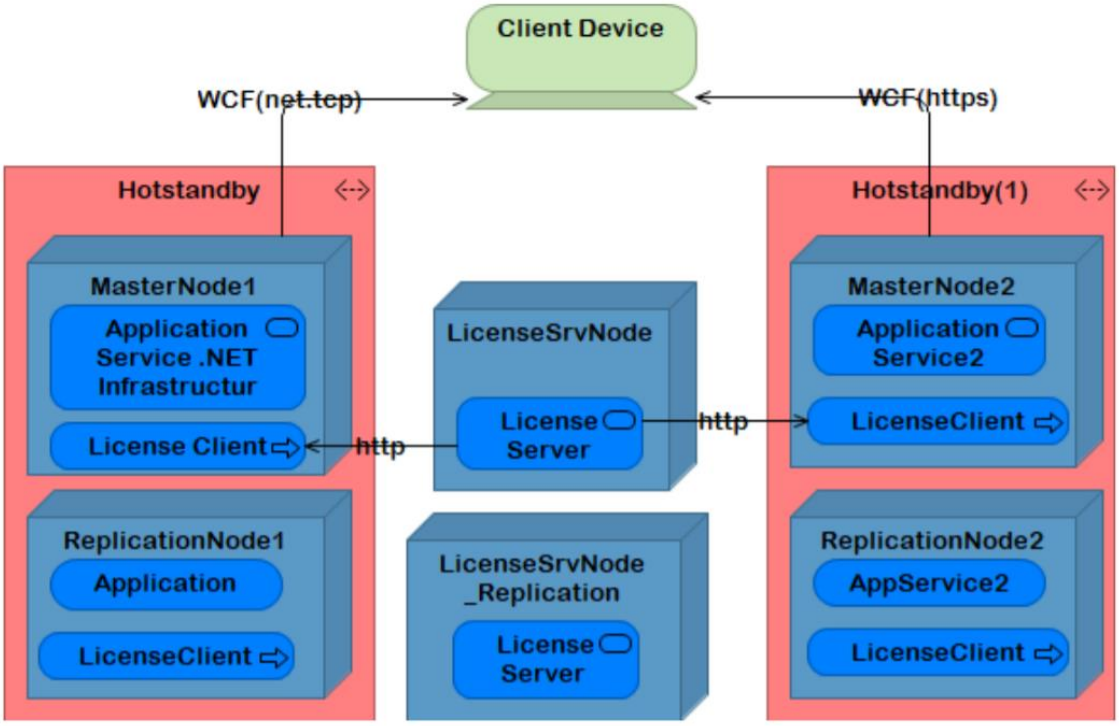


Fig. 6: Arquitectura AS-IS desplegada [13]

La investigación detalla el desarrollo de un microservicio por cada módulo del sistema "as-is", y se planifica que los microservicios sean instalados sobre el mismo equipo hardware, dado que los módulos actualmente trabajan sobre equipos hardware independientes; el desarrollo no es completo porque únicamente se detalla el estudio del sistema "as-is" y si modela un diseño de arquitectura con microservicios, se describen conclusiones con tiempos de ejecución medidos con un microservicio implementado.

Debido a la gran complejidad lógica de grandes aplicaciones tradicionales con una estructura monolítica, dependencias entre librerías de diferentes frameworks, la migración de un sistema monolítico implica muchos retos y dificultades a los cuales enfrentar, debido principalmente a que la migración de monolitos existentes aún se encuentra en una etapa temprana, por lo que casos de éxito o fracaso no son muy abundantes. Dentro del contexto actual las metodologías ágiles son cada vez más utilizadas como metodologías de desarrollo, principalmente para satisfacer necesidades en cuanto al cambio continuo, retroalimentación con el cliente además de entregables funcionales de manera continua. Davide Taibi, Valentina Lenarduzzi, Claus Pahl, Andrea Janes [14], realizan un estudio en este contexto, en el cual se identifican los problemas, ventajas y desventajas en la migración de aplicaciones monolíticas a microservicios basado en un contexto ágil. La información recopilada para el estudio se lo realizó en un taller celebrado durante la Conferencia Internacional de 2017 sobre Procesos Ágiles en Ingeniería de Software y (XP) Programación Extrema. Como resultado del estudio, el principal problema identificado es la descomposición del monolito, la cual es una tarea compleja, principalmente el entendimiento de cuál es la estrategia de descomposición más adecuada. A continuación, se detallan los principales hallazgos de este estudio:

Problemas:

- Descomposición del monolito
- Continuos Cambios en la Arquitectura

- Monitoreo
- Versionamiento
- Administración del Estado

Principales Ventajas:

- Escalabilidad
- Límites claros, desde el punto de vista del servicio como del equipo de desarrollo.
- Despliegue independiente.
- Entregas continuas, y rápida retroalimentación del cliente.

Principales Desventajas:

- Mayor complejidad.
- Se necesita un equipo de trabajo más experimentado.
- Mayor dificultad de aprendizaje.

El éxito de una migración de un sistema monolítico depende de varios factores, uno de ellos la descomposición del monolito, lo que es una de las principales dificultades al migrar a una arquitectura de microservicios. Para afrontar este importante factor, se han realizado varios estudios en cuanto a la extracción de microservicios, estos estudios pretenden apoyar la toma de decisiones por parte del equipo de desarrollo, a la hora de extraer microservicios candidatos. Se han realizado diversos estudios utilizando distintos enfoques, uno de ellos a través de técnicas para extraer microservicios de forma automática. En la publicación de Sinan Eski y Feza Buzluca [15], se propone un nuevo enfoque basado en el análisis de repositorios de código. En este enfoque se analiza el código estático y repositorios de software para extraer relaciones y cambios en el sistema de software, aplicando técnicas de acoplamiento de grafos para identificar microservicios candidatos. Como casos de estudio fueron analizados dos proyectos java utilizando este enfoque: eQuality y Academics. En este análisis experimental, el enfoque propuesto en su publicación alcanza hasta el 89% de la tasa de éxito al comparar microservicios extraídos con resultados reales.

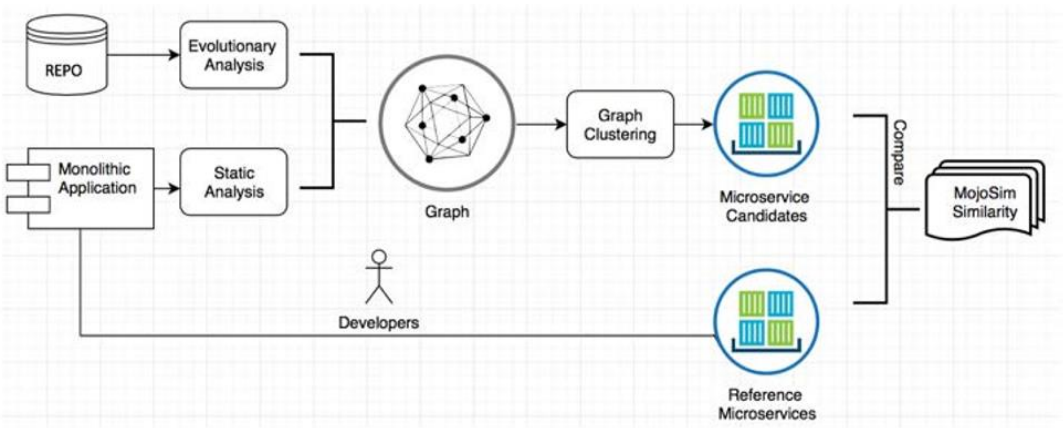


Fig. 7: Arquitectura del sistema de la Metodología de Extracción [15]

Otro enfoque metodológico para extraer microservicios es el presentado por Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, Tao Huang, en su publicación "Migrating Web Applications from Monolithic Structure to Microservices Architecture" [16], se resalta la importancia del comportamiento del monolito en tiempo de ejecución, combinando el análisis estático y dinámico del código de la aplicación monolítica. En este caso se emplea el acoplamiento entre funciones para evaluar el grado de dependencia entre ellas, y a través del agrupamiento de funciones para lograr migrar el monolito a la arquitectura de microservicios. Además de ello, al igual que en el enfoque presentado por Sinan Eski y Feza Buzluca, se realiza un análisis estático del código fuente, una vez obtenidas las características estáticas y dinámicas de la aplicación monolítica, se obtiene una colección de microservicios candidatos a través del agrupamiento de características.

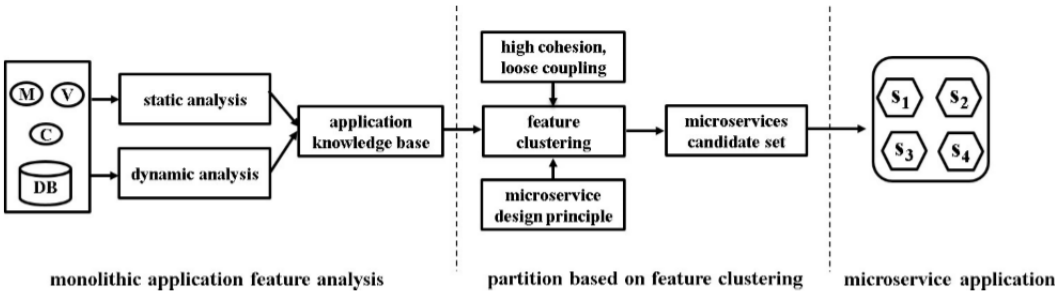


Fig. 8; Visión General del Enfoque [16]

Además de las técnicas de extracción automática de microservicios, y la investigación enfocada a los microservicios en los últimos años, aún existe poco conocimiento en cuanto a la extracción de microservicios en la práctica. Debido a ello estudios exploratorios con profesionales e investigadores con una importante expertica en cuanto a migración de sistemas monolíticos, han sido base para la extracción de conocimiento colectivo, basado en la experiencia de cada participante, revelando datos de gran importancia para una migración exitosa de un sistema monolítico a una arquitectura de microservicios. Varias entrevistas realizadas a estos especialistas han ayudado a encontrar criterios útiles para la migración en un caso práctico. En el análisis presentado por Luiz Carvalho, Alessandro García, Wesley K. G. Assunção, Rafael de Mello y Maria Julia de Lima [17], en su trabajo "*Análisis del Criterio Adoptado por la Industria para Extraer Microservicio"s*, presentan un caso real en el cual, la selección de un criterio particular, puede afectar las decisiones tomadas al momento de extraer microservicios de un sistema existente. Basado en la utilidad relativa de siete criterios posibles para apoyar la toma de decisiones en la extracción de microservicios, estos criterios se detallan en Fig. 9. Los resultados sugieren al menos cuatro criterios dominantes que son considerados para apoyar la toma de decisiones en la extracción de microservicios, cuestionando las soluciones académicas las cuales sugieren por lo general solamente dos criterios: acoplamiento y cohesión. Además de estos dos criterios, los cuales fueron considerados los más útiles, los requerimientos de la aplicación, tanto funcionales, como no funcionales, y la reutilización del código complementan la toma de decisiones y aportan un marco más amplio a la hora de extraer microservicios en la práctica.

Criterion	Responses															Median
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	
Coupling	3	3	3	4	4	4	4	5	4	4	5	5	4	5	4	4
Cohesion	5	4	5	4	4	5	4	5	2	4	5	5	5	3	3	4
Communication Overhead	2	2	1	4	3	4	2	3	4	3	5	1	4	3	2	3
Reuse Potential	4	4	4	3	5	2	4	5	1	2	1	4	2	2	5	4
Database Schema	4	4	1	3	2	4	5	3	2	1	5	5	1	4	3	3
Requirements Impact	4	4	4	3	3	5	5	3	4	1	4	5	5	4	4	4
Visual Models	3	2	5	3	2	2	4	2	1	5	3	5	3	5	3	3

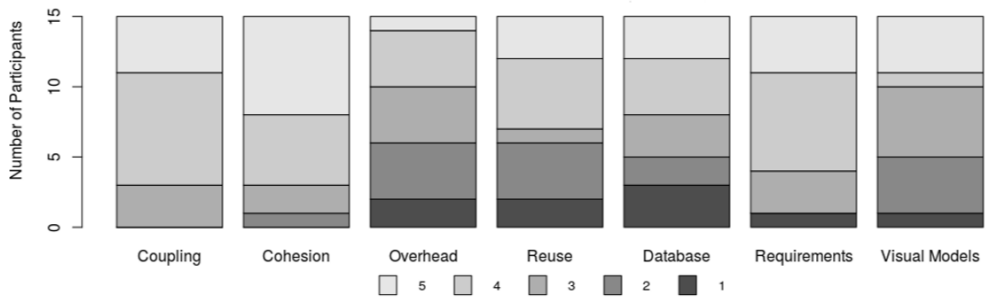


Fig. 9: Criterios Útiles: Distribución de Respuestas [17]

En un estudio posterior, la variabilidad se considera otro criterio de gran utilidad para la extracción de microservicios. Además, se evidencia que la extracción de microservicios puede incrementar la personalización del software [18].

El presente proyecto aborda la fase de desarrollo de una arquitectura de microservicios a partir de una propuesta definida, considerando una fase previa al desarrollo de análisis de la propuesta y llevar a cabo posibles cambios sobre la misma. La seguridad del sistema se manejará mediante la implementación de un microservicio entre las capas de Gateway y el ecosistema de microservicios funcionales del sistema. En los documentos de investigación encontrados no se detalla un proceso o un manejo sobre la arquitectura de monolito para ser migrada a una arquitectura de microservicios, la arquitectura propuesta en este proyecto fue obtenida mediante la aplicación de la propuesta metodológica de Carlos Arboleda [[1]], de igual manera no se encontró información sobre las tecnologías para el desarrollo de un sistema con arquitecta de microservicios, en el presente documentos se detalla un estudio de tecnologías disponibles y las utilizadas para el desarrollo en el capitulo 5.1. Herramientas de desarrollo.

CAPÍTULO 3 – ESTUDIO DE PROPUESTA METODÓLOGICA

En este capítulo se describe el proyecto de titulación realizado por Carlos Arboleda [[2]], quien desarrolla una propuesta metodológica de migración de monolitos hacia microservicios; este proyecto se constituyó en el documento base para este estudio. El desarrollo del presente proyecto tomará en cuenta las justificaciones redactadas en las etapas propuestas para decidir las herramientas a ser utilizadas y correcciones en la propuesta final a nivel de arquitectura y base de datos.

3.1. Análisis del procedimiento Arquitectura Monolítica a Microservicios

A medida que pasa el tiempo nuevas tecnologías y formas de construir software han ido emergiendo, de la mano con nuevos requerimientos y necesidades, por lo que es importante conocer cómo y cuándo utilizar una arquitectura de microservicios. El presente trabajo se basa en la propuesta metodológica desarrollada en el proyecto de titulación "*Propuesta metodológica para migración de sistemas web con arquitectura monolítica hacia una arquitectura basada en microservicios"*. En la propuesta presentada se realiza un análisis inicial basándose en la "Ley de Conway", en la cual se argumenta que el diseño de un sistema de software imitará la estructura de comunicación de la organización que la produjo.

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure". Melvyn Conway, 1967 [2]

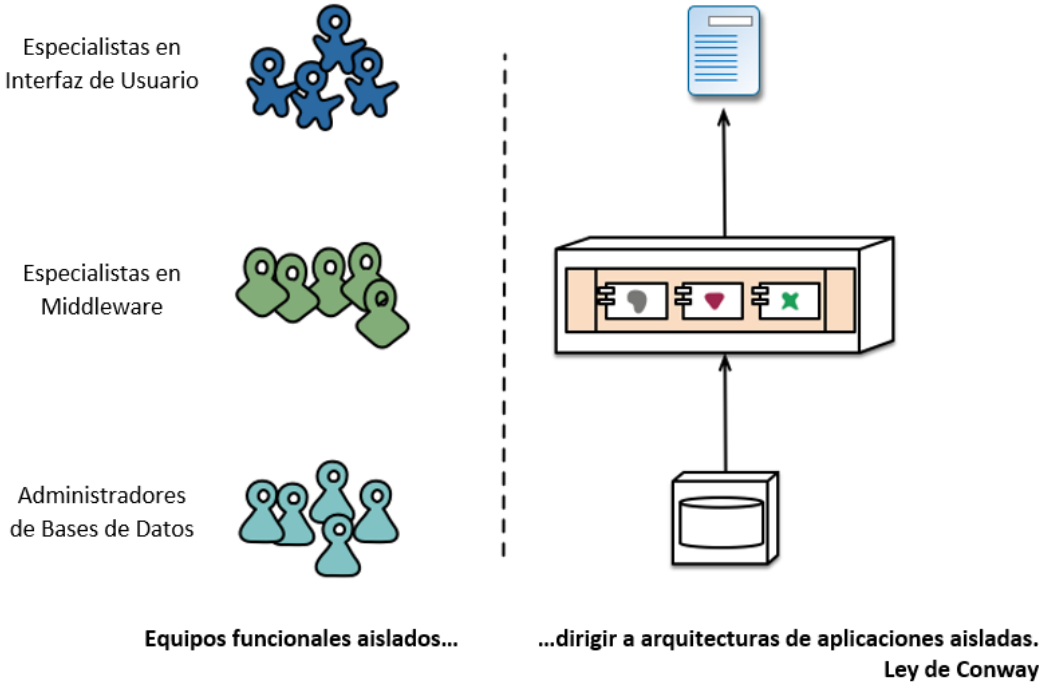


Fig. 10: Ley de Conway en Acción [2]

El enfoque en arquitectura de microservicios es diferente, donde cada equipo es organizado según la capacidad empresarial, por lo que cada equipo es dividido según una funcionalidad diferente, cada equipo es responsable de la construcción y operación de cada "producto", siendo responsables de la construcción del mismo en diferentes contextos.

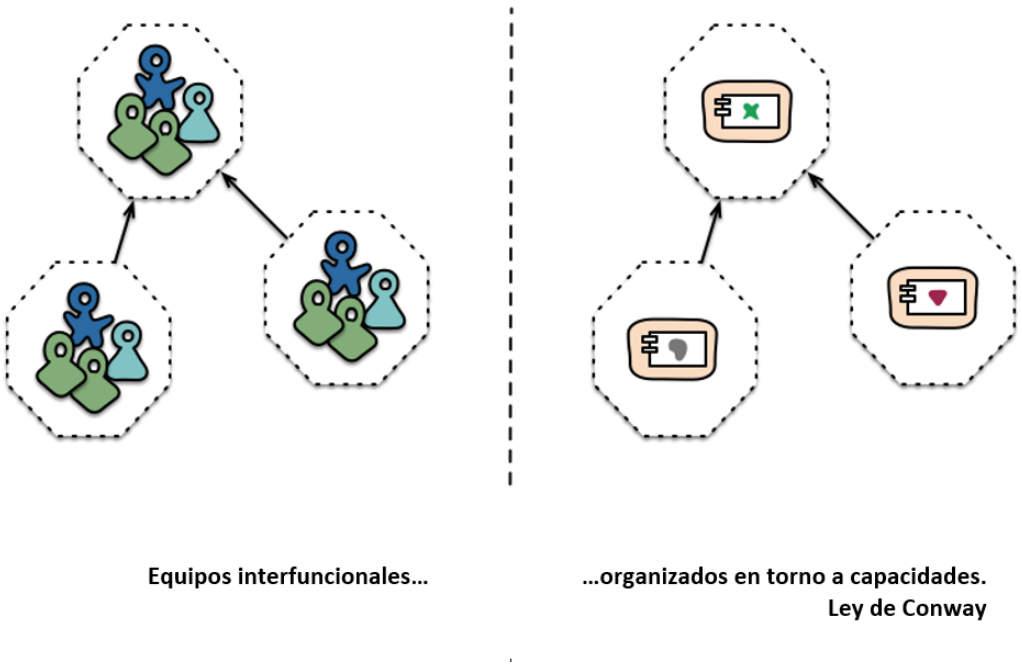


Fig. 11: Enfoque de microservicios [2]

Mediante el análisis realizado se propone la siguiente estrategia de transformación:

- Analizar el estado y documentación del sistema a migrar en relación con la organización.
- Identificar áreas problemáticas en su diseño organizacional.
- Identificar transformaciones seguras (cambios que no cambian el comportamiento existente) [20].

Grandes aplicativos monolíticos pueden siempre ser divididos en microservicios, sin embargo, la aplicación de este enfoque no es lo más común en la migración de un monolito, debido a que se necesita un equipo de trabajo grande para poder dividir el monolito entre cada línea de negocio. Sin embargo, monolitos desarrollados con enfoques tradicionales, generalmente utilizan su equipo de trabajo en varios contextos dentro del monolito, lo cual dificulta alinearlos en una solo línea de negocio.

La migración de un sistema monolítico a microservicios depende de varios factores: la estructura organizativa, la actualidad del sistema, si el sistema se encuentra en producción, nuevos requerimientos, problemas presentados por el sistema etc. Debido a estos factores no hay una sola forma para realizar este procedimiento, esto dependerá de las características propias de cada sistema, y la metodología a ser escogida, la que debería adaptarse a las necesidades, características del sistema y la organización propietaria del mismo.

3.1.1. Descomposición de sistemas de software monolíticos

Existen diferentes procedimientos para la descomposición de un monolito, siendo su principal dificultad, la complejidad de la aplicación monolítica a migrar, para ellos existen diferentes posibilidades las que se han explorado para la elaboración de la propuesta son las siguientes:

Fig. 12: Tipos de Descomposición [19]

Además de los procedimientos en Fig. 12, se puede tomar a consideración algunas recomendaciones de Zhamak Dehghani, consultor principal de tecnología en ThoughtWorks en su artículo, "*How to break a Monolith into Microservices*" [3]. Decidir qué capacidades desacoplar, cómo y dónde es uno de los mayores retos al descomponer un monolito a un ecosistema de microservicios, para ello se muestran algunas técnicas que serán de utilidad en el proceso.

1. Empezar con las características más simples y menos acopladas al monolito.
2. Minimizar las dependencias en el monolito.
3. Dividir las características con mayor número de dependencias de forma temprana.
4. Desacoplar verticalmente y liberar los datos temprano.
5. Desacoplar lo que es importante para la lógica del negocio, y las características con cambios más frecuentes.
6. Desacoplar capacidades y no código.
7. Macro primero, luego micro.
8. Migrar en pasos evolutivos pequeños.

3.1.2. Ecosistema de microservicios

El ecosistema de microservicios se refiere al entorno donde se construyen los microservicios. En la propuesta se divide al ecosistema de microservicios en cuatro capas como se muestra en la Tabla 1:

Layer 4: Microservices
Layer 3: Application Form
Layer 2: Communication
Layer 1: Hardware

Tabla 1: Modelo de cuatro capas de Ecosistema de Microservicios [21]

3.1.3. Delimitación del modelo conceptual

Basado en la información analizada y las características del monolito, se plantea las directrices a seguir, para la migración hacia microservicios. Estas directrices tienen como fin proporcionar una hoja de ruta para todo el proceso, mejorando la comprensión de todos los involucrados, en cada etapa. De esta manera se establece un modelo conceptual en cuatro etapas detallado en Fig. 13:

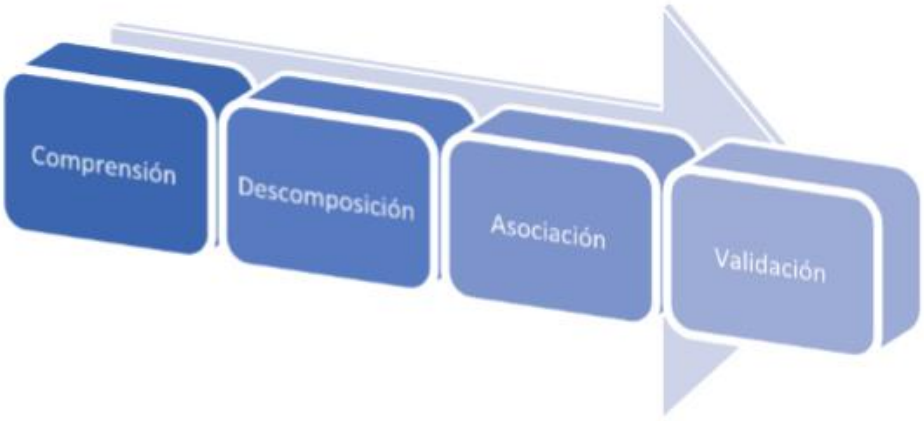


Fig. 13: Modelo Conceptual Migración Microservicios[22]

3.1.4. Descomposición

Culminada la etapa de comprensión del monolito, se procede a la descomposición del mismo, para ello ya habiendo hecho el análisis de las diferentes técnicas de descomposición, se escogerá la que más se adapte al aplicativo en cuanto a los módulos de software y a la base de datos. Para ello en la propuesta se utilizaron las siguientes herramientas, para facilitar el estudio de las operaciones que realiza el sistema:

- Software de Modelamiento de clases
- Modelo de Dominio
- Tabla con funciones del sistema

Una vez identificadas las funciones del sistema se agrupa las funcionalidades para posibles descomposiciones mediante:

- Lista de posibles microservicios.
- Diagrama de Modelo de Dominio Dividido.
- Diagrama de Base de Datos Descompuesto.
- Software de Modelamiento de Entidades de la Base de Datos.
- Tabla con entidades sensibles por número de relaciones.
- Query's sobre la Base de Datos.
- Reuniones sobre el funcionamiento con usuarios del sistema.
- Cuadro comparativo de lectura y escritura de entidades.
- Diagrama de Base de Datos Descompuesto.

Resultado del estudio y análisis realizado en 3.1.4. Descomposición, se agrupan los posibles microservicios y finalmente se realiza una comparativa para determinar cuál descomposición resulta en una menor dependencia entre los microservicios, ya que no se puede eliminar por completo la dependencia entre ellos, pero si minimizar esta dependencia. En la propuesta metodológica se plantea la utilización y realización de las siguientes pruebas y herramientas:

- Pruebas en ambiente de producción del sistema.
- Software de Monitorización.
- Matriz con llamadas de la capa de persistencia a la Base de Datos.
- Diagrama final de descomposición [23].

La descomposición del monolito a través de las funcionalidades del mismo, es adecuada ya que, al tratarse de una migración de un monolito construido con una metodología tradicional, los módulos que componen el sistema pueden ser agrupados según una funcionalidad específica, ayudando a tener una menor dependencia entre cada microservicio, además la complejidad de esta técnica es menor en comparación de las otras técnicas estudiadas.

3.1.5. Lineamientos de transición para una descomposición en Microservicios

La descomposición se llevará a cabo a través de actividades contenidas en etapas, las cuales se describen a continuación:

Etapa de comprensión y descomposición

En la etapa de comprensión se describen actividades de estudio del sistema a migrar, su lógica de negocio y estado actual entorno a sus módulos, base de datos y arquitectura. La descomposición a través de patrones se aplica para tener una propuesta final. Lineamientos, su justificación y las actividades a realizarse se describen en la Tabla 2.

	Lineamientos	Justificación	Actividades
Comprensión	Análisis del cambio, en base a lógica del negocio.	Constatar que la institución impulsará un cambio en la forma de estructurar software.	Comprender cambios con director o jefe de proyecto
			Analizar con equipo de desarrollo posibles implicaciones de migración.
	Estudio de los antecedentes del sistema a migrar.	Comprender el origen de creación del sistema a ser migrado para entender las implicaciones de su evolución en el tiempo.	Extraer información general del aplicativo.
			Extraer información de los Módulos Iniciales del aplicativo.
			Extraer información de la Base de Datos del aplicativo.
	Estudio del estado actual del sistema a migrar.	Identificar las deficiencias actuales del sistema, así como analizar que la migración de arquitectura creará un cambio significativo a futuro.	Extraer información de los Módulos Actuales del aplicativo.
			Extraer información de la Base de Datos del aplicativo.
			Extraer información de la Arquitectura del aplicativo.
Descomposición	Análisis de la información de las operaciones del sistema	Establecer un punto de referencia para la descomposición.	Crear un nuevo esquema que muestre las clases principales del aplicativo.
			Describir el comportamiento del sistema en base al Modelo de Dominio.
	Aplicación de patrones de descomposición en la capa de negocio del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema.	Dividir la capa de negocio en base al Modelo de Dominio.
			Definir la primera propuesta de descomposición de capa de negocio.
			Extraer la primera propuesta de descomposición de la base de datos.
	Aplicación de patrones de descomposición en la capa de datos del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema	Definir el número de relaciones padre e hija de tablas.
			Definir el tamaño y número de registros de las tablas
			Clasificar las tablas en base a lectura y escritura intensiva.
			Abstraer la segunda propuesta de descomposición de la Base de Datos.
	Identificación de una propuesta de descomposición final	Obtener un diagrama de descomposición que se acople con la lógica de negocio del sistema	Comparar la independencia entre propuestas de capa de negocio con descomposición en Base de Datos.
			Elegir mejor propuesta de descomposición entre capa de negocio y BD.

Tabla 2: Etapa Comprensión y Descomposición

Etapa de Asociación y Validación

En la etapa de asociación se trabaja en la estructura de los microservicios, componentes de despliegue y comunicación entre microservicios. Finalmente, una etapa de validación para evaluar los microservicios y el ecosistema que los contiene. Lineamientos, su justificación y las actividades a realizarse se describen en la Tabla 3.

	Lineamientos	Justificación	Actividades
Asociación	Selección de componentes para acoplamiento en capa cuatro de ecosistema de microservicios.	Estructurar los microservicios que en sí facilitaran el núcleo de la nueva arquitectura.	Analizar el acoplamiento y configuración de los microservicios previamente definidos.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa tres de ecosistema de microservicios.	Integrar los componentes que faciliten la configuración y monitorización de los microservicios.	Analizar el acoplamiento de los componentes para la plataforma de aplicación de microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa dos de ecosistema de microservicios.	Integrar los componentes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento de los componentes para la comunicación entre microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa uno de ecosistema de microservicios.	Integrar los componentes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento en cuanto a requerimientos de hardware requieren los microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
Validación	Evaluación de consideraciones para producción que posee un sistema de microservicios.	Validar que el aplicativo a migrar puede ser tratado como un sistema basado en microservicios.	Analizar los patrones para un despliegue de microservicios.
			Analizar las estrategias para un despliegue de microservicios.
	Evaluación de cumplir las características básicas de un sistema distribuido.	Comprobar que el sistema analizado puede cumplir los requisitos de un ecosistema de microservicios.	Analizar el sistema como un ecosistema de microservicios.

Tabla 3: Etapa Validación y Asociación

3.2. Análisis de propuesta metodológica Monolítica a Microservicios

El sistema SISLAB ha sufrido varios cambios desde su creación en el año 2010, donde el Centro de Investigación de Control Ambiental CICAM), requería inicialmente, de un sistema que permita automatizar los procesos que se llevaban a cabo en los laboratorios. El sistema cubriría las necesidades de las unidades de laboratorio existentes, en aquel momento (30 unidades), basándose en los requerimientos generales del CICAM [24]. Es así como se planteó el desarrollo de los siguientes módulos:

- Módulo Administrativo,
- Administración de Servicios,
- Administración de Análisis
- Facturación

Quedando abierta la posibilidad de futuras versiones e implementación de los módulos restantes: Administración de Usuarios, Control de Inventarios e Información General [24].

Debido a su desarrollo con un esquema tradicional monolítico, además de su crecimiento y cambio continuo, las necesidades y requisitos a satisfacer por parte del sistema, han cambiado en igual medida. Durante varios años el Sistema de Gestión centralizada de Laboratorios SISLAB ha tenido varios cambios, los cuales no han sido documentados, por lo que en 2017 se realiza

un estudio de la actualidad y estado del mismo, ya que es escogido como un estudio de caso en el trabajo de titulación: Propuesta Metodológica para Migración de Sistemas Web con Arquitectura Monolítica hacia una Arquitectura Basada en Microservicios.

El sistema fue considerado ideal para identificar su respectivo cambio, principalmente en cuanto a la arquitectura, dentro de una migración hacia Microservicios. Además de los nuevos requerimientos que surgieron debido a su naturaleza monolítica, las cuales principalmente son:

- Nuevas funcionalidades.
- Funcionalidades específicas demandadas por nuevos laboratorios.
- Limitaciones en cuanto a escalamiento.
- Se requiere alto nivel de estabilidad y resiliencia [25].

3.2.1. Descomposición de sistemas de software monolíticos

Para la migración de este sistema monolítico a microservicios, basado en el análisis realizado en la tesis anterior y la actualidad del sistema, se establece un modelo conceptual en 4 etapas:

1. Comprensión
2. Descomposición
3. Asociación
4. Validación

- *Etapa Uno – Comprensión*

En esta primera etapa se debe analizar cómo se encuentra estructurado el aplicativo antes de la migración, actualmente parte del SII (Sistema Integrado de la Información), se ha mantenido en funcionamiento con cambios en su estructura física y lógica, cambios los cuales no han sido documentados por lo que se hace un estudio actual del SISLAB para comprender la actualidad del sistema a migrar. Los cambios principales del SISLAB son:

Metodología de desarrollo

El sistema inició con la aplicación de la metodología de desarrollo RUP, en la actualidad el mantenimiento se lo trabaja con SCRUM como un marco de referencia dentro de la metodología ágil, mediante este se pueden trabajar correcciones y actualizaciones de sistemas de software.

