UNIVERSIDAD TECNOLOGICA ECOTEC

FACULTAD DE INGENIERÍAS, ARQUITECTURA Y CIENCIAS DE LA NATURALEZA

TITULO DEL TRABAJO:

EVALUACIÓN DE LENGUAJES DE PROGRAMACIÓN PARA EL DESARROLLO DE UN SISTEMA CON PROCESAMIENTO PARALELO EN UNA EMPRESA DE SOLUCIONES MEDIOAMBIENTALES

LINEA DE INVESTIGACIÓN:

TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

MODALIDAD DE TITULACIÓN:

TRABAJO DE INTEGRACIÓN CURRICULAR

CARRERA:

INGENIERÍA EN SOFTWARE

TÍTULO A OBTENER:

INGENIERO EN SOFTWARE

AUTOR:

KENNY ANDRÉS PIZARRO LUZÓN

TUTOR:

ING. ALEJANDRA COLINA VARGAS, PHD.

GUAYAQUIL, ECUADOR

2024

ANEXO No. 9

PROCESO DE TITULACIÓN
CERTIFICADO DE APROBACIÓN DEL TUTOR

Samborondón, 19 de diciembre de 2024.

Magíster
Erika del Pilar Ascencio Jordán
Unidad Académica: Facultad de Ingenierías, Arquitectura y Ciencias de la Naturaleza
Universidad Tecnológica ECOTEC

De mis consideraciones:

Por medio de la presente comunico a usted que el trabajo de titulación TITULADO: **EVALUACIÓN DE LENGUAJES DE PROGRAMACIÓN PARA EL DESARROLLO DE UN SISTEMA CON PROCESAMIENTO PARALELO EN UNA EMPRESA DE SOLUCIONES MEDIOAMBIENTALES**, fue revisado, siendo su contenido original en su totalidad, así como el cumplimiento de los requerimientos establecidos en la guía para su elaboración, por lo que se autoriza al estudiante: **Kenny Andrés Pizarro Luzón** para que proceda con la presentación oral del mismo.

ATENTAMENTE,

Firmado electrónicamente por:
ALEJANDRA MERCEDES COLINA VARGAS

Firma

Alejandra Mercedes Colina Vargas, PhD.
Tutora

ANEXO No. 10

PROCESO DE TITULACIÓN
CERTIFICADO DEL PORCENTAJE DE COINCIDENCIAS
DEL TRABAJO DE TITULACIÓN

Habiendo sido revisado el trabajo de titulación TITULADO: EVALUACIÓN DE LENGUAJES DE PROGRAMACIÓN PARA EL DESARROLLO DE UN SISTEMA CON PROCESAMIENTO PARALELO EN UNA EMPRESA DE SOLUCIONES MEDIOAMBIENTALES elaborado por KENNY ANDRÉS PIZARRO LUZÓN fue remitido al sistema de coincidencias en todo su contenido **el mismo que presentó un porcentaje del 3% mismo que cumple con** el valor aceptado para su presentación que es inferior o igual al 10% sobre el total de hojas del documento. **Adicional se adjunta print de pantalla de dicho resultado.**

ATENTAMENTE,

ALEJANDRA MERCEDES COLINA VARGAS

Firma
Ing. Alejandra Mercedes Colina Vargas, PhD.
Tutora

Dedicatoria

Dedico este trabajo con amor y gratitud a mi familia, quienes han sido mi mayor fuente de inspiración y apoyo incondicional durante este largo camino académico. A mis padres, por su confianza y esfuerzo constante, que me enseñaron el valor del trabajo duro y la perseverancia, este logro es tan suyo como mío.

Agradecimiento

Quiero expresar mi más profundo agradecimiento a mis padres, cuyo apoyo incondicional a lo largo de estos años ha sido esencial para alcanzar este logro. Su confianza, esfuerzo y amor han sido la base sólida sobre la cual he construido este objetivo. Sin su respaldo constante, este éxito no hubiera sido posible. También quiero agradecer a mis docentes y tutores, quienes me acompañaron a lo largo de mi carrera, por su orientación, dedicación y los conocimientos que me transmitieron, los cuales han sido fundamentales en la elaboración de este trabajo. A mis compañeros de carrera, César y Jorge, les agradezco las experiencias compartidas y el trabajo en equipo, que nos permitió crecer juntos como profesionales. Finalmente, quiero agradecerme por la perseverancia y el esfuerzo constante a lo largo de mi carrera. A pesar de los desafíos, siempre busqué seguir adelante y superar los obstáculos para alcanzar mis metas.

Resumen

Una empresa de soluciones medioambientales enfrenta el desafío de generar un creciente número de reportes trimestrales en formato PDF, lo que ha sobrecargado el sistema actual que lo mantiene. Este Trabajo de Integración Curricular busca encontrar una solución a este problema mediante la evaluación de diferentes lenguajes de programación para mejorar el proceso de generación de estos informes en PDF.

El presente Trabajo de Integración Curricular tiene como objetivo evaluar la eficiencia de diferentes lenguajes de programación en la mejora del proceso de generación de PDFs de reportes trimestrales mediante la implementación de procesamiento paralelo.

Para lograr una evaluación integral, se empleó una metodología mixta que combina enfoques cuantitativos y cualitativos. Se utilizó la observación empírica para medir aspectos técnicos como la velocidad de procesamiento, el uso de CPU y el margen de fallos en las pruebas de 1000, 3000 y 6000 archivos. Se realizaron entrevistas y encuestas para evaluar el nivel de satisfacción y expectativa de los usuarios, así como el potencial impacto en la productividad empresarial.

Los resultados de la evaluación realizada muestran que Golang destaca por su balance entre velocidad, y estabilidad, siendo ideal para entornos críticos. Node.js, por su parte, demostró gran flexibilidad y escalabilidad, situándose como una opción viable. Python y F# presentaron limitaciones de rendimiento bajo cargas intensivas de documentos. Las conclusiones indican que Golang y Node.js son las mejores opciones para el desarrollo de sistemas robustos y eficientes en tareas concurrentes en la nube de Google Drive, contribuyendo significativamente al avance tecnológico en los procesos internos empresariales.

Palabras clave: Procesamiento paralelo, Lenguajes de programación, Nube de Google Drive, Optimización del sistema, Avance tecnológico.

Abstract

An environmental solutions company faces the challenge of generating an increasing number of quarterly reports in PDF format, which has overloaded the current system that maintains it. This Curriculum Integration Work seeks to find a solution to this problem by evaluating different programming languages to improve the process of generating these PDF reports.

The objective of this Curriculum Integration Work is to evaluate the efficiency of different programming languages in improving the process of generating PDFs of quarterly reports by implementing parallel processing.

To achieve a comprehensive evaluation, a mixed methodology was used that combines quantitative and qualitative approaches. Empirical observation was used to measure technical aspects such as processing speed, CPU usage, and the margin of errors in tests of 1000, 3000, and 6000 files. Interviews and surveys were conducted to assess the level of user satisfaction and expectation, as well as the potential impact on business productivity.

The results of evaluation show that Golang stands out for its balance between speed and stability, being ideal for critical environments. Node.js, on the other hand, demonstrated great flexibility and scalability, positioning itself as a viable option. Python and F# showed performance limitations under intensive document loads. The conclusions indicate that Golang and Node.js are the best options for the development of robust and efficient systems in concurrent tasks in Google Drive's cloud, significantly contributing to technological advancement in internal business processes

Keywords: Parallel processing, Programming languages, Google Drive cloud, System optimization, Technological advancement.

Índice de Contenido

Índice de Figuras

Índice de Tablas

1 Introducción

1.1 Contexto histórico social del objeto de estudio

La transformación digital es un fenómeno que ha estado en desarrollo desde hace décadas, aunque no siempre fue identificado con ese nombre, su evolución está ligada a hitos históricos que han moldeado la dirección de la sociedad, como la creación del sistema binario, las revoluciones industriales, la invención de las computadoras, el surgimiento de internet y la World Wide Web (Verhoef et al., 2021).

La transformación digital no solo implica la adopción de nuevas tecnologías, sino también un cambio profundo en la forma de pensar y trabajar, ya que las organizaciones deben adoptar una mentalidad innovadora y flexible para adaptarse a los avances tecnológicos y responder eficazmente a las demandas actuales y futuras del mercado (Medina et al., 2022).

En las empresas, la transformación digital es vista como un viaje hacia la madurez tecnológica, puesto que al incorporar tecnologías emergentes y promover una cultura de innovación, las organizaciones pueden mejorar su agilidad, reducir costos y optimizar la toma de decisiones, además es clave aplicar la tecnología en negocios que ya cuenten con trayectoria donde la tecnología facilita nuevos modelos de negocio y garantiza el éxito (Medina et al., 2022).

Uno de los principales motores de la transformación digital en las empresas es la automatización, ya que disminuye la carga física y mental de los trabajadores lo que permite mejorar la productividad, calidad y eficiencia de los procesos internos, y a su vez permite a la organización lograr una ventaja competitiva clave al adaptarse rápidamente a las demandas del mercado (Prasad, 2018).

Jiménez (2020) y Voz et al., (2022) señalan que las organizaciones están invirtiendo cada vez más en herramientas digitales, como los sistemas informáticos, para adaptarse al

mercado por lo que buscan optimizar su eficiencia interna mediante sistemas que integren y coordinen los procesos de manera eficaz.

Gracias a los sistemas informáticos, las organizaciones pueden tomar decisiones más informadas y estratégicas, logrando optimizar sus operaciones y adaptarse a un entorno empresarial cada vez más dinámico, por esta razón, los sistemas informáticos se han convertido en una prioridad para organizaciones y gestores a nivel internacional y nacional (Ramírez, 2021).

El diseño de los sistemas informáticos requiere una comprensión profunda del problema a resolver y de los datos necesarios, además de una adecuada implementación a través de algoritmos y lenguajes de programación (Gómez et al., 2018). Los lenguajes de programación, tanto de bajo como de alto nivel, han evolucionado para facilitar la comunicación entre los desarrolladores y las máquinas, haciendo posible la implementación de soluciones complejas (Carla et al., 2021).

Con la evolución tecnológica, nuevos paradigmas de programación han surgido para hacer frente a problemas cada vez más complejos, tradicionalmente se utilizaba la programación secuencial donde las instrucciones se ejecutan en serie (Ledesma, 2024). Sin embargo, la programación concurrente y paralela permite resolver problemas de manera más eficiente, dividiendo tareas y ejecutándolas simultáneamente en varios procesadores (Moreno, 2022).

Numerosas aplicaciones informáticas que requieren procesar grandes volúmenes de datos están surgiendo gracias al uso de tecnologías de computación paralela y distribuida, que hasta hace unos años eran poco comunes (Díaz et al., 2021). Por tal motivo, la presente propuesta pretende analizar cómo el procesamiento en paralelo, en diferentes lenguajes de programación, influye en un proceso interno de una empresa de soluciones medioambientales en la ciudad de Guayaquil.

1.2 Antecedentes

Diversas investigaciones han abordado el impacto de la transformación digital en distintos sectores, destacando la relevancia de la automatización, los sistemas inteligentes y el uso de las tecnologías emergentes. Un estudio realizado por Zaoui y Souissi (2020), resaltan la necesidad de que las empresas definan una hoja de ruta clara para guiar sus procesos de transformación digital.

La investigación subraya el carácter estratégico de la digitalización y su influencia multidimensional en las operaciones empresariales, lo que pone de manifiesto la importancia de establecer un enfoque organizado y adaptado a las necesidades de cada compañía (Zaoui & Souissi, 2020).

Por otro lado, la investigación de Ivančić (2019), señala que la digitalización no solo busca mejorar las operaciones internas, sino también expandir el alcance hacia clientes y socios externos, afectando servicios e integrando procesos. Los hallazgos de la investigación destacan la importancia de la capacidad organizacional para el cambio, así como la excelencia operativa en la integración de servicios digitales externos con el soporte TI interno (Ivančić et al., 2019).

En el contexto de la industria 4.0, el estudio realizado por Tyagi et al., (2020) discuten cómo la automatización inteligente, combinada con tecnologías como la inteligencia artificial (IA) y el internet de las cosas (IoT), es esencial para lograr la transformación digital en las empresas. El trabajo destaca que la integración de múltiples tecnologías será fundamental para los procesos automatizados en el futuro (Tyagi et al., 2020).

En el área de manufactura inteligente, un artículo publicado por Lu et al., (2020) presentan un análisis de los estándares actuales de automatización y la integración de procesos manufactureros. El estudio revisa los avances en la automatización de sistemas que permiten la producción eficiente de productos personalizados, destacando la

importancia de estándares que faciliten la automatización completa en la manufactura (Lu et al., 2020).

En el análisis comparativo de tecnologías, Soberón y Jesús (2020) evaluaron los sistemas gestores de bases de datos MySQL y PostgreSQL en la manipulación de registros mediante procesos CRUD (operaciones básicas que se pueden realizar en una base de datos). El objetivo del estudio fue determinar cuál de estas tecnologías ofrece un mejor rendimiento en el manejo de la información. Los resultados incluyen cuadros comparativos que muestran el desempeño de ambas tecnologías en pruebas de rendimiento, destacando aspectos como el alto consumo de recursos y los tiempos de respuesta.

Finalmente, un estudio comparativo realizado por Chuquimarca y Maita (2022) analizaron dos arquitecturas y un modelo de referencia comúnmente utilizados en sistemas IoT, destacando sus características, funcionalidades, ventajas y desventajas. Proporcionaron una guía para seleccionar una arquitectura o modelo adecuado, garantizando que cuente con un respaldo significativo y un nivel de madurez suficiente para su implementación efectiva. El análisis realizado busca simplificar el desarrollo de sistemas IoT y promover su adopción en proyectos complejos.

1.3 Planteamiento del Problema

La empresa de soluciones medioambientales ubicada en la ciudad de Guayaquil es el objeto de estudio de la presente investigación. Esta empresa se dedica a la gestión optimizada de recursos críticos como el agua, los residuos y la energía a nivel internacional, además cuenta con una presencia global significativa, con varias sucursales alrededor del mundo, que desarrolla e implementa soluciones que contribuyen a la sostenibilidad ambiental y al progreso social.

La empresa de soluciones medioambientales, a través de sus diversas actividades, contribuye al desarrollo sostenible, fomentando el acceso a recursos esenciales y promoviendo prácticas sostenibles en comunidades e industrias (Veolia, nd). En el caso de

Ecuador, esta compañía ha demostrado su compromiso con la región al ofrecer soluciones personalizadas y de alta calidad a sus clientes municipales e industriales, garantizando el tratamiento eficiente de aguas y residuos en varias ciudades del país (Interagua, 2024).

Durante una entrevista al jefe del área de Transformación Digital de la Gerencia de DB&T (Digital Business and Technology), indicó que la compañía tiene su sede en la ciudad de Guayaquil, lugar donde se desarrollará el objeto de estudio. En esa sede, la compañía cuenta con varias direcciones que mantienen la operación, entre ellas destacan: Dirección General, Dirección Comercial, Dirección Administrativa Financiera, Dirección Técnica, Dirección de Operaciones, Dirección de Recursos Humanos, Dirección de Operaciones Comerciales, Dirección de Servicios Industriales y Ambientales, Gerencia DB&T y Gerencia Legal (ver figura 1).

Figura 1

Organigrama de la Empresa

El jefe del área de Transformación Digital menciona que dentro de la Gerencia de DB&T abarca varias áreas: Desarrollo y Aplicaciones, Administración de Seguridades, Control y Calidad (donde se ubica el entrevistado, Administración de Servidores y el área de Business Intelligence (ver figura 2). Por su parte, el Técnico de Desarrollo y Tecnología

explicó que la Dirección Técnica cuenta con las siguientes gerencias: Gerencia de Control de Proyectos (PMO), Gerencia de Sostenibilidad (donde se encuentra el entrevistado), Gerencia de Obras y la Gerencia de Estudios y Diseños (ver figura 3).

Además, el jefe de Transformación Digital indica que, en el área de Gerencia de Sostenibilidad se ha implementado un proceso automatizado para la generación de reportes trimestrales sobre los fondos propios de los diferentes tipos de alcantarillado. Esta automatización se realizó debido al elevado volumen de reportes que deben generarse en formato PDF. El entrevistado subrayó que, aunque existe un proceso automatizado para la generación de reportes trimestrales en formato PDF, este resulta ser lento y susceptible a errores.

Figura 2

Organigrama de la Gerencia de DB&T

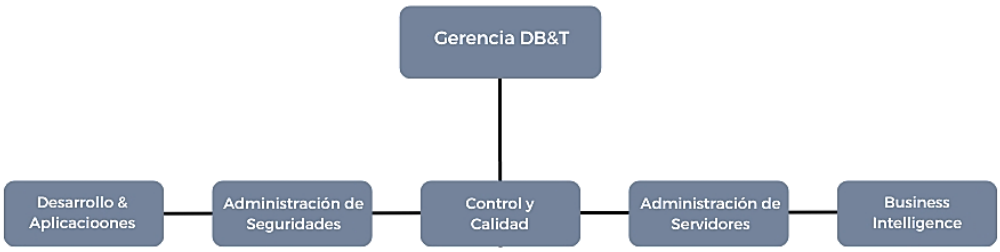
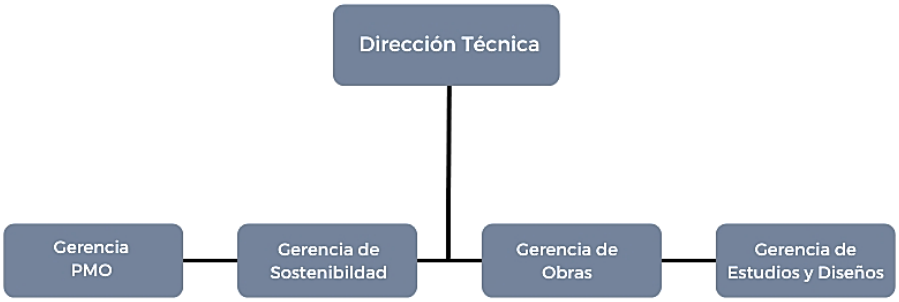


Figura 3

Organigrama de la Dirección Técnica



De acuerdo con sus observaciones, la creación masiva de archivos PDF incrementa significativamente el tiempo de procesamiento y, en ocasiones, provoca interrupciones que requieren reiniciar el proceso. Esta situación afecta negativamente la eficiencia del sistema y aumenta tanto la carga operativa como el riesgo de errores en los informes finales.

Por su parte, el Técnico de Tecnología y desarrollo señaló que, a pesar de que el proceso fue programado con el sistema Google Apps Script para automatizar la generación de reportes trimestrales, presenta limitaciones para manejar grandes volúmenes de datos. Según el experto, se requiere una solución más robusta y eficiente para llevar a cabo estas tareas de manera más rápida y precisa.

Dado el contexto expuesto, se evidencia que la Gerencia de Sostenibilidad enfrenta desafíos significativos en la generación automatizada de reportes trimestrales debido a la lentitud y propensión de errores del sistema actual. Para abordar estos problemas, se ve necesario explorar y desarrollar una solución que permita una gestión eficiente y precisa de los reportes.

1.3.1 Pregunta General

¿Cuáles son los criterios fundamentales que determinan el lenguaje de programación adecuado para mejorar el rendimiento del proceso de generación de PDFs de reportes trimestrales mediante procesamiento paralelo?

1.3.2 Preguntas Específicas

¿Cuáles son las características esenciales que debe tener un lenguaje de programación para optimizar procesos y qué técnicas de procesamiento se han aplicado?

¿Cuáles son los lenguajes de programación más adecuados para desarrollar sistemas de procesamiento paralelo?

¿Cuáles son los cambios que se deben implementar para hacer más eficiente y efectivo el proceso de generación de PDFs de reportes trimestrales?

¿Cómo diseñar el proceso en estudio utilizando los lenguajes de programación seleccionados?

¿De qué manera se puede escoger el lenguaje indicado para mejorar el proceso de generación de PDFs de reportes trimestrales?

1.4 Objetivos

1.4.1 Objetivo General

Evaluar la eficiencia de diversos lenguajes de programación en la mejora del proceso de generación de archivos PDF de reportes trimestrales a través de la implementación de procesamiento paralelo para una empresa de soluciones medioambientales.

1.4.2 Objetivos Específicos

- Investigar los fundamentos relacionados con lenguajes de programación y los tipos de procesamiento, identificando al menos tres características clave de cada uno mediante una revisión bibliográfica.
- Definir los criterios de evaluación para medir la eficiencia de los lenguajes de programación seleccionados en bases de datos científicas.
- Analizar el proceso actual de generación de reportes trimestrales en formato PDF en la empresa de soluciones medioambientales, identificando sus limitaciones y áreas de mejora.
- Desarrollar los cuatro módulos del proceso en estudio en los lenguajes de programación seleccionados, aplicando procesamiento paralelo para optimizar el tiempo y los recursos.
- Comparar los resultados obtenidos de las pruebas de rendimiento, identificando el lenguaje de programación más eficiente y su impacto en la mejora del proceso de generación de PDFs.

1.5 Justificación

Hoy en día, las nuevas tecnologías, por sí mismas, juegan un papel fundamental, facilitando la creación de modelos innovadores que permiten a las organizaciones llevar a cabo la transformación digital y evaluar sus beneficios, midiendo la competitividad y rentabilidad en función de la inversión realizada en los procesos empresariales (Álvarez Echeverría, 2015).

La implementación de sistemas informáticos se ha convertido en una ventaja competitiva para las empresas especialmente en la optimización de procesos internos, dado que, al integrar información de clientes y operaciones, no solo facilitan la toma de decisiones estratégicas, sino que también permiten identificar y aplicar mejoras en los procesos existentes. Además, ofrecen flexibilidad y escalabilidad, adaptándose a las necesidades cambiantes del mercado (Zabala et al., 2021).

La automatización avanzada de procesos mejora la satisfacción de clientes y empleados al reducir el tiempo y esfuerzo en tareas repetitivas, permitiendo que se enfoquen en actividades más significativas y valiosas. Además, los clientes reciben servicios y productos más rápidamente, también reduce costos al minimizar la intervención manual y los errores humanos (Djurica, nd).

El presente proyecto está sujeto a la importancia de identificar el lenguaje de programación más eficiente y adecuado para mejorar la automatización en una empresa de soluciones medioambientales mediante el procesamiento paralelo. Según Piovani y Krawczyk (2017), la comparación es crucial porque resolverá problemas en diversos ámbitos del conocimiento entonces un análisis comparativo basado en rendimiento, escalabilidad y facilidad de uso permitirá una evaluación sistemática de las opciones disponibles, facilitando una toma de decisiones más informada en el ámbito tecnológico.

Desde el punto de vista metodológico la implementación de soluciones paralelas se utiliza ampliamente hoy en día, especialmente en campos como la ciencia y la ingeniería,

donde es necesario llevar a cabo simulaciones de escenarios que demandan un gran volumen de cálculos (Carrasco & otros, 2022).

Martini (2020) afirma que el procesamiento paralelo es esencial para abordar problemas computacionales de alta complejidad que requieren una gran cantidad de recursos computacionales. Al dividir problemas en partes manejables, se obtendrá soluciones en tiempos razonables, lo que antes era inviable.

2 Revisión de la Literatura

2.1 Antecedentes de la Investigación

En la presente sección se muestran estudios previos relacionados con el uso de lenguajes de programación en entornos de procesamiento y tecnología avanzada. Las investigaciones destacan el comportamiento y rendimiento de distintos lenguajes y frameworks en escenarios como plataformas Serverless, procesamiento de flujos de datos en Big Data, optimización de procesos, entre otros.

Rodríguez et al., (2024) llevaron a cabo una evaluación del comportamiento de diversos lenguajes de programación en la inicialización y el arranque en frío dentro de una plataforma Serverless, tomando como caso de estudio Amazon Web Services (AWS). El estudio se centró en comparar el rendimiento de los lenguajes en operaciones CRUD (Crear, Leer, Modificar, Eliminar) para analizar su eficiencia en la inicialización de funciones Lambda.

Los resultados revelaron que Python fue el lenguaje más eficiente, con tiempos de inicialización más rápidos en todas las operaciones, seguido de cerca por NodeJS, cuyas diferencias en tiempos fueron insignificantes en comparación con Python. Por otro lado, Java demostró ser el menos eficiente, debido a la necesidad de levantar una máquina virtual y cargar el código en memoria, lo cual impactó negativamente su tiempo de arranque (Rodríguez et al., 2024).

Sin embargo, se concluyó que Java presenta un mayor margen de mejora en transiciones de estado frío a caliente, lo que podría justificar su uso en sistemas de tráfico moderado a alto. Se sugirió para futuros trabajos explorar optimizaciones en la inicialización de funciones lambda en Java y evaluar otros lenguajes como GO, C#, Ruby y PowerShell. Además, resultados preliminares indicaron que la memoria aprovisionada no afectó significativamente los tiempos de inicialización en NodeJS y Python (Rodríguez et al., 2024).

Por su parte, Fajardo (2023), llevó a cabo un estudio comparativo entre Apache Spark y Apache Flink en el contexto del procesamiento Streaming en entornos Big Data. Utilizando un diseño experimental, se desarrollaron dos aplicaciones en Python para realizar el tratamiento de flujos de datos, una implementada en Apache Spark y la otra en Apache Flink. Ambas aplicaciones abordaron el mismo problema de procesamiento continuo de datos ilimitados desde diversas fuentes.

Los resultados del estudio permitieron evaluar y contrastar el rendimiento de cada framework en términos de eficiencia en el procesamiento Streaming. La investigación concluyó que tanto Apache Spark como Apache Flink son herramientas competentes para el manejo de flujos de datos en tiempo real, siendo la elección entre ellas dependiente de las características específicas del proyecto y las necesidades particulares del entorno de Big Data en el que se implementen (Fajardo, 2023).

En cuanto a Milla (2022), realizó una investigación que comparó el rendimiento y el esfuerzo de programación entre diferentes traductores de Python (PyPy, Numba y Cython) en arquitecturas multicore, centrándose en su capacidad de optimizar y paralelizar aplicaciones CPU-bound. El estudio utilizó un diseño experimental con el caso de estudio N-Body, un problema de simulación que demanda alta capacidad computacional.

Los resultados indicaron que PyPy y Python tuvieron un desempeño limitado debido a sus restricciones en la paralelización de algoritmos. En contraste, Numba y Cython mostraron rendimientos notablemente superiores, comparables a los obtenidos con una implementación en C+OpenMP (Milla, 2022).

Sin embargo, Cython requirió un mayor esfuerzo de programación, dado que demandó conocimientos adicionales en C+OpenMP. Por otro lado, aunque Numba necesitó más líneas de código, resultó más sencillo de implementar gracias a su enfoque basado en decoradores. Se concluyó que tanto Numba como Cython son opciones viables para optimizar aplicaciones CPU-bound en Python, y la elección entre ellos debe basarse en las

necesidades específicas del equipo de desarrollo y las características de cada traductor (Milla, 2022).

Así mismo, Costanzo (2021), en su investigación comparó el rendimiento y el esfuerzo de programación entre Rust y C en arquitecturas multicore. Utilizando un diseño experimental aplicado al caso de estudio de la simulación de N cuerpos computacionales (N-Body), se enfocó en evaluar estos dos lenguajes en el contexto de la computación de alto rendimiento (HPC).

Los resultados indicaron que, aunque Rust fue diseñado para igualar la eficiencia de C, logró reducir el esfuerzo de programación sin comprometer el rendimiento, posicionándolo como una alternativa viable a C para aplicaciones HPC. Además, se concluyó que Rust no solo mantiene un rendimiento aceptable, sino que también mejora la seguridad y productividad del código, ofreciendo una opción más moderna y manejable para desarrolladores en este campo (Costanzo, 2021).

De manera complementaria Pérez et al., (2020) realizaron un análisis comparativo de lenguajes de programación para el desarrollo de aplicaciones en Ciencias de Datos. Este estudio se llevó a cabo con un diseño experimental no correlacional, utilizando una muestra de 10 bases de datos y 30 muestras específicas de la base de datos 3D_spatial_network. Entre los resultados, se destacó que R presentó un tiempo de ejecución promedio de 1.373 segundos, mientras que Python tuvo un promedio de 1.737 segundos, lo que indica que R fue 1.2 veces más rápido que Python en esta experimentación.

Sin embargo, en términos de calidad de agrupamiento, Python logró mejores resultados en 18 ocasiones frente a las 22 de R, con una pérdida de calidad de 0.209% y 1.124% respectivamente. Se concluyó que, aunque Python ofrece una mayor precisión numérica en los resultados, ambos lenguajes son competentes para manejar grandes volúmenes de datos, con la limitante de la capacidad de memoria del sistema operativo (Pérez et al., 2020).

Los estudios revisados se relacionan directamente con este trabajo de integración curricular, ya que se centran en la eficiencia de varios lenguajes de programación y tecnologías para entornos de alto rendimiento. En particular, los estudios resaltan cómo las características intrínsecas de un lenguaje, como el modelo de concurrencia, la administración de la memoria y las limitaciones del hardware, influyen en el rendimiento de las aplicaciones intensivas. Se destaca la importancia de realizar una evaluación detallada antes de elegir una tecnología, ya que la ejecución solo es un aspecto del problema; la escalabilidad, el consumo de energía, y la implementación también deben considerar la facilidad, y este enfoque permite la optimización del rendimiento en entornos industriales o de producción a gran escala, a menudo se considera esencial.

2.2 Fundamentación Teórica

2.2.1 Aplicación del procesamiento paralelo en sistemas computacionales

El procesamiento paralelo se basa en la división de tareas computacionales complejas en subtareas más pequeñas, que se ejecutan simultáneamente en múltiples procesadores, lo cual es una estrategia ampliamente implementada en sistemas multicore y multiprocesador que permite reducir de manera significativa el tiempo total de ejecución (Aziz et al., 2021).

En el ámbito del cómputo paralelo, los patrones de diseño juegan un papel clave al ofrecer soluciones recurrentes para problemas específicos, un ejemplo relevante es el patrón SPMD (Single Program Multiple Data) que permite a múltiples procesadores ejecutar el mismo programa de forma simultánea, pero aplicándolo a distintos conjunto de datos, optimizando así el rendimiento y la eficiencia (Peña et al., 2022).

Además, el procesamiento paralelo ha encontrado aplicaciones prácticas en diversos lenguajes de programación de alto nivel, como Python, a través de bibliotecas especializadas, puesto que facilita creación y ejecución de algoritmos paralelos, lo que ha

impulsado su adopción en áreas como multimedia, seguridad informática y computación de alto rendimiento (Aziz et al., 2021).

2.2.2 Procesamiento Secuencial en bases algorítmicas y aplicación en la automatización de sistemas

Desde la creación de la informática como disciplina, el procesamiento secuencial se estableció como una base y un método principal para ejecutar instrucciones en sistemas computacionales. La operación secuencial fue un enfoque popular; fue fácil de controlar en términos de flujo de ejecución y los resultados siempre se podían predecir. Este enfoque fue un facilitador vital en el desarrollo de la tecnología informática y se convirtió en una solución para automatizar operaciones manuales además de proporcionar una forma lógica de gestionar información. Tal modelo lógico de operación sacó a cientos de organizaciones de la oscuridad al automatizar diversas tareas y permitir que dichas organizaciones operen de manera más eficiente. Esto resultó en una productividad significativamente mejorada y una mejor optimización de los recursos organizacionales. Este avance marcó el inicio de la revolución de la computación dentro de organizaciones más grandes y contribuyó a las grandes empresas (Bahit, 2021).

En general, en el área de la ingeniería de software, el procesamiento secuencial se ha utilizado como un enfoque crítico de procesamiento para desarrollar aplicaciones empresariales y sistemas críticos donde la coherencia e integridad de las operaciones son críticos. Algunas de las tecnologías populares operan utilizando paradigmas secuenciales para garantizar que las tareas se realicen correctamente y que la información se manipule de manera segura. Aquí, las tareas se realizan en una secuencia definida que preserva la consistencia del proceso y, por lo tanto, evita los problemas de conflicto en los accesos a recursos. Por ejemplo, es evidente en el software financiero o los sistemas de gestión de inventario corrupto de datos. Los informes actuales sugieren que se ha producido un avance significativo en la ingeniería de software y que nuestra sociedad se basa en estos productos

profesionales a gran escala para las aplicaciones industriales, gubernamentales y comerciales (Moreira & Cruz, 2021).

Además, debido a la naturaleza del pensamiento algorítmico mencionado anteriormente, este desempeña un papel crucial en la implementación de sistemas basados en procesamiento secuencial. Esta habilidad se relaciona con ser capaz de ejecutar, evaluar, entender y formar procedimientos computacionales, y la secuencialidad prevalece como una forma lógica y estructurada. Al especializarse en la resolución de problemas utilizando algoritmos eficientes, el modelo implica que cada paso en el proceso debe basarse en el resultado de lo anterior, lo que garantiza soluciones precisas y correctas. Aún más, investigaciones científicas confirman que el pensamiento algorítmico es la capacidad de abstraer procesos, modelar problemas, realizar deducciones lógicas y sintetizar soluciones, lo que finalmente se convertiría en escribir algoritmos secuenciales para lograr los objetivos establecidos (Ramírez, 2020).

Con la aparición de la tecnología y la explosión de la cantidad de datos, la normalidad del procesamiento secuencial pronto se reveló insuficiente. Mientras que las instalaciones y la capacidad de respuesta de las aplicaciones desarrolladas hoy en día imponen requisitos más altos, el enfoque se vuelve impracticable en condiciones de funcionamiento de alta carga y alta y variabilidad. Si bien estas restricciones han animado al estudio y utilización de paradigmas de seguimiento, como el procesamiento paralelo y distribuido, los cuales trabajan dividiendo un trabajo en varias secciones y ejecutándolos al mismo tiempo. De hecho, este desarrollo ha ido de la mano de la ciencia de datos, la inteligencia artificial, las simulaciones y otros campos que procesan una cantidad abrumadora de información. En consecuencia, los lenguajes se han trasladado bajo la influencia de la próxima cadena, se han apropiado de la acumulación de estructuras y bibliotecas de procedimiento, lo que permite una implementación más fácil del procesamiento paralelo y la reducción de los tiempos (Carriel & Galarza, 2022).

2.2.3 Los lenguajes de programación

Los lenguajes de programación actúan como un puente entre la lógica humana y las instrucciones que entiende la máquina (Olarte, 2018). Se clasifican en bajo nivel y alto nivel, los lenguajes de bajo nivel están estrechamente vinculados al hardware de una computadora específica, lo que limita su portabilidad (Carla et al., 2021).

Los lenguajes de programación de bajo nivel están diseñados a medida para aprovechar al máximo las características del hardware, por lo tanto, se consideran más cercanos al lenguaje de máquina (Carla et al., 2021). Entre los lenguajes de programación de bajo nivel se encuentran: los códigos binarios, el lenguaje ensamblador y el lenguaje de máquina (Coro, 2022).

En cuanto a los lenguajes de programación de alto nivel, han evolucionado la programación al acercar el lenguaje de las máquinas al lenguaje humano, ya que al utilizar palabras y estructuras gramaticales familiares, permiten a los programadores crear software de manera más intuitiva y eficiente, convirtiéndolos en la herramienta preferida para el desarrollo de software (Carla et al., 2021).

2.2.4 Integración de servicios de la computación en la nube

La computación en la nube se define como un entorno distribuido y virtualizado que agrupa diversos recursos informáticos, presentándolos al usuario como un conjunto unificado. Esto permite a los usuarios acceder a la capacidad computacional necesaria sin preocuparse por la gestión de la infraestructura subyacente. Los términos de servicio, incluyendo rendimiento y capacidad, son negociados entre el proveedor y el consumidor (Bermúdez, 2020).

La creciente adopción de los servicios en la nube se debe en gran medida a su facilidad de uso y la flexibilidad que ofrecen. Al permitir a los usuarios ejecutar sus aplicaciones directamente en la nube, se elimina la necesidad de realizar inversiones significativas en Hardware y Software (Hosseini & Sahragard, 2019).

Grandes proveedores de servicios en la nube, como Amazon Web Services, Google Cloud Platform y Microsoft Azure, han desempeñado un papel fundamental en la popularización de la computación en la nube, puesto que ofrecen un ecosistema completo de servicios como: Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS), Software como Servicio(SaaS), entre otros (Bermúdez, 2020).

