Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería de Sistemas

DESARROLLO DE UN FRAMEWORK CON UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA EL UPGRADE DEL SISTEMA DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES DE LA UCSM

Tesis presentada por el bachiller:

Bobadilla Chara Sebastian Gonzalo

Para optar el Título Profesional de:

Ingeniero de Sistemas

Asesora:

Dra. Castro Gutierrez Eveling Gloria

Arequipa - Perú

2022

UCSM-ERP

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

INGENIERIA DE SISTEMAS

TITULACIÓN CON TESIS

DICTAMEN APROBACIÓN DE BORRADOR

Arequipa, 08 de Junio del 2022

Dictamen: 006317-C-EPIS-2022

Visto el borrador del expediente 006317, presentado por:

2017204541 - BOBADILLA CHARA SEBASTIAN GONZALO

Titulado:

DESARROLLO DE UN FRAMEWORK CON UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA EL UPGRADE DEL SISTEMA DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES DE LA UCSM

Nuestro dictamen es:

APROBADO

1221 - PAREDES MARCHENA FERNANDO GERMAN
DICTAMINADOR

1425 - MARTINEZ MUÑOZ JORGE LUIS
DICTAMINADOR

1564 - CORRALES DELGADO CARLO JOSE LUIS
DICTAMINADOR

Agradecimientos

Debo agradecer en primer lugar a mis padres, Caterine y Juan Manuel. Quienes además de darme la vida, me formaron como persona y me enseñaron a caminar por mi mismo.

Agradezco también a los docentes de quienes aprendí a lo largo de mi carrera profesional, que además de impartir conocimientos me demostraron lo que es ser un buen profesional.

Finalmente agradecer a mis amigos, en especial a Diego y Marcelo, quienes me acompañaron desde el primer año de universidad, y que me apoyaron en todo momento.

Dedicatoria

Dedico este trabajo en primer lugar a mis padres, sin quienes no habría podido completarlo. Dedico también este logro a mis amigos, que en ningún momento dudaron de mí y siempre me brindan ánimo para seguir adelante.

Resumen

La tecnología se encuentra en constante evolución, y constantemente podemos ver que surgen nuevas maneras de hacer software que buscan estar mejor preparadas para el entorno de alto crecimiento que estamos viviendo. Sin embargo, muchas organizaciones han decidido mantener sistemas con tecnologías antiguas y presentan una alta resistencia al cambio. Esto en consecuencia genera una mayor cantidad de riesgos, además de generar un mayor costo de mantenimiento. Es por estos motivos que en el presente trabajo se busca proponer un framework para realizar la refactorización y mejora del backend de un sistema, tomando como caso de estudio el ERP de la Universidad Católica de Santa María.

Palabras clave: Framework, refactorización de software, escalabilidad de software, mantenibilidad de software, microservicios

Abstract

Technology is constantly evolving, and so new ways to make software emerge in order to be better prepared for the high-growth environment we are living in. However, many organizations have decided to maintain systems with old technologies and are highly resistant to change. This, in consequence, generates a greater number of risks, in addition to generating a higher maintenance cost. It is for these reasons that in this work I propose a framework to perform the refactoring and improvement of the backend of a system, taking as a case study the ERP of the Catholic University of Santa María.

Keywords: Framework, software refactoring, software escalability, software maintainability, microservices

Introducción

Al igual que la tecnología, el software evoluciona constantemente y da lugar a nuevas maneras de construirlo, ya sea a través de nuevos lenguajes de programación o nuevas metodologías, e incluso arquitecturas. Sin embargo, en muchos casos las organizaciones no actualizan su software y lo mantienen como se construyó inicialmente basados en que no requiere de una mejora ya que cumple con los objetivos del negocio. A esto se puede sumar el hecho de que las organizaciones no quieren realizar una inversión en realizar la modernización de sus sistemas, o que consideran que lo único que es necesario para que sigan operando es aplicarles mantenimiento de manera frecuente. Este problema se manifiesta de diferentes maneras y a diferentes niveles, que puede abarcar desde seguir utilizando procedimientos en Cobol para cálculos financieros a usar motores de plantillas para el desarrollo de un sistema.

Tal es el caso del sistema ERP de la Universidad Católica de Santa María, que actualmente utiliza para su sistema PHP y el motor de plantillas Smarty. PHP, sin embargo, es una tecnología web que ya tiene más de 25 años, y desde su origen han surgido diferentes frameworks, tanto en PHP como en tecnologías más modernas que ofrecen una mayor gama de funcionalidades y una mejor experiencia para los desarrolladores. Actualmente, debido al uso de PHP y Smarty para el Desarrollo del sistema ERP, este ya se ha convertido en inmantenible, debido a la magnitud del software. Esto sumado a las limitaciones de la arquitectura actual han hecho que el código Fuente de la aplicación requiera más tiempo en mantenimiento que en el desarrollo de nuevas funcionalidades o incluso optimización de los procesos ya implementados. Y con el paso del tiempo, el mantenimiento es más complicado, y el desempeño del sistema continúa degradándose.

Por lo anteriormente mencionado, se propone la elaboración de un framework para llevar a cabo la modernización del sistema ERP de la Universidad Católica de Santa María, considerando puntos centrales como la escalabilidad del software, la facilidad de mantenimiento, y el desempeño del sistema. Esta propuesta se realizará con el objetivo de descubrir la mejor manera de diseñar la arquitectura del backend, de manera que pueda ponerse a prueba y así generar una plantilla que pueda ser usada para el Desarrollo del sistema. Es así que el interés del Proyecto es meramente académico.

Para la realización del proyecto primero se llevará a cabo una revisión bibliográfica de las arquitecturas más usadas actualmente y de esa manera elegir la que cuente con más casos de éxito y así tomarla de referencia. Luego se seleccionará uno de los módulos del actual sistema ERP de la UCSM para refactorizarlo y aplicar en la arquitectura y tecnologías elegidas tras la revisión bibliográfica. Para la evaluación de la arquitectura propuesta se realizará una validación en primer lugar a través de métricas de desempeño del sistema. Por las características mencionadas anteriormente, esta es una investigación aplicada tecnológica.

El objetivo del trabajo es desarrollar un framework con una arquitectura basada en microservicios para la mejora del sistema ERP de la UCSM, para lo cual se propone 3 objetivos específicos: (i) Determinar la mejor manera de construir la API en base a las tendencias actuales del mercado, así como también definir con qué tecnología realizarlo. (ii) Refactorizar un módulo del ERP utilizando la arquitectura y tecnología propuesta. (iii) Realizar la validación del modulo para evaluar su usabilidad y desempeño.

En el presente documento y en los capítulos correspondientes se expone el proyecto a desarrollar. En el primer capítulo se presenta el plan de la investigación, con el objetivo general, específicos, las preguntas de investigación, y la hipótesis y variables. En el capítulo 2 se presenta los fundamentos teóricos, en el capítulo 3 el marco metodológico, en el capítulo 4 el desarrollo de la propuesta, en el capítulo 5 el análisis y validación de resultados, en el capítulo 6 la discusión, y finalmente las conclusiones.

Índice

Índice de figuras

Índice de tablas

1 Plan de la investigación

1.1. Planteamiento del problema

A lo largo de los años las tecnologías y frameworks para el desarrollo de aplicaciones web han evolucionado. Sin embargo, muchas instituciones han continuado utilizando tecnologías antiguas a pesar de las evidente tendencia hacia no sólo nuevas herramientas, sino también a nuevas maneras de realizar la arquitectura de sus servicios en el backend.

El lugar elegido para realizar el estudio es la oficina del ERP de la Universidad Católica de Santa María. Donde se presentan problemas relacionados al software que si bien no presenta fallas aún, el repositorio del código fuente del sistema ya es inmantenible debido a la manera en la que está estructurado, además de estar desactualizado.

Las causas que originan este problema son variadas, y como indican (Hussain y col., 2017) la principal causa es que las organizaciones consideran que dichos sistemas funcionan adecuadamente y cumplen con los objetivos del negocio, además de que consideran que la inversión en actualizarlos y mejorarlos no es necesaria, y que no ofrecería un retorno significativo.

Las consecuencias de esto es que los sistemas desactualizados continúan creciendo con los problemas y fallas propios de su implementación, y por lo tanto hacen que el trabajo con ellos sea más complicado. Su código es más complicado de entender, su mantenimiento se vuelve más costoso, y su desempeño se va degradando. Esto por ejemplo origina que en lugar de optimizar las operaciones actuales, o implementar nuevas funcionalidades, los desarrolladores pasen la mayor parte de su tiempo resolviendo problemas y dando mantenimiento al sistema.

Por ello, la presente investigación pretende elaborar un framework para realizar la refactorización y

upgrade del backend del sistema ERP de la Universidad Católica de Santa María, de manera que se usen tecnologías modernas y una arquitectura que considere la escalabilidad del sistema.

1.2. Objetivos de la investigación

1.2.1. Objetivo general

Desarrollar un framework utilizando tecnologías y arquitecturas modernas para realizar el upgrade del sistema ERP de la Universidad Católica de Santa María en el 2021.

1.2.2. Objetivos específicos

- Analizar las tendencias actuales en el mercado y el estado del arte para la construcción de la arquitectura de la API del sistema.
- Definir la arquitectura a utilizar para el desarrollo del sistema.
- Refactorizar un módulo con el framework propuesto.
- Validar el desempeño del módulo desarrollado, así como también la usabilidad del framework propuesto.

1.3. Preguntas de investigación, hipótesis, y variables

1.3.1. Preguntas de investigación

- ¿Cuál es la mejor manera de realizar el upgrade del sistema ERP de la Universidad Católica de Santa María mediante un framework que utilice tecnologías y arquitecturas modernas?
- ¿Qué impacto trae utilizar tecnologías y arquitecturas de backend modernas?
- ¿Cuáles son las características que debe cumplir un framework orientado a la escalabilidad de un software?

1.3.2. Hipótesis y variables

a) Hipótesis

En este caso se planteó la siguiente hipótesis: "Utilizar un framework con una arquitectura basada en microservicios y con tecnologías modernas mejora la productividad de los desarrolladores".

b) Variable dependiente

En base a la hipótesis podemos definir como variable dependiente "la productividad de los desarrolladores", para la cual sus indicadores se muestran en la Tabla 1.1.

c) Variable independiente

En base a la hipótesis podemos definir como variable independiente "el framework utilizado para el desarrollo del sistema", para la cual sus indicadores se muestran en la Tabla 1.2.

Tabla 1.1

Indicadores e índices para la variable "Productividad de los desarrolladores"

Indicadores	Índices
Esfuerzo requerido para realizar el mantenimiento a un módulo	Tiempo promedio de 13 horas
Velocidad de implementación de una nueva funcionalidad	Tiempo promedio de 8 horas

Tabla 1.2

Indicadores e índices para la variable "Framework utilizado para el desarrollo del sistema"

Indicadores	Índices
Complejidad del árbol de archivos del código fuente	Tiempo promedio en encontrar un archivo < 10s
Complejidad del archivo de código fuente	Número de líneas del archivo de código Número de funciones contenidas en un archivo de código

1.4. Línea y Sublínea de Investigación

La línea a la que pertenece el presente proyecto es “Ingeniería del Software”, y la sublínea es “Arquitectura del Software”.

1.5. Palabras clave

Framework, refactorización de software, escalabilidad, mantenibilidad, microservicios.

1.6. Solución Propuesta

1.6.1. Justificación e Importancia

Actualmente, para el ERP de la Universidad Católica de Santa María se utiliza PHP para el desarrollo del sistema, una tecnología de hace más de 25 años. Esto ha hecho que el sistema mantenga ciertas deficiencias en lo que respecta a la arquitectura del software, y hoy en día, dado el crecimiento del sistema ERP, es inmantenible.

Debido a las deficiencias técnicas, este presenta una alta entropía. En consecuencia, el tiempo de mantenimiento es más elevado, y existe una mayor posibilidad de que se presenten errores en su funcionamiento. Así como sucede esto en la UCSM, se da el caso en diversas organizaciones a lo largo de no sólo el Perú, sino también el mundo. Estas organizaciones no realizan la mejora de sus sistemas por diversos motivos, sin embargo, el mantener sistemas con tecnologías desactualizadas sólo genera que los costos de mantenimiento se eleven constantemente, y que el desempeño del sistema se vea degradado conforme las necesidades de los usuarios aumentan.

1.6.2. Descripción de la Solución

La solución que se propone es la elaboración de un framework que permita realizar la refactorización del backend del sistema ERP, de manera que se use tecnologías y arquitecturas de API modernas. Este framework además, debe considerar otros aspectos para el desarrollo del sistema en general, de

manera que se tenga también un enfoque en la escalabilidad del sistema.

Para esta solución se realizará una revisión y evaluación de las tecnologías y arquitecturas que actualmente tienen una mayor presencia en el mercado, de manera que se las pueda comparar y así obtener un framework que recopile los mejores aspectos encontrados.

1.6.3. Aporte

En una primera instancia el aporte que se obtiene es el framework que le permitiría a la Unidad de Implementación del ERP de la Universidad Católica de Santa María llevar a cabo el upgrade del sistema, de manera que este pase a estar construido con tecnologías y arquitecturas modernas y mejor preparadas para las necesidades actuales de los usuarios; además, fruto del trabajo se tendrá también identificados los beneficios de realizar este tipo de refactorización en un sistema de gestión empresarial.

Sin embargo, se pretende llegar a elaborar un framework que pueda ser aplicado en cualquier institución, sin importar su rubro o tamaño. Esto a su vez ayudará a que los gerentes presenten una menor resistencia al cambio, al poder fundamentar adecuadamente la necesidad, beneficios, y pasos a seguir de la actualización de sus sistemas.

1.6.4. Matriz de Consistencia

A manera de resumir lo desarrollado en el presente capítulo se elaboró una matriz de consistencia, de manera que se pueda tener todos los elementos clave para la investigación en una sola tabla que pueda ser consultada con mayor facilidad. La matriz de consistencia se puede ver en la Figura 1.1, y contiene el problema de investigación, los objetivos, la hipótesis, las variables e indicadores, la metodología, y la técnicas e instrumentos para la recolección de datos.

Figura 1.1. Matriz de consistencia

Problema de investigación	Objetivos	Hipótesis	Variables e Indicadores	Metodología	Técnicas e Instrumentos
Problema general: ¿Cómo influye el framework utilizado en la productividad de los desarrolladores y la escalabilidad del producto? **Problemas específicos:** ¿En qué medida influye el framework utilizado en la mantenibilidad del sistema desarrollado? ¿En qué medida influye el framework utilizado en la escalabilidad del sistema desarrollado?	**Objetivo general:** Desarrollar un framework utilizando tecnologías y arquitecturas modernas para realizar el upgrade del backend del sistema ERP de la Universidad Católica de Santa María en el 2021. **Objetivos específicos:** • Analizar las tendencias actuales en el mercado para la construcción de la arquitectura de la API del sistema. • Definir la arquitectura a utilizar para el desarrollo del sistema. • Refactorizar un módulo con el framework propuesto. • Validar el desempeño del módulo desarrollado, así como también la usabilidad del framework propuesto.	Utilizar un framework con una arquitectura basada en microservicios y con tecnologías modernas, mejora la productividad de los desarrolladores.	**Variable independiente:** Framework con una arquitectura basada en microservicios. **Indicadores:** • Complejidad del árbol de archivos del código fuente. • Complejidad del archivo de código fuente. **Variable dependiente:** Productividad de los desarrolladores. **Indicadores:** • Esfuerzo requerido para realizar el mantenimiento a un módulo. • Tiempo de implementación de una nueva funcionalidad.	**Tipo de investigación:** El tipo de investigación es aplicada tecnológica. El nivel es exploratorio, y transversal. **Diseño de la investigación:** El diseño de la investigación es experimental, se modificará y controlará la variable independiente y se medirá sus efectos sobre la variable dependiente. Además, tiene un enfoque mixto ya que tiene aspectos cualitativos y cuantitativos.	**Población:** Equipo de desarrollo de la unidad de implementación del ERP de la UCSM N=12 **Muestra:** Profesionales de dedicados al desarrollo y mantenimiento de módulos críticos del ERP n=4 **Técnicas de recolección de datos:** • Encuestas. • Análisis estático y dinámico de un sistema. • Juicio de expertos (técnica Delphi) **Instrumentos de recolección de datos:** • Cuestionario. • Logs del servidor y scripts complementarios.

2 Fundamentos teóricos

2.1. Estado del Arte

La refactorización de código siempre ha sido una tarea importante. Es así que (Jindal & Khurana, 2013), con el objetivo de hacer que esta tarea sea más fácil para los desarrolladores, propusieron una herramienta que ayuda a identificar aquellos puntos que pueden ser refactorizados, al igual que la magnitud de los cambios que podrían realizarse. En esta propuesta se menciona que una manera de mejorar el tiempo de procesamiento del algoritmo es a través de la selección de atributos significativos para el análisis.

Se tiene por otro lado la propuesta de (Pérez-Castillo y col., 2013), quienes propusieron un proceso de reingeniería para recuperar y adaptar los web services asociados con las bases de datos antiguas o actualmente en uso. De esta manera, dichas bases de datos podían continuar en funcionamiento acelerando el proceso de refactorización y obteniendo mejores resultados.

Ambas propuestas mencionadas anteriormente buscan hacer que la refactorización sea un proceso más efectivo y menos disruptivo. Tanto (Pérez-Castillo y col., 2013) como (Jindal & Khurana, 2013) mencionan que la manera de hacer que un proceso de refactorización sea más efectivo es a través de identificar los elementos críticos para el sistema. Es decir, aquellos elementos como servicios importantes o funcionalidades invocadas constantemente, de manera que se tenga un trabajo más efectivo sobre dichos puntos y realizar la refactorización adecuadamente.

Llevar a cabo la refactorización de un software con motivo de modernizarlo es una tarea que se ejecuta con el objetivo de mantener el sistema operando adecuadamente, y es una tarea que puede tener diferentes causas para realizarse. Sobre este tema se encontró dos propuestas que coinciden en que

realizar una refactorización es una tarea complicada. Primero, como lo indica en su estudio (Gerasimou y col., 2018), puede ser que con el objetivo de mejorar la mantenibilidad de un sistema se usen librerías de terceros, y que esas librerías hayan quedado obsoletas o hayan cambiado de tal manera que es necesario actualizar el sistema desarrollado con ellas. Sin embargo, también identificaron que muchas veces un motivo para no llevar a cabo la mejora es evitar los riesgos asociados a esta, como pueden ser errores de los programadores, incompatibilidades con otros elementos del sistema, entre otros. Esto lleva a que las organizaciones por mantener el software en su estado actual queden expuestas a otros riesgos de mayor gravedad, como errores que quedaron sin resolver, o incluso vulnerabilidad frente ataques cibernéticos. Es por ello que proponen una metodología para la modernización del software, en la cual principalmente se basan en métricas (como la cantidad de invocaciones que se hace a una API) para determinar el nivel de dependencia que se tiene frente a una librería. Luego, se procesa el código fuente para obtener patrones y así generar posibles plantillas para que estas luego sean completadas por los desarrolladores y así tener una transición con menos complicaciones a la nueva librería.