Módulos del sistema

En su primera versión se contemplaron los siguientes módulos:

1. Gestión de Inventarios
2. Facturación
3. Gestión de Usuarios

4. Gestión de Servicios
5. Ordenes de Trabajo
6. Información Gerencial
7. Reporte

Hasta el año 2017 donde se realizó el estudio para la propuesta metodológica se encontraban en funcionamiento los siguientes módulos:

- Gestión de Inventarios
- Gestión de servicios
- Ordenes de trabajo
- Reportes
- Facturación [26]

Sin embargo, actualmente el módulo de Facturación fue separado del SISLAB, así mismo las respectivas tablas en el modelo de la base de datos. Además de estos cambios se han aislado las siguientes tablas:

1. pureza
2. control_existencia_metodo
3. norma
4. hidratación
5. tipo_justificacion
6. laboratorio_usuario
7. saldo_existencia

Esquema Actual SISLAB

El backup de la base de datos del sistema actual SisLab en producción generó el modelo entidad relación mostrado en Fig. 14.

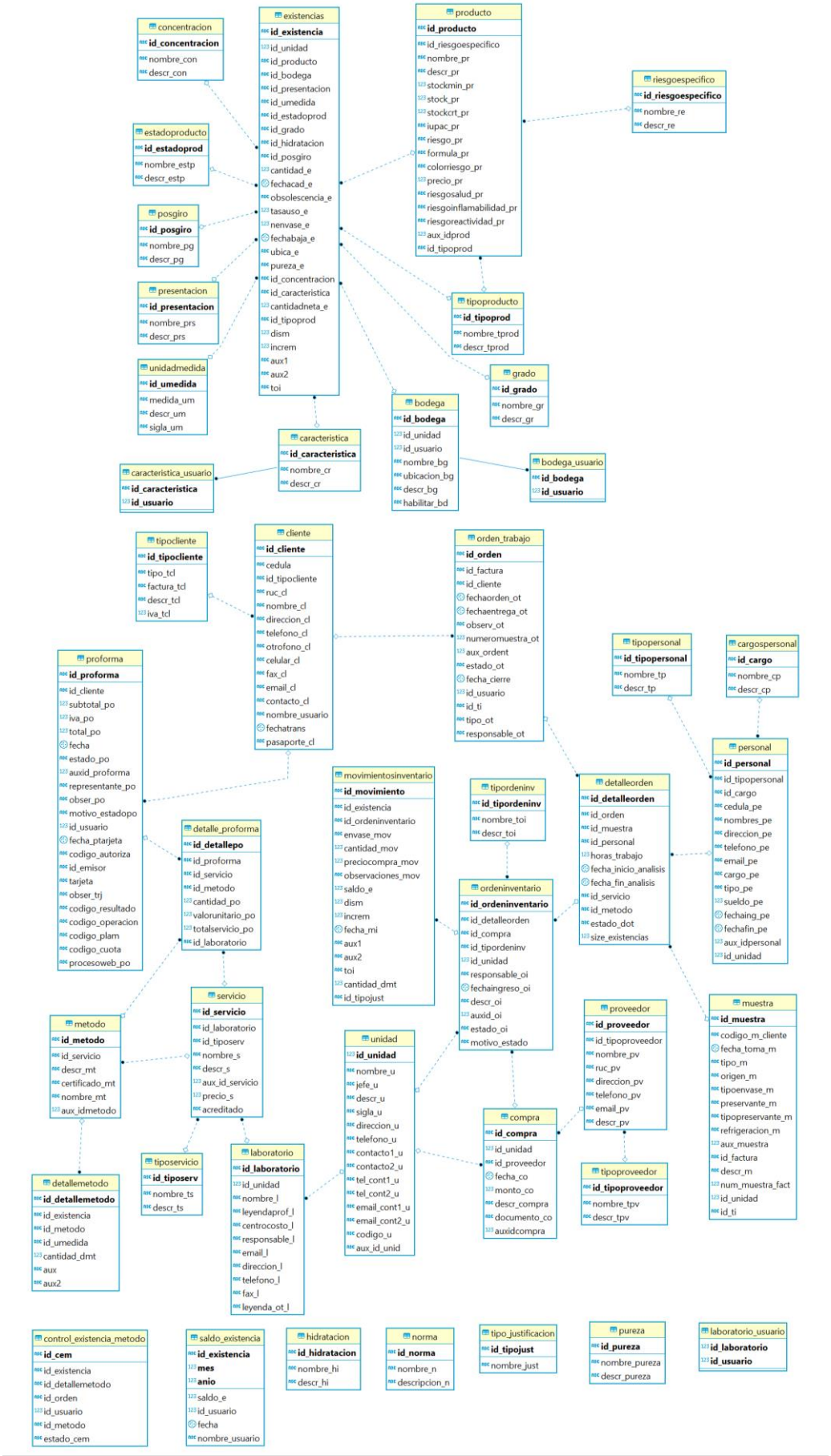


Fig. 14: Esquema Base de Datos SISLAB

- *Etapa Dos – Descomposición*

Una vez realizado el estudio de las técnicas de descomposición de monolitos, la descomposición por funcionalidad fue la que mejor se adecuó a las características del SISLAB, es así que a pesar de los cambios ocurridos en el sistema se puede mantener la misma estructura.

De acuerdo a las principales funcionalidades se extraen 4 microservicios:

- El primer microservicio corresponde a la primera funcionalidad, la cual es el eje principal del sistema, ya que aquí se genera las proformas, en base a una petición de un cliente.
- El segundo microservicio se orienta a las órdenes de trabajo.
- Tercer microservicio orientado a la gestión de productos.
- Cuarto microservicio exclusivo a la facturación del sistema.

Actualmente el módulo de facturación ha sido separado del SISLAB, sin embargo, al haber sido considerado como un microservicio independiente se puede seguir la misma alineación del resto de microservicios.

Una vez estudiado estos cambios se mantiene la descomposición de la propuesta original, excluyendo el microservicio de facturación (microservicio No. 4), manteniendo en este, las tablas aisladas en el monolito, pero que aún se encuentran en uso dentro del modelo de datos del SISLAB, para que en caso de poner en producción el sistema migrado, se analice la eliminación, modificación o mantenimiento de las mismas en un trabajo futuro [27].

- *Etapa Tres – Asociación*

Una vez culminada la etapa de descomposición del monolito, es necesario acoplar cada una de las partes dentro de las capas del ecosistema de microservicios. Cabe destacar, que varios de los elementos y herramientas usados para la asociación pueden variar según la aplicación y las necesidades y recursos de la empresa que desee implementar esta arquitectura. Partiendo de la capa cuatro, desde el núcleo de los microservicios ya separados, se debe acoplar con los elementos de la plataforma de aplicación, tanto en cuanto a comunicación, como a las funcionalidades requeridas para cada microservicio [23].

- *Etapa Cuatro – Validación*

La etapa validación, en nuestro caso se basará en el checklist de Susan Fowler, presentado en su libro “Production Ready Microservices”, cabe destacar, que se han seleccionado las más adecuadas para el alcance del presente proyecto, ya que en su libro se abarca hasta la etapa donde el sistema se encuentra en producción, debido a esto, se tomara únicamente el checklist ya que este abarca hasta cuando el aplicativo se encuentra en etapas de prueba del sistema. El cuestionario completo de la propuesta se puede observar en el anexo [28].

Estabilidad y Confiabilidad

1. ¿El microservicio tiene un repositorio central donde se almacena todo el código?

2. ¿El ecosistema de microservicio tiene una línea de implementación estandarizada?

3. ¿Qué acceso tiene el entorno de transición a los servicios de producción?

4. ¿Las implementaciones para la producción se hacen todas al mismo tiempo, o se implementan incrementalmente?

5. ¿Qué son estas dependencias de microservicios?

6. ¿Cómo este microservicio mitiga los fallos de dependencia?

7. ¿Hay copias de seguridad, alternativas, retrocesos o almacenamiento en caché defensivo para cada dependencia?

8. ¿Son confiables los controles de salud al microservicio?

9. ¿Hay interruptores en su lugar para evitar que el tráfico de producción se envíe a hosts y microservicios poco saludables?

9. ¿Existen procedimientos para depreciar los puntos finales API de microservicios?

10. ¿Existen procedimientos para depreciar los puntos finales API de microservicios?

Escalabilidad y Rendimiento

1. ¿Cuáles son los cuellos de botella de recursos de este microservicio?

2. ¿Escalarán las dependencias con el crecimiento esperado de estos microservicios?

3. ¿Se puede enrutar automáticamente el tráfico a otros centros de datos en caso de falla?

4. ¿El microservicio está escrito en un lenguaje de programación que permitirá que el servicio sea escalable y funcional?

5. ¿Hay alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio maneja las solicitudes?

6. ¿Existe alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio procesa las tareas?

7. ¿Este microservicio maneja los datos de una manera escalable y eficiente?

8. ¿Qué tipo de datos necesita cada microservicio para almacenar?

9. ¿Cuál es el esquema necesario para sus datos?

10. ¿Este microservicio necesita un mayor rendimiento de lectura o escritura?

11. ¿Es de lectura pesada, de escritura pesada o ambas cosas?

12. ¿La base de datos de este servicio se escala horizontal o verticalmente? ¿Es replicado o particionado?

13. ¿Es este microservicio usando un dedicado una base de datos compartida?

14. ¿El microservicio tiene un único punto de falla?

15. ¿Se han identificado todos los escenarios de falla y posibles catástrofes del microservicio?

16. ¿Cuáles son las fallas comunes en el ecosistema de los microservicios?

17. ¿Cuáles son los escenarios de falla de la capa de hardware que pueden afectar este microservicio?

18. ¿Qué fallas en la capa de comunicación y en la capa de aplicación pueden afectar este microservicio?

19. ¿Cómo impactan las fallas y las interrupciones de este microservicio en el negocio?

20. ¿Hay niveles de falla bien definidos? [28]

Resultado y discusión de la propuesta en tesis base

La arquitectura de microservicios es un término relativamente nuevo dentro de la arquitectura de software, por lo que no hay una definición precisa de lo que es esta arquitectura, sin embargo, hay ciertas características que generalmente son comunes en este tipo de arquitecturas:

- ✓ Diversidad de tecnologías.
- ✓ Implementación independiente
- ✓ Organizado en torno a las capacidades del negocio
- ✓ Gobierno descentralizado
- ✓ Gestión de datos descentralizada
- ✓ Automatización de la infraestructura
- ✓ Diseño a prueba de fallos
- ✓ Diseño evolutivo escalable.

De la misma forma no existe una sola técnica para migrar sistemas monolíticos, por lo que, un análisis a profundidad del monolito es fundamental, para la identificación de la metodología más apropiada.

Independientemente de la metodología que se elija, debe proporcionar las características anteriormente mencionadas, en el caso del SISLAB debido a sus características se propone la metodología dividida en cuatro etapas: Comprensión, Descomposición, Asociación y Validación. Cada una de estas etapas son completamente necesarias, ya que inicialmente se debe tener un dominio completo de la aplicación desde su creación en 2010, su metodología de desarrollo, los módulos implementados, sus cambios a través del tiempo, sus modelos de datos etc.

Una vez recopilada toda la información la etapa más compleja es la descomposición del monolito, tanto en la capa lógica como de datos. Múltiples técnicas hacen de esta etapa la más delicada, siendo la más apropiada para el SISLAB la descomposición por funcionalidad, minimizando la independencia entre cada microservicio tanto en lógica como datos. Además de la aplicación de este tipo de descomposición se debería tener en cuenta algunas consideraciones adicionales al separar un monolito.

La etapa de asociación depende principalmente a los recursos de la organización donde se seleccionarán varios de los componentes y herramientas propuestos, pero además se pueden añadir otros componentes según el contexto en el que se planea poner en producción el sistema.

Finalmente, la etapa de validación lo más apropiado es basarse en el checklist de Susan Fowler, tomando en cuenta solamente las preguntas según el alcance del presente proyecto, excluyendo las que tienen relación con el sistema ya puesto en producción. Adicionalmente se puede realizar una comparación rápida en cuanto al rendimiento del sistema migrado y el monolito, además de la verificación de cumplimiento de los nuevos requisitos planteados antes de la migración, además de los requisitos que ya cumplía el monolito [29].

CAPÍTULO 4 – ESTUDIO DE CASO

El capítulo detalla la aplicación de las etapas de la propuesta metodológica sobre un estudio de caso que proporcione información sobre funcionalidades, posibles correcciones en la fase de migración de la arquitectura y la fase de implementación.

4.1. Selección del Estudio de Caso

El estudio de caso escogido es el Sistema de Gestión Centralizada de Laboratorios (SISLAB), perteneciente al Sistema Integrado de Información (SII) de la Escuela Politécnica Nacional, el mismo que cuenta con las siguientes características:

- SISLAB se encuentra en un estado de cambio continuo, sin embargo, está limitado a una base de tecnología tradicional desarrollada en el año 2009 y por ende posee grandes limitaciones en cuanto a escalamiento se refiere.
- SISLAB es un sistema web que se encuentra en producción y funciona como servicio para un gran conjunto de laboratorios dentro de la EPN, sin embargo, existen otras entidades que demandan funcionalidades específicas que posee este aplicativo, pero por la intrínseca relación que posee como monolito no es posible extraer únicamente los servicios requeridos.

SISLAB requiere un nivel de estabilidad y resiliencia alto puesto que posee un conjunto de consumidores bastante amplio, por ende, debe estar activo la mayor cantidad de tiempo posible y para esto se necesita reforzar su estructura y por consiguiente su arquitectura [30].

4.1.1. ANTECEDENTES DEL SISLAB

La información mostrada a continuación corresponde al año 2009-2010, año en que fue planteado y tuvo inicio el desarrollo del SISLAB, esta información es un resumen con los aspectos más relevantes de la documentación oficial proporcionada por la DGIP.

a. Información General

El proyecto de desarrollo e implementación de un Sistema de Administración Centralizada de Laboratorios en la EPN permitirá llevar el control centralizado de la facturación y del inventario de los laboratorios que brindan servicio a la comunidad politécnica y al público en general; y será desarrollado por la Unidad de Gestión de Información.

b. Módulos

El proyecto pretende diseñar, construir e implementar un sistema de administración centralizado de laboratorios, el mismo que permita obtener información actualizada cuando se requiera. En el análisis realizado se han identificado los siguientes módulos:

1. Gestión de Inventarios,
2. Facturación,
3. Gestión de Usuarios,
4. Gestión de Servicios,
5. Ordenes de Trabajo,
6. Información Gerencial
7. Reportes

Sin embargo, en el acta de cierre del proyecto en su Versión 1, comprendía la entrega de los siguientes módulos:

1. Gestión de Inventarios
2. Gestión de Servicios
3. Ordenes de Trabajo
4. Reportes [31]

4.1.2. ESTADO ACTUAL DEL SISLAB

El sistema desde su creación ha tenido varios cambios es toda su estructura física y lógica. La mayoría de los cuales no han sido documentados, por lo cual se procede a obtener la siguiente información a través de un análisis del SISLAB actualizado al año 2017 [1].

a. Información General

El objetivo general del Sistema de Administración Centralizada de Laboratorios mantiene su meta de permitir llevar el control centralizado de la facturación y del inventario de los laboratorios que brindan servicio a la comunidad politécnica y al público en general; sin embargo, existen algunos cambios en la lógica de negocio puesto que se implementaron algunas adaptaciones debido al gran escalamiento de laboratorios que posee actualmente la EPN.

Metodología de Desarrollo

Si bien la metodología para el desarrollo del sistema fue RUP, en la actualidad dentro de la DGIP, se utiliza únicamente SCRUM como marco de referencia para desarrollo con metodología ágil, así como para correcciones y actualización de sistemas de software [1].

b. Módulos

Existen un aumento en el número de módulos funcionales que posee la aplicación, a la fecha de este análisis se encontraron en funcionamiento:

1. Gestión de Inventarios
2. Gestión de Servicios
3. Ordenes de Trabajo
4. Reportes
5. Facturación

Paquetes

En el proyecto se encontraron los siguientes paquetes

- Paquete Persistencia: Utilizado para el almacenamiento de información de las clases en las respectivas tablas de la base de datos.
- Paquete VO: Contiene los métodos que serán utilizados por cada clase para los módulos dentro del SISLAB.
- Paquete Recurso: Posee recursos adicionales que serán de utilidad para todo el sistema, como por ejemplo encriptación de información.
- Paquete Conexión: Utilizado exclusivamente para la conexión a la base de datos PostgreSQL del SISLAB.

- Paquete Reporte: Contiene la clase destinada a la configuración de parámetros adicionales para la producción adecuada de reportes.
- Paquete Servicios: Posee servicios consumidos para funcionalidades específicas como por ejemplo la consulta de datos del registro civil.

Cabe mencionar que al estar basado en el patrón Modelo-Vista-Controlador, los diferentes archivos de presentación (HTML, CSS, JS, etc.) se encuentran almacenados en otra sección diferente de los paquetes previamente mencionados.

A continuación, se muestra un diagrama de los paquetes encontrados en SISLAB, Fig. 15, con sus respectivas dependencias, teniendo en consideración que *ec.edu.epn.laboratorios.reporte* y *ec.edu.epn.laboratorios.servicios* no presentan dependencia con ningún otro paquete [1].

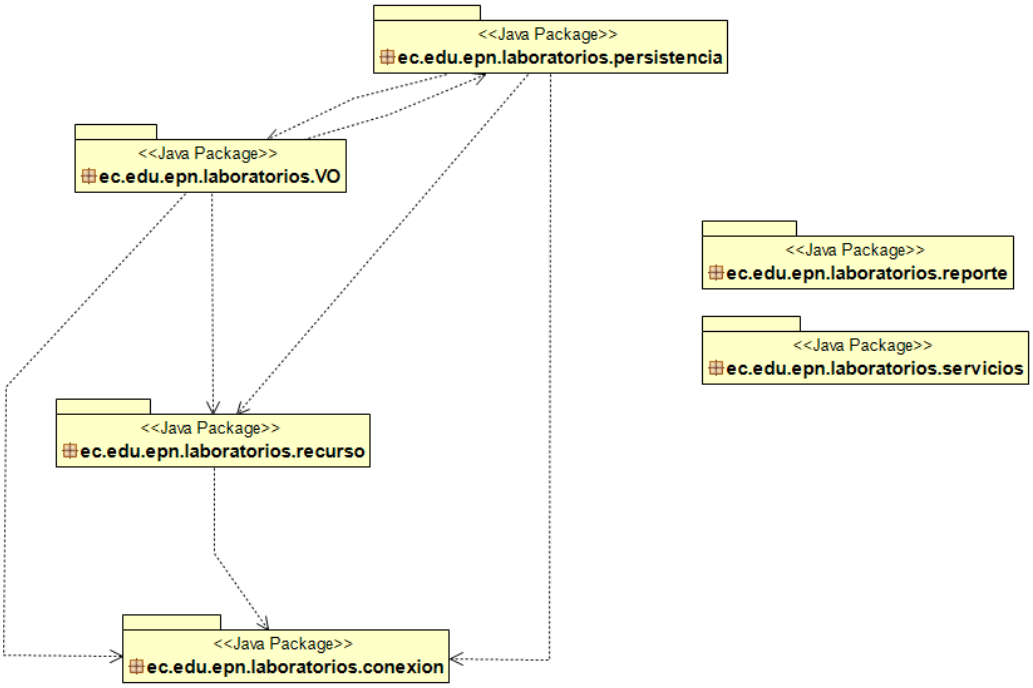


Fig. 15: Diagrama de Paquetes actuales en SISLAB [1]

Diagrama de Clases

- Paquete Recurso: Las clases encontradas dentro de este paquete fueron, Fig. 16:

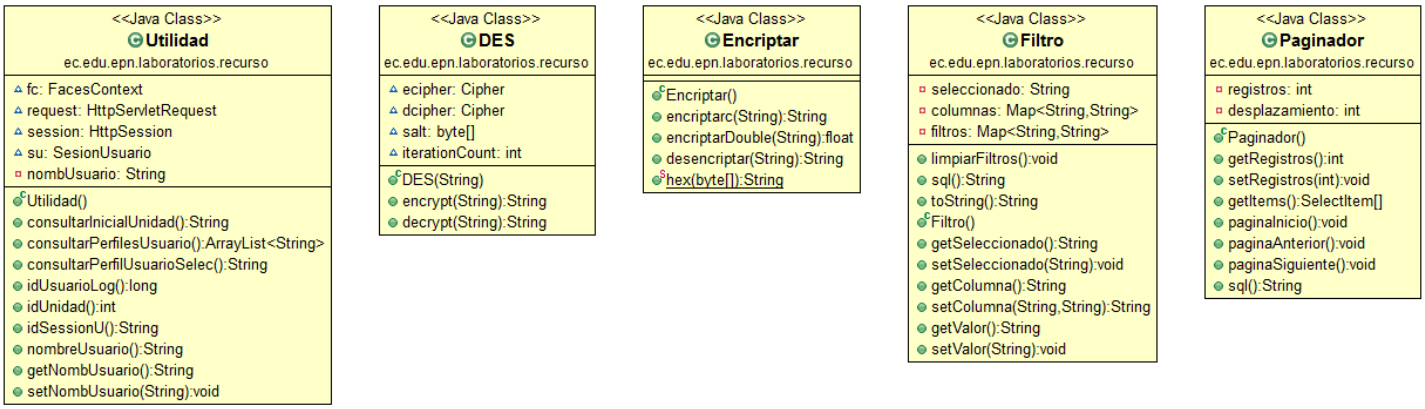


Fig. 16: Clases en Paquete Recurso de SISLAB [1]

- Paquete Conexión. - Las clases encontradas dentro de este paquete fueron, Fig. 17:

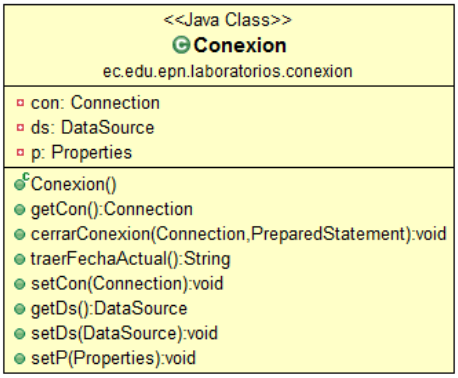


Fig. 17: Clases en Paquete Conexión de SISLAB [1]

- Paquete Persistencia: Se encontró un aumento significativo del número de clases con el que inicio el sistema en 2010, lo que implica que la base de datos de igual manera debió ser modificada, Fig. 18:

Fig. 18: Clases en Paquete Persistencia de SISLAB [1]

- Paquete VO. - Las clases encontradas dentro de este paquete fueron, Fig. 19:

Fig. 19: Clases en Paquete VO de SISLAB [1]

c. Base De Datos

La base de datos ha sufrido varios cambios considerables, entre los cuales destaca la creación de tablas adicionales y aislamiento de tablas ya existentes en otros esquemas. El número inicial de tablas en la creación del sistema fue de 26, a la fecha se encontraron un total de 44 tablas en el primer esquema analizado, puesto que ahora existe un esquema separado destinado exclusivamente a facturación. A continuación, se muestra un resumen del esquema *"bddcorpepn.Laboratorios"* con los respectivos comentarios realizados por los creadores de las tablas, Tabla 4.

TABLA	*N.º TABLAS HIJAS*	N.º TABLAS PADRE	COMENTARIOS
bodega	3	1	Bodegas de la EPN
bodega_usuario	0	1	
caracteristica	2	0	
caracteristica_usuario	0	1	
cargospersonal	1	0	Cargo que tiene un Personal
cliente	2	1	Clientes de la EPN
compra	1	2	
concentracion	1	0	
control_existencia_metodo	0	4	
detalle	0	10	
detalle_proforma	0	4	
detallemetodo	1	3	
detalleorden	1	3	
estadoproducto	1	0	
existencias	4	12	Existencias reales en inventario.
grado	1	0	
hidratacion	1	0	
laboratorio	2	1	Laboratorios pertenecientes a una Unidad
laboratorio_usuario	0	0	
metodo	4	1	
movimientosinventario	0	3	
muestra	2	1	
norma	0	0	
orden_trabajo	3	1	
ordeninventario	1	4	
personal	2	3	No existe relación de tipo Foreing Key entre personal - laboratorio y personal – unidad.
posgiro	1	0	
presentacion	1	0	
producto	1	2	
proforma	2	1	
proveedor	1	1	
pureza	0	0	
riesgoespecifico	1	0	
saldo_existencia	0	0	
servicio	3	2	

tipo_justificacion	2	0	
tipocliente	1	0	Tipos definidos para identificar a los Clientes
tipopersonal	1	0	Tipos definidos para clasificar a Personal
tipoproducto	2	0	Tipos definidos para identificar a los Productos
tipoproveedor	1	0	Tipo definido para clasificar a un proveedor
tipordeninv	1	0	
tiposervicio	1	0	
unidad	7	0	Unidad organizacional de la Politécnica
unidadmedida	3	0	

Tabla 4: Entidades en Base de Datos de SISLAB [1]

El modelamiento de la estructura de la base de datos actual mostrado en Fig. 20: Estructura de Base de Datos SISLABFig. 20:

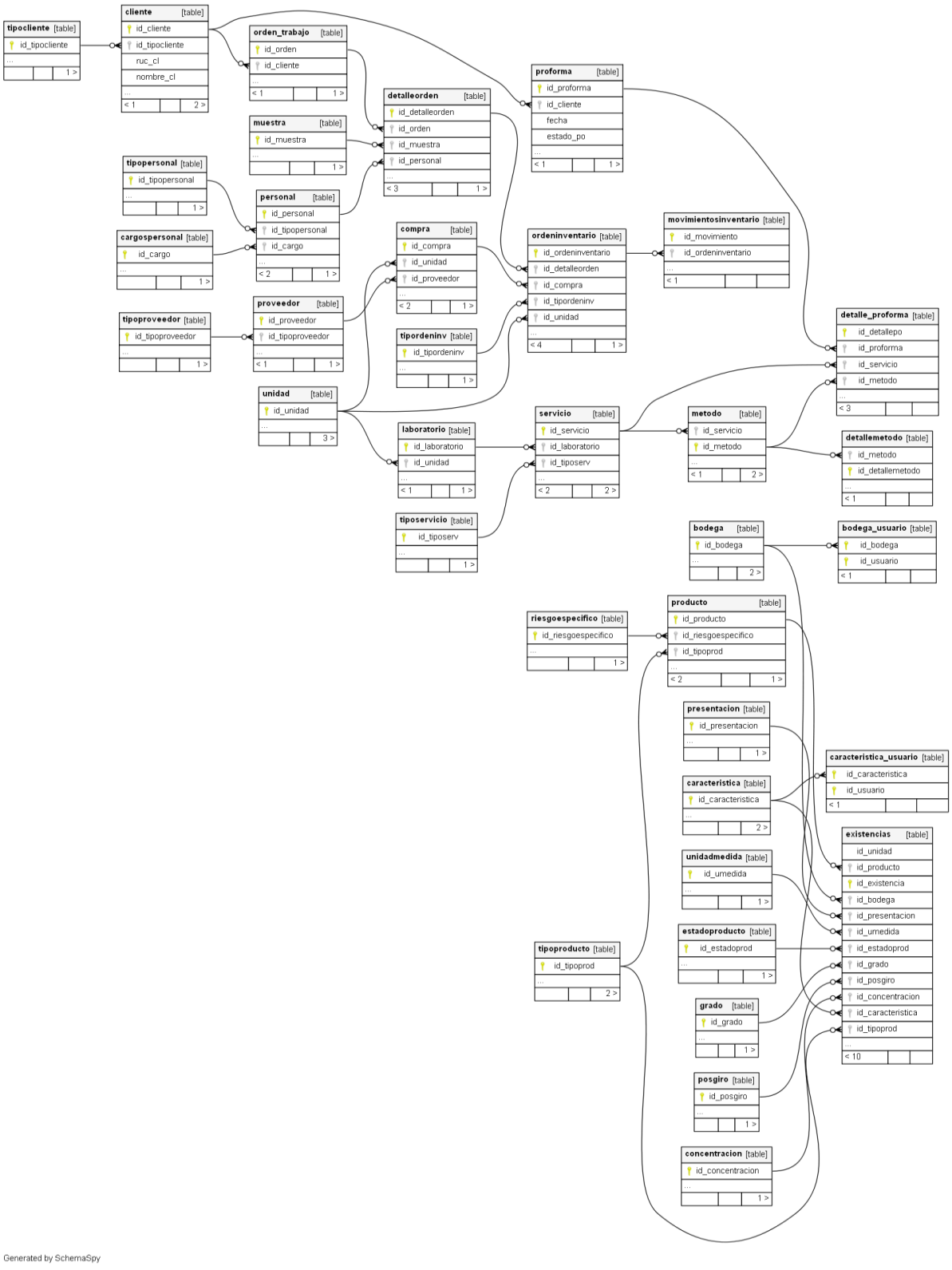


Fig. 20: Estructura de Base de Datos SISLAB [1]

4.1.3. PLATAFORMA MULTICAPA

La arquitectura física impuesta por el modelo de seguridad de red define un modelo de tres capas. Dicho modelo presenta además una distribución de componentes software,

distinguiéndose tres niveles lógicos con funciones distintas: uno para los servicios de usuario, otro para los servicios de la lógica de negocio (la lógica principal de la aplicación), y otro nivel para los servicios de datos [32].

- Presentación: En la capa de presentación se engloban al menos los contenidos estáticos necesarios para la aplicación (CSS, JavaScript, imágenes, etc.) que se desplegarán en el servidor web JBoss del contexto y entorno correspondiente.

- Negocio o aplicación: Como medio para la implementación y soporte de esta capa lógica, así como para el funcionamiento de toda esta arquitectura, estará el servidor de aplicaciones para tecnología Java. El servidor proporciona los servicios necesarios para gestionar el tratamiento de los componentes de negocio desarrollados, así como su control transaccional.

- Datos o Back-end: Se implementan estrategias de persistencia JPA para independizar esta capa propiamente del motor de base de datos escogido. La capa de datos estará en el servidor de Base de Datos PostgreSQL y se administrará utilizando la herramienta Pg Admin.

- Integración: Se añade una capa de integración como:

  a) Intermediario y agregador de funciones de interés general suministrados por distintos sistemas para su exposición y consumo.

  b) Suministrador de servicios con formato de servicio web.

  c) Ámbito de despliegue de procesos de negocio inter-sistemas (orquestación de servicios)

  d) Proveedor de adaptadores que faciliten el acceso a distintas tecnologías subyacentes.

  e) Propuesta de ámbitos de publicación y consumo de eventos con un único punto de entrada, pero con diversas tecnologías de publicación y recolección.

Todos estos componentes se distribuyen entonces en capas lógicas, pero con disposición en forma de servicios para aquellas funciones de interés general como una Arquitectura Orientada a Servicios (SOA) [32].

El sistema fue desarrollado como una aplicación web que permite a los usuarios gestionar la información de los laboratorios de la EPN. Es por esta razón que el proyecto fue desarrollado en base al estándar empresarial J2EE, utilizando la arquitectura de tres capas: Capa del Cliente, Capa Intermedia la cual contiene la presentación y la lógica del negocio y la Capa de Datos.

La aplicación fue completamente codificada en lenguaje JAVA y se implementó las tres capas descritas anteriormente de la siguiente manera, Fig. 21:

- La Capa de Datos fue implementada utilizando PostgreSQL 8.4.
- La Capa Intermedia fue implementada utilizando el Servidor JBoss y el framework JSF.
- El Cliente se comunica con la lógica del negocio a través de un navegador.