Google Apps Script es un claro representante de un servicio PaaS, que facilita el desarrollo ágil de aplicaciones mediante una integración sencilla con herramientas como Gmail, Google Docs y Google Sheets. Esta herramienta utiliza el lenguaje JavaScript y cuenta con bibliotecas integradas que optimizan el uso dentro del el ecosistema de aplicaciones de Google (Petrović et al., 2020).

En el caso de Amazon Web Services, se destaca en el mercado de la computación en la nube, ofrece una plataforma de Función como Servicio (FaaS) conocida como AWS Lambda, que permite a los usuarios implementar código basado en eventos. AWS Lambda actualmente es compatible con varios lenguajes, incluyendo C# (.Net Core), Java, Node.js (JavaScript), GO y Python (Hosseini & Sahragard, 2019).

La convergencia de tecnologías en la nube, el internet de las cosas y la cadena de bloques ha creado un ecosistema digital altamente interconectado, lo cual ha dado lugar a nuevas oportunidades para generar valor a partir de los datos, automatizar procesos y desarrollar soluciones innovadoras (Herrera et al., 2021).

Este enfoque en la integración de servicios de computación en la nube proporciona a las empresas una forma eficiente y escalable de manejar tareas computacionales intensivas, optimizando tanto los costos como el rendimiento de los sistemas en la nube, evitando grandes inversiones, ya que las tecnologías en la nube suponen un ahorro importante, además de darle flexibilidad y competitividad (Bermúdez, 2020).

2.2.5 Criterios de evaluación de velocidad de los lenguajes de programación

La velocidad es una consideración crítica en lo que respecta a la evaluación de los lenguajes de programación en el desarrollo de software, ya que afectará directamente al rendimiento y la eficiencia de las aplicaciones. Hasta recientemente, la velocidad se evaluaba típicamente en términos de métricas de tiempo de ejecución y uso de la memoria. Sin embargo, estudios recientes como (Martínez et al., 2020) han demostrado que hay mucho más en juego en términos de velocidad.

La selección y la evaluación de la velocidad de un lenguaje de programación se consideran en términos de la eficiencia de cronometraje y hasta qué punto su uso facilitará el desarrollo de las habilidades claves de los programadores. En última instancia, la velocidad se vuelve a evaluar como un criterio, junto con la legibilidad del código, impulsando las comunidades y las bibliotecas disponibles, y la funcionalidad basada en criterios prácticos. Como resultado, la evaluación de la velocidad es una consideración más amplia, orientada no solo al software, sino también a los desarrolladores.

Verificar la eficiencia requiere un análisis más detallado que comparar los lenguajes basado en la simplicidad y el encriptamiento. La eficiencia real de la velocidad de ejecución es significativamente influenciada por el compilador o el intérprete. Como señala el estudio, C y C++ a menudo generan código altamente optimizado, lo que los hace increíblemente rápidos en los casos de apuros. Por otro lado, los lenguajes interpretados, como Python, son mucho más lentos en términos de velocidad de ejecución.

Dicho esto, el software basado en Python es conocido por la facilidad de uso y la velocidad de expansión rápida de nuevas funciones. Estos hechos claramente indican que la elección del lenguaje debe depender de la situación en términos de velocidad, desarrollo rápido y facilidad (Tymoschuk, 2019).

Para determinar el lenguaje de programación más rápido, hay que tomar en cuenta más que la sintaxis y la estructura del lenguaje. De acuerdo con varios análisis, los lenguajes modernos, como Rust, presentan una alta eficiencia y seguridad que compiten con los tipos clásicos static C++. Sin embargo, la velocidad también depende del tipo de aplicación y de la plataforma. Como ejemplo, Go y Java logran utilizar los recursos del sistema de manera eficiente en las aplicaciones modernas. Tales pensamientos admiten la necesidad de adaptar el lenguaje de programación al entorno dedicado (J. Ruiz, 2024).

2.2.6 Criterios de evaluación de lenguajes de programación para el uso eficiente de la CPU en sistemas de procesamiento paralelo

La eficiencia computacional es otro criterio importante para tener en cuenta para maximizar el uso de CPU. Se usa para medir qué tan bien un lenguaje de programación gasta los recursos de hardware, como núcleos de procesamiento. En tareas paralelas, es importante que el lenguaje pueda procesar múltiples operaciones simultáneamente para reducir el tiempo del proceso. Además, es necesario que el código generado se ajuste al hardware lo más posible; por ejemplo, en los lenguajes se permiten un control granular de la CPU y la memoria, lo que los hace optar por estas aplicaciones de alta computación. La eficiencia no se mide solo en la rapidez con la que la aplicación procesa los datos sino en la energía utilizada y la capacidad del sistema para evitar cuellos de botella. En tareas paralelas, varios procesos compiten por recursos, y el lenguaje no debería crear conflictos adicionales y, en cambio, usar la distribución de carga o el dumping para dividirlos de manera justa entre núcleos (Izquierdo, 2020).

La escalabilidad evalúa el rendimiento del sistema bajo una creciente carga de trabajo o el aumento del número de recursos. Para sistemas paralelos, la escalabilidad es crítica para garantizar un aumento proporcional del rendimiento agregando núcleos de CPU o agregando máquinas al clúster distribuido. Los lenguajes de programación que admiten los modelos de programación paralela, como OpenMP para la programación de CPU, suelen

implementarse con éxito en este tipo de sistemas. Además, un sistema escalable debe poder manejar problemas de sincronización de procesos y multidisciplinares eficazmente. Esto significa eliminar carreras, reducir tiempos de bloqueo y mejorar las comunicaciones. Lenguajes como Python, fácil de escribir, elevarían problemas de escalabilidad debido a GIL, en el entorno de CPU. Java sería más preferible por sus modelo de concurrencia definido (Costanzo, 2024).

2.2.7 Criterios de evaluación de lenguajes de programación en sistemas paralelos con enfoque en manejo de errores y tolerancia a fallos

En un sistema paralelo, un fallo puede tener diferentes razones, como problemas de hardware, errores en el código, condiciones de carrera o fallos de sincronización entre procesos. La gestión de errores en tales sistemas es la capacidad del lenguaje de programación de identificar, controlar, y reportar los errores sin amenazar el funcionamiento del sistema en su totalidad. Algunos lenguajes logran esto por medio de excepciones y de mecanismos de control que permiten a cada aplicación o a un programador sobresalir un error y tomar medidas de corrección. Aun así, es más difícil identificar fallos y errores en un entorno paralelo debido al hecho de que las tareas concurrentes pueden encubrir el lugar de origen del error. Un criterio principal será la habilidad de los lenguajes para prevenir la consistencia en los datos. La gestión de los errores en los datos puede ser crítica, especialmente al lidiar con un plazo importante o con un alto nivel de datos, o con cálculos científicos de alto nivel de dificultad, como lo podemos ver en una operación ambiental (Soto, 2024).

La monitorización y el diagnóstico de errores son aspectos críticos para predecir y corregir problemas en sistemas paralelos. Muchas veces, los errores no son obviables, razón por la cual es difícil determinar la causa y el impacto. Los lenguajes de programación que incluyen bibliotecas y asistentes técnicos, como Python y Java, pueden monitorear explícitamente el procedimiento y la memoria del proceso en ejecución, lo que permite

detectar errores antes de que se produzca daños. Lenguajes que incluyen herramientas de diagnóstico en tiempo real, como los perfiles, los registros de fallos y las trazas de procedimiento, son críticos en sistemas paralelos. Eso ayuda al desarrollador a detectar los errores y, a menudo, realizar modificaciones en el código en implementación del sistema para prevenir la autenticación de los errores la próxima ves (Sarmiento & Zhizhpon, 2022).

2.2.8 Criterios de evaluación de lenguajes de programación en sistemas paralelos basados en el nivel de satisfacción del usuario

La satisfacción del usuario en estos sistemas también está relacionada con otros aspectos técnicos cuando se trata de sistemas de procesamiento paralelo. Uno de los más significativos es el rendimiento y la fiabilidad de los enfoques de sistemas. El hecho es que el paralelismo supone que un sistema puede realizar varias tareas simultáneamente y, por lo tanto, se espera que complete incluso las tareas más complejas de manera rápida y sin problemas. Por lo tanto, la percepción de qué tan rápido puede funcionar el sistema con grandes cantidades de datos o cuánto tiempo puede tomar al sistema completar las operaciones de cálculo esencial también afecta la satisfacción. En estos sistemas, la selección y el desarrollo específicos del lenguaje de programación también afectarán la percepción del usuario final. En consecuencia, los sistemas que contribuyen a la optimización del uso de recursos a través de la implementación de paralelismo y distribución de tareas avanzados mejorarán la satisfacción del usuario. Por lo tanto, el criterio de evaluación está relacionado con decisiones técnicas en el desarrollo del sistema, en particular sobre qué lenguaje de programación y herramientas de procesamiento paralelo utilizar, han afectado la percepción de un usuario sobre un sistema en términos de su eficiencia y accesibilidad (Checasaca et al., 2022).

La encuesta de satisfacción del usuario es una de las metodologías ampliamente utilizadas que cada usuario ha experimentado o escuchado. Los encuestadores emplean una amplia variedad de preguntas a fin de obtener la evaluación más completa de todos los

aspectos de la interacción de los usuarios con el sistema, incluida la simplicidad de uso, la velocidad y la estabilidad, la comprensión del sistema, etc. En caso de la utilización de sistemas que involucran la computación paralela, también es necesario evaluar la capacidad de los encuestados para evaluar la eficiencia del sistema en las tareas paralelizadas y su capacidad para trabajar con grandes volúmenes de datos correspondientes. Aparte de esa pregunta directa, también es recomendable abordar la satisfacción de los usuarios con al menos dos componentes: la interfaz del sistema como tal y la frecuencia de errores. En cuanto a los sistemas paralelos, los usuarios pueden enfrentarse a errores inevitables debido a la complejidad de las operaciones paralelas. Se puede obtener mucha información valiosa relativa a cómo los usuarios ven la fiabilidad del sistema (Huachaca, 2023).

2.2.9 Criterios de evaluación de lenguajes de programación en sistemas paralelos basados en el nivel de expectativa del usuario

En los sistemas de procesamiento paralelo, lo que el usuario espera es crítico ya que, como se ha mencionado, estos sistemas generalmente se utilizan para tareas que requieren un alto rendimiento y la eficiencia, como el análisis de grandes volúmenes de datos o la ejecución de algoritmos complicados. Los usuarios en un entorno comercial o industrial, por ejemplo, esperarían poder realizar tareas en el sistema sin ningún “cuello de botella” y sin interrupciones. Por lo tanto, simplemente pasar por alto lo que el usuario espera al diseñar un sistema de procesamiento paralelo no será tolerado y puede llevar al fracaso del sistema. Al hacerlo, dicho conocimiento puede obtenerse durante el desarrollo de un sistema en sus primeras etapas y los desarrolladores pueden decidir qué aspectos son más críticos según la percepción y el conocimiento de los usuarios. Las expectativas de los usuarios son también su comparación de su experiencia previa con sistemas similares. Cuanto más el sistema las satisface o las excede, es más probable que acepten el sistema, y viceversa (Villadoma & Melendez, 2024).

Entre las encuestas que son esenciales para medir la expectativa del usuario, se incluyen las que se realizan antes y durante el desarrollo del sistema. Estas encuestas son de las diversas preguntas sobre lo común antes mencionado, como el rendimiento esperado de la solución, las preocupaciones sobre su confiabilidad, y lo fácil que es de usar. Para tener una visión precisa de las expectativas del usuario, estas encuestas deben estar diseñadas de manera que aborden tanto las explícitas y las implícitas. Las preguntas estructurantes de múltiple opción, escalas de Likert que van desde “muy de acuerdo” a “totalmente en desacuerdo” y preguntas abiertas permiten a los usuarios proporcionar más información sobre sus necesidades. Adicionalmente, estas encuestas pueden tener las secciones donde los usuarios clasifican la importancia de varios factores, como la velocidad de respuesta versus la facilidad de uso. La retroalimentación de las encuestas tiene una influencia crítica en qué decisiones tomar acerca de los objetivos de productividad (Narváez, 2021).

2.2.10 Criterios de evaluación del impacto en la productividad empresarial de sistemas informáticos con procesamiento paralelo

En las ventajas de la evaluación empresarial que las empresas buscan medir el efecto de la eficiencia en la producción. En sistemas de procesamiento paralelo, los sistemas prometen reducir los tiempos de ejecución desde análisis de datos en tiempo real hasta la ejecución de procesos complejos que utilizan una gran cantidad de recursos computacionales. En la mayoría de los casos, las empresas esperan ver incrementos, en la producción si los costos operativos se reducen o si la empresa toma mejores decisiones basadas en los datos que se procesan más rápido y con una mayor precisión. La rapidez de procesamiento en sistemas de procesamiento paralelo es uno de los elementos que se destacan según el efecto (Sánchez & Vega, 2020).

Una vez que se haya completado la recopilación de información utilizando encuestas, el análisis de los resultados permitirá determinar las áreas de mejora y confirmar si el

sistema de procesamiento paralelo es clave para aumentar la eficiencia empresarial. Los datos recopilados aportarán información a los desarrolladores sobre cómo el sistema cumple con los objetivos iniciales y si se requieren ajustes. Por ejemplo, si las encuestas revelan que los usuarios no ven una velocidad de procesamiento efectivamente superior, la implementación del sistema debe mejorarse para hacer con que la velocidad de procesamiento sea más alta (Astera, 2023).

2.3 Marco Legal

La evaluación de lenguajes de programación para el desarrollo de un sistema con procesamiento paralelo requiere de un análisis exhaustivo del marco legal vigente, ya que es primordial que este estudio comparativo cumpla con todas las normativas legales que no impidan que se realice la investigación.

2.3.1 Constitución de la República del Ecuador: Ciencia, tecnología, innovación y saberes ancestrales

En el artículo 385 de la Constitución de la República del Ecuador tiene como objetivo la integración de conocimientos y tecnologías en favor del desarrollo sostenible y del bienestar social, el cual se ajusta con la cultura de la empresa de soluciones medioambientales. El artículo 385 expresa que: El sistema nacional de ciencia, tecnología, innovación y saberes ancestrales, en el marco del respeto al ambiente, la naturaleza, la vida, las culturas y la soberanía, tendrá como finalidad:

1. Generar, adaptar y difundir conocimientos científicos y tecnológicos.
2. Desarrollar tecnologías e innovaciones que impulsen la producción nacional, eleven la eficiencia y productividad, mejoren la calidad de vida y contribuyan a la realización del buen vivir (Asamblea Constituyente Del Ecuador, 2008, p. 186).

El artículo 385 de la Constitución es clave para el tema de investigación al resaltar la importancia de la ciencia, tecnología e innovación para el desarrollo, especialmente en

un contexto de respeto al ambiente. Este artículo apoya la creación de soluciones tecnológicas para mejorar la productividad y calidad de vida, aspectos fundamentales en empresas medioambientales.

El artículo 386 de la Constitución de la República del Ecuador, establece que se realicen actividades relacionadas con investigación, desarrollo tecnológico, innovación y saberes ancestrales: El sistema comprenderá programas, políticas, recursos, acciones, e incorporará a instituciones del Estado, universidades y escuelas politécnicas, institutos de investigación públicos y particulares, empresas públicas y privadas, organismos no gubernamentales y personas naturales o jurídicas, en tanto realizan actividades de investigación, desarrollo tecnológico, innovación y aquellas ligadas a los saberes ancestrales. El Estado, a través del organismo competente, coordinará el sistema, establecerá los objetivos y políticas, de conformidad con el Plan Nacional de Desarrollo, con la participación de los actores que lo conforman (Asamblea Constituyente Del Ecuador, 2008, p. 186).

El artículo 386 de la Constitución dispone la creación de un sistema que integra instituciones públicas y privadas, universidades, empresas y otros actores claves en la investigación, desarrollo tecnológico e investigación. Esto es relevante porque muestra el rol del Estado en fomentar un entorno para el avance tecnológico, alineado a mi investigación sobre el uso de tecnologías avanzadas.

Así mismo el artículo 388, de la Constitución de la República del Ecuador, indica cómo se gestionarán los recursos para la investigación tecnológica: El Estado destinará los recursos necesarios para la investigación científica, el desarrollo tecnológico, la innovación, la formación científica, la recuperación y desarrollo de saberes ancestrales y la difusión del conocimiento. Un porcentaje de estos recursos se destinará a financiar proyectos mediante fondos concursables. Las organizaciones que reciban fondos públicos estarán sujetas a la

rendición de cuentas y al control estatal respectivo (Asamblea Constituyente Del Ecuador, 2008, pp. 186-187).

El artículo 388 es importante para el presente tema de investigación porque asegura que el Estado destinará recursos para la investigación científica, el desarrollo tecnológico y la innovación, lo cual es fundamental para impulsar proyectos tecnológicos como el que planteo en mi investigación.

2.3.2 Ley orgánica de protección de datos personales

La Ley Orgánica de Protección de Datos Personales tiene como objetivo principal garantizar el derecho a la protección de los datos personales de los individuos en Ecuador, según el artículo 1: El objeto y finalidad de la presente ley es garantizar el ejercicio del derecho a la protección de datos personales, que incluye el acceso y decisión sobre información y datos de este carácter, así como su correspondiente protección. Para dicho efecto regula, prevé y desarrolla principios, derechos, obligaciones y mecanismos de tutela (Secretaría General, 2021, p. 5).

El artículo 1 de la Ley Orgánica de protección de datos personales es esencial para el presente trabajo integrador curricular, ya que establece la base legal para la protección de los datos personales, lo que es fundamental en el desarrollo de cualquier sistema tecnológico. Dado que la presente investigación involucra el manejo de datos en la nube, esta ley garantiza que el tratamiento de esos datos cumpla con los principios y derechos necesarios.

El artículo 8 de la misma ley establece el consentimiento de los datos personales: Se podrán tratar y comunicar datos personales cuando se cuente con la manifestación de la voluntad del titular para hacerlo. El consentimiento será válido, cuando la manifestación de la voluntad sea:

1) Libre, es decir, cuando se encuentre exenta de vicios del consentimiento;

2) Específica, en cuanto a la determinación concreta de los medios y fines del tratamiento;

3) Informada, de modo que cumpla con el principio de transparencia y efectivice el derecho a la transparencia,

4) Inequívoca, de manera que no presente dudas sobre el alcance de la autorización otorgada por el titular (Secretaría General, 2021, p. 8).

El artículo 8 de la Ley Orgánica de Protección de datos personales es esencial, debido a que regula el consentimiento necesario para el tratamiento de datos personales. Al desarrollar un sistema que manejará datos en la nube, es fundamental garantizar que los usuarios otorguen un consentimiento libre, específico, informado e inequívoco, conforme a los principios de transparencia.

2.3.3 Política para la transformación digital del Ecuador 2022 – 2025

El artículo 2 de la Política para la transformación digital del Ecuador establece las directrices para fomentar el proceso de transformación digital en el Ecuador: El objetivo de la Política es establecer los lineamientos para fomentar la Transformación Digital del Ecuador, considerando la investigación, desarrollo e innovación sobre infraestructuras y capacidades digitales, así como la digitalización de las empresas y servicios públicos, fomentando el uso de tecnologías emergentes, gestión de datos, seguridad de la información e interoperabilidad hacia todos los sectores sociales del país, considerando el desarrollo de un entorno normativo, regulatorio e institucional (DE LA SOCIEDAD, 2022, p. 68).

El artículo 2 de la Política para la transformación digital del Ecuador es relevante porque establece los lineamientos para fomentar la digitalización de las empresas y servicios públicos, promoviendo el uso de tecnologías emergentes y la gestión de datos. Es fundamental en el desarrollo de sistemas que incorporan procesamiento paralelo ya que asegura que estén alineadas con principios de seguridad de la información y regulación.

2.4 Marco Conceptual

2.4.1 API (Application Programming Interfaces)

Las API son interfaces de programación que facilitan la comunicación entre sistemas de software, permiten la creación de servicios web (REST o SOAP) escalables y fáciles de mantener, puesto que operan sobre el protocolo HTTP. Además, simplifican la integración de diferentes sistemas, permitiendo que los desarrolladores se concentren en la lógica de negocio sin preocuparse de los detalles de la comunicación entre aplicaciones (Soni & Ranga, 2019).

2.4.2 Automatización de procesos

La automatización de un proceso implica la integración de elementos y dispositivos tecnológicos que garanticen su control y funcionamiento eficiente. Un sistema automatizado debe ser capaz de responder tanto a situaciones planificadas como a imprevistos, con el fin de optimizar el proceso y posicionar a los recursos humanos en la mejor situación posible (García, 2020).

2.4.3 Concurrencia de un sistema

La concurrencia se refiere a la capacidad de un sistema para manejar múltiples programas que parecen estar haciendo procesos simultáneamente en una unidad de tiempo. Antes de la proliferación de procesadores multinúcleo, los sistemas operativos permitían la ejecución concurrente de múltiples tareas en un solo núcleo mediante la partición del tiempo (Moreno, 2022).

2.4.4 Formato de archivo JSON

JSON (JavaScript Object Notation) es un formato flexible y ampliamente utilizado para intercambiar datos entre sistemas diferentes. Su estructura se asemeja a una lista de elementos, donde cada uno está compuesto por una clave y un valor, aunque su origen está en el ecosistema JavaScript, JSON es compatible con una gran variedad de lenguajes

de programación, lo que facilita su adopción en distintos entornos de desarrollo (Brucker, 2022).

2.4.5 Google apps script

Es una plataforma de desarrollo en la nube basada en JavaScript que te permite crear aplicaciones personalizadas para G suite, ya que ofrece una forma sencilla de integrar y automatizar tareas en servicios como Gmail, Google Docs y Google Sheets, es como tener una API para interactuar con los servicios de Google y crear soluciones a medida (Petrović et al., 2020).

2.4.6 Metodología Kanban de desarrollo de software

Kanban es un método Agile que enfatiza la entrega continua y el flujo constante de trabajo. Kanban no impone rígidos ciclos de tiempo o roles específicos, en su lugar, se centra en visualizar el trabajo en progreso, identificar cuellos de botella y mejorar el proceso de manera incremental (Alaidaros et al., 2021).

2.4.7 Lenguajes de programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que permiten traducir instrucciones legibles por humanos en códigos que las máquinas puedan entender y ejecutar. Su función principal es servir de puente entre el pseudocódigo, que es comprensible para los programadores, y las instrucciones específicas que una máquina necesita para realizar una acción (Olarte, 2018).

2.4.8 Lenguajes de programación de alto nivel

Según Arenaza & Erickson (2019) y Pereira & Rosario (Pereyra & Rosario, 2021), los lenguajes de programación de alto nivel están diseñados para facilitar la interacción entre humanos y computadoras al utilizar palabras y estructuras que se asemejan al lenguaje natural, generalmente en inglés, por lo tanto, sus características hacen que la programación sea más accesible y comprensible para los desarrolladores.

2.4.9 Metodología Ágil de desarrollo de software

La metodología ágil es una filosofía de desarrollo de software que prioriza la flexibilidad y la colaboración, dado que, en lugar de entregar un producto completo al final del proyecto, ágil se basa en entregas incrementales y frecuentes, lo que permite adaptarse a los cambios y garantizar que el producto satisfaga las necesidades del cliente en todo momento (Gheorghe et al., 2020).

2.4.10 Procesamiento Paralelo

El procesamiento paralelo es una técnica que maximiza el rendimiento de los sistemas informáticos al permitir que múltiples tareas se ejecuten de forma concurrente, puesto que al dividir un problema en subproblemas más pequeños y asignarlos a diferentes procesadores, se logra una aceleración considerable en la resolución de problemas complejos y una mejor utilización de los recursos computacionales (Aziz et al., 2021).

2.4.11 Servicios web

Los servicios web son un conjunto de protocolos y estándares abiertos que facilitan la interacción entre un cliente y un servidor, permitiendo la interoperabilidad entre distintas aplicaciones. Existen dos enfoques principales para el desarrollo de los servicios web: REST (Representational State Transfer) que sigue un estilo arquitectónico, y SOAP (Simple Object Access Protocol), un protocolo más forma (Soni & Ranga, 2019).

2.4.12 Transformación digital

La transformación digital es un proceso de adaptación y actualización de las tecnologías digitales, las cuales forman parte de la vida cotidiana de las personas. Impulsa a las organizaciones y empresas a adoptar procedimientos innovadores, con el fin de ofrecer productos o servicios que respondan a las crecientes demandas y expectativas de sus clientes (Medina et al., 2022).

3 Marco Metodológico

En este capítulo se detallan los métodos, enfoques, técnicas de recolección y análisis de datos, así como los elementos del desarrollo y pruebas de rendimiento de los lenguajes de programación Golang, Python, F# y Node.js, seleccionados con base en criterios específicos. Estos incluyen su popularidad y uso industrial, capacidad para procesamiento paralelo, compatibilidad con herramientas y bibliotecas necesarias para la generación de PDFs, flexibilidad y curva de aprendizaje, además de su rendimiento en aplicaciones similares según estudios previos. La investigación se enfoca en evaluar su eficiencia en un contexto de alta demanda, representado por una empresa de soluciones medioambientales que requiere generar reportes en PDF de manera masiva y optimizada, asegurando que los lenguajes seleccionados respondan a las necesidades de escalabilidad y desempeño del sistema propuesto.

3.1 Enfoque de la investigación

En este estudio, se adopta un enfoque mixto, combinando métodos cualitativos y cuantitativos, para evaluar de manera integral la eficiencia de diferentes lenguajes de programación en el desarrollo de un sistema con procesamiento paralelo. La elección de este enfoque se justifica por la complejidad inherente al procesamiento paralelo en TI, que requiere no solo mediciones cuantitativas de rendimiento, sino también una comprensión cualitativa del proceso de generación de reportes trimestrales en formato PDF.

El aspecto cuantitativo del enfoque permite medir objetivamente el rendimiento de cada lenguaje en términos de tiempo de ejecución, uso de CPU, margen de error, nivel de satisfacción del usuario. Estas métricas son cruciales para evaluar la capacidad de los lenguajes para optimizar los procesos computacionales en un entorno de procesamiento paralelo en la nube.

Por otro lado, el aspecto cualitativo es esencial para evaluar factores como el análisis del proceso de generación de reportes trimestrales en formato PDF, requerimientos

para desarrollar los módulos en lenguajes diferentes y el impacto de la productividad empresarial al contexto específico de la empresa de soluciones medioambientales. Estos aspectos son particularmente relevantes en el procesamiento paralelo, donde la complejidad de la programación puede variar significativamente entre lenguajes.

La combinación de ambos enfoques permite una evaluación más completa y equilibrada. Por ejemplo, mientras que las métricas cuantitativas pueden mostrar que un lenguaje es más rápido en la ejecución paralela, el análisis cualitativo podría revelar que otro lenguaje, aunque ligeramente más lento, ofrece una mejor mantenibilidad y escalabilidad a largo plazo en el contexto empresarial (Mite et al., 2024).

Este enfoque mixto se alinea con las investigaciones de Cuaran & Miranda (2019), quienes destacan la importancia de considerar tanto el rendimiento cuantificable como los aspectos cualitativos de usabilidad y escalabilidad en sistemas de procesamiento paralelo. En el contexto específico de una empresa de soluciones medioambientales, este enfoque integral es crucial para seleccionar un lenguaje de programación que no solo sea eficiente en términos de procesamiento, sino que también se adapte a las necesidades y capacidades específicas de la organización.

3.2 Alcance de la investigación

La presente investigación es considerada aplicada, ya que se orienta a resolver un problema pragmático específico. De este modo, el estudio surgió con el propósito de abordar la necesidad identificada en la empresa de soluciones medioambientales al momento de optimizar al proceso de creación de los reportes en PDF para manejar la carga en la Gerencia de Sostenibilidad. Por consiguiente, los resultados de la investigación tienen una aplicación concreta, porque la selección del lenguaje de programación adecuado puede reducir el tiempo invertido en el proceso de creación de reportes y los recursos empleados para éste (Zegarra & Reyes, 2023).

El estudio explorará el rendimiento de diferentes lenguajes de programación en un contexto de procesamiento paralelo, un área que no ha sido abordada exhaustivamente en la empresa. Esto permite descubrir nuevas perspectivas y optimizaciones para sistemas similares (Constanzo, 2021).

A través de esta investigación se presentan los resultados cuantitativos y cualitativos de los lenguajes Golang, Python, F#, y Node.js, seleccionados por su capacidad para el procesamiento paralelo, popularidad industrial, compatibilidad con herramientas para generación de PDFs, y estudios previos que destacan su rendimiento en aplicaciones intensivas en recursos. Los criterios, definidos según las necesidades de la empresa de soluciones medioambientales, permiten evaluar su eficiencia, consumo de recursos y escalabilidad en la generación masiva de reportes, describiendo cómo cada lenguaje maneja cargas elevadas para determinar su idoneidad en el sistema propuesto (Cusi, 2023).

3.3 Delimitación del proyecto

La delimitación del proyecto abarca diferentes dimensiones, lo cual permite acotar la investigación para que los resultados sean específicos y aplicables al contexto de la empresa de soluciones medioambientales:

Delimitación geográfica: El ámbito geográfico de esta investigación es la sede de la empresa de soluciones medioambientales en la ciudad de Guayaquil, República del Ecuador. El sistema de informes se probará in situ para su respectivo análisis del rendimiento y funcionamiento.

Delimitación temporal: La investigación se realiza durante el año 2024. Las pruebas por realizar en este marco temporal permitirán que, en cada uno de los lenguajes, se cuente con pruebas exhaustivas bajo condiciones de trabajo en temperaturas y otras variables, lo que facilitará la realización de ajustes en base al aprendizaje en las primeras pruebas.

Población y muestra: La población del caso de estudio corresponde al sistema completo de generación de reportes en formato PDF implementado en el área de Sostenibilidad de la empresa de soluciones medioambientales, donde se centralizan las actividades administrativas relacionadas con la sostenibilidad. La muestra, por su parte, está constituida por cuatro implementaciones específicas del sistema, desarrolladas en distintos lenguajes de programación: Golang, Python, F# y Node.js. Estas implementaciones representan versiones funcionales del sistema diseñadas para evaluar y comparar su desempeño en términos de eficiencia y capacidad de procesamiento paralelo.

De esta forma, la delimitación permitirá obtener resultados específicos y aplicables para el entorno operativo de la empresa, abriendo las puertas a otros procesos similares que se desarrollarán en el futuro.

3.4 Métodos empleados

Los métodos de adquisición y análisis de datos incluidos en el presente trabajo de investigación es una mezcla de métodos empíricos y pruebas de carga, y se seleccionan para evaluar el rendimiento técnico de cada lenguaje de programación. A continuación, se presentan estos métodos de trabajo a realizar:

Observación empírica: Una observación directa y continua del comportamiento de cada lenguaje bajo condiciones de carga moderada y alta. Los datos observados incluyen tiempos de respuesta, uso de CPU, y margen de fallos. Esta observación se realiza mediante las herramientas de monitoreo del sistema integradas y otros módulos de análisis de rendimiento (Suárez & Henry, 2023).

Entrevista: Se llevaron a cabo entrevistas semiestructuradas con el personal clave involucrado en el proceso de generación de reportes trimestrales. El objetivo principal fue analizar en detalle el proceso actual e identificar los requerimientos específicos para el desarrollo de los módulos del sistema.

Encuesta: Se diseñaron y aplicaron encuestas a los usuarios finales del sistema para evaluar aspectos cualitativos y cuantitativos. Las métricas evaluadas fueron: nivel de satisfacción del usuario, impacto en la productividad Empresarial. Las encuestas incluyeron preguntas con escala Likert para obtener datos cuantitativos, así como preguntas abiertas para recopilar información cualitativa más detallada.

Pruebas de carga y rendimiento: La última etapa consistirá en las pruebas de carga, que involucrarán el uso de scripts personalizados para simular cada vez más usuarios que acceden al sistema simultáneamente. Este proceso permitirá evaluar cómo se comporta el sistema cuando se somete a demandas máximas y analizarlo en términos de los parámetros deseados. El objetivo de estas pruebas es medir los parámetros específicos, ya sea el tiempo de ejecución promedio, máximo y la capacidad del sistema para soportar cargas completas y picos agudos. Además, se monitorearán los posibles cuellos de botella, la capacidad del sistema para alojar una concurrencia alta y moderada y la intensidad con la que maneja los posibles problemas. (Peñafiel, 2022).

Análisis de datos: Para la recolección de datos cuantitativos sobre el rendimiento de cada módulo, se desarrollaron scripts personalizados que evaluaban métricas clave como la velocidad de procesamiento, el uso de CPU y el margen de fallos. Estos scripts generaban automáticamente archivos PDF que presentaban de manera estructurada la información obtenida en cada prueba de rendimiento, facilitando así el análisis posterior.

En cuanto a los datos cualitativos, se empleó un análisis de contenido riguroso para procesar la información recopilada a través de entrevistas y las respuestas abiertas de las encuestas. Este enfoque permitió identificar patrones, temas recurrentes y percepciones relevantes de los usuarios.

Para el procesamiento y visualización de los datos, se utilizó una hoja de cálculo como herramienta central. Esta elección permitió la generación eficiente de gráficos comparativos y tablas detalladas, ofreciendo una representación clara y visual de los

resultados obtenidos. Este método facilitó la comparación directa entre los diferentes lenguajes de programación evaluados.

La integración de métodos cuantitativos y cualitativos proporcionó una visión holística del rendimiento técnico de cada lenguaje de programación. Además, este enfoque permitió evaluar de manera efectiva el impacto de cada solución en la satisfacción del usuario y la productividad empresarial, aspectos cruciales para la toma de decisiones en el contexto organizacional.

Con el objetivo de garantizar la transparencia y replicabilidad del estudio, todos los scripts utilizados, los conjuntos de datos anonimizados y las herramientas empleadas han sido meticulosamente documentados y almacenados en un repositorio de GitHub.

3.5 Procesamiento y análisis de la información

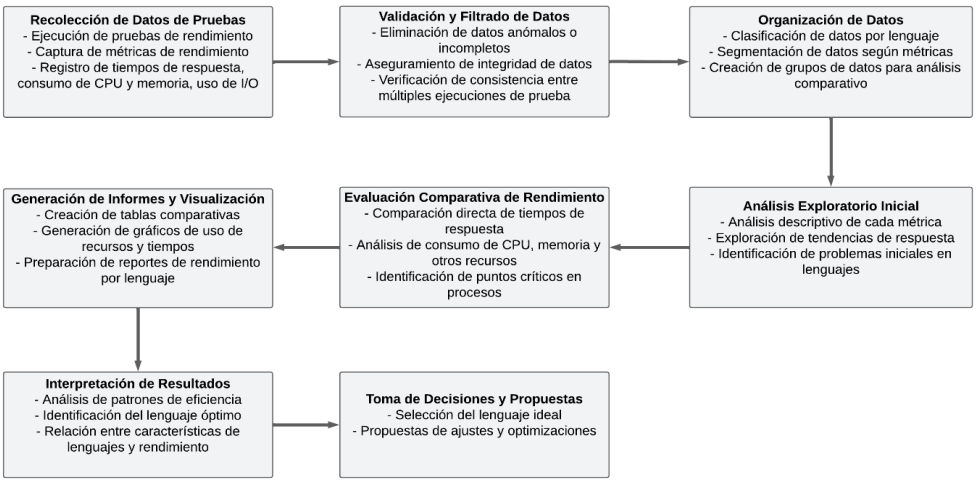
El procesamiento y análisis de la información constituyen una de las etapas más esenciales de la investigación como se observa en la figura 4. El procesamiento se realizó con el apoyo de herramientas de software especializadas, implementadas dentro de la metodología de desarrollo de software Kanban, y comprendió los siguientes pasos:

- Los escenarios de carga para cada lenguaje de programación involucraban condiciones de carga bajas y altas, simulando el estado real de las operaciones. Como tal, el primer paso incluyó configurar as un número específico de usuarios concurrentes y el alcance de las tareas delimitadas para su ejecución en el sistema generador para un reporte PDF.
- Las pruebas de carga controladas para cada lenguaje de programación representaron escenarios de uso. Dichas pruebas automáticamente procesaron una serie de evaluaciones del tiempo de respuesta, el rendimiento de los módulos de carga y la respuesta del sistema en los procesos de aplicaciones de carga altos y prolongados (P. V. Ruiz, 2020).