Luego, de manera similar al trabajo anterior se elaboró un método para evaluar un sistema monolítico de manera que se obtenga las características estáticas y dinámicas del sistema. Este trabajo fue realizado por (Ren y col., 2018), quienes proponen que en base a dicho análisis se obtenga una primera lista de funcionalidades candidatas a refactorización, debido a que presentan una leve dependencia de otras funciones presentes en el monolito o de librerías de terceros. Sin embargo, a través de este método se puede generar un trabajo iterativo que permita refactorizar la totalidad del monolito de manera gradual. Los autores comentan que se alcanzó un nivel de desempeño óptimo para el método propuesto, y que este puede ser automatizado fácilmente de manera que se agilice su ejecución.

A los dos trabajos mencionados anteriormente se puede agregar la perspectiva que ofrecen (Al-Debagy & Martinek, 2020). En su propuesta desarrollaron una metodología no sólo para identificar funciones candidatas a ser migradas a microservicios, sino que también definieron una propuesta para determinar si un microservicio necesita mayor granularidad. En su trabajo resaltan la utilización de

dos métricas, la cohesión y la complejidad. El objetivo detrás de esto es conseguir microservicios consistentes y así tener una aplicación estable que minimice la ocurrencia de errores. Además, al evaluar los microservicios de esta manera podemos tener una mejor guía a la hora de agruparlos de acuerdo a qué tan relacionadas están las acciones que desarrollan entre sí, como por ejemplo si trabajan con la misma entidad, o si es que todos tienen un objetivo similar en diferentes etapas de un proceso.

De manera similar a los anteriores trabajos, se tiene la propuesta de (Hussain y col., 2017), quienes describen también otros motivos por lo que algunas empresas presentan una cierta resistencia al cambio o modernización de su sistema. El principal motivo que encontraron para esta resistencia es que el sistema con el que cuentan actualmente, por más desactualizado que pueda estar, aún cumple con los objetivos del negocio y no presenta una necesidad de ser actualizado. Sin embargo, eso hace que mantener dicho sistema sea cada vez más costoso, y la preservación del mismo se convierte en una tarea constantemente más complicada debido a los problemas inherentes a la tecnología o herramienta utilizada en ese entonces. Frente a este problema los autores proponen 4 posibles caminos a seguir, que son: descartar el sistema completamente y evaluar nuevamente los requerimientos del negocio, seguir realizando el mantenimiento, reescribir algunas partes del sistema que sean necesarias, reescribir el sistema gradualmente hasta reemplazarlo por completo. Los autores llegaron a la conclusión de que si no se lo puede permitir la empresa, por lo menos deben realizar la reingeniería de los módulos más críticos, para que se pueda dar pase luego a una renovación total.

En vista de la creciente necesidad de realizar una modernización del software (Somogyi & Kovesdan, 2021) han propuesto una herramienta para asistir en esta tarea. Ellos mencionan que es innegable que realizar la modernización del software es una tarea altamente susceptible a errores, y que además consume bastante tiempo. Sin embargo, para hacer que esta tarea sea más sencilla realizaron una herramienta que a través de técnicas de machine learning asista a los desarrolladores al momento de realizar esta actividad. Esto se basa en la identificación de punteros en código en C para que sean convertidos a variables de Java. De manera similar al trabajo propuesto por (Gerasimou y col., 2018) podemos ver que uno de los puntos centrales de llevar a cabo un proceso de modernización es el

identificar objetos críticos para el software, como son las invocaciones a una API externa, o las variables esenciales que se usan para los diferentes procesos. Al tener en cuenta esos elementos podemos asegurarnos de que el proceso de modernización cumplirá con sus objetivos.

Se puede considerar como ejemplo el trabajo realizado por (Wolfart y col., 2021), quienes realizaron una propuesta de una hoja de ruta que seguir al momento de realizar la modernización de sistemas utilizando microservicios. Al inicio de su trabajo, indican que realizar este tipo de trabajos de modernización es una de las mejores estrategias para promover una mejor evolución de los sistemas implementados al aprovechar nuevas tecnologías y metodologías. En su trabajo proponen organizar el proyecto en 4 fases, y en un total de 8 actividades. Llegaron a la conclusión de que esta organización del trabajo presenta ligeras variaciones en trabajos relacionados, sin embargo, indican que en futuros trabajos esperan llegar a tener una validación completa de esta metodología de manera que pueda ser considerada un estándar o ser tomada como referencia por los desarrolladores y así tener un trabajo mejor organizado.

A lo anterior mencionado se puede sumar el trabajo realizado por (Yin & Du, 2021), quienes mencionan que la resiliencia es una de las características clave de las arquitecturas basadas en microservicios. Es por ello, que en su propuesta ofrecen una definición de la resiliencia, y un modelo con el cual evaluarla. La definición que nos ofrecen los autores es "la capacidad de la arquitectura basada en microservicios de superar la ocurrencia de fallos y recuperar los servicios afectados para que regresen a su estado normal". Así mismo, el modelo que proponen está basado en 6 elementos: el objetivo del sistema, la relación entre los sub objetivos y el objetivo general, los obstáculos a los que se enfrenta el sistema, el agente activo del sistema (quien realiza el procesamiento), el requerimiento del sistema, y el dominio en el que se encuentra el sistema, es decir, su entorno. Los autores llegaron a la conclusión de que el modelo propuesto es efectivo al proveer un valor numérico de la resiliencia de un sistema, de manera que pueda ser contrastado con los valores esperados para este.

También se puede tomar en cuenta el trabajo de (Di Francesco y col., 2018), quienes realizaron un estudio sobre empresas que estaban realizando el proceso de migración de sus servicios a una arqui-

tectura basada en microservicios. Esta encuesta se realizó con el objetivo de conocer mejor las actividades que llevan a cabo los desarrolladores y los retos que encontraron en el camino. Dentro de las diferentes respuestas que obtuvieron, llama la atención la pregunta referente a si se implementaron nuevas funcionalidades durante el proceso de migración, en la que sólo uno de los desarrolladores encuestados respondió que no. En cuanto a los resultados que obtuvieron, se tiene una confirmación en lo visto en otros trabajos, como el de (Gerasimou y col., 2018), y es que el proceso de migración a una arquitectura basada en microservicios debe ser realizada de manera progresiva, y debe ser organizada como un proyecto de carácter iterativo e incremental. Además, encontraron que la razón por la que la mayoría de encuestados implementó nuevas funcionalidades es debido a que el sistema anterior no lo permitía.

Un trabajo de similar metodología al anterior mencionado es el realizado por (Carvalho y col., 2019). En su propuesta los autores realizaron una encuesta a 15 especialistas en llevar a cabo un proceso de migración de una arquitectura monolítica a una basada en microservicios sobre los criterios que consideran más útiles a la hora de extraer microservicios y cómo agruparlos. Como resultado de la encuesta, encontraron que la cohesión de los microservicios es el criterio más útil, ya que mediante dicho criterio se puede determinar los límites de un microservicio de una manera más rápida y sencilla. A esto se suman los otros dos criterios identificados como más útiles, que son la reusabilidad, y el posible acoplamiento que pueden tener en futuras implementaciones con nuevos microservicios.

Por otro lado se tiene el trabajo realizado por (Knoche & Hasselbring, 2018), quienes proponen el uso de una arquitectura de microservicios para la modernización de software antiguo. Ellos indican que este estilo arquitectural ha ganado popularidad por su facilidad de escalamiento en entornos de cloud computing. Sin embargo, en el mercado se está adoptando esta arquitectura no por ese aspecto, sino porque es altamente mantenible. Esto se debe a que mantienen bases de código más pequeñas por su aislamiento en componentes. Así mismo, dentro de su trabajo proponen una metodología para realizar una transición de los sistemas antiguos a uno moderno basado en microservicios sin que el proceso sea muy disruptivo para las operaciones de la organización.

Se puede ver que en la literatura el uso de microservicios esta siendo adoptado como la mejor arquitectura de software debido a los diversos beneficios que aporta, como lo mencionan (Balalaie y col., 2018). Además, en ambos trabajos los autores proponen patrones para llevar a cabo una migración de un sistema antiguo o monolítico a uno basado en microservicios. Estos patrones forman parte de una metodología que es considerada como un plan de migración, debido a que se pueden elegir varios para acomodarlos a las necesidades de la organización. A pesar de que son trabajos diferentes, ambos llegaron a obtener modelos similares, que tras la validación correspondiente llegaron a obtener un desempeño adecuado, enfocados en mejorar la escalabilidad, y mantenibilidad del sistema que se desarrolle o se migre a una arquitectura basada en microservicios.

Sin embargo, algo que es necesario mencionar es lo que indican (Costa y col., 2020), quienes en su trabajo dicen que si bien es cierto que los microservicios están siendo ampliamente aceptados en la industria a nivel mundial, aún no se tiene una guía general para el trabajo con esta arquitectura. Es así que su trabajo tiene como objetivo investigar el estado actual de esta área en aspectos centrales como los estándares más adoptados, y qué técnicas y herramientas se usan actualmente en la literatura. Los resultados que obtuvieron es que la mayoría de las investigaciones se centran en resolver retos relacionados con la comunicación entre microservicios en sistemas distribuidos a nivel mundial, y que en base a esto muchos sistemas más pequeños se han visto altamente beneficiados. Además, en los entornos cloud, encontraron que las prácticas de CI/CD utilizan en su mayoría contenedores aislados como Docker para realizar la gestión de los microservicios desarrollados.

A esto se puede sumar el trabajo realizado por (Carrasco y col., 2018), quienes identificaron características que son propensas a generar fallas a la hora de la implementación de microservicios en un sistema, ya sea de cero o a la hora de llevar a cabo un proceso de refactorización. De todos los puntos que identificaron es importante mencionar primero que muchas veces se incurre en fallas al tener una sola persona haciéndose cargo de todo el flujo de desarrollo, y que los equipos que trabajan en un microservicio deben constar de personas con diferentes habilidades de manera que se alcance un desarrollo integral. Un segundo punto importante que ya fue mencionado por otros autores como

(Gerasimou y col., 2018; Knoche & Hasselbring, 2018), es que no se debe realizar una refactorización súbita. Es decir, que todo proceso de migración a una arquitectura basada en microservicios debe ser realizada de manera progresiva, manteniendo de guía la lógica del negocio y los modelos existentes.

Considerando la implementación de microservicios, se puede considerar también la propuesta de (De Iasio & Zimeo, 2021), quienes además de resaltar los beneficios de usar una arquitectura basada en microservicios, indican que en sistemas de mayor tamaño y complejidad, esos servicios requieren de comunicación y sincronización entre sí, porque puede ser que se consuman dos o más de ellos al momento de realizar un proceso de la organización. Es así que ellos proponen un framework para mejorar el trabajo y sincronización de los microservicios de un sistema, de manera que puedan optimizarse al esperar a que culminen antes de poder ser invocados por el siguiente servicio en la cadena de ejecución. Tras realizar la validación del trabajo se vió que tener en cuenta esta sincronización efectivamente mejora el desempeño del sistema, de manera que se debe tener en cuenta su aplicación para cuando se esté trabajando con sistemas grandes que consuman una gran cantidad de microservicios.

Es importante también considerar que al trabajar con microservicios vamos a tener que gestionar una gran cantidad de componentes, y cuya complejidad se encuentra en función de la naturaleza del sistema en la cual se aplican, como lo indican (Engel y col., 2018). En su trabajo los autores proponen un sistema de evaluación para modelos arquitectónicos basados en microservicios, todo basado en un estudio realizado que muestra los principales retos en dichas arquitecturas actualmente. Algunos de estos puntos de evaluación son el tamaño del microservicio, la cohesión de la arquitectura, la complejidad de la dependencia entre microservicios, entre otros. Los resultados obtenidos por los autores son bastante positivos, y al aplicar la propuesta en un sistema desarrollado con una arquitectura basada en microservicios lograron identificar diferentes puntos de mejora, además de servir como base para la construcción de módulos adicionales.

Además, en el trabajo realizado por (Pietrantuono y col., 2020) podemos ver otro aspecto importante al momento de trabajar con microservicios: el testing. Como mencionan los autores, asegurar la confiabilidad del sistema desarrollado es crucial para los gerentes de la organización. Es por ello que

propusieron un método denominado EMART para llevar a cabo la evaluación de la confiabilidad del sistema en un entorno operacional. Dicha herramienta aprovecha los casos de test definidos por los desarrolladores, pero también es capaz de generar algunos casos representativos, de manera que su monitoreo durante la fase de evaluación sea efectiva. Los resultados que obtuvieron los autores fueron positivos, y determinaron que su propuesta puede ser usada con éxito para realizar la evaluación de la confiabilidad de un sistema con una arquitectura basada en microservicios, dando un resultado bastante acertado, y señalando los aspectos que se deben mejorar en el sistema analizado.

Así mismo, se propuso un flujo de trabajo que automatice el testing del software en el trabajo de (Kargar & Hanifizade, 2018). Este flujo propuesto aprovecha diferentes herramientas usadas para construir entornos de CI/CD, como GitLab, Jenkins, y Docker. Su objetivo es mejorar la calidad y estabilidad del software al tener una ejecución de testing de regresión cada vez que se tiene un candidato a producción. De esta manera, se puede tener un mecanismo de seguridad adicional que puede detectar bugs antes de que estos se manifiesten cuando los clientes utilicen el sistema. Además, en el flujo que proponen los autores se tiene un nodo final que en base a los resultados del testing identifica en qué servicios se produjeron las fallas, de manera que los desarrolladores puedan conocer inmediatamente el origen del problema.

Por otro lado, es importante considerar cómo es que alrededor del mundo cada vez más empresas y proyectos decidieron dejar de construir monolitos para pasar a tener una arquitectura más granular. Sobre este punto (Miguel y col., 2022) mencionan que la empresa de streaming Netflix jugó un rol importante en este aspecto, al decidir ser una de las primeras en adoptar el nuevo paradigma arquitectónico que eran los microservicios. El rápido crecimiento de la empresa fue demostración de el impacto que puede llegar a tener en una empresa seguir este paradigma, en el que pequeños grupos de desarrolladores se encargan de un servicio y se encargan de él a lo largo del ciclo de vida de desarrollo de software.

A lo anterior mencionado se puede sumar el hecho de que tener una arquitectura basa en microservicios permite que estos sean autónomos e independientes, como lo indican (Gan & Delimitrou, 2018;

Heinrich y col., 2017). En sus respectivos trabajos cada uno resalta las ventajas de que cada servicio sea una unidad aislada y tecnológicamente agnóstica que se puede comunicar con otros servicios y responder a las solicitudes de los clientes a través de interfaces como REST APIs y formatos universales como JSON. Adicionalmente, la propuesta de (Khazaei y col., 2020) concuerda y expande el trabajo realizado por (Heinrich y col., 2017). Ambos hacen énfasis en que el principal objetivo de seguir esta arquitectura es mejorar y agilizar el proceso de desarrollo e implementación del software; y también que una vez que se tiene implementada esta arquitectura algo que no se debe dejar de lado es el monitoreo del desempeño de los servicios implementados, de manera que en futuras implementaciones se tenga un mejor modelo de microservicios.

Adicionalmente se encontró un trabajo en el que los autores mencionan que debido a la naturaleza de los microservicios es difícil determinar qué tamaño deben tener, ya que en diferentes casos de uso puede llegar a tener diferentes significados. En la propuesta de (Jamshidi y col., 2018) se detalla los diferentes argumentos para cada posible definición, pero debido a la flexibilidad inherente que se quiere tener con los microservicios esto se deja a discreción de los desarrolladores y modelo de negocio de la empresa en la que se implementan. Además, mencionan que tener un backend con una arquitectura de microservicios no es suficiente para aprovechar los beneficios que ofrece, sino que también la interfaz debe ser modular, y agrupar los microservicios que estén relacionados entre sí de manera que se alcance un verdadero entorno de microservicios a lo largo de todo el stack de desarrollo.

Un ejemplo de lo mencionado en el trabajo anterior se puede ver reflejado en la propuesta de (Naik y col., 2021), quienes desarrollaron una plataforma educativa que comprende nada más 3 microservicios, uno para la gestión de los usuarios, otro para los cursos, y finalmente uno para la gestión de pagos. En su propuesta los autores además aprovechan otras herramientas usadas en los entornos cloud como Docker y Kubernetes, de manera que se llegue a tener un mayor control sobre la manera en la que se realiza la implementación de los microservicios que desarrollaron. Al finalizar la etapa de

validación,su propuesta alcanzó los objetivos planteados, resaltando las ventajas de la utilización de microservicios en conjunto con contenedores en un entorno cloud.

2.2. Bases Teóricas de la Investigación

2.2.1. Enterprise Resource Planning

En primer lugar, debemos entender lo que es un sistema de planificación de recursos empresariales (ERP por sus siglas en inglés). Según (Oracle, 2021), un sistema ERP es aquel software que se usa en las organizaciones para gestionar las actividades del día a día que forman parte del negocio, como las referentes a contabilidad, compras, logística, entre otras. Es debido a esto que existe una gran variedad de sistemas ERP en el mercado, algunos que cuentan con una gran reputación como SAP, que según su página web (SAP, s.f.), son el líder del mercado de software de aplicaciones empresariales. Sin embargo, existe la opción de que las empresas desarrollen su propio sistema ERP, de manera que este sea construido en base a las necesidades de la empresa y no se tenga que pagar por la compra y soporte de los sistemas de grandes proveedores.

En dicho escenario, se puede tener el software que la empresa necesita con precisión. Sin embargo, esto implica una gran cantidad de decisiones que se deben tomar para el desarrollo de este. Se debe considerar la arquitectura general del sistema, el motor de base de datos, entre otros aspectos propios de un proyecto de desarrollo de software. Dejando de lado el motor de base de datos debido a que no es un factor que afecte significativamente la elaboración de la arquitectura, se tiene dos elementos fundamentales de todo sistema, que resaltan más cuando se trata de aplicaciones web: el frontend, y el backend.

2.2.2. Frontend

En lo referente al frontend, es el lado que interactúa con el cliente. Esto quiere decir que comprende todo lo referente a la interfaz, el diseño y experiencia del usuario en la aplicación, así como la interacción con el usuario, de acuerdo con (Singhal y col., 2021). En el caso de aplicaciones web se

utiliza principalmente HTML, CSS, y JavaScrip. Dichos lenguajes de programación permiten cumplir con todos los requerimientos de la interfaz. Además, para facilitar el desarrollo de la interfaz, con el pasar de los años se han creado diferentes librerías y frameworks como AngularJS, ReactJS, SASS, entre muchos otros. Estas librerías engloban todo el desarrollo del frontend de la aplicación, y de esta manera se podía completar la codificación en un menor tiempo, además de contar con funcionalidades extendidas debido a las herramientas incluidas.