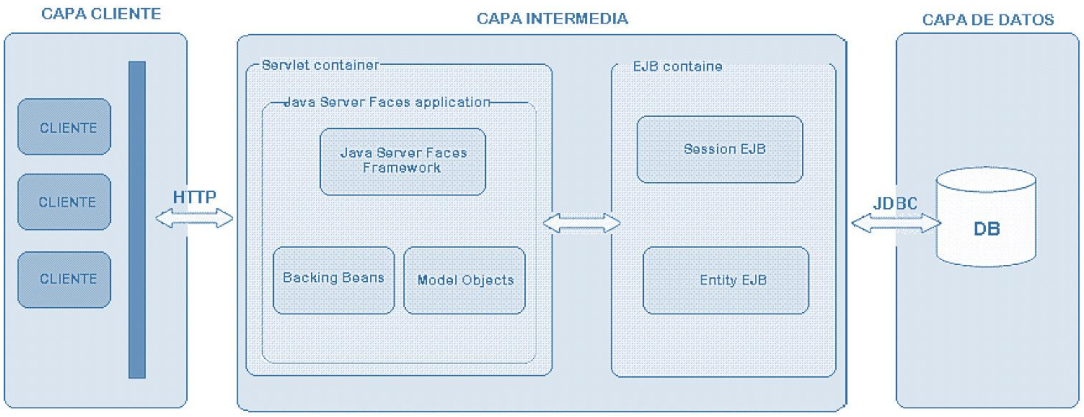
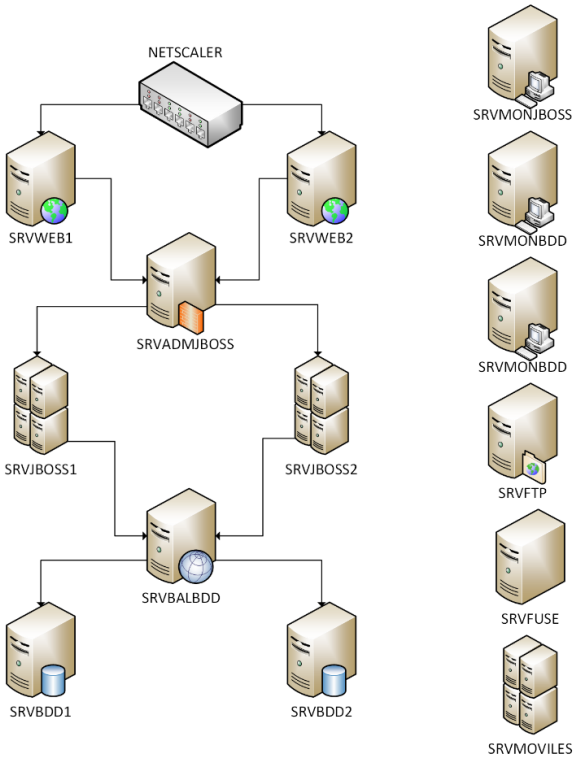


Fig. 21: Arquitectura por Capas Inicial SISLAB [32]

4.1.4. ARQUITECTURA FÍSICA

La Arquitectura física del ambiente de producción del sistema se muestra en Fig. 22:

RESUMEN ARQUITECTURA SII CON POWER8



SO	NOMBRE SERVER	DETALLE
RHEL 7.3	SRVBDD1	Servidor utilizado de motor de bdd nodo 1
RHEL 7.3	SRVWEB1	Servidor utilizado para frontear las peticiones hacia los servidores de aplicaciones web 1
RHEL 7.3	SRVJBOSS1	Servidor utilizado para el despliegue de aplicaciones en JBOSS
RHEL 7.3	SRVADMJBOSS	Controlador de Dominio de JBOSS
RHEL 7.3	SRVMONJBOSS	Servidor utilizado para le monitoreo de JBOSS
RHEL 7.3	SRVMONBDD	Servidor utilizado para monitorear los servidores de BDD
RHEL 7.3	SRVWEB2	Servidor utilizado para frontear las peticiones hacia los servidores de aplicaciones web 2
RHEL 7.3	SRVJBOSS2	Servidor utilizado para el despliegue de aplicaciones en JBOSS EAP -Nodo 2
RHEL 7.3	SRVBALBDD	Servidor utilizado para le balanceo de BDD
RHEL 7.3	SRVBDD2	Servidor utilizado de motor de BDD nodo 1
RHEL 7.3	SRVFTP	Servidor FTP
RHEL 7.3	SRVFUSE	Servidor para despliegue de FUSE – Bus de Servicios Politécnico
RHEL 7.3	SRVMOVILES	Servidor web para proporcionar data a aplicativos móviles
RHEL 7.3	NETSCALER	Balanceo de carga http y HTTPS

Fig. 22: Arquitectura Física SISLAB [33]

En el esquema adicional tenemos de SISLAB se evidencia su estructura de monolito únicamente haciendo referencia a dos esquemas de base de datos, BD1 en donde engloba la mayor cantidad de entidades de SISLAB y BD2 en lo referente a Facturación que si se lo maneja independientemente para todos los sistemas de la DGIP, está constituido como lo muestra la Fig. 23.

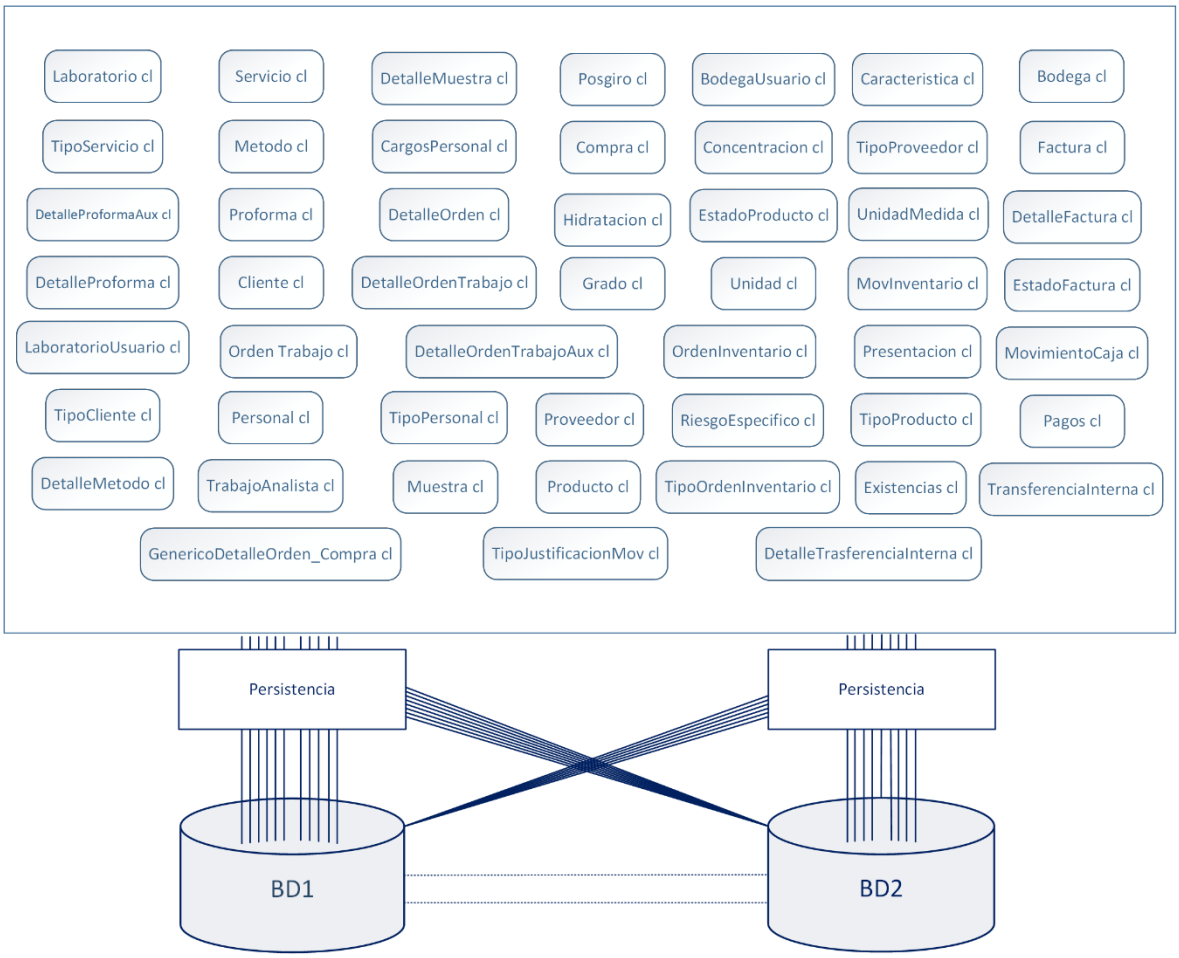


Fig. 23: Relación Capa Negocio y Datos en SISLAB [33]

Cabe acotar que existe un alto grado de dependencia entre las bases de datos debido a que varias entidades del esquema de facturación en BD2 utilizan datos de las entidades de BD1, y simultáneamente algunas tablas dentro del BD1 utilizan datos que se obtienes de la segunda base de datos.

CAPÍTULO 5 – MIGRACIÓN A MICROSERVICIOS

Las actividades iniciales de las distintas metodologías para el proceso de desarrollo de software describen una planificación mediante la selección de las herramientas a ser utilizadas, una metodología de desarrollo y el diseño de la arquitectura. En el presente capítulo serán descritas estas actividades, los problemas encontrados durante el desarrollo y una comparación del funcionamiento del actual sistema en producción con el sistema final del presente proyecto.

5.1. Herramientas de desarrollo

La selección de herramientas para el desarrollo se llevó a cabo después de un estudio de las disponibles, se tomó en cuenta que la propuesta metodológica de la tesis base existe también detalle sobre las herramientas y frameworks para la migración del sistema a Microservicios.

5.1.1. Herramientas disponibles para desarrollo de microservicios

En la actualidad existen frameworks de desarrollo basados en distintos lenguajes de programación, en su mayoría apoyados en Node.js, JavaScript y Java, el mismo que ha sobresalido por diversos motivos, principalmente por los años de vigencia que ha tenido con su continua evolución y el soporte mantenido por Oracle. Existen lenguajes que pueden ser compilados en código byte de Java, a la vez que pueden estar conectados y conformar sistemas completos, estos sistemas usan estructuras de datos como JSON y en algunos casos lenguajes de programación antiguos aún son utilizados como microservicios.

Herramientas de código abierto utilizadas como base de la arquitectura de microservicios:

- *Eclipse MicroProfile*

Eclipse MicroProfile es una unión de librerías de JavaEE y otras tecnologías que forman una base para proyectos con arquitectura en microservicios compatibles con múltiples sistemas operativos, cuenta con especificaciones para la inyección de dependencias y procesamiento de datos JSON, versiones posteriores cuentan con funciones como Tolerancia a Fallos, JWT, Métricas y Health Check. Este proyecto es mantenido por la comunidad microprofile.io dedicada a optimizar el lenguaje Java para microservicios, en la actualidad las principales empresas de este grupo son IBM, Red Hat, Tomitribe, Payara, London Java Community (LJC) y SouJava [34].

Su última versión 3.0 cuenta con una actualización a Health Check 2.0, Metrics 2.0.0, y Rest Client 1.3. con base en Java 8.

- *WildFly Thorntail*

Red Hat desarrollo una versión propia de MicroProfile que cuenta con una herramienta de configuración, este framework originalmente se llamó WildFly Swarmy finalmente fue nombrada con el nombre de Thorntail. En el sitio web de la herramienta se puede crear un archivo personalizado de compilación de Maven para un nuevo proyecto con arquitectura en microservicios, las dependencias de Maven se encargan de ensamblar todo. Thorntail elimina las partes de JEE que no son de utilidad y creará un archivo JAR listo para ser implementado.

WildFly Thorntail está basado en el servidor de aplicaciones de Red Hat WildFly, trabaja de igual manera que con los servidores de aplicaciones monolíticos tradicionales, una aplicación se implementa sobre WildFly, como se detalla en la Fig. 24.

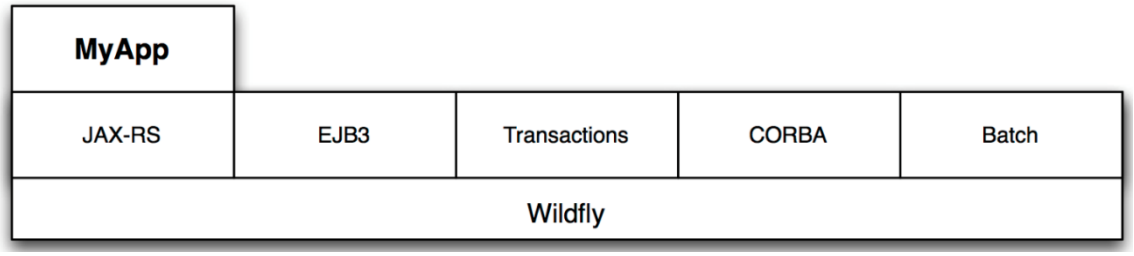


Fig. 24: Estructura de aplicación sobre Wildfly [35]

- *Spring Boot*

Spring Boot es una herramienta de código abierto basado en Java utilizada para simplificar el desarrollo de aplicaciones que usan el framework Spring, de esta manera el trabajo de los desarrolladores está centrado en el negocio del sistema sin vínculos necesarios con la configuración del mismo. La herramienta ofrece conexión con tecnologías como:

- Inyección de dependencias
- Manejo de eventos
- Manejo de recursos, i18n
- Testing, Spring MVC Test, mock
- Acceso a bases de datos, DAO, JDBC, ORM
- Integración remota, email, cache, JMS
- Interacción con lenguajes, Kotlin, Groovy [36].

Permite ejecutar proyectos como una aplicación Stand-alone (sistema o programa que no requiere ser instalado y no necesita conexión de red), pero también es posible ejecutar aplicaciones web ya que tiene embebido un servidor web que permite desplegar mediante Tomcat, Jetty o Undertow. Spring Boot tiene servicios para consultar el "estado de salud" de las aplicaciones, características como si está en funcionamiento o no, cantidad de memoria utilizada y disponible, número y detalle de los Bean's creados por la aplicación, etc.

- *Red Hat*

Para el desarrollo del proyecto inicialmente se optó por Red Hat por ser la tecnología utilizada en el actual del sistema SISLAB, dispone de su propio plugin para ser instalado en un IDE compatible, eclipse por ejemplo o instalarlo con sus propias configuraciones iniciales, Red Hat JBoss Developer Studio, Fig. 25. Dispone de la implementación de Maven para el control de dependencias, creación y registros de proyectos Maven.

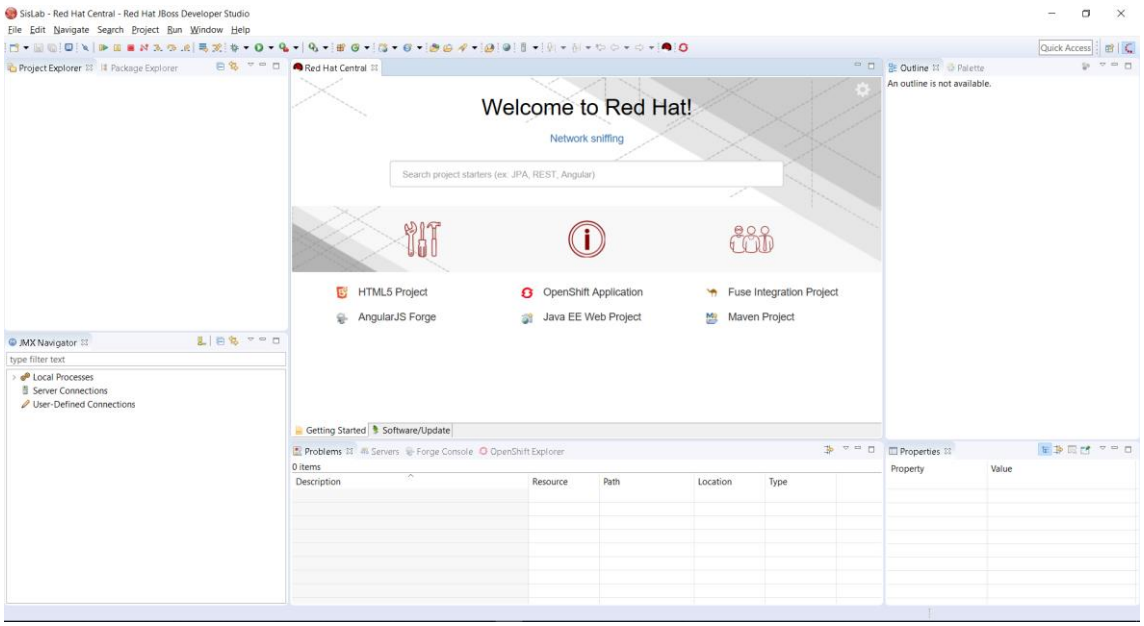


Fig. 25: Red Hat JBoss Developer Studio

El IDE cuenta con distintas versiones del servidor de aplicaciones JBoss (como se muestra en la Fig. 26), esto permite desplegar la aplicación en desarrollo sin necesidad de personalizar los archivos de configuración.

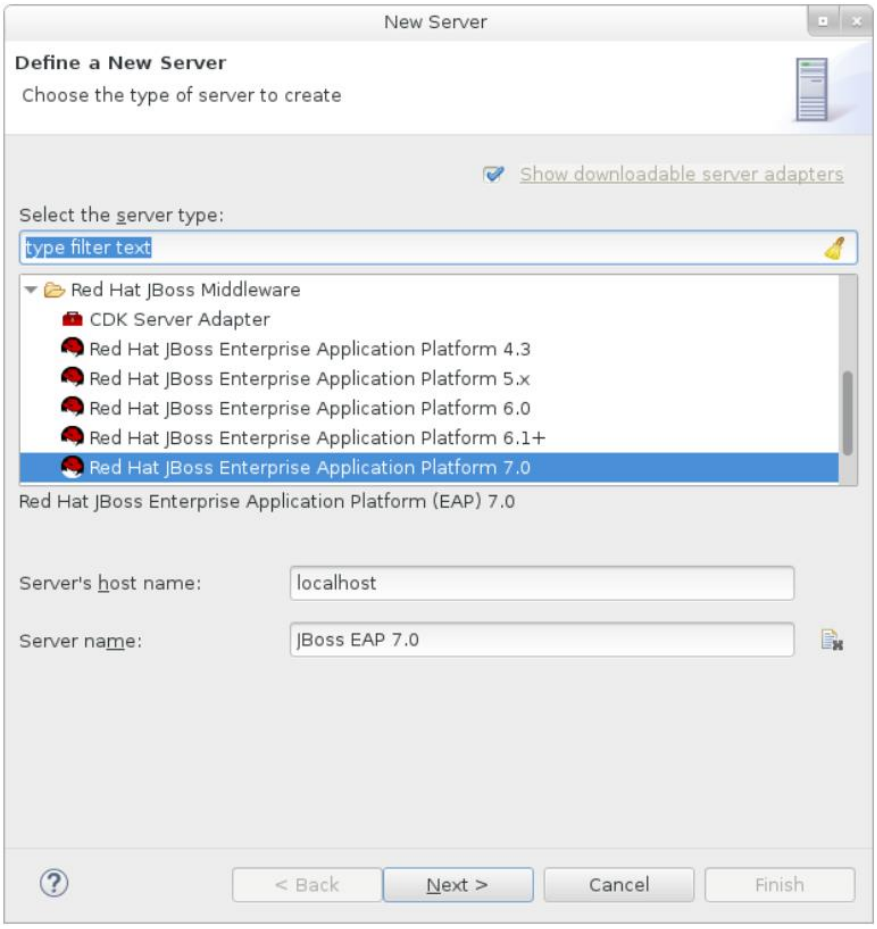


Fig. 26: Servidores disponibles en Red Hat JBoss Developer Studio [37]

En la arquitectura de microservicios existe una capa encargada de orquestar la instancia y estado de cada microservicio, en gran número de textos es nombrada como el ecosistema de microservicios. En Red Hat se dispone de RED HAT OPENSHIFT, “es una plataforma de aplicaciones en contenedores completa que integra de manera nativa tecnologías, como Docker y Kubernetes, un poderoso sistema de administración y desarrollo de clústeres de contenedores, y las combina con una base empresarial en Red Hat Enterprise Linux. Red Hat OpenShift integra la arquitectura, los procesos, las plataformas y los servicios necesarios para capacitar a los equipos de desarrollo y operaciones. Se implementa de forma fiable en todos los entornos y le permite satisfacer la demanda del cliente a la vez que reduce los costos de infraestructura” [38].

Red Hat OpenShift cuenta con:

- Kubernetes empresariales
- Soporte de aplicaciones con y sin estado
- Seguridad de los contenedores
- Compatibilidad de servicios en la nube (Amazon Web Services, Azure, Google Cloud Platform, VMware)
- Trabajo con DevOps [38]

Los frameworks revisados hasta el momento están enfocados en la parte de back-end de una aplicación web, pero no son indiferentes a la parte front-end porque cada uno tiene una extensión propia para el desarrollo de este.

- *Spring Framework*

Este framework tiene disponibles un conjunto de herramientas para el desarrollo de aplicaciones orientados a servicios en la nube, seguridad en servicios, persistencia en base de datos, etc. Utilizar solo el framework representa tener amplios conocimientos en configuración XML, el núcleo de Spring tiene la siguiente estructura, Fig. 27:

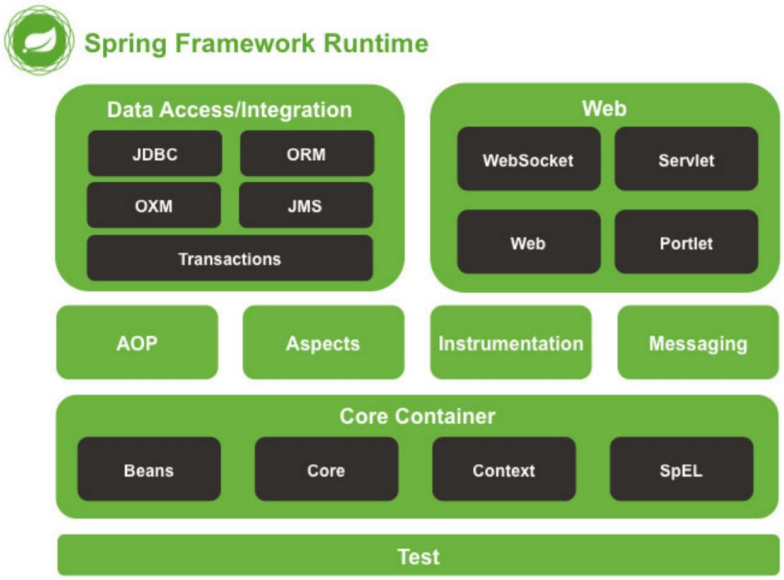


Fig. 27: Estructura de Framework Spring [39]

El framework cuenta con inyección de dependencias mediante la especificación JSR250, pero también cuenta con su propio enfoque nombrado "Spring IoC Container" y trabaja como contenedor de beans; para el trabajo con persistencia de datos y transacciones tiene compatibilidad con JPA (Java Persistence API), Hibernate, JDBC (Java Database Connectivity), frameworks IBATIS y TopLink, JDO (Java Data Objects), la ejecución de transacciones con JTA (Java Transaction API).

Las herramientas para la parte de front-end compatibles con el framework son diversas, tales como: JSF (JavaServer Faces), Struts, Tapestry, WebWork, y su propia herramienta adaptada al framework Spring MVC.

- *AngularJS*

Es un framework de desarrollo enfocado en el front-end de una aplicación web, la programación es a través de typeScript un lenguaje tipado. AngularJS permite que una aplicación web utilice únicamente las bibliotecas que se necesitan durante el desarrollo, un ejemplo es la navegación entre rutas para lo cual importa en el proyecto el módulo de ruteo. La arquitectura de AngularJS se encuentra descrita en Fig. 28, y está basada en:

- ✓ Templates en HTML con directivas propias de AngularJS, definen el orden y la manera que se ve la información de los componentes.
- ✓ Componentes para el manejo de Templates, permiten mostrar y procesar información en pantalla mediante la ejecución de eventos.
- ✓ Servicios que contienen la lógica de negocio, en ocasiones son utilizados para la recepción de la data mediante mensajes JSON y pueden ser reutilizados en más de un componente, como se lo hará en el presente proyecto. El uso del mecanismo "Injector"

permite crear una instancia de un servicio en un componente, en la programación está definido como un patrón singleton.

- ✓ Módulos para organizar los componentes y servicios creados.

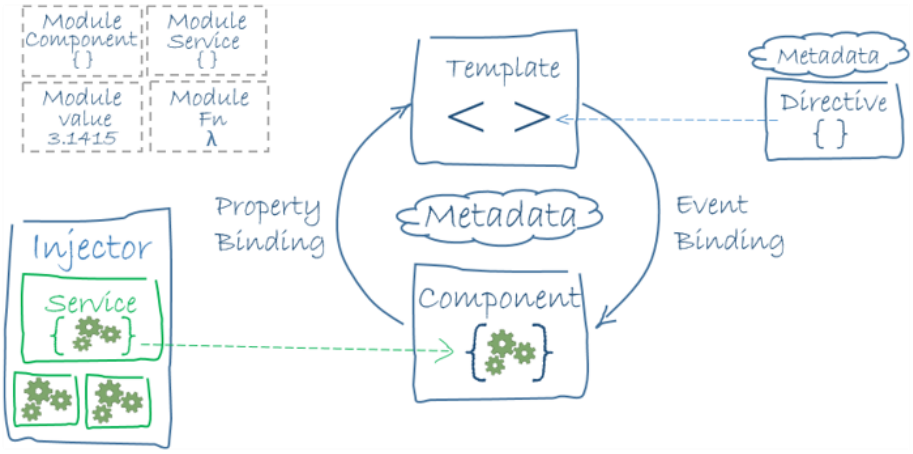


Fig. 28: Arquitectura Angular [40]

El manejo y la creación de proyectos, componentes, servicios, etc. en AngularJS se facilita mediante el uso de la Angular CLI (command line interface), herramienta de línea de comandos encargada de la estructuración de los archivos dentro del proyecto [40].

5.1.2. Herramientas seleccionadas

El proyecto se va a realizar en Java con Spring (versión 1.6) en la parte del back-end, mediante la herramienta Spring Tool Suite 4. Angular (versión 7) framework en front-end y para la presentación hacia el usuario se utilizará Angular Material este se asemeja a librerías de Bootstrap, donde se consumirán los servicios Rest generados.

Los servicios Rest serán enviados para comunicar las tecnologías en back y front-end mediante mensajes JSON, se utilizará tokens JWT (JSON Web Tokens) para seguridad en la comunicación de los servicios.

La conexión en back-end, cada microservicio con la base de datos será mediante Spring Data JPA Repository, esto permite a través de la herramienta de Spring manejar Primary Key, Foreing Key, Rollback y commit transactions, relaciones entre tablas según sea necesario, tomando en cuenta que es una base de datos ya en producción.

- *Spring Tool Suite 4*

"Spring Tools 4 es la próxima generación de herramientas Spring para su entorno de codificación favorito. En gran parte reconstruido desde cero, proporciona soporte de clase mundial para desarrollar aplicaciones empresariales basadas en Spring, ya sea que prefiera Eclipse, Visual Studio Code o Theia IDE" [36].

El IDE seleccionado para la instalación de la extensión es Eclipse (interfaz en Fig. 29), por el basto conocimiento sobre este por parte de los desarrolladores del proyecto, en conjunto con las funcionalidades que brinda al momento de trabajar en el código.

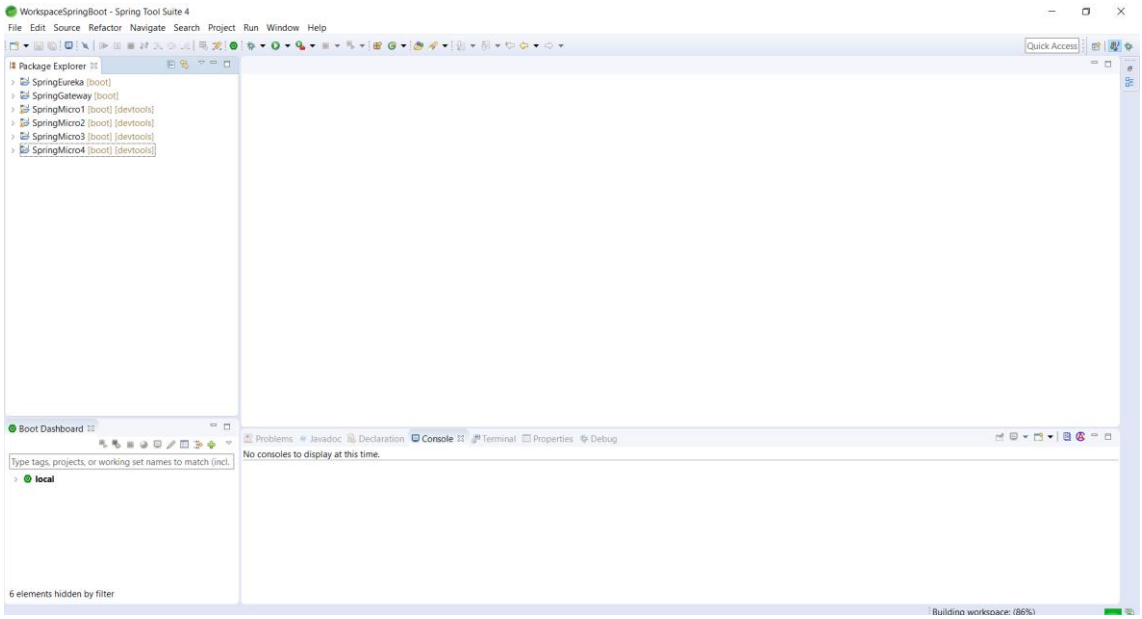


Fig. 29: Spring Tool Suite 4

- *Visual Studio Code*

"Visual Studio Code es un editor de código fuente ligero pero potente que se ejecuta en escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity)" [41].

El IDE tiene la opción de instalar extensiones para la ayuda durante el desarrollo, entre ellas control con Git, resaltado de sintaxis, autocompletación de código, etc., tiene integrado una ventana de consola para poner manejar los proyectos en desarrollo mediante comandos, esta consola puede obedecer a comandos tanto de Linux, Windows o macOs (interfaz en Fig. 30).

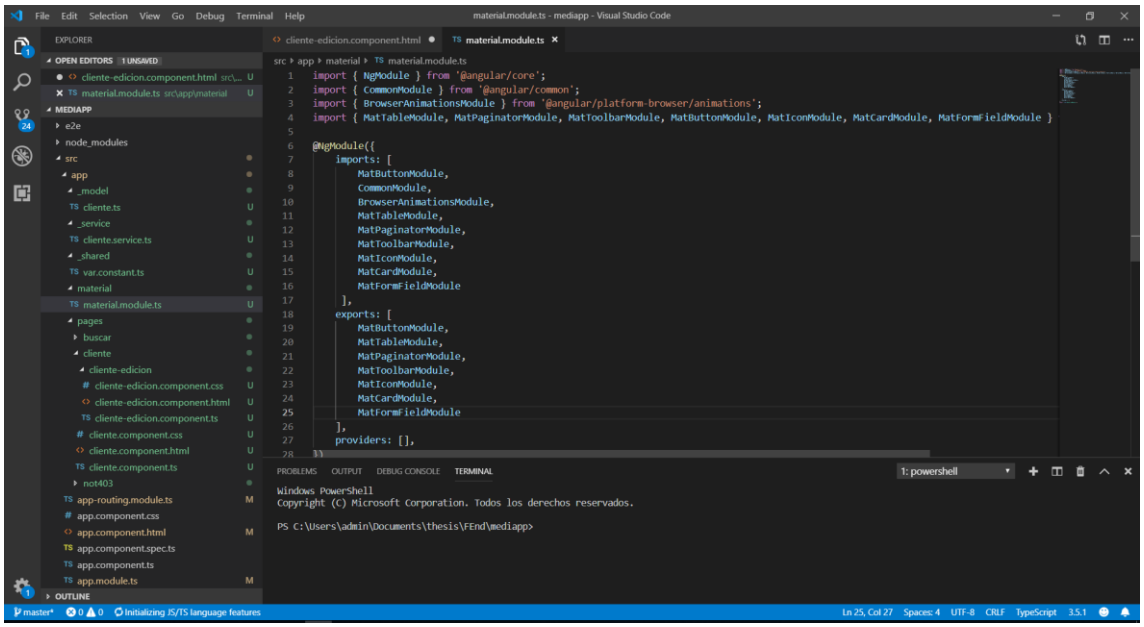


Fig. 30: Visual Studio Code

5.2. Metodología de desarrollo

El desarrollo del proyecto tendrá como guía la Metodología de Desarrollo Ágil Scrum, mediante esta se pretende realizar una estimación de tiempos adecuada, tener un control de la participación del equipo de trabajo, desarrollar incrementos funcionales del producto, realizar revisiones de lo construido en comparación con el objetivo de cada sprint y en el final del proyecto obtener el sistema SisLab con la funcionalidad completa esperada.

5.2.1. FASE 1: DEFINICIÓN DEL BACKLOG DEL PRODUCTO

A. DESCRIPCIÓN DEL PROYECTO

La descripción completa del sistema SisLab se realizó en el capítulo 4 del presente escrito.

B. DEFINIENDO LA PILA DE PRODUCTO

Se define el Product backlog del proyecto, mediante una lista de requerimientos de usuario priorizada y definida por los participantes del proyecto, tal como se muestran en la Tabla 5.

PRODUCT BACKLOG			
ID	Descripción	Importancia	Notas
1	Levantamiento de requerimientos para los Microservicios mediante el estudio del proyecto de titulación: Propuesta Metodológica Para Migración De Sistemas Web Con Arquitectura Monolítica Hacia Una Arquitectura Basada En Microservicios [6].	50	

2	Identificar el modelo de Microservicios final y diseñar la estructura de los patrones de diseño de software utilizando notación UML (Unified Modeling Language).	50	
3	Estudio y selección de herramientas de desarrollo .	20	
4	Desarrollo de microservicios 1 y 2 en back-end.	80	
5	Pruebas de servicios Rest generados por microservicios 1 y 2.	60	
6	Desarrollo de microservicios 3 y 4 en back-end.	80	
7	Pruebas de servicios Rest generados por microservicios 3 y 4.	60	
8	Implementar interfaces graficas web para comprobar el funcionamiento de los Microservicios.	80	
9	Validación de modelo de microservicios final.	80	

Tabla 5: Product Backlog

En la tabla, se puede notar que el Product backlog tiene términos para una persona que tiene dominio técnico básico del tema y la misma permite determinar el avance y los tiempos asignados para el desarrollo de cada sprint.