- Durante las pruebas, los módulos recogieron indicadores de rendimiento en todos los aspectos de la prueba, desde el tiempo promedio de respuesta y la desviación estándar en los tiempos de ejecución hasta el uso de la CPU y el número de errores. Todos los datos se recogieron y presupuestaron para organizarlos postproducción y análisis (Villamarín, 2023).
- La recolección de pruebas permitió comparar los indicadores de rendimiento, con el fin de identificar un patrón y determinar cuál es el lenguaje que se destaca en términos de eficiencia en el manejo de la carga, implementación estable y desempeño escalable. Los resultados en muchos casos se presentaron visual y cuantitativamente mediante gráficos y tablas que mostraban el máximo y el mínimo estimado de cada lenguaje (Montes, 2023).

Figura 4

Diagrama del análisis y procesamiento de datos del proyecto



3.6 Elementos metodológicos específicos para TI

La implementación y evaluación de los módulos de procesamiento paralelo exigidos por esta investigación requieren herramientas y enfoques metodológicos creados exclusivamente para los entornos tecnológicos y de desarrollo de software. A continuación,

se presentan los elementos metodológicos, desde la organización del flujo de trabajo hasta las pruebas de rendimiento y eficiencia de los lenguajes de programación deseados programando.

3.6.1 Diseño del Proyecto

- **Tipo de Proyecto:** Proyecto de desarrollo tecnológico
- **Modelo de Desarrollo:** Se implementa un modelo Kanban del desarrollo ágil, que permite gestionar las tareas de forma visual y flexible mediante un tablero Kanban, desarrollar e integrar módulos de manera iterativa e incremental, monitorear y controlar el flujo de trabajo en las etapas de planificación, desarrollo, pruebas y validación.

3.6.2 Recopilación de Información

- **Fuentes de Datos:** Resultados generados en las pruebas de cada módulo, encuestas y entrevistas.
- **Herramientas de Recopilación:** Google Formularios, Google Hojas de Cálculo y scripts de registros.
- **Procedimientos de Recopilación:** Se configura el entorno para recopilar las métricas de procesamiento mediante scripts personalizados que generan archivos de registros (logs). Las encuestas serán completadas por los usuarios involucrados en esta investigación mediante Google Formularios. Los resultados se recopilarán y exportarán a Google Hojas de Cálculo para su posterior análisis. Se realizarán entrevistas con los usuarios en la presente investigación para conocer el nivel de satisfacción.

3.6.3 Desarrollo y Diseño

- **Planificación:** El proceso se organiza en el tablero Kanban con columnas:

Por hacer → En progreso → Validación→ Finalizado. De esa manera, se puede rastrear cualquier actividad en detalle y, en consecuencia, identificar posibles cuellos de botella (García, 2024).

Las actividades planificadas incluyen: Selección de lenguajes de programación, diseño de algoritmos, desarrollo de los módulos, pruebas de cada módulo, recolección y análisis de los datos (ver figura 5).

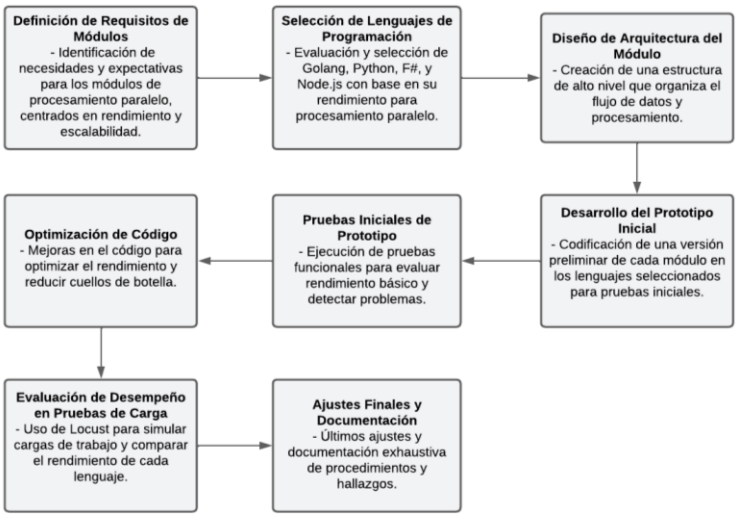
Límites de trabajo en proceso: Se establecen límites en la cantidad de trabajo en curso para cada columna, lo que significa que solo puede haber un número limitado de tareas en una fase a la vez. Esto evita que se acumule demasiada carga en una sola etapa y, por lo tanto, mejora la asignación de recursos (Miranda, 2024).

Análisis del flujo de trabajo: Al rastrear todas las tareas en el tablero se observa el flujo de trabajo en vivo, lo que proporciona áreas susceptibles para la mejora y los ajustes necesarios en los recursos o la reasignación (Fuentes & Pérez, 2022).

Revisión: Con el uso de Kanban como se observa en los anexos, es sencillo revisar con frecuencia el progreso, de manera que se puede hacer ajustes dependiendo de los resultados de las pruebas carga (Torres, 2024).

Figura 5

Diagrama del desarrollo y diseño del proyecto



- **Requisitos del Sistema:** Entre los requisitos funcionales están la conversión de hojas de cálculo a archivos PDF, organizar los archivos convertidos en directorios específicos en la nube.
- **Diseño del Sistema:** El sistema incluye módulos independientes en Python, Go, F#, Node.js para procesamiento paralelo.
- **Tecnologías y Herramientas:** Lenguajes de programación Python, Go, F#, Node.js, librerías necesarias para la conexión a Google Drive, uso de la API de Google, Visual Studio Code, Google Formulario y Google Hojas de Cálculo.

3.6.4 Implementación

- **Descripción del Proceso:** La implementación del sistema de pruebas se lleva a cabo en los siguientes pasos: Se desarrollan módulos específicos a cada lenguaje de programación, especialmente para procesamiento paralelo y generación de documentos PDF.
- **Ambiente de Desarrollo:** IDE Visual Studio Code.
- **Pruebas y Validación:** Pruebas automatizadas para evaluar rendimiento y eficiencia. Validación de resultados mediante análisis comparativo entre lenguajes.

3.6.5 Análisis de Datos

- **Métodos de Análisis:** Se utilizará un análisis comparativo basado en:

  Métricas cuantitativas como los tiempos de procesamientos, uso de CPU/memoria.

  Métricas cualitativas para medir el impacto de la productividad empresarial

  Herramientas de Análisis: Google Hojas de Cálculo (Gráficos y tablas).
- **Interpretación de Resultados:** Se interpretarán los resultados de cada lenguaje de programación mediante el análisis de métricas de rendimiento evaluando así el comportamiento del sistema bajo diferentes cargas de trabajo. Además, se analizarán los resultados de las encuestas para medir el impacto en la productividad

empresarial y el nivel de satisfacción del usuario con la finalidad de afianzar los resultados respecto al sistema.

3.6.6 Evaluación del Proyecto

- **Criterios de Evaluación:** Eficiencia de los lenguajes en términos de tiempo de procesamiento, uso de CPU y margen de fallos, impacto de la productividad empresarial en el contexto de la empresa de soluciones medioambientales y nivel de satisfacción del usuario.
- **Instrumentos de Evaluación:** Scripts de medición automática (tiempo, cpu). Comparación gráfica de resultado.

3.6.7 Consideraciones Éticas

El proyecto cumple con los principios éticos al no comprometer datos privados ni sensible y al desarrollar pruebas en un entorno controlado.

3.6.8 Cronograma de actividades

A continuación, se detalla el cronograma de actividades del desarrollo del proyecto:

TABLA 1

Cronograma de actividades para el proceso de desarrollo

Actividades	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10
Selección de lenguajes de programación- Diseño de algoritmos										
Desarrollo del módulo en lenguaje Golang										
Desarrollo del módulo en lenguaje NodeJS										
Desarrollo del módulo en lenguaje Python										
Desarrollo del módulo en lenguaje F#										
Prueba de procesamiento de 1000 documentos en los 4 lenguajes										
Prueba de procesamiento de 3000 documentos en los 4 lenguajes										
Prueba de procesamiento de 6000 documentos en los 4 lenguajes										
Recolección de datos de prueba- Validación y filtrado de datos										
Organización de datos- Análisis exploratorio inicial										
Evaluación comparativa de rendimiento										
Generación de informes y visualización										
Interpretación de resultados- Toma de decisiones y propuestas										
Documentación										

4 Análisis de Resultados

Una evaluación de los resultados obtenidos durante la implementación y verificación del sistema que genera documentos PDF en la nube utilizando procesamiento paralelo. Basándose en las recomendaciones metodológicas anteriormente consideradas, A continuación, se realizan una serie de comparaciones de los lenguajes de programación Golang, Python, F# y Node.js seleccionados. Se evalúa el rendimiento y eficiencia en el manejo de incrementos en los lotes de documentos.

Cada lenguaje es evaluado en función de su desempeño frente a incrementos en los lotes de documentos, generando 1000, 3000 y 6000 archivos para analizar cómo responden ante una carga creciente. Este enfoque permite identificar cuál de los lenguajes demuestra mayor rendimiento, eficiencia en el uso de recursos y capacidad de manejo de carga.

Con el fin de simplificar la organización y la ejecución de cada etapa de desarrollo, se empleó una metodología Kanban. Esto permitió una perspectiva visual sobre todas las tareas, cómo priorizarlas, planificarla y favoreció un flujo continuo de trabajo que resultó fundamental a medida que se completaba el desarrollo y las pruebas.

4.1 Fase 1: Análisis sistemático

4.1.1 Análisis del proceso actual de generación de reportes trimestrales en formato PDF

Este análisis tuvo como objetivo profundizar en el proceso actual que utiliza la empresa de soluciones medioambientales para la generación de reportes trimestrales en formato PDF, identificando sus limitaciones y áreas de mejora. Para obtener información detallada y precisa, se realizó una entrevista con el Técnico de Desarrollo y Tecnología de la gerencia de sostenibilidad, quien es el encargado del proceso (ver anexo 15).

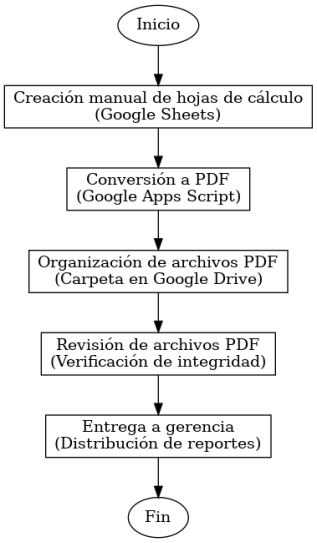
Actualmente, el proceso de generación de reportes trimestrales consiste en los siguientes pasos:

- Creación manual de hojas de cálculo: Los datos necesarios para los reportes se ingresan y organizan manualmente en archivos de hojas de cálculo (Excel o Google Sheets).
- Conversión a PDF: Cada hoja de cálculo se convierte a un archivo PDF de forma automatizada mediante Google Apps Script.
- Organización de archivos PDF: Los archivos PDF generados se almacenan y organizan en carpetas en la nube de Google Drive.
- Revisión y entrega: Los reportes PDF se revisan para verificar su integridad y se entregan a la gerencia correspondiente.

Para una mejor comprensión de lo mencionado, en la figura 6 se puede visualizar un diagrama de flujo del proceso de generación de reportes trimestrales en formato PDF.

Figura 6

Diagrama de flujo del proceso de generación de reportes trimestrales



A partir de la entrevista, se identificaron las siguientes limitaciones en el proceso actual:

Restricción de tiempo de ejecución: Google Apps Script tiene un límite de ejecución de 30 minutos por proceso, lo que resulta insuficiente para procesar grandes volúmenes de archivos.

Interrupciones frecuentes: Cuando el número de archivos a procesar es elevado, el script se detiene antes de completarse, generando interrupciones y la necesidad de intervención manual.

Resultados inconsistentes: En casos de alta demanda, algunos archivos PDF generados resultan incompletos o vacíos, afectando la integridad de los reportes finales.

Intervención manual ineficiente: La necesidad de que el técnico verifique manualmente el punto de interrupción y reinicie el proceso genera pérdida de tiempo y esfuerzo, lo que impacta en la productividad.

Escalabilidad limitada: El proceso no puede manejar grandes cantidades de reportes (por ejemplo, miles de archivos) de manera eficiente debido a las restricciones mencionadas

A partir de las limitaciones observadas, se proponen las siguientes áreas de mejora.

Optimización del tiempo de procesamiento: Implementar un sistema basado en procesamiento paralelo que permita distribuir la carga de trabajo y reducir los tiempos de generación de archivos PDF.

Automatización robusta y continua: Sustituir el actual flujo de Google Apps Script por un sistema más robusto, capaz de manejar procesos más extensos sin interrupciones.

Manejo de errores y validación: Incorporar mecanismos automáticos de verificación de archivos para asegurar que los archivos PDF generados sean correctos y completos.

Eliminación de intervención manual: Minimizar la intervención humana mediante sistemas que permitan la reanudación automática del procesamiento en caso de fallos o interrupciones.

Escalabilidad del sistema: Diseñar un sistema escalable, que pueda procesar miles de archivos simultáneamente sin restricciones de tiempo ni inconsistencias en los resultados.

4.1.2 Selección de lenguajes de programación

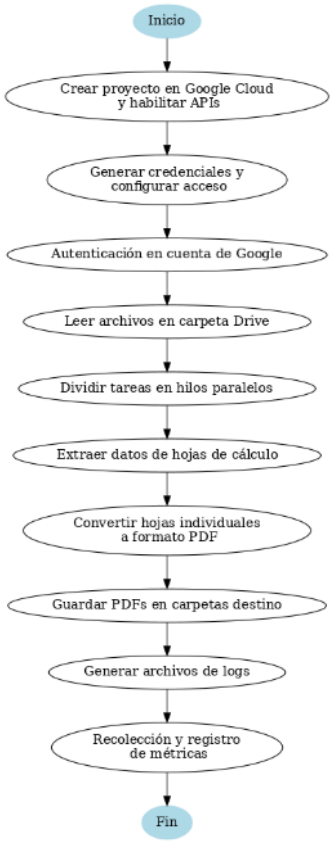
En base a la información proporcionada durante la entrevista con el jefe de Transformación Digital, se seleccionaron los siguientes lenguajes de programación para la implementación del sistema de generación de reportes PDF con procesamiento paralelo: Golang, Node.js, Python y F#.

4.1.3 Diseño de algoritmos

En esta fase se describe el algoritmo general implementado para el desarrollo de los módulos en los lenguajes de programación seleccionados: Golang, Node.js, Python y F#. En la figura 7 se puede visualizar el algoritmo el cual detalla los pasos lógicos y secuenciales que se ejecutan para realizar el procesamiento paralelo de archivos de hojas de cálculo en Google Drive y convertirlos en archivos PDF de manera automatizada.

Figura 7

Diagrama de flujo algoritmo el cual detalla los pasos lógicos y secuenciales



4.1.4 Análisis de Requerimientos

Los requerimientos del sistema para este desarrollo fueron recolectados a través de un proceso exhaustivo que consideró los requerimientos de la empresa y las fuentes de información. En otras palabras, se evaluaron tanto las necesidades funcionales como las no funcionales del sistema, teniendo en cuenta que estaría sometido a altas exigencias. Asimismo, se analizaron las condiciones operativas, considerando factores como el volumen de documentos, factibilidad y contabilidad del aumento de alcance y requerimientos de informes y procesos, escalabilidad y habilidad para soportar cargas pesadas.

Requerimientos Funcionales

Los requerimientos funcionales fueron necesarios para describir cómo debería trabajar el sistema en términos de desempeño y confiabilidad. Dado que la tarea principal fue la generación de documentos a gran escala, los requerimientos debían garantizar que el sistema genere documentos sin sacrificar el rendimiento ni la estabilidad. Estos se derivan de las necesidades específicas observadas en el análisis de las actividades operativas y administrativas en la empresa. En general, el requerimiento referente a la fase de uso fue:

Capacidad de Generación Masiva**:** El sistema debería generar documentos PDFs en gran cantidad de manera confiable, ya sea de manera secuencial o en paralelo. La evaluación de la capacidad de generación masiva ocurrió en pruebas que simulaban cargas de trabajo para 1000, 3000 y 6000 documentos. Esa gama de volúmenes permite detectar si cada lenguaje conservaba la rapidez y exactitud necesarias a medida que la demanda de documentos crecía

Requerimientos No Funcionales

Los requerimientos no funcionales se derivaron del análisis de las expectativas de eficiencia, estabilidad y experiencia de usuario necesarias para garantizar que el sistema

cumpla con los estándares de calidad requeridos por la empresa. Estos requisitos complementan los funcionales al enfocarse en aspectos clave como el desempeño bajo demanda y la confiabilidad del sistema.

Tiempos de Ejecución y Respuesta Óptimos: Un punto clave del sistema era la rapidez con la que podía generar y procesar documentos. Considerando que el enfoque del procesamiento paralelo era mejorar la velocidad de generación, se asumió una expectativa de velocidad de ambos lenguajes antes y después de las pruebas de carga. Para cada prueba, se aseguró medir y monitorear los tiempos de ejecución de ambos lenguajes y optimizar su generación de documentos y procesamiento por demanda masiva.

Estabilidad de funcionamiento: El programa debe ser capaz de lograr la consistencia de los resultados y una operación vigorosa con cargas elevadas. Esto puede lograrse si se implementa el patrón de tolerancia a fallas, no se produce conflicto o inconsistencia con las listas de documentos y se relativizan los experimentos de muchas solicitudes y generaciones. La estabilidad se mide por la cantidad de errores y el tiempo de fallo del sistema, asegurando que los estudios abarquen todas las circunstancias que el lenguaje pueda realizar sin una falta esencial en la producción de los PDFs.

4.2 Fase 2: Desarrollo de los módulos

El diseño del sistema de módulos permite una evaluación precisa y un análisis controlado de cada lenguaje. Entre los módulos y sistemas de soporte puedo destacar la configuración específica para ambos métodos de funcionamiento paralelo; permitió analizar el rendimiento de los lenguajes.

4.2.1 Desarrollo del módulo en lenguaje Python

El programa en Python desarrollado está enfocado en la gestión y conversión de archivos en Google Drive. El propósito de la creación del sistema automatizado es organizar y convertir Spreadsheets de Google en archivos PDF, finalmente almacenarlos en una determinada ubicación del Google Drive. La estructura del programa consiste en cuatro

módulos principales: main.py, GoogleDrive.py, auth.py y log_config.py, cada uno de los cuales tiene su propia funcionalidad. La implementación utiliza las propias bibliotecas del lenguaje Python y de terceros para establecer una conexión con Google Drive, controlar el flujo de trabajo y realizar una relación de registro de acceso. Las bibliotecas destacadas utilizadas en Python se mencionan a continuación:

Google API Client (googleapiclient): Es necesaria para integrar aplicaciones de Python con los servicios de Google. Se utilizan para realizar casi todas las operaciones de API en Google Drive: enumeración de archivos, descarga y carga de archivos. Algunos de los métodos en el código son build, que crea una instancia del servicio;.files().list(), que lista los archivos;.files().export_media().

PyDrive2 (pydrive2): Es también una biblioteca necesaria e ideada para autenticar y hacer llamadas a la API de Google Drive. Simplifica el manejo y la pasar credenciales utilizando una sesión. Ayuda a reanudar operaciones después de que una sesión caduque sin problemas debido al token de autenticación.

Ratelimit: Es una biblioteca que nos permitirá prolongar llamadas entre solicitudes a la API de Google Drive. Usamos la función limits para establecer la cantidad de solicitudes en un período específico. Define la cantidad en GoogleDrive.py. gracias a esta línea de código, el programa no excederá el número máximo de llamadas permitidas por la cuota para la API de Google.

Logging: Es esencial para registrar la ejecución del programa para su rutina y resolución de problemas. En el siguiente módulo log_config.py, durante su ejecución, se genera un archivo de registro en la carpeta dada por una configuración de RotatingFileHandler para que el tamaño del archivo de log siempre esté controlado.

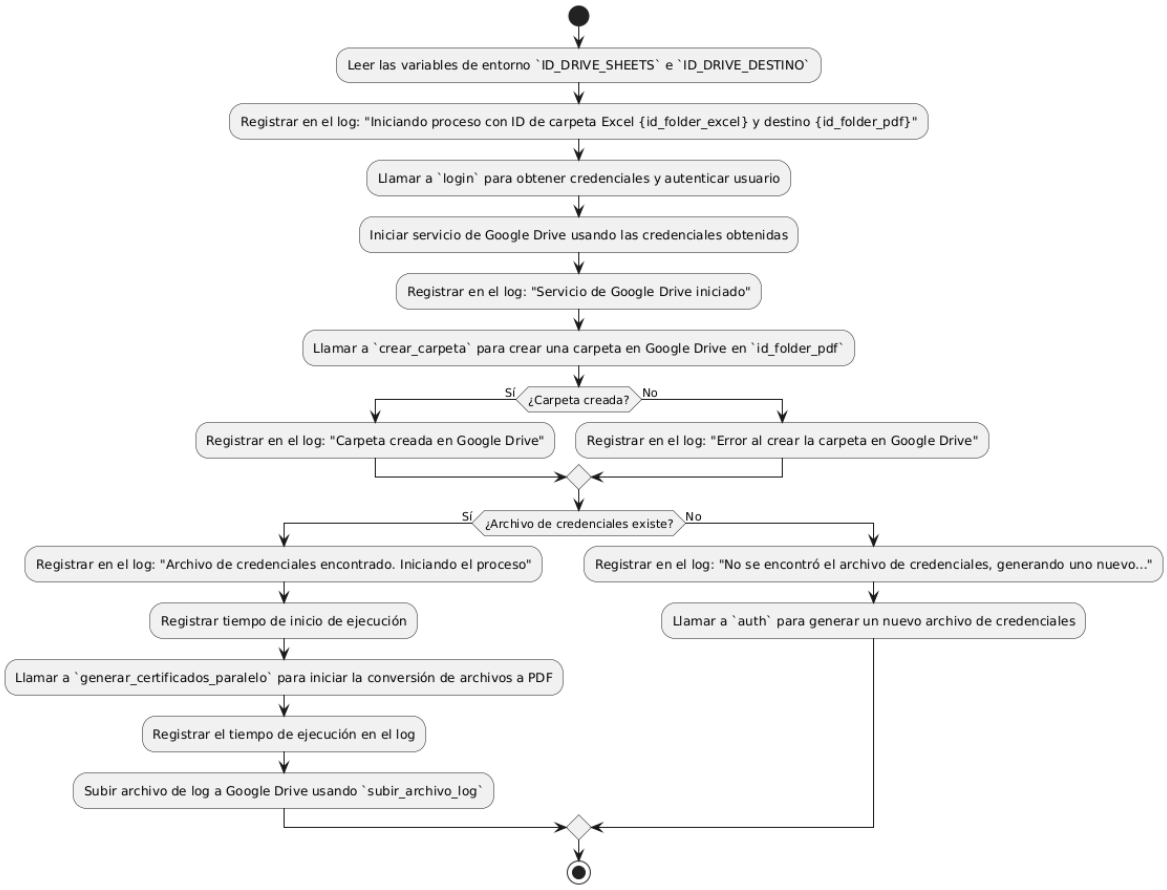
Concurrent.futures: Es una biblioteca nativa de Python que facilita la realización de tareas paralelas. Se usa para llevar a cabo múltiples conversiones de archivo a PDF al

mismo tiempo con ThreadPoolExecutor. Mejora significativamente el rendimiento y el tiempo de ejecución del programa al dividir las tareas.

En el diagrama de la figura 8 se observa el establecimiento de conexión con Google Drive y el inicio de sesión con la función login para obtener las credenciales del usuario. Posteriormente, se inicializa el servicio de Google Drive y se crea una carpeta de destino. Si ya hay credenciales guardadas, se registra el tiempo de inicio, se procesan y convierten en PDFs los archivos Excel con generar_certificados_paralelo, se mide el tiempo de operación y se sube el archivo de log a Google Drive; de lo contrario, se generan credenciales con la función auth.

Figura 8

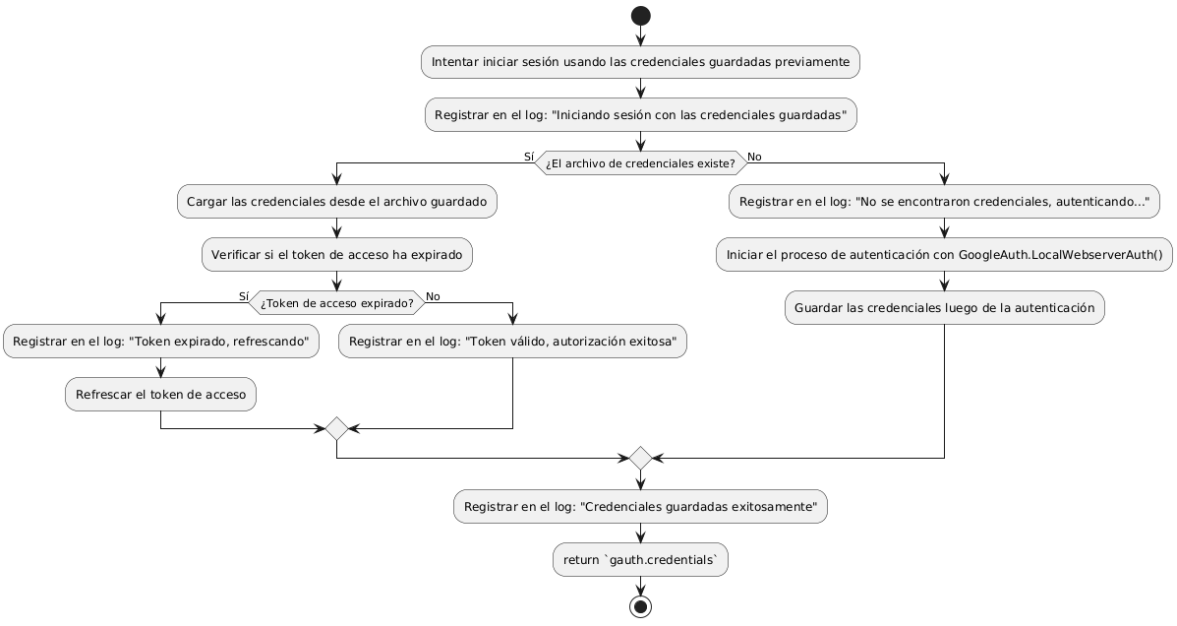
Diagrama de flujo del main en Python



El diagrama de la figura 9 se observa el proceso de autenticación del usuario en Google. Primero, comprueba si el archivo de credenciales ya existe. Si es así, carga las credenciales almacenadas y valida si el token de acceso ha caducado. En caso afirmativo, refresca el token para recibir uno nuevo. De cualquier modo, simplemente autoriza al usuario si el token actual es válido.Compruebe la validez de Token. De lo contrario, si no existe el archivo de credenciales, el proceso se inicia con GoogleAuth.LocalWebserverAuth(). Pide al usuario que inicie sesión y autentique el acceso del cliente. Una vez completado el operativo, se almacenas las credenciales recién obtenidas en el archivo para próximas sesiones. Por último, devuelva las credenciales para usarlas con las otras actividades.

Figura 9

Diagrama de flujo del iniciar sesión y autenticación en Python

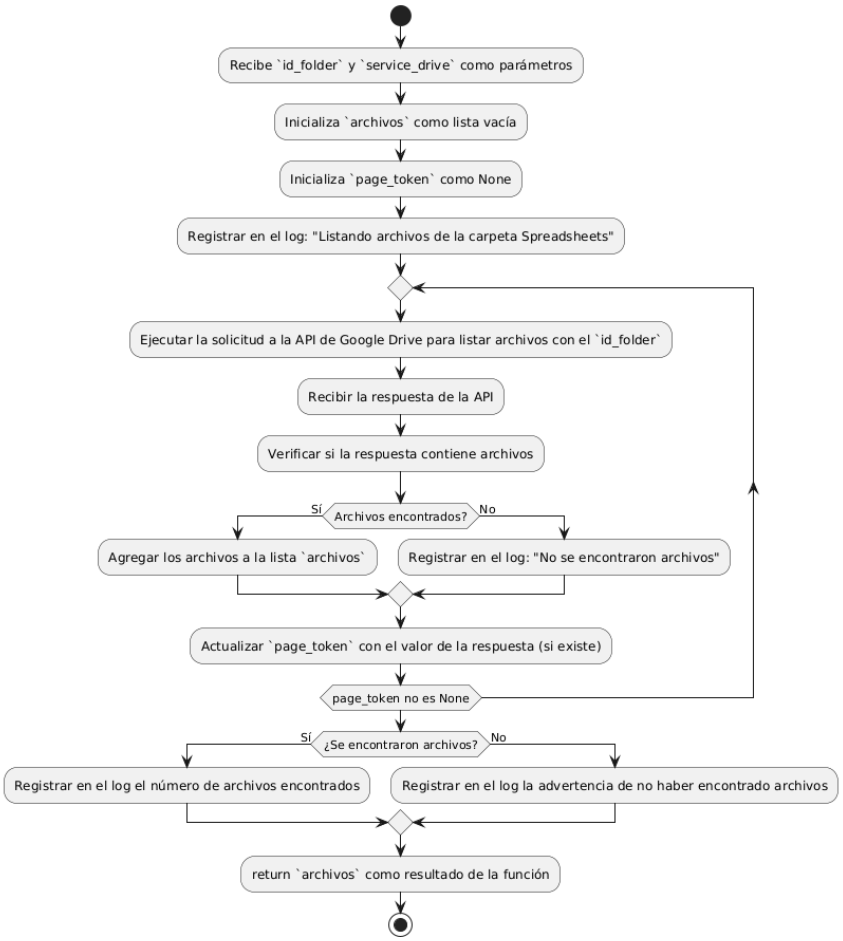


En el diagrama de la figura 10 se describe el proceso de listar los archivos contenidos en una carpeta en Google Drive. Para ello, la función recibe los parámetros id_folder y service_drive, luego inicializa una lista vacía para guardar los archivos y un

page_token para la paginación. Posteriormente, envía una solicitud a la API de Google Drive para listar todos los archivos. En caso de encontrar archivos, los agrega a la lista, de lo contrario, escribe un mensaje en el log. El paso anterior se repetirá si hay más archivos para mostrar, es decir, page_token no es None, luego, al final del proceso, se registran los resultados, así como la lista de archivos encontrados. Si no encuentra ningún archivo, un mensaje de advertencia estará presente en los resultados.

Figura 10

Diagrama de flujo para listar los archivos en Google Drive en Python

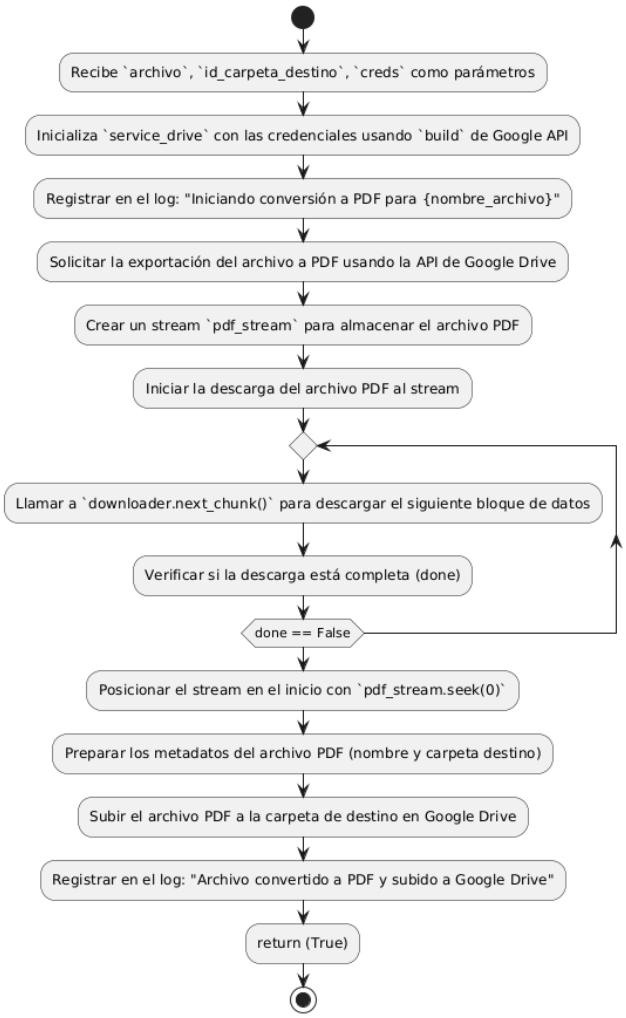


A continuación, se muestra en la figura 11 un diagrama del proceso para convertir un archivo de Google Drive en formato PDF, así como cargarlo de nuevo en Google Drive. Los parámetros son el archivo, la carpeta de destino, y las credenciales. Primero, se

inicializa el servicio con las credenciales proporcionadas para Google Drive. Luego, se presenta una solicitud a la API de Google Drive para exportar el archivo a PDF. Luego, el archivo PDF se descarga en un stream y una vez que se completa la descarga está listo para subirse a Google Drive. Luego, se configuran los metadatos del archivo PDF , es decir, se establecen el nombre y la ubicación previa al envió y se sube a la carpeta correspondiente. En el último paso, se registra en el log que el archivo dada se ha convertido y se ha subido correctamente.

Figura 11

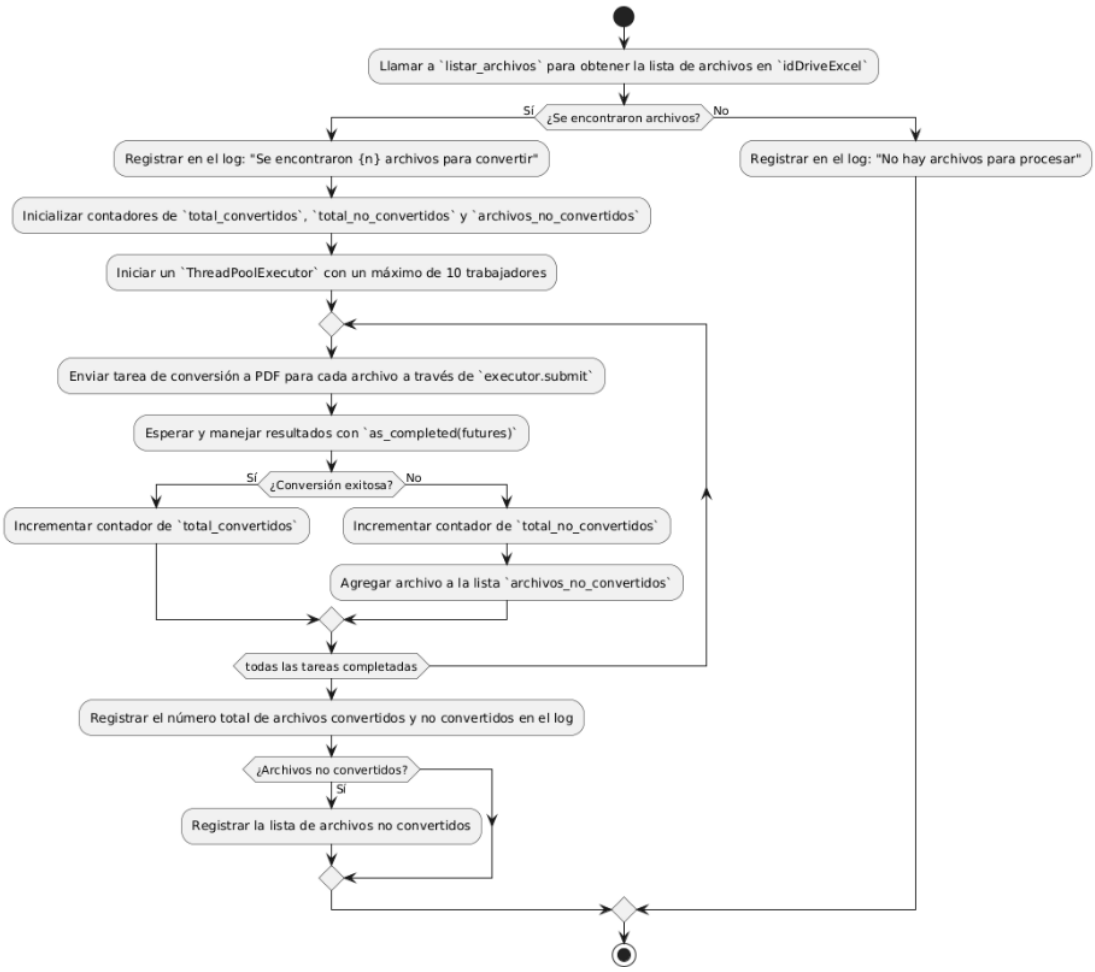
Diagrama de flujo para convertir los PDFs en Google Drive en Python



En la figura 12 se observa diagrama de flujo que muestra el proceso de generación de certificados en formato PDF esquematizado de la siguiente manera, primero se obtiene la lista de archivos a convertir con la función listar_archivos. Si los archivos se encuentran, inicia ThreadPoolExecutor con un número máximo de trabajados de 10. Se envía una tarea de conversión a PDF para cada archivo. Las tareas se ejecutan de forma paralela y después de completarse todas se verifica su éxito. Si se ha convertido con éxito se incrementa el contador de los archivos convertidos, en caso contrario, incrementamos el de no convertidos y agregándolo a la lista de los archivos fallidos.