2.2.3. Backend

Por otro lado, tenemos el backend, que es el lado del servidor. Es la parte de la aplicación que procesa los datos e información que se capturan y se presentan por la interfaz. Esta parte de la aplicación no puede ser vista ni manipulada directamente por los usuarios. Dentro del backend se debe considerar aspectos como la conexión e interacción la base de datos, los servicios o funcionalidades que se proveerán a la interfaz, y el procesamiento en general de datos, de acuerdo con (Singhal y col., 2021). Para el trabajo y desarrollo del backend se utilizan lenguajes que se ejecutan del lado del servidor, y por lo general no presentan facilidades para la construcción de interfaces, sino que se enfocan en las operaciones y funcionalidades de la aplicación. Algunos lenguajes usados para el desarrollo del backend son PHP, C++, Java, Python, entre otros.

2.2.4. API

Sin embargo, se necesita una manera para que ambas partes de la aplicación se comuniquen entre sí, y ahí entra a tallar lo que es la API, o en el caso de aplicaciones más antiguas, se puede hablar de un web service. Según (Red Hat, 2017), una API (Application Programming Interface) permite que un producto o servicio se comunique con otros sin necesidad de conoces detalles de la implementación. Esto hace que el desarrollo sea más rápido y flexible. La mejor a que presentan las API sobre los web service tradicionales es que pueden procesar solicitudes en HTTPS, y también con la estructura de datos JSON. Cabe resaltar, que con estos conceptos, todos los web services son API, pero no todas las API son web service.

2.2.5. REST

Es un acrónimo para **RE**presentational **S**tate **T**ransfer, y es un estilo arquitectónico para sistemas distribuidos (Fielding, 2000). Se basa en el concepto de generalidad de las entidades, y el uso de verbos para la diferenciación de los recursos que se invocan a través de las solicitudes Web, como GET, POST, PUT, PATCH, DELETE, entre otros. Este estilo arquitectural es uno de los más utilizados actualmente debido a la facilidad que ofrece a los desarrolladores para la construcción de servicios, sobretodo en una arquitectura cliente-servidor. Es así que el servidor no almacena los estados del cliente, y simplemente responde a las solicitudes que recibe, dejando el control del estado a cada cliente.

2.2.6. Bundler

Un bundler, o más específicamente un "module bundler", es una herramienta que toma archivos de archivos de JavaScript y sus dependencias y las empaqueta en un solo archivo, generalmente utilizado para subirlo a un entorno de producción, de acuerdo con (Kelly, 2018).

2.2.7. Mantenimiento

Finalmente, se debe definir lo que es el mantenimiento. Según (Ganney y col., 2020), el mantenimiento de software es que la actividad que consiste en modificar un producto de software después de que fue puesto en producción o fue entregado al cliente para corregir fallas, mejorar su desempeño, o adaptarlo a un nuevo ambiente. Esta actividad se realiza durante todo el tiempo de vida útil del sistema, y comprende un conjunto de actividades. Esta actividad forma parte del ciclo de vida de desarrollo, y por lo tanto es regular. Sin embargo, una señal de que se están dando problemas es que esta actividad empieza a presentar irregularidades, como la necesidad de realizarla más frecuentemente, o que requiera de mayor tiempo.

2.2.8. JSON

JavaScript Object Notation (JSON) es, según (JSON, s.f.), un formato de intercambio de datos que se basa en ser ligero, universal, y sencillo de escribir tanto para humanos como para computadoras. Este formato nace dentro de las definiciones de JavaScript, sin embargo ha demostrado ser extremadamente útil como un formato de transferencia de información universal, ya que su estructura de llave-valor es soportada en prácticamente todos los lenguajes de programación modernos. Es debido a su simplicidad que se adoptó rápidamente en reemplazo del formato XML.

2.2.9. JWT

Según (Auth0 & Peyrott, s.f.), JSON Web Token (JWT) es una manera compacta e independiente para transmitir de manera segura información entre dos entes mediante un objeto JSON. Los JWT se encuentran definidos en el estándar RFC 7519 (Jones y col., 2015). Un JWT consta de tres partes:

- La cabecera: contiene información sobre el token, como el algoritmo que se usó para la firma.
- El payload: contiene datos sobre el usuario que se autenticó, como por ejemplo su código, o nombre de usuario, además de el tiempo de validez del token.
- La firma: comprende el hash generado por el algoritmo elegido para la creación del token de la cabecera, el payload, y la clave privada definida al momento de crear el token.

Los JWT son seguros ya que de manipularse cualquiera de sus secciones la firma ya no será válida, por lo que el servidor de autenticación puede determinar fácilmente si un token es auténtico o no. Además, el hash generado no es reversible, por lo que tenemos la seguridad de que la firma es genuina.

2.2.10. React JS

React es una librería de JavaScript para el desarrollo de interfaces de usuario (Facebook Open Source, s.f.). Fue desarrollada en Facebook que fue implementada por primera vez en el 2011, dos años más tarde fue publicada como open-source y hoy en día es mantenida tanto por Meta como

por la comunidad. Actualmente, según la encuesta anual de Stack Overflow (Stack Overflow, 2021) es el "web framework" más utilizado a nivel mundial.

2.2.11. Go

También llamado Golang, es un lenguaje de programación creado en Google en el 2007, que desde que se hizo publico fue adoptado por diversas empresas debido a su rapidez y capacidad de desarrollo de aplicaciones robustas, concurrentes, y escalables (Google, s.f.). Es sintácticamente similar a C, debido a que uno de sus creadores es Ken Thompson, quien colaboró en la creación de dicho lenguaje junto a Dennis Ritchie.

3 Marco Metodológico

3.1. Alcances y Limitaciones

3.1.1. Alcance

La necesidad de desarrollar la presente propuesta de investigación es evidente debido que se requiere tener una mejor manera de desarrollar el sistema de ERP de la UCSM, debido a que en su situación actual cuenta con deficiencias en su estructura. Esto ocasiona que el desarrollo del sistema sea más complicado debido a que la arquitectura actual no es escalable, y también hace que el sistema requiera de un mayor tiempo de desarrollo. Por estos motivos, el framework a desarrollar deberá cubrir todos estos aspectos para que se pueda realizar el upgrade progresivo del sistema ERP.

Los objetivos del proyecto son obtener en un periodo de 6 meses un framework con una arquitectura basada en microservicios que sea aplicable para llevar a cabo la mejora del backend del sistema ERP de la UCSM. El grado de su aplicabilidad se medirá a través de métricas de desempeño del sistema, así como también la usabilidad del framework para el desarrollo del sistema.

En base a lo anteriormente mencionado, se define las siguientes fases para la elaboración del proyecto, dentro de las cuales se define el alcance de cada una:

a) Revisión bibliográfica

   La revisión bibliográfica se limitará a los artículos relacionados al tema que hayan sido publicados en los últimos 5 años en revistas o repositorios de alto impacto, como IEEE, ACM digital library, Scopus, o Web of Science.

b) Análisis de la funcionalidad del módulo a refactorizar

Se elegirá un modulo del sistema ERP, y se analizará las funcionalidades con las que cuenta de manera que se puedan abstraer para la elaboración del framework. Este análisis estará limitado a las interacciones necesarias dentro del sistema, y no considerará la lógica del negocio ya que no es un factor determinante.

c) Elaboración de la arquitectura y framework

Se elaborará el planteamiento del diseño de arquitectura y framework, que comprenderá el diseño del backend del sistema, y se emitirá una recomendación para el desarrollo de las interfaces.

d) Refactorización del módulo elegido

Se realizará la codificación del módulo considerando la arquitectura y framework propuesto. Dicho módulo operará con datos de prueba obtenidos de la base de datos real, de manera que se respete la implementación y los requerimientos de este.

e) Validación de la propuesta

La validación del módulo servirá para evaluar la arquitectura propuesta, por lo que se realizará en base a métricas como tiempo de respuesta, tolerancia a la concurrencia, entre otros.

Por otro lado, el framework debe cumplir con los requerimientos de usabilidad, por lo que se evaluará este criterio a través de una encuesta de opinión a los trabajadores de la unidad de implementación del ERP de la UCSM para que emitan su opinión sobre la propuesta.

3.1.2. Limitaciones

Las limitaciones que afectan al proyecto son las restricciones propias del servidor donde se encuentra alojado el ERP de la UCSM, además de no contar con libertad total para el momento de realizar las pruebas e implementación del módulo propuesto debido a que en dicho servidor se encuentran otros sistemas que no pueden dejar de funcionar.

3.2. Tipo y Nivel de Investigación

3.2.1. Tipo de Investigación

En primer lugar, es importante mencionar que la presente investigación es de carácter experimental, ya que como indican (Hernández Sampieri y col., 2010) se quiere determinar el impacto que tienen una o más variables independientes sobre una dependiente. En este caso se planteó la siguiente hipótesis: "Utilizar un framework con una arquitectura basada en microservicios y con tecnologías modernas mejora la productividad de los desarrolladores".

3.2.2. Nivel de Investigación

El nivel de la investigación según sus características es exploratorio, debido a que se busca conocer más acerca de los servicios desarrollados con las tecnologías elegidas y la manera en la que pueden beneficiar el trabajo en una organización. Además, es transversal debido a que se busca evaluar los resultados en un momento determinado, más no su evolución en el tiempo.

3.3. Población y Muestra

3.3.1. Población

La población que se define es el equipo de desarrollo de la Unidad de Implementación del ERP de la UCSM, con un valor de N = 7.

3.3.2. Muestra

La muestra que se considerará para el estudio, son aquellos profesionales dedicados al desarrollo y mantenimiento de módulos críticos del ERP, con un valor de n = 3.

3.3.3. Métodos, Técnicas e Instrumentos de Recolección de Datos

El presente trabajo será diseñado tomando un enfoque mixto, ya que tiene aspectos cualitativos y cuantitativos. Esto debido a que se utilizará mediciones numéricas para la validación y medición del objetivo; además de recurrir a las experiencias de los participantes en el estudio para contribuir en la construcción del conocimiento, como mencionan (Hernández Sampieri y col., 2010).

3.3.4. Técnicas de Recolección de Datos

La recolección de datos para la presente investigación se realizará a través de dos técnicas:

- Encuestas

  Se aplicará esta técnica para conocer de manera anónima y rápida las opiniones de los participantes en el estudio que tiene que ver con la evaluación de la usabilidad del framework que se propone.

- Análisis estático y dinámico de un sistema

  Se aplicará esta técnica para poder conocer la manera en la que se desempeña el módulo refactorizado en un entorno de producción, de manera que se puedan evaluar sus métricas en tiempo de respuesta, tolerancia a concurrencia, entre otros.

- Juicio de expertos

  Debido a las limitaciones con el número de elementos de la población y muestra, se utilizará la técnica Delphi para la recolección y evaluación de los datos obtenidos de la encuesta.

3.3.5. Instrumentos de Recolección de Datos

Los instrumentos de recolección de datos que se utilizarán para la presente investigación son los siguientes:

- Cuestionario

  Será a través de 4 preguntas cerradas bajo la escala de Likert y dos preguntas abiertas. Para esto se utilizará Google Forms, y las preguntas planteadas se pueden ver en el Anexo A.

- Logs del servidor y scripts complementarios

  Se recuperarán los registros de las solicitudes que almacena el sistema de manera automática, además de incluir los resultados de scripts y herramientas adicionales que registren información que sea de relevancia para la evaluación adecuada de todas las métricas correspondientes.

3.3.6. Técnicas de Procesamiento de Datos

En cuanto a los resultados de las encuestas la técnica que se utilizará es la estadística descriptiva, de manera que se pueda presentar los resultados obtenidos de una manera sencilla y visual a través de gráficos de barras. De esta manera la comunicación de los resultados obtenidos será más efectiva. Así mismo, el framework será evaluado a través de las métricas internas de la calidad del producto de software establecidas en el estándar ISO 9126. Debido a la naturaleza del sistema al que se aplicará el framework, se seleccionó las métricas detalladas en las Tablas 3.1, 3.2, y 3.3. Por otro lado, para el análisis de los datos recopilados del análisis estático y dinámico de un sistema se utilizará las pruebas de análisis estadístico de Shapiro Wilk y de t de Student. Primero a través de la prueba de Shapiro Wilk se determinará si es que los datos siguen una distribución normal, si es así, se aplicará la prueba de t de Student para determinar si los resultados obtenidos del análisis entre el sistema actual y el sistema nuevo presentan diferencia significativa. En caso que los datos no presenten distribución normal se aplicará la prueba de U de Mann-Whitney. Además, se elaborará una distribución de frecuencias y diagramas de cajas y bigotes para analizar la distribución de los datos obtenidos.

3.3.7. Herramientas para el Procesamiento de Datos

Para llevar a cabo la tabulación y análisis de los datos, al igual que la obtención de los gráficos respectivos, se utilizará la aplicación de hojas de cálculo. En cuanto a la realización de las pruebas estadísticas correspondientes para la validación de las pruebas se utilizará el software estadístico SPSS.

Tabla 3.1

Detalle de la métrica de adecuidad

Nombre	Completitud de la implementación
Propósito	Evaluar qué tan completa está la implementación del módulo
Método de aplicación	Contar la cantidad de requisitos funcionales propios del módulo que faltan implementar
Fórmula	$X = 1 - \frac{A}{B}$ A = número de funciones faltantes B = número de funciones especificadas
Interpretación	$0 \leq X \leq 1$ Entre más cercano 1, más completo
Tipo de Escala	Absoluta
Tipo de medida	A = contador B = contador
Fuente de medición	Especificación de requisitos Código fuente

3.4. Estudio económico

El presupuesto para el proyecto consta de los gastos del uso de las herramientas de trabajo. Los gastos considerados para el proyecto se detallan en la Tabla 3.4.

Luego, se realizó el análisis de costo beneficio del proyecto, que se puede ver en la Tabla 3.5. Como se observa en dicha tabla, el proyecto si es viable; ya que a pesar de que no produce ingresos en sí reduce los gastos de la institución, generando así un ahorro en los tiempos de trabajo. Además, revalorizará el sistema en un monto proporcional a lo que costó realizarlo.

Tabla 3.2

Detalle de la métrica de eficiencia

Nombre	Tiempo de desarrollo
Propósito	Evaluar tiempo requerido para completar una tarea
Método de aplicación	Realizar una medición del tiempo necesario para actividades de desarrollo con el framework propuesto
Fórmula	X = tiempo
Interpretación	Entre más corto, mejor
Tipo de Escala	Proporcional
Tipo de medida	X = tiempo
Fuente de medición	Logs del sistema operativo Logs de herramientas de trabajo

Tabla 3.3

Detalle de la métrica de mantenibilidad

Nombre	Tiempo de mantenimiento
Propósito	Evaluar tiempo requerido para corregir errores o cambiar aspectos
Método de aplicación	Realizar una medición del tiempo necesario para actividades de mantenimiento con el framework propuesto
Fórmula	X = tiempo
Interpretación	Entre más corto, mejor
Tipo de Escala	Proporcional
Tipo de medida	X = tiempo
Fuente de medición	Logs del sistema operativo Logs de herramientas de trabajo

Tabla 3.4

Detalle de los costos mensuales para el desarrollo del proyecto

Elemento	Costo
Servicio de internet	180
Electricidad	200
Total	**380**

Tabla 3.5

Análisis del costo/beneficio de realizar el proyecto

Categoría	Descripción	Tipo	Monto (S/)
Costos	Electricidad	Costo indirecto variable	200 mensual
	Servicio de internet	Costo indirecto fijo	180 mensual
	Costo de trabajo	Costo directo fijo	450 mensual
	Total costos		**830 mensual**
Beneficios	Reducción en tiempo de mantenimiento al sistema	Ahorro variable	2600[1]
	Revalorización del sistema	Ingreso variable	30000[2]
	Reducción en el tiempo de procesamiento de solicitudes al servidor	Ahorro variable	-
	Reducción en el tiempo de espera para la solución de errores	Ahorro variable	-
	Reducción en el tiempo requerido para el cumplimiento de tareas administrativas	Ahorro variable	-
	Total beneficios		**32600**

[1] De acuerdo con el cálculo mostrado en la tabla del anexo B, tomando en cuenta 4 desarrolladores principales.

[2] De acuerdo con el cálculo mostrado en la tabla del anexo C.

3.5. Plan del proyecto

Para el desarrollo del proyecto se eligió la metodología SCRUM, de manera que se tenga un trabajo iterativo e incremental. Para esto, se ha definido el trabajo en 5 fases, cada una de las cuales será un sprint. Las fases definidas y las actividades que se considera por cada una de ellas se puede ver representado en un diagrama de Gantt en la Figura 3.1.

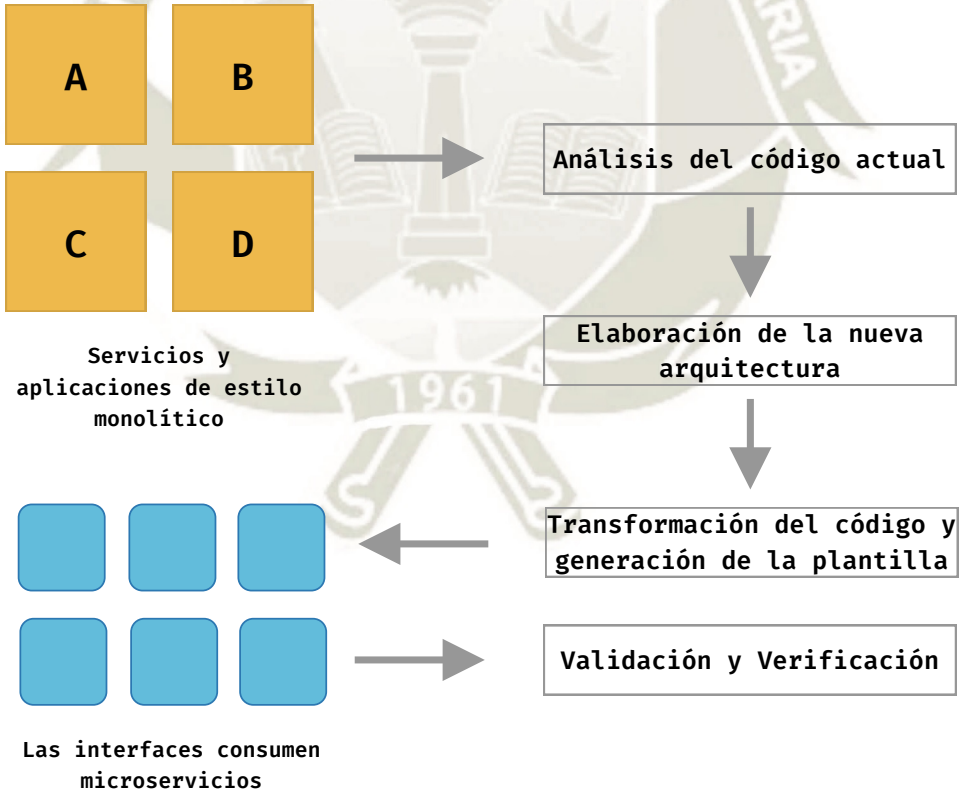
Figura 3.1. Planificación del proyecto presentada en un diagrama de Gantt

N°	ACTIVIDAD	DURACIÓN (DÍAS)	FASE 1				FASE 2		FASE 3			FASE 4		FASE 5	
			S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
1	**Revisión bibliográfica**														
1.1	Búsqueda y lectura de artículos científicos	8													
1.2	Elaboración del estado del arte	10													
1.3	Elaboración del marco teórico	10													
2	**Análisis de la funcionalidad del módulo a refactorizar**														
2.1	Selección del módulo	3													
2.2	Análisis de sus casos de uso y modelo de datos	7													
2.3	Análisis de la interacción que tiene con el backend	7													
3	**Elaboración de la arquitectura y**														
3.1	Selección de tecnologías y librerías	7													
3.2	Diseño de la arquitectura para el backend	7													
3.3	Construcción del framework	7													
4	**Refactorización del módulo**														
4.1	Configuración del entorno de desarrollo	3													
4.2	Codificación del módulo utilizando el framework desarrollado	10													
4.3	Fase de pruebas locales	3													
5	**Validación de la propuesta**														
5.1	Medición del desepeño del sistema	7													
5.2	Evaluación de la usabilidad del framework	7													

4 Desarrollo de la propuesta

Dentro del presente capítulo se desarrolla los aspectos correspondientes al desarrollo de la propuesta. Las fases que se proponen se obtuvieron tras la revisión del estado del arte, en donde se observó que todas las propuestas siguen el mismo procedimiento en general con ligeras modificaciones. Sin embargo, la propuesta más representativa es la de (Wolfart y col., 2021), quienes establecieron una hoja de ruta para la modernización de sistemas. El esquema resumen de la metodología se puede ver en la Figura 4.1.