5.2.2. FASE 2: PLANIFICACION DEL SPRINT

El equipo disponible para el desarrollo está conformado por los dos autores del presente proyecto de tesis, por este motivo las asignaciones de cargos para la metodología Scrum serán varias para cada desarrollador, tomando en cuenta esta dificultad se conformó las designaciones de la siguiente manera:

Product Owner: Ing. Roberto García, PhD. Pamela Flores

Scrum Master: Cristhian Chulca

Scrum Team:

- Cristhian Chulca
- Raúl Molina

En el proyecto el Product Owner verdadero es la DGIP en la Escuela Politécnica Nacional, pero por motivos de tiempos y obligaciones del equipo de desarrollo dicho Product Owner realizara una revisión final cuando el sistema sea desarrollado en su totalidad.

En la reunión de planificación de Sprints el equipo Scrum estructura los Sprints necesarios y sus estimaciones de tiempo inicial, además verificar las importancias establecidas en el backlog producto para el desarrollo del proyecto.

Primera reunión de planificación de Sprint (SPRINT 1):

Fecha: lunes 01/10/2018

Hora: 17:00p.m. – 22:00p.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 29/10/2018

Metas del Sprint:

- Realizar una revisión y análisis de la tesis, Propuesta Metodológica Para Migración De Sistemas Web Con Arquitectura Monolítica Hacia Una Arquitectura Basada En Microservicios [6].
- Realizar un estudio del modelo de arquitectura detallado en la tesis y realizar una propuesta propia enfocada en las funcionalidades actuales del sistema SisLab descritas en el documento de tesis.
- Revisar información, roles y obligaciones en la metodología Scrum para la designación en la siguiente reunión.

En la presente reunión se tomó en cuenta inconvenientes con los tiempos planificados, debido a que el sr. Raúl Molina ingresó al proyecto de tesis en reemplazo de un integrante que por motivos personales solicitó ser excluido. Se acordó tener reuniones de pie como lo indica la metodología en horas de la noche durante los días de cada sprint para comunicar aspectos importantes evidenciados en la revisión de la tesis base. En la metodología Scrum el propósito de la reunión sprint planning es dar al equipo suficiente información para desarrollar el trabajo durante el tiempo del sprint.

Durante la reunión se dio a conocer que los participantes se encuentran laborando a tiempo completo en sus trabajos respectivos, y por este motivo las tareas asignadas de los Sprints serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serian 20 distribuidas de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 1	Desarrollador 1 – Raúl Molina	✓ Realizar una revisión y análisis de la tesis base páginas 1 – 39.	7
			✓ Realizar un estudio del modelo de arquitectura detallado en la tesis base.	3
2	Sprint 1		✓ Realizar una revisión y análisis de	6

		Desarrollador 2 – Cristhian Chulca	la tesis base páginas 40 – 86.	
			✓ Realizar una propuesta propia enfocada en las funcionalidades actuales del sistema SisLab descritas en el documento de tesis.	3
			✓ Revisar información, funciones y obligaciones en la metodología Scrum.	1

Tabla 6: Sprints backlog definidos en la primera reunión de planificación

La Tabla 6 tiene una descripción de tareas para cada desarrollador en el equipo dentro del primer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Segunda reunión de planificación de Sprint (SPRINT 2):

Fecha: lunes 29/10/2018

Hora: 18:00p.m. – 22:00p.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 26/11/2018

Metas del Sprint:

- Definir un plan de desarrollo entorno a la tesis base y la propuesta personal de los miembros del equipo de desarrollo
- Designar roles dentro del equipo para cumplimiento con metodología scrum.
- Definir un modelo de arquitectura para el desarrollo de back-end y la conexión con front-end.
- Realizar un estudio de herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.
- Revisión inicial de código actual del sistema SisLab para un entendimiento de la estructura.

En el inicio de la reunión se designaron los cargos para la metodología scrum con apoyo del Ingeniero Cristian León, especialista en la misma con varios años de experiencia en su aplicación en el cargo de scrum máster. Se acordó que Cristhian Chulca desempeñará el cargo de scrum máster en el equipo del presente proyecto bajo la ayuda del especialista Cristian León. En la designación del rol de Product Owner (PO) para el proyecto se tienen inconvenientes porque la persona que tiene completo conocimiento funcional sobre el sistema SisLab es el ingeniero Roberto García quien labora en la DGIP, por este motivo no se puede tener una revisión

constante de los entregables de cada sprint, en consultas sobre la funcionalidad y data en la base de datos si estará disponible en su lugar de trabajo y al final del proyecto podrá revisar la funcionalidad del nuevo sistema SisLab desarrollado, tomando en cuenta esto se acordó que durante el desarrollo los integrantes del equipo también desempañaran el rol de PO, bajo la condición que la revisión se haga inversamente de los entregables de modo que Cristhian Chulca recibirá los desarrollos o documentos de Raúl Molina y a su vez Raúl Molina de Cristhian Chulca.

En la designación de Tester para el equipo se tomó la decisión de nombrar a Raúl Molina, contando con el apoyo de Doctora Pamela Flores en los temas de arquitectura, para las historias o tareas del Raúl Molina que se crea necesario las pruebas las realizará Cristhian Chulca.

Durante el anterior sprint se mantuvo reuniones periódicamente para la revisión de la tesis base; los problemas, observaciones o cambios decididos para la implementación del sistema están descritos en el capítulo 5 del presente escrito. Existió una reunión de los desarrolladores con el ingeniero Roberto García para una descripción inicial de las funcionalidades del sistema SisLab y estructura actual del código del proyecto. En reuniones posteriores los miembros del equipo decidieron la arquitectura a desarrollar descrita en un capítulo del presente documento.

En la reunión se manifestó que para los tiempos de los desarrolladores se debe tomar en cuenta que se encuentran laborando a tiempo completo en sus trabajos, y por este motivo las tareas asignadas de este sprint serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serian 20 distribuidas de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 2	Desarrollador 1 – Raúl Molina	✓ Definir el plan de desarrollo (tarea en conjunto).	4
			✓ Definir un modelo de arquitectura para el desarrollo (tarea en conjunto).	4
			✓ Realizar un estudio de herramientas para back-end.	1
			✓ Realizar un estudio de herramientas para front-end.	1
2	Sprint 2	Desarrollador 2 – Cristhian Chulca	✓ Definir el plan de desarrollo (tarea en conjunto).	4
			✓ Definir un modelo de arquitectura para el desarrollo (tarea en conjunto).	4

			✓ Revisión de código actual de sistema SisLab.	2

Tabla 7: Sprints backlog definidos en la segunda reunión de planificación

La Tabla 7 tiene una descripción de tareas para cada desarrollador en el equipo en el segundo sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Tercera reunión de planificación de Sprint (SPRINT 3):

Fecha: lunes 26/11/2018

Hora: 18:00p.m. – 23:00p.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: miércoles 26/12/2018

Metas del Sprint:

- Seleccionar herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.
- Estudio a profundidad de código actual del sistema SisLab y separación de clases bean para cada microservicio dentro de la arquitectura diseñada.
- Escritura inicial de documento de tesis, capitulo 1 – Marco Teórico.
- Conexión a red interna de DGIP de manera remota mediante VPN.
- Trabajar con los roles y obligaciones designados de la metodología scrum para planificar posibles cambios en el siguiente sprint.

En la reunión se acordó trabajar la mitad del tiempo total del sprint en el estudio del código actual del sistema de manera independiente, después se realizarán reuniones diarias entre los desarrolladores para cumplir con la tarea de separar las clases bean para los microservicios. De la misma manera se trabajará para la selección de herramientas de desarrollo, primero cada desarrollador revisará por su cuenta las posibles opciones tomando en cuenta la experiencia adquirida durante su vida de estudiantes y la nueva adquirida en sus respectivos trabajos, posteriormente se mantendrán reuniones de discusión para elegir las herramientas y frameworks más apropiados; se toma en consideración que en la tesis base también existe una recomendación para las herramientas. El modelo de arquitectura decidido en el anterior sprint tendrá una revisión pues existen dependencias entre los microservicios que deberán ser resueltas.

De igual manera que los anteriores sprint las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serian 20 distribuidas de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 3	Desarrollador 1 – Raúl Molina	✓ Conexión a red interna de DGIP de manera remota mediante VPN.	1
			✓ Estudio del código actual del sistema SisLab.	4
			✓ Separación de clases bean para cada microservicio (tarea en conjunto).	1
			✓ Estudio de herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.	2
			✓ Seleccionar herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end (tarea en conjunto).	2
2	Sprint 3	Desarrollador 2 – Cristhian Chulca	✓ Estudio de herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.	2
			✓ Estudio del código actual del sistema SisLab.	3
			✓ Separación de clases bean para cada microservicio (tarea en conjunto).	1
			✓ Revisión de roles y obligaciones designados de la metodología scrum.	1
			✓ Seleccionar herramientas y frameworks para el desarrollo de la aplicación en back-	2

			end y front-end (tarea en conjunto).	

Tabla 8: Sprints backlog definidos en la tercera reunión de planificación

La Tabla 8 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Cuarta reunión de planificación de Sprint (SPRINT 4):

Fecha: miércoles 26/12/2018

Hora: 18:00p.m. – 23:00p.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 21/01/2019

Metas del Sprint:

- Instalar y configurar de IDE Red Hat JBoss Developer Studio.
- Crear 4 proyectos en IDE, uno para cada microservicio.
- Copiar backup de base de datos de sistema SisLab y obtener metadata de cada tabla.
- Instalar y configurar sistema gestor de base de datos PostgreSQL.
- Realizar un estudio del orquestador de microservicios disponible en Red Hat.

En la reunión se acordó para el inicio del desarrollo del proyecto en asignar las tareas entorno a la base de datos a Raúl Molina y las tareas referentes al IDE de desarrollo a Cristhian Chulca. En este sprint se tendrán reuniones los 3 últimos días de trabajo para unificar los avances obtenidos.

De igual manera que los anteriores sprint las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serian 18 distribuidas de la siguiente manera:

Días disponibles

Raúl Molina 9

Cristhian Chulca 9

Días disponibles 18

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 4	Desarrollador 1 – Raúl Molina	✓ Copiar backup de base de datos de sistema SisLab.	3
			✓ Obtener metadata por tabla en base de datos de SisLab	4

			✓ Instalar y configurar sistema gestor de base de datos PostgreSQL.	2
2	Sprint 4	Desarrollador 2 – Cristhian Chulca	✓ Instalar y configurar de IDE Red Hat JBoss Developer Studio.	3
			✓ Crear proyectos para microservicios.	3
			✓ Instalar y configurar sistema gestor de base de datos PostgreSQL.	1
			✓ Realizar un estudio del orquestador de microservicios disponible en Red Hat.	2

Tabla 9: Sprints backlog definidos en la cuarta reunión de planificación

La Tabla 9 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 26 días (4 semanas laborables, 4 horas diarias por desarrollador).

Quinta reunión de planificación de Sprint (SPRINT 5):

Fecha: lunes 21/01/2019

Hora: 18:00p.m. – 23:00p.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 18/02/2019

Metas del Sprint:

- Copiar archivos de objetos Beans a proyectos de microservicios 1 y 2 en IDE Red Hat.
- Revisar creación y configuración de proyecto Openshift para la llamada y orquestación de microservicios.
- Copiar archivos JSF (Java Server Faces) a proyectos de microservicios 1 y 2 en IDE Red Hat.
- Revisar información, instalación y configuración de Kubernetes o contenedores en Red Hat.

En la reunión se expuso los contratiempos referentes a la instalación del IDE seleccionado para el desarrollo debido a requerimientos para su descarga en la página oficial de Red Hat, esto representó un retraso en el desarrollo de las tareas del sprint anterior. La creación de los proyectos para cada microservicio tuvo menos tiempo del asignado y se crearon únicamente los

proyectos para el microservicio 1 y 2 con las configuraciones que se optaron con la revisión inicial del IDE.

El proceso de copiado de backup también genero retrasos porque el actual sistema SisLab se encuentra en producción y se definió un punto de corte óptimo en relación a la cantidad de data. La revisión que se hizo sobre el orquestador disponible en Red Hat, se llevó a cabo con éxito, pero se anticipan posibles contrariedades porque existen versiones gratuitas y de pago disponibles, en el presente sprint se investigará los tiempos de acceso a las versiones gratuitas y el costo de las versiones pagadas, se acordó tener una reunión para conocer el posible acceso a las versiones.

De igual manera que los anteriores sprint las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serian 20 distribuidas de la siguiente manera:

Días disponibles	
Raúl Molina	10
Cristhian Chulca	10
Días disponibles	20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	**Sprint**	**Responsable**	**Tareas**	**Días asignados**
1	Sprint 5	Desarrollador 1 – Raúl Molina	✓ Copiar archivos de objetos Beans a proyecto de microservicio 2 en IDE Red Hat.	3
			✓ Copiar archivos JSF a proyecto de microservicio 2 en IDE Red Hat.	3
			✓ Revisar información, instalación y configuración de Kubernetes o contenedores en Red Hat.	2
			✓ Reunión, disponibilidad de Openshift (tarea en conjunto).	2
2	Sprint 5	Desarrollador 2 – Cristhian Chulca	✓ Copiar archivos de objetos Beans a proyecto de microservicio 1 en IDE Red Hat.	3
			✓ Copiar archivos JSF a proyecto de microservicio 1 en IDE Red Hat.	3

			✓ Revisar creación y configuración de proyecto Openshift para la llamada y orquestación de microservicios.	2
			✓ Reunión, disponibilidad de Openshift (tarea en conjunto).	2

Tabla 10: Sprints backlog definidos en la quinta reunión de planificación

La Tabla 10 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Sexta reunión de planificación de Sprint (SPRINT 6):

Fecha: lunes 18/02/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 18/03/2019

Metas del Sprint:

- Revisión de framework Spring, instalación, configuración, interfaz para conexión con base de datos y front-end, JPA, Eureka.
- Descarga, instalación y configuración del IDE disponible en el framework, Spring Tools Suite.
- Revisión de framework Angular, IDE para desarrollo, instalación de herramientas necesarias.

El anterior sprint se decidió cambiar el software de desarrollo por motivo de vigencia de licencias en Red Hat y otros inconvenientes debidamente descritos en el capítulo 5 del presente escrito. En una revisión de las herramientas disponibles para el desarrollo realizada en un sprint anterior el equipo consideró como una segunda opción a Spring framework para el desarrollo de microservicios en back-end y Angular framework para front-end, debido a los problemas con Red Hat desde este sprint se trabajará con los frameworks nombrados. Por la reducción del tiempo disponible para el desarrollo se optó por dividir las tareas enfocada en front-end y back-end para Raúl Molina y Cristhian Chulca respectivamente, los últimos 4 días del sprint se llevarán a cabo reuniones en conjunto para la compartición de la información y los dos integrantes tengan instalado las herramientas necesarias.

De igual manera que los anteriores sprint las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo. Los días disponibles del equipo en total serán 20 distribuidos de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 6	Desarrollador 1 – Raúl Molina	✓ Revisión de framework Spring, instalación, configuración.	2
			✓ Revisión de framework Angular, IDE para desarrollo.	4
			✓ Instalación de herramientas en equipo (tarea en conjunto).	4
2	Sprint 6	Desarrollador 2 – Cristhian Chulca	✓ Revisión de framework Spring, instalación, configuración.	2
			✓ Descarga, instalación y configuración del IDE disponible en el framework, Spring Tools Suite.	4
			✓ Instalación de herramientas en equipo (tarea en conjunto).	4

Tabla 11: Sprints backlog definidos en la sexta reunión de planificación

La Tabla 11 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Séptima reunión de planificación de Sprint (SPRINT 7):

Fecha: lunes 18/03/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 15/04/2019

Metas del Sprint:

- Crear y configurar proyectos en IDE Spring Tools Suite para microservicios 1 y 2.

- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 1.
- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 2.
- Generación de scripts de tablas integrantes de microservicio 1.
- Generación de scripts de tablas integrantes de microservicio 2.

En la reunión del presente sprint se llegó al acuerdo de aumentar los días de duración del sprint para tener más reuniones en conjunto debido al cambio de framework y herramientas para el desarrollo del sistema. En la planificación anterior el número de días fue 4 pero se necesitaron un total de 10 días en reuniones del equipo para llevar a cabo las tareas planificadas con el framework Spring y sus componentes. Se llevó a cabo la instalación de los IDE: Spring Tools Suite y Visual Studio Code, para el desarrollo con los framework Spring y Angular respectivamente.

Las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo por parte de los desarrolladores del equipo. Los días disponibles del equipo en total serán 30, el número de días aumenta en comparación con los anteriores Sprints debido al cambio de herramientas de desarrollo, se acuerda tener reuniones en conjunto la mayor cantidad de tiempo total del sprint, los días serán distribuidos de la siguiente manera:

Días disponibles

Raúl Molina 15

Cristhian Chulca 15

Días disponibles 30

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 7	Desarrollador 1 – Raúl Molina	✓ Generación de scripts de tablas integrantes de microservicio 1.	2
			✓ Generación de scripts de tablas integrantes de microservicio 2.	2
			✓ Crear proyecto en IDE Spring Tools Suite para microservicios 2 (tarea en conjunto).	2
			✓ Configurar proyecto en IDE Spring Tools Suite para microservicios 2 (tarea en conjunto).	3
			✓ Desarrollo de modelos de base de datos en proyecto Spring para	3

			microservicio 1 (tarea en conjunto).	
			✓ Desarrollo de modelos de base de datos en proyecto Spring para microservicio 2 (tarea en conjunto).	3
2	Sprint 7	Desarrollador 2 – Cristhian Chulca	✓ Crear proyecto en IDE Spring Tools Suite para microservicios 1.	2
			✓ Configurar proyecto en IDE Spring Tools Suite para microservicios 1.	2
			✓ Crear proyecto en IDE Spring Tools Suite para microservicios 2 (tarea en conjunto).	2
			✓ Configurar proyecto en IDE Spring Tools Suite para microservicios 2 (tarea en conjunto).	3
			✓ Desarrollo de modelos de base de datos en proyecto Spring para microservicio 1 (tarea en conjunto).	3
			✓ Desarrollo de modelos de base de datos en proyecto Spring para microservicio 2 (tarea en conjunto).	3

Tabla 12: Sprints backlog definidos en la séptima reunión de planificación

La Tabla 12 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 6 horas diarias por desarrollador).

Octava reunión de planificación de Sprint (SPRINT 8):

Fecha: lunes 15/04/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 13/05/2019

Metas del Sprint:

- Desarrollo de servicios y controladores en back-end para microservicio 1.
- Desarrollo de servicios y controladores en back-end para microservicio 2.
- Crear secuenciales para microservicio 1 y 2.
- Pruebas de servicios Rest en back-end microservicio 1.
- Pruebas de servicios Rest en back-end microservicio 2.

El anterior sprint se creó y configuró los proyectos para los microservicios 1 y 2, para la comunicación con la base de datos mediante JPA, para esto fue necesario completar las tareas de generación de scripts de tablas de la base de datos actual del sistema SisLab tareas que estuvieron a cargo del desarrollador Raúl Molina. En la creación de modelos dentro de los proyectos se realizaron tareas de consulta en documentación de JPA para las relaciones de Foreing Key, además se encontraron problemas con la creación de Primary Key y sus respectivos secuenciales. La actual base de datos del sistema SisLab tiene más del 90% de tablas con Primary Key de tipo String y la versión utilizada para el desarrollo del sistema de la herramienta JPA tiene una implementación que debe ser personalizada para adecuar este tipo de secuenciales.

Las tareas serán desarrolladas a tiempo parcial, específicamente medio tiempo por parte de los desarrolladores del equipo. Los días disponibles del equipo en total serán 30, igual al sprint anterior debido al cambio de herramientas de desarrollo, se acuerda tener reuniones en conjunto si es posible todos los días del sprint, los días serán distribuidos de la siguiente manera:

Días disponibles

Raúl Molina	15
Cristhian Chulca	15
Días disponibles	30

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 8	Desarrollador 1 – Raúl Molina	✓ Desarrollo de servicios y controladores en back-end para microservicio 1 (tarea en conjunto).	5
			✓ Desarrollo de servicios y controladores en back-end para microservicio 2 (tarea en conjunto).	4
			✓ Pruebas de servicios Rest en back-end microservicio 1 (tarea en conjunto).	3

			✓ Pruebas de servicios Rest en back-end microservicio 2 (tarea en conjunto).	3
2	Sprint 8	Desarrollador 2 – Cristhian Chulca	✓ Desarrollo de servicios y controladores en back-end para microservicio 1 (tarea en conjunto).	4
			✓ Desarrollo de servicios y controladores en back-end para microservicio 2 (tarea en conjunto).	4
			✓ Crear secuenciales para microservicio 1.	1
			✓ Crear secuenciales para microservicio 2.	1
			✓ Pruebas de servicios Rest en back-end microservicio 1 (tarea en conjunto).	3
			✓ Pruebas de servicios Rest en back-end microservicio 2 (tarea en conjunto).	2

Tabla 13: Sprints backlog definidos en la octava reunión de planificación

La Tabla 13 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 6 horas diarias por desarrollador).

Novena reunión de planificación de Sprint (SPRINT 9):

Fecha: lunes 13/05/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 10/06/2019

Metas del Sprint:

- Creación y configuración de proyecto front-end en framework Angular.
- Desarrollo de componentes CRUD en front-end para microservicio 1.
- Desarrollo de componentes CRUD en front-end para microservicio 2.
- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 3.

En la reunión se evidenció el avance conseguido con la extensión de días en los dos Sprints anteriores y se decidió volver a tener una duración total de 20 días laborables para el presente sprint. En los proyectos back-end para los microservicios 1 y 2 se ponderó tener un avance aproximado del 80%, la creación de modelos con JPA (Java Persistence API) fue desarrollada con éxito y esto fue justificado mediante la comparación entre los diagramas entidad-relación de la base de datos actual del sistema SisLab y la obtenida del sistema en el desarrollo de este proyecto. En las pruebas de servicios Rest de los microservicios 1 y 2 se obtuvieron resultados exitosos, el medio para evidenciar estas pruebas se decidirá en los siguientes Sprints. El presente Sprint de igual manera se intentará tener la mayor cantidad de reuniones para el desarrollo conjunto por parte de los desarrolladores.

Las tareas serán desarrolladas a tiempo parcial, específicamente a medio tiempo por parte de los desarrolladores del equipo. Los días disponibles del equipo en total serán 20, esto debido a que se acordó tener un desarrollo avanzado de los proyectos de los microservicios 1 y 2, los días serán distribuidos de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 9	Desarrollador 1 – Raúl Molina	✓ Creación de proyecto front-end en framework Angular.	2
			✓ Configuración de proyecto front-end en framework Angular.	2
			✓ Desarrollo de componentes CRUD en front-end para microservicio 1 (tarea en conjunto).	3
			✓ Desarrollo de componentes CRUD en front-end para microservicio 2 (tarea en conjunto).	3
2	Sprint 9	Desarrollador 2 – Cristhian Chulca	✓ Desarrollo de modelos de base de datos en proyecto Spring para microservicio 3.	4
			✓ Desarrollo de componentes CRUD en front-end para	3

			microservicio 1 (tarea en conjunto).	
			✓ Desarrollo de componentes CRUD en front-end para microservicio 2 (tarea en conjunto).	3

Tabla 14: Sprints backlog definidos en la novena reunión de planificación

La Tabla 14 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Décima reunión de planificación de Sprint (SPRINT 10):

Fecha: lunes 10/06/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 08/07/2019

Metas del Sprint:

- Desarrollo de servicios y controladores en back-end para microservicio 3.
- Generación de scripts de tablas integrantes de microservicio 3.
- Pruebas de servicios Rest en back-end microservicio 3.
- Pruebas de componentes CRUD en front-end de microservicio 1.

En la reunión de planning fue comentada la revisión que se tuvo con el PO del equipo Dra. Pamela Flores, el desarrollo hasta el actual Sprint fue aprobado y se solicitó seguir avanzando también con el escrito del proyecto de tesis. El backlog del proyecto no tiene en consideración el escrito de tesis, por este motivo se acuerda tiempos extras para el avance formal del escrito porque los desarrolladores hasta el momento tienen apuntes en forma de borrador de los avances, se asignará una tarea para cada desarrollador fuera del tiempo total. El proyecto en front-end tuvo los avances esperados, las tareas referentes a este fueron cumplidas, los componentes para los microservicios 1 y 2 serán probados cuando se resuelvan los problemas con las dependencias entre microservicios.

Las tareas serán desarrolladas a tiempo parcial, específicamente a medio tiempo por parte de los desarrolladores del equipo, el tiempo destinado para el desarrollo del escrito queda exento del tiempo descrito a continuación. Los días disponibles del equipo en total serán 20, esto debido a que se acordó tener un desarrollo avanzado de los proyectos de los microservicios 1 y 2, los días serán distribuidos de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1	Sprint 10	Desarrollador 1 – Raúl Molina	✓ Generación de scripts de tablas integrantes de microservicio 3.	1
			✓ Pruebas de servicios Rest en back-end microservicio 3.	3
			✓ Pruebas de componentes CRUD en front-end de microservicio 1 (tarea en conjunto).	6
2	Sprint 10	Desarrollador 2 – Cristhian Chulca	✓ Desarrollo de servicios en back-end para microservicio 3.	2
			✓ Desarrollo de controladores en back-end para microservicio 3.	2
			✓ Pruebas de componentes CRUD en front-end de microservicio 1 (tarea en conjunto).	6

Tabla 15: Sprints backlog definidos en la décima reunión de planificación

La Tabla 15 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Undécima reunión de planificación de Sprint (SPRINT 11):

Fecha: lunes 08/07/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 05/08/2019

Metas del Sprint:

- Desarrollo de componentes CRUD en front-end para microservicio 3.
- Pruebas de componentes CRUD en front-end de microservicio 2.
- Pruebas de servicios Rest microservicio 3.

- Creación y configuración de proyecto Gateway, Eureka.
- Estudio y solución de dependencias entre microservicios.
- Pruebas de sistema en front-end.

El principal tema de la reunión de planning fue el desarrollo del escrito, se decide aumentar el tiempo destinado. Las tareas del anterior sprint para el desarrollo del microservicio 3 no fueron completadas en su totalidad, quedando pendiente las pruebas de servicios Rest en back-end para el microservicio. Las pruebas ejecutadas para el microservicio 1 en front-end se realizaron desde el buscador Google Chrome y se tiene planificado realizar pruebas desde dos diferentes buscadores más al final del desarrollo del proyecto.

Las tareas serán desarrolladas a tiempo parcial, específicamente a medio tiempo por parte de los desarrolladores del equipo, el tiempo destinado para el desarrollo del escrito queda exento del tiempo descrito a continuación. Los días disponibles del equipo en total serán 20, distribuidos de la siguiente manera:

Días disponibles

Raúl Molina 10

Cristhian Chulca 10

Días disponibles 20

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1.	Sprint 11	Desarrollador 1 – Raúl Molina	✓ Desarrollo de componentes CRUD en front-end para microservicio 3.	3
			✓ Pruebas de servicios Rest microservicio 3.	2
			✓ Creación de proyecto Gateway (tarea en conjunto).	1
			✓ Configuración de proyecto Gateway (tarea en conjunto).	1
			✓ Creación de proyecto Discovery Server (Eureka) (tarea en conjunto).	1
			✓ Configuración de proyecto Discovery Server (Eureka) (tarea en conjunto).	1
			✓ Pruebas de sistema en front-end.	1
2			✓ Pruebas de componentes CRUD	1

	Sprint 11	Desarrollador 2 – Cristhian Chulca	en front-end de microservicio 2.	
			✓ Creación de proyecto Gateway (tarea en conjunto).	1
			✓ Configuración de proyecto Gateway (tarea en conjunto).	1
			✓ Creación de proyecto Discovery Server (Eureka) (tarea en conjunto).	1
			✓ Configuración de proyecto Discovery Server (Eureka) (tarea en conjunto).	1
			✓ Pruebas de sistema en front-end.	2
			✓ Estudio de dependencias entre microservicios.	2
			✓ Prueba de soluciones de dependencias entre microservicios.	1

Tabla 16: Sprints backlog definidos en la undécima reunión de planificación

La Tabla 16 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 4 horas diarias por desarrollador).

Duodécima reunión de planificación de Sprint (SPRINT 12):

Fecha: lunes 05/08/2019

Hora: 18:00p.m. – 24:00a.m.

Lugar: Domicilio de Raúl Molina

Próxima reunión: lunes 02/09/2019

Metas del Sprint:

- Escritura de documento de tesis.
- Desarrollo de proyecto Gateway.
- Desarrollo de proyecto Discovery Server (Eureka).
- Desarrollo de soluciones para dependencias entre microservicios.
- Pruebas de funcionamiento del sistema con llamada a microservicios.
- Pruebas finales del sistema parte 1.
- Pruebas finales del sistema parte 2.

La reunión de planning se centró en el tiempo disponible para la finalización del proyecto tanto el sistema como el escrito de tesis, para las pruebas finales del sistema se decidió solicitar acceso al centro DGIP a el sistema SisLab actual en producción para hacer las comparaciones en tiempo de respuesta y trabajo normal con el sistema desarrollado con arquitectura de Microservicios. Existen varias opciones para el desarrollo de las soluciones de dependencias entre microservicios, en el actual sprint serán implementadas. El proyecto Gateway y Discovery Server para la orquestación de los microservicios será desarrollado en reuniones en conjunto, porque el anterior sprint se hizo el estudio y se llegó a la conclusión que el desarrollo será más complejo que los proyectos de los microservicios y el front-end por tanto el tiempo será mayor.

Las tareas serán desarrolladas a tiempo parcial, específicamente a medio tiempo por parte de los desarrolladores del equipo, el tiempo destinado para el desarrollo del escrito queda exento del tiempo descrito a continuación. Los días disponibles del equipo en total serán 30, distribuidos de la siguiente manera:

Días disponibles

Raúl Molina	15
Cristhian Chulca	15
Días disponibles	30

- Día disponible se toma en cuenta como 8 horas de trabajo

Ítem	Sprint	Responsable	Tareas	Días asignados
1.	Sprint 12	Desarrollador 1 – Raúl Molina	✓ Desarrollo de proyecto Gateway (tarea en conjunto).	4
			✓ Desarrollo de proyecto Discovery Server (Eureka) (tarea en conjunto).	4
			✓ Desarrollo de soluciones para dependencias entre microservicios (tarea en conjunto).	4
			✓ Pruebas de funcionamiento del sistema con llamada a microservicios (tarea en conjunto).	1
			✓ Pruebas finales del sistema parte 1 (tarea en conjunto).	1
			✓ Pruebas finales del sistema parte 2 (tarea en conjunto).	1
2	Sprint 12	Desarrollador 2 – Cristhian Chulca	✓ Desarrollo de proyecto Gateway (tarea en conjunto).	4
			✓ Desarrollo de proyecto Discovery	4

			Server (Eureka) (tarea en conjunto).	
			✓ Desarrollo de soluciones para dependencias entre microservicios (tarea en conjunto).	4
			✓ Pruebas de funcionamiento del sistema con llamada a microservicios (tarea en conjunto).	1
			✓ Pruebas finales del sistema parte 1 (tarea en conjunto)	1
			✓ Pruebas finales del sistema parte 2 (tarea en conjunto).	1

Tabla 17: Sprints backlog definidos en la duodécima reunión de planificación

La Tabla 17 tiene una descripción de tareas para cada desarrollador en el equipo en el tercer sprint, este tendrá una duración de 28 días (4 semanas laborables, 6 horas diarias por desarrollador).

5.2.3. FASE 3: SCRUM DIARIO

COMUNICACIÓN DE SPRINT BACKLOGS

El avance de los Sprints es manejado mediante reuniones de pie diarias, las que se llevaron a cabo personal y telefónicamente, con la participación de los desarrolladores del proyecto. Las reuniones sirvieron para tener un control debido sobre las tareas designadas para los miembros del equipo, como sugiere la metodología el control de tareas se lo ha hecho con un tablero físico en concreto una pizarra y post-its.

CUADRO BURNDOWN (GRÁFICOS DE PROGRESO)

La Fig. 31 evidencia el progreso para el Sprint Backlog 1, este tuvo una duración de 20 días laborables.