Figura 12

Diagrama de flujo para generar los certificados en paralelo en Python



4.2.2 Desarrollo del módulo en lenguaje F#

El objetivo del programa desarrollado en F# es automatizar la conversión de archivos de Google Sheets a PDF en Google Drive, y agregar la funcionalidad de registros de logs compuesta de dos estructuras, que firmen cualquier proceso dentro del flujo que acaezca durante la ejecución del programa, y efectuarse de todas las excepciones que puedan surgir durante la ejecución del programa. El programa se estructura en módulos (DotEnv.fs, GoogleDriveService.fs, logConfig.fs, main.fs) que contienen las funciones que son específicos al manejo de la configuración y operación del servicio de Google Drive, tareas de manejo de logs y el flujo principal de la aplicación. La organización modular del programa implementa una clara separación de responsabilidades, por lo cual el código es más fácil de depurar y mantener. Las bibliotecas destacadas utilizadas en F# se mencionan a continuación:

Dotenv.net: Es una librería que nos permite cargar variables de entorno desde un archivo.env. Esto es útil, principalmente, para configurar la aplicación con credenciales y configuraciones externas sin correr el riesgo de exposición al código de manera insegura. Contribuye así a la portabilidad y seguridad del programa.

Google.Apis: Es la librería SDK base para las APIs de Google. Implica la comunicación con, y la gestión de, los servicios en la nube de Google. Contiene el core necesario para autenticar y gestionar el acceso a los distintos servicios, en este caso Google Drive.

Google.Apis.Auth: Expone las funciones específicas para la gestión de la autenticación OAuth 2.0. Este componente es básico para la utilización segura del archivo client_secrets.json que contiene la identidad del cliente y le permite iniciar sesión en Google Drive.

Google.Apis.Drive.v3: Es específico para el servicio de Google Drive y se emplea para la creación de listas y la manipulación de archivos y carpetas. Adicionalmente, este

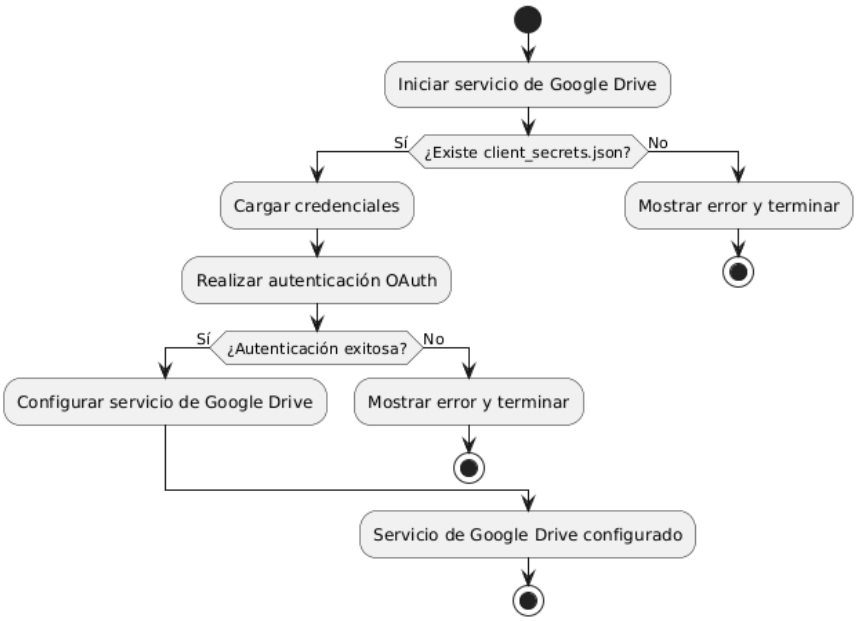
paquete facilita la exportación en varios formatos, en el caso de este programa, la conversión de Google Sheets a PDFs.

En la figura 13 se muestra el proceso mediante el cual se configura adecuadamente el proyecto en GoogleDrive.fsproj antes de comenzar a ejecutarse. El sistema lee el archivo fsproj del proyecto para averiguar cuáles son los archivos fuente y las dependencias necesarios para el correcto funcionamiento del programa.

Los archivos de código DotEnv.fs, GoogleDriveService.fs, logConfig.fs, y main.fs y las dependencias que se leen con el archivo fsproj están relacionados con autenticación y manejo de Google Drive: dotenv.net, Google.Apis, entre otros. Al hacer esto se garantiza que el proyecto estará correctamente configurado para el proceso de compilar y que todas las bibliotecas estarán disponibles antes de que comience la ejecución del programa.

Figura 13

Diagrama de flujo para Configuración del Servicio de Google Drive en F#

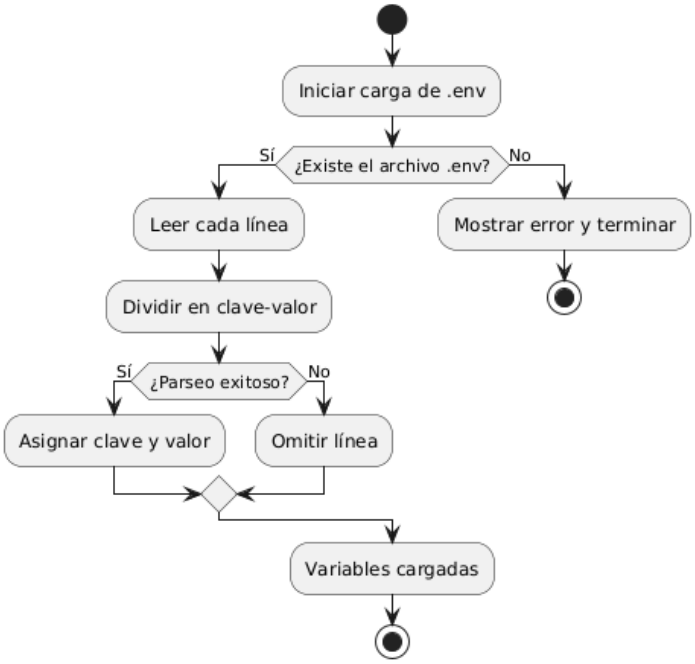


En la figura 14 se observa cómo el programa lee las variables de entorno desde un archivo.env. En este caso, primero verifica si el archivo.env realmente existe. Si el archivo

existe, carga cada línea; cada línea parseada de la siguiente manera: la línea se divide en dos porciones, una clave y un valor. Cada par clave-valor actual ha sido ajustado como una configuración esencial que requerirá más tarde. Si no puede analizar una línea o el archivo.env no existe, general un error y cierre el programa. Como resultado, el flujo de trabajo. las variables necesarias para configurar otros componentes como google service drive están ajustadas y listas.

Figura 14

Diagrama de flujo para carga de variables de entorno en F#

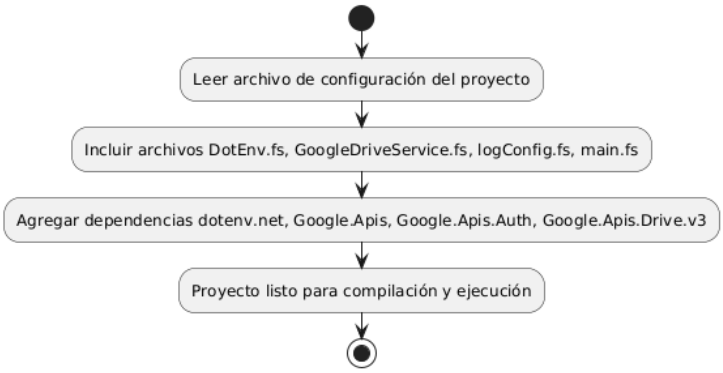


En la figura 15 se observa las condiciones para configurar el servicio de Google Drive, el cual será crucial para trabajar con archivos en la nube. En primer lugar, el programa busca el archivo con credenciales de Google, client_secrets.json. Si se encuentra, se intenta autenticar con Google mediante OAuth. Si tiene lugar la autenticación, el programa crea una conexión activa con Google Drive. En caso contrario, si no se logra autenticar o faltan credenciales, se genera un error y se suspende el proceso. Por lo tanto, de esta

forma, el programa se abre a la posibilidad de leer y modificar archivos en Google Drive de manera segura y verificada.

Figura 15

Diagrama de flujo para configuración del proyecto y dependencias en F#



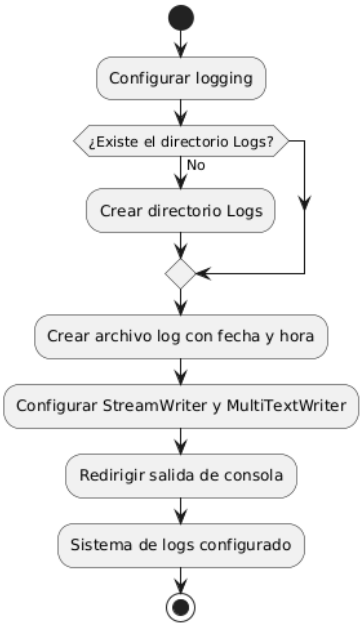
La figura 16 muestra el diagrama, el cual inicia con la verificación si el directorio ya existe en el sistema. Debido a que todos los archivos registrados por el sistema se grabarán en este lugar, es necesario asegurarse de que sea accesible. Si el directorio no existe, el programa se encarga de crearlo, garantizando que siempre haya un lugar en el que cada ejecución pueda almacenar su log.

En segundo lugar, se configura el archivo log propiamente, que se nombra con la fecha y hora actual. Esta nomenclatura facilita la consulta posterior en caso de que sea necesario revisar el registro de una ejecución particular o encontrar rápidamente la causa de algún problema. En este archivo, se inicializan dos componentes.

Primero, un StreamWriter, que como se indicó se encarga de administrar la escritura directa al archivo de log. Segundo, un \textit{MultiTextWriter}, que es una clase que cree para redirigir la salida de la consola: éxito, advertencias y errores, en el archivo de log. Esto se debe a que el propósito del log es asegurarse de que absolutamente todas las acciones, exitosas o no, sean registradas.

Figura 16

Diagrama de flujo para configuración del sistema de logs en F#



El diagrama de la figura 17 muestra las operaciones principales del programa y la estructura, subordinación de las operaciones entre sí. Primero, el sistema registra eventos en logging y luego se inicia el servicio principal, que se da de una manera estructurada, manteniendo la coherencia necesaria en el programa. Por lo tanto, se configura un entorno que permitirá que programa se ejecute de manera ordenada y documentada.

La configuración del logging asegura que todos los datos se ingresan en los logs para que todas las operaciones se realicen correctamente y sumultánea ya que algunos advanced data operations se realizan rápidamente o más lentos. Luego, el programa crea una carpeta dedicada para los archivos pdf convertidos.

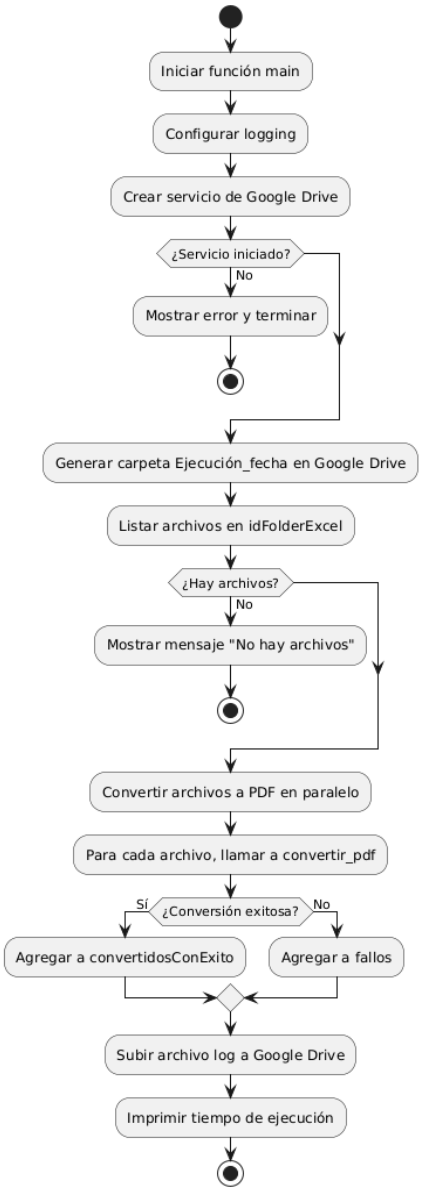
Gracias a esto, el sistema crea una estructura para todos los archivos convertidos, simplificando así el flujo de trabajo con excelibility y la administración de los archivos en Google Drive. Ya que esta carpeta ha sido configurada, el programa encuentra y obtiene la lista de archivos de hojas de cálculo en idFolderExcel de Google Drive. Si los archivos están

presentes, el programa los convierte en paralelo, lo que significa que todos los archivos se convierten simultáneamente con el programa.

Los archivos convertidos se registran, mientras que los no convertidos se registran en archivos de error. Durante el proceso, cada error que ocurre dentro del programa se documenta mediante archivos de errores, mientras que los archivos terminados se cargan en Google Drive, finalmente el archivo processing se almacena en Google Drive.

Figura 17

Diagrama de flujo función principal y conversión de archivos a PDF en F#



4.2.3 Desarrollo del módulo en lenguaje Golang

El programa, compuesto por los archivos main.go, auth.go, go.mod, go.sum, tiene el propósito de conectarse a Google Drive, listar los archivos de Google Sheets en una carpeta especificada, exportarlos en formato PDF y guardarlos en una carpeta de destino también en Google Drive. La conversión y la carga de los PDFs pueden realizarse de forma paralela, permitiendo disminuir el tiempo de procesamiento en caso de múltiples archivos. Por último, el programa realiza un seguimiento minucioso de los eventos y errores en logs que son impresos en la consola y guardados en un archivo. Tras finalizar la ejecución, el archivo de logs es incluso subido a Google Drive, en una carpeta que específicamente fue creada para la ejecución realizada. Las bibliotecas destacadas utilizadas en GO se mencionan a continuación:

Context: Es la librería de GO usada para manejar el contexto de ejecución de nuestras funciones abstrayendo varias operaciones del tiempo de ejecución de las rutinas y permitiendo el seguimiento en la cancelación de procesos en paralelos. Es fundamental al interactuar con Google API, ya nuestra gran cantidad de solicitudes HTTP a los servicios sean ejecutadas dentro del contexto adecuado.

Google.golang.org/api/drive/v3: Es la librería oficial de Google utilizada para acceder a la API de Google Drive. Da soporte en distintas funcionalidades, entre ellas listar, crear, actualizar y eliminar archivos en Google Drive, exportar a otro formato, como PDF, archivos de Google Sheets, etc.

Os y io: En este programa, la librería os está utilizando para la manipulación de archivos y directorios entre varias operaciones por nuestro conjunto de funciones mientras i está siendo usado para abstraer las operaciones de leer y escribir en archivos.

Sync: Es una librería que provee el paquete sync para herramientas de sincronización necesarias en procesamientos paralelos. En este caso, se utiliza sync.mutex y sync.waitgroup para coordinar rutinas y asegurar que las operaciones en paralelo no se

interfieran cuando estén modificando datos compartidos, ya sea, los contadores de conversiones exitosas y fallidos.

Time: Se utiliza para crear nombres de archivo con la fecha y hora actual, calcular el tiempo total tomado por el programa y hacer que las carpetas creado para la salida sean ordenadas y únicas para cada sesión separada.

Log: Se utiliza para permitir que el programa cree logs que puedan registrar los eventos importantes, los errores y los resultados en la consola y un archivo que facilite la depuración y monitoreo en ejecución.

Fmt: Es la librería estándar de GO que se sustenta en el formateo de la de cadenas. En este programa, se usa principalmente para formatear los mensaje que se van a ingresar en el log, los nombres de los archivos y generar consultas específicas para el Google Drive.

Bytes: Se utiliza para manejar datos en un buffer de memoria. se emplea para guardar el contenido del archivo PDF por parte del buffer y se sube a Google Drive.

El diagrama de la figura 18 representa el proceso de setup y autenticación inicial del programa durante la ejecución. Este paso es crítico para que todos los demás procesos ocurran sin ningún tipo de problema. El flowchart comienza con la setup completa del sistema de logging, que incluye también una configuración detallada del directorio en donde los registros de esta ejecución serán salvados.

Este punto es crucial atacando que diese ejecución de error o mala ejecución quieres verificar en cualquier punto del time. El logging también está setteado para insertar mensajes tanto en la terminal como en un file log, en un intento de balancear visualización inmediata con la post-contabilidad de un historial.

Después de este paso el programa empieza el proceso de asegurar que las credenciales del programa para acceder al sistema de API de Drive están presentes y con los parámetros adecuados. Este paso se llama auth y es necesario para establecer una

retrieval segura al authorized contra Google Drive para que le programa pueda llamar los comandos de reades y writers en el Drive account upload

Cuando esta etapa se cumple el sistema genera una única carpeta identifier para la foam basada en la hora y fecha actual, con las cuales el programa genera la foam en Google drive donde los programas files serán salvados una vez que sean driven y el log file. Si este paso resulta en un error el programas capture el hazard en el log de error y el programa termina la ejecución para evitar errores; si se loga, es no grabado el id y el programa cambia a la siguiente conversión de foam stages.

Figura 18

Diagrama de flujo de la configuración y autenticación en GO

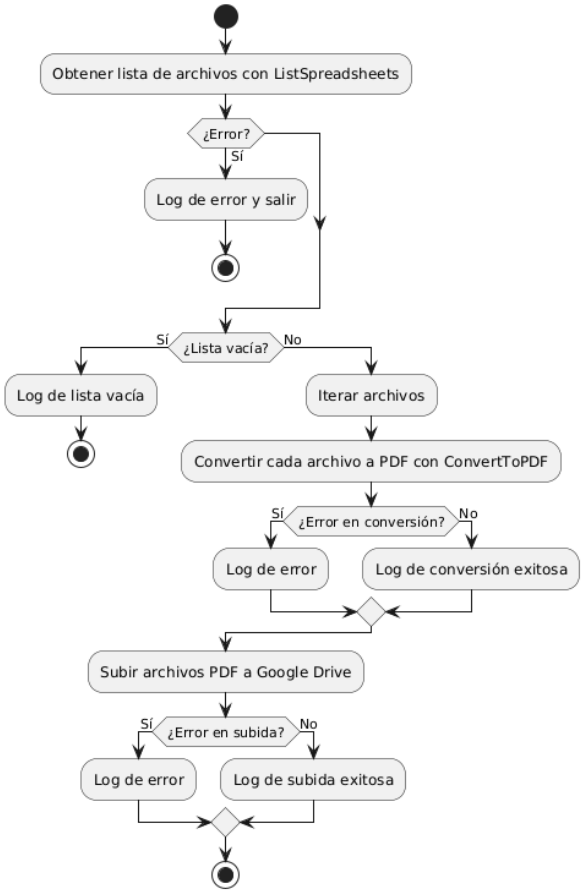
El diagrama de la figura 19 ilustra la secuencia de obtener y convertir archivos de Google Sheets en la ubicación original. En este, el programa devuelve la lista de archivos

en la ubicación de Google Drive dada, luego los filtra para dejar solo archivos de Google Sheets que no se encuentran en la papelera y verifica que haya archivos para convertir.

Después de eso, cada archivo se convierte a PDF utilizando una función que exporta el archivo de Sheets en el formato necesario. Al convertir, se pueden recorrer los archivos de manera paralela o secuencial. La secuencia de paralelismo, semáforo de goroutines y WaitGroup utilizado para limitar la cantidad de goroutines concurrentes y garantizar el uso eficiente de los recursos.

Figura 19

Diagrama de flujo de la generación y subida de PDFs en GO



El diagrama de la figura 20 corresponde al flujo de finalización del programa. En este paso, se calcula y registra el tiempo total que tomó toda la ejecución después de

completarse en el conversor de archivos. Después de eso, cargo el archivo de log generado a Google Drive en la carpeta de la ejecución.

Esto ayuda en el rastreo completo y se adhiere a toda la operación que se realizó. Si la carga del archivo de log falla, entonces el programa registra el error y el proceso se detiene. Finalmente, muestra un resumen de cuántos archivos fueron convertidos con éxito y cuántos fallaron en el proceso de conversión.

Figura 20

Diagrama de flujo de generación de PDFs y manejo de resultados en GO

4.2.4 Desarrollo del módulo en lenguaje Nodejs

El siguiente es un programa en Node.js que autentica al usuario a través de OAuth2, lo lleva a su unidad de Google Drive para enlistar sus archivos y luego convierte todas las hojas de cálculo de Google en un formato PDF. El programa está dividido en los siguientes archivos de código devolviendo: auth.js, que autentica al usuario por la utilización de la credencial OAuth2 del usuario y crea el cliente necesario para interactuar con la API de Google.

Otro es main.js, que contiene el flujo principal del programa, es decir, autenticación, la enunciación de archivos en Google Drive y la conversión de hojas de cálculo de Google en PDF, así como la organización de conversión en paralelo. Un archivo más es package.json, que rastrea las dependencias del proyecto, googleapis, express, dotenv, node-fetch y vencedor, y ejecuta los guiones necesarios para ejecutar el programa. Por lo tanto, se debe integrar varias funcionalidades con Google Drive y Google Sheets para enlistar y convertir archivos en PDF en un solo ensayo automatizado. Las bibliotecas destacadas utilizadas en GO se mencionan a continuación:

Googleapis: Esta biblioteca es crítica, ya que proporciona acceso a la API de Google, que a su vez se utiliza para acceder a Google Drive y OAuth2. Por lo tanto, se encarga de la autenticación y la emisión de comandos, para listar archivos y posteriormente exportarlos a un archivo.pdf, es fundamental para la interacción con los servicios de Google.

Express: Por otro lado, es un servidor web Node.js middleware de aplicación que también se utiliza para asegurarse de que las rutas, por ejemplo, /auth/google para iniciar la autenticación y /google/redirect para obtener los resultados de Google, estén habilitadas por ende, hace que las rutas y las solicitudes HTTP sean mucho más sencillas.

Dotenv: Se utiliza para cargar las configuraciones sensibles, como las credenciales de autenticación, desde un archivo.env. Esto nos permite mantener nuestras credenciales privadas mientras se configura nuestro programa sin realizar cambios en el código.

Node-fetch: Es una biblioteca que proporciona una forma de realizar solicitudes HTTP. En nuestro programa se utiliza para realizar solicitudes personalizadas a Google Drive, específicamente tratando las respuestas en formato JSON y administrando las cabeceras necesarias para la autenticación.

Winston: Es una biblioteca de registro que almacena eventos e información sobre el funcionamiento del programa. Este registro es útil para Depurar errores y errores del programa y rastrear el flujo del programa, y se necesitará para mantener un registro de las autenticaciones exitosas, las exportaciones de archivos y de cualquier error que surja durante la conversión.

Bottleneck: Es una biblioteca de limitación de la tasa que controla el número de solicitudes enviadas a la API de google cron jobs y represented queries. Al limitar la cantidad de solicitudes que se envían a la API de Google en un corto período, el uso de la API se mantendrá dentro de los límites.

Nodemon: Es una herramienta utilizada para la recarga automática para facilitar el desarrollo del programa. Nodemon se usa para facilitar el desarrollo, ya que reiniciará el programa después de que se realice un cambio y se pueda ver de inmediato cómo se verá sin necesidad de reiniciarlo manualmente.

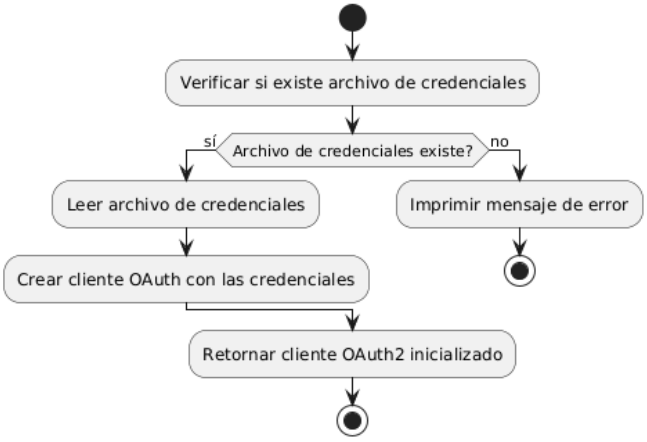
El diagrama de la figura 21 corresponde a la autenticación del usuario usando OAuth2. Como la autenticación es un proceso fundamental para asegurarse de que el programa solo tiene acceso a los servicios de Google autorizado, en este caso, Google Drive), sin comprometer la seguridad de la cuenta del usuario, auth.js se encarga de este proceso.

En primer lugar, verifica la existencia de credenciales válidas (por ejemplo, tal archivo, si existe, contendrá tokens de acceso previamente obtenidos) y si es necesario, redirige al usuario a la página de autorización de Google. Luego, recibe los permisos otorgados por el usuario a la aplicación para acceder a su cuenta de Google.

Luego, después de que Google redirige al usuario con el código de autorización, auth.js lo convierte en un token de acceso. Una vez obtenido, esta moneda es utilizada por el programa para acceder a los archivos en Google Drive. Además, este archivo recopila y configura el cliente OAuth2 con ayuda de las credenciales y, finalmente, devuelve una instancia inicializada para que el programa pueda interactuar con los servicios de Google de una forma permitida.

Figura 21

Diagrama de flujo de gestión de la autenticación OAuth2 en Nodejs



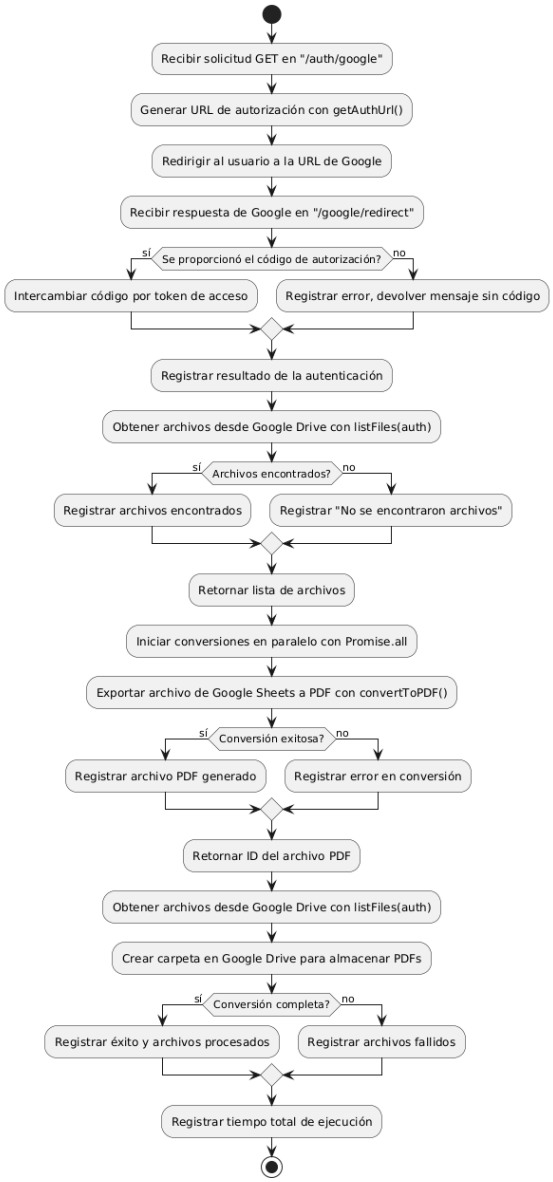
El diagrama de la figura 22 se observa la parte esencial de las funcionalidades de la aplicación. En primer lugar, se encarga de la autenticación del usuario cuando recibe una solicitud GET a la ruta /auth/google. Luego, dirige el acceso del usuario a Google para completar la autorización.

Adicionalmente, maneja la respuesta de Google, que incluye un código de autorización, y lo intercambia por un token de acceso, lo que permite autorizar los intentos subsiguientes. Una vez realizada la autorización, fileAccesseste archivo se encargado de accesar los archivos de Google Drive a través de la API de Google mediante el token de acceso, sí hay archivos en la carpeta, fileAccess igualmente lee los archivos presentes.

La conversión de la hoja de cálculo de Google a PDF es critical. Esto es posible exportando los documentos individualmente en el formato requerido y guardándolos en una carpeta de Google Drive. Las conversiones de la hoja de cálculo se leen de manera paralela a través de una biblioteca especial y de esta manera puede maniobrar múltiples archivos al mismo tiempo.

Figura 22

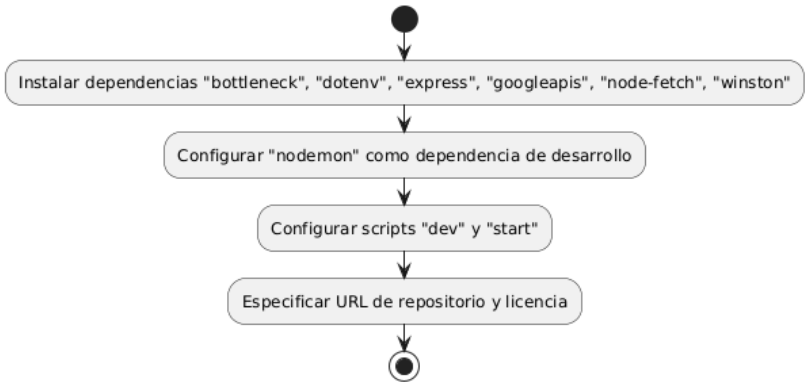
Diagrama de flujo de la autenticación, listado de archivos y conversión de Google Sheets a PDF en Nodejs



En el diagrama de la figura 23 se aprecia el registro de toda la información de configuración del proyecto y determinar las diversas dependencias necesarias para su funcionamiento. En este sentido, se muestran las bibliotecas que el programa utiliza para ejecutarse adecuadamente, como googleapis, que le permite conectarse a servicios de Google como Drive, Sheets, entre otros, la configuración del módulo de roles, que es utilizado para manipular las variables de entorno de forma más segura, tales como claves API y credenciales, y la configuración de winston, otro módulo utilizado para registrar los registros y algunos eventos del sistema.

Figura 23

Diagrama de flujo de la configuración de dependencias y scripts del proyecto en Nodejs



En particular, esta función también se ocupa de establecer las configuraciones para nodemon, utilizada como una ayuda para desarrolladores que permite iniciar el programa en modo de desarrollo y vuelve a cargarlo si se realiza algún cambio en el código. En un conjunto de scripts, describe que se ha creado el comando dev, start, que inician el programa ya sea para fines de producción o desarrollo, respectivamente.

Finalmente, el archivo también contiene información adicional sobre el proyecto, como el nombre, la versión, la URL del repositorio y la licencia, para facilitar su distribución

y control de versiones. Con base en estos detalles, el archivo package.json es decisivo para organizar y administrar las bibliotecas, módulos y controladores.

4.3 Fase 3: Pruebas de rendimiento

Esta sección se aborda el análisis de las pruebas realizadas en los cuatro lenguajes de programación utilizados para la generación de documentos PDF de forma paralela. Las pruebas realizadas por diferentes volúmenes de documentos, a saber, 1000, 3000 y 6000, y los tiempos de ejecución se midieron en algunas características clave que incluyen el inicio de sesión, la creación del servicio drive, la creación de la carpeta de destino, y la generación de los certificados.

4.3.1 Pruebas de rendimiento del lenguaje Golang

Las pruebas realizadas en Golang mostraron un excelente rendimiento general en todos los procesos. Para la prueba con 1000 documentos, el tiempo total de ejecución fue de 214972.02 ms, mientras que el tiempo de 1000 fue el más rápido entre los demás. Incluso el tiempo de inicio de sesión fue extremadamente bajo, es decir, 0.09 ms, lo que indica la eficiencia de Golang en la inicialización. El proceso de creación de la carpeta en Drive tomó 1286.71 ms y la generación de los certificados se completó en 212584.27 ms, lo que influyó en el tiempo total. Respecto al número de fallos, en esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU fue de 4.98%.

TABLA 2

Prueba de carga con 1000 documentos en Golang

Proceso	Tiempo (milisegundos)
1000 documentos	214972.02
Login	0.09
Construcción del servicio Drive	0
Creación de la carpeta Drive	1286.71
Generación de certificados	212584.27

TABLA 3

Margen de error con 1000 documentos en Golang

Archivos	Margen de error
1000	0.0%

TABLA 4

Uso del CPU con 1000 documentos en Golang

Archivos	Consumo CPU
1000	4.98%

Para 3000, la prueba de los documentos tuvo un total de 636791.45 ms, ya que era de esperarse debido al aumento de los documentos. La creación de la carpeta en Drive fue de 1424.75 ms y la generación de los certificados fue de 632593.01 ms. Respecto al número de fallos, en esta prueba, de los 3000 no se procesó correctamente 1 archivo. El uso del CPU promedio fue de 5.13 %.

TABLA 5

Prueba de carga con 3000 documentos en Golang

Proceso	Tiempo (milisegundos)
3000 documentos	636791.45
Login	0.1
Construcción del servicio Drive	0
Creación de la carpeta Drive	1424.75
Generación de certificados	632593.01

TABLA 6

Margen de Error con 3000 documentos en Golang

Archivos	Margen de error
3000	0.03%

TABLA 7

Uso del CPU con 3000 documentos en Golang

Archivos	Consumo CPU
3000	5.13 %

Para 6000 documentos, Golang el tiempo total de 1272590.84 ms. Aunque el tiempo de ejecución aumentó, Golang se aseguró de mostrar constantemente un rendimiento rápido con tiempos de 0.10 ms para el inicio de sesión y 1259.62 ms para crear la carpeta.

Respecto al número de fallos, en esta prueba, de los 6000 no se procesaron correctamente 7 archivos. El uso del CPU promedio fue de 5.21%.

TABLA 8

Prueba de carga con 6000 documentos en Golang

Proceso	Tiempo (milisegundos)
6000 documentos	1272590.8
Login	0.1
Construcción del servicio Drive	0
Creación de la carpeta Drive	1259.62
Generación de certificados	1265711.5

TABLA 9

Margen de Error con 6000 documentos en Golang

Archivos	Margen de error
6000	0.12%

TABLA 10

Uso del CPU con 6000 documentos en Golang

Archivos	Consumo CPU
6000	5.21%

Para analizar el rendimiento general del lenguaje Golang, en la tabla se puede apreciar el resumen de los resultados de las pruebas realizadas que incluye tiempo de procesamiento, convertido en minutos para una mejor comprensión, consumo de CPU y Margen de fallos (ver tabla 11).

TABLA 11

Resultados generales de las pruebas en Golang

Archivos	Tiempo (min)	Consumo CPU	Margen de Error
1000	3.54	4.98%	0.00%
3000	10.54	5.13%	0.03%
6000	21.09	5.21%	0.12%

4.3.2 Pruebas de rendimiento del lenguaje Node.js

En general, el rendimiento de Node.js fue en general bueno, pero no tanto como Golang. Al principio, con 1000 documentos, se obtiene un tiempo total de 282322.00ms. En lo que involucra el login, y la creación del servicio Drive, fue casi instantáneo con 0.00 ms para ambos. Luego el crear la carpeta implicó 914.00 ms y la generación de certificados fue

de 281408.00ms. Respecto al número de fallos, en esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU promedio fue de 8.30%.