Figura 4.1. Esquema de la metodología de desarrollo, adaptado de (Wolfart y col., 2021)



4.1. Análisis del Código Actual

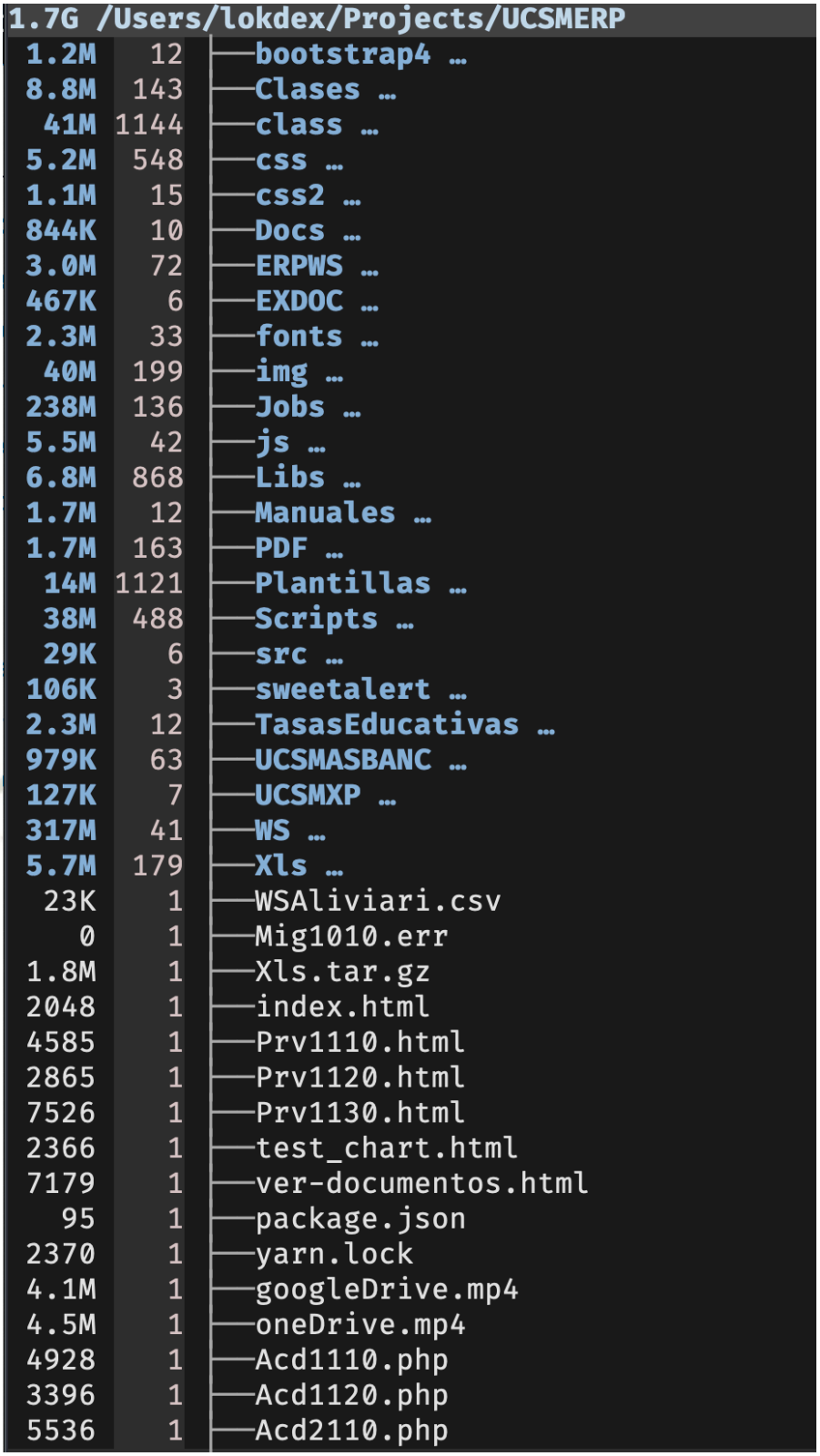
El primer paso es llevar a cabo la etapa de análisis del código actual, durante la cual se busca llegar a entender mejor el sistema que se quiere refactorizar, de manera que se pueda descomponer en microservicios.

Actualmente el sistema ERP de la UCSM se encuentra desarrollado bajo un estilo monolítico, por lo que el repositorio de código fuente es bastante extenso, y presenta una alta entropía. Esto en consecuencia hace que sea cada vez más difícil de mantener, y la solución de problemas requiere de un mayor periodo de tiempo. Esto a su vez, afecta la productividad de los desarrolladores ya que tanto para el desarrollo de nuevas funcionalidades como para las actividades de mantenimiento tienen que lidiar con un repositorio complejo y en consecuencia requieren de más tiempo.

Actualmente se usa la tecnología PHP como base del sistema, y utiliza para las interfaces el motor de plantillas de Smarty. En la Figura 4.2 se muestra el resultado del comando *ls* en la raíz del repositorio. Como podemos ver, se tiene una gran variedad de carpetas y archivos, con 776 elementos en el directorio raíz. Esto hace que ubicar un archivo sea bastante complicado, y por la manera en la que se nombraron los archivos no se puede saber que funcionalidad cubre cada uno. A esto, se suma el hecho de que por el uso de Smarty y de PHP, cada funcionalidad se encuentra dividida en dos archivos, un archivo con el código en PHP que controla la lógica de la vista, y otro archivo en HTML que presenta la interfaz del usuario. Algunas veces se puede llegar incluso a necesitar 3 archivos cuando se tiene algún comportamiento o necesidad más compleja, como por ejemplo en las tablas que presentan información de una manera más detallada o con frecuente actualización. A lo anterior mencionado, se suma el hecho de que se generan archivos de cache, lo que si bien puede ayudar en el tiempo de carga de las interfaces, esto puede ocasionar problemas en el desarrollo al momento de resolver un error.

Como caso de estudio del presente trabajo se realizará la implementación de un módulo para la Botica Aliviari. En dicho centro de costo se presenta un problema con las notas de crédito y/o bonos de descuento que emiten los proveedores. Esto se debe a que el registro de dichos documentos se realiza

Figura 4.2. Captura del listado de archivos y directorios en la raíz del repositorio



de manera manual, y se debe emitir un informe para que en la dirección de contabilidad se realice el pago de los comprobantes respectivos aplicando los descuentos correspondientes. Como punto de comparación se tendrá el módulo implementado bajo el esquema de desarrollo actual, y por otro lado se tendrá una implementación considerando el framework propuesto. El flujo de trabajo actual consiste en que al tener productos que están por caducar en la botica, estos se devuelven al proveedor, quien puede reponerlos, o caso contrario emite una nota de crédito o un bono de descuento para futuras compras al proveedor. Es así que se lleva un registro manual, y cada mes se elabora un informa manualmente por la encargada de la botica que indica que notas de crédito y bonos fueron entregados por los proveedores.

Este informe se envía a la dirección de logística, quienes dan el visto bueno y luego se envía dicho informe a la dirección de contabilidad para que se registre el pago y se den los bonos y las notas de crédito por contabilizadas. Sin embargo, este procesos puede automatizarse y puede simplificarse a través del ERP.

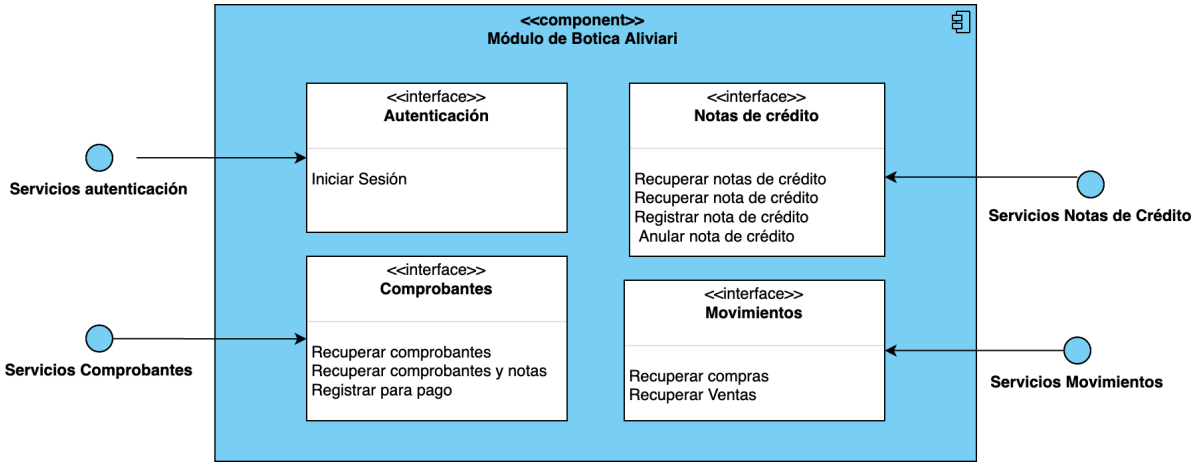
Se llevará a cabo la implementación de un módulo que cuente con 3 opciones principales. La primera que permita recuperar las compras y ventas realizadas en la botica del sistema de facturación electrónica; la segunda opción consistirá en una pantalla de registro de notas de crédito, en la cual se podrá ingresar todos los datos necesarios para dicho documento; y finalmente, la tercera opción permitirá registrar a la base de datos los comprobantes que se verán afectados por las notas de crédito, de manera que la dirección de contabilidad tenga un acceso directo a esa información y la elaboración de un informe ya no sea necesaria.

Por otro lado, se debe mencionar que el sistema ERP actualmente se encuentra construido bajo el paradigma de la programación orientada a objetos, por lo que se tiene archivos en los que se define las clases que se utilizan en el sistema. Estos archivos por lo general suelen exceder las 5000 líneas de código, por lo que el trabajo con este tipo de archivos es tedioso, y genera complicaciones a la hora de desarrollar nuevas funcionalidades o de realizar el mantenimiento a opciones anteriores.

4.2. Elaboración de la Nueva Arquitectura

En base a las necesidades del módulo se puede elaborar una interfaz de servicios, agrupándolos de acuerdo a las entidades con las que van a interactuar. Dicho diagrama se puede ver en la Figura 4.3. En base a la interfaz de servicios podemos separar los servicios requeridos en dos categorías, aquellos

Figura 4.3. Diagrama de interfaz de servicios del módulo para la Botica Aliviari



Nota: Elaboración propia.

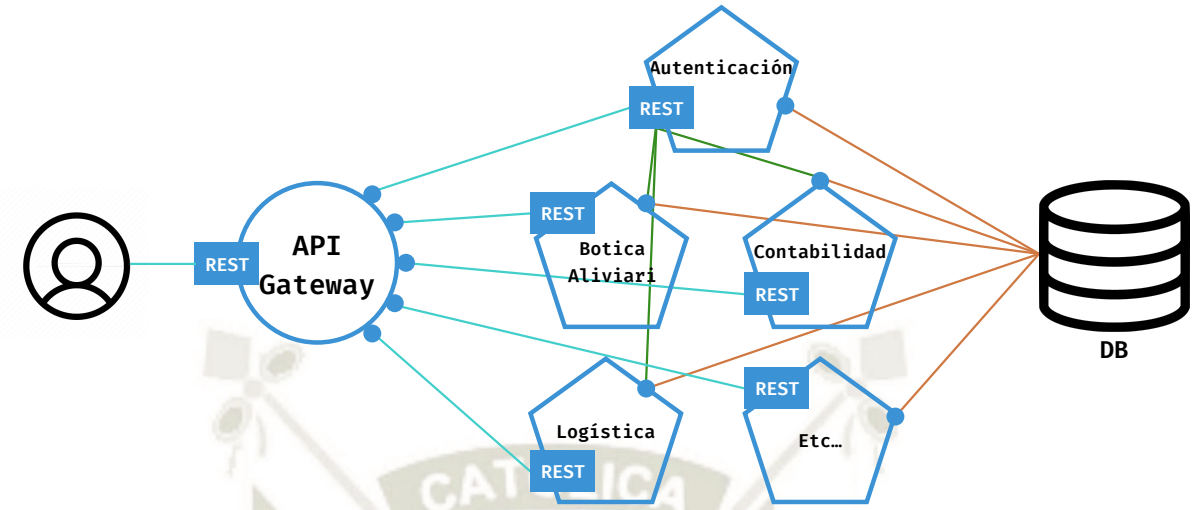
que forman parte de la funcionalidad del módulo en sí, y el servicio de autenticación que puede ser reutilizado por los demás servicios que se implementen en el futuro. Es así que la arquitectura que se tendría para los servicios del ERP se puede ser en la Figura 4.4. En dicha figura podemos observar los diferentes módulos que formarían parte de la arquitectura, y que todos se comunican con el servicio de autenticación. De esta manera se mantiene el código aislado e independiente de otros módulos.

4.3. Transformación del Código y Generación de la Plantilla

Una vez que se tiene definidos la arquitectura y los servicios, el siguiente paso fue el desarrollo de un microservicio, de manera que pueda ser tomado de ejemplo y construir la plantilla de código en base a este.

El primer microservicio que se eligió para ser desarrollado es el de autenticación. Este comprende

Figura 4.4. Diagrama de la arquitectura que tendrían los servicios del ERP



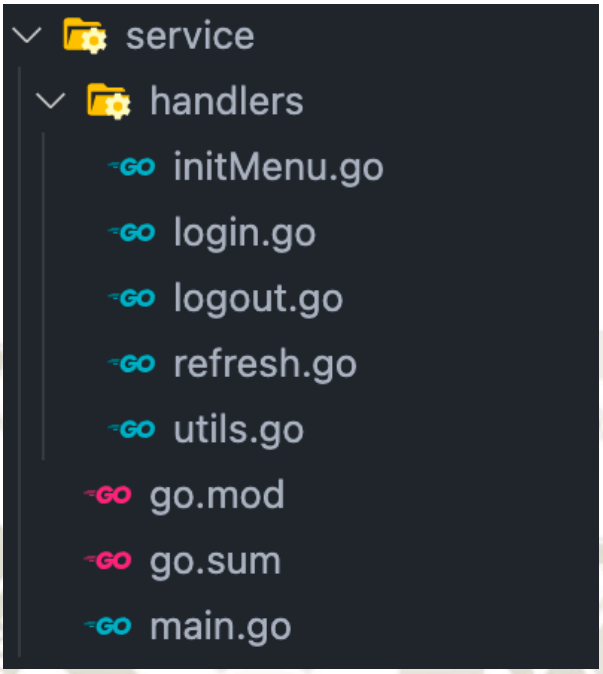
Nota: Elaboración propia.

que se validen las credenciales de un usuario. Al tener una arquitectura basada en microservicios que serán accedidos mediante REST se debe tener un mecanismo que permita gestionar la autorización de acceso a los servicios. Para esto se implementó en servidor los métodos adecuados para generar y verificar JSON Web Tokens (JWT). Manteniendo para el access token un tiempo de 5 minutos y para el refresh token un tiempo de validez de 15 minutos.

Es así que para construir la aplicación se realizó la configuración del lenguaje de programación Go en el entorno de desarrollo, y se creó los directorios correspondientes para el desarrollo. Tras crear los archivos correspondientes se llegó a tener la estructura que se muestra en la Figura 4.5, en donde podemos ver que se tiene el punto de entrada en la raíz del directorio, y dentro de la carpeta handlers se tiene los archivos de las diferentes acciones que están presentes en el servicio de autenticación. Cabe resaltar la existencia del archivo *utils.go*, que contiene funciones y variables globales que son reutilizadas.