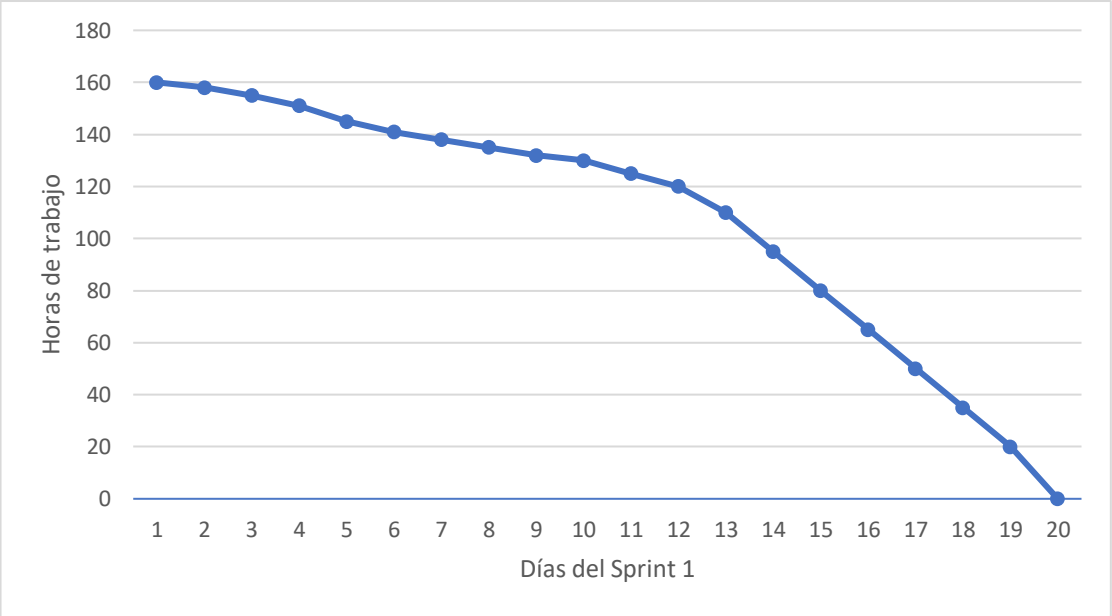


Fig. 31: Diagrama burndown para el Sprint 1

El control del trabajo avanzado se lo ha llevado a cabo mediante las reuniones diarias entre los miembros del equipo, en este primer Sprint la mayoría de reuniones se llevaron a cabo telefónicamente. En la Fig. 31 se denota el avance extenso a partir del día 12 desde el cual existieron reuniones por parte de los desarrolladores.

La Fig. 32 evidencia el progreso para el Sprint Backlog 2, este tuvo una duración de 20 días laborables.

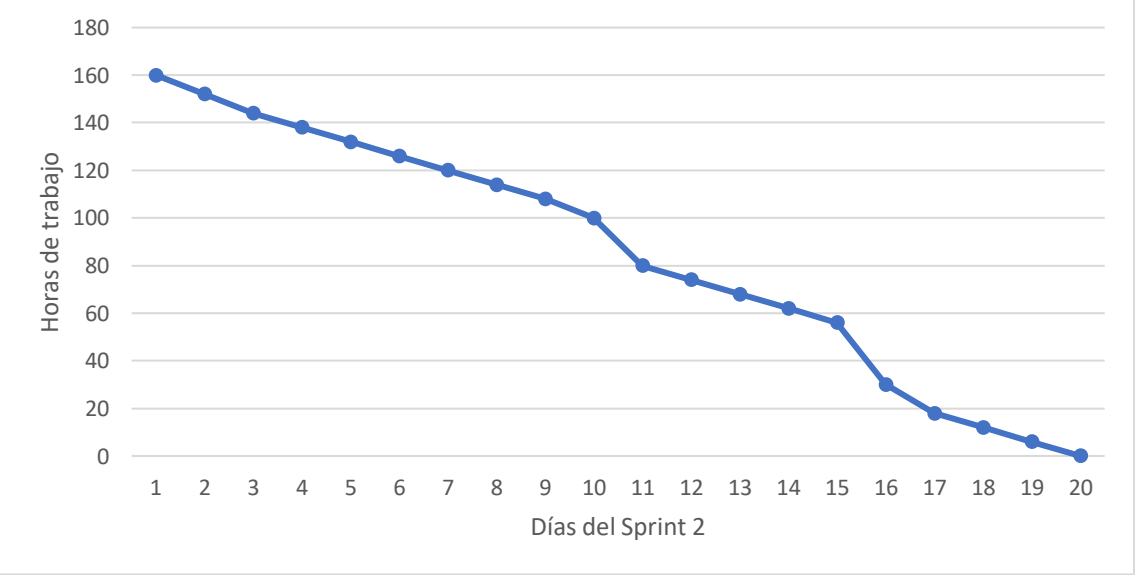


Fig. 32: Diagrama burndown para el Sprint 2

En el Sprint 2 se denota avances extensos durante dos días de trabajo debido a que existieron reuniones los fines de semana para tener un control sobre los retrasos en las tareas en conjunto.

La Fig. 33 evidencia el progreso para el Sprint Backlog 3, este tuvo una duración de 20 días laborables.

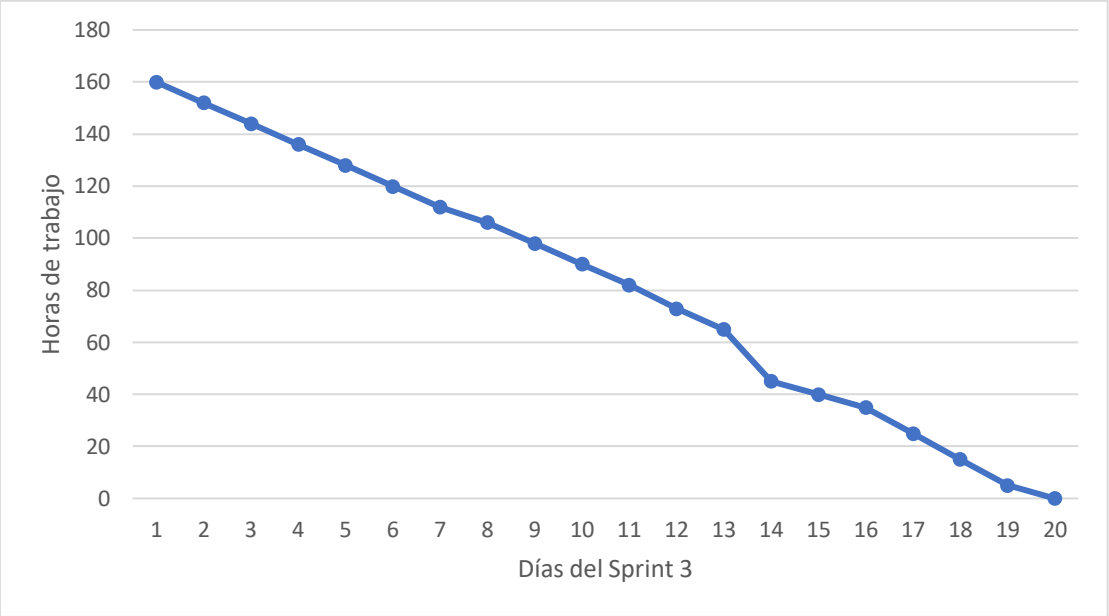


Fig. 33: Diagrama burndown para el Sprint 3

Los avances hasta el día 13 son lineales con las tareas individuales, en los siguientes días existe una variación con las tareas en conjunto específicamente para la selección de las herramientas de desarrollo, avances prolongados debido a los fines de semana y retrasos en 2 días por inconvenientes en la coordinación de tiempos con las labores de los desarrolladores.

La Fig. 34 evidencia el progreso para el Sprint Backlog 4, este tuvo una duración de 18 días laborables.

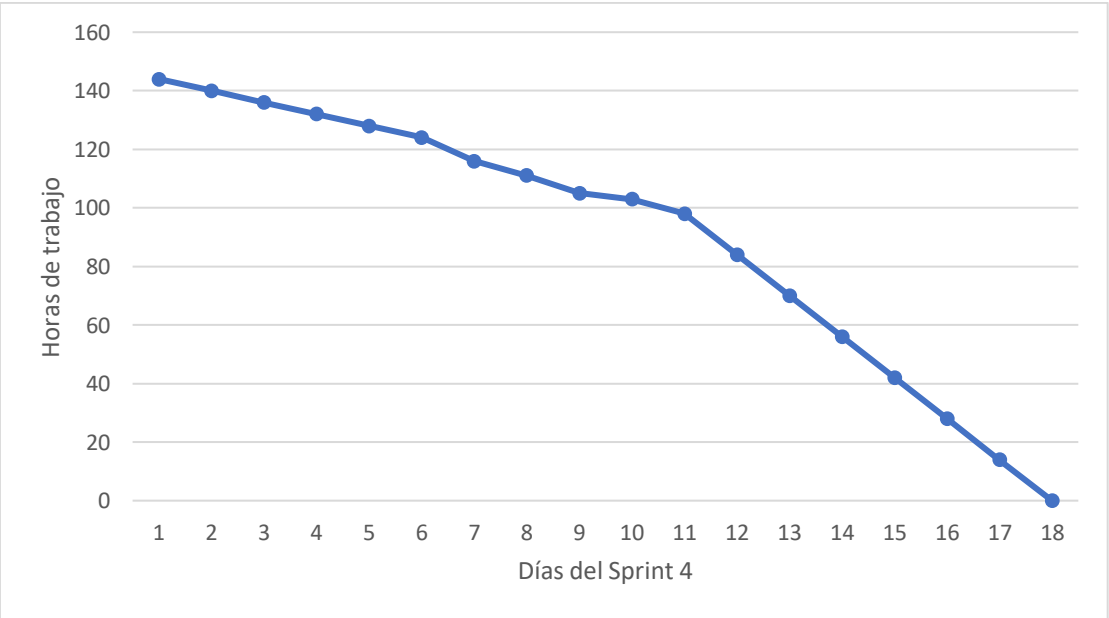


Fig. 34: Diagrama burndown para el Sprint 4

En el Sprint 4 en la tarea de realizar un estudio del orquestador de microservicios de Red Hat y la tarea entorno al IDE Red Hat JBoss Developer Studio existieron retrasos por tratarse de herramientas con total desconocimiento para los desarrolladores, para lograr cumplir con el total de tareas a partir del día 11 se llevaron a cabo reuniones de los desarrolladores.

La Fig. 35 evidencia el progreso para el Sprint Backlog 5, este tuvo una duración de 20 días laborables.

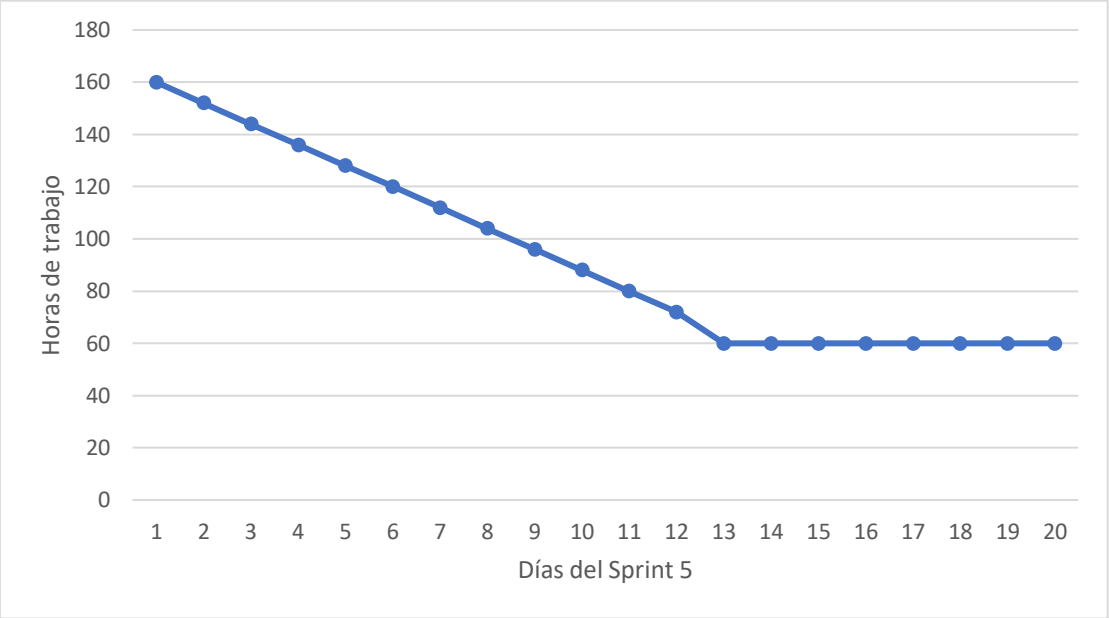


Fig. 35: Diagrama burndown para el Sprint 5

En el Sprint 5 se encontró el problema de la disponibilidad de la herramienta Red Hat Openshift, el tiempo de disponibilidad para el acceso es de 60 días y el equipo decidió que no sería el tiempo suficiente para completar el desarrollo del sistema. En las reuniones finales del Sprint se decidió cambiar la selección de herramientas para el desarrollo del sistema del presente proyecto.

La Fig. 36 evidencia el progreso para el Sprint Backlog 6, este tuvo una duración de 20 días laborables.

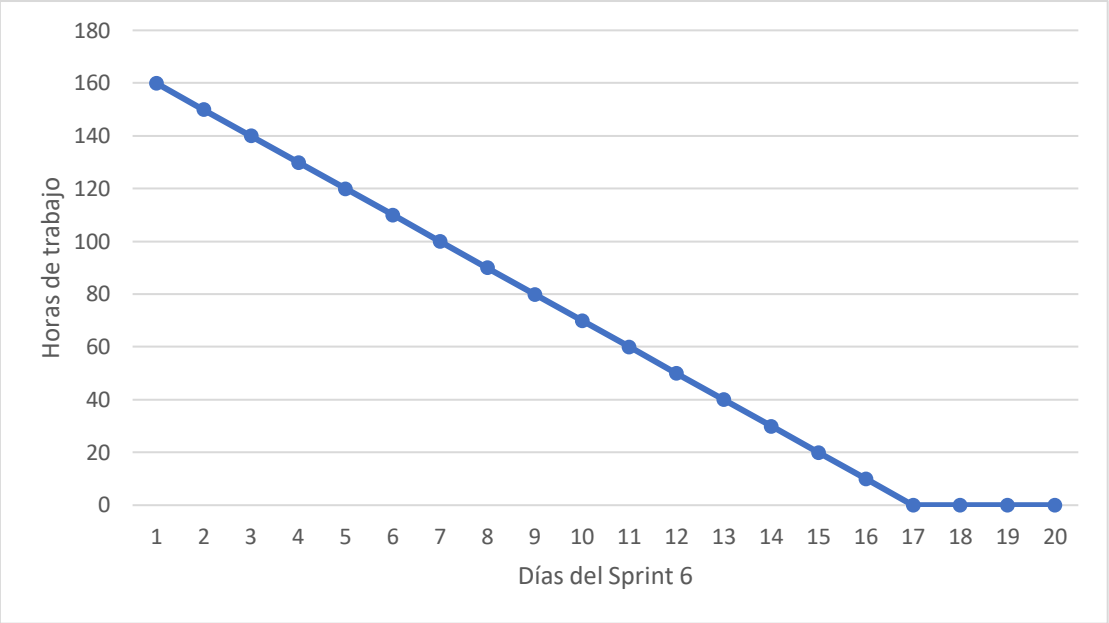


Fig. 36: Diagrama burndown para el Sprint 6

En el Sprint 6 las tareas asignadas tuvieron un avance mayor al planeado porque en un Sprint inicial se realizaron revisiones de herramientas posibles para el desarrollo, para la selección de los frameworks Spring y Angular se hicieron estudios que ya se habían iniciado. El equipo decidió los días finales disponibles del Sprint realizar instalaciones en conjunto y revisiones entorno a la disponibilidad de tiempo de los nuevos frameworks seleccionados.

La Fig. 37 evidencia el progreso para el Sprint Backlog 7, este tuvo una duración de 30 días laborables.

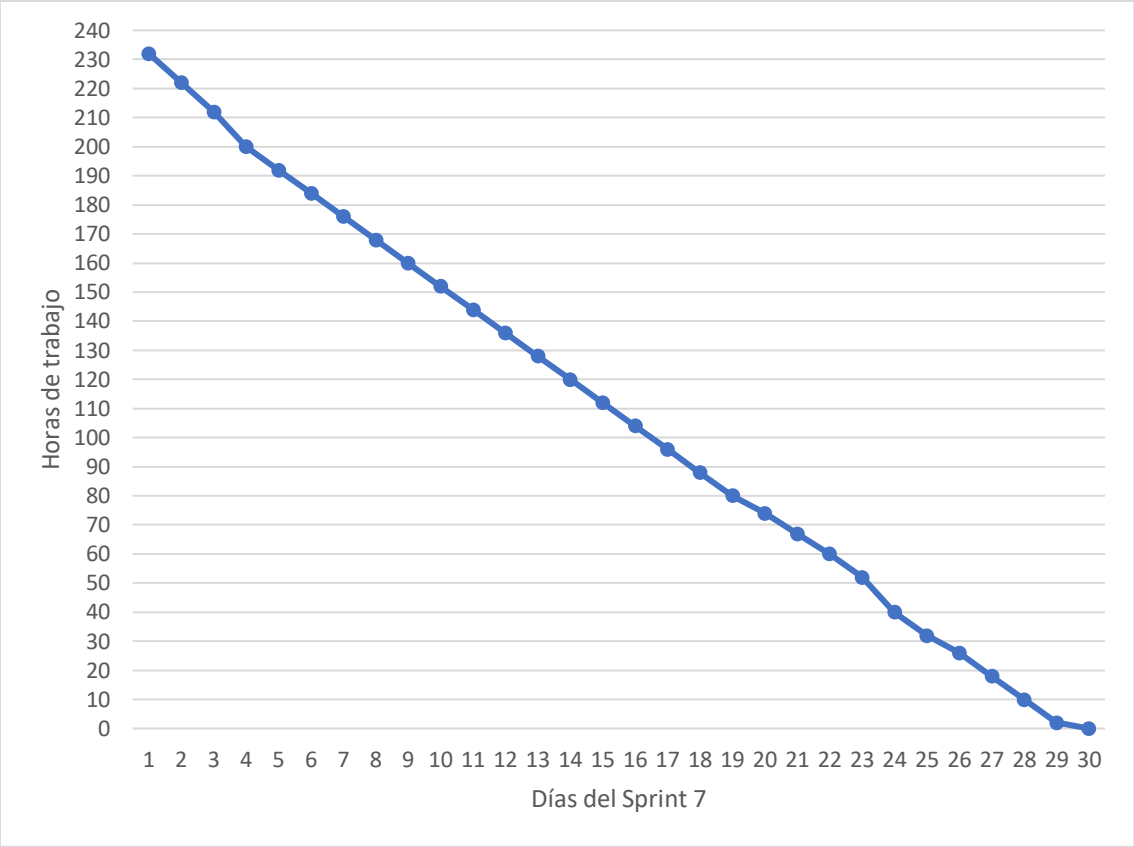


Fig. 37: Diagrama burndown para el Sprint 7

El Sprint 7 fue el primero en tener una extensión de días disponibles a 30 y el número de horas de trabajo a 240, por el cambio de herramientas de desarrollo, pero el número días laborables se mantuvo en 28 porque se incrementó la cantidad de horas de trabajo al día.

La Fig. 38 evidencia el progreso para el Sprint Backlog 8, este tuvo una duración de 30 días laborables.

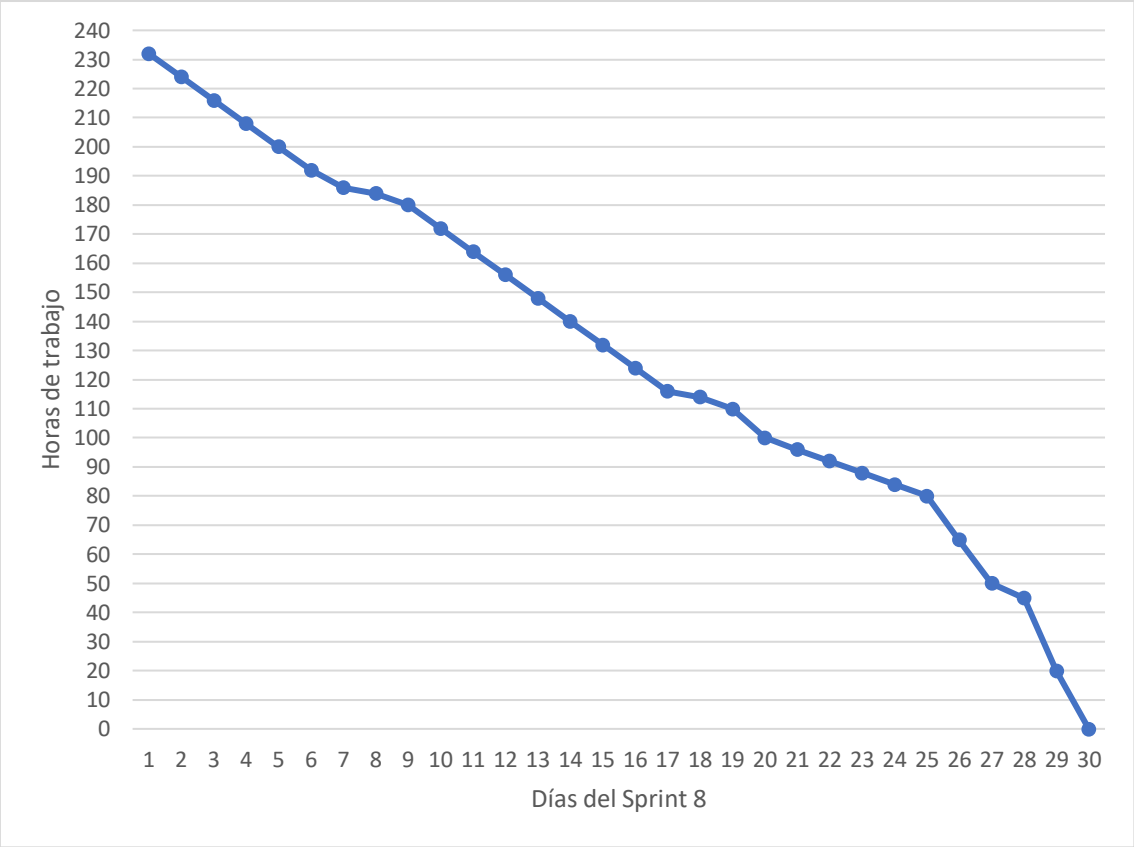


Fig. 38: Diagrama burndown para el Sprint 8

En el Sprint 8 existieron problemas que demoraron el desarrollo normal dentro de las tareas de servicios y controladores para los microservicios 1 y 2, uno de ellos el tipo de clave primaria de las tablas en la base de datos actual del sistema SisLab. Para las pruebas de los servicios Rest generados se utilizó la herramienta Postman, de esta se tenía un previo conocimiento por parte de los desarrolladores, pero la planificación de las mismas tuvo mayor tiempo por dependencias entre los modelos de los microservicios.

La Fig. 39 evidencia el progreso para el Sprint Backlog 9, este tuvo una duración de 20 días laborables.

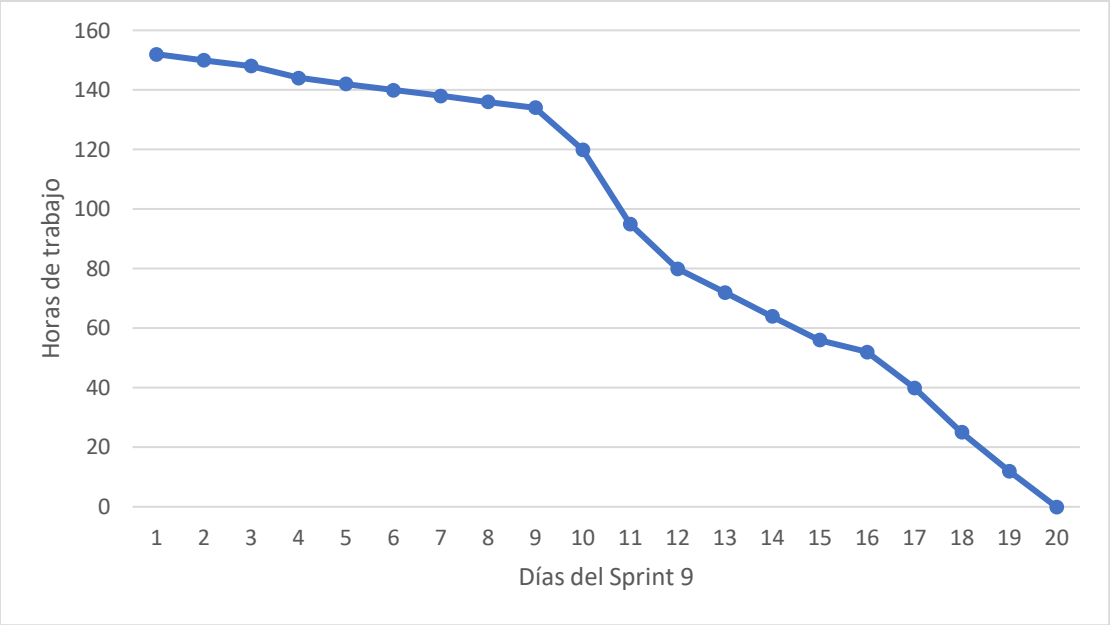


Fig. 39: Diagrama burndown para el Sprint 9

El Sprint 9 tuvo problemas con las tareas de creación y configuración del proyecto front-end en el framework Angular y por esto existieron retrasos los primeros días del Sprint, el desarrollo de componentes se pudo realizar con prontitud debido a un conocimiento ya adquirido con anterioridad por parte de los desarrolladores del equipo. El desarrollo de modelos para el microservicio 3 contó con un avance rápido por la experiencia adquirida en los microservicios anteriores.

La Fig. 40 evidencia el progreso para el Sprint Backlog 10, este tuvo una duración de 20 días laborables.

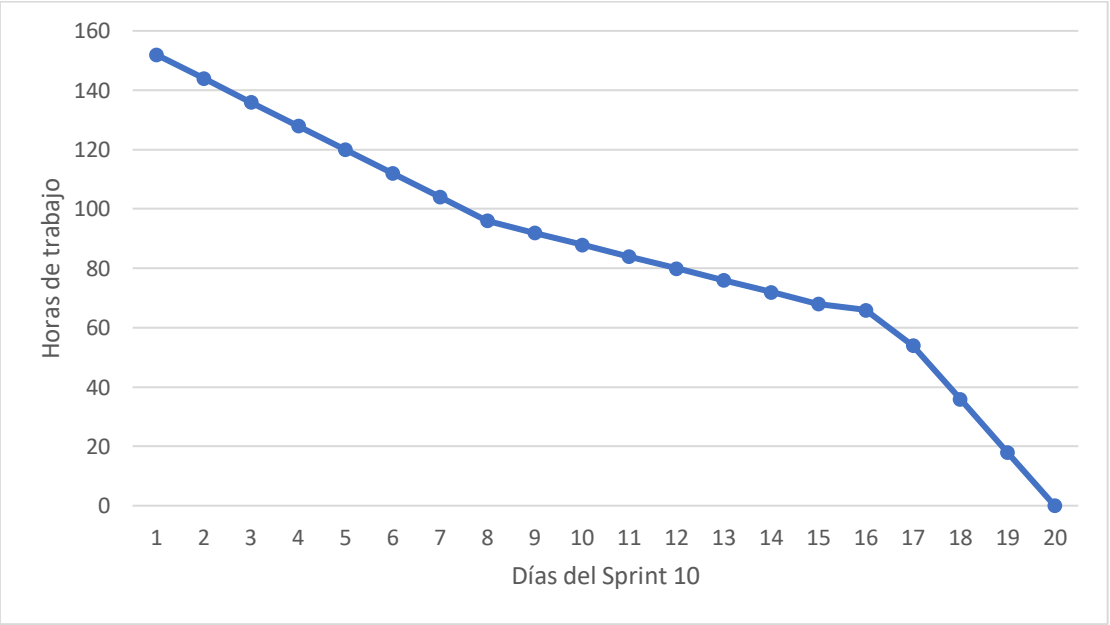


Fig. 40: Diagrama burndown para el Sprint 10

En el Sprint 10 existieron tareas para el microservicio 3 que ya se realizaron para los microservicios 1 y 2, estas fueron desarrolladas sin retrasos en el tiempo. Los retrasos

observados en la gráfica son debidos a consultas que se hicieron para poder realizar las pruebas de componentes CRUD para el microservicio 1.

La Fig. 41 evidencia el progreso para el Sprint Backlog 11, este tuvo una duración de 20 días laborables.

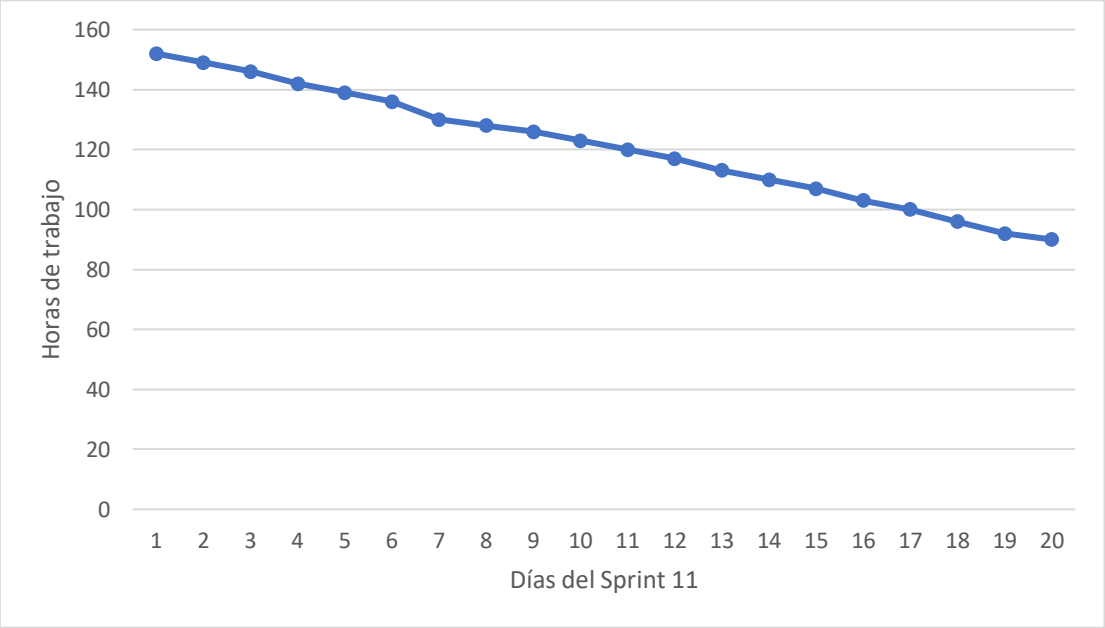


Fig. 41: Diagrama burndown para el Sprint 11

En el presente Sprint 11 no se logró concluir las tareas entorno a la creación y configuración de los proyectos Gateway y Discovery Server.

5.2.4. FASE 4: REVISIÓN DEL SPRINT

El producto entregable para cada Sprint está definido en las tareas asignadas a los desarrolladores en las reuniones de planning. En el presente documento se detallan evidencias mediante imágenes de las tareas que representan creación de proyectos, configuración de proyectos o desarrollo de código, las tareas que se enfocaron a la consulta de tecnologías (IDE's, frameworks, documentación), revisión de tesis base o revisión de código fueron cumplidas, pero no se presentarán pruebas en el presente documento.

Tareas Sprint 1:

- Realizar una revisión y análisis de la tesis, Propuesta Metodológica Para Migración De Sistemas Web Con Arquitectura Monolítica Hacia Una Arquitectura Basada En Microservicios[19].

- Realizar un estudio del modelo de arquitectura detallado en la tesis y realizar una propuesta propia enfocada en las funcionalidades actuales del sistema SisLab descritas en el documento de tesis.
- Revisar información, roles y obligaciones en la metodología Scrum para la designación en la siguiente reunión.

Las revisiones de la tesis base fueron llevadas a cabo con éxito con ayuda de la entrega de documentación de la misma por parte del PO del equipo, PhD. Pamela Flores. En un inicio los desarrolladores cumplieron las tareas por separado tomando notas propias como se muestra en Fig. 42, y después se llevaron a cabo reuniones para la discusión y toma de decisiones entorno al diseño de arquitectura del sistema en desarrollo.