TABLA 12

Prueba de carga con 1000 documentos en Node.js

Proceso	*Tiempo (milisegundos)*
1000 documentos	**282322**
Login	0
Construcción del servicio Drive	0
Creación de la carpeta Drive	914
Generación de certificados (paralelo)	281408

TABLA 13

Margen de error con 1000 documentos en Node.js

Archivos	Margen de error
1000	0.0%

TABLA 14

Uso del CPU con 1000 documentos en Node.js

Archivos	Consumo CPU
1000	8.30%.

Preliminarmente, con 3000 documentos, se incrementó a 812977.00 ms, la creación de la carpeta a 1262.00 y la generación de certificados a 811713.00 ms. A medida que el número de documentos aumentaba, el rendimiento también se veía afectado, aunque no de forma exponencial. Respecto al número de fallos, en esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU promedio fue de 10.41%.

TABLA 15

Prueba de carga con 3000 documentos en Node.js

Proceso	Tiempo (milisegundos)
3000 documentos	812977
Login	0
Construcción del servicio Drive	0
Creación de la carpeta Drive	1262
Generación de certificados (paralelo)	811713

TABLA 16

Margen de error con 3000 documentos en Node.js

Archivos	Margen de error
3000	0.0%

TABLA 17

Uso del CPU con 3000 documentos en Node.js

Archivos	Consumo CPU
3000	10.41%

Con 6000 documentos, el tiempo total es de 1711342.00 ms. Aquí se puede ver el sujeto de la capacidad de procesar un número mayor a Go, pero con resultados levemente más pobres, cada una de las operaciones subió. El tiempo de carpeta a 1214.00 ms y generación de certificados se elevó a 1710128.00 ms. Respecto al número de fallos, de los 6000 no se procesaron correctamente 4 archivos. El uso del CPU promedio fue de 12.85%.

TABLA 18

Prueba de carga con 6000 documentos en Node.js

Proceso	Tiempo (milisegundos)
6000 documentos	1711342
Login	0
Construcción del servicio Drive	0
Creación de la carpeta Drive	1214
Generación de certificados (paralelo)	1710128

TABLA 19

Margen de error con 6000 documentos en Node.js

Archivos	Margen de error
6000	0.06%

TABLA 20

Uso del CPU con 6000 documentos en Node.js

Archivos	Consumo CPU
6000	12.85%

Para analizar el rendimiento general del lenguaje Node.js, en la tabla se puede apreciar el resumen de los resultados de las pruebas realizadas que incluye tiempo de procesamiento, convertido en minutos para una mejor comprensión, consumo de CPU y Margen de fallos (ver tabla 21).

TABLA 21

Resultados generales de las pruebas en Node.js

Archivos	Tiempo (min)	Consumo CPU	Margen de Error
1000	4.69	8.30%	0.00%
3000	13.52	10.41%	0.03%

6000	28.5	12.85%	0.12%

4.3.3 Pruebas de rendimiento del lenguaje F#

En el caso de F#, el rendimiento era intermedio: mucho mejor que Python, pero peor que Golang y Node.js. Con la prueba de 1000 documentos, el tiempo total de ejecución igual a 341355.18 ms, el tiempo del login fue de 3.45 ms y la creación de carpeta en Drive se realiza en 856.97 ms. La etapa más lenta en crear los certificados igual a 340493.54 ms. Respecto al número de fallos, en esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU promedio fue de 5.83%.

TABLA 22

Prueba de carga con 1000 documentos en F#

Proceso	Tiempo (milisegundos)
1000 documentos	341355.18
Login	3.45
Construcción servicio Drive	1.21
Creación de la carpeta en Drive	856.97
Generación de certificados	340493.54

TABLA 23

Margen de error con 1000 documentos en F#

Archivos	Margen de error
1000	0.0%

TABLA 24

Uso del CPU con 1000 documentos en F#

Archivos	Consumo CPU
1000	5.83%

Con la prueba de 3000 documentos, el tiempo total de ejecución aumenta hasta 1019033.70 ms, y el tiempo de creación de la carpeta subida hasta 819.87 ms, mientras la generación toma 1018209.92 ms. A medida que el número de documentos aumenta, el rendimiento de F# también disminuye, pero no tan drástico como con el ejemplo de Python. En esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU promedio fue de 5.94%.

TABLA 25

Prueba de carga con 3000 documentos en F#

Proceso	Tiempo (milisegundos)
3000 documentos	1019033.7
Login	2.85
Construcción servicio Drive	1.05
Creación de la carpeta en Drive	819.87
Generación de certificados	1018209.92

TABLA 26

Margen de error con 3000 documentos en F#

Archivos	Margen de error
3000	0.0%

TABLA 27

Uso del CPU con 3000 documentos en F#

Archivos	Consumo CPU
3000	5.94%

Con la prueba de 6000 documentos, el tiempo total de ejecución igual a 2193278.27 ms, el tiempo mayor del login marcó con 22.75 ms, la generación de certificados fue igual

a 2192308.26 ms. Sorpresivamente, se procesaron todos los documentos sin ningún fallo. El uso del CPU promedio fue de 6.07%.

TABLA 28

Prueba de carga con 6000 documentos en F#

Proceso	Tiempo (milisegundos)
6000 documentos	2193278.27
Login	22.75
Construcción servicio Drive	1.5
Creación de la carpeta en Drive	945.75
Generación de certificados	2192308.26

TABLA 29

Margen de error con 6000 documentos en F#

Archivos	Margen de error
6000	0.0%

TABLA 30

Uso del CPU con 6000 documentos en F#

Archivos	Consumo CPU
6000	6.07%

Para analizar el rendimiento general del lenguaje F#, en la tabla se puede apreciar el resumen de los resultados de las pruebas realizadas que incluye tiempo de procesamiento, convertido en minutos para una mejor comprensión, consumo de CPU y Margen de fallos (ver tabla 31).

TABLA 31

Resultados generales de las pruebas en F#

Archivos	Tiempo (min)	Consumo CPU	Margen de Error
1000	5.67	5.83%	0.00%
3000	16.97	5.94%	0.03%
6000	36.53	6.07%	0.12%

4.3.4 Pruebas de rendimiento del lenguaje Python

Finalmente, Python fue el peor en las pruebas. Por 1000 documentos, el tiempo total de ejecución fue de 379073.47 ms, el tiempo de creación de la carpeta del usuario fue de 119.98 ms y para crear la carpeta, de 969.73 ms. De esta forma, para crear un documento PDF, se necesitaron 377980.75 ms, lo que expandió significativamente el tiempo total. Respecto al número de fallos, en esta prueba se procesaron todos los documentos correctamente sin ningún fallo. El uso del CPU fue de 15.65%.

TABLA 32

Prueba de carga con 1000 documentos en Python

Proceso	*Tiempo (milisegundos)*
1000 documentos	379073.47
Login	119.98
Construcción servicio Drive	2.99
Creación carpeta destino	969.73
Generación de PDF paralelo	377980.75

TABLA 33

Margen de error con 1000 documentos en Python

Archivos	Margen de error
1000	0.0%

TABLA 34

Uso del CPU con 1000 documentos en Python

Archivos	Consumo CPU
1000	15.65%

Para 3000 documentos, el tiempo total fue de 925647.91 ms, lo que provocó un aumento significativo y malos resultados en comparación con la cantidad de 1000 documentos. Sin embargo, los tiempos de login y creación de la carpeta del usuario fueron bajos a moderados, mientras que los de la creación de un PDF estuvieron en un nivel extremadamente alto. Respecto al número de fallos, en esta prueba, de los 3000 no se procesaron correctamente 3 archivos. El uso del CPU promedio fue de 17.10%.

TABLA 35

Prueba de carga con 3000 documentos en Python

Proceso	Tiempo (milisegundos)
3000 documentos	925647.91
Login	99.49
Construcción servicio Drive	2.94
Creación carpeta destino	1061.13
Generación de PDF paralelo	924484.35

TABLA 36

Margen de error con 3000 documentos en Python

Archivos	Margen de error
3000	0.1%

TABLA 37

Uso del CPU con 3000 documentos en Python

Archivos	Consumo CPU
3000	17.10%

Mientras tanto, la generación de PDF para 6000 documentos expandió el tiempo total hasta 1912470.80 ms, mostrando que Python no puede ser rápido y eficiente en grandes volúmenes. Finalmente, 1911344.52 ms es tiempo que duro la creación del PDF. Respecto al número de fallos, en esta prueba, de los 6000 no se procesaron correctamente 13 archivos superando la cantidad de fallos respecto a los demás lenguajes. El uso del CPU fue de 18.77%.

TABLA 38

Prueba de carga con 6000 documentos en Python

Proceso	Tiempo (milisegundos)
6000 documentos	1912470.8
Login	98
Construcción servicio Drive	3.06
Creación carpeta destino	1025.21
Generación de PDF paralelo	1911344.52

TABLA 39

Margen de error con 6000 documentos en Python

Archivos	Margen de error
6000	0.22%

TABLA 40

Uso del CPU con 6000 documentos en Python

Archivos	Consumo CPU
6000	18.77%

Para analizar el rendimiento general del lenguaje Python, en la tabla se puede apreciar el resumen de los resultados de las pruebas realizadas que incluye tiempo de procesamiento, convertido en minutos para una mejor comprensión, consumo de CPU y Margen de fallos (ver tabla 41).

TABLA 41

Resultados generales de las pruebas en Python

Archivos	Tiempo (min)	Consumo CPU	Margen de Error
1000	6.29	15.65%	0.00%
3000	15.4	17.10%	0.03%
6000	31.86	18.77%	0.12%

4.4 Fase 4: Recolección de datos de Encuestas

Para evaluar la eficiencia y percepción del sistema propuesto, se realizaron dos encuestas cuantitativas y una encuesta cualitativa realizadas con Google Formularios. Las encuestas están orientadas a medir diferentes aspectos clave del proyecto. Estas herramientas permiten analizar tanto la percepción numérica como la apreciación descriptiva de los actores involucrados.

Dado que el proyecto involucra el procesamiento de información, la empresa aplicó restricciones en el uso de datos y en la participación de personal. Por este motivo, y en cumplimiento con la Ley de Protección de Datos, las encuestas fueron dirigidas únicamente a las dos personas involucradas directamente en el desarrollo de la investigación del presente trabajo: El jefe de Transformación Digital y el Técnico de Desarrollo y Tecnología.

Las encuestas cuantitativas estuvieron enfocadas en:

Encuesta de satisfacción del usuario: Evaluar qué tan satisfechos están los actores con el sistema propuesto en términos de su funcionamiento, desempeño y utilidad.

El nivel de expectativa del usuario o las expectativas previas a la implementación del sistema, especialmente después de conocer las capacidades y beneficios del procesamiento paralelo con diferentes lenguajes de programación

La encuesta cualitativa se enfocó en evaluar el nivel de impacto en la productividad empresarial. Para ello, los participantes completaron la encuesta después de visualizar el rendimiento de los diferentes lenguajes de programación utilizados en el procesamiento paralelo. El objetivo de esta encuesta es recopilar las opiniones y percepciones de los usuarios sobre cómo la solución propuesta podría generar un impacto en la productividad de la empresa de soluciones medioambientales.

4.4.1 Encuesta de satisfacción del usuario

A continuación, se presenta el análisis detallado de las respuestas proporcionadas por los dos usuarios involucrados en el proceso de evaluación de los módulos de conversión con procesamiento paralelo.

Pregunta 1: Indique su cargo actual:

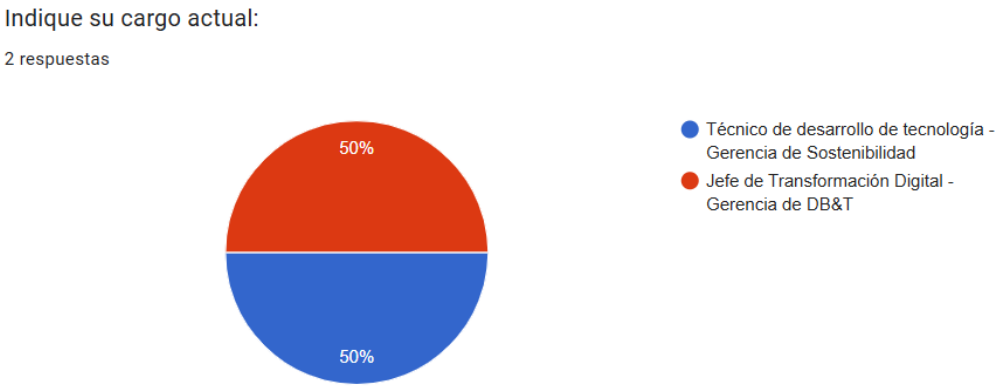
TABLA 42

Pregunta 1 - Encuesta de satisfacción del usuario

Indique su cargo actual:
Técnico de desarrollo de tecnología - Gerencia de Sostenibilidad
Jefe de Transformación Digital - Gerencia de DB&T

Figura 24

Pregunta 1 - Encuesta de satisfacción del usuario



Pregunta 2: ¿Cuál es su nivel de experiencia en sistemas de procesamiento paralelo?

TABLA 43

Pregunta 2 - Encuesta de satisfacción del usuario

¿Cuál es su nivel de experiencia en sistemas de procesamiento paralelo?
Principiante
Intermedio

Figura 25

Pregunta 2 - Encuesta de satisfacción del usuario

Análisis

Ambos participantes poseen conocimientos relevantes para la evaluación del sistema; sin embargo, indicaron un nivel de experiencia intermedio y principiante en sistemas de procesamiento paralelo. Esto sugiere que la percepción de los módulos está influenciada no solo por su rendimiento, sino también por la facilidad de uso y comprensión.

Pregunta 3: Estoy satisfecho con el rendimiento general de los módulos de conversión.

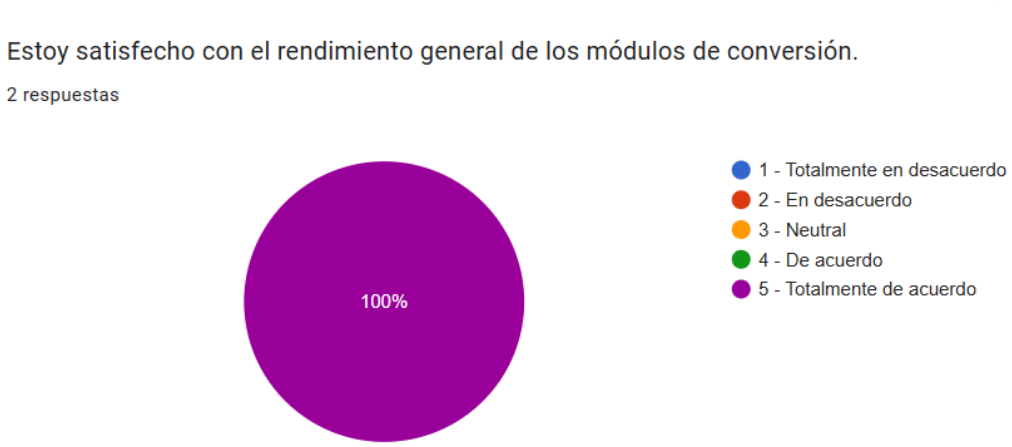
TABLA 44

Pregunta 3 - Encuesta de satisfacción del usuario

Estoy satisfecho con el rendimiento general de los módulos de conversión.
5 - Totalmente de acuerdo
5 - Totalmente de acuerdo

Figura 26

Pregunta 3 - Encuesta de satisfacción del usuario



Análisis

Ambos usuarios expresaron un alto nivel de satisfacción con el rendimiento general del sistema. Esto indica que los módulos cumplen con las expectativas iniciales, logrando un desempeño eficiente en la conversión de archivo.

Pregunta 4: Los módulos de conversión son fáciles de utilizar.

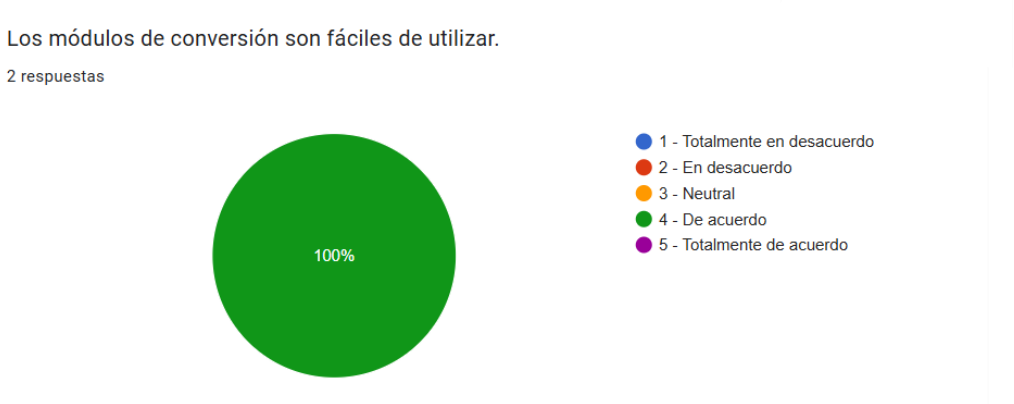
TABLA 45

Pregunta 4 - Encuesta de satisfacción del usuario

	Los módulos de conversión son fáciles de utilizar.
	4 - De acuerdo
	4 - De acuerdo

Figura 27

Pregunta 4 - Encuesta de satisfacción del usuario



Análisis

Los usuarios consideran que la usabilidad de los módulos es buena, aunque existe margen de mejora para optimizar aún más la experiencia de usuario. El resultado es positivo y sugiere que los módulos tienen una curva de aprendizaje baja.

Pregunta 5: La velocidad de conversión de los archivos es adecuada.

TABLA 46

Pregunta 5 - Encuesta de satisfacción del usuario

La velocidad de conversión de los archivos es adecuada.
5 - Totalmente de acuerdo
5 - Totalmente de acuerdo

Figura 28

Pregunta 5 - Encuesta de satisfacción del usuario

Análisis

Ambos usuarios están plenamente satisfechos con la velocidad de conversión proporcionada por los módulos. Esto valida la eficiencia del procesamiento paralelo en la mejora de los tiempos de ejecución.

Pregunta 6: He notado mejoras significativas en el tiempo de procesamiento al utilizar el procesamiento paralelo.

TABLA 47

Pregunta 6 - Encuesta de satisfacción del usuario

He notado mejoras significativas en el tiempo de procesamiento al utilizar el procesamiento paralelo.
5 - Totalmente de acuerdo
5 - Totalmente de acuerdo

Figura 29

Pregunta 6 - Encuesta de satisfacción del usuario

Análisis

Los participantes perciben mejoras claras en los tiempos de procesamiento gracias a la implementación del procesamiento paralelo. Esto refuerza el éxito de la evaluación comparativa realizada entre los lenguajes de programación.

Pregunta 7: La conversión de Hojas de Cálculo a PDF es precisa y mantiene la integridad de los datos

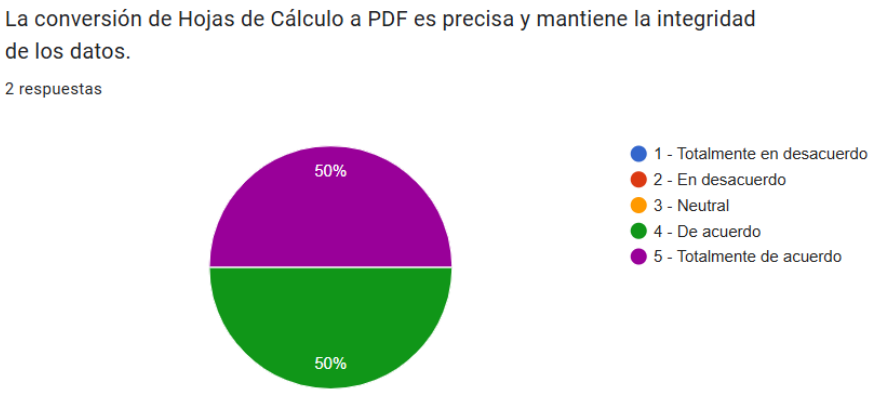
TABLA 48

Pregunta 7 - Encuesta de satisfacción del usuario

La conversión de Hojas de Cálculo a PDF es precisa y mantiene la integridad de los datos.
4 - De acuerdo
5 - Totalmente de acuerdo

Figura 30

Pregunta 7 - Encuesta de satisfacción del usuario



Análisis

Existe consenso en que la conversión es precisa y confiable, aunque uno de los usuarios señaló que aún hay espacio para garantizar la total integridad de los datos. Esto puede relacionarse con casos específicos de conversión bajo cargas elevadas.

Pregunta 8: Los módulos funcionan sin errores o fallos significativos.

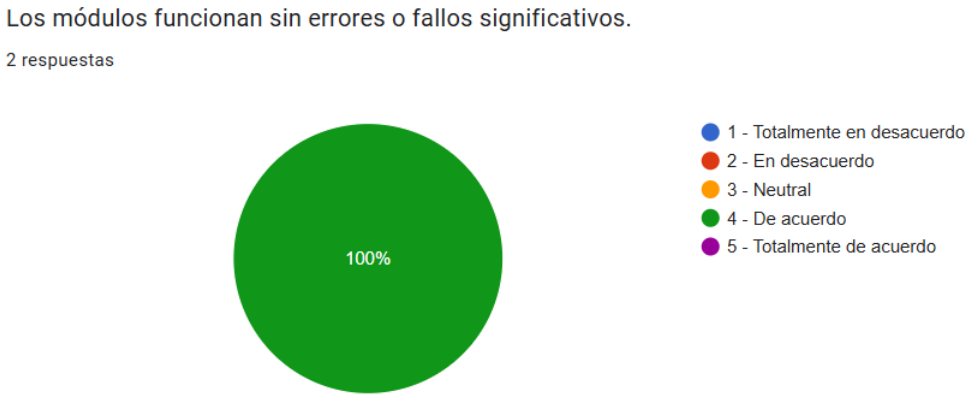
TABLA 49

Pregunta 8 - Encuesta de satisfacción del usuario

Los módulos funcionan sin errores o fallos significativos.
4 - De acuerdo
4 - De acuerdo

Figura 31

Pregunta 8 - Encuesta de satisfacción del usuario



Análisis

Si bien los usuarios reconocen que los módulos son funcionales, ambos coincidieron en que existen pequeños errores o fallos. Este resultado sugiere la necesidad de realizar ajustes adicionales para mejorar la estabilidad en escenarios extremos de procesamiento**.**

Pregunta 9: Los módulos cumplen con todas las necesidades de conversión de Hojas de Cálculo a PDF.

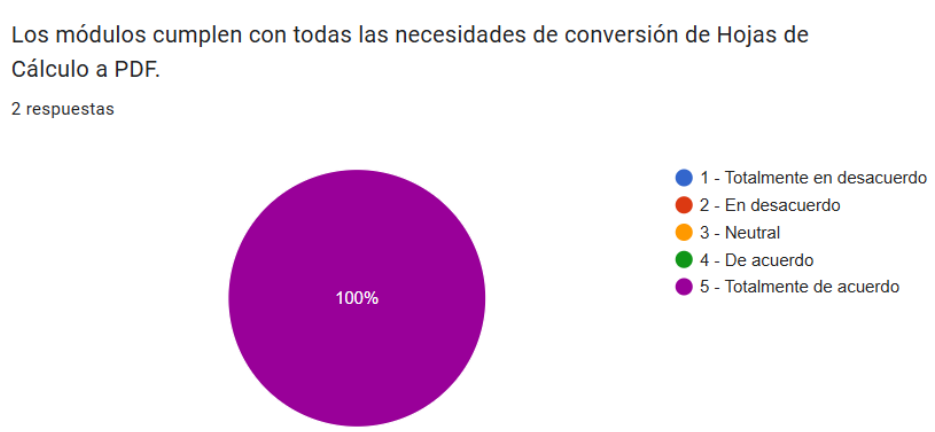
TABLA 50

Pregunta 9 - Encuesta de satisfacción del usuario

Los módulos cumplen con todas las necesidades de conversión de Hojas de Cálculo a PDF.
5 - Totalmente de acuerdo
5 - Totalmente de acuerdo

Figura 32

Pregunta 9 - Encuesta de satisfacción del usuario



Análisis

Los usuarios están totalmente satisfechos con la cobertura de necesidades ofrecida por los módulos, lo que indica que la solución evaluada aborda correctamente los requerimientos de conversión del proceso.

Pregunta 10: Recomendaría este módulo de conversión con procesamiento paralelo a otros profesionales o empresas.

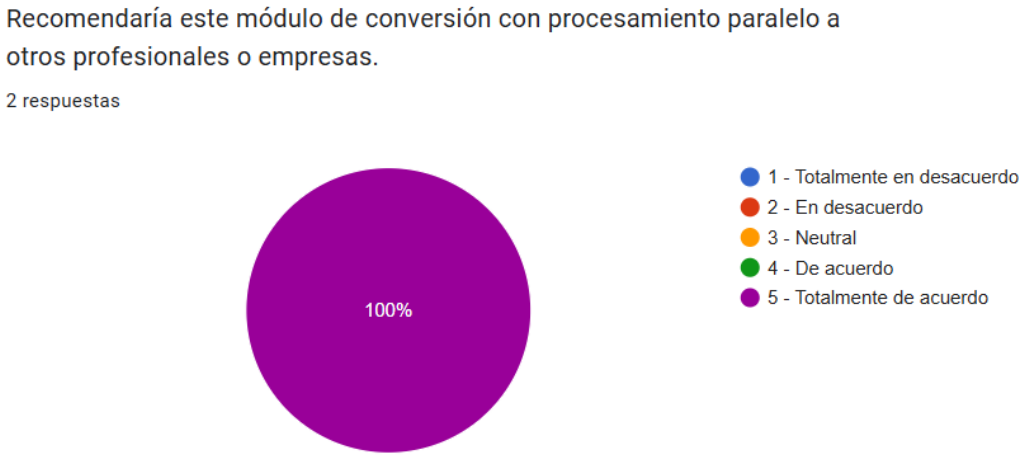
TABLA 51

Pregunta 10 - Encuesta de satisfacción del usuario

Recomendaría este módulo de conversión con procesamiento paralelo a otros profesionales o empresas.
5 - Totalmente de acuerdo
5 - Totalmente de acuerdo

Figura 33

Pregunta 10 - Encuesta de satisfacción del usuario



Análisis

Ambos usuarios recomendarían el módulo a otros profesionales o empresas, lo que valida su percepción positiva sobre la utilidad y eficiencia del sistema evaluado.

Conclusión

Las respuestas a la encuesta de satisfacción reflejan un alto nivel de aceptación y percepción positiva de los módulos evaluados. Los puntos clave que se destacan son:

El rendimiento general y la velocidad de conversión cumplen con las expectativas de los usuarios.

Los usuarios han notado mejoras significativas en los tiempos de procesamiento gracias al uso de procesamiento paralelo.

Aunque la usabilidad y la estabilidad son valoradas positivamente, existen observaciones sobre pequeños errores que deben ser atendidos para perfeccionar la solución.

4.4.2 Encuesta del Impacto en la Productividad Empresarial

A continuación, se presenta el análisis detallado de las respuestas proporcionadas por el **jefe de Transformación Digital**. La decisión de enfocarse en esta persona se debe a su amplio conocimiento del negocio y su rol clave en la gestión de las áreas operativas y tecnológicas de la empresa. Esta encuesta permite obtener una perspectiva fundamentada sobre cómo la solución propuesta, basada en el procesamiento paralelo, podría influir en la eficiencia y productividad de la organización.

Pregunta 1: Basándose en la evaluación de los lenguajes de programación presentados, ¿cómo cree que podrían impactar en su trabajo diario?

Respuesta: La evaluación de los lenguajes demuestra una mejora significativa en los tiempos de procesamiento, lo cual optimizaría las tareas diarias del usuario final en la generación de reportes.

Análisis

El jefe de Transformación Digital destaca la mejora en los tiempos de procesamiento como un factor clave que optimizaría las tareas diarias relacionadas con la generación de reportes. Esto refleja que los lenguajes evaluados tienen un impacto directo en la eficiencia

operativa, reduciendo el tiempo invertido en procesos repetitivos y permitiendo un mejor aprovechamiento de los recursos**.**

Pregunta 2: ¿Qué expectativas tiene sobre los tiempos de procesamiento de datos con los lenguajes evaluados?

Respuesta: Mis expectativas son altas, ya que los lenguajes evaluados han demostrado tiempos de procesamiento muy rápidos. Esto permitirá manejar grandes volúmenes de datos en menos tiempo**.**

Análisis: El entrevistado tiene altas expectativas debido al rendimiento observado durante las pruebas. La capacidad de procesar grandes volúmenes de datos en menos tiempo representa un avance significativo para la empresa, lo que contribuiría a la entrega oportuna de reportes y análisis.

Pregunta 3: ¿Cómo cree que la elección del lenguaje de programación podría afectar la calidad de los informes y análisis medioambientales que produce la empresa?

Respuesta: Un lenguaje eficiente asegura que los datos se procesen de manera precisa y rápida, minimizando errores en la conversión a PDF y garantizando la integridad de la información.

Análisis: La respuesta subraya la importancia de un lenguaje eficiente para garantizar la precisión y rapidez en el procesamiento de datos. Esto minimiza errores en la conversión de hojas de cálculo a PDF y asegura la integridad de la información, elementos cruciales para la calidad y confiabilidad de los informes generados.

Pregunta 4: ¿Considera que alguno de los lenguajes evaluados podría mejorar la capacidad de la empresa para manejar proyectos más grandes o complejos? Si es así, ¿puede describir cómo?

Respuesta: Sí, considero que lenguajes como Golang y Nodejs podrían mejorar nuestra capacidad para manejar proyectos más grandes o complejos. Golang, por su diseño

enfocado en el procesamiento paralelo y su alto rendimiento y Nodejs al ser unos lenguajes para desarrollo web nos permitiría complementar soluciones mejores.

Análisis

Se identifican Golang y Node.js como lenguajes clave para mejorar la capacidad de la empresa. Golang destaca por su rendimiento en procesamiento paralelo, lo que facilita el manejo de grandes volúmenes de datos, mientras que Node.js, por su enfoque en desarrollo web, complementa la creación de soluciones integradas. Esto resalta el potencial de estos lenguajes para enfrentar proyectos más exigentes.

Pregunta 5: ¿De qué manera cree que la elección del lenguaje de programación podría impactar en la colaboración entre diferentes departamentos o equipos dentro de la empresa?

Respuesta: La elección del lenguaje de programación podría mejorar la colaboración entre departamentos al facilitar la integración de herramientas y procesos. Lenguajes como Node.js y Python, por su versatilidad y uso extendido, permiten desarrollar soluciones que se adaptan a diferentes necesidades y plataformas.

Análisis

El entrevistado menciona que lenguajes como **Node.js** y **Python**, debido a su versatilidad y uso extendido, podrían facilitar la integración de herramientas y procesos entre departamentos. Esto permitiría una colaboración más eficiente, eliminando problemas de compatibilidad y mejorando la comunicación y coordinación entre equipos.

La encuesta confirma que la evaluación de lenguajes de programación tiene un impacto significativo en la productividad empresarial. El jefe de transformación digital percibe mejoras en eficiencia, calidad y escalabilidad, lo que contribuiría a optimizar las operaciones y preparar a la empresa para abordar desafíos más complejos en el futuro

4.5 Fase 5: Interpretación de los resultados

4.5.1 Comparación de rendimiento de los lenguajes de programación

A continuación, se presenta un análisis de los resultados obtenidos para el procesamiento de 1000 archivos utilizando los lenguajes de programación Golang, Node.js, Python y F#. Los parámetros evaluados incluyen el tiempo de procesamiento (en minutos), consumo de CPU y el margen de error.

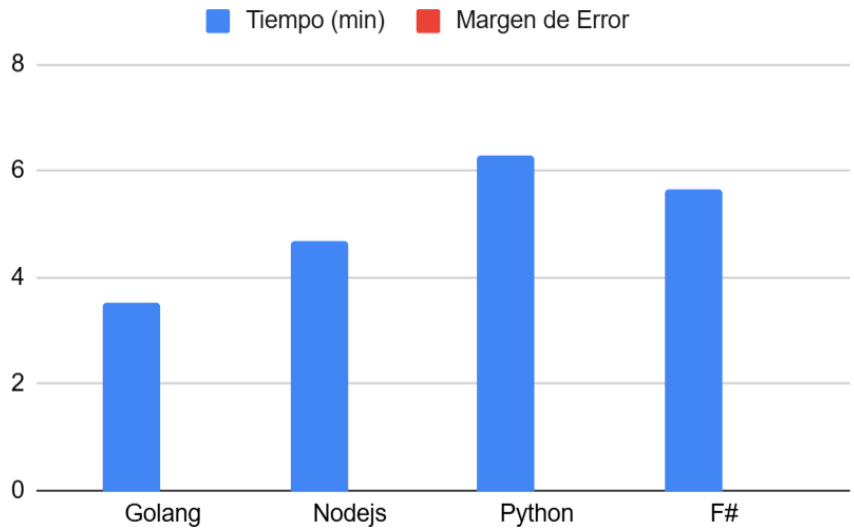
En las pruebas de rendimiento de 1000 archivos (ver figura 34), el margen de error fue del **0.00%** en todos los lenguajes, lo que indica que no se presentaron fallos en la conversión. Se observa que Golang es el lenguaje con el mejor desempeño en términos de tiempo de procesamiento, completando la tarea en 3.54 minutos con el uso del CPU 4.98%. Esto refleja la ventaja de Golang en el manejo de procesamiento paralelo, gracias a su diseño eficiente y su modelo de concurrencia basado en goroutines.

Node.js tuvo un tiempo de 4.69 minutos con uso de CPU 8.30%, posicionándose como la segunda opción más rápida, aunque su naturaleza orientada a aplicaciones web es eficiente, su desempeño en tareas puramente computacionales es inferior a Golang. Con F# el tiempo de procesamiento fue de 5.67 minutos, lo que lo sitúa en el tercer lugar.

A pesar de ser un lenguaje funcional con capacidades de procesamiento paralelo, su rendimiento no fue tan eficiente como Golang o Node.js. Por último, Python, que tuvo el tiempo más alto con 6.29 minutos. Esto se debe a que, aunque Python es un lenguaje versátil y ampliamente utilizado, no está optimizado para tareas de procesamiento paralelo intensivo en comparación con los otros lenguajes evaluados.

Figura 34

Comparación de los lenguajes de programación en las pruebas de 1000 archivos



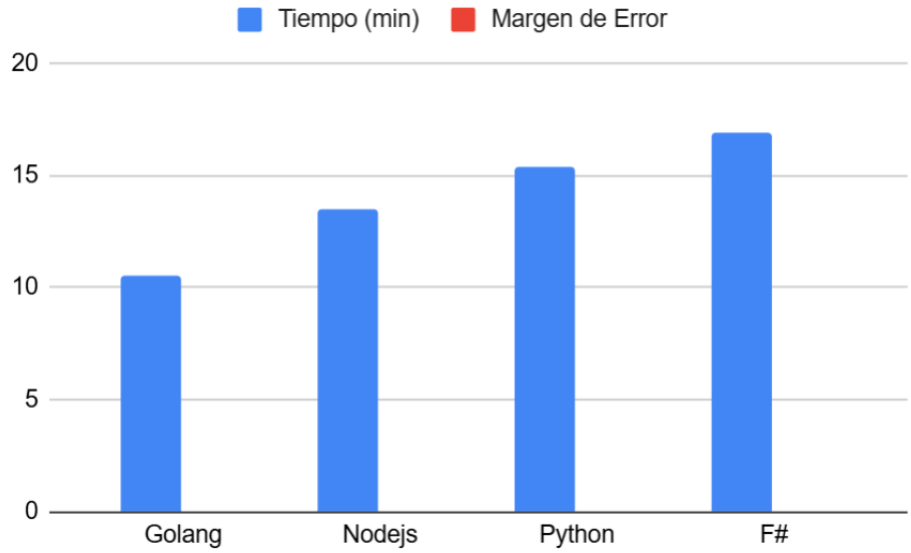
Para el procesamiento de 3000 archivos, el tiempo de procesamiento de Golang fue de 10.54 minutos, manteniendo su posición como el lenguaje más eficiente. Aunque aparece un ligero 0.03% de margen de error, es insignificante en comparación con su desempeño en tiempo y consumo de CPU, siendo 5.13%. Node.js completó la tarea en 13.52 minutos, posicionándose en segundo lugar. Se mantuvo en 0.00% con un consumo de 10.41% de CPU lo que indica estabilidad y precisión en la generación de archivos.