Por otro lado, para la interfaz se utilizará ReactJS, debido a que es la librería de desarrollo web más usada a nivel mundial según la encuesta realizada por (Stack Overflow, 2021), ya que ofrece una gran gama de funcionalidades y herramientas a los desarrolladores para crear interfaces dinámicas de una

Figura 4.5. Captura de la estructura de archivos del servicio de autenticación

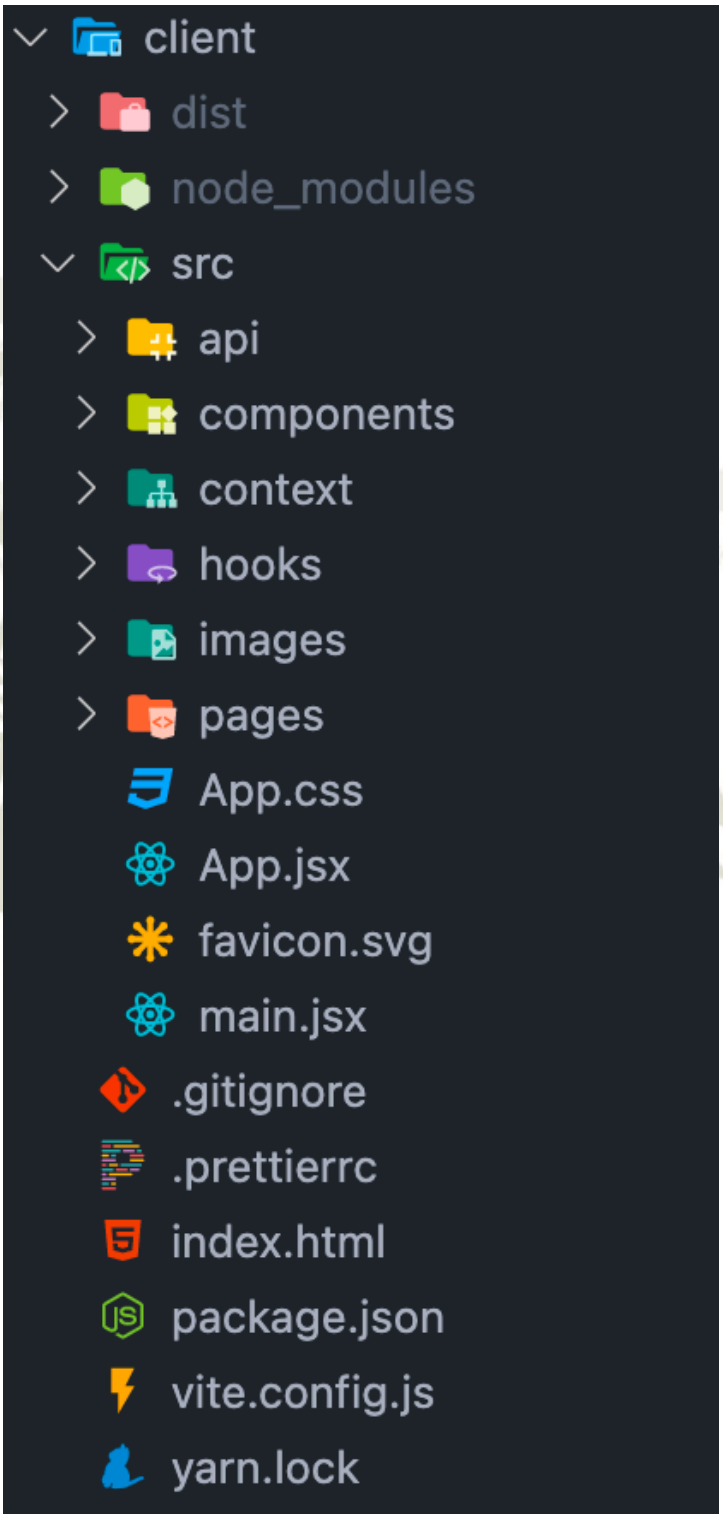


manera sencilla y eficaz, además, al ser la librería más utilizada cuenta con una comunidad bastante activa, por lo que si se tiene problemas o si se desea conocer cómo construir algo se puede encontrar la solución en los diferentes foros de internet sin problemas. Para la creación del proyecto de ReactJS se utilizó la herramienta Vite, que realiza el scaffolding de todos los archivos y configuraciones necesarias para el desarrollo. Además, Vite usa ESBuild como bundler para el servidor de desarrollo, de manera que se tenga un inicio rápido, y cuando se realiza la compilación del bundle de producción utiliza una configuración personalizada de Rollup.

Es así que tenemos la estructura de archivos que se muestra en la Figura 4.6 una vez completado el desarrollo de las interfaces. De esta estructura es importante mencionar que el desarrollo se realizará principalmente en el subdirectorio de pages y de components, mientras que el resto de los archivos contiene funcionalidad adicional para gestionar la autenticación. Además, una vez se complete el desarrollo, se tendrá únicamente 3 archivos para subir al servidor, uno de HTML, uno de JavaScript, y uno de CSS, además de las imágenes que hayamos incluido.

En este punto ya se tiene la estructura básica del proyecto, que puede tomarse de referencia para el

Figura 4.6. Captura de la estructura de archivos de la interfaz de autenticación



desarrollo de los demás microservicios que conformarán el sistema.

Como se puede ver, cada microservicio se enfoca en mantener la cantidad mínima e indispensable de archivos y de código fuente, aprovechando las herramientas que nos ofrecen las tecnologías elegidas lo más que se puede. Además, por la manera en la que está todo organizado, se puede encontrar los archivos sobre los que se necesita trabajar de una manera mucho más rápida y efectiva. En la Figura 4.7 podemos ver el árbol de archivos completo del servicio de autenticación, considerando tanto la interfaz como el backend.

Dentro del proyecto se incluyen así mismo archivos necesarios para un trabajo más eficiente. Por ejemplo se sugiere la utilización de la herramienta Prettier para que formatee el código del frontend de manera automática, por lo que considera un archivo de configuración para dicha herramienta. Además, se tiene el archivo .gitignore para que los directorios donde se descargan las dependencias no sean consideraros en el control de versionamiento.

4.4. Testing

Adicionalmente, se consdiera proporcionar dentro del framework las herramientas necesarias para llevar cabo diferentes tipos de testing, los cuales se detallan a continuación con los casos definidos y sus resultados correspondientes.

4.4.1. Pruebas Unitarias

Para las pruebas unitarias se definieron los siguientes casos de prueba, a través de los cuales se busca asegurar que el sistema está preparado para responder a las solicitudes que se realice desde el cliente. Los casos definidos para las pruebas unitarias se pueden ver en la Tabla 4.1.

A través de la propia librería de testing de Go, se pudo definir diferentes casos de test unitarios para el microservicio desarrollado. Estos más que nada están orientados a probar la validación de parámetros, de manera que se asegure que se está realizando las verificaciones adecuadas. Además, Go nos

Figura 4.7. Captura del árbol de archivos del microservicio de iniciar sesión

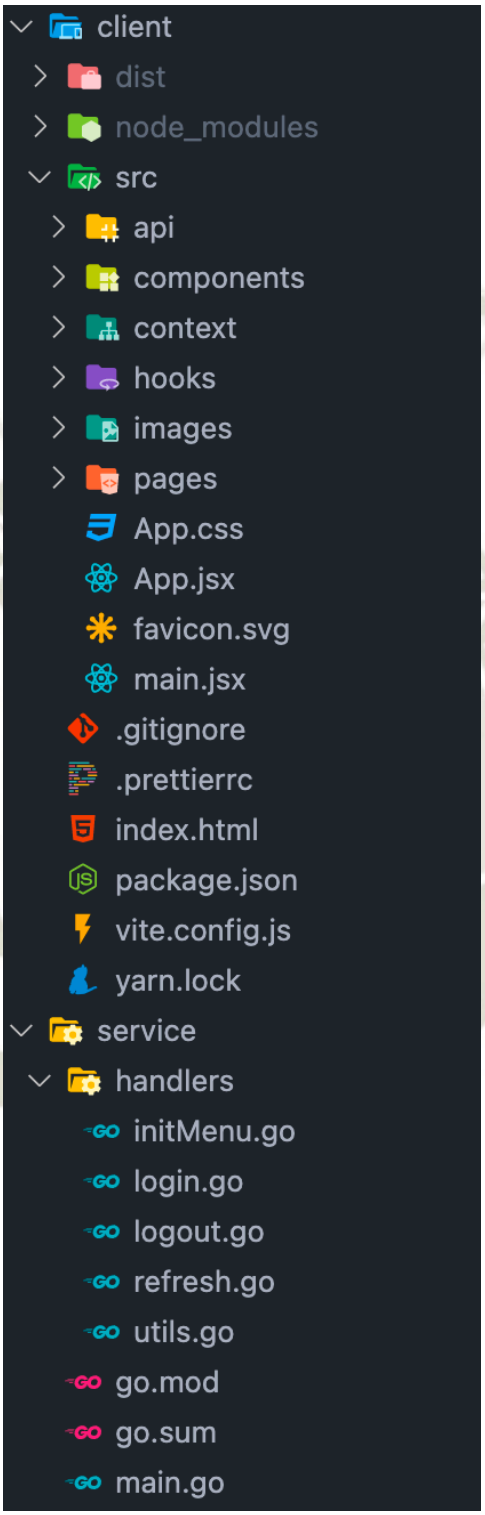


Tabla 4.1

Casos de prueba definidos para las pruebas unitarias

ID	Nombre	Resultado esperado	Estado
PU-01	TestWrongDNILenght	El sistema retorna un mensaje indicando errores en el campo DNI	Exitoso
PU-02	TestMissingDNI	El sistema retorna un mensaje indicando errores en el campo DNI	Exitoso
PU-03	TestMissingClave	El sistema retorna un mensaje indicando errores en el campo Clave	Exitoso
PU-04	TestMissingBoth	El sistema retorna un mensaje indicando errores en los campos DNI y Clave	Exitoso
PU-05	TestCorrectParams	El sistema procede con la ejecución de la consulta	Exitoso

presenta la facilidad de que todos los archivos que terminen con el sufijo *_test* serán ejecutados bajo el comando test, por lo que podemos correr todas las pruebas definidas automáticamente. Además, mediante la misma librería de testing podemos generar un reporte de cobertura de los casos de test que definimos.

En la Figura 4.8 podemos ver la ejecución de los casos de prueba definidos y el porcentaje de cobertura que se alcanzó.

4.4.2. Pruebas Integrales

En cuando a las pruebas integrales, se utilizó la herramienta Jest de ReactJS, ya que a través de ella desde la interfaz podemos probar si es que los datos que regresaron del backend son los adecuados. Para esta prueba se definieron los casos de test definidos en la Tabla 4.2.

Al haber definido esta prueba a través de Jest para probar los resultados desde la interfaz, los resultados de la prueba se muestran más adelante en la Figura 4.9, junto con los resultados de las pruebas funcionales.

Figura 4.8. Ejecución de pruebas unitarias y cobertura

```
ucsmerp-go/auth on  main [!?] via  v1.17.3 via  v16.13.0
❯ make test_project
Ejecución de todos los casos de test definidos para los modelos de datos:
go test -v ./data
=== RUN   TestUserWrongDNILength
--- PASS: TestUserWrongDNILength (0.00s)
=== RUN   TestMissingDNI
--- PASS: TestMissingDNI (0.00s)
=== RUN   TestMissingClave
--- PASS: TestMissingClave (0.00s)
=== RUN   TestMissingBothParams
--- PASS: TestMissingBothParams (0.00s)
=== RUN   TestCorrectParams
--- PASS: TestCorrectParams (0.00s)
PASS
ok      auth/data       (cached)

ucsmerp-go/auth on  main [!?] via  v1.17.3 via  v16.13.0
❯ make test_coverage
Ejecución de la prueba de cobertura con los casos de test definidos:
cd data && go test -cover && cd ..
PASS
coverage: 75.0% of statements
ok      auth/data       0.890s
```

Tabla 4.2

Casos de prueba definidos para las pruebas integrales

ID	Nombre	Resultado esperado	Estado
PI-01	TestIntegracion	El microservicio de autenticación se comunica adecuadamente con el de menú	Exitoso

4.4.3. Pruebas Funcionales

Al haberse desarrollado un microservicio de autenticación, se puede realizar pruebas funcionales al verificar que al enviar diferentes DNIs y contraseñas la API nos retorna la información adecuada para dicho usuario. Para hacer esto, podemos utilizar nuevamente la herramienta Jest, a través de la cual verificamos que los datos que nos presenta el menú de opciones de usuario regresa los valores adecuados. Los casos de prueba definidos se detallan en la Tabla 4.3.

Los resultados de esta prueba se muestran en la Figura 4.9, donde a través de la terminal se ejecutó todos los test definidos con Jest. En dicha figura se puede ver los dos tests funcionales definidos, así como también el test de integración, cada uno dentro de su propio directorio.

Tabla 4.3

Casos de prueba definidos para las pruebas funcionales

ID	Nombre	Resultado esperado	Estado
PF-01	TestAutenticacion	El sistema valida las credenciales correctamente	Exitoso
PF-02	TestInitMenu	El sistema presenta las opciones a las que tiene acceso el usuario	Exitoso

Figura 4.9. Resultados de la ejecución de las pruebas definidas con Jest

```
PASS src/__tests__/functional_tests/menu.test.js
PASS src/__tests__/functional_tests/iniciar-sesion.test.js
PASS src/__tests__/integration_tests/index.test.js

Test Suites: 3 passed, 3 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.844 s, estimated 4 s
Ran all test suites.

Watch Usage: Press w to show more.
```

4.5. Estrategia de Despliegue

Una primera parte de la estrategia de despliegue es la publicación del framework en un repositorio público de GitLab, de manera que pueda ser accedido públicamente, y clonado por los desarrolladores que lo necesiten. De esta manera también se puede gestionar los cambios y actualizaciones en la versión del framework.

En cuanto a la estrategia de refactorización que seguirá para el ERP es una combinación del rolling deployment y canary deployment. Esto debido a que el despliegue de los módulos o funcionalidades refactorizadas se realizará por fases, conforme se vaya completando el desarrollo de las funcionalidades; y a su vez, estas funcionalidades no estarán disponibles a la totalidad de usuarios, sino que se dará acceso a ciertos usuarios al inicio de manera que se pueda probar que todo funciona adecuadamente antes de publicar el módulo en su totalidad. Adicionalmente, a manera de mantener un respaldo de seguridad, ambas versiones se mantendrán en un entorno de producción hasta que se tenga la seguridad de que el módulo refactorizado cumple con todos las funcionalidades requeridas de manera satisfactoria.

5 Análisis y Validación de Resultados

5.1. Validación Cualitativa

Tras haber desarrollado el primer microservicio, se procedió a tener reuniones con las personas pertenecientes al grupo elegido para la muestra del presente estudio.

Los participantes, quienes aceptaron participar del presente estudio, cuentan con el siguiente perfil:

1. Desarrollador 1
    - Egresado de la carrera de Ingeniería de Sistemas de la UCSM hace 3 años
    - Estudiante de maestría en la UCSM
    - 4 años de experiencia en el desarrollo con PHP y HTML
    - En el ERP realiza el desarrollo y mantenimiento de módulos críticos
2. Desarrollador 2
    - Egresado de la carrera de Ingeniería de Sistemas de la UCSM hace 1 año y medio
    - 2 años de experiencia en el desarrollo con PHP y HTML
    - En el ERP realiza el desarrollo y mantenimiento de módulos críticos
    - Recopila requerimientos para el desarrollo de nuevas funcionalidades o módulos
3. Desarrollador 3
    - Egresado de la carrera de Ingeniería de Sistemas de la UCSM hace 1 año y medio
    - 2 años de experiencia en el desarrollo con PHP y HTML

- En el ERP realiza el desarrollo y mantenimiento de módulos críticos
- Desarrolla, mantiene, y gestiona los web service internos y externos de la UCSM

A dichas personas se les presentó el framework desarrollado para conocer sus opiniones. Se les explicó los aspectos técnicos y el flujo de trabajo que se suele tener con estas herramientas de acuerdo a lo revisado en el estado del arte y en las buenas prácticas. Posteriormente, se les realizó las preguntas planteadas en el cuestionario.

Como fruto de los resultados de la encuesta, se obtuvo como resultados de las preguntas cerradas los valores mostrados en la Tabla 5.1.

En cuanto a las preguntas abiertas, se obtuvo la validación correspondiente. Todos los participantes

Tabla 5.1

Resultados de las preguntas cerradas de la encuesta aplicada, donde 5 es "totalmente de acuerdo" y 0 es "totalmente en desacuerdo"

Ítem	**Valoración Promedio**
El framework presentado puede mejorar la calidad de software que usted desarrolla	4.67
El framework presentado facilitaría las tareas de mantenimiento de software	4.67
El framework presentado le parece sencillo de aprender y utilizar	4.67
Usted se adaptaría fácilmente a un flujo de trabajo utilizando el framework presentado	4.67

respondieron que el framework propuesto mejoraría de manera significativa el tiempo de desarrollo. Resaltando las facilidades que se tiene para la validación de datos, así como también la facilidad de crear proyectos utilizando la base presentada, y también la simplicidad de la puesta en producción ya que los archivos compilados son independientes.

Así mismo, al presentar la propuesta al Ing. Fernando Paredes Marchena, jefe del área en la que se desarrolló el presente trabajó, manifestó su conformidad. Por lo que se utilizará el framework propuesto para proyectos del área, como se puede ver en la carta del anexo D.

5.2. Validación cuantitativa

Debido a la naturaleza del sistema que se está desarrollando, se realizaron algunos tipos de test para asegurar que el framework desarrollado mantendrá los estándares requeridos por la organización. Para realizar las pruebas se utilizó la herramienta de K6, que como se indica en (Grafana, 2021), es una herramienta de testing de carga moderna diseñada para desarrolladores y testers de la era moderna. Dicha herramienta nos permite realizar diferentes pruebas y analizar diferentes métricas de una manera muy sencilla, al igual que simular diferentes escenarios.

5.2.1. Pruebas en Entorno de Desarrollo

Prueba de carga

En una primera instancia se realizó una prueba de carga al microservicio desarrollado, la misma que se aplicó al sistema actual. Esta prueba simplemente mide el tiempo de respuesta de una solicitud GET para ingresar a la página de inicio de sesión. Ambas pruebas se llevaron a cabo en un entorno de desarrollo, sin embargo, pueden reflejar el comportamiento que se tendrá en un entorno de producción. Los resultados completos de esta primera prueba se pueden ver en el anexo E.1 para el sistema actual, en el anexo E.2 para el sistema desarrollado con el framework propuesto, y en la Tabla 5.2 se muestra la comparación de los resultados entre el microservicio desarrollado con el framework propuesto y el sistema en su estado actual. En esta primera prueba, se puede ver que el uso del framework propuesto presenta una mejora de más del 95 % en todos los casos en comparación con los valores obtenidos del sistema actual.

Luego, se pasó a realizar una prueba de carga en la que se tenga una interacción con la base de datos, de manera que se pueda tener una simulación de cómo será la interacción el sistema. Para esto, se realizó la prueba de carga simulando 1 único usuario, 10 usuarios simultáneos, y 200 usuarios simultáneos, quienes realizarán la consulta de “Iniciar sesión” de manera repetitiva durante 10 segundos. Los resultados de ejemplo se pueden ver en el anexo E.3 para el sistema actual y en el anexo E.4 para el framework propuesto.