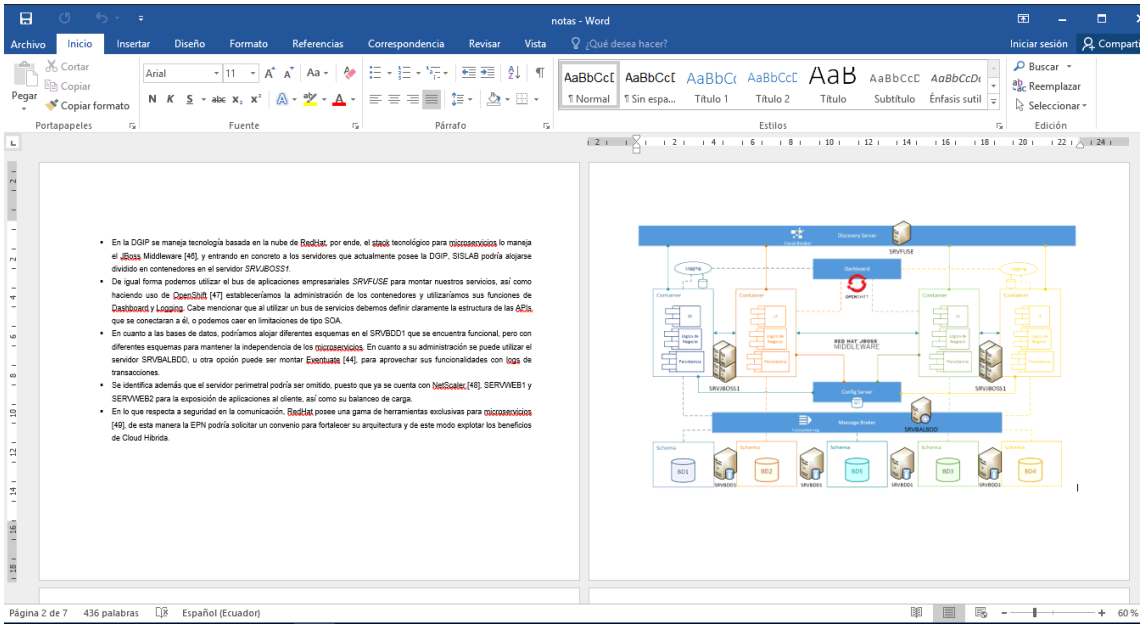


Fig. 42: Ejemplo de revisión de tesis base

Tareas Sprint 2:

- Definir un plan de desarrollo entorno a la tesis base y la propuesta personal de los miembros del equipo de desarrollo
- Designar roles dentro del equipo para cumplimiento con metodología scrum.
- Definir un modelo de arquitectura para el desarrollo de back-end y la conexión con front-end.
- Realizar un estudio de herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.
- Revisión inicial de código actual del sistema SisLab para un entendimiento de la estructura.

La designación de roles para la metodología scrum y la definición de un plan de desarrollo se encuentran descritas y justificadas en el presente documento en 5.2.1. FASE 1: DEFINICIÓN DEL BACKLOG DEL PRODUCTO, de igual manera el estudio de tecnologías y herramientas para el desarrollo del sistema en 5.1.1. Herramientas disponibles para desarrollo de microservicios. Se llevaron a cabo reuniones con la institución propietaria del sistema a desarrollar DGIP, en estas se pudo facilitar el código del sistema actual como se muestra en Fig. 43, y tener una reunión de exposición acerca de su funcionamiento.

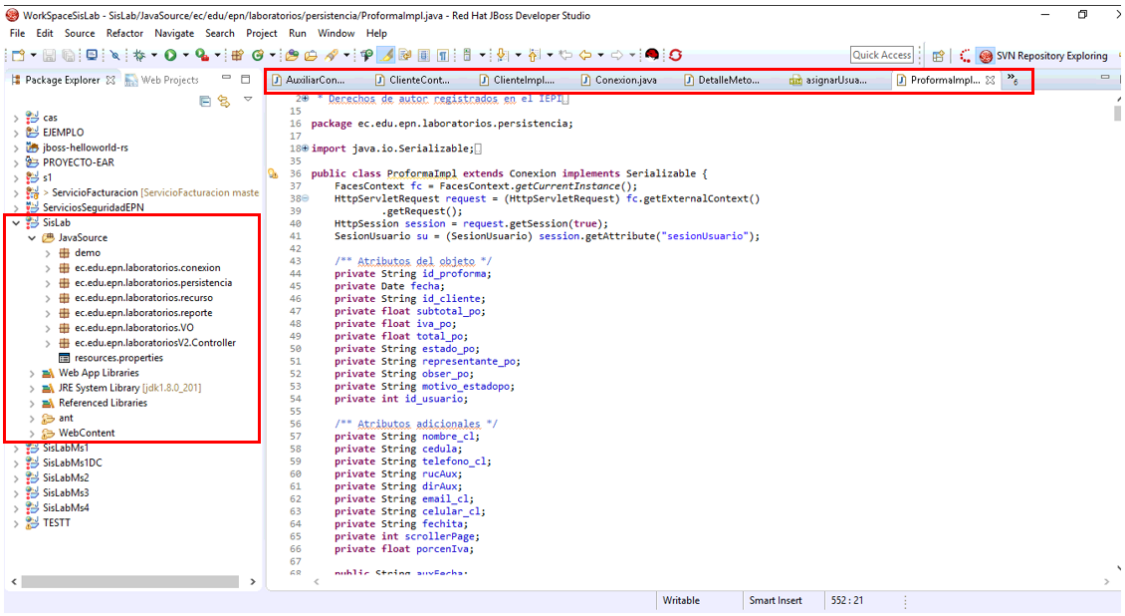


Fig. 43: Código actual de SisLab

Tareas Sprint 3:

- Seleccionar herramientas y frameworks para el desarrollo de la aplicación en back-end y front-end.
- Estudio a profundidad de código actual del sistema SisLab y separación de clases bean para cada microservicio dentro de la arquitectura diseñada.
- Conexión a red interna de DGIP de manera remota mediante VPN.
- Trabajar con los roles y obligaciones designados de la metodología scrum para planificar posibles cambios en el siguiente sprint.

La tarea entorno a herramientas y framework se detalla en 5.1.2. Herramientas seleccionadas, el trabado con el código fue llevado a cabo y se lo puede observar en:

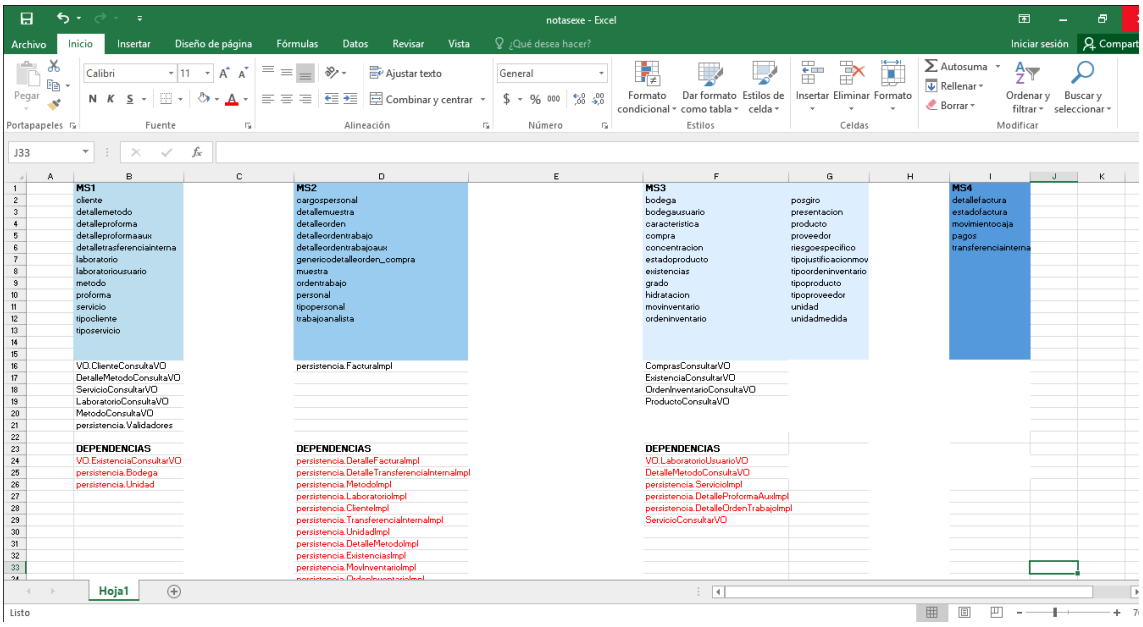


Fig. 44: Separación de clases Bean y dependencias

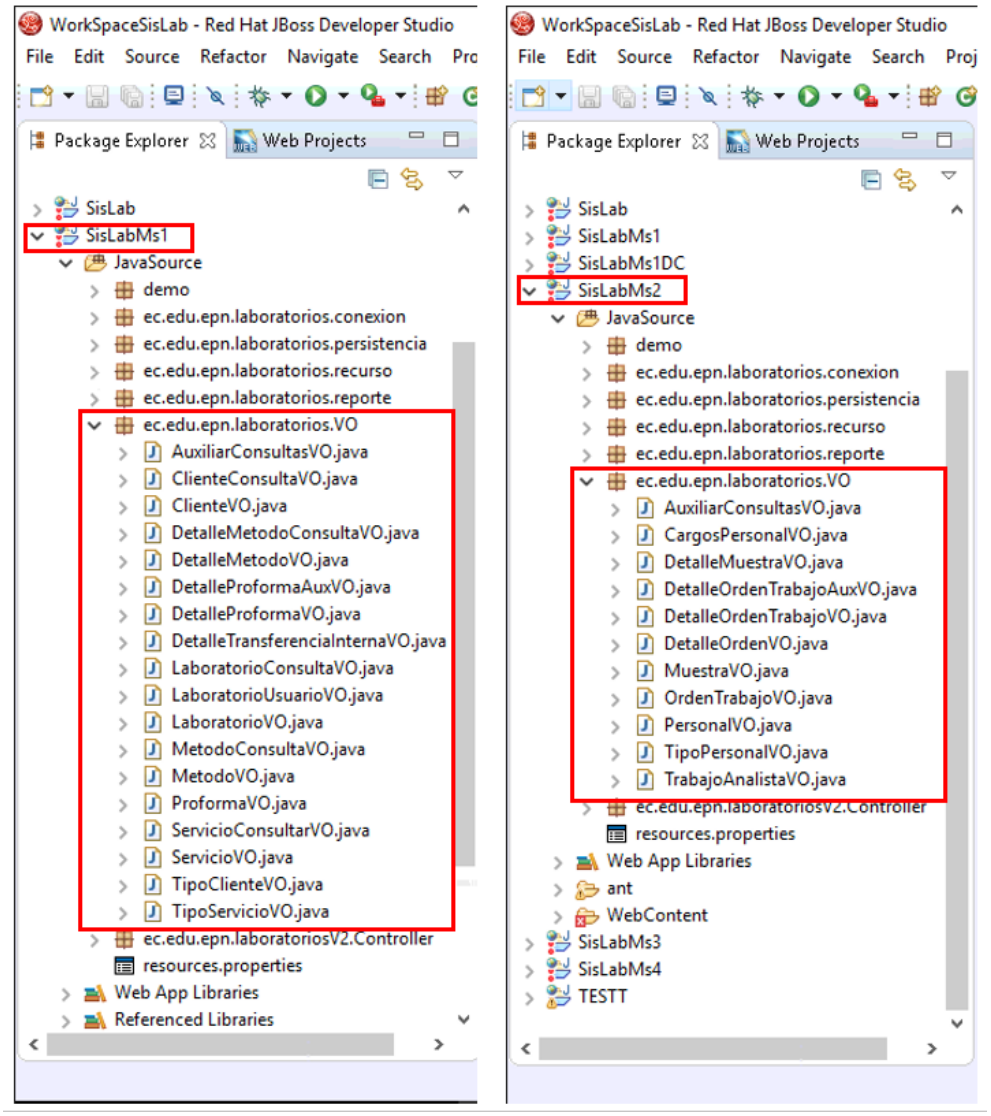


Fig. 45: Clases Bean Microservicio 1 y Microservicio 2

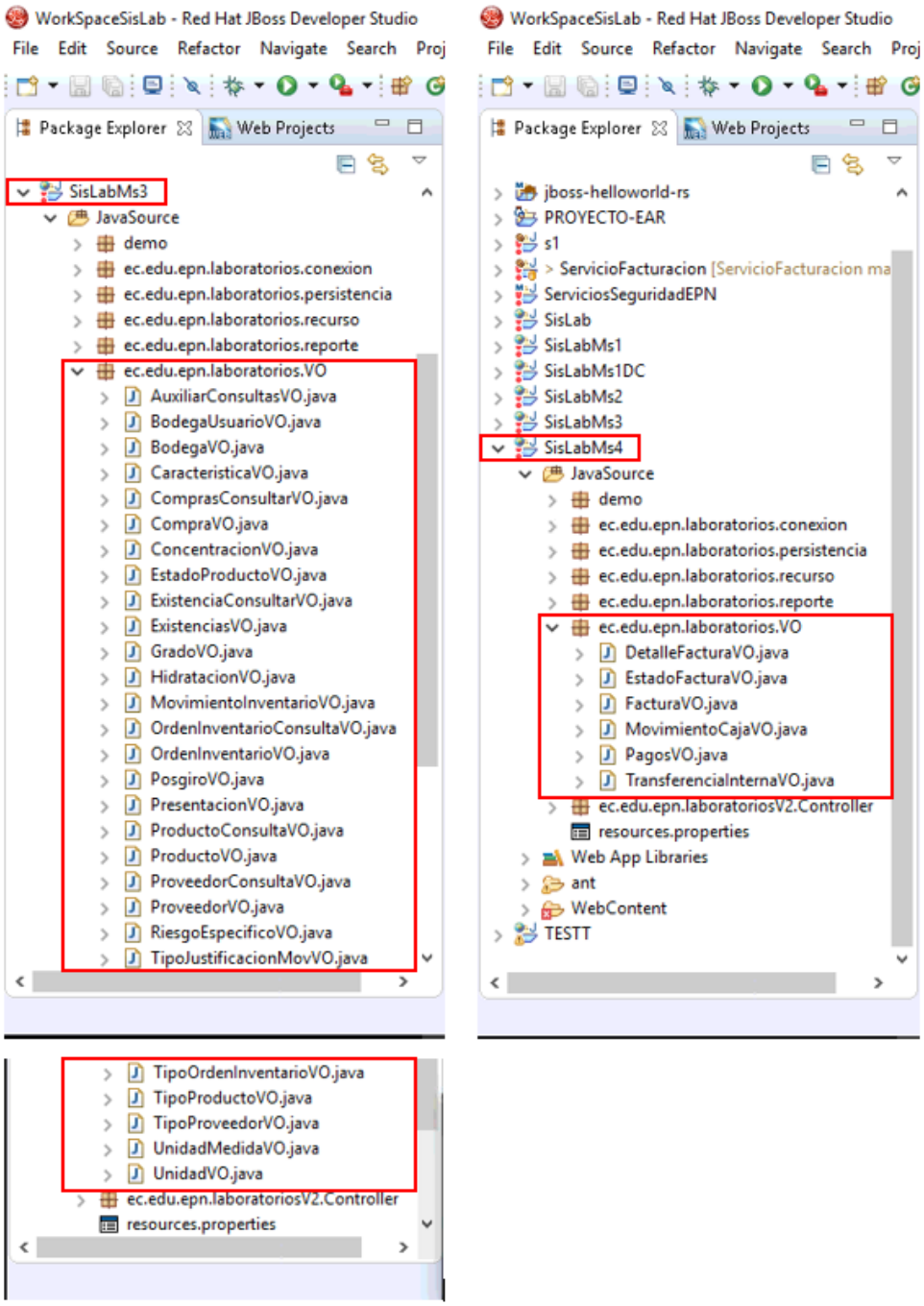


Fig. 46: Clases Bean Microservicio 3 y Microservicio 4

Tareas Sprint 4:

- Instalar y configurar de IDE Red Hat JBoss Developer Studio.
- Crear 4 proyectos en IDE, uno para cada microservicio.
- Copiar backup de base de datos de sistema SisLab y obtener metadata de cada tabla.
- Instalar y configurar sistema gestor de base de datos PostgreSQL.
- Realizar un estudio del orquestador de microservicios disponible en Red Hat.

La instalación y configuración del IDE se llevó a cabo en conjunto para que los desarrolladores tengan bajo las mismas características el IDE en sus respectivas máquinas y la creación de los proyectos se puede observar en Fig. 45 y Fig. 46. Las tareas con la base de datos se trabajaron en una reunión con el ingeniero Roberto García, quien facilitó el acceso y extendió una capacitación del funcionamiento actual de las tablas en la base de datos (base de datos descomprimida se puede ver en Fig. 47). La revisión del orquestador de servicios disponible en Red Hat se la hizo en línea y se encontraron problemas descritos en Disponibilidad de RED HAT OPENSHIFT.

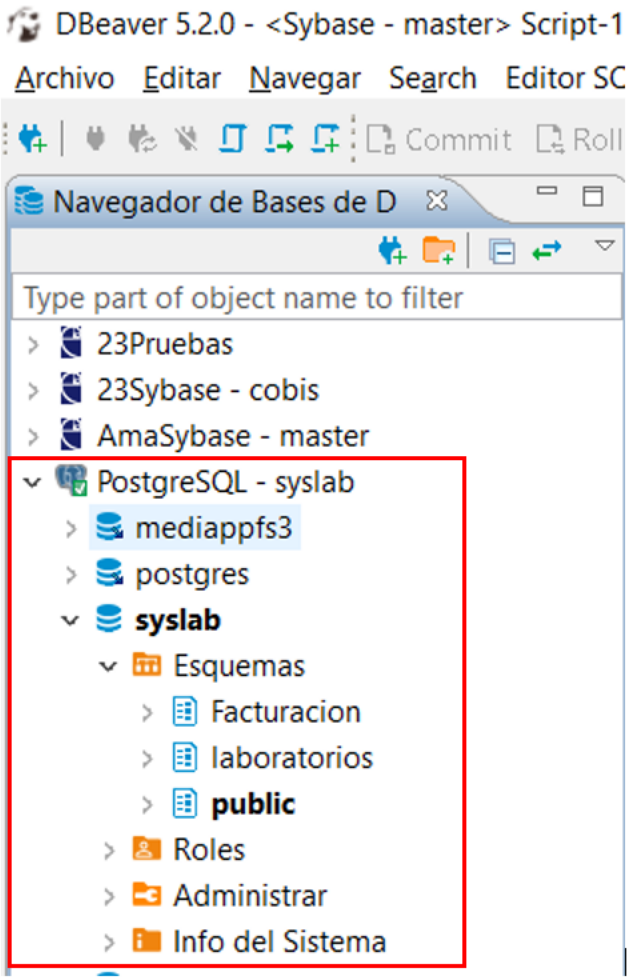


Fig. 47: Backup descomprimido de Base de Datos de SisLab

Tareas Sprint 5:

- Copiar archivos de objetos Beans a proyectos de microservicios 1 y 2 en IDE Red Hat.
- Revisar creación y configuración de proyecto Openshift para la llamada y orquestación de microservicios.
- Copiar archivos JSF (Java Server Faces) a proyectos de microservicios 1 y 2 en IDE Red Hat.

- Revisar información, instalación y configuración de Kubernetes o contenedores en Red Hat.

Las tareas del sprint no se concluyeron debido a los problemas encontrados con OpenShift, los desarrolladores centraron el tiempo en encontrar posibles vías de solución, en el final del sprint se decidió aplicar el cambio de framework e IDE de desarrollo.

Tareas Sprint 6:

- Revisión de framework Spring, instalación, configuración, interfaz para conexión con base de datos y front-end, JPA, Eureka.
- Descarga, instalación y configuración del IDE disponible en el framework, Spring Tools Suite.
- Revisión de framework Angular, IDE para desarrollo, instalación de herramientas necesarias.

Las tareas para el sprint se completaron conforme al planning, los IDE's seleccionados tienen documentación en línea que sirvió para su instalación y configuración.

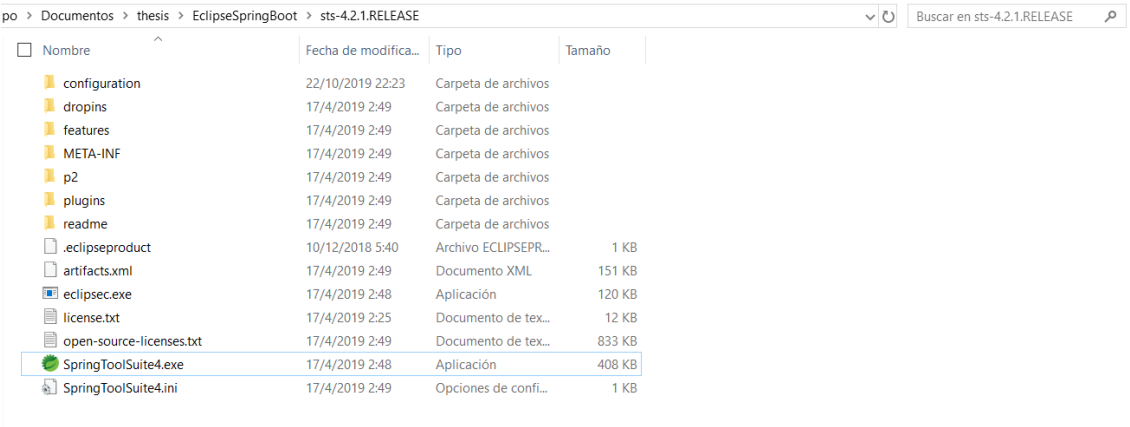


Fig. 48: Descarga e instalación de Spring Tools Suite

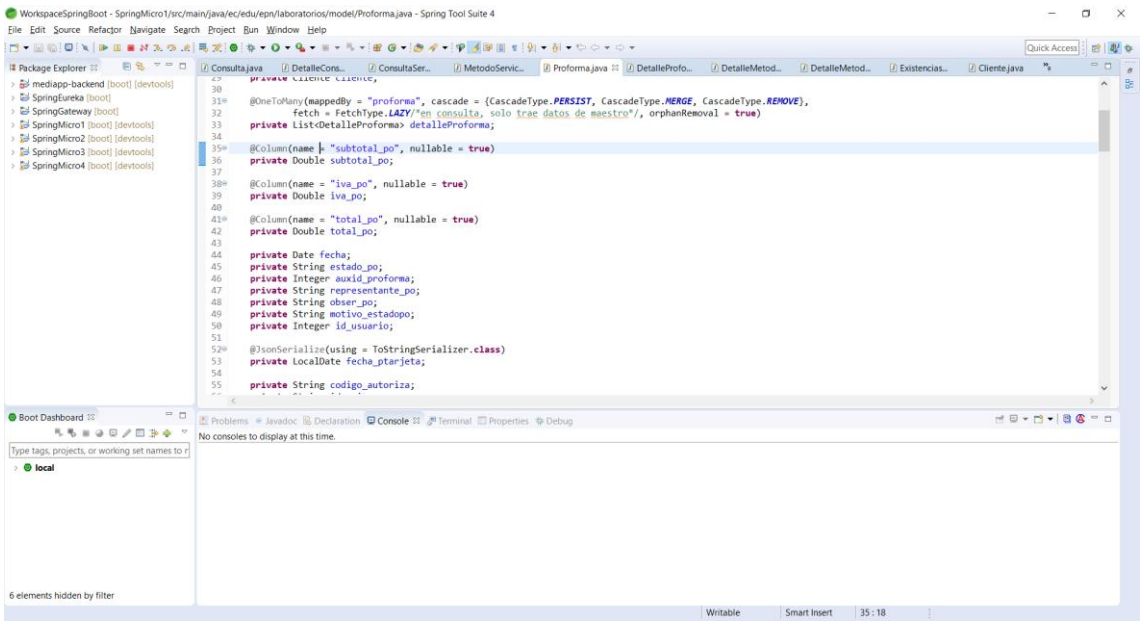


Fig. 49: IDE Spring Tools Suite

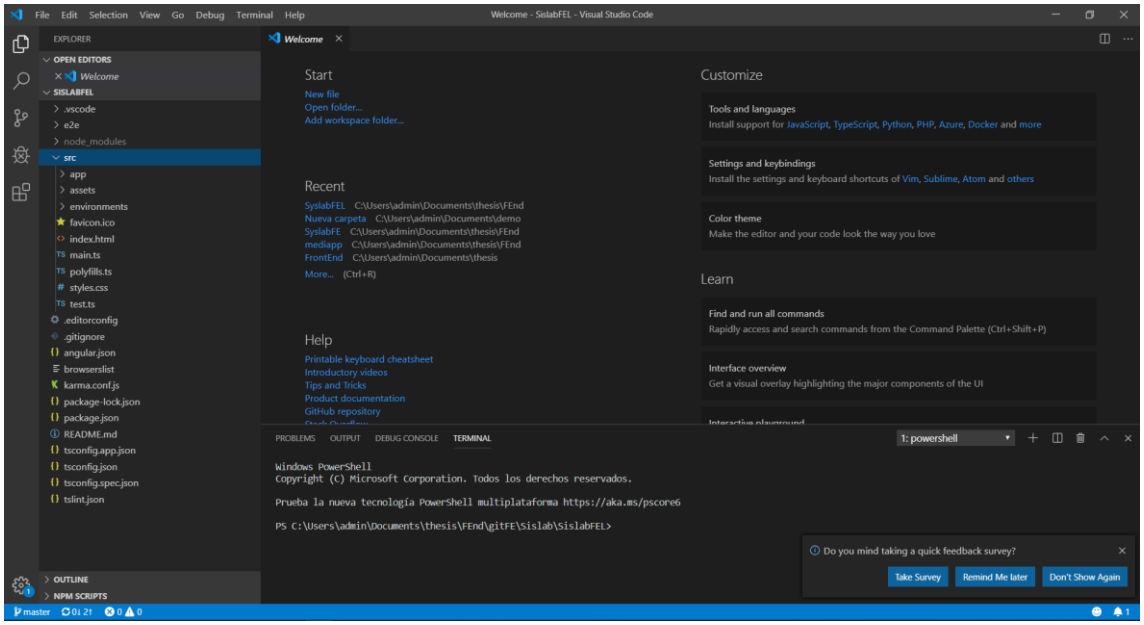


Fig. 50: IDE Visual Studio Code

Tareas Sprint 7:

- Crear y configurar proyectos en IDE Spring Tools Suite para microservicios 1 y 2.
- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 1.
- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 2.
- Generación de scripts de tablas integrantes de microservicio 1.
- Generación de scripts de tablas integrantes de microservicio 2.

Las tareas fueron completadas en los tiempos planificados, tomando en cuenta que las tareas de desarrollo de modelos en proyectos Spring tuvieron como dependencia las tareas de generación de scripts.

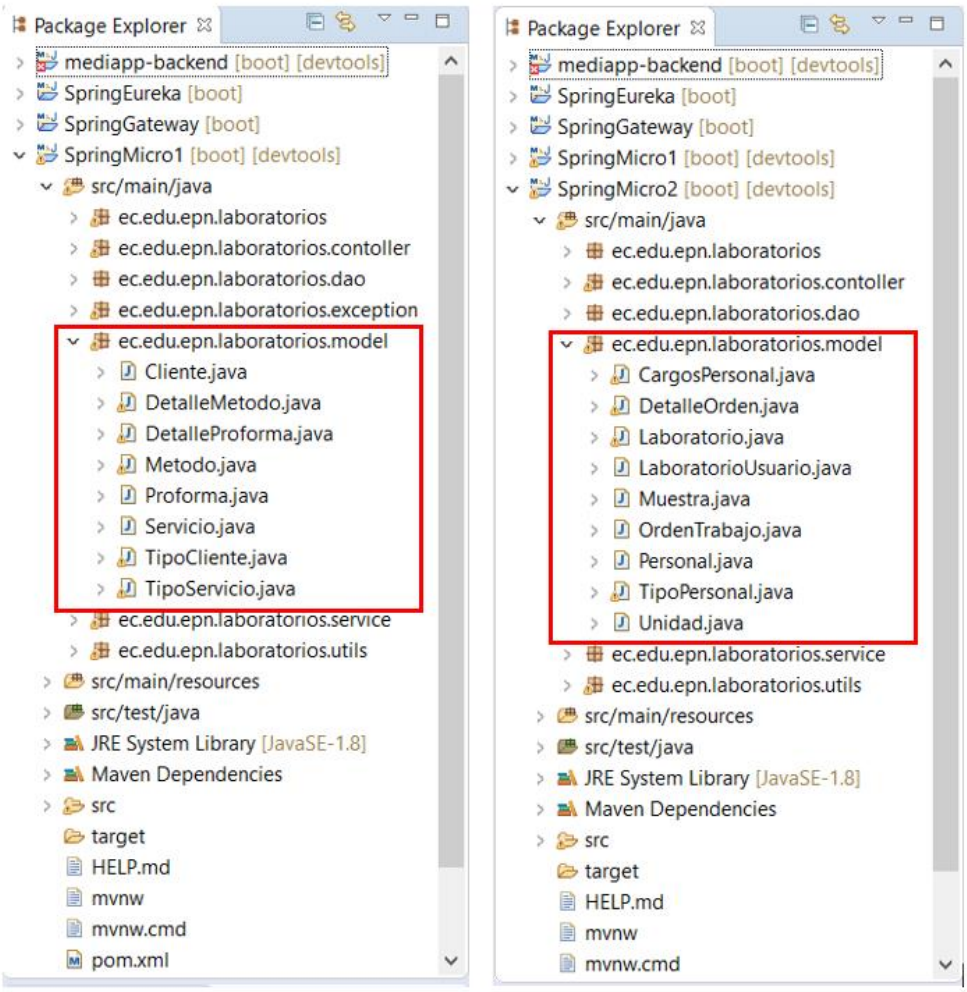


Fig. 51: Modelos en proyectos Microservicio 1 y Microservicio 2

```
CREATE TABLE "Laboratorios".cliente (
    id_cliente varchar(10) NOT NULL DEFAULT nextval('"Laboratorios".secuencia_cliente'::regclass),
    cedula bpchar(10) NULL,
    id_tipocliente varchar(10) NOT NULL,
    ruc_cl bpchar(13) NULL,
    nombre_cl bpchar(200) NOT NULL,
    direccion_cl bpchar(100) NOT NULL,
    telefono_cl bpchar(15) NULL,
    otrofono_cl bpchar(15) NULL,
    celular_cl bpchar(15) NULL,
    fax_cl bpchar(15) NULL,
    email_cl bpchar(50) NULL,
    contacto_cl bpchar(100) NULL,
    nombre_usuario varchar(30) NULL,
    fechatrans timestamp NULL,
    pasaporte_cl varchar(20) NULL,
    CONSTRAINT pk_cliente PRIMARY KEY (id_cliente),
    CONSTRAINT fk_cliente_cliente_t_tipoclie FOREIGN KEY (id_tipocliente) REFERENCES "Laboratorios".tipocliente(id_tipocliente) ON UPDATE
    RESTRICT ON DELETE RESTRICT
);
COMMENT ON TABLE "Laboratorios".cliente IS 'Clientes de la EPN';

-- Permissions

ALTER TABLE "Laboratorios".cliente OWNER TO postgres;
GRANT ALL ON TABLE "Laboratorios".cliente TO postgres;
```

Fig. 52: Ejemplo de script de tabla de base de datos

Tareas Sprint 8:

- Desarrollo de servicios y controladores en back-end para microservicio 1.
- Desarrollo de servicios y controladores en back-end para microservicio 2.
- Crear secuenciales para microservicio 1 y 2.
- Pruebas de servicios Rest en back-end microservicio 1.
- Pruebas de servicios Rest en back-end microservicio 2.

Los microservicios publican sus API REST en back-end mediante el patrón de arquitectura Modelo Vista Controlador MVC, en el presente sprint se desarrollaron los servicios y controladores para los microservicios 1 y 2 para tener servicios Create, Read, Update, Delete (CRUD) disponibles de los modelos. En la actual base de datos del sistema SisLab se manejan secuenciales generados para las Primary Key PK en las tablas, para esto se implementaron secuenciales manuales en JPA.