Por su parte, Python, mostró un desempeño menos eficiente con 15.40 minutos, ubicándose en el tercer lugar. Presentó un 0.10% de margen de error y uso de CPU de 17.10% el valor más alto entre los lenguajes, lo que podría indicar limitaciones en la estabilidad al procesar grandes volúmenes de archivos.

F# registró el tiempo más alto con 16.97 minutos, siendo el menos eficiente en esta prueba, a pesar de su tiempo más alto, F# logró un 0.00% de margen de error y uso de CPU de 5.94% lo que muestra fiabilidad en la conversión (ver figura 35).

Figura 35

Comparación de los lenguajes de programación en las pruebas de 3000 archivos



Por último, en las pruebas de procesamiento de 6000 archivos, el tiempo de Golang fue 21.09 minutos con uso de CPU de 5.21%, y se mantiene como el lenguaje más rápido, demostrando una alta eficiencia en tareas de gran escala. Aunque el margen de error subió a 0.12%, sigue siendo aceptable considerando la cantidad de archivos procesados.

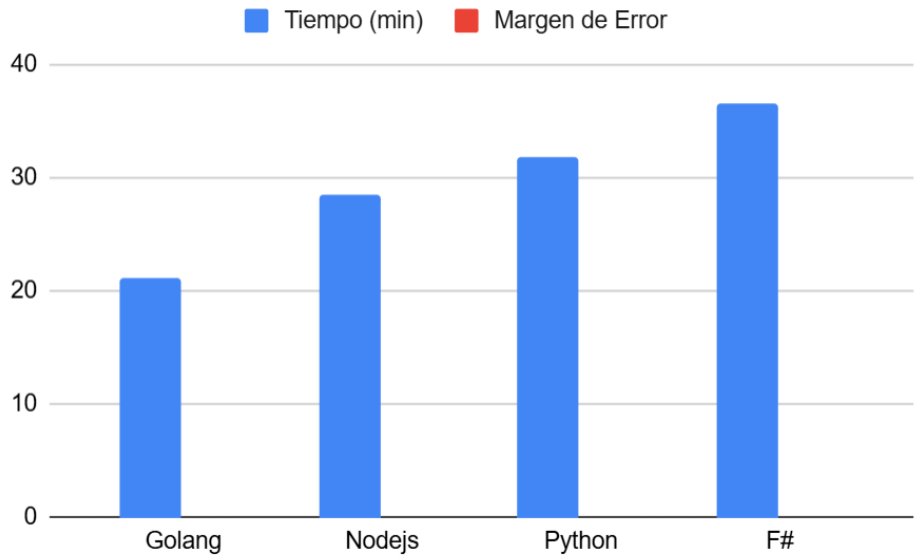
Node.js completó la tarea en 28.50 minutos, posicionándose en el segundo lugar. Con 0.06% de margen de error y uso de CPU de 12.85%, presenta una alta precisión y estabilidad en la conversión de archivos, mejor que Golang.

Python tuvo un desempeño más lento con 31.86 minutos, ubicándose en el tercer lugar. Presenta el margen de error y uso CPU más alto con 18.7%, lo que refleja una menor estabilidad y precisión en la conversión a medida que aumenta la cantidad de archivos.

Mientras que F#, registró el tiempo más alto con 36.53 minutos, siendo el lenguaje menos eficiente. A pesar de su tiempo elevado, logró un 0.00% de margen de error, destacándose en precisión y confiabilidad en la generación de archivos (ver figura 36).

Figura 36

Comparación de los lenguajes de programación en las pruebas de 6000 archivos.



4.5.2 Análisis de las encuestas realizadas

Para analizar los resultados de la encuesta de satisfacción del usuario, se consolidaron las respuestas de los dos participantes involucrados en el proceso: Técnico de Desarrollo de Tecnología y el jefe de Transformación Digital

Las respuestas fueron evaluadas en una escala de 1 a 5:

1: Totalmente en desacuerdo

2: En desacuerdo

3: Neutral

4: De acuerdo

5: Totalmente de acuerdo

Donde el Promedio General de Satisfacción fue de 4.69 sobre 5.0 puntos que refleja un alto nivel de satisfacción de los usuarios con los módulos de conversión (ver figura 37). A continuación, se detallan las áreas de mayor satisfacción:

- Rendimiento general y velocidad de conversión (promedio de 5.0).

- Cumplimiento de las necesidades y recomendación del módulo a otros (5.0).

Cómo área de mejora está el funcionamiento sin errores (promedio de 4.0).

Los resultados de la encuesta muestran que los módulos desarrollados cumplen con las expectativas de los usuarios en cuanto a rendimiento, velocidad y precisión, lo cual genera un alto nivel de satisfacción.

Figura 37

Métricas de la encuesta de satisfacción de usuario

Pregunta	Promedio
Estoy satisfecho con el rendimiento general de los módulos de conversión	5.0
Los módulos de conversión son fáciles de utilizar	4.0
La velocidad de conversión de los archivos es adecuada	5.0
He notado mejoras significativas en el tiempo de procesamiento	5.0
La conversión de Hojas de Cálculo a PDF es precisa y mantiene la integridad de los datos	4.5
Los módulos funcionan sin errores o fallos significativos	4.0
Los módulos cumplen con todas las necesidades de conversión	5.0
Recomendaría este módulo de conversión con procesamiento paralelo a otros profesionales o empresas	5.0

Con respecto a la encuesta del impacto en la productividad empresarial, se destaca que la implementación de procesamiento paralelo con los lenguajes evaluados generaría un impacto positivo significativo en la eficiencia de los procesos actuales de la empresa. Según el jefe de Transformación Digital, las mejoras en los tiempos de procesamiento permitirían reducir los cuellos de botella existentes, optimizando así la entrega de reportes trimestrales y facilitando una toma de decisiones más oportuna

Además, se resalta que el procesamiento paralelo aumentaría la capacidad de manejo de datos, permitiendo gestionar volúmenes de información más grandes sin afectar la productividad, lo cual sería fundamental para abordar proyectos de mayor complejidad en el futuro.

Conclusiones

La revisión bibliográfica permitió identificar aspectos clave del procesamiento paralelo y secuencial, así como de los lenguajes de programación. El procesamiento paralelo destaca por su capacidad para dividir tareas complejas en subtareas más pequeñas y ejecutarlas simultáneamente, logrando una optimización significativa en tiempos de ejecución.

Por otro lado, el procesamiento secuencial, aunque garantiza mayor consistencia y control en la ejecución, muestra limitaciones frente a las demandas actuales de alto rendimiento. En cuanto a los lenguajes de programación, los de bajo nivel se reconocen por su cercanía al hardware y su eficiencia en tareas específicas, mientras que los de alto nivel se distinguen por su facilidad de uso y adaptabilidad, cualidades que los convierten en herramientas esenciales para el desarrollo de aplicaciones modernas.

Estas características fundamentan la selección de lenguajes de alto nivel en este proyecto, especialmente por su compatibilidad con las necesidades de procesamiento paralelo. Con respecto a los criterios de evaluación, la definición se llevó a cabo con base en información obtenida de fuentes científicas confiables, como Scopus y Ebsco.

Los criterios seleccionados, como la velocidad de ejecución, el uso eficiente del CPU, el margen de fallos, la satisfacción del usuario y el impacto en la productividad empresarial, respondieron a la necesidad de contar con un enfoque práctico y balanceado para medir la eficiencia de los lenguajes de programación. Este proceso permitió establecer parámetros claros que orientaron el análisis y aseguraron resultados relevantes tanto desde una perspectiva técnica como empresarial, facilitando la identificación de oportunidades de mejora en los procesos evaluados.

Referente al análisis del proceso actual de generación de reportes trimestrales en formato PDF, se identificaron varias limitaciones que impactan la eficiencia operativa de la empresa. Entre ellas destacan el tiempo restringido de ejecución del sistema actual, las

interrupciones frecuentes al procesar grandes volúmenes de datos, la inconsistencia en los resultados generados y la dependencia de intervención manual para completar el proceso. Estas dificultades afectan directamente la productividad y limitan la capacidad del sistema para manejar mayores volúmenes de trabajo.

Con base en estos hallazgos, se concluye que es necesario implementar un sistema más robusto, capaz de optimizar el tiempo de procesamiento, reducir la intervención manual y garantizar la consistencia de los reportes generados. Esto no solo mejoraría la eficiencia del proceso, sino que también permitiría a la empresa adaptarse a mayores demandas, asegurando la calidad y oportunidad de la información entregada.

El desarrollo de los cuatro módulos en Python, F#, Golang y Node.js demuestra la viabilidad de integrar múltiples lenguajes de programación en un sistema distribuido y modular que aprovecha las fortalezas específicas de cada tecnología. El objetivo principal, que consistía en automatizar la conversión de hojas de cálculo de Google a PDF dentro del entorno de Google Drive, se cumplió con éxito, garantizando tanto la eficiencia del procesamiento como la robustez del sistema.

Cada lenguaje aportó características únicas que enriquecieron el desarrollo: Python destacó por su facilidad para gestionar flujos de trabajo paralelos y conexión con la API de Google, F# ofreció una estructura sólida y modular ideal para mantener un control detallado de excepciones y registros, Golang resaltó por su capacidad para optimizar procesos paralelos con un manejo eficiente de concurrencia, y Node.js facilitó la integración de servicios web con autenticación segura mediante OAuth2.

La organización modular permitió una clara separación de responsabilidades, facilitando la escalabilidad y el mantenimiento del sistema. Además, el uso de bibliotecas específicas en cada lenguaje optimizó las operaciones, asegurando un manejo eficiente de los recursos y el cumplimiento de las cuotas de la API de Google.

El análisis comparativo de los lenguajes de programación para la generación de archivos PDF revela que Golang es el más eficiente en términos de tiempo de procesamiento y consumo de CPU, superando a los demás lenguajes en todas las pruebas. A pesar de un pequeño aumento en el margen de error, su rendimiento sigue siendo el mejor, especialmente en tareas de procesamiento paralelo.

Node.js se posiciona en segundo lugar, con tiempos de procesamiento más altos que Golang, pero con un consumo de CPU también elevado. Sin embargo, se mantiene estable y preciso a lo largo de las pruebas. F# presenta un bajo consumo de recursos, pero su desempeño en tiempo no es competitivo frente a Golang o Node.js.

Por último, Python mostró los peores resultados en cuanto a tiempo de procesamiento y consumo de CPU, lo que refleja sus limitaciones para tareas intensivas de procesamiento paralelo. A pesar de su versatilidad, no es la mejor opción para este tipo de tareas.

En cuanto a los resultados de la encuesta de satisfacción reflejan un alto nivel de aceptación y conformidad por parte de los usuarios involucrados en el proyecto, con un promedio general de 4.69 sobre 5. Las áreas de mayor satisfacción incluyen el rendimiento general, la velocidad de conversión y la adecuación a las necesidades del usuario, obteniendo una puntuación perfecta de 5.0. Sin embargo, se identificó como área de mejora el funcionamiento sin errores, con un promedio de 4.0, lo que sugiere que aún hay margen para optimizar la estabilidad del sistema.

En cuanto al impacto en la productividad empresarial, la implementación de procesamiento paralelo se percibe como un cambio positivo, con mejoras significativas en la eficiencia operativa. La reducción de cuellos de botella en la entrega de reportes y el aumento de la capacidad de manejo de datos posicionan a la empresa en un nivel competitivo para abordar proyectos más complejos y de mayor escala.

Recomendaciones

Basándose en los resultados obtenidos en este estudio sobre la evaluación de lenguajes de programación para el proceso de generación de reportes trimestrales en formato PDF, se recomienda que, para optimizar el proceso de generación de reportes trimestrales en PDF, se implemente Golang como el lenguaje principal.

Se propone desarrollar un nuevo módulo en Golang específicamente para esta tarea, aprovechando sus capacidades de procesamiento paralelo. Este módulo debe diseñarse para manejar eficientemente la generación simultánea de múltiples reportes en formato PDF, utilizando las goroutines y canales de Golang para maximizar el rendimiento.

Para mejorar la eficiencia en el acceso y procesamiento de datos, se sugiere implementar un sistema de caché para los datos más frecuentemente utilizados en los reportes trimestrales. Esto reducirá el tiempo de acceso a los archivos y acelerará la generación de los PDFs, especialmente para aquellos reportes que comparten información común.

Para garantizar la calidad y consistencia de los reportes generados, se sugiere implementar un sistema automatizado de verificación y validación de los PDFs. Este sistema debe comprobar la integridad de los datos, la correcta aplicación de los formatos y la inclusión de todos los elementos requeridos en cada reporte trimestral.

Se recomienda fortalecer las capacidades técnicas de los desarrolladores, dado su papel esencial en la creación de sistemas de alto rendimiento y eficiencia, especialmente en entornos con alta demanda. La capacitación en tecnologías emergentes es igualmente prioritaria. Además, es fundamental que los desarrolladores comprendan los conceptos de programación paralela y concurrente en la nube, para garantizar implementaciones basadas en mejores prácticas en escenarios de cargas intensivas. Estas acciones son clave para garantizar la estabilidad y el rendimiento del sistema una vez implementado.

Referencias Bibliográficas

Alaidaros, H., Omar, M., & Romli, R. (2021). The state of the art of agile kanban method: Challenges and opportunities. *Independent Journal of Management & Production*, *12*(8), 2535-2550.

Alvarez Echeverría, F. A. (2015). *Implementación de nuevas tecnologías: Valuación, variables, riesgos y escenarios tecnológicos*.

Arenaza, R., & Erickson, R. (2019). *Lenguajes de programación Javascript Java y Javascript. Características. Norma de escritura. Variables y operadores lógicos. Mensajes. Ejercicios. Estructuras condicionales. Funciones y objetos. Aplicaciones*.

Asamblea Constituyente Del Ecuador. (2008). Constitución de la República del Ecuador. *Quito: Tribunal Constitucional del Ecuador. Registro oficial Nro*, *449*, 79-93.

Astera. (2023, mayo 9). *Procesamiento de datos*. https://www.astera.com/es/knowledge-center/what-is-data-processing-definition-and-stages/

Aziz, Z. A., Abdulqader, D. N., Sallow, A. B., & Omer, H. K. (2021). Python parallel processing and multiprocessing: A rivew. *Academic Journal of Nawroz University*, *10*(3), 345-354.

Bahit, E. (2021, marzo 31). *Arquitectura de Sistemas Informáticos*. ResearchGate. https://www.researchgate.net/publication/352049828_Arquitectura_de_Sistemas_Informaticos

Bermúdez León, M. J. (2020). *La computación en la nube, ventajas y retos*.

Brucker, A. D. (2022). Nano JSON: working with JSON formatted data in Isabelle/HOL and Isabelle/ML. *Archive of Formal Proofs*.

Carla, M. V., Alfonso, U. M., & Ángel, R. G. M. (2021). *Lenguajes de programación*. Editorial UNED.

Carrasco Fernández, R. & otros. (2022). *Abstracciones para el cálculo automático de comunicaciones en computación paralela distribuida*.

Carriel, R., & Galarza, J. (2022, junio 30). *Evolución de los sistemas de lenguaje de programación a lo largo de la historia.* https://revista.estudioidea.org/ojs/index.php/esci/article/download/237/321/511

Checasaca, J. R., Sánchez, L. K., & Malpartida, J. N. (2022). Importancia de la herramienta Customer Relationship Management (CRM) en las empresas de Latinoamérica. Una revisión sistemática de la literatura científica los últimos diez años. *Revista Científica de la UCSA*, *9*(3), 97-119. https://doi.org/10.18004/ucsa/2409-8752/2022.009.03.097

Chuquimarca, C. E. T., & Maita, S. S. (2022). Análisis comparativo entre arquitecturas de sistemas IoT. *Revista de Investigación en Tecnologías de la Información: RITI*, *10*(21), 55-70.

Constanzo, M. (2021). *Comparación de rendimiento y esfuerzo de programación entre Rust y C en arquitecturas multicore* [Tesis, Universidad Nacional de La Plata]. http://sedici.unlp.edu.ar/handle/10915/120119

Coro Arcayne, R. D. (2022). *Programación (Informática)—Qué es, información, lenguajes*. https://es.slideshare.net/slideshow/programacin-informtica-qu-es-informacin-lenguajespdf/251485923

Costanzo, M. (2021). *Comparación de rendimiento y esfuerzo de programación entre Rust y C en arquitecturas multicore* [PhD Thesis]. Universidad Nacional de La Plata.

Costanzo, M. (2024). *Estudio de viabilidad de SYCL como modelo de programación unificado para sistemas heterogéneos basados en GPUs en bioinformática* [Tesis, Universidad Nacional de La Plata]. https://doi.org/10.35537/10915/164928

Cuaran, S., & Miranda, M. Y. (2019). *Análisis sistemático de información para el estudio de la implementación de un laboratorio de Testing para los desarrollos de aplicativos*

móviles en Popayán en la Universidad Cooperativa de Colombia. https://hdl.handle.net/20.500.12494/7254

Cusi, J. A. (2023). *Aplicación web con chatbot inteligente en la gestión de cobranza en el Centro de idiomas de la Universidad Nacional José María Arguedas*. http://repositorio.unajma.edu.pe/handle/20.500.14168/820

DE LA SOCIEDAD, S. D. F. (2022). *POLÍTICA PARA LA TRANSFORMACIÓN DIGITAL DEL ECUADOR 2022-2025*. 68.

Díaz Acevedo, K., Ponce de León, A. G., Caymes Scutari, P. G., & Bianchini, G. (2021). *Cómputo paralelo para la resolución de la multiplicación de matrices de gran tamaño*. Universidad Tecnológica Nacional.

Djurica, D. (nd). Automatización de procesos para la eficiencia organizacional [Blog]. *La automatización de procesos como un instrumento valioso para mejorar la eficiencia, reducir los costos y aumentar la productividad*. https://www.boc-group.com/es/blog/bpm/automatizacion-de-procesos-su-camino-hacia-la-eficiencia-operativa/

Fajardo, H. M. (2023). *Estudio comparativo entre Apache Spark y Apache Flink en el procesamiento de streaming en entornos Big Data* [PhD Thesis]. Universidad Nacional de La Plata.

Fuentes, J., & Pérez, M. Á. (2022). *Análisis comparativo de la herramienta tablero en las metodologías ágiles Scrum, Kanban y Scrumban en proyectos de software*. http://dspace.aeipro.com/xmlui/handle/123456789/3286

García, J. J. (2024). *Aplicación Web Tablero Kanban Compartido*. https://ruc.udc.es/dspace/handle/2183/39559

García Moreno, E. (2020). *Automatización de procesos industriales: Robótica y automática*.

Gheorghe, A.-M., Gheorghe, I. D., & Iatan, I. L. (2020). Agile Software Development. *Informatica Economica*, *24*(2).

Gómez Forero, D. T., Gómez Prada, U. E., & Viola Villamizar, J. B. (2018). *Fundamentos de Programación*. http://hdl.handle.net/20.500.11912/4143

Herrera, D. A., Hamon, H. H., Prieto, C. A. F., & Oficina, C. (2021). *Preprocesamiento y procesamiento de imágenes Sentinel 2 con Google Earth Engine*.

Hosseini, M., & Sahragard, O. (2019). *Aws lambda language performance*.

Huachaca, C. J. (2023). *Desarrollo e implementación de un sistema de colas para mejorar la calidad de servicio en la plataforma de la municipalidad distrital de Cieneguilla*. http://repositorio.uncp.edu.pe/handle/20.500.12894/10112

Interagua, E. (2024). Durante 23 años, hemos tenido el honor de servir a los guayaquileños con un servicio de agua potable de calidad [Red Social]. *Publicaciones*. https://www.linkedin.com/company/interagua-ec/mycompany/

Ivančić, L., Vukšić, V. B., & Spremić, M. (2019). Mastering the digital transformation process: Business practices and lessons learned. *Technology Innovation Management Review*, *9*(2).

Izquierdo, V. (2020). *Análisis de rendimiento de lenguajes de desarrollo de aplicaciones y servicios web* [Universitat Politècnica de Catalunya]. https://upcommons.upc.edu/bitstream/handle/2117/407069/181292.pdf?sequence=2&isAllowed=y

Jiménez, J. A. F. (2020). La implementación de un sistema automatizado reduce los tiempos de atención en los procesos aplicables a la ventanilla única de turismo en la Municipalidad Provincial del Callao. *Industrial Data*, *23*(2), 31-37.

Ledesma, A. C. (2024). *Introducción al Cómputo en Paralelo*.

Ley Orgánica de Protección de Datos Personales, No. 0 (2021). https://www.finanzaspopulares.gob.ec/wp-content/uploads/2021/07/ley_organica_de_proteccion_de_datos_personales.pdf

Lu, Y., Xu, X., & Wang, L. (2020). Smart manufacturing process and system automation–a critical review of the standards and envisioned scenarios. *Journal of Manufacturing Systems*, *56*, 312-325.

Martínez, F., Aguilera, D., & González, J. (2020, diciembre 2). *Lenguajes de programación y desarrollo de competencias clave. Revisión sistemática | Revista Electrónica de Investigación Educativa*. https://redie.uabc.mx/redie/article/view/2869

Martini, N. F. (2020). *Implementación de un sitio web para un concurso de programación paralela* [PhD Thesis]. Universitat Politècnica de València.

Medina Chicaiza, P., Chango Guanoluisa, M., Corella Cobos, M., & Guizado Toscano, D. (2022). Transformación digital en las empresas: Una revisión conceptual. *Journal of Science and Research*, *7*(CININGEC II), 756-769.

Milla, A. (2022). *Un estudio comparativo entre traductores de Python para aplicaciones paralelas de memoria compartida* [PhD Thesis]. Universidad Nacional de La Plata.

Miranda, G. I. (2024). *Un gestor de tareas basado en Kanban para la planificación operativa: Un enfoque ágil para la gestión y entrega continua de un producto software : Implementación de las prácticas de DevOps para automatizar el desarrollo de una aplicación web*. http://bibdigital.epn.edu.ec/handle/15000/25885

Mite, G., Vargas, J., Franco, O., & Maliza, W. (2024). Uso de un Software Contable en el proceso de Enseñanza del Módulo de Paquete Contable. *593 Digital Publisher CEIT*, *9*(2), 916-940.

Montes, G. A. (2023). *Análisis comparativo entre los lenguajes de programación R y Python en cuanto a facilidad de uso, eficiencia y visualización en el análisis de datos.*

[bachelorThesis, Babahoyo: UTB-FAFI. 2023]. http://dspace.utb.edu.ec/handle/49000/15002

Moreira, D., & Cruz, I. (2021, febrero). *Análisis del Estado Actual de Procesamiento de Lenguaje Natural—ProQuest*. https://www.proquest.com/openview/a44d67c88cfaada2563dc16f94ccd3c8/1?pq-origsite=gscholar&cbl=1006393

Moreno Segura, G. J. (2022). *Programación paralela y distribuida. Arquitecturas paralelas y distribuidas. Programación de aplicaciones multiproceso*.

Narváez, E. A. (2021). *Las tecnologías de la información y comunicación orientadas a la calidad del servicio en la gestión empresarial: Una revisión sistemática* [bachelorThesis]. http://dspace.ups.edu.ec/handle/123456789/20929

Olarte Gervacio, L. (2018). Lenguaje de Programación. *Conogasi*. https://conogasi.org/articulos/lenguaje-de-programacion/

Peñafiel, M. (2022). *Definición y automatización de pruebas de carga y escalabilidad para una aplicación web colaborativa*. https://uvadoc.uva.es/handle/10324/57349

Peña-Maciel, D., Parra-Guevara, D., & Skiba, Y. N. (2022). Formulación de una estrategia para el control puntual de un contaminante y su implementación usando cómputo paralelo. *Información tecnológica*, *33*(1), 35-48.

Pereyra, P., & Rosario, R. (2021). Desarrollo e implementación de un analizador sintáctico utilizando el compilador Javacc para el reconocimiento de errores sintácticos en el lenguaje PHP. *Revista Ciencia y Tecnologia*, *17*(1), 85-96.

Pérez, L., Gudiel, R., & others. (2020). *Análisis comparativo de lenguajes de programación para el desarrollo de aplicaciones en Ciencia de Datos*.

Petrović, N., Roblek, V., Radenković, M., & Nejković, V. (2020). *Approach to rapid development of data-driven applications for smart cities using appsheet and apps script*. 77-81.

Piovani, J. I., & Krawczyk, N. (2017). Los Estudios Comparativos: Algunas notas históricas, epistemológicas y metodológicas. *Educação & Realidade*, *42*(3), 821-840.

Prasad, P. (2018). *AUTOMATION TECHNOLOGY AND TOOLS*.

Ramírez Chocce, M. (2021). *Implementación de un sistema informático para la gestión administrativa de la empresa QBA SAC sede Lircay, 2020*.

Ramírez, M. L. V. (2020). Resolución de problemas algorítmicos y objetos de aprendizaje: Una revisión de la literatura. *RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo*, *10*(20), Article 20. https://doi.org/10.23913/ride.v10i20.630

Rodríguez, M., Rodríguez, N. R., & Murazzo, M. A. (2024). Evaluación de la inicialización y el arranque en frio de los lenguajes de programación en una plataforma serverless: Amazon Web Services como caso de estudio. *XXIX Congreso Argentino de Ciencias de la Computación (CACIC)(Luján, 9 al 12 de octubre de 2023)*.

Ruiz, J. (2024, agosto 13). *¿Cuál es el lenguaje de programación más rápido?* https://vikinguard.com/programacion/cual-es-el-lenguaje-de-programacion-mas-rapido/

Ruiz, P. V. (2020). *Modelo emergente para el desarrollo de sistemas informáticos eficientes, utilizando una arquitectura de microservicios en Sunat. División de desarrollo de software aduanero*. http://repositorio.uncp.edu.pe/handle/20.500.12894/6618

Sánchez, J., & Vega, S. (2020, marzo). *Implementación de sistemas de evaluación de personal en empresas chilenas, etapa inicial para medir su impacto en la gestión de la empresa*. ResearchGate. https://www.researchgate.net/publication/340004141_Implementacion_de_sistemas_de_evaluacion_de_personal_en_empresas_chilenas_etapa_inicial_para_medir_su_impacto_en_la_gestion_de_la_empresa

Sarmiento, D. B., & Zhizhpon, C. E. (2022). *Diseño, desarrollo e implementación de una herramienta web de análisis estadístico y monitoreo para evaluar la accesibilidad en*

páginas web según los criterios de conformidad de la WCAG 2.1 [bachelorThesis]. http://dspace.ups.edu.ec/handle/123456789/23410

Soberón, L., & Jesús, J. (2020). *Análisis comparativo de sistemas gestores de bases de datos postgresql y mysql en procesos crud*.

Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9), 664-671.

Soto, N. (2024). *Desarrollo de un Framework de Automatizacion de Pruebas para los Modulos Login, Admin, Pim, Recruitmen y Directory del Sistema de Recursos Humanos Orange Hrm, para Mejorar la Calidad del Sistema* [Thesis]. http://localhost:8080/jspui/handle/123456789/48084

Suárez, R., & Henry, J. (2023). *El software en la transformación digital*. https://hdl.handle.net/20.500.12494/53242

Torres, H. J. (2024). *Análisis de adopción de Metodologías Ágiles (Scrum, Kanban) para mejorar la eficiencia y calidad en el desarrollo de software en la Compañía Ingeniería Integrasayox S.A.* [bachelorThesis, Babahoyo: UTB-FAFI. 2024]. http://dspace.utb.edu.ec/handle/49000/16969

Tyagi, A. K., Fernandez, T. F., Mishra, S., & Kumari, S. (2020). *Intelligent automation systems at the core of industry 4.0*. 1-18.

Tymoschuk, J. (2019). Análisis comparativo de eficiencia de lenguajes de programación. *Universidad Tecnológica Nacional, Facultad Regional Córdoba*. http://www.conaiisi.frc.utn.edu.ar/PDFsParaPublicar/1/schedConfs/5/13-507-2-DR.pdf

Veolia, E. (nd). Un líder global en la gestión optimizada de recursos. *El grupo Veolia en el mundo*. https://www.interagua.com.ec/node/13

Verhoef, P. C., Broekhuizen, T., Bart, Y., Bhattacharya, A., Qi Dong, J., Fabian, N., & Haenlein, M. (2021). Digital transformation: A multidisciplinary reflection and research

agenda. *Journal of Business Research*, *122*, 889-901. https://doi.org/10.1016/j.jbusres.2019.09.022

Villadoma, J. J., & Melendez, H. A. (2024). Optimización del proceso de atención de requerimientos bajo el método basado en casos de uso en una empresa de desarrollo. *Universidad Peruana de Ciencias Aplicadas (UPC)*. https://repositorioacademico.upc.edu.pe/handle/10757/674681

Villamarín, A. (2023). *Introducción a DevSecOps para la mejora de los procesos de desarrollo de software con herramientas Open Source*. https://openaccess.uoc.edu/handle/10609/148098

Voz, O. A., de Fundesarrollo, D. E., Castilla, N. A., de investigaciones de Fundesarrollo, C., Arciniegas, G., Ibarra, O., & Pérez, E. O. (2022). *TRANSFORMACIÓN DIGITAL DE LAS EMPRESAS AFILIADAS A LA ANDI SECCIONAL ATLÁNTICO-MAGDALENA Agosto 2022*.

Zabala, R. M., Granja, L. G., Calderón, H. A., & Velasteguí, L. E. (2021). Efecto en la gestión organizacional y la satisfacción de los usuarios de un sistema informático de planificación de recursos empresariales (ERP) en Riobamba, Ecuador. *Información tecnológica*, *32*, 101-110.

Zaoui, F., & Souissi, N. (2020). Roadmap for digital transformation: A literature review. *Procedia Computer Science*, *175*, 621-628.

Zegarra, J. A., & Reyes, J. L. (2023). Propuesta de automatización de procesos administrativos de la empresa Halconservis S.A.C. *Universidad Peruana de Ciencias Aplicadas (UPC)*. https://repositorioacademico.upc.edu.pe/handle/10757/671112

Anexos

Anexo 1: Elaboración de actividades usando Metodología Kanban

Fase inicial

Por hacer	En progreso	Validación	Finalizado
Selección de lenguajes de programación			
Diseño e implementación de algoritmos			
Desarrollo del módulo en lenguaje Golang			
Desarrollo del módulo en lenguaje NodeJS			
Desarrollo del módulo en lenguaje Python			
Desarrollo del módulo en lenguaje F#			
Prueba de procesamiento de 1000 documentos en los 4 lenguajes			
Prueba de procesamiento de 3000 documentos en los 4 lenguajes			

Prueba de procesamiento de 6000 documentos en los 4 lenguajes			
Recolección de datos de prueba			
Validación y filtrado de datos			
Organización de datos			
Análisis exploratorio inicial			
Evaluación comparativa de rendimiento			
Generación de informes y visualización			
Interpretación de resultados			
Toma de decisiones y propuestas			

Fase 1

Por hacer	En progreso	Validación	Finalizado

			Selección de lenguajes de programación
			Diseño e implementación de algoritmos
	Desarrollo del módulo en lenguaje Golang		
Desarrollo del módulo en lenguaje NodeJS			
	Desarrollo del módulo en lenguaje Python		
	Desarrollo del módulo en lenguaje F#		
Prueba de procesamiento de 1000 documentos en los 4 lenguajes			
Prueba de procesamiento de 3000 documentos en los 4 lenguajes			
Prueba de procesamiento de 6000 documentos en los 4 lenguajes			
Recolección de datos de prueba			

Validación y filtrado de datos			
Organización de datos			
Análisis exploratorio inicial			
Evaluación comparativa de rendimiento			
Generación de informes y visualización			
Interpretación de resultados			
Toma de decisiones y propuestas			

Fase 2

Por hacer	En progreso	Validación	Finalizado
			Selección de lenguajes de programación
			Diseño e implementación de algoritmos
			Desarrollo del módulo en lenguaje Golang

	Desarrollo del módulo en lenguaje NodeJS		
			Desarrollo del módulo en lenguaje Python
		Desarrollo del módulo en lenguaje F#	
Prueba de procesamiento de 1000 documentos en los 4 lenguajes			
Prueba de procesamiento de 3000 documentos en los 4 lenguajes			
Prueba de procesamiento de 6000 documentos en los 4 lenguajes			
Recolección de datos de prueba			
Validación y filtrado de datos			
Organización de datos			
Análisis exploratorio inicial			

Evaluación comparativa de rendimiento			
Generación de informes y visualización			
Interpretación de resultados			
Toma de decisiones y propuestas			

Fase 3

Por hacer	En progreso	Validación	Finalizado
			Selección de lenguajes de programación
			Diseño e implementación de algoritmos
			Desarrollo del módulo en lenguaje Golang
			Desarrollo del módulo en lenguaje NodeJS
			Desarrollo del módulo en lenguaje Python
			Desarrollo del módulo en lenguaje F#

		Prueba de procesamiento de 1000 documentos en los 4 lenguajes	
		Prueba de procesamiento de 3000 documentos en los 4 lenguajes	
		Prueba de procesamiento de 6000 documentos en los 4 lenguajes	
	Recolección de datos de prueba		
	Validación y filtrado de datos		
	Organización de datos		
Evaluación comparativa de rendimiento			
Generación de informes y visualización			
Interpretación de resultados			
Toma de decisiones y propuestas			

Fase 4

Por hacer	En progreso	Validación	Finalizado
			Selección de lenguajes de programación
			Diseño e implementación de algoritmos
			Desarrollo del módulo en lenguaje Golang
			Desarrollo del módulo en lenguaje NodeJS
			Desarrollo del módulo en lenguaje Python
			Desarrollo del módulo en lenguaje F#
			Prueba de procesamiento de 1000 documentos en los 4 lenguajes
			Prueba de procesamiento de 3000 documentos en los 4 lenguajes
			Prueba de procesamiento de

			6000 documentos en los 4 lenguajes
			Recolección de datos de prueba
			Validación y filtrado de datos
			Organización de datos
		Evaluación comparativa de rendimiento	
	Generación de informes y visualización		
	Interpretación de resultados		
	Toma de decisiones y propuestas		

Fase 5

Por hacer	En progreso	Validación	Finalizado
			Selección de lenguajes de programación
			Diseño e implementación de algoritmos

			Desarrollo del módulo en lenguaje Golang
			Desarrollo del módulo en lenguaje NodeJS
			Desarrollo del módulo en lenguaje Python
			Desarrollo del módulo en lenguaje F#
			Prueba de procesamiento de 1000 documentos en los 4 lenguajes
			Prueba de procesamiento de 3000 documentos en los 4 lenguajes
			Prueba de procesamiento de 6000 documentos en los 4 lenguajes
			Recolección de datos de prueba
			Validación y filtrado de datos
			Organización de datos

			Evaluación comparativa de rendimiento
			Generación de informes y visualización
			Interpretación de resultados
			Toma de decisiones y propuestas

Anexo 2: Primer avance del desarrollo del sistema

Entrega del Primer Avance en la implementación del producto tecnológico del Proyecto de Integración Curricular.

Información general

- **Nombre del estudiante:**

  Kenny Pizarro Luzón

- **Título del proyecto:**

Evaluación de lenguajes de programación para el desarrollo de un sistema con procesamiento paralelo en una empresa de soluciones medioambientales

- **Fecha de presentación:**

  29 de septiembre del 2024

Descripción del avance

La propuesta es crear cuatro sistemas que realizarán el mismo proceso en lenguajes distintos con la finalidad de evaluar el rendimiento de los lenguajes para el escenario en el cual se encuentra planteada la problemática. El proceso es navegar en un directorio de

Google Drive, recoger información de los archivos Hojas de Cálculo y convertir en PDF las hojas que contiene cada archivo aplicando procesamiento paralelo, almacenar estos archivos dentro de un directorio en la nube de Google y registrar las ejecuciones en una base de datos.