Tabla 5.2

Comparación de la prueba de carga realizada

Ítem evaluado	Tecnología		Mejora (%)
	Sistema Actual	Framework Propuesto	
Data recibida	57 kB	6.75 kB	88.16
Tiempo promedio de respuesta del servidor	37.63 ms	23.06 ms	38.72
Promedio de la duración de la solicitud http	37.97 ms	23.38 ms	38.43

Tabla 5.3

Análisis estadístico de la prueba de carga durante 10 segundos al sistema actual y al sistema desarrollado con el framework propuesto según el número de usuarios

ESTADÍSTICOS	Sistema Actual			Framework propuesto		
	NÚMERO DE USUARIOS					
	1	10	200	1	10	200
n	10	10	10	10	10	10
Promedio (ms)	26,84	162,84	1878,00	16,07	35,76	68,26
DE (ms)	6,40	5,80	53,29	0,20	4,43	14,29
Vmín (ms)	21,58	25,35	16,29	13,84	24,06	34,46
Vmáx (ms)	36,50	198,78	2377,00	28,35	66,50	113,88
Tiempo de mejora (%)		-		37,39	77,99	96,36

Prueba de hipótesis t de Student	**NÚMERO DE USUARIOS**	1	$t_{investigacion} = 5{,}32\ t_{critico} = 1{,}83\ p < 0{,}05$
		10	$t_{investigacion} = 53{,}96\ t_{critico} = 1{,}83\ p < 0{,}05$
		200	$t_{investigacion} = 103{,}73\ t_{critico} = 1{,}83\ p < 0{,}05$

n: número de repeticiones
DE: desviación estándar
Vmín: valor mínimo
Vmáx: valor máximo
Tiempo de mejora(%): $\frac{Duracion de solicitud en sistema actual - Duracion de solicitud en el framework propuesto}{Duracion de solicitud en el framework propuesto} \times 100$

Los resultados obtenidos tras realizar 10 iteraciones se muestran en la Tabla 5.3. A estos datos se aplicó inicialmente la prueba de Shapiro Wilk, a través de la cual se encontró que los resultados se ajustan a una distribución normal con un valor $p > 0,05$. Sabiendo eso, se aplicó la prueba de contraste de t de Student, en donde se encontró que si existe diferencia significativa ($p < 0,05$) entre la duración de la solicitud (ms) del sistema actual y del framework propuesto, encontrando en esta última tecnología menores promedios. Es importante mencionar que el porcentaje de tiempo de mejora es mayor mientras se tenga un mayor número de usuarios concurrentes, siendo en el caso de 200 usuarios concurrentes una mejora de 96,36 %. Así mismo, los datos obtenidos se graficaron para presentar los resultados de una manera visual y sencilla. Dicho gráfico se puede ver en la Figura 5.1.

Prueba de Estrés

En cuanto a las pruebas de estrés se realizó una prueba en la que se simula un escenario que podría presentarse en la realidad. El escenario que se programó con la herramienta de K6 es que en los primeros 30 segundos se tenga una conexión de 200 usuarios quienes realizan solicitudes constantemente, luego en el minuto y medio siguientes este número de usuarios disminuye progresivamente a 100, pero sigue realizando solicitudes al servidor, y finalmente, en los últimos 20 segundos se tiene una desconexión progresiva de todos los usuarios.

En el anexo E.5 se puede ver una captura de ejemplo de la prueba de estrés realizada al sistema actual y en el anexo E.6 se puede ver para el framework propuesto. Y los resultados que se obtuvieron de esta prueba se pueden ver en la Tabla 5.4.

De igual manera que con los datos de la prueba de Shapiro Wilk para verificar si los datos presentan una distribución normal. Al ser así podemos aplicar la prueba de t de Student, con la cual se obtuvo que existe una diferencia significativa ($p < 0,05$) entre la duración de la solicitud (ms) del sistema actual y del framework propuesto, encontrando que en esta última tecnología se tiene un menor tiempo promedio.

Así mismo, el framework propuesto presenta un mayor porcentaje de respuestas exitosas (99,86 %),

Figura 5.1. Promedio de la duración de la solicitud del sistema actual y el framework propuesto según el número de usuarios

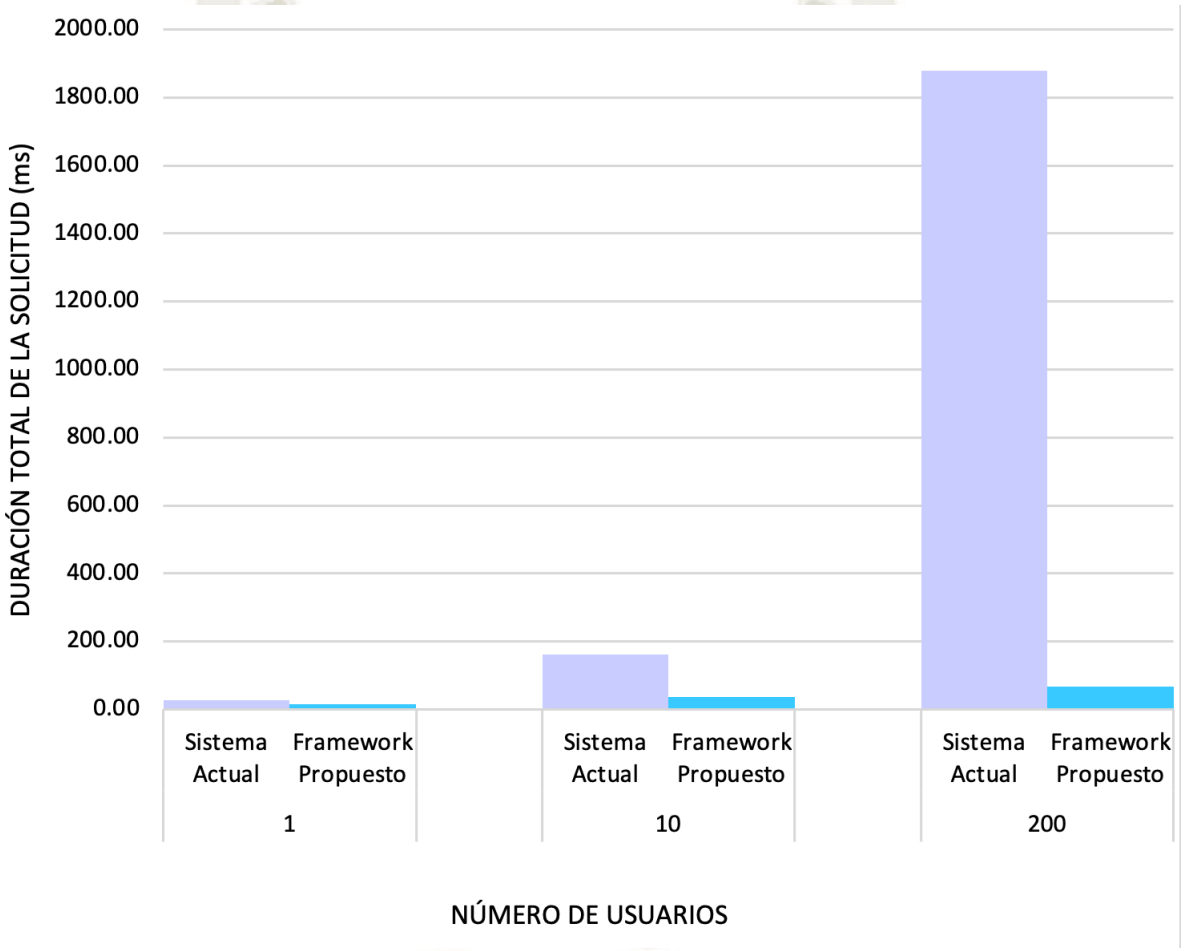


Tabla 5.4

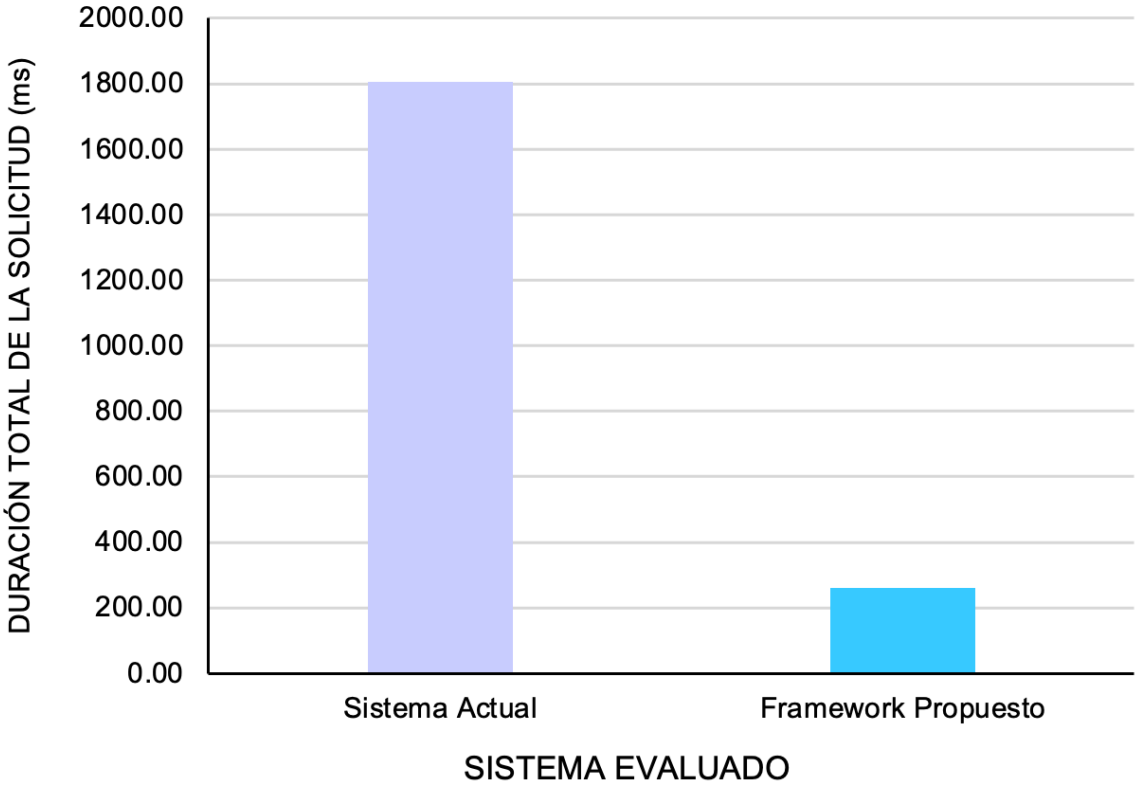
Análisis estadístico de la prueba de estrés con 200 usuarios al sistema actual y al framework propuesto

ESTADÍSTICOS	**200 USUARIOS**	
	Sistema Actual	Framework Propuesto
n	10	10
Promedio (ms)	1807,00	261,58
DE (ms)	65,16	34,61
Vmín (ms)	15,59	13,80
Vmáx (ms)	2909,00	4488,00
Tiempo de mejora (%)	-	85,50
Respuesta exitosa (%)	98,53	99,86
Respuesta fallida (%)	1,47	0,14
Prueba de hipótesis t de Student	Prueba de estrés	$t_{investigacion}$ = 66,24 $t_{critico}$ = 1,83 p < 0,05
	Respuesta exitosa	$t_{investigacion}$ = 36,61 $t_{critico}$ = 1,83 p < 0,05

n: número de repeticiones
DE: desviación estándar
Vmín: valor mínimo
Vmáx: valor máximo
Tiempo de mejora(%): $\frac{Duraciondesolicitudenelsistemaactual - Duraciondesolicitudenframeworkpropuesto}{Duraciondesolicitudenelsistemaactual} \times 100$

existiendo también diferencia significativa ($p < 0,05$) en relación con la tecnología actualmente implementada, teniendo una mejora del 85,50 %. Así mismo se graficaron los resultados obtenidos para su mejor visualización, lo que se puede ver en la Figura 5.2.

Figura 5.2. Promedio de la duración de la solicitud en una prueba de estrés con 200 usuarios al sistema actual y al framework propuesto



5.2.2. Pruebas en Entorno de Producción

Prueba de estrés

Para tener una mejor representación estadística de la prueba de estrés realizada al servidor de producción, se realizaron 40 iteraciones con K6 configurado a 200 VU (virtual users). De esta manera al realizar el análisis correspondiente podemos tener una mayor certeza de los datos. Los resultados de esto se pueden ver en la Tabla 5.5. En la Tabla 5.5 se observa que el 12,50 % de iteraciones se registran

Tabla 5.5

Distribución de frecuencias de la duración de solicitud promedio (ms) de pruebas de estrés al sistema actual

DURACIÓN DE SOLICITUD PROMEDIO (ms)	NÚMERO DE ITERACIONES (/200 VU)	NÚMERO DE ITERACIONES ACUMULADO (/200 VU)	PORCENTAJE (%)	PORCENTAJE ACUMULADO (%)
6789 - 6875	6	6	15,00	15,00
6876 - 6962	9	15	22,50	37,50
6963 - 7049	11	26	27,50	65,00
7050 - 7136	2	28	5,00	70,00
7137 - 7223	7	35	17,50	87,50
7224 - 7310	5	40	12,50	100,00
TOTAL	**40**		**100,00**	

VU: virtual user generado por K6

en el último intervalo que va de 7224 ms a 7310 ms. Es importante mencionar que en esta distribución la amplitud de intervalo es de 87 ms. Así mismo, si se considera el porcentaje acumulado, el 65,00 % registraron una duración promedio entre 6789 ms y 7049 ms. En la Tabla 5.6 se observa que el 5,00 % de iteraciones se registran en el último intervalo que va de 4219 ms a 4250 ms. En este caso la distribución presenta una amplitud de intervalo de 32 ms. Además, considerando el porcentaje acumulado, el 62,50 % registraron una duración promedio entre 4059 ms y 4154 ms.

La duración de solicitud promedio del framework propuesto (Tabla 5.5) es menor a la del sistema actual (Tabla 5.6). En la Figura 5.3 se observa que el 50,00 % de iteraciones que se encuentran entre el cuartil 1 y cuartil 3 presentan más dispersión en los datos registrados con el sistema actual que con el framework propuesto, siendo menores los valores de duración de la solicitud promedio (ms) de esta última.

Figura 5.3. Diagrama de cajas y bigotes de la duración de solicitud promedio (ms) de pruebas de estrés al sistema actual y al framework propuesto

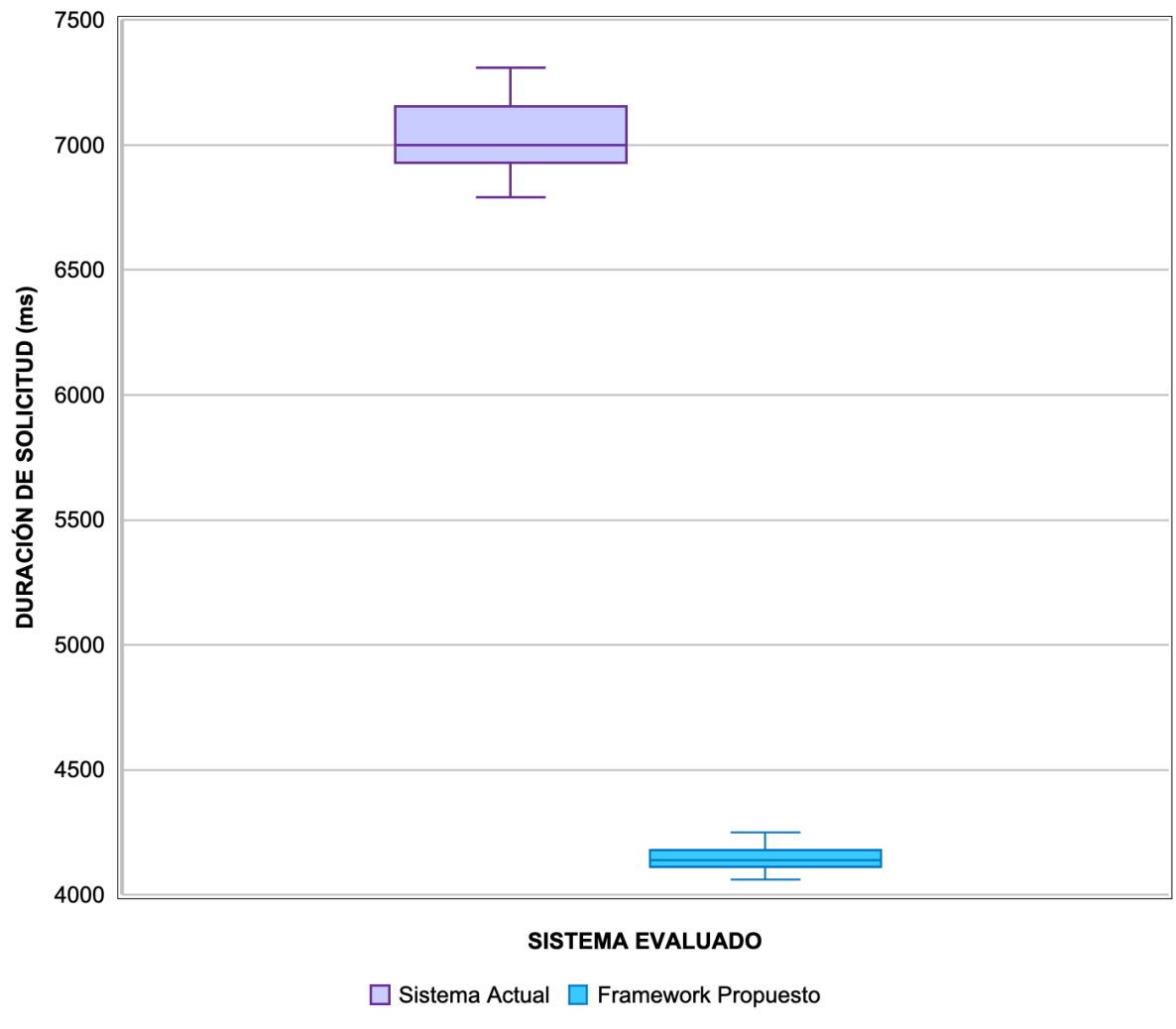


Tabla 5.6

Distribución de frecuencias de la duración de solicitud promedio (ms) de pruebas de estrés al framework propuesto

DURACIÓN DE SOLICITUD PROMEDIO (ms)	NÚMERO DE ITERACIONES (/200 VU)	NÚMERO DE ITERACIONES ACUMULADO (/200 VU)	PORCENTAJE (%)	PORCENTAJE ACUMULADO (%)
4059 - 4090	8	8	20,00	20,00
4091 - 4122	6	14	15,50	35,00
4123 - 4154	11	25	27,50	62,50
4155 - 4186	6	31	15,00	77,50
4187 - 4218	7	38	17,50	95,00
4219 - 4250	2	40	5,00	100,00
TOTAL	**40**		**100,00**	

VU: virtual user generado por K6

Prueba con Usuarios

Para las pruebas con usuarios, se construyó una interfaz sencilla que cumpla con la funcionalidad de invocar la función del backend usada en las pruebas anteriores y que adicionalmente registre en pantalla el tiempo total de la duración de la solicitud, de manera que esta sea accesible a los usuarios y pueda ser recopilada de manera rápida y sencilla.

Esta prueba se llevó a cabo con 77 estudiantes de medicina, quienes estando desde sus casas realizaron la solicitud correspondiente y luego registraron los valores de duración que se mostraban en pantalla.