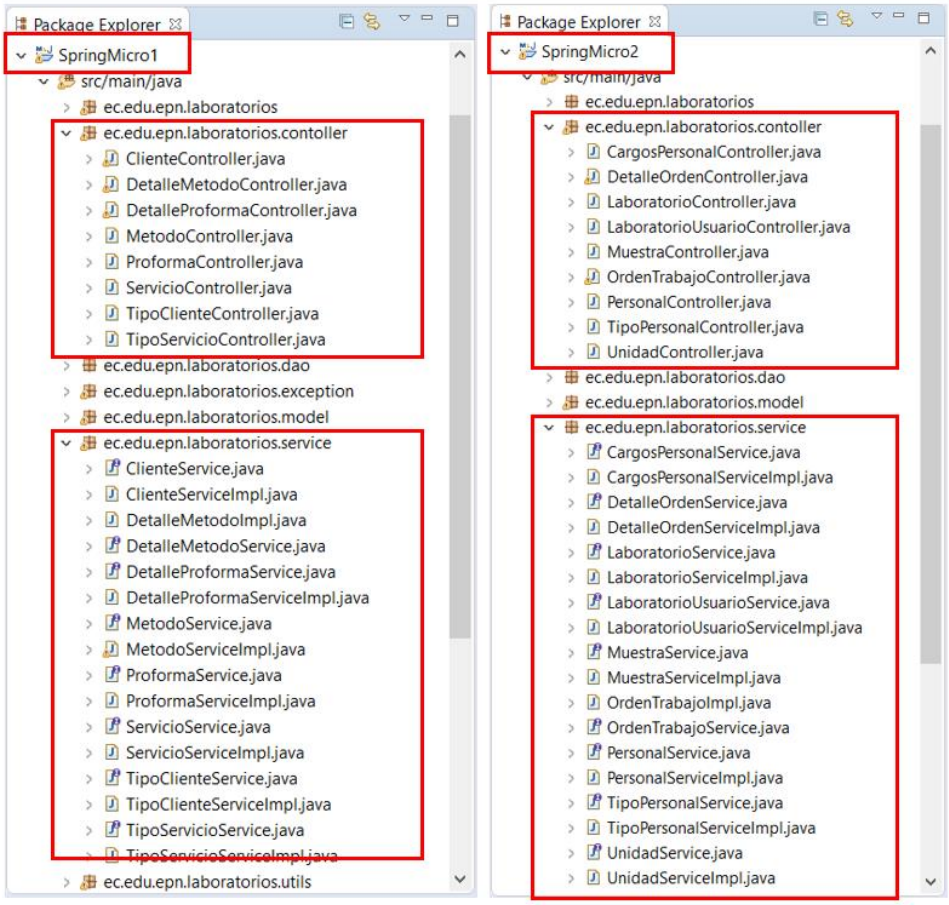


Fig. 53: Servicios y controladores en proyectos Microservicio 1 y Microservicio 2

```java
public class MyGenerator implements Configurable, IdentifierGenerator {

    private String sequenceName;

    @Override
    public Serializable generate(SessionImplementor arg0, Object arg1) throws HibernateException {

        Connection connection = (Connection) arg0.connection();
        PreparedStatement ps = null;
        String consultaSecuencia = "select nextval ('\"laboratorios\"." + sequenceName + "')";
        try {
            ps = connection.prepareStatement(consultaSecuencia);
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                return rs.getString(1);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public void configure(Type type, Properties params, ServiceRegistry serviceRegistry) throws MappingException {
        sequenceName = params.getProperty("sequenceName");
    }

}

@Table(name = "cliente")
public class Cliente {

    @Id
    @GeneratedValue(generator = "Clientegenerator")
    @GenericGenerator(name = "Clientegenerator",
                      parameters = @Parameter(name = "sequenceName", value = "secuencia_cliente"),
                      strategy = "ec.edu.epn.laboratorios.utils.MyGenerator")
    private String id_cliente;
```

Fig. 54: Generador de secuenciales y ejemplo de implementación

Tareas Sprint 9:

- Creación y configuración de proyecto front-end en framework Angular.
- Desarrollo de componentes CRUD en front-end para microservicio 1.
- Desarrollo de componentes CRUD en front-end para microservicio 2.
- Desarrollo de modelos de base de datos en proyecto Spring para microservicio 3.

El proyecto en angular fue desarrollado en el IDE Visual Studio Code con la configuración de:

- ✓ Angular CLI: 8.0.6
- ✓ Node: 10.16.0
- ✓ OS: win32 x64
- ✓ Angular: 8.0.3

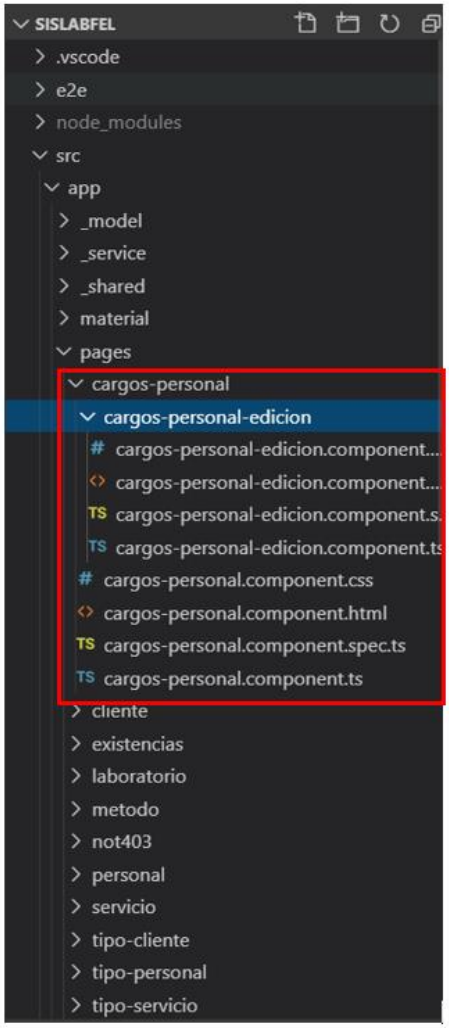


Fig. 55: Componentes de Microservicio 1 - Componente CRUD ejemplo

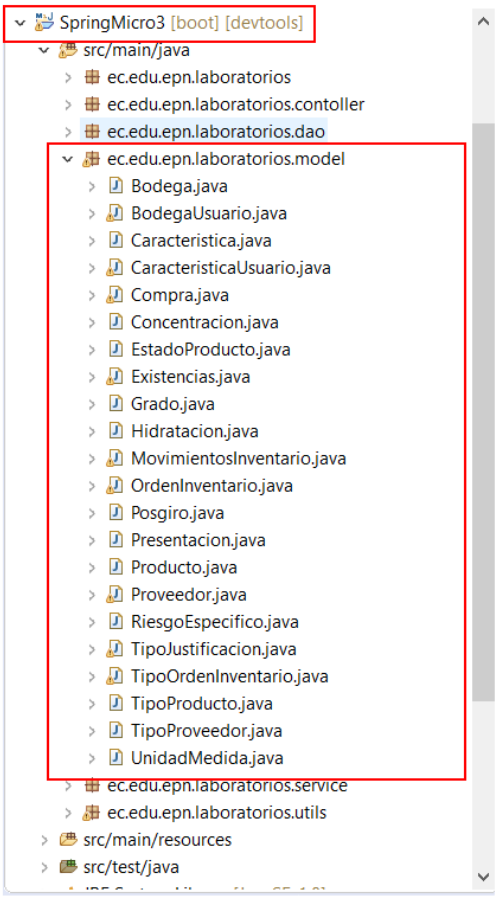


Fig. 56: Modelos en proyecto Microservicio 3

Tareas Sprint 10:

- Desarrollo de servicios y controladores en back-end para microservicio 3.
- Generación de scripts de tablas integrantes de microservicio 3.
- Pruebas de servicios Rest en back-end microservicio 3.
- Pruebas de componentes CRUD en front-end de microservicio 1.

Tareas Sprint 11:

- Desarrollo de componentes CRUD en front-end para microservicio 3.
- Pruebas de componentes CRUD en front-end de microservicio 2.
- Pruebas de servicios Rest microservicio 3.
- Creación y configuración de proyecto Gateway, Eureka.
- Estudio y solución de dependencias entre microservicios.
- Pruebas de sistema en front-end.

Tareas Sprint 12:

- Escritura de documento de tesis.
- Desarrollo de proyecto Gateway.
- Desarrollo de proyecto Discovery Server (Eureka).
- Desarrollo de soluciones para dependencias entre microservicios.
- Pruebas de funcionamiento del sistema con llamada a microservicios.
- Pruebas finales del sistema parte 1.
- Pruebas finales del sistema parte 2.

5.2.5. FASE 5: RETROSPECTIVA DEL SPRINT

Los Sprint tuvieron tiempo dedicado para reuniones de retrospectiva, para estas se siguieron indicaciones por parte del especialista en Scrum Cristian León, se aplicaron las siguientes preguntas:

1. ¿Qué deberíamos empezar a hacer al siguiente sprint?
2. ¿Qué deberíamos hacer más?
3. ¿Qué deberíamos continuar haciendo durante el siguiente sprint?
4. ¿Qué deberíamos hacer menos o dejar de hacer?

Las reuniones de retrospectiva tuvieron una duración aproximada de 30 minutos, mediante las respuestas a las preguntas se llegaron a acuerdos en el equipo para resolver problemas de manejo de tiempo en su mayoría, los cuales eran planificar más tiempo en desarrollos o investigaciones en conjunto por parte de los desarrolladores. En los sprint 5 y 6 existió mayor novedad y se acordaron medidas que requerían un mayor esfuerzo esto debido a el problema que se encontró con la tecnología Openshift.

5.3. Arquitectura final del sistema

El esquema de funcionamiento está descrito como una aplicación web con arquitectura SPA (Single Page Applications) llamadas también aplicaciones de una sola página, debido a que la lógica de los componentes de una página con sus recursos (archivos HTML, CSS, JS) se encuentra en el navegador y ésta no es recargada durante su uso, los cambios dentro de la página web son construidos utilizando el lenguaje de programación JavaScript.

El servidor y sus componentes proporcionan un API REST al código del cliente, la comunicación entre servidor y base de datos se realiza mediante JPA (Java Persistence API) con el leguaje de consulta JPQL (Java Persistence Query Language), la arquitectura del sistema en desarrollo de muestra en Fig. 57.

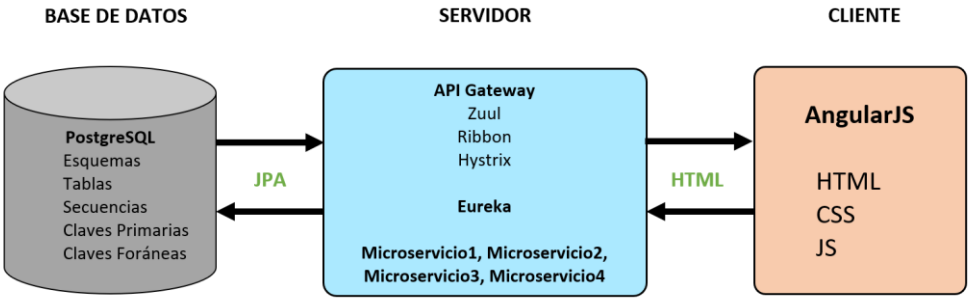


Fig. 57: Arquitectura de SisLab en desarrollo

El ciclo de vida de una página en SPA se detalla en Fig. 58, en una arquitectura distinta se detalla en Fig. 59.

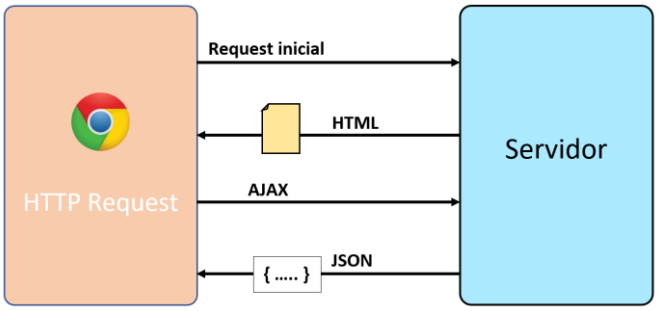


Fig. 58: Ciclo de vida de página en SPA

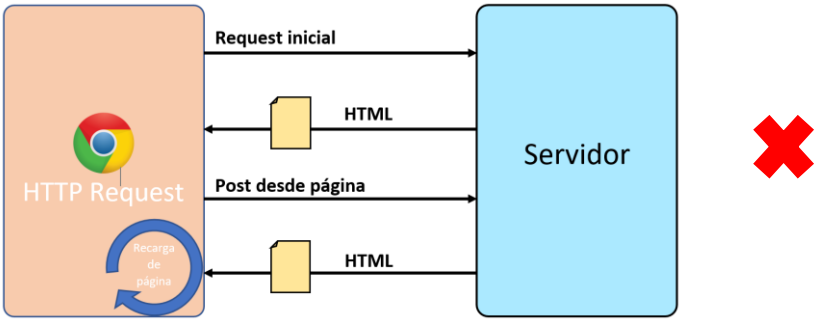


Fig. 59: Ciclo de vida de página en arquitecturas antiguas

5.3.1. Arquitectura en servidor

El servidor (como se muestra en Fig. 60) tiene componentes internos para la exposición de la Api Rest a front-end y la comunicación con la data en base de datos, dentro del servidor se encuentran implementados patrones de diseño para la orquestación dentro del ecosistema de microservicios y la respuesta a peticiones desde el cliente. Las peticiones desde cliente son respondidas desde servidor de tal manera que el manejo de cada petición es completamente desconocido para el cliente.

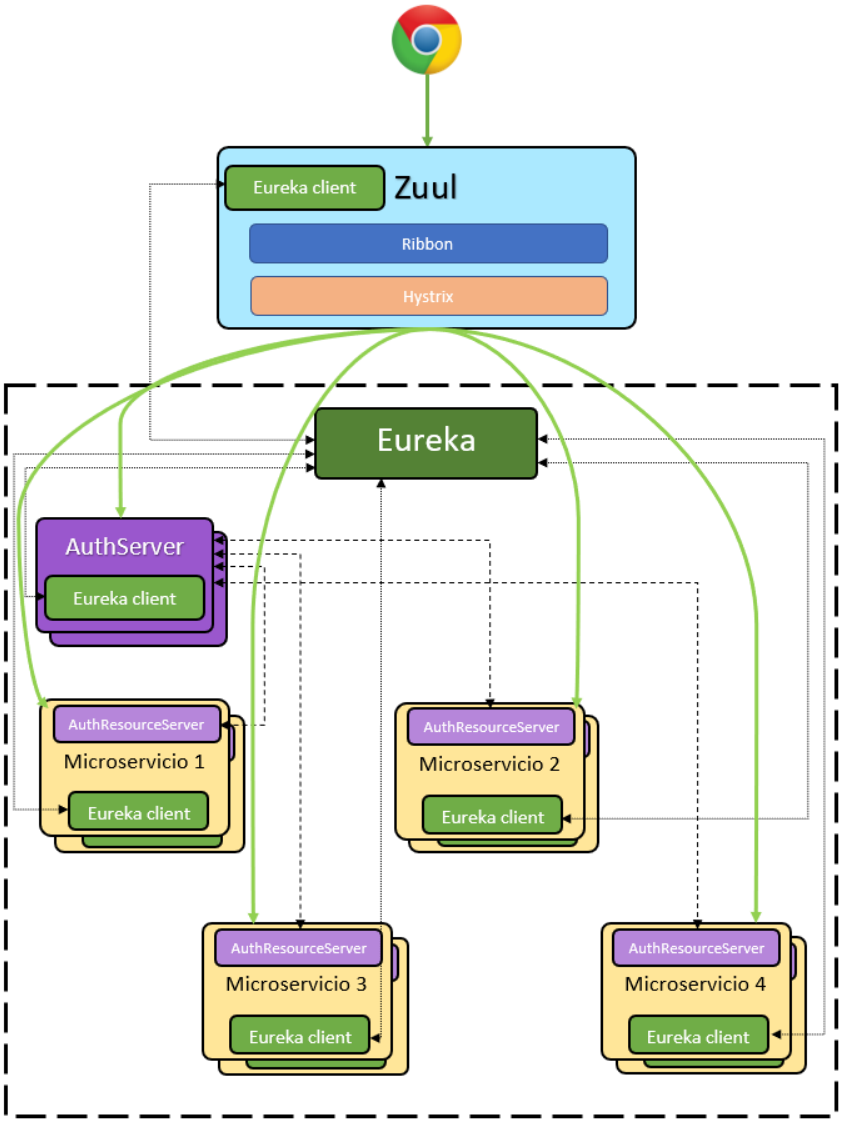


Fig. 60: Arquitectura de SisLab en desarrollo – Servidor

5.3.2. Patrón API Gateway

El patrón API Gateway define un API a partir de las ya existentes para funcionar como enrutador desde un único punto de entrada similar al funcionamiento de un proxy inverso, en el caso del sistema desarrollado las API Rest expuestas por los cuatro microservicios (como se muestra en Fig. 61). El patrón esta descrito en el artículo 'Pattern: API Gateway / Backends for Frontends' [42].

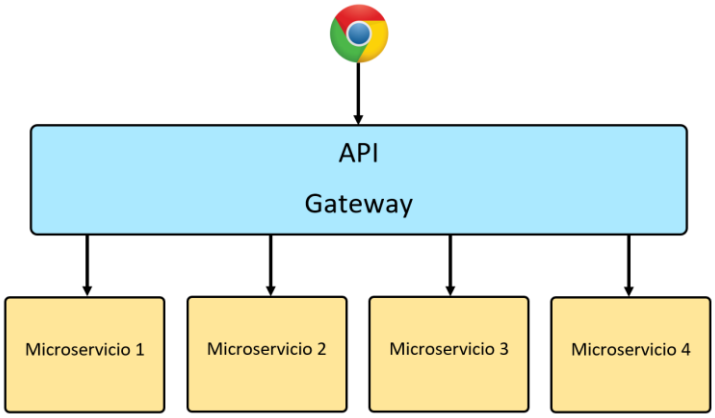


Fig. 61: Arquitectura API Gateway – Servidor

La herramienta Zuul actúa como Edge Service proporcionando el punto de entrada a el ecosistema de microservicios del sistema, provee enrutamiento dinámico, seguridad y monitorización de las peticiones realizadas al servidor. Mediante Zuul el sistema resuelve:

- Las instancias de los microservicios y su ruta de acceso.
- La granularidad de los microservicios y su transparencia para la capa de cliente.
- Seguridad (autenticación y autorización) y protección ante amenazas (inyección de código).
- Monitorización, supervisión y análisis de las API's expuestas por cada microservicio.
- Orquestación de API's internas y sus posibles dependencias con API's externas (Google Maps, Paypal) [43].

Filtros en Zuul

Zuul como Edge Service tiene la funcionalidad de control de filtros, permiten tener un control de acciones de las peticiones HTTP realizadas desde el cliente, existe 4 tipos de filtro (el orden de los filtros se detalla en Fig. 62):

- PRE
- ROUTING
- POST
- ERROR

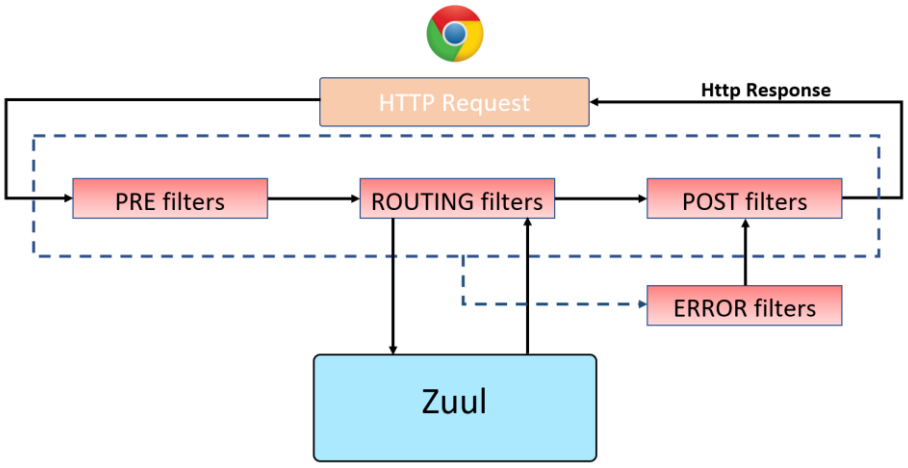


Fig. 62: Filtros en Zuul

El sistema desarrollado tiene la implementación de un tipo de filtro, ERROR filters, para la impresión de mensajes de error en pantalla.

5.3.3. Proceso de petición de cliente a servidor

El patrón API Gateway implementado en el sistema maneja una petición desde el cliente hacia el microservicio responsable de responder de una manera diferente a un sistema con arquitectura monolítica. El sistema transacciona una petición con los pasos detallados en Fig. 63:

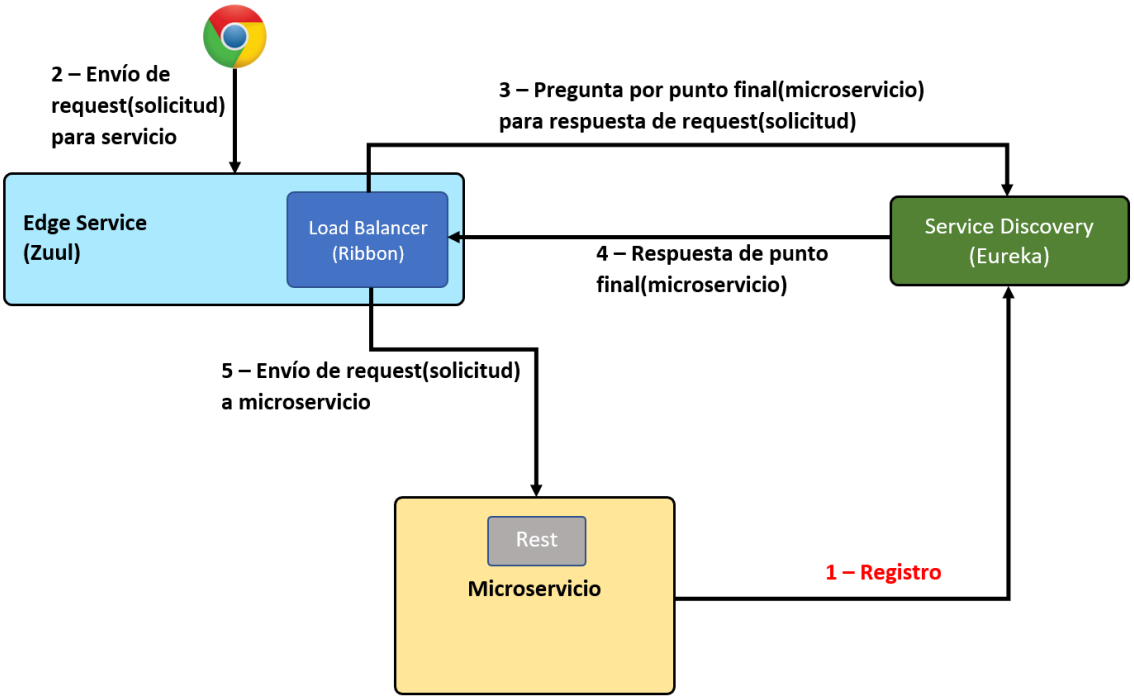


Fig. 63: Proceso de petición de cliente a servidor

El cliente realiza una petición a el servidor mediante el API Rest expuesta por el Gateway, el manejo de la petición y su respuesta es completamente transparente para el cliente, la infraestructura consta de:

- Edge Service (Zuul). Provee un punto de entrada para el cliente a el servidor del sistema, mediante el componente Ribbon las API's de los microservicios son localizadas y una petición del cliente es enrutada a una instancia del microservicio encargado de su respuesta.
- Dynamic Routing y Load Balancer (Ribbon). Tiene la responsabilidad como balanceador de carga, si el componente Ribbon encuentra una o más instancias disponibles distribuye las peticiones del cliente de la manera más óptima mediante el algoritmo RoundRobin, mediante una instancia de Eureka tiene información de ruteo para localizar cada microservicio.
- Service Discovery (Eureka). Permite la localización entre el Gateway y cada microservicio, para el manejo de las peticiones. Si fuera necesario Eureka permite el registro de un servicio en tiempo de ejecución.

5.4. Problemas e impedimentos encontrados durante el desarrollo

El desarrollo de una aplicación lleva consigo un conjunto de problemas tales como: una inexacta definición del alcance, incorrecta definición de tiempos, etc., posibles desacuerdos en la toma de decisiones con respecto a: gestión de costos, gestión y selección de recursos humanos, manejo de riesgos, etc. [44].

En el desarrollo del presente sistema se encontraron problemas en términos de metodología de desarrollo, uso de tecnologías y problemas de diseño de arquitectura con respecto a la tesis base para el desarrollo del sistema SisLab. En el presente documento se redacta con mayor énfasis los problemas de tecnología y diseño de arquitectura.

5.4.1. Problemas con tecnologías de desarrollo

Las tecnologías de desarrollo fueron estudiadas en los Sprints iniciales del proyecto, fueron seleccionadas por motivos de conocimiento previo por parte de los desarrolladores, disponibilidad de documentación existente de las herramientas y opciones de licencias.

- *Disponibilidad de RED HAT OPENSHIFT*

Durante el proceso de desarrollo con la tecnología proporcionada por Red Hat, se encontró el problema de pago de la versión de OpenShift debido a que esta herramienta está orientada para el ambiente empresarial.

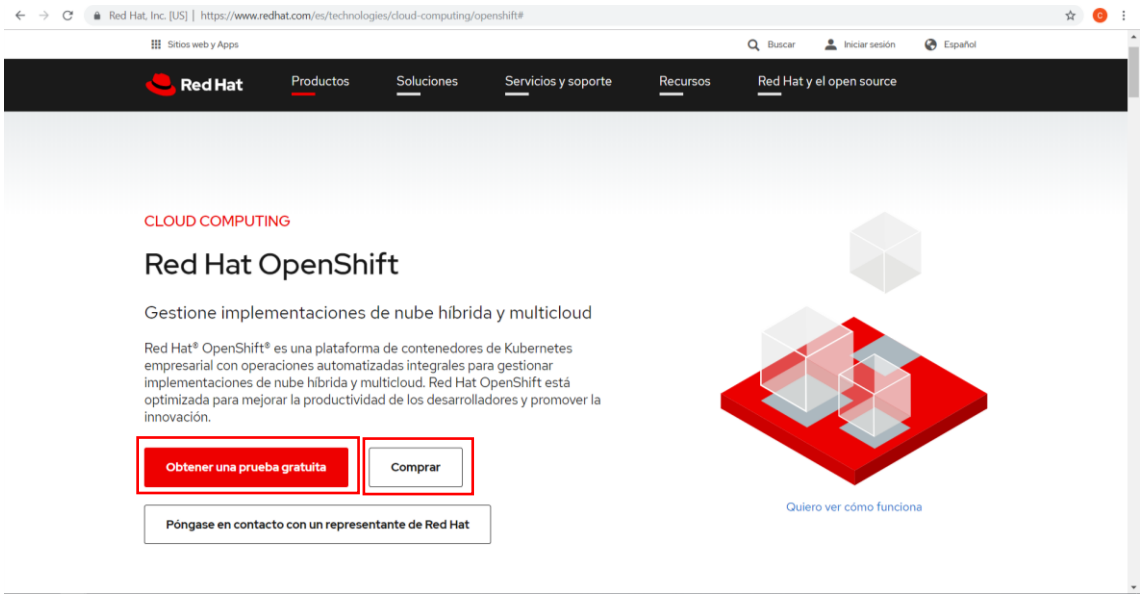


Fig. 64: Red Hat OpenShift [45]

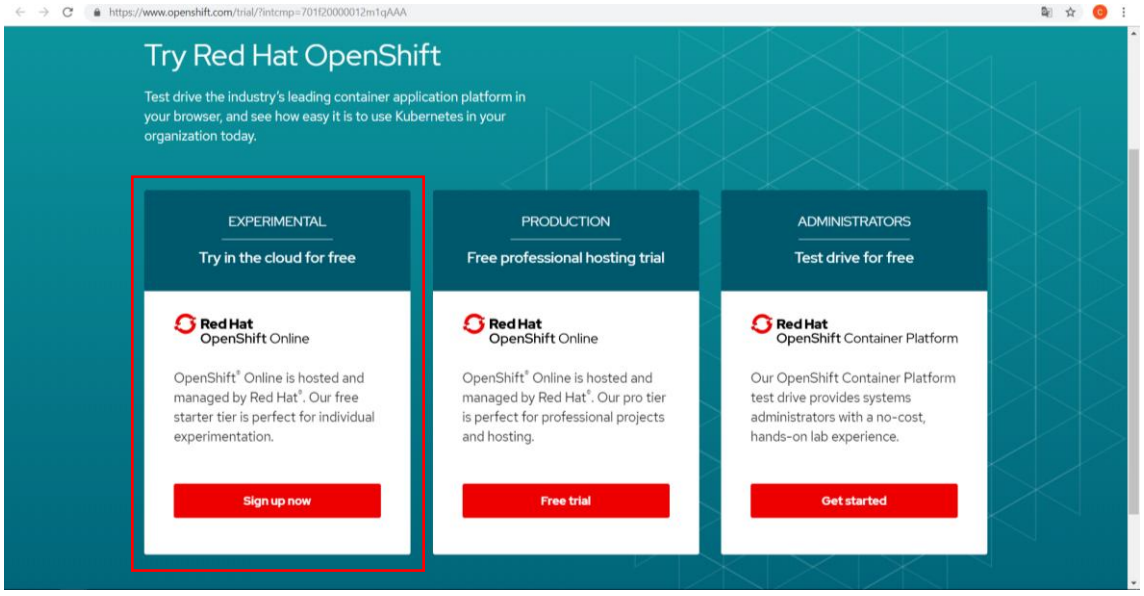


Fig. 65: Paquetes disponibles en Red Hat OpenShift [46]

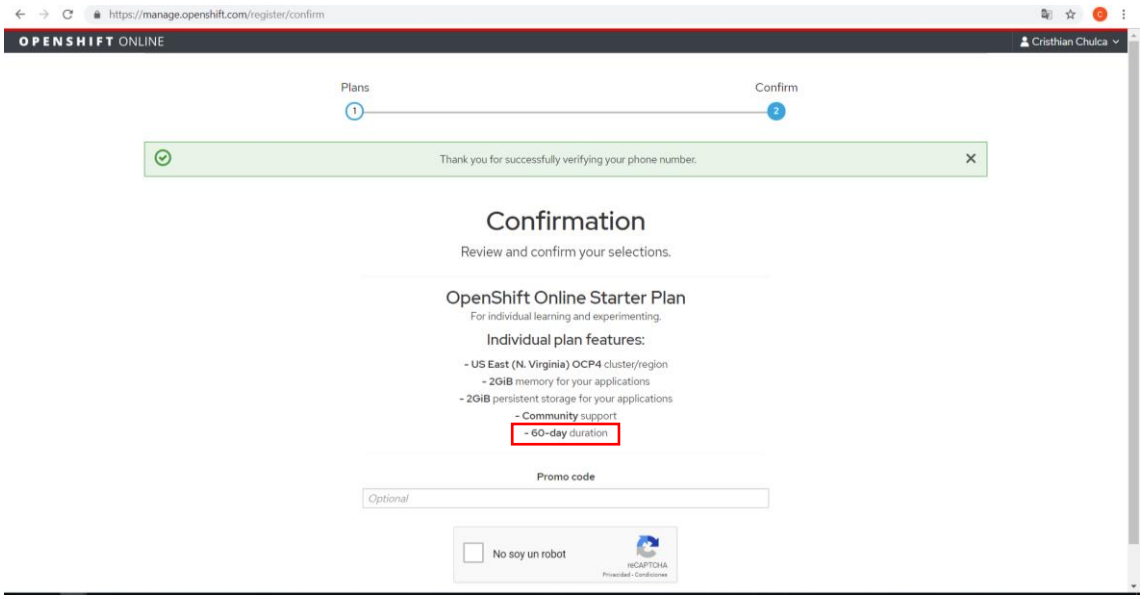


Fig. 66: Registro para versión gratuita de Red Hat OpenShift [47]

En la suscripción de versión gratuita se puede evidenciar que el tiempo de duración es de 60 días, por tratarse de la migración de un sistema en su arquitectura y un refactor en el código fuente, representa un nivel complejo de desarrollo debido a que el sistema SISLAB tiene cambios a partir de su versión inicial de los cuales no se tiene ningún tipo de documentación. El tiempo de desarrollo estimado fue mayor al tiempo que la herramienta OpenShift ofrece.

En su versión pagada la herramienta presenta las siguientes opciones:

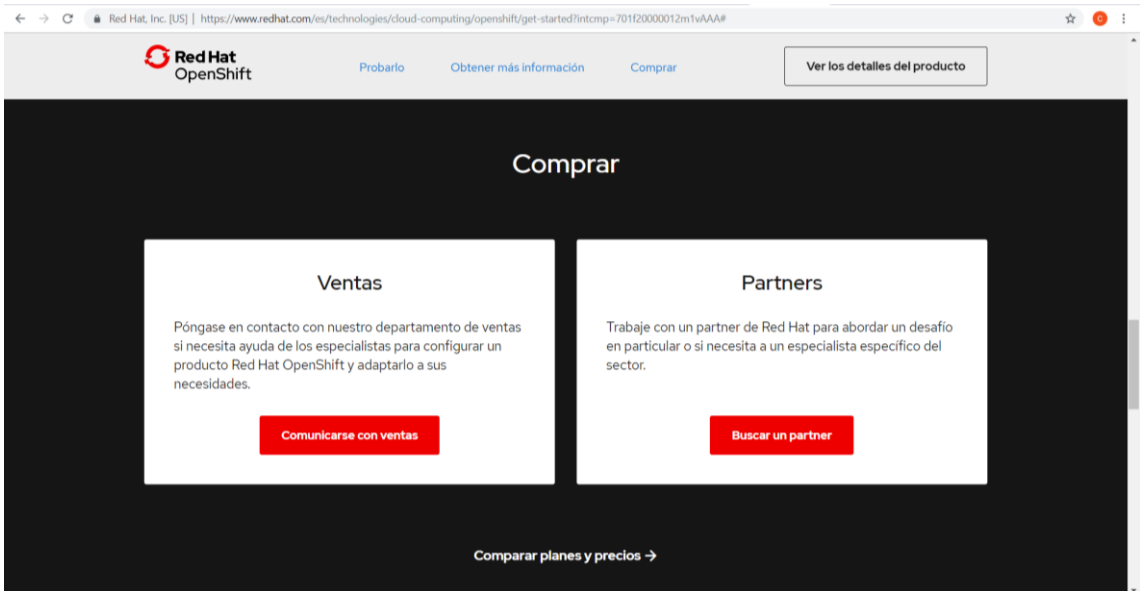


Fig. 67: Paquetes disponibles para versión de pago de Red Hat OpenShift [47]

- Solución

En el Sprint 6 se decidió cambiar de framework para el desarrollo del sistema, las tareas para el cambio se encuentran descritas y justificadas en Sexta reunión de planificación de Sprint (SPRINT 6):, la selección y características del framework están detalladas en Spring Tool Suite 4.