La propuesta ha avanzado un 35% en su totalidad, con el progreso distribuido en las siguientes funcionalidades y componentes, cada uno con sus respectivos porcentajes de avance:

Componentes y Funcionalidades:

El sistema consiste en los siguientes componentes y funcionalidades:

1) Creación de credenciales de acceso y de la API de Google Drive en la consola de Google, generadas en formato .json (100%)

2) Sistema desarrollado en lenguaje Python (70%)

3) Sistema desarrollado en lenguaje Go (70%)

4) Sistema desarrollado en lenguaje F# (70%)

5) Sistema desarrollado en lenguaje Node.js (0%)

6) Conexión a la base de datos (0%)

7) Creación de Lambdas en Amazon Web Services (AWS)

8) Creación del servidor virtual (0%)

9) Creación de los contenedores (0%)

- **Avances Específicos:**

Se ha completado la propuesta inicial del proyecto en los lenguajes Go, Python y F#. Inicialmente la propuesta incluía también realizar la propuesta en lenguaje Erlang, pero lastimosamente no se pudo completar ya que no tiene soporte para la API de Google Drive.

Se consultó con el experto de Transformación Digital de la empresa de soluciones medioambientales donde recomendó hacerlo con Javascript (Node.js)

1. Conexión a Google Drive: Se ha implementado y probado la conexión a Google Drive para leer archivos en Go, Python y F#.

2. Control de versiones: Se crearon los repositorios en la nube de Github para manejar y controlar las versiones de modificación del código fuente.

3. Directorios de Google Drive: Se crearon los directorios en Google Drive en una cuenta personal para realizar las pruebas iniciales

4. Generación de PDF: Se ha creado el código para generar PDF de forma secuencial y en paralelo en los lenguajes mencionados.

5. Procesamiento Paralelo: Se ha logrado implementar en el código y verificar el funcionamiento del procesamiento paralelo en Go, Python y F#.

6. Pruebas iniciales: Aunque se han realizado pruebas con una pequeña muestra de 11 archivos para verificar el funcionamiento del procesamiento paralelo, aún no se ha probado el sistema con el escenario real que involucra un mayor número de archivos.

7. Retroalimentación Recibida: El 26 de septiembre, el experto en Transformación Digital evaluó el proyecto, destacando su potencial aplicabilidad en otros procesos dentro de la empresa. Además, sugirió ejecutar las pruebas reales en un servidor virtual con contenedores en Docker y utilizar Amazon Web Services (AWS) para implementar las funciones Lambdas, que luego podrían ser invocadas desde Google Apps Script.

Herramientas y Tecnologías Utilizadas:

1. Entorno de Desarrollo: Visual Studio Code.

2. Bibliotecas: Dependencias específicas para la conexión a la API de Google Drive en cada lenguaje distinto

3. Metodologías: Sin el uso de frameworks, dado que no es una aplicación completa sino un proceso automatizado.

4. Lenguaje Golang

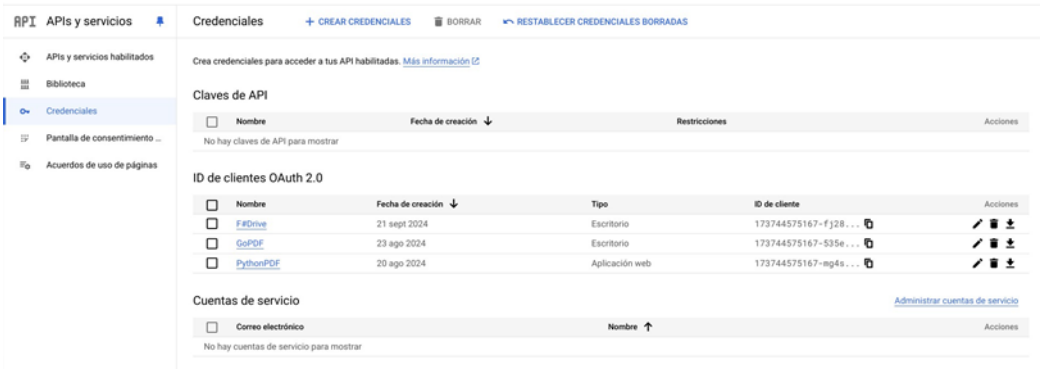
5. Lenguaje Python

6. Lenguaje F#

3. Evidencia visual

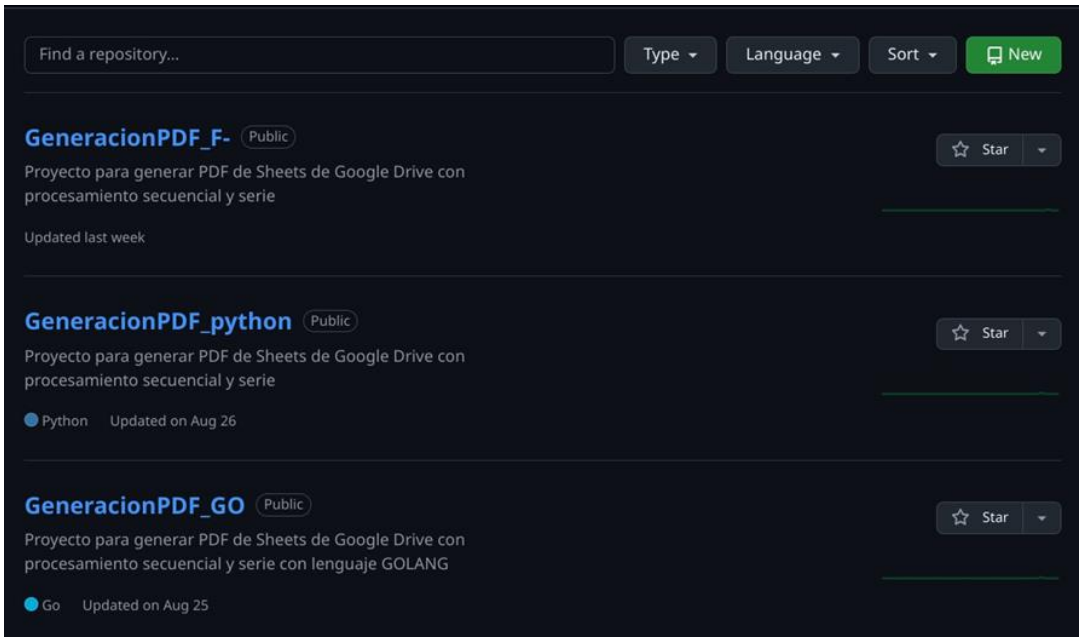
Instrucciones: Inserta una imagen o captura de pantalla que evidencie el avance logrado. Asegúrate de que la imagen sea clara y relevante para el avance que estás reportando.

- **Imagen 1**



Descripción: Configuración de las credenciales de autenticación para la API de Google Drive, se generaron tres credenciales, cada una será consumida en los lenguajes mencionados: Go, Python y F#.

- Imagen 2:



Descripción:

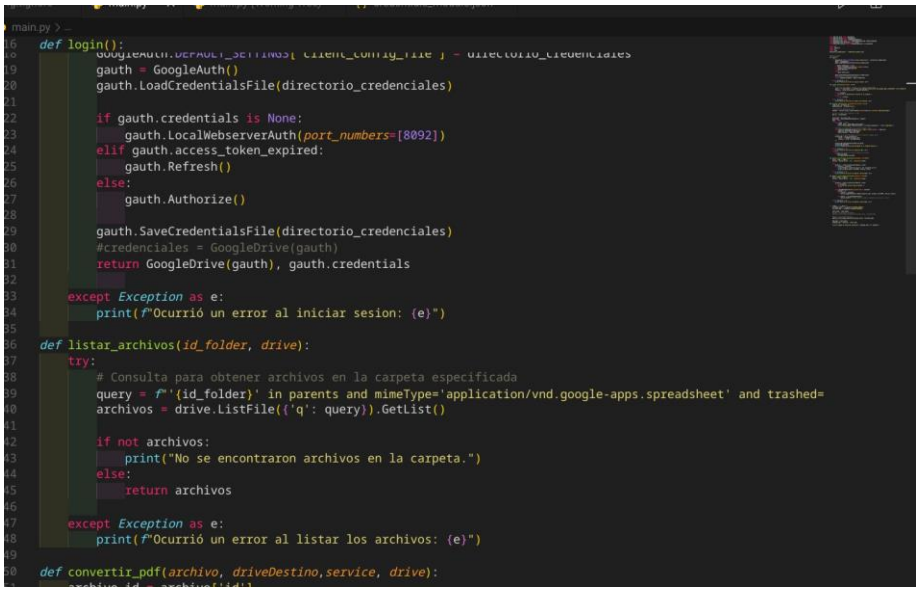
Creación de los repositorios para alojar el código en Github para tener un mejor control de las modificaciones que se realizaron en el código.

- Imagen 3:

Descripción:

Se crean los directorios en mi cuenta personal de Google Drive para testear inicialmente la funcionalidad de generación de PDF.

- Imagen 4:

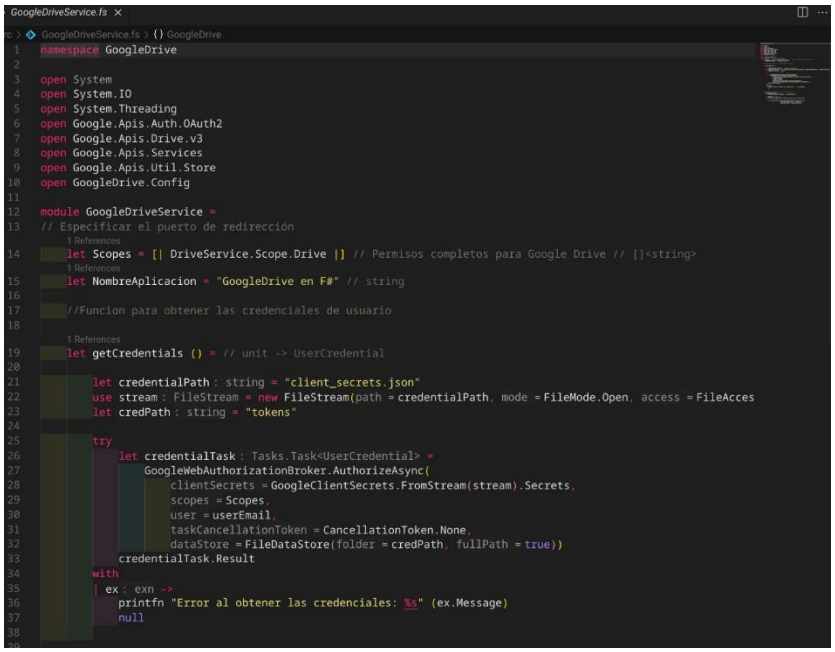


```python
def login():
        gauth = GoogleAuth()
        gauth.LoadCredentialsFile(directorio_credenciales)

        if gauth.credentials is None:
            gauth.LocalWebserverAuth(port_numbers=[8092])
        elif gauth.access_token_expired:
            gauth.Refresh()
        else:
            gauth.Authorize()

        gauth.SaveCredentialsFile(directorio_credenciales)
        #credenciales = GoogleDrive(gauth)
        return GoogleDrive(gauth), gauth.credentials

    except Exception as e:
        print(f"Ocurrió un error al iniciar sesion: {e}")

def listar_archivos(id_folder, drive):
    try:
        # Consulta para obtener archivos en la carpeta especificada
        query = f"'{id_folder}' in parents and mimeType='application/vnd.google-apps.spreadsheet' and trashed=
        archivos = drive.ListFile({'q': query}).GetList()

        if not archivos:
            print("No se encontraron archivos en la carpeta.")
        else:
            return archivos

    except Exception as e:
        print(f"Ocurrió un error al listar los archivos: {e}")

def convertir_pdf(archivo, driveDestino,service, drive):
```

Descripción:

Se desarrolla el sistema en el lenguaje Python donde se implementó la conexión a Google Drive, gestión de los datos de cada archivo mediante Buffers, creación de la función para generar PDF y la lógica con procesamiento paralelo

- Imagen 5:

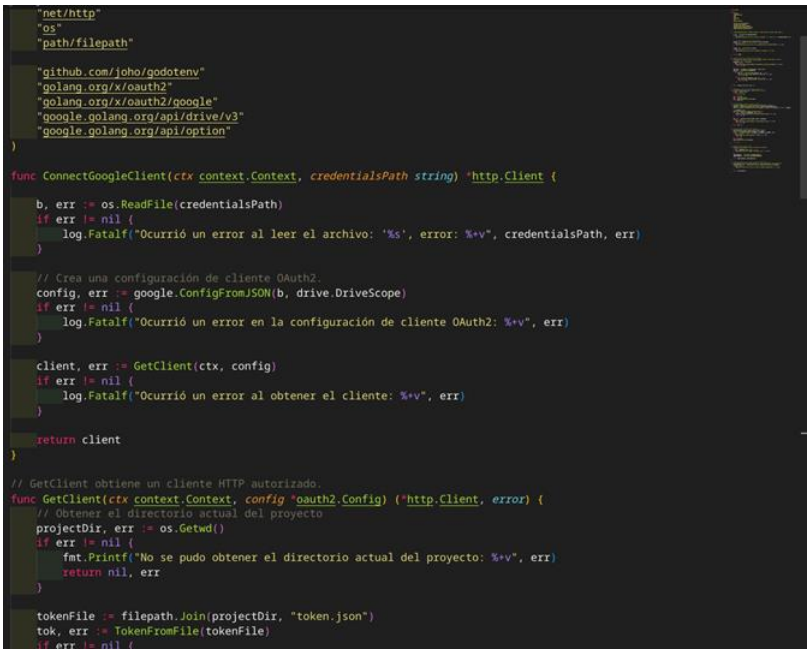


```fsharp
namespace GoogleDrive

open System
open System.IO
open System.Threading
open Google.Apis.Auth.OAuth2
open Google.Apis.Drive.v3
open Google.Apis.Services
open Google.Apis.Util.Store
open GoogleDrive.Config

module GoogleDriveService =
// Especificar el puerto de redirección
    let Scopes = [| DriveService.Scope.Drive |] // Permisos completos para Google Drive // []<string>
    let NombreAplicacion = "GoogleDrive en F#" // string

    //Funcion para obtener las credenciales de usuario

    let getCredentials () = // unit -> UserCredential

        let credentialPath : string = "client_secrets.json"
        use stream : FileStream = new FileStream(path = credentialPath, mode = FileMode.Open, access = FileAcces
        let credPath : string = "tokens"

        try
            let credentialTask : Tasks.Task<UserCredential> =
                GoogleWebAuthorizationBroker.AuthorizeAsync(
                    clientSecrets = GoogleClientSecrets.FromStream(stream).Secrets,
                    scopes = Scopes,
                    user = userEmail,
                    taskCancellationToken = CancellationToken.None,
                    dataStore = FileDataStore(folder = credPath, fullPath = true))
            credentialTask.Result
        with
        | ex : exn ->
            printfn "Error al obtener las credenciales: %s" (ex.Message)
            null
```

Descripción:

Se desarrolla el sistema en el lenguaje F# donde se implementó la conexión a Google Drive, gestión de los datos de cada archivo mediante Buffers, creación de la función para generar PDF y la lógica con procesamiento paralelo

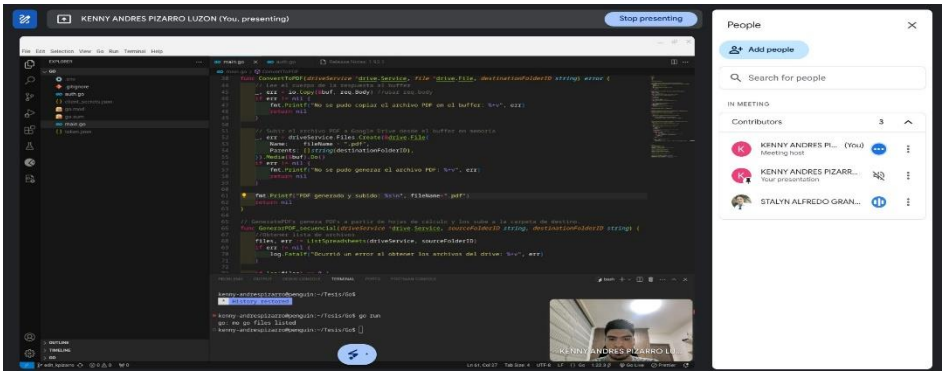
- Imagen 6



Descripción:

Se desarrolla el sistema en el lenguaje Go donde se implementó la conexión a Google Drive, gestión de los datos de cada archivo mediante Buffers, creación de la función para generar PDF y la lógica con procesamiento paralelo

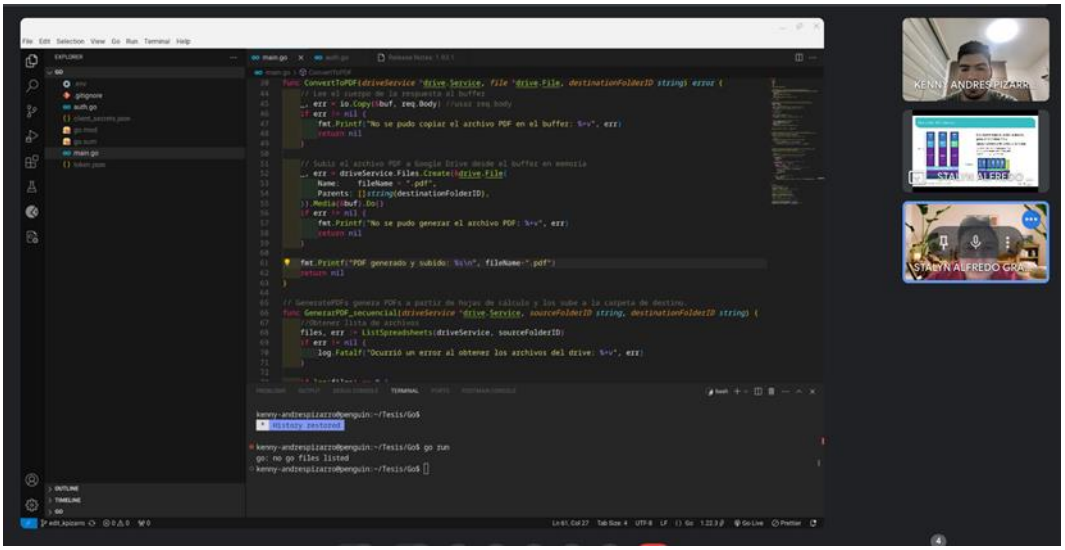
- Imagen 7



Descripción:

Revisión en conjunto con el experto en Transformación Digital de los módulos del sistema (GO, Python, F#).

- Imagen 8:



Descripción

Sugerencias para ejecutar las pruebas reales en un servidor virtual con contenedores en Docker y utilizar Amazon Web Services (AWS) para implementar las funciones Lambdas, que luego podrían ser invocadas desde Google Apps Script.

4. Reflexión sobre el avance

- Retos encontrados:

Python: El principal reto fue la conexión a Google Drive utilizando credenciales en formato .json y cómo manejar la generación de PDFs sin ocupar excesiva memoria local.

Golang: El aprendizaje del lenguaje, manejo de funciones y estructuras, así como el manejo de buffers, fue un desafío al ser un lenguaje nuevo.

F#: La configuración inicial del entorno de desarrollo y la instalación de dependencias en Visual Studio Code resultaron retadoras.

Generar y conectar las credenciales de Google Drive también presentó dificultades en todos los lenguajes.

- **Soluciones adoptadas:**

Para superar estos retos, se adoptaron las siguientes estrategias:

Documentación y Recursos: Se revisó exhaustivamente la documentación oficial de cada lenguaje, así como la documentación de la API de Google Drive.

Aprendizaje y Exploración: Se exploraron videos tutoriales y se analizaron proyectos similares para obtener un mejor entendimiento de cómo otros desarrolladores han abordado problemas similares.

Iteración y Optimización: Se realizaron múltiples pruebas y ajustes en el código, lo que permitió iterar y optimizar las soluciones implementadas para que funcionen de manera eficiente en cada lenguaje.

- **Pasos siguientes:**

El siguiente paso es completar la implementación del sistema en el lenguaje Node.js y preparar la lógica de cada lenguaje para el escenario real del proceso de la empresa. La configuración de servidores, contenedores y la implementación de Lambdas en AWS se realizará en etapas posteriores.

Anexo 3: Segundo avance del desarrollo del sistema

Avance en la implementación del producto o servicio tecnológico del Proyecto Integrador Curricular

Información general

- **Nombre del estudiante:**

  Kenny Pizarro Luzón

- **Título del proyecto:**

Evaluación de lenguajes de programación para el desarrollo de un sistema con procesamiento paralelo en una empresa de soluciones medioambientales

- **Fecha de presentación:**

13 de noviembre del 2024

- **Descripción del avance:**

La propuesta consiste en desarrollar módulos que generen informes trimestrales en formato PDF a partir de hojas de cálculo almacenadas en Google Drive, utilizando procesamiento paralelo.

Estos módulos se implementarán en diferentes lenguajes de programación, como Golang, Node.js, Python y F#, con el objetivo de evaluar el rendimiento de cada uno. El desarrollo de la propuesta se ha completado en su totalidad.

Componentes y Funcionalidades:

A continuación se presentan los componentes y funcionalidades que contiene el sistema:

Funcionalidades	Avance (%)
Creación de credenciales de acceso a la API de Google para cada lenguaje	100
Desarrollo del módulo en lenguaje Golang	100
Desarrollo del módulo en lenguaje Python	100
Desarrollo del módulo en lenguaje F#	100
Desarrollo del módulo en lenguaje Nodejs	100
Creación de archivos Log que registren los tiempos de procesamiento para cada lenguaje	100

Avances realizados

1. Desarrollo del módulo con Nodejs esto incluye la creación de las credenciales de acceso a la API para Nodejs, creación de la carpeta de Google Drive donde

se almacenarán los archivos PDF, creación del código en visual studio code, repositorio en Github y las pruebas iniciales para testear el funcionamiento del proceso en paralelo.

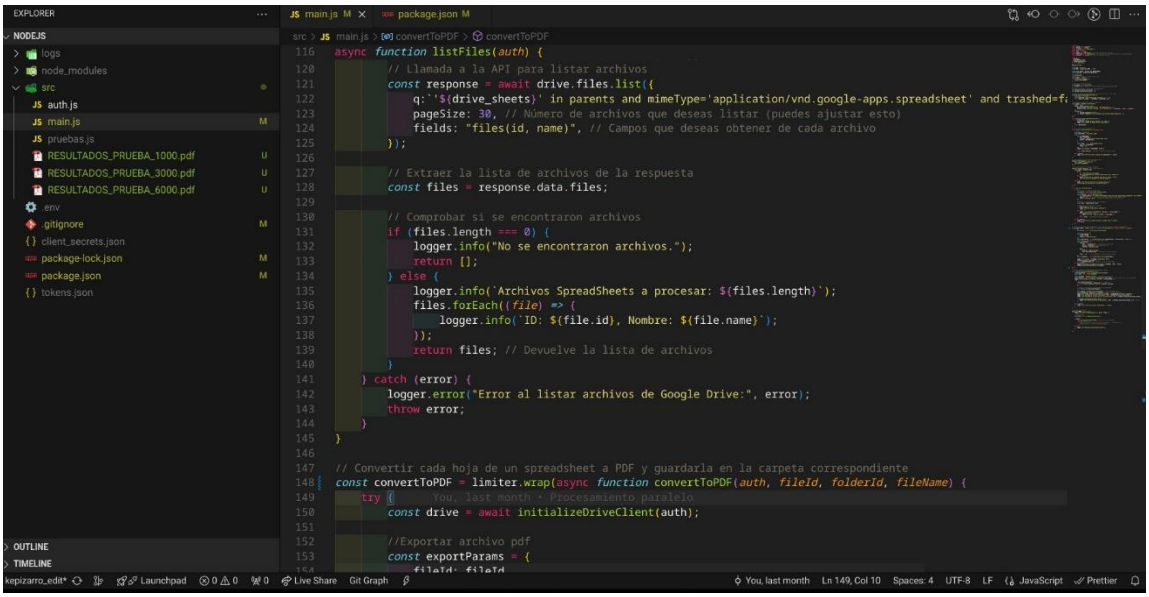
2. Creación de archivos Log en todos los módulo del sistema (Go, Python, Nodejs y F#) para registrar los tiempos de procesamiento de cada módulo

3. Preparación del escenario para las pruebas, se modificó el código en todos los módulos del sistema con la finalidad de ajustar el escenario real para las pruebas de rendimiento

4. Pruebas de rendimiento, se realizaron las pruebas de rendimiento en cada lenguaje con las siguientes muestras de procesamiento: 1000 documentos, 3000 documentos y 6000 documentos.

3. Evidencia visual

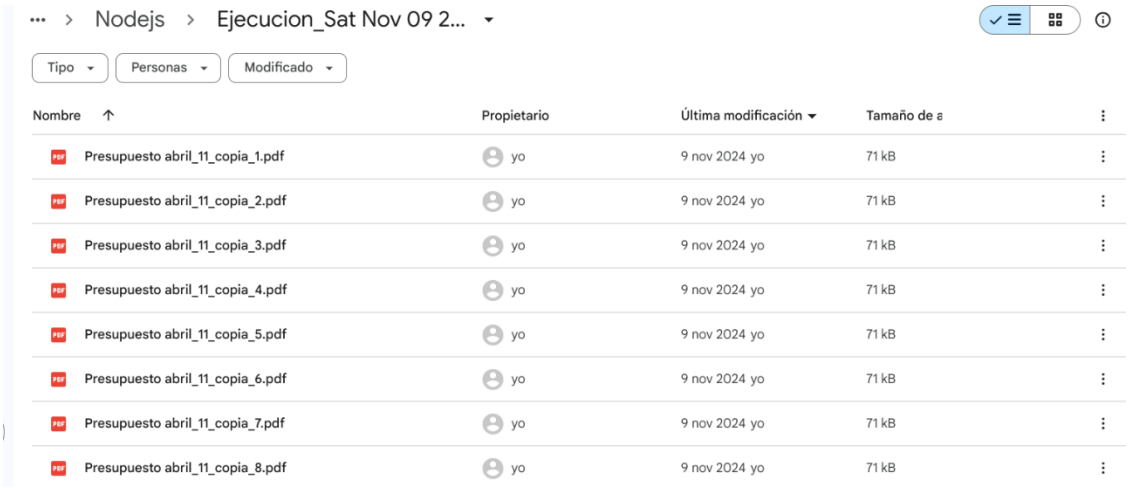
- Imagen 1



Descripción:

Se desarrolla el código en lenguaje Nodejs donde se procesan paralelamente los archivos en un directorio de Google Drive.

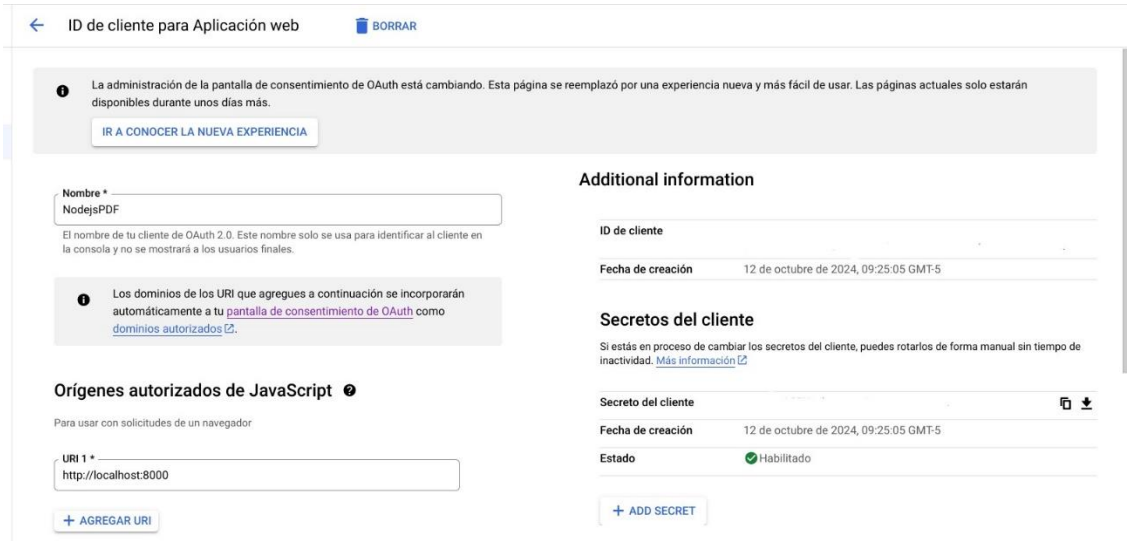
Imagen 2



Descripción:

Se crea el directorio de NODEJS donde se almacenarán los documentos PDF convertidos.

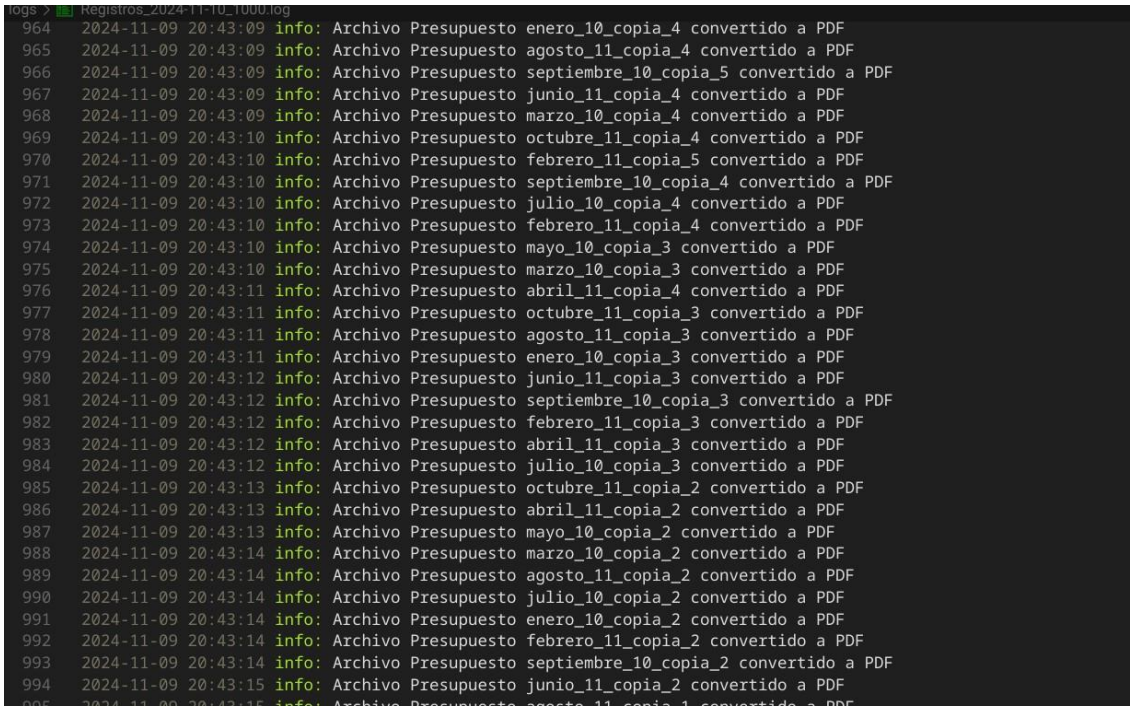
- Imagen 3



Descripción:

Se crean las credenciales de acceso a la API de Google para el módulo con NodeJS

- Imagen 4



Descripción:

Creación de archivos Logs para registrar información como documentos en proceso, tiempos de procesamiento, tiempo de ejecución del programa, cantidad de archivos procesados.

- **Imagen 5**

Descripción:

Se realizaron las pruebas de rendimiento en cada lenguaje propuesto, cada carpeta contiene los archivos procesados para las muestras de 1000, 3000 y 6000 documentos.

Reflexión sobre el avance

Retos encontrados:

Preparar el escenario para las pruebas en cada lenguaje fue un reto, ya que a medida que se ajustaba el código para el escenario de las pruebas, nuevos inconvenientes se presentaban en cada lenguaje. Por ejemplo, en Python, dió el siguiente error: error [SSL:

DECRYPTION_FAILED_OR_BAD_RECORD_MAC] decryption failed or bad record mac (_ssl.c:2546) python

Lo cual indicaba un problema de red, de la dirección MAC y problema en la certificación de uso de la API. En NodeJS la mayoría de los documentos no se procesaban por el número de peticiones realizadas a la API de Google.

Soluciones adoptadas:

Para el inconveniente presentado en Python se investigó en foros de internet programas que hayan dado el mismo error presentado, los cuales me dieron una idea más clara de cómo poder solventar la novedad. Para el caso de NodeJS incluí librerías para controlar el número de peticiones a la API de Google.

Anexo 4: Carta de Compromiso de labor tutorial

ANEXO No. 2

PROCESO DETITULACIÓ

CARTA DE COMPROMISO DE LABOR TUTORIAL

Yo, Alejandra Colina Vargas docente de la Unidad Académica Facultad de Ingenierías, Arquitectura y Ciencias de la Naturaleza, tutor designado de Pizarro Luzón Kenny Andrés perteneciente a la carrera de Ingeniería en Software. Reconozco y acepto que, brindaré asistencia y guía tutorial al estudiante señalado, que se encuentra inscrito en el proceso de titulación 2 del año 2024, hasta la culminación satisfactoria de su trabajo de titulación; informando periódicamente los respectivos avances del tutoreado. El estudiante

se compromete a trabajar y colaborar con el docente asignado como tutor en la presentación de los avances de su trabajo de titulación en las respectivas fechas asignadas, así como acatar las normas establecidas en el formato de presentación de trabajo de titulación, de acuerdo con la modalidad establecida. Para constancia de todo lo acordado y resuelto, se firma este documento en esta sede el día 30/08/2024.