En la Tabla 5.7 se observa que el 94,81 % de usuarios registraron una duración de solicitud entre 367 ms y 2474 ms. En esta distribución la amplitud de intervalo es de 1054 ms. En la Tabla 5.8 se observa que el 94,81 % de usuarios registraron una duración de solicitud entre 213 ms y 1436 ms. En esta distribución la amplitud de intervalo es de 306 ms. Estos valores son inferiores a los presentados en la Tabla 5.7.

En la Figura 5.4 se observa que el 50,00 % de iteraciones que se encuentran entre el cuartil 1 y cuartil 3 presentan más dispersión en los datos registrados con el sistema actual que con el framework propuesto, siendo menores los valores de duración de la solicitud (ms) de esta última.

Tabla 5.7

Distribución de frecuencias de la duración de solicitud (ms) de prueba con usuarios al sistema actual

DURACIÓN DE SOLICITUD (ms)	NÚMERO DE PARTICIPANTES	NÚMERO DE PARTICIPANTES ACUMULADO	PORCENTAJE (%)	PORCENTAJE ACUMULADO (%)
367 - 1420	64	64	83,12	83,12
1421 - 2474	9	73	11,69	94,81
2475 - 3528	1	74	1,30	96,11
3529 - 4582	2	76	2,60	98,70
4583 - 5636	0	76	0,00	98,70
5637 - 6690	0	76	0,00	98,70
6691 - 7744	1	77	1,30	100,00
TOTAL	**77**		**100,00**	

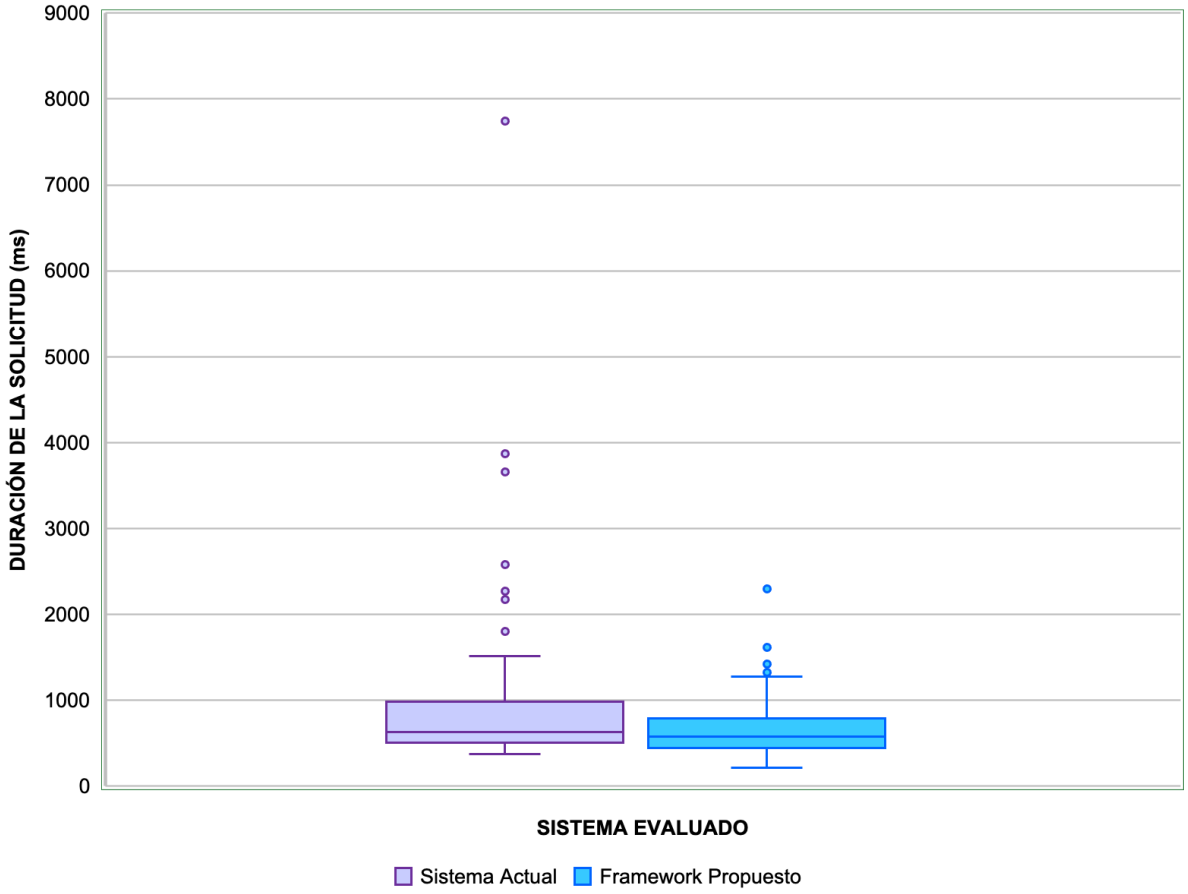
Tabla 5.8

Distribución de frecuencias de la duración de solicitud (ms) de prueba con usuarios al framework propuesto

DURACIÓN DE SOLICITUD (ms)	NÚMERO DE PARTICIPANTES	NÚMERO DE PARTICIPANTES ACUMULADO	PORCENTAJE (%)	PORCENTAJE ACUMULADO (%)
213 - 518	32	32	41,56	41,56
519 - 824	28	60	36,36	77,92
825 - 1130	5	65	6,49	84,42
1131 - 1436	8	73	10,39	94,81
1437 - 1742	2	75	2,60	97,40
1743 - 2048	0	75	0,00	97,40
2049 - 2354	2	77	2,60	100,00
TOTAL	**77**		**100,00**	

Cabe destacar que con el sistema actual se presentan mayor número de registros atípicos, observando uno extremo que corresponde a 7744 ms. En el caso del framework propuesto solo se presentaron 03 casos.

Figura 5.4. Diagrama de cajas y bigotes de la duración de solicitud (ms) de prueba de usuarios al sistema actual y al framework propuesto



Comparación de Resultados de las Pruebas

En la Tabla 5.9 se observa que los valores de los diferentes estadísticos son mayores en el sistema actual que con el framework propuesto, tanto en las pruebas de estrés como en la de usuarios. El coeficiente de asimetría en todos los casos es mayor que cero, lo cual indica que los registros realizados grafican una distribución asimétrica con presencia de valores extremos hacia la derecha.

Con la prueba de Kolmogorov-Smirnov se encontró que los datos no se ajustan a una distribución normal ($p < 0,05$), excepto en la prueba de estrés al framework propuesto ($p > 0,05$).

La prueba de contraste de hipótesis de U Mann-Whitney establece que en ambas pruebas existe diferencia significativa ($p < 0,05$) de la duración de la solicitud (ms) entre el sistema actual y el framework propuesto, encontrando en esta última tecnología menores valores.

Tabla 5.9

Análisis estadístico de las pruebas de estrés y de usuarios realizadas al sistema actual y al framework propuesto

ESTADÍSTICOS	**K6**		**USUARIOS**	
	Sistema actual	**Framework Propuesto**	**Sistema Actual**	**Framework Propuesto**
n	40	40	77	77
Me(RIC)	7000(225)	4140(70)	633(483)	577(349)
Vmín	6790	4060	369	215
Vmáx	7310	4250	7744	2354
R	520	190	7375	2139
As	0,39	0,11	4,49	2,01
Prueba de Kolmogorov-Smirnov	p = 0,014	p = 0,200	p = 0,000	p = 0,000
p-crítico	0,050		0,050	
p-valor	0,000		0,023	
Significancia U de Mann- Whitney	p < 0,05		p < 0,05	

n: número de datos
Me(RIC): Mediana (Rango Intercuartílico)
Vmín: valor mínimo
Vmáx: valor máximo
R: Rango
As: coeficiente de asimetría

6 Discusión

Con el presente trabajo desarrollado se validó la propuesta de (Gerasimou y col., 2018; Knoche & Hasselbring, 2018; Somogyi & Kovesdan, 2021), quienes indican que la tarea de refactorización de software es un proceso que puede consumir bastante tiempo dependiendo de la complejidad del caso. Pero al evaluar y obtener las funciones críticas del sistema se puede facilitar este proceso, lo que en el presente trabajo se vio reflejado al elaborar la interfaz de servicios del sistema.

Por otro lado, la metodología propuesta para el desarrollo de la refactorización del software fue efectiva en el desarrollo del proyecto y al estar alineada con el estado del arte, como con la propuesta de (Wolfart y col., 2021), se puede afirmar que es una metodología valida.

También es importante mencionar que la mantenibilidad de un sistema construido con microservicios es superior, además de presentar ventajas en la flexibilidad de estos, de acuerdo a lo indicado por (Khazaei y col., 2020; Knoche & Hasselbring, 2018). Todo esto fue indicado por los trabajadores de la Unidad de Implementación del ERP durante las entrevistas que se tuvieron; las que estuvieron alineadas con el trabajo realizado por (Di Francesco y col., 2018), y se encontró un flujo de trabajo similar al que fue desarrollado en el caso de estudio llevado a cabo por (Balalaie y col., 2018).

Finalmente, se tiene el aspecto del testing, que según lo mencionado por (Pietrantuono y col., 2020) es uno de los aspectos que muchas veces se pasa por alto. En el presente trabajo también se vio la importancia de realizar pruebas, y se aplicaron algunas de las recomendadas por (Pietrantuono y col., 2020), obteniendo los resultados esperados.

Conclusiones

PRIMERO. Se logró desarrollar un framework utilizando tecnologías y arquitecturas modernas para realizar el upgrade del sistema ERP de la Universidad Católica de Santa María, llegando a obtener una arquitectura, así como también una plantilla para que esta sea replicada.

SEGUNDO. Se analizó las tendencias actuales en el mercado, donde se determinó que ReactJS es la mejor librería para el desarrollo de las interfaces, debido a que es la más utilizada actualmente y cuenta con una gran comunidad de usuarios que contribuyen en la solución de errores y creación de plantillas y otras librerías.

TERCERO. Se definió que la arquitectura a utilizar para el desarrollo del sistema es una arquitectura basada en microservicios debido a los beneficios que ofrece en cuanto a la mantenibilidad y escalabilidad del sistema. Además, se realizó la refactorización del módulo gestión de notas de crédito y costo promedio de la botica Aliviari, y se realizó la validación correspondiente; obteniendo que su desempeño es el esperado y que su uso es sencillo.

CUARTO. El uso de un framework con una arquitectura basada en microservicios utilizando tecnologías modernas mejora la productividad de los desarrolladores. Esto se ve evidenciado al tener un árbol de archivos con una menor complejidad, además de presentar una mejor manera de organizar el código fuente del sistema.

QUINTO. El uso de Go y ReactJS bajo el framework propuesto es una mejor alternativa para el desarrollo del sistema ERP. Se determinó que el uso del framework propuesto reduce el tiempo de duración promedio de la solicitud http. En un entorno de producción se obtuvo que el 65 % de iteraciones de prueba al sistema en PHP se encuentra entre 6789 ms y 7049 ms; mientras que al sistema desarrollado en el caso de estudio presenta en el 62.50 % de iteraciones un promedio entre 4059 ms

y 4154 ms. Esto debido que Go es un lenguaje compilado, mientas que PHP es un lenguaje interpretado.

SEXTO. A lo anterior mencionado se suma los resultados de la prueba con usuarios, donde a pesar de tener una variable fuera de nuestro control como es la velocidad de internet de cada usuario, el framework propuesto presenta mejores tiempos promedio de solicitudes, estando estos entre 213 ms y 2354 ms, a comparación del sistema actual, con el que se alcanzaron valores entre 367 y 7744 ms.

SÉPTIMO. Tomando en cuenta la validación cualitativa, se concluye que el framework presentado mejora la productividad de los desarrolladores, brindándoles facilidades tanto en el momento de desarrollo como al momento de darle mantenimiento al sistema. Esto a través de una separación de los módulos del ERP dentro de sus respectivos proyectos, y teniendo una estructura del repositorio de código fuente más organizada y fácil de gestionar.

Recomendaciones

Primero: Utilizar la arquitectura basada en microservicios aplicando DevOps para el ciclo de desarrollo, implementando flujos de automatización a través de pipelines de CI/CD; de manera que se tenga una entrega más rápida y confiable del software desarrollado.

Segundo: Realizar el despliegue del sistema a través de contenedores, dependiendo del crecimiento del mismo usar orquestadores de contenedores y si es posible implementar un service mesh y balanceadores de carga para los servicios.

Tercero: Usar un entorno cloud, iniciando con los servicios y opciones del sistema que reciben un mayor flujo de usuarios. Así como también, una migración de las bases de datos a esta arquitectura cloud.

Estándares

- ISO/IEC 25000:

  Estándar conocido como SQuaRE (System and Software Quality Requirements and Evaluation), que tiene como objetivo definir un framework para la evaluación de la calidad de un producto software. Se divide en 5 sub estándares, cada uno orientado a cumplir con la calidad en diferentes momentos del ciclo de vida del desarrollo de software.

- ISO/IEC 9126:

  Estándar que define métricas internas y externas para determinar la calidad de una aplicación. Dichas métricas se determinan a partir de una serie de atributos como la funcionalidad, la confiabilidad, la usabilidad, entre otros.

- ISO/IEC 9241-11:

  La sección 11 del estándar se centra en medir la efectividad, eficiencia, y satisfacción en el contexto de qué tanto el usuario puede alcanzar los objetivos para cuales se desarrolló la aplicación, así como también el cambio en el nivel de usabilidad a lo largo del tiempo.

- ISO/IEC 12207:

  Establece un framework para los procesos del ciclo de vida del desarrollo de software, estableciendo una terminología común, así como fases, tareas, y roles que se llevan a cabo.

Glosario

- **CLI**: Siglas de "Command-line interface", es un programa que acepta texto como entrada del usuario y lo procesa para acceder o invocar funcionalidades del sistema o de una herramienta.
- **Go**: Es un lenguaje de programación concurrente y compilado, con un sistema de tipado estático. Es sintácticamente similar la familia de lenguajes C, pero busca ser tan dinámico como Python.
- **PHP**: Es un lenguaje de programación de propósito general que está orientado al desarrollo web. Fue creado en 1994 con el objetivo de dar mayor funcionalidad a las páginas web.
- **ReactJS**: Es una librería de JavaScript para la construcción de interfaces de usuario. Es actualmetne el framework preferido por los desarolladores alrededor del mundo según la encuesta de Stack Overflow.
- **T de Student**: Prueba estadísitca que sirve para determinar si existe una diferencia significativa entre las medidas de dos grupos que presentan distribución normal.
- **U de Mann-Whitney**: Prueba estadísitca que sirve para determinar si existe una diferencia significativa entre las medidas de dos grupos que presentan una distribución no paramétrica.
- **Shapiro Wilk**: Es una prueba de normalidad, en la que se determina si los datos que se tiene de una población o muestra presentan distribución normal.

Referencias

Al-Debagy, O. & Martinek, P. (2020). Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach. *SOSE 2020 - IEEE 15th International Conference of System of Systems Engineering, Proceedings*, 289-293. https://doi.org/10.1109/SOSE50414.2020.9130466

Auth0 & Peyrott, S. (s.f.). JSON Web Token Introduction - jwt.io. https://jwt.io/introduction

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A. & Lynn, T. (2018). Microservices migration patterns. *Software - Practice and Experience*, *48*(11), 2019-2042. https://doi.org/10.1002/SPE.2608

Carrasco, A., Van Bladel, B. & Demeyer, S. (2018). Migrating towards microservices: Migration and architecture smells. *IWoR 2018 - Proceedings of the 2nd International Workshop on Refactoring, co-located with ASE 2018*, 1-6. https://doi.org/10.1145/3242163.3242164

Carvalho, L., Garcia, A., Assuncao, W. K., De Mello, R. & Julia De Lima, M. (2019). Analysis of the Criteria Adopted in Industry to Extract Microservices. *Proceedings - 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice, CESSER-IP 2019*, 22-29. https://doi.org/10.1109/CESSER-IP.2019.00012

Costa, D. I. C., Filho, E. P. S., Da Silva, R. F., Thiago, T. D. & Cortés, M. I. (2020). Microservice architecture: A tertiary study. *ACM International Conference Proceeding Series*, 61-70. https://doi.org/10.1145/3425269.3425277

De Iasio, A. & Zimeo, E. (2021). A framework for microservices synchronization. *Software - Practice and Experience*, *51*(1), 25-45. https://doi.org/10.1002/SPE.2877

Di Francesco, P., Lago, P. & Malavolta, I. (2018). Migrating Towards Microservice Architectures: An Industrial Survey. *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, 29-38. https://doi.org/10.1109/ICSA.2018.00012

Engel, T., Langermeier, M., Bauer, B. & Hofmann, A. (2018). Evaluation of microservice architectures: A metric and tool-based approach. *Lecture Notes in Business Information Processing*, *317*, 74-89. https://doi.org/10.1007/978-3-319-92901-9_8

Facebook Open Source. (s.f.). React – A JavaScript library for building user interfaces. https://reactjs.org/

Gan, Y. & Delimitrou, C. (2018). The architectural implications of cloud microservices. *IEEE Computer Architecture Letters*, *17*(2), 155-158. https://doi.org/10.1109/LCA.2018.2839189

Ganney, P. S., Pisharody, S. & Claridge, E. (2020). Software engineering. *Clinical Engineering*, 131-168. https://doi.org/10.1016/B978-0-08-102694-6.00009-7

Gerasimou, S., Kechagia, M., Kolovos, D., Paige, R. & Gousios, G. (2018). On software modernisation due to library obsolescence. *Proceedings - International Conference on Software Engineering*, 6-9. https://doi.org/10.1145/3194793.3194798

Google. (s.f.). The Go Programming Language. https://go.dev/

Grafana. (2021). Open-source load testing tool for developers | k6 OSS. https://k6.io/open-source

Heinrich, R., Van Hoorn, A., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., Schulte, S. & Wettinger, J. (2017). Performance engineering for microservices: Research challenges & directions. *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, 223-226. https://doi.org/10.1145/3053600.3053653

Hernández Sampieri, R., Fernández Collado, C. & del Pilar Baptista Lucio, M. (2010). *Metodología de la investigación* (5ta Ed.). McGraw-Hill.