- *Clave Primaria en entidades de Base de datos*

La Base de Datos del sistema SisLab tiene entidades con Clave Primaria (PK) de tipo String, pero en la columna los registros guardados no tienen caracteres alfabéticos sino valores enteros que son generados por secuencias de la base de datos, existe creada una secuencia para cada tabla como se muestra en Fig. 68.

```
CREATE TABLE "Laboratorios".cliente (
    id_cliente varchar(10) NOT NULL DEFAULT nextval('"Laboratorios".secuencia_cliente'::regclass),
    cedula bpchar(10) NULL,
    id_tipocliente varchar(10) NOT NULL,
    ruc_cl bpchar(13) NULL,
    nombre_cl bpchar(200) NOT NULL,
    direccion_cl bpchar(100) NOT NULL,
    telefono_cl bpchar(15) NULL,
    otrofono_cl bpchar(15) NULL,
    celular_cl bpchar(15) NULL,
    fax_cl bpchar(15) NULL,
    email_cl bpchar(50) NULL,
    contacto_cl bpchar(100) NULL,
    nombre_usuario varchar(30) NULL,
    fechatrans timestamp NULL,
    pasaporte_cl varchar(20) NULL,
    CONSTRAINT pk_cliente PRIMARY KEY (id_cliente),
    CONSTRAINT fk_cliente_cliente_t_tipoclie FOREIGN KEY (id_tipocliente) REFERENCES "Laboratorios".tipocliente(id_tipocliente) ON UPDATE
    RESTRICT ON DELETE RESTRICT
);
COMMENT ON TABLE "Laboratorios".cliente IS 'Clientes de la EPN';

-- Permissions

ALTER TABLE "Laboratorios".cliente OWNER TO postgres;
GRANT ALL ON TABLE "Laboratorios".cliente TO postgres;
```

Fig. 68: Ejemplo de secuencia en Base de Datos

En back-end cuando se crearon los modelos en un inicio sus Claves Primarias (PK) fueron creadas de tipo String como lo indican los scripts obtenidos de la Base de Datos. En JPA existe la anotación *@GeneratedValue* que permite la conexión entre un secuencial y un atributo en una entidad, pero el atributo debe ser de tipo entero. En la propuesta redactada en CAPÍTULO 3 – ESTUDIO DE PROPUESTA METODÓLOGICA no se toma en cuenta esta particularidad de la Base de Datos.

- Solución

Se creó una clase utilitaria que permite la generación de un valor secuencial de tipo String con caracteres numéricos como se muestra en Fig. 69. Mediante la anotación *@GeneratedValue* personalizada se asigna el secuencial a su respectiva Clave Primaria en una entidad en la Base de Datos.

```java
public class MyGenerator implements Configurable, IdentifierGenerator {

    private String sequenceName;

    @Override
    public Serializable generate(SessionImplementor arg0, Object arg1) throws HibernateException {

        Connection connection = (Connection) arg0.connection();
        PreparedStatement ps = null;
        String consultaSecuencia = "select nextval ('\"laboratorios\"." + sequenceName + "')";
        try {
            ps = connection.prepareStatement(consultaSecuencia);
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                return rs.getString(1);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public void configure(Type type, Properties params, ServiceRegistry serviceRegistry) throws MappingException {
        sequenceName = params.getProperty("sequenceName");
    }

}
```

Fig. 69: Clase utilitaria para generación de valor secuencial

CAPITULO 6 – PRUEBAS Y RESULTADOS

6.1. Pruebas de Stress del Sistema

Las pruebas de Stress fueron aplicadas con la herramienta “RESTful Stress”, aplicación que permite probar, verificar y estresar un servicio web REST mediante la invocación del cliente con verbos HTTP (GET, POST, PUT, DELETE), mediante el ingreso de la dirección URL del servicio web REST y el número de peticiones para ejecutar. Esta permite tener los siguientes resultados:

- Gráfica de rendimiento (peticiones con su respuesta).
- Tiempos de ejecución de peticiones (máxima y mínima duración, tiempos promedio).
- Cálculo de usuarios concurrentes y usuarios concurrentes paralelos (multi-processors, multi-threading, etc).

Los servicios web REST de los microservicios fueron desarrollados en distintos sprint durante la implementación del sistema, y fueron probados con la herramienta Postman, aplicación que permite hacer llamadas a servicios REST mediante su dirección URL, envío y recepción de estructuras JSON según el verbo HTTP que sea ejecutado. El presente proyecto tiene como objetivo principal la implementación de la arquitectura de microservicios por este motivo se adjunta en el presente documento una prueba de stress de una dirección URL por cada microservicio desarrollado.

6.1.1. Escenario de pruebas

- Computador Intel Core i7 de 2.40 GHz.

- Memoria RAM 8 GB.
- Sistema Operativo Windows 10 pro de 64 bits.
- Aplicación RESTful Stress 1.6.0.

6.1.2. Resultados obtenidos

Las pruebas se llevaron a cabo en distintos escenarios para la verificación de baja dependencia entre microservicios, variando la disponibilidad del servidor Apache Tomcat de cada microservicio.

MICROSERVICIO 1

Funcionalidad:	Consulta de tipos de servicio (CRUD), creación de orden de trabajo, creación de proforma.
URL:	http://localhost:8099/micro1/tipoServicio
Verbo HTTP:	GET
Número de peticiones:	2000
Tiempo de retardo:	100 ms.
Tiempo de espera:	30000 ms

Servidores en funcionamiento:

Fig. 70: Servidores de Aplicación en funcionamiento Microservicio 1

Resultados:

- Historial de peticiones

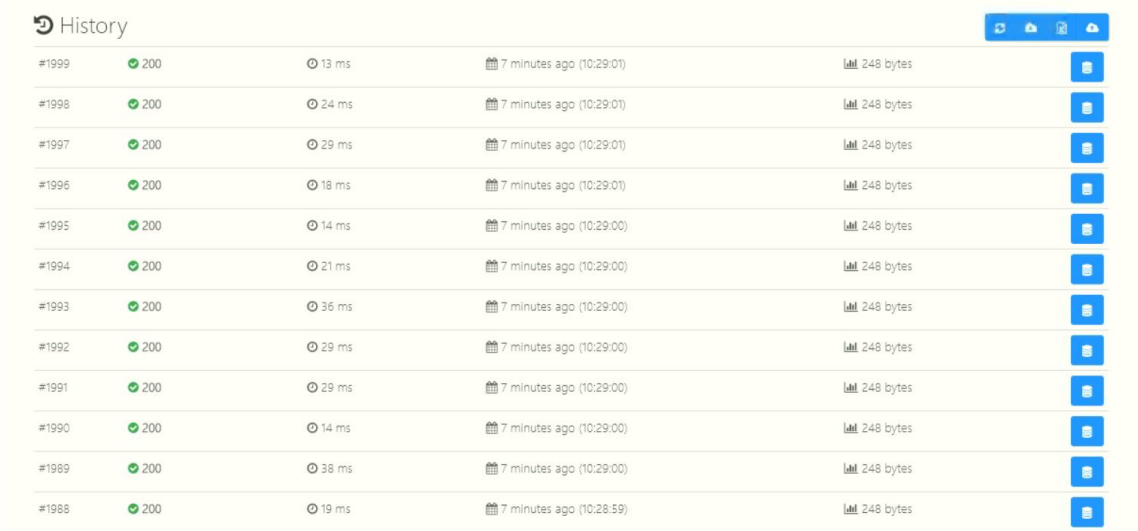


Fig. 71: Historial de peticiones para pruebas Microservicio 1

- Gráfica de rendimiento

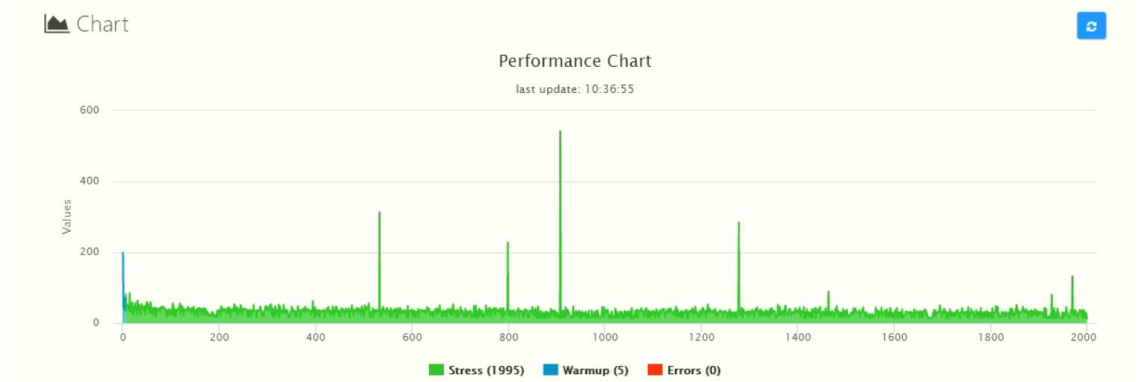


Fig. 72: Gráfica de peticiones Microservicio 1

- Tiempos promedio de respuesta, mínimo y máximo tiempo de respuesta de una petición, tiempo total para completar 2000 peticiones, número de peticiones con error.

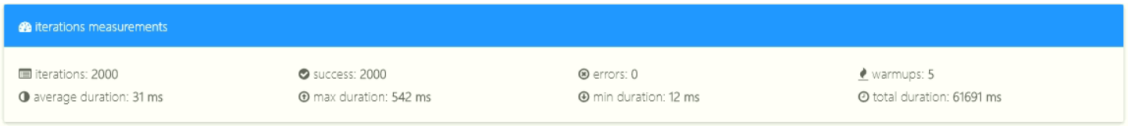


Fig. 73: Tiempos de respuesta Microservicio 1

- Cálculos de usuarios concurrentes.

	client (all)	client (best)	server (all)	server (best)
average duration	31 ms	28 ms	29 ms	26 ms
throughput	32.26 req/sec	35.71 req/sec	34.48 req/sec	38.46 req/sec
inter-arrival time	5031 ms	5028 ms	5029 ms	5026 ms
arrival rate	0.0001987676	0.0001988862	0.0001988467	0.0001989654
concurrent users	162	180	173	193
parallel conc. users (+120% / +500%, observ.)	194 810	216 900	208 865	232 965

Fig. 74: Cálculo de promedio de peticiones Microservicio 1

Los cálculos se realizan con el tiempo promedio de cliente (aplicación que hace la petición), mejor tiempo de cliente, tiempo promedio de servidor (respuesta de petición) y mejor tiempo de servidor.

Descripción de campos:

CAMPO	DESCRIPCION
Average duration	Promedio de duración de entrega de peticiones en milisegundos. Tiempo total / número de peticiones
Throughput	Número de peticiones por segundo que la aplicación REST puede manejar y procesar, entregando una respuesta correcta al cliente. 1000 ms / average duration
Inter-arrival time	Tiempo de diferencia entre el inicio de dos peticiones continuas considerando Think time (tiempo entre el final e inicio de dos peticiones continuas, en un ambiente de producción este tiempo debe ser considerado de 5 segundos). average duration + think time
Arrival rate	Inverso de inter-arrival time 1 / inter-arrival time
Concurrent users	Número de usuarios que la aplicación REST puede manejar y procesar en ambiente de producción con uso normal de recursos, el valor es calculado mediante valores empíricos. Throughput / (arrival rate * 1000)

MICROSERVICIO 2

Funcionalidad: Consulta de órdenes de trabajo (CRUD).

URL: http://localhost:8099/micro2/ordenTrabajo

Verbo HTTP: GET

Número de peticiones: 2000

Tiempo de retardo: 100 ms.

Tiempo de espera: 30000 ms

Servidores en funcionamiento:

Fig. 75: Servidores de Aplicación en funcionamiento Microservicio 2

Resultados:

Fig. 76: Resultados de pruebas de Microservicio 2

MICROSERVICIO 3

Funcionalidad:	Consulta de bodegas (CRUD), asignación de productos a detalles de órdenes de trabajo.
URL:	http://localhost:8099/micro3/bodega
Verbo HTTP:	GET
Número de peticiones:	2000
Tiempo de retardo:	100 ms.
Tiempo de espera:	30000 ms

Servidores en funcionamiento:

Fig. 77: Servidores de Aplicación en funcionamiento Microservicio 3

Resultados:

Fig. 78: Resultados de pruebas de Microservicio 3

6.2. Validación de Microservicios implementados mediante checklist de Susan Fowler

En la metodología propuesta por Arboleda [6] detalla una etapa de validación mediante el checklist propuesto por Susan Fowler, presentado en su libro “Production Ready Microservices”, en el mencionado libro se abarca la etapa donde el sistema se encuentra en producción, debido a esto, se tomará únicamente el checklist ya que este abarca hasta cuando el sistema se encuentra en etapas de pruebas.

El conjunto de preguntas en el siguiente checklist están relacionadas a verificar que los microservicios implementados tengan estabilidad, confiabilidad, escalabilidad y rendimiento, pero en un ambiente de producción. Las observaciones en cada preguntan son descritas considerando el comportamiento de los microservicios en producción y se toma como punto de partida los resultados obtenidos en las pruebas funcionales que se hicieron en el computador personal de un desarrollador del equipo.

6.2.1. Estabilidad y Confiabilidad

	SI	**NO**	**OBSERVACIÓN**
¿El microservicio tiene un repositorio central donde se almacena todo el código?	X		Un repositorio en Git.
¿El ecosistema de microservicios tiene una línea de implementación estandarizada?		X	N/A. El proyecto no contempla la fase de instalación en producción.
¿Qué acceso tiene el entorno de transición a los servicios de producción?		X	N/A. El proyecto no contempla la fase de instalación en producción.
¿Las implementaciones para la producción se hacen todas al mismo tiempo, o se implementan incrementalmente?		X	N/A. El proyecto no contempla la fase de instalación en producción.
¿Dependencias de microservicios?	X		En la arquitectura propuesta por Arboleda [6] las dependencias son minimizadas mas no solucionadas.
¿Cómo este microservicio mitiga los fallos de dependencia?	X		Los fallos de dependencia podrían ser solucionados a través del bus de servicios de aplicaciones que actualmente posee la DGIP, en ambiente de producción.
¿Hay copias de seguridad, alternativas, retrocesos o almacenamiento en caché defensivo para cada dependencia?	X		El IDE de desarrollo Spring Tools Suite (STS) cuenta con la funcionalidad de impresión de logs de transacciones ejecutadas en el back-end.
¿Son confiables los controles de salud al microservicio?	X		Eureka tiene el objetivo de registrar y localizar microservicios existentes, informa su estado e información relevante.

¿Hay interruptores en su lugar para evitar que el tráfico de producción se envíe a hosts y microservicios poco saludables?	X		En el Api Gateway las librerías de Ribbon y Hystrix se encargan del balanceo de carga y switcheo de peticiones hacia las instancias de los microservicios.
¿Existen procedimientos para depreciar los puntos finales API de microservicios?	X		En el Api Gateway las librerías de Ribbon y Hystrix se encargan del balanceo de carga y switcheo de peticiones hacia las instancias de los microservicios.

Tabla 18: Checkbox Susan Fowler, Estabilidad y Confiabilidad

6.1.2. Escalabilidad y Rendimiento

	SI	**NO**	**OBSERVACIÓN**
¿Cuáles son los cuellos de botella de recursos de estos microservicios?		X	La dependencia con el sistema de Facturación centralizado la DGIP, este fue quitado del SisLab actual.
¿Escalarán las dependencias con el crecimiento esperado de este microservicio?			N/A
¿Se puede enrutar automáticamente el tráfico a otros centros de datos en caso de falla?	X		Mediante la configuración en ambiente de producción del Api Gateway.
¿El microservicio está escrito en un lenguaje de programación que permitirá que el servicio sea escalable y funcional?	X		Los microservicios están implementados en lenguaje Java.
¿Hay alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio maneja las solicitudes?	X		En el Api Gateway las librerías de Ribbon y Hystrix se encargan del balanceo de carga y switcheo de peticiones hacia las instancias de los microservicios.
¿Existe alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio procesa las tareas?	X		En el Api Gateway las librerías de Ribbon y Hystrix se encargan del balanceo de carga y switcheo de peticiones hacia las instancias de los microservicios.
¿Este microservicio maneja los datos de una manera escalable y eficiente?	X		Mediante las peticiones Rest los microservicios ejecuta transacciones mediante JPA.
¿Qué tipo de datos necesita cada microservicio para almacenar?	X		Los microservicios reciben peticiones Rest, con datos en formato JSON.
¿Cuál es el esquema necesario para sus datos?	X		Peticiones Rest con datos en formato JSON
¿Cada microservicio necesita un mayor rendimiento de lectura o escritura?		X	Los microservicios se encuentran divididos en base a la lógica de negocio y datos.
¿Son de lectura pesada, de escritura pesada o ambas cosas?		X	Las transacciones son ejecutas y optimizadas mediante el uso de JPA en los microservicios.

¿La base de datos se escala horizontal o verticalmente? ¿Es replicado o particionado?			Los microservicios implementados trabajan con el schema actual de la base de datos del sistema SisLab, debido a esto, escala horizontalmente puesto que existe un acople con el servidor SRVBALBDD que gestiona el aumento de servidores de base de datos que pueden irse incluyendo para optimizar el índice de respuesta.
¿Los microservicios usan una base de datos compartida o dedicada?	X		Los microservicios implementados trabajan con el schema actual de la base de datos del sistema SisLab.
¿El microservicio tiene un único punto de falla?		X	Las dependencias entre microservicios representan puntos de falla en el sistema.
¿Se han identificado todos los escenarios de falla y posibles catástrofes del microservicio?		X	Se han identificado durante su implementación, pero no se pueden detectar en producción porque el proyecto no contempla la puesta en producción del sistema.
¿Cuáles son las fallas comunes en el ecosistema de los microservicios?	X		El control de dependencias entre microservicios.
¿Cuáles son los escenarios de falla de la capa de hardware que pueden afectar este microservicio?	X		La desconexión del servidor sobre el cual esté instalado cada microservicio.
¿Qué fallas en la capa de comunicación y en la capa de aplicación pueden afectar este microservicio?	X		Errores en la petición del servicio Rest desde el front-end, error en la generación del token de seguridad.
¿Cómo impactan las fallas y las interrupciones de los microservicios en el negocio?	X		Se espera disminuir cualquier tipo de interrupción mediante el Api Gateway, con las librerías de Ribbon y Hystrix que se encargan del balanceo de carga y switcheo de peticiones hacia las instancias de los microservicios.
¿Hay niveles de falla bien definidos?		X	N/A. El proyecto no contempla la fase de instalación en producción.

Tabla 19: Checkbox Susan Fowler, Escalabilidad y Rendimiento

CAPITULO 7 – CONCLUSIONES Y RECOMENDACIONES

El desarrollo de aplicativos con Arquitectura de Microservicios implica tener un conocimiento de sus componentes y funciones de cada uno dentro del ecosistema de Microservicios, en el momento de diseñar la Arquitectura tomar en cuenta que durante la implementación pueden existir dependencias entre microservicios que necesitan ser resueltas o mitigadas lo mayor posible. Se determinó que los componentes propuestos por Arboleda [6] son necesarios para la

implementación del caso de estudio, pero en el estudio de tecnologías disponibles se tomó la decisión de utilizar librerías y herramientas disponibles en el framework Spring.

El diseño de arquitectura del sistema SISLAB obtenida por Arboleda [6] después de aplicar su propuesta de metodología para migrar un sistema web con arquitectura Monolítica hacia una Arquitectura de Microservicios fue analizado y se tomaron en cuenta posibles cambios en el inicio de la implementación del presente proyecto para incluir la comunicación entre microservicios. La revisión de la arquitectura fue llevada a cabo posterior a una revisión de la funcionalidad actual del sistema SISLAB, porque como se indica en el presente documento el sistema se encuentra en un continuo desarrollo para incrementar su funcionalidad o corregir errores no contemplados.

Durante la implementación de la nueva arquitectura fue absolutamente necesaria la colaboración del ingeniero Roberto García como encargado del sistema SISLAB, esto debido a que la gran mayoría de cambios del aplicativo no han sido documentados y a pesar que durante el desarrollo se consideró la revisión de código fuente actual. De esta alta dependencia de un especialista en el sistema a migrar, se concluye que en la metodología propuesta debería existir una etapa de estudio y comprensión de los procesos de negocio en conjunto con una persona especialista en el sistema.

En la etapa dos de la propuesta metodológica se detalla un Modelo de Dominio y se describe funciones breves del sistema, pero como recomendación se podría optar por diagramas UML (casos de uso, casos de usuario, etc.) diseñados después de un completo entendimiento obtenido de parte del especialista del sistema. En la fase de implementación de la migración la cual es el objetivo del presente proyecto los diagramas UML representarían requerimientos funcionales para el desarrollo, y se lograría el ahorro de tiempo en reuniones con la persona encargada del sistema.

La funcionalidad de los microservicios desarrollados fue probada en un ambiente de pruebas en equipos de los desarrolladores con conexión a una base de datos copiada del ambiente de desarrollo del sistema SISLAB. Se realizaron procesos de negocio y se verificó que la información sea registrada de igual manera como lo hace el monolito actual en producción. Estas pruebas fueron ejecutadas de manera incremental con la funcionalidad de cada microservicio, porque su desarrollo fue en paralelo. El acceso al sistema actual únicamente fue a un backup de la base de datos en un ambiente de desarrollo y a el código fuente del sistema actual.

El Sistema Integrado de Información (SII) de la Dirección de Gestión de la Información y Procesos (DGIP) en la Escuela Politécnica Nacional cuenta con un sistema de autenticación y facturación, el SISLAB tiene dependencia con estos de diferentes maneras. El sistema de autenticación está encargado del ingreso y asignación de los usuarios a una unidad interna en la distribución jerárquica de empleados de la EPN y el sistema de facturación es manejado en la tesorería de la EPN. Estos sistemas fueron simulados por microservicios implementados para realizar pruebas del sistema.

Las pruebas funcionales no se ejecutaron con casos de prueba debido a que el sistema actual no tiene una documentación de levantamiento de requerimientos, pero se hicieron pruebas funcionales comparando el sistema actual en un ambiente de desarrollo con el implementado en microservicios, estas fueron llevadas a cabo al final de los sprint con las implementaciones de los microservicios en front-end y back-end. Las pruebas no funcionales fueron llevadas a cabo en seguridad, carga de peticiones, usabilidad y escalabilidad, mediante algunas herramientas de

software disponibles, se ejecutaron en ambiente de desarrollo en computadores de los miembros del equipo.

La aplicación de SCRUM para el desarrollo del sistema permitió el cambio de herramientas tecnológicas en un punto del proyecto en el Sprint 5, porque permitió la nueva planificación de tiempos para los siguientes Sprints. Además, tener un control de los tiempos de cada desarrollador en el proyecto y la organización para la solución de problemas encontrados durante la implementación, o decisiones que tomaron tiempo de estudio las cuales no se encontraban definidas en la Arquitectura propuesta (por ejemplo, soluciones de dependencias entre microservicios).

En la revisión del estado del arte de migración a microservicios se encontró documentación con proyectos que también se centraban solo en proponer una arquitectura para la migración, pero con la diferencia que se implementaba al menos un microservicio para obtener estadísticas en tiempos de respuesta y tiempos de desarrollo. Se recomienda en futuras propuestas de Arquitecturas para un sistema implementar una parte del mismo para detectar y dar solución a posibles problemas durante el desarrollo, en el caso de Arquitectura de Microservicios implementar un microservicio y entorno a los componentes del ecosistema el Discovery Server para poder instanciar el microservicio. Para la implementación de arquitecturas relativamente nuevas (Arquitectura de Microservicios en el presente proyecto, Arquitecturas con servicios cloud) una recomendación factible es realizar un estudio previo de las herramientas tecnologías disponibles especializadas para el tipo de arquitectura porque en la actualidad existen frameworks, IDE's, editores de código con pluggins, incluso lenguajes de programación, entre otros.

REFERENCIAS BIBLIOGRAFICAS

[1] M. Fowler, «Microservices,» ThoughtWorks, 25 Marzo 2014. [En línea]. Available: https://martinfowler.com/articles/microservices.html. [Último acceso: 10 Julio 2019].

[2] T. Yarygina y A. H. Bagge, «Overcoming Security Challenges in Microservice Architectures,» de *in 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, doi: 10.1109/SOSE.2018.00011. , 26-29 March 2018 2018, pp. 11-20.

[3] I. Amazon Web Services, «¿Qué son los microservicios? | AWS,» Amazon Web Services, Inc., 2019. [En línea]. Available: https://aws.amazon.com/es/microservices/. [Último acceso: 10 Julio 2019].

[4] S. J. Fowler, Production-Ready Microservices, United States of America: O'Reilly Media, 2016.

[5] E. Wolff, Microservices: Flexible Software Architecture, United States of America: Addison-Wesley Educational Publishers Inc, 2016.

[6] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO*

DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, p. 29.

[7] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, p. 29.

[8] Microsoft, «Arquitecturas de aplicaciones web comunes | Microsoft Docs,» Microsoft, 29 Enero 2019. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures. [Último acceso: 10 Julio 2019].

[9] Microsoft, «Estilo de arquitectura de microservicios - Azure Application Architecture Guide | Microsoft Docs,» Microsoft, 12 Noviembre 2018. [En línea]. Available: https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices. [Último acceso: 10 Julio 2019].

[10] D. B. Cahyono, Suhardi y N. B. Kurniawan, «Developing statistical business register service system based on microservice architecture,» de *in 2018 International Conference on Information and Communications Technology (ICOIACT)*, doi: 10.1109/ICOIACT.2018.8350712. , 6-7 March 2018 2018, pp. 500-505.

[11] A. Akbulut y H. G. Perros, «Software Versioning with Microservices through the API Gateway Design Pattern,» de *in 2019 9th International Conference on Advanced Computer Information Technologies (ACIT)*, doi: 10.1109/ACITT.2019.8779952, 5-7 June 2019 2019, pp. 289-292.

[12] T. Yarygina y A. H. Bagge, «Overcoming Security Challenges in Microservice Architectures,» de *in 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, doi: 10.1109/SOSE.2018.00011., 26-29 March 2018 2018, pp. 11-20.

[13] R. Wongsakthawom y Y. Limpiyakorn, «Development of IT Helpdesk with Microservices,» de *in 2018 8th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, doi: 10.1109/ICEIEC.2018.8473557. , 15-17 June 2018 2018, pp. 31-34.

[14] S. Sarkar, G. Vashi y P. Abdulla, «Towards Transforming an Industrial Automation System from Monolithic to Microservices,» de *in 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, doi: 10.1109/ETFA.2018.8502567. , 4-7 Sept. 2018 2018, pp. vol. 1, pp. 1256-1259.

[15] D. Taibi, V. Lenarduzzi, C. Pahl y A. Janes, «Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages,» de *presented at the Proceedings of the XP2017 Scientific Workshops*, Cologne, Germany, 2017.

[16] S. Eski y F. Buzluca, «An automatic extraction approach: transition to microservices architecture from monolithic application,» de *presented at the Proceedings of the 19th*

International Conference on Agile Software Development: Companion, Porto, Portugal, 2018.

[17] Z. Ren et al, «Migrating Web Applications from Monolithic Structure to Microservices Architecture,» de *presented at the Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, Beijing, China, 2018.

[18] L. Carvalho et al., «Analysis of the criteria adopted in industry to extract microservices,» de *presented at the Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, Montreal, Quebec, Canada, 2019.

[19] L. Carvalho et al., «Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study,» de *presented at the Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, Paris, France, 2019.

[20] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, p. 25.

[21] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 22-23.

[22] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 25-26.

[23] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, p. 32.

[24] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 34-35.

[25] L. F. CHACHA BURGOS y E. J. VISCARRA PRUNA, «DESARROLLO E IMPLANTACIóN DEL SISTEMA DE GESTIóN DE ANáLISIS FíSICO-QUíMICO Y MICROBIOLóGO DE AGUAS,

SUELOS Y LODOS PARA EL CENTRO DE INVESTIGACIóN DE CONTROL AMBIENTAL (CICAM) DE LA EPN,» QUITO, ESCUELA POLITéCNICA NACIONAL, 2010, pp. 40-49.

[26] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, p. 40.

[27] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 32-33.

[28] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 33-34.

[29] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 34-35.

[30] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 35-38.

[31] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 90-91.

[32] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 40-41.

[33] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO*

DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 41-46.

[34] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 47-55.

[35] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 56-58.

[36] A. C. C. AUGUSTO, «PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS,» de *TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN*, Quito, ESCUELA POLITÉCNICA NACIONAL - FACULTAD DE INGENIERÍA DE SISTEMAS, 2018, pp. 58-59.

[37] I. Eclipse Foundation, «Eclipse MicroProfile | projects.eclipse.org,» Eclipse Foundation, Inc., 06 2019. [En línea]. Available: https://projects.eclipse.org/proposals/eclipse-microprofile. [Último acceso: 15 07 2019].

[38] M. Redlich, «Thorntail 2.2.0 Features Automated Migration from WildFly Swarm,» InfoQ, 18 Septiembre 2018. [En línea]. Available: https://www.infoq.com/news/2018/09/red-hat-releases-thorntail-2.2.0. [Último acceso: 15 Julio 2019].

[39] 2. P. Software, «Spring Framework,» 2019 Pivotal Software, Inc., 2019. [En línea]. Available: https://spring.io/projects/spring-framework. [Último acceso: 15 Julio 2019].

[40] 2. R. Hat, «CAPÍTULO 3. DESARROLLO DE APLICACIONES MEDIANTE JBOSS EAP Red Hat JBoss Enterprise Application Platform 7.0 | Red Hat Customer Portal,» 2019 Red Hat, Inc., 2019. [En línea]. Available: https://access.redhat.com/documentation/es-es/red_hat_jboss_enterprise_application_platform/7.0/html/getting_started_guide/developing_apps_using_jboss_eap. [Último acceso: 15 Julio 2019].

[41] 2. R. Hat, «Características de Red Hat Openshift,» 2019 Red Hat, Inc., 2019. [En línea]. Available: https://www.redhat.com/es/technologies/cloud-computing/openshift/features. [Último acceso: 15 Julio 2019].

[42] Pivotal, «Spring Framework Documentation,» 31 01 2019. [En línea]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/. [Último acceso: 15 10 2019].

[43] Angular, «Angular - Architecture overview,» [En línea]. Available: https://angular.io/guide/architecture. [Último acceso: 20 Julio 2019].

[44] Microsoft, «Documentation for Visual Studio Code,» 2019 Microsoft, 2019. [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 20 Julio 2019].

[45] C. Richardson, «API gateway pattern,» Kong, 2019. [En línea]. Available: https://microservices.io/patterns/apigateway.html. [Último acceso: 21 Julio 2019].

[46] S. V. -. I. Analyst, «API Gateway en tu arquitectura de microservicios,» ITDO, 10 Enero 2019. [En línea]. Available: https://www.itdo.com/blog/api-gateway-en-tu-arquitectura-de-microservicios/. [Último acceso: 21 Julio 2019].

[47] I. J. J. M. Román, «PROBLEMAS DESARROLLO SOFTWARE,» Fábrica de Software USMP, [En línea]. Available: https://www.usmp.edu.pe/publicaciones/boletin/fia/info86/articulos/problemasDesarrolloSoftware.html. [Último acceso: 17 Septiembre 2019].

[48] 2. R. Hat, «Plataforma de aplicaciones en contenedores - Red Hat OpenShift,» 2019 Red Hat Inc., 22 Mayo 2019. [En línea]. Available: https://www.redhat.com/es/technologies/cloud-computing/openshift#. [Último acceso: 20 Julio 2019].

[49] 2. R. Hat, «Start a Free Trial - Red Hat OpenShift,» 2019 Red Hat, Inc., 22 Mayo 2019. [En línea]. Available: https://www.openshift.com/trial/?intcmp=701f20000012m1qAAA. [Último acceso: 20 Julio 2019].

[50] 2. R. Hat, «Register Openshift,» 2019 Red Hat, Inc., 22 Mayo 2019. [En línea]. Available: https://manage.openshift.com/register/confirm. [Último acceso: 20 Julio 2019].