Firma del estudiante

Kenny Pizarro Luzón

Firma del Tutor

Alejandra Colina Vargas

Anexo 4: Perfil del Experto

Nombres y Apellidos: Granda Chipre Stalyn Alfredo

Título profesional: Ingeniero en Sistemas Computacionales

Empresa donde trabaja actualmente: Veolia Ecuador S.A. (Operadora de Interagua)

Cargo actual: jefe de Transformación Digital DB&T Veolia

Cursos realizados:

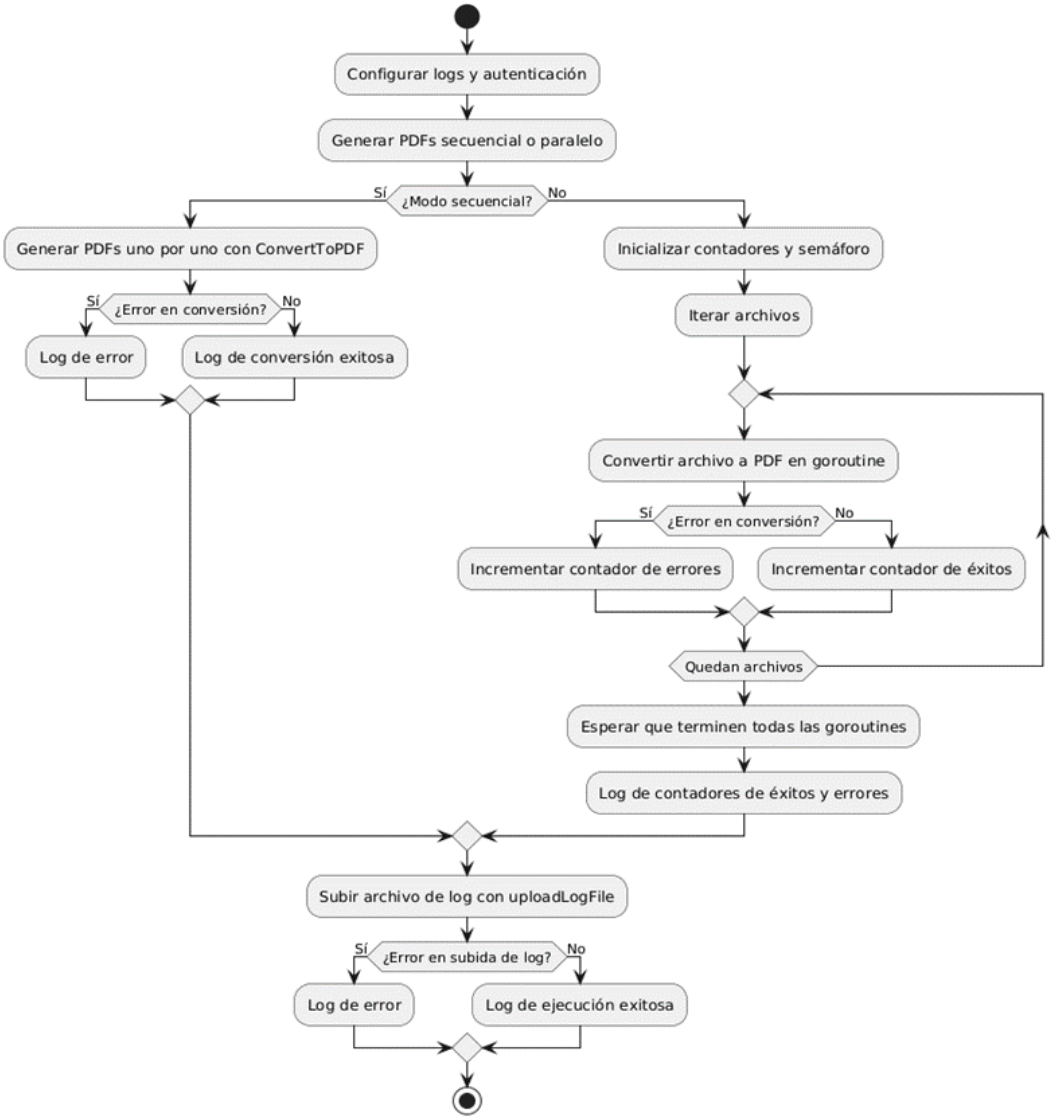
- Introducción de CMMI DEV (2015)
- Introducción de la norma ISO 9001:2008 (2011)
- Framework Java Server Faces JSF (2015)
- Programación en Shell de Unix (2015)

- Android Studio (2016)
- Talleres de Metodologías Ágiles, SCRUM, KANBAN (2015)

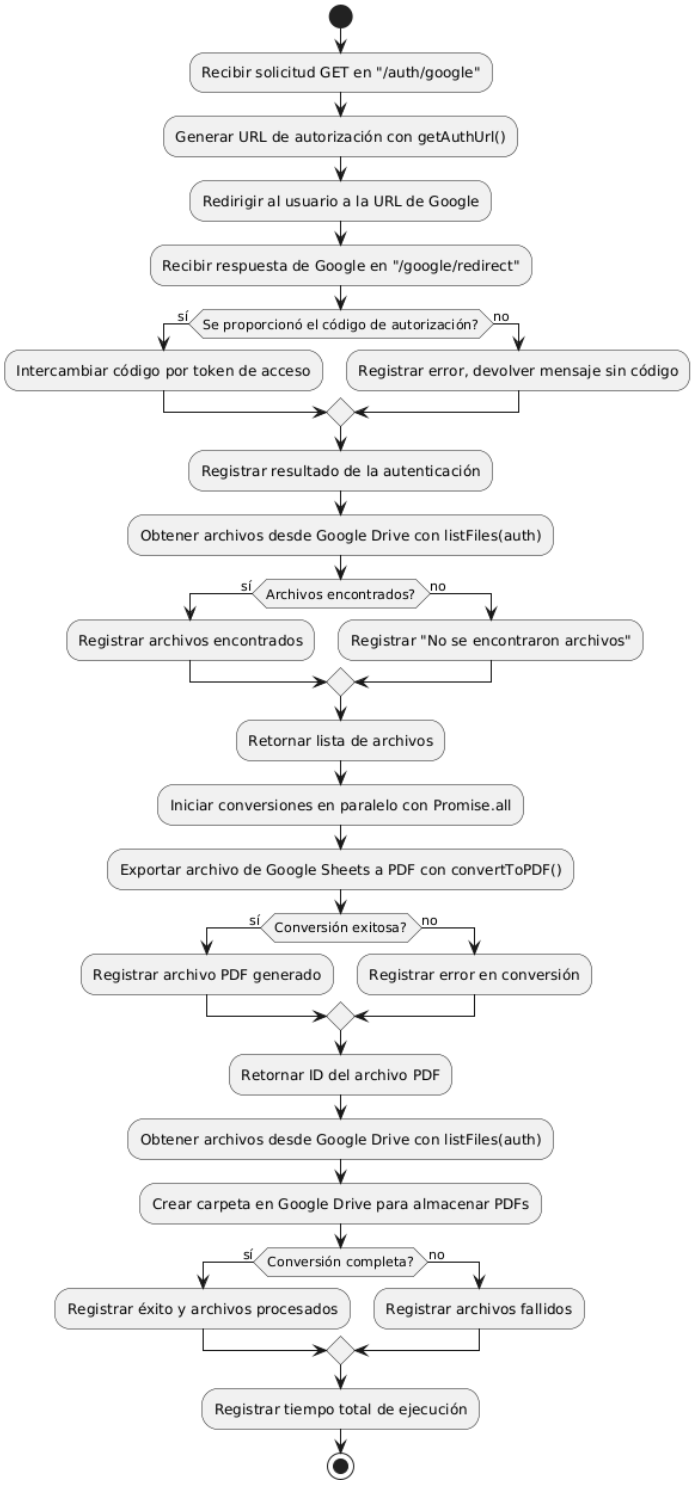
Conocimientos:

- Manejo de Herramientas de Programación: Java, GlassFish, Tomcat v8+, Android, JavaScript, PHP, Shell Script (S.O.), Visual Basic 6.0, C# .Net
- Manejo de Base de Datos: Oracle 10 g+, BigQuery, MySQL, PostgreSQL, SQL SERVER 2008R2
- Instalación y Administración de Sistemas Operativos: Centos 6.+, Ubuntu 16.+, Debian 8.+, Windows XP, 7, 8 y 10, Windows Server 2003, Administración de redes, Administración de base de datos.
- Manejo de herramientas: Open Smartflex (FCED, FWCGR) - Reportes, G-Suit by Google Cloud, GitHUB, GitLab (Versionadores), Netbeans, Eclipse, PLSQL, PgAdmin, Putty, Notepad++, Filezilla, WinSCP, SQLYog, HeidiSQL.

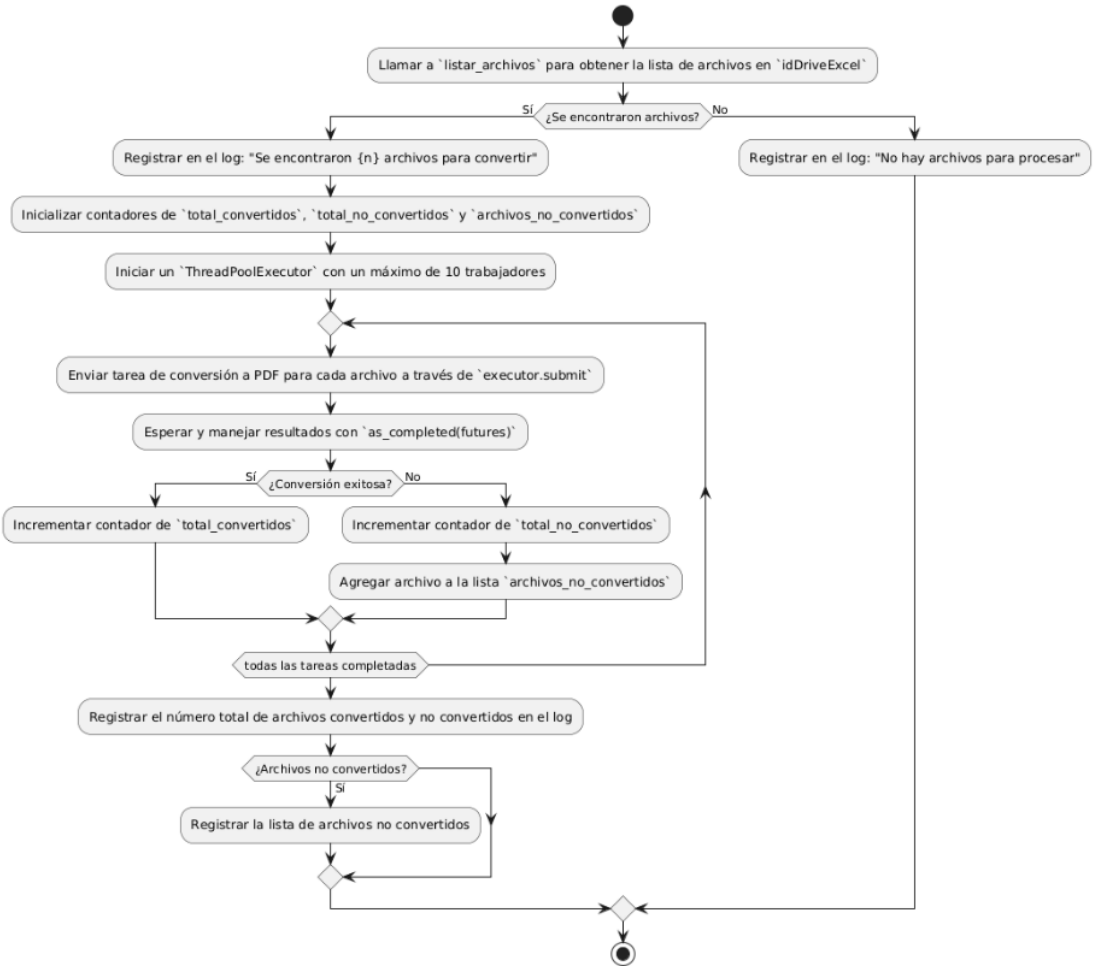
Anexo 5: Diagrama de flujo del módulo Golang



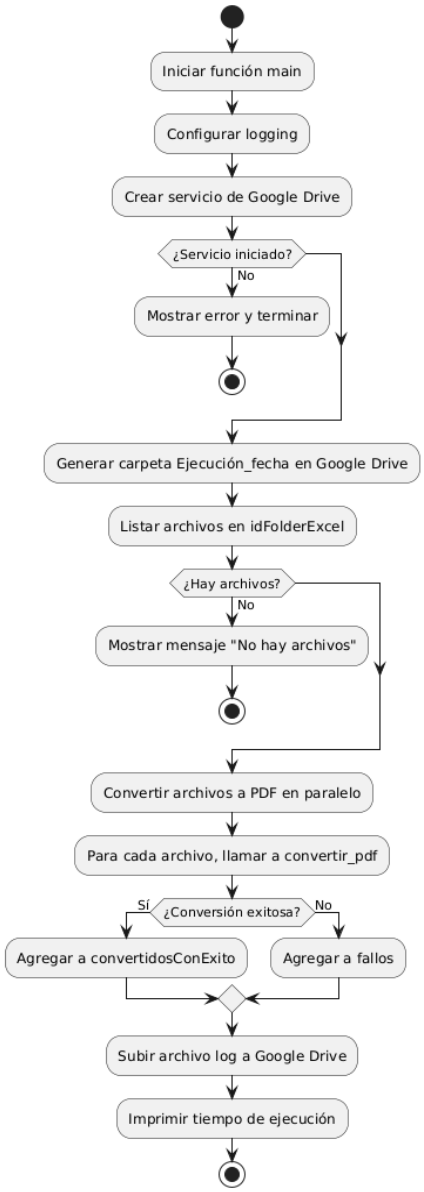
Anexo 6: Diagrama de flujo del módulo NodeJS



Anexo 7: Diagrama de flujo del módulo Python



Anexo 8: Diagrama de flujo del módulo F#



Anexo 9: Código principal del módulo Golang

```
package main


import (
	"bytes"
	"context"
```

```
	"fmt"
	"io"
	"log"
	"os"
	"path/filepath"
	"sync"
	"time"

	"google.golang.org/api/drive/v3"
)

// setupLogging configura el sistema de logs para escribir en un archivo y en la consola.
func setupLogging() string {
	// Crear el directorio "logs" si no existe
	err := os.MkdirAll("logs", os.ModePerm)
	if err != nil {
		log.Fatalf("No se pudo crear el directorio de logs: %v", err)
	}

	// Establecer el nombre del archivo de log con la fecha actual
	logFileName := filepath.Join("logs", fmt.Sprintf("app-%s.log", time.Now().Format("2006-01-02_15-04-05")))

	// Abrir el archivo de log
	logFile, err := os.OpenFile(logFileName, os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0666)
```

```
	if err != nil {
		log.Fatalf("No se pudo abrir el archivo de log: %v", err)
	}

	// Configurar el log para escribir en el archivo y en la consola
	multiWriter := io.MultiWriter(os.Stdout, logFile)
	log.SetOutput(multiWriter)
	log.SetFlags(log.Ldate | log.Ltime | log.Lshortfile)

	return logFileName
}

func uploadLogFile(driveService *drive.Service, logFilePath, destinationFolderID string)
error {
	f, err := os.Open(logFilePath)
	if err != nil {
		return fmt.Errorf("no se pudo abrir el archivo de registro: %w", err)
	}
	defer f.Close()

	_, err = driveService.Files.Create(&drive.File{
		Name:    filepath.Base(logFilePath), // Usar el nombre del archivo original
		Parents: []string{destinationFolderID},
	}).Media(f).Do()
	if err != nil {
		return fmt.Errorf("no se pudo subir el archivo de registro: %w", err)
```

```
	}

	log.Printf("Archivo de registro subido a Google Drive: %s\n", filepath.Base(logFilePath))
	return nil
}

// ListSpreadsheets lista los archivos de tipo spreadsheet en la carpeta especificada.
func ListSpreadsheets(driveService *drive.Service, folderID string) ([]*drive.File, error) {
	query := fmt.Sprintf("'%s' in parents and mimeType='application/vnd.google-apps.spreadsheet' and trashed=false", folderID)
	r, err := driveService.Files.List().
		Q(query).
		PageSize(30).
		Fields("nextPageToken, files(id, name)").
		Do()
	if err != nil {
		log.Printf("No se pueden leer los archivos: %+v", err)
		return nil, err
	}
	return r.Files, nil
}

// ConvertToPDF convierte un archivo de Google Sheets a PDF y lo sube a una carpeta de destino.
```

```
func ConvertToPDF(driveService *drive.Service, file *drive.File, destinationFolderID string)
error {
	fileID := file.Id
	fileName := file.Name

	// Solicitar la exportación a PDF
	req, err := driveService.Files.Export(fileID, "application/pdf").Download()
	if err != nil {
		log.Printf("Ocurrió un error al exportar el archivo: %+v", err)
		return err
	}
	defer req.Body.Close()

	var buf bytes.Buffer

	// Lee el cuerpo de la respuesta al buffer
	_, err = io.Copy(&buf, req.Body) //usar req.body
	if err != nil {
		log.Printf("No se pudo copiar el archivo PDF en el buffer: %+v", err)
		return nil
	}

	// Subir el archivo PDF a Google Drive desde el buffer en memoria
	_, err = driveService.Files.Create(&drive.File{
		Name:    fileName + ".pdf",
		Parents: []string{destinationFolderID},
```

```
	}).Media(&buf).Do()
	if err != nil {
		log.Printf("No se pudo generar el archivo PDF: %+v", err)
		return nil
	}

	log.Printf("PDF generado y subido: %s\n", fileName+".pdf")
	return nil
}
func GenerarPDF_paralela(driveService *drive.Service, sourceFolderID string, destinationFolderID string) (int, int, error) {
	files, err := ListSpreadsheets(driveService, sourceFolderID)
	if err != nil {
		log.Fatalf("Ocurrió un error al obtener los archivos del drive: %+v", err)
	}

	if len(files) == 0 {
		log.Println("No se encontraron archivos de hoja de cálculo en la carpeta.")
		return 0, 0, nil
	} else {
		log.Printf("Se procesarán %d archivos", len(files))
		contador_conversion := 0
		contador_fallidos := 0

		var mu sync.Mutex	// Mutex para acceso concurrente seguro a los contadores
```

```
var wg sync.WaitGroup // Grupo de espera para sincronizar las goroutines
wg.Add(len(files))
// Crear un canal para esperar a que todas las goroutines terminen
/*done := make(chan struct {
	fileName string
	err      error
}, len(files))*/

// Límite de goroutines concurrentes (para evitar sobrecargar el sistema)
const maxGoroutines = 10
semaphore := make(chan struct{}, maxGoroutines)

for _, file := range files {
	semaphore <- struct{}{} // Adquirir un cupo en el semáforo
	go func(f *drive.File) {
		defer func() {
			<-semaphore
			wg.Done()
		}() // Liberar el cupo en el semáforo al terminar

		err := ConvertToPDF(driveService, f, destinationFolderID)
		if err != nil {
			log.Printf("Error al convertir el archivo '%s' a PDF: %+v\n", f.Name, err)
			mu.Lock()
			contador_fallidos++
```

```
                                        mu.Unlock()
                                } else {
                                        mu.Lock()
                                        contador_conversion++
                                        mu.Unlock()
                                }
                        }(file)
                }

                wg.Wait() // Esperar a que todas las goroutines terminen

                return contador_conversion, contador_fallidos, nil
        }
}

func crearCarpetaEjecuciones(ctx context.Context, driveService *drive.Service,
destinationFolderID string) (string, error) {
        horaActual := time.Now().Format("2006-01-02_15:04:05")
        nombreCarpeta := fmt.Sprintf("Ejecución_%s", horaActual)

        //crear carpeta
        folderMetaData := &drive.File{
                Name:     nombreCarpeta,
                MimeType: "application/vnd.google-apps.folder",
                Parents:  []string{destinationFolderID},
        }
```

```
	crearCarpeta, err := driveService.Files.Create(folderMetaData).Context(ctx).Do()
	if err != nil {
		return "", fmt.Errorf("error al crear la carpeta de ejecuciones: %w", err)
	}

	log.Printf("Carpeta ejecuciones creada: %s (ID: %s)\n", nombreCarpeta, crearCarpeta.Id)
	return crearCarpeta.Id, nil // Return the ID of the newly created folder
}

func main() {
	// Configurar el logging
	logFileName := setupLogging()

	//Cargar los datos
	ctx := context.Background() //contexto

	driveSheets, driveDestino := LoadEnv()
	credenciales := "client_secrets.json" //credenciales aplicacion

	//Conectarse con el cliente de Google
	client := ConnectGoogleClient(ctx, credenciales)

	// Crear el servicio de Google Drive
	driveService := CreateDriveService(ctx, client)
```

```
	log.Println("Conectado exitosamente a Google Drive")

	carpetaEjecucionesID, err := crearCarpetaEjecuciones(ctx, driveService, driveDestino)
	if err != nil {
		log.Fatalf("Error al crear la carpeta de ejecuciones: %v\n", err)
	}
	startTime := time.Now() // Capturar el tiempo de inicio

	// Generar y subir PDFs de forma secuencial
	//GenerarPDF_secuencial(driveService, driveSheets, carpetaEjecucionesID)

	//Generar y subir PDFs de forma paralela
	convertidos, fallidos, err := GenerarPDF_paralela(driveService, driveSheets, carpetaEjecucionesID)
	if err != nil {
		log.Fatalf("Error al generar PDFs: %v", err)
	}

	log.Printf("Archivos convertidos a PDF: %d\n", convertidos)
	log.Printf("Archivos que no se convirtieron a PDF: %d\n", fallidos)

	tiempoEjecucion := time.Since(startTime) //Calcular el tiempo de ejecucion
	log.Printf("Tiempo total de ejecución: %s\n", tiempoEjecucion)

	err = uploadLogFile(driveService, logFileName, carpetaEjecucionesID)
```

```
        if err != nil {
                log.Fatalf("Error al subir el archivo de registro: %v\n", err)
        }
}
```

Anexo 10: Código principal del módulo Python

```
import os
import time
from src.auth import *
from src.GoogleDrive import *
from src.log_config import *
from googleapiclient.discovery import build
from dotenv import load_dotenv

load_dotenv()
if __name__ == "__main__":
    id_folder_excel = os.getenv("ID_MUESTRA_UNO")
    id_folder_pdf = os.getenv("ID_DRIVE_DESTINO")
    creds = login()

    service_drive = build('drive', 'v3', credentials=creds)
    id_drive_destino = crear_carpeta(service_drive, id_folder_pdf)

    if os.path.exists(directorio_credenciales):
        logger.info("Archivo de credenciales encontrado. Iniciando el proceso.")
        archivos = listar_archivos(id_folder_excel, service_drive)
        start_time = time.time()
```

```
        #Generar certificados paralelo
        generar_certificados_paralelo(id_folder_excel, id_drive_destino, service_drive, creds)
        end_time = time.time()
        elapsed_time = end_time - start_time

        logger.info(f"Tiempo de ejecución paralelo: {elapsed_time:.2f} segundos")

        subir_archivo_log(service_drive, os.path.basename(log_filename), log_filename, id_drive_destino)
    else:
        logger.warning("No se encontró el archivo de credenciales. Generando uno nuevo...")
        auth()
```

Anexo 11: Código principal del módulo NodeJS

```
// main.js
import express from "express";
import dotenv from "dotenv";
import { google } from "googleapis";
import fs from 'fs';
import path from 'path';
import fetch from 'node-fetch';  //Realizar peticiones al servidor
import Bottleneck from 'bottleneck'; // Para validar el limite de solicitudes a la Api de Google
import winston from 'winston'; // Para los registros en logs

import {
  getAuthUrl,
```

```
  handleOAuthCallback,
  loadSavedCredentials,
} from "./auth.js";

dotenv.config();

const app = express();
const PORT = process.env.PORT || 8000;

const drive_sheets = process.env.DRIVE_SHEETS;
const drive_pdf = process.env.DRIVE_PDF;

let driveClient;
// Asegurarse de que la carpeta logs exista
const logDir = 'logs';
if (!fs.existsSync(logDir)) {
    fs.mkdirSync(logDir);
}

// Configuración de Bottleneck
const limiter = new Bottleneck({
    minTime: 110, // Añadir un retardo mínimo de 110 ms entre solicitudes
});
// Añadir contadores y listas para el seguimiento
let contador_archivos_convertidos = 0;
let contador_conversion_fallidos = [];
```

```
// Función para generar un nombre de archivo con la fecha y hora actual
function generateLogFileName(baseName) {
    const now = new Date();
    const timestamp = now.toISOString().replace(/[:.]/g, '-');
    return path.join(logDir, `${baseName}_${timestamp}.log`);
}

// Configuración de Winston para los registros
const logger = winston.createLogger({
    level: 'info',
    format: winston.format.combine(
        winston.format.timestamp({ format: 'YYYY-MM-DD HH:mm:ss' }),
        winston.format.printf(({ timestamp, level, message }) => `${timestamp} ${level}: ${message}`)
    ),
    transports: [
        new winston.transports.Console(),
        new winston.transports.File({ filename: generateLogFileName('Registros') }),
    ],
});

async function initializeDriveClient(auth) {
    if (!driveClient) {
        driveClient = google.drive({ version: 'v3', auth });
        logger.info('Cliente de Google Drive inicializado');
```

```
    }
    return driveClient;
}

// Crear una carpeta en Google Drive
async function createFolder(drive, driveDestino) {

    const now = new Date();
    const folderName = `Ejecucion_${now}`
    try{
        const fileMetadata = {
            name: folderName,
            mimeType: 'application/vnd.google-apps.folder',
            parents: [driveDestino],
        };

        const folder = await drive.files.create({
            resource: fileMetadata,
            fields: 'id',
        });
        logger.info(`Carpeta ${folderName} creada`);

        return folder.data.id; // Retornar el ID de la carpeta creada

    }catch(error){
        logger.error("Error al crear carpetas de spreadsheets: ", error);
```

```
    }
}

// Ruta para iniciar el proceso de autenticación
app.get("/auth/google", (req, res) => {
    const authUrl = getAuthUrl();
    res.redirect(authUrl);
});

// Ruta para manejar el callback de OAuth2
app.get("/google/redirect", async (req, res) => {
    const code = req.query.code;
    if (code) {
        try {
            await handleOAuthCallback(code);
            res.send("Autenticacion exitosa! Puedes cerrar esta ventana.");
        } catch (error) {
            logger.error("Error al recuperar el token de acceso", error);
            res.send("Error al recuperar el token de acceso");
        }
    } else {
        logger.info("No se proporcionó un codigo de acceso");
        res.send("No se proporcionó un codigo de acceso");
    }
});
```

```
async function listFiles(auth) {
    try {
        //Inicializar cliente de Google Drive
        const drive = await initializeDriveClient(auth);
        // Llamada a la API para listar archivos
        const response = await drive.files.list({
            q:`'${drive_sheets}' in parents and mimeType='application/vnd.google-apps.spreadsheet' and trashed=false`, // carpeta sheets
            pageSize: 30, // Número de archivos que deseas listar (puedes ajustar esto)
            fields: "files(id, name)", // Campos que deseas obtener de cada archivo
        });

        // Extraer la lista de archivos de la respuesta
        const files = response.data.files;

        // Comprobar si se encontraron archivos
        if (files.length === 0) {
            logger.info("No se encontraron archivos.");
            return [];
        } else {
            logger.info(`Archivos SpreadSheets a procesar: ${files.length}`);
            files.forEach((file) => {
                logger.info(`ID: ${file.id}, Nombre: ${file.name}`);
            });
            return files; // Devuelve la lista de archivos
        }
```

```
    } catch (error) {
        logger.error("Error al listar archivos de Google Drive:", error);
        throw error;
    }
}

// Convertir cada hoja de un spreadsheet a PDF y guardarla en la carpeta correspondiente
async function convertToPDF(auth, fileId, folderId, fileName) {
    try {
        const drive = await initializeDriveClient(auth);

        //Exportar archivo pdf
        const exportParams = {
            fileId: fileId,
            mimeType: 'application/pdf',
        };
        const exportedFile = await drive.files.export(exportParams, { responseType: 'stream'
});
        // 2. Crear un nuevo archivo en Drive para el PDF
        const createParams = {
            resource: {
                name: `${fileName}.pdf`, // Nombre del archivo PDF
                parents: [folderId], // Carpeta de destino
                mimeType: 'application/pdf',
            },
            media: {
```

```
            mimeType: 'application/pdf',
            body: exportedFile.data, // Contenido del PDF exportado
        },
        fields: 'id', // Solo necesitamos el ID del archivo creado
    };
    const createdFile = await drive.files.create(createParams);

    logger.info(`Archivo ${fileName} convertido a PDF`);
    contador_archivos_convertidos++;
    return createdFile.data.id; // Devuelve el ID del PDF creado
  } catch (error) {
    logger.error(`Error al convertir el archivo ${fileName} a PDF:`, error);
    contador_conversion_fallidos.push(fileName);
  }
}

// Proceso completo de conversión de spreadsheets y hojas
async function processSpreadsheets(auth) {
  const driveClient = await initializeDriveClient(auth);
  const files = await listFiles(auth);

  const startTime = process.hrtime(); // Marca de tiempo inicial
  const folderId = await createFolder(driveClient, drive_pdf); // Crear carpeta para cada spreadsheet
  logger.info(`Cantidad de archivos a procesar: ${files.length}`);
  try{
```

```
        //Procesamiento paralelo
        const conversionPromises = files.map(async (file) => {
            return convertToPDF(auth, file.id, folderId, file.name)
        });

        // Esperar a que todos los spreadsheets sean procesados
        await Promise.all(conversionPromises);

        const endTime = process.hrtime(startTime); // Marca de tiempo final
        const elapsedTime = endTime[0] + endTime[1] / 1e9; // Tiempo en segundos
        logger.info(`Tiempo total de ejecución del programa: ${elapsedTime.toFixed(2)} segundos`);
        logger.info(`Cantidad de archivos convertidos a PDF: ${contador_archivos_convertidos}`);
        logger.info(`Cantidad de archivos que no se convirtieron a PDF: ${contador_conversion_fallidos.length}`);

        if (contador_conversion_fallidos.length > 0) {
            logger.info(`Archivos que no se convirtieron a PDF: ${contador_conversion_fallidos.join(', ')}`);
        }

    }catch(error){
        logger.error("Error al procesar spreadsheets: ", error);
    }
}
```

```
// Iniciar el servidor
app.listen(PORT,async () => {
    logger.info(`Server ejecutandose en el puerto ${PORT}`);

    // Autenticarse
    const auth = await loadSavedCredentials();

    if(auth){
        await processSpreadsheets(auth); // Iniciar el procesamiento
        /*server.close(() => {
            logger.info('Server cerrado después de completar el procesamiento.');
            process.exit(0); // Salir del proceso Node.js
        });*/
    }else{
        logger.info("Necesitas autenticarte primero");
    }

});
```

Anexo 12: Código principal del módulo F#

```
open System
open GoogleDrive.GoogleDriveService
open GoogleDrive.Config
open Google.Apis.Drive.v3
open System.Threading.Tasks
open System.IO
```

```
open System.Collections.Concurrent
open LogConfig
let crear_carpeta (nombreCarpeta: string) (driveDestino: string) (service: DriveService): string =
    try
        let fileMetadata = new Google.Apis.Drive.v3.Data.File()
        fileMetadata.Name <- nombreCarpeta
        fileMetadata.MimeType <- "application/vnd.google-apps.folder"
        fileMetadata.Parents <- [| driveDestino |]

        let request = service.Files.Create(fileMetadata)
        request.Fields <- "id"
        let carpeta = request.Execute()
        printfn "Se creó la carpeta: %s" nombreCarpeta
        carpeta.Id
    with
    | ex ->
        printfn "Ocurrió un error al crear la carpeta: %s" ex.Message
        null
let listar_archivos idfolderSheet (service: DriveService): Google.Apis.Drive.v3.Data.File list=
    try
        let query = sprintf "'%s' in parents and mimeType='application/vnd.google-apps.spreadsheet' and trashed=false" idfolderSheet

        let request = service.Files.List()// Definir la solicitud para listar archivos
        request.Q <- query
```

```
        request.PageSize <- 11
        request.Fields <- "nextPageToken, files(id, name, mimeType)"

        // Ejecutar la solicitud y obtener los archivos
        let result = request.Execute()
        let files = result.Files

        if files = null || files.Count = 0 then
            printfn "No hay archivos."
            []
        else
            files |> Seq.toList  // Convertir a una lista de F#
    with
    | ex ->
        printfn "Ocurrió un error al listar los archivos: %s" ex.Message
        []

let convertir_pdf (archivo: Google.Apis.Drive.v3.Data.File) driveDestino (service: DriveService) (convertidosConExito: ConcurrentBag<unit>)(fallos: ConcurrentBag<Google.Apis.Drive.v3.Data.File>) =
    let archivoId = archivo.Id
    let nombreArchivo = archivo.Name
    let request = service.Files.Export(archivoId, "application/pdf")

    use pdfIo = new MemoryStream()
```

```
    try
        // Descargar el archivo directamente en el MemoryStream
        request.DownloadWithStatus(pdfIo) |> ignore
        pdfIo.Position <- 0L

        //El buffer contiene el archivo en memoria
        let fileMetadata = new Google.Apis.Drive.v3.Data.File()
        fileMetadata.Name <- nombreArchivo + ".pdf"
        fileMetadata.Parents <- [| driveDestino |]

        let fileMetadata = new Google.Apis.Drive.v3.Data.File()
        fileMetadata.Name <- nombreArchivo + ".pdf"
        fileMetadata.Parents <- [| driveDestino |]

        let uploadRequest = service.Files.Create(fileMetadata, pdfIo, "application/pdf")
        uploadRequest.Fields <- "id"
        uploadRequest.Upload() |> ignore
        printfn "%s convertido a PDF." nombreArchivo

        convertidosConExito.Add()
    with
    | ex ->
        fallos.Add(archivo)
        printfn "Ocurrió un error al convertir_pdf: %s" ex.Message

// Generar Certificados Paralelo
```

```
let generar_certificados_paralelo driveExcel drivePDF service =
    try
        let archivos = listar_archivos driveExcel service
        let totalArchivos = List.length archivos

        //Contadores
        let convertidosConExito = ConcurrentBag<unit>()
        let fallos = ConcurrentBag<Google.Apis.Drive.v3.Data.File>()

        if List.isEmpty archivos then
            printfn "No hay archivos para procesar."
        else
            printfn "Cantidad de archivos a procesar: %d" totalArchivos

            let opciones = ParallelOptions(MaxDegreeOfParallelism = 10)
            Parallel.ForEach(archivos, opciones, (fun archivo ->
                convertir_pdf archivo drivePDF service convertidosConExito fallos
            )) |> ignore

        printfn "Cantidad de archivos convertidos a PDF: %d" (convertidosConExito.Count)
        printfn "Cantidad de archivos no convertidos a PDF: %d" (fallos.Count)

        if fallos.Count > 0 then
            printfn "Archivos que no se convirtieron a PDF:"
            for archivo in fallos do
                printfn "Nombre: %s" archivo.Name
```

```
    with
    | ex ->
        printfn "Ocurrió un error al generar certificado: %s" ex.Message

let subirArchivoLog (rutaLog: string) (driveDestino: string) (service: DriveService) =
    try
        use stream = new FileStream(rutaLog, FileMode.Open)

        let fileMetadata = new Google.Apis.Drive.v3.Data.File()
        fileMetadata.Name <- Path.GetFileName(rutaLog)
        fileMetadata.Parents <- [| driveDestino |]

        let uploadRequest = service.Files.Create(fileMetadata, stream, "text/plain")
        uploadRequest.Fields <- "id"
        let file = uploadRequest.Upload()

        if file.Status = Google.Apis.Upload.UploadStatus.Completed then
            printfn "El archivo log fue subido exitosamente: %s" fileMetadata.Name
        else
            printfn "Hubo un problema al subir el archivo log: %s" file.Exception.Message
    with
    | ex -> printfn "Ocurrió un error al subir el archivo log: %s" ex.Message


[<EntryPoint>]
let main argv =
```

```
let logFilePath = LogConfig.configureLogging()

let idFolderExcel = idDriveSheets // Obtener el ID de la carpeta con sheets
let idFolderDestino = idDriveDestino // Obtener el ID de la carpeta con PDFs
let service = createDriveService ()//Crear servicio Google Drive

let fechaActual = DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss")
let nombreCarpetaEjecucion = sprintf "Ejecucion_%s" fechaActual
let idCarpetaPDF = crear_carpeta nombreCarpetaEjecucion idFolderDestino service

if isNull service then
    printfn "No se pudo iniciar sesión en Google Drive."
    1 // devolver un código de error
else
    let startTime = DateTime.Now

    //Generar certificados paralelo
    generar_certificados_paralelo idFolderExcel idCarpetaPDF service
    subirArchivoLog logFilePath idCarpetaPDF service

    let endTime = DateTime.Now
    let elapsedTime = endTime - startTime

    printfn "Tiempo de ejecución: %f segundos" elapsedTime.TotalSeconds
    0
```

Anexo 13: Resultados de las pruebas de rendimiento

Lenguajes / Tiempos	1000 archivos	3000 archivos	6000 archivos
Golang	212584 ms	632593 ms	1265711 ms
NodeJS	281408 ms	812977 ms	1711342 ms
Python	377980 ms	925647 ms	1911344 ms
F#	341355 ms	1019037 ms	2192308 ms

Anexo 14: Entrevista al Jefe de Transformación Digital

Entrevista realizada

• Entrevistador: Kenny Pizarro Luzón

• Entrevistado: Jefe de transformación Digital

• Fecha: lunes, 04 de noviembre de 2024

• Duración: Aproximadamente 10 - 15 minutos

Entrevistador: Muchas gracias por aceptar esta entrevista. Para empezar, ¿podría describir brevemente la estructura organizacional de la compañía en su sede de Guayaquil?

Jefe de Transformación Digital: Claro, la empresa tiene una estructura bastante organizada para garantizar la gestión eficiente de sus operaciones. En la sede de Guayaquil contamos con diversas direcciones que abarcan áreas clave como Dirección General, Dirección Comercial, Dirección Administrativa Financiera, Dirección Técnica, Dirección de Operaciones, Dirección de Recursos Humanos, Dirección de Operaciones Comerciales, Dirección de Servicios Industriales y Ambientales, Gerencia de Digital Business & Technology (DB&T) y Gerencia Legal.

Entrevistador: Entendido. ¿Podría explicarnos un poco más sobre la estructura interna de la Gerencia de DB&T?

Jefe de Transformación Digital: Por supuesto. La Gerencia de DB&T está formada por varias áreas especializadas: Desarrollo y Aplicaciones, Administración de Seguridades, Control y Calidad, Administración de Servidores y el área de Business Intelligence. Yo me encuentro en el área de Control y Calidad, que supervisa el correcto funcionamiento y la eficiencia de los sistemas implementados, además de evaluar las oportunidades de mejora tecnológica.

Entrevistador: ¿podría explicarnos brevemente cuáles son las principales responsabilidades del área de Control y Calidad en esta empresa?

Jefe de Transformación