Hussain, S. M., Bhatti, S. N. & Rasool, M. F. U. (2017). Legacy system and ways of its evolution. *International Conference on Communication Technologies, ComTech 2017*, 56-59. https://doi.org/10.1109/COMTECH.2017.8065750

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J. & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, *35*(3), 24-35. https://doi.org/10.1109/MS.2018.2141039

Jindal, S. & Khurana, G. (2013). The statistical analysis of source-code to determine the refactoring opportunities factor (ROF) using a machine learning algorithm. *IET Conference Publications*, *2013*(645 CP), 396-403. https://doi.org/10.1049/CP.2013.2244

Jones, M., Bradley, J. & Sakimura, N. (2015). JSON Web Token (JWT). https://doi.org/10.17487/RFC7519

JSON. (s.f.). JSON. https://www.json.org/json-en.html

Kargar, M. J. & Hanifizade, A. (2018). Automation of regression test in microservice architecture. *2018 4th International Conference on Web Research, ICWR 2018*, 133-137. https://doi.org/10.1109/ICWR.2018.8387249

Kelly, A. (2018). Let's learn how module bundlers work and then write one ourselves. https://www.freecodecamp.org/news/lets-learn-how-module-bundlers-work-and-then-write-one-ourselves-b2e3fe6c88ae/

Khazaei, H., Mahmoudi, N., Barna, C. & Litoiu, M. (2020). Performance Modeling of Microservice Platforms. *IEEE Transactions on Cloud Computing*, 1-1. https://doi.org/10.1109/TCC.2020.3029092

Knoche, H. & Hasselbring, W. (2018). Using Microservices for Legacy Software Modernization. *IEEE Software*, *35*(3), 44-49. https://doi.org/10.1109/MS.2018.2141035

Miguel, F. S., Mareddy, N., Moorthy, A. K. & Liu, X. (2022). Microservices for multimedia, 89-89. https://doi.org/10.1145/3510450.3517298

Naik, A., Choudhari, J., Pawar, V. & Shitole, S. (2021). Building an EdTech Platform Using Microservices and Docker. *2021 IEEE Pune Section International Conference, PuneCon 2021*. https://doi.org/10.1109/PUNECON52575.2021.9686535

Oracle. (2021). What is ERP? https://www.oracle.com/erp/what-is-erp/

Pérez-Castillo, R., De Guzmán, I. G. R., Caballero, I. & Piattini, M. (2013). Software modernization by recovering Web services from legacy databases. *Journal of software: Evolution and Process*, *25*(5), 507-533. https://doi.org/10.1002/SMR.1554

Pietrantuono, R., Russo, S. & Guerriero, A. (2020). Testing microservice architectures for operational reliability. *Software Testing Verification and Reliability*, *30*(2). https://doi.org/10.1002/STVR.1725

Red Hat. (2017). What is an API? https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., Wei, J. & Huang, T. (2018). Migrating web applications from monolithic structure to microservices architecture. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3275219.3275230

SAP. (s.f.). Información de la empresa | Acerca de SAP SE. https://www.sap.com/latinamerica/about/company.html

Singhal, P., Kumar, A., Irshad, A. & Gumber, V. (2021). Frontend vs Backend. https://www.geeksforgeeks.org/frontend-vs-backend/

Somogyi, N. & Kovesdan, G. (2021). Software Modernization Using Machine Learning Techniques. *SAMI 2021 - IEEE 19th World Symposium on Applied Machine Intelligence and Informatics, Proceedings*, 361-365. https://doi.org/10.1109/SAMI50585.2021.9378659

Stack Overflow. (2021). Stack Overflow Developer Survey 2021. https://insights.stackoverflow.com/survey/2021#most-popular-technologies-database-prof

Wolfart, D., Assunção, W. K., Da Silva, I. F., Domingos, D. C., Schmeing, E., Villaca, G. L. & Paza, D. D. N. (2021). Modernizing legacy systems with microservices: A roadmap. *ACM International Conference Proceeding Series*, 149-159. https://doi.org/10.1145/3463274.3463334

Yin, K. & Du, Q. (2021). On Representing Resilience Requirements of Microservice Architecture Systems. *International Journal of Software Engineering and Knowledge Engineering*, *31*(6), 863-888. https://doi.org/10.1142/S0218194021500261

Anexos

A. Cuestionario propuesto

El cuestionario se aplicó a través de Google Forms.

Las preguntas cerradas definidas son:

1. El framework presentado puede mejorar la calidad del software que usted desarrolla.
2. El framework presentado facilitaría las tareas de mantenimiento de software.
3. El framework presentado le parece sencillo de aprender y utilizar.
4. Usted se adaptaría fácilmente a un flujo de trabajo utilizando el framework presentado.

Las preguntas abiertas definidas son:

1. Indique, según su opinión, ¿Qué aspectos mejoraría del framework presentado?
2. Indique su opinión sobre el framework presentado, considerando qué impacto puede tener sobre su productividad.

Las respuestas que fueron registradas se pueden ver en el siguiente enlace: Ver Respuestas.

B. Costos de mantenimiento

Cálculo de los costos de mantenimiento mensuales tomando en cuenta únicamente un desarrollador.

Elemento	**Monto (S/)**	
Sueldo promedio	1300	
Elemento	**Valor promedio(horas)**	**Monto (S/)**
Tiempo ideal	46	373.75
Tiempo real	80	650

C. Cálculo de la revalorización del sistema

Cálculo de la revalorización del sistema en función de los costos de desarrollo del proyecto.

Elemento	**Valor**
Costos totales por mes	S/830
Duración del proyecto	3 años
Total	**S/29,880**

D. Carta de validación del jefe del área

VALIDACIÓN DEL TRABAJO DE TESIS

Yo, Ing. Fernando Germán Paredes Marchena, jefe de la Unidad de Implementación del ERP de la Universidad Católica de Santa María doy conformidad del trabajo llevado a cabo por el egresado Sebastián Gonzalo Bobadilla Chara durante el desarrollo de su tesis titulada "DESARROLLO DE UN FRAMEWORK CON UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA EL UPGRADE DEL SISTEMA DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES DE LA UCSM".

Además, su propuesta se está utilizando actualmente para la implementación de Web Service para la consulta y pago de deudas de los estudiantes de la universidad con el banco BBVA y la Caja Arequipa.

Ing. Fernando Germán Paredes Marchena
DNI 29244573

E. Capturas de las pruebas realizadas

E.1. Captura del resultado de la prueba de carga realizada con la CLI de K6 al localhost con el sistema desarrollado en PHP.

```
     /\      |‾‾| /‾‾/   /‾‾/
    /\  /  \     |  |/  /   /  /
   /  \/    \    |     (   /   ‾‾\
  /          \   |  |\  \ |  (‾)  |
 / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: testGo.js
     output: -

  scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
           * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)


running (00m00.0s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  00m00.0s/10m0s  1/1 iters, 1 per VU

     ✓ Status was 200

     checks.........................: 100.00% ✓ 1         ✗ 0
     data_received..................: 57 kB   1.4 MB/s
     data_sent......................: 214 B   5.3 kB/s
     http_req_blocked...............: avg=1.5ms    min=1.5ms    med=1.5ms    max=1.5ms    p(90)=1.5ms    p(95)=1.5ms
     http_req_connecting............: avg=245µs    min=245µs    med=245µs    max=245µs    p(90)=245µs    p(95)=245µs
     http_req_duration..............: avg=37.97ms min=37.97ms med=37.97ms max=37.97ms p(90)=37.97ms p(95)=37.97ms
       { expected_response:true }...: avg=37.97ms min=37.97ms med=37.97ms max=37.97ms p(90)=37.97ms p(95)=37.97ms
     http_req_failed................: 0.00%   ✓ 0         ✗ 1
     http_req_receiving.............: avg=179µs    min=179µs    med=179µs    max=179µs    p(90)=179µs    p(95)=179µs
     http_req_sending...............: avg=158µs    min=158µs    med=158µs    max=158µs    p(90)=158µs    p(95)=158µs
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=37.63ms min=37.63ms med=37.63ms max=37.63ms p(90)=37.63ms p(95)=37.63ms
     http_reqs......................: 1       24.546503/s
     iteration_duration.............: avg=39.78ms min=39.78ms med=39.78ms max=39.78ms p(90)=39.78ms p(95)=39.78ms
     iterations.....................: 1       24.546503/s
```

E.2. Captura del resultado de la prueba de carga realizada con la CLI de K6 al localhost con el microservicio desarrollado en Go.

```
     /\      |‾|/‾/   /‾/
    /\ /  \   |  /   /  /
   /  \/    \  |\  \ |  (‾)  |
  / __________ \ |__| \__\ \_____/ .io

  execution: local
     script: script.js
     output: -

  scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
           * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)


running (00m01.0s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  00m01.0s/10m0s  1/1 iters, 1 per VU

     data_received..................: 1.9 kB 1.9 kB/s
     data_sent......................: 80 B   80 B/s
     http_req_blocked...............: avg=1.12ms min=1.12ms med=1.12ms max=1.12ms p(90)=1.12ms p(95)=1.12ms
     http_req_connecting............: avg=144µs  min=144µs  med=144µs  max=144µs  p(90)=144µs  p(95)=144µs
     http_req_duration..............: avg=391µs  min=391µs  med=391µs  max=391µs  p(90)=391µs  p(95)=391µs
       { expected_response:true }...: avg=391µs  min=391µs  med=391µs  max=391µs  p(90)=391µs  p(95)=391µs
     http_req_failed................: 0.00%  ✓ 0       ✗ 1
     http_req_receiving.............: avg=114µs  min=114µs  med=114µs  max=114µs  p(90)=114µs  p(95)=114µs
     http_req_sending...............: avg=65µs   min=65µs   med=65µs   max=65µs   p(90)=65µs   p(95)=65µs
     http_req_tls_handshaking.......: avg=0s     min=0s     med=0s     max=0s     p(90)=0s     p(95)=0s
     http_req_waiting...............: avg=212µs  min=212µs  med=212µs  max=212µs  p(90)=212µs  p(95)=212µs
     http_reqs......................: 1      0.99617/s
     iteration_duration.............: avg=1s     min=1s     med=1s     max=1s     p(90)=1s     p(95)=1s
     iterations.....................: 1      0.99617/s
     vus............................: 1      min=1      max=1
     vus_max........................: 1      min=1      max=1
```

E.3. Captura de ejemplo de la prueba de carga a PHP.

```
     /\      |‾|/‾/   /‾/
    /\ /  \   |  /   /  /
   /  \/    \  |\  \ |  (‾)  |
  / __________ \ |__| \__\ \_____/ .io

  execution: local
     script: test-post.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 40s max duration (incl. graceful stop):
           * default: 10 looping VUs for 10s (gracefulStop: 30s)


running (10.2s), 00/10 VUs, 564 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  10s

     ✓ status was 200

     checks.........................: 100.00% ✓ 564       ✗ 0
     data_received..................: 312 kB  31 kB/s
     data_sent......................: 104 kB  10 kB/s
     http_req_blocked...............: avg=287.61µs min=201µs   med=262µs    max=1.19ms   p(90)=342.7µs  p(95)=373µs
     http_req_connecting............: avg=202.46µs min=146µs   med=194µs    max=737µs    p(90)=244µs    p(95)=272µs
     http_req_duration..............: avg=178.38ms min=83.38ms med=176.23ms max=244.76ms p(90)=186.71ms p(95)=193.59ms
       { expected_response:true }...: avg=178.38ms min=83.38ms med=176.23ms max=244.76ms p(90)=186.71ms p(95)=193.59ms
     http_req_failed................: 0.00%   ✓ 0         ✗ 564
     http_req_receiving.............: avg=210.43µs min=93µs    med=204µs    max=1.08ms   p(90)=253.7µs  p(95)=283.7µs
     http_req_sending...............: avg=65.76µs  min=35µs    med=62µs     max=225µs    p(90)=91µs     p(95)=101.85µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=178.1ms  min=82.12ms med=175.97ms max=244.58ms p(90)=186.4ms  p(95)=193.32ms
     http_reqs......................: 564     55.488381/s
     iteration_duration.............: avg=178.8ms  min=84.73ms med=176.65ms max=246.22ms p(90)=187.11ms p(95)=194.1ms
     iterations.....................: 564     55.488381/s
     vus............................: 10      min=10      max=10
     vus_max........................: 10      min=10      max=10
```

E.4. Captura de ejemplo de la prueba de carga a Go.

```
  execution: local
     script: test-post.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 40s max duration (incl. graceful stop):
           * default: 10 looping VUs for 10s (gracefulStop: 30s)


running (10.0s), 00/10 VUs, 3018 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  10s

     ✓ status was 200

     checks.........................: 100.00% ✓ 3018       ✗ 0
     data_received..................: 1.0 MB  104 kB/s
     data_sent......................: 570 kB  57 kB/s
     http_req_blocked...............: avg=299.79µs min=146µs   med=283µs   max=3.01ms   p(90)=374µs   p(95)=408µs
     http_req_connecting............: avg=228.73µs min=99µs    med=213µs   max=2.71ms   p(90)=306µs   p(95)=337µs
     http_req_duration..............: avg=32.73ms  min=23.06ms med=31.63ms max=148.64ms p(90)=37.81ms p(95)=40.74ms
       { expected_response:true }...: avg=32.73ms  min=23.06ms med=31.63ms max=148.64ms p(90)=37.81ms p(95)=40.74ms
     http_req_failed................: 0.00%   ✓ 0          ✗ 3018
     http_req_receiving.............: avg=65.35µs  min=26µs    med=66µs    max=216µs    p(90)=83µs    p(95)=90.15µs
     http_req_sending...............: avg=57.88µs  min=27µs    med=55µs    max=444µs    p(90)=72µs    p(95)=79µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=32.61ms  min=22.97ms med=31.52ms max=148.5ms  p(90)=37.7ms  p(95)=40.62ms
     http_reqs......................: 3018    300.966653/s
     iteration_duration.............: avg=33.16ms  min=23.38ms med=32.07ms max=150.07ms p(90)=38.25ms p(95)=41.2ms
     iterations.....................: 3018    300.966653/s
     vus............................: 10      min=10       max=10
     vus_max........................: 10      min=10       max=10
```

E.5. Captura de ejemplo de una prueba de estrés realizada al servidor en PHP.

```
  execution: local
     script: test-post.js
     output: -

  scenarios: (100.00%) 1 scenario, 200 max VUs, 2m50s max duration (incl. graceful stop):
           * default: Up to 200 looping VUs for 2m20s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)

WARN[0021] Request Failed                                error="Post \"http://localhost:8080/index.php\": read tcp 12
7.0.0.1:62534->127.0.0.1:8080: read: connection reset by peer"
WARN[0022] Request Failed                                error="Post \"http://localhost:8080/index.php\": dial tcp 12
7.0.0.1:8080: connect: connection reset by peer"
WARN[0024] Request Failed                                error="Post \"http://localhost:8080/index.php\": dial tcp 12
7.0.0.1:8080: connect: connection reset by peer"
WARN[0025] Request Failed                                error="Post \"http://localhost:8080/index.php\": dial tcp 12
7.0.0.1:8080: connect: connection reset by peer"
WARN[0025] Request Failed                                error="Post \"http://localhost:8080/index.php\": read tcp 12
7.0.0.1:63292->127.0.0.1:8080: read: connection reset by peer"
WARN[0026] Request Failed                                error="Post \"http://localhost:8080/index.php\": dial tcp 12
7.0.0.1:8080: connect: connection reset by peer"
running (2m20.0s), 000/200 VUs, 8843 complete and 0 interrupted iterations
default ✓ [======================================] 000/200 VUs  2m20s

     ✗ status was 200
      ↳  98% — ✓ 8719 / ✗ 124

     checks.........................: 98.59% ✓ 8719      ✗ 124
     data_received..................: 4.8 MB 34 kB/s
     data_sent......................: 1.6 MB 12 kB/s
     http_req_blocked...............: avg=47.92ms  min=0s       med=248µs max=19.5s   p(90)=299µs p(95)=320µs
     http_req_connecting............: avg=47.85ms  min=0s       med=185µs max=19.5s   p(90)=226µs p(95)=241µs
     http_req_duration..............: avg=1.69s    min=0s       med=2.03s max=2.18s   p(90)=2.09s p(95)=2.11s
       { expected_response:true }...: avg=1.72s    min=15.14ms  med=2.03s max=2.18s   p(90)=2.09s p(95)=2.11s
     http_req_failed................: 1.40%  ✓ 124       ✗ 8719
     http_req_receiving.............: avg=192.49µs min=0s       med=191µs max=17.55ms p(90)=223µs p(95)=236µs
     http_req_sending...............: avg=61.25µs  min=0s       med=59µs  max=782µs   p(90)=83µs  p(95)=92µs
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s    max=0s      p(90)=0s    p(95)=0s
     http_req_waiting...............: avg=1.69s    min=0s       med=2.03s max=2.18s   p(90)=2.09s p(95)=2.11s
     http_reqs......................: 8843   63.15587/s
     iteration_duration.............: avg=2.05s    min=409.81µs med=2.03s max=25.91s  p(90)=2.09s p(95)=2.12s
     iterations.....................: 8843   63.15587/s
     vus............................: 1      min=1       max=200
     vus_max........................: 200    min=200     max=200
```

E.6. Captura de ejemplo de una prueba de estrés realizada al servidor en Go.

```
  execution: local
     script: test-post.js
     output: -

  scenarios: (100.00%) 1 scenario, 200 max VUs, 2m50s max duration (incl. graceful stop):
           * default: Up to 200 looping VUs for 2m20s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)

WARN[0071] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
WARN[0075] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
WARN[0075] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
WARN[0075] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
WARN[0075] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
WARN[0075] Request Failed                                error="Post \"http://localhost:9090/iniciar-sesion\": dial:
i/o timeout"
running (2m34.4s), 000/200 VUs, 18148 complete and 0 interrupted iterations
default ✓ [======================================] 000/200 VUs  2m20s

     ✗ status was 200
      ↳  99% — ✓ 18129 / ✗ 19

     checks.........................: 99.89% ✓ 18129      ✗ 19
     data_received..................: 6.3 MB 41 kB/s
     data_sent......................: 3.4 MB 22 kB/s
     http_req_blocked...............: avg=726.25ms min=0s      med=291µs    max=21.38s p(90)=2.46ms   p(95)=6.7s
     http_req_connecting............: avg=726.18ms min=0s      med=218µs    max=21.38s p(90)=2.42ms   p(95)=6.7s
     http_req_duration..............: avg=256.89ms min=0s      med=174.45ms max=4.43s  p(90)=448.43ms p(95)=850.33ms
       { expected_response:true }...: avg=257.15ms min=13.01ms med=174.84ms max=4.43s  p(90)=448.67ms p(95)=850.82ms
     http_req_failed................: 0.10%  ✓ 19         ✗ 18129
     http_req_receiving.............: avg=62.78µs  min=0s      med=64µs     max=251µs  p(90)=79µs     p(95)=84µs
     http_req_sending...............: avg=64.37µs  min=0s      med=57µs     max=4.49ms p(90)=74µs     p(95)=82µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s     p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=256.76ms min=0s      med=174.25ms max=4.43s  p(90)=448.3ms  p(95)=850.23ms
     http_reqs......................: 18148  117.566149/s
     iteration_duration.............: avg=1.01s    min=13.75ms med=209.99ms max=25.9s  p(90)=1.61s    p(95)=6.71s
     iterations.....................: 18148  117.566149/s
     vus............................: 1      min=1        max=200
     vus_max........................: 200    min=200      max=200
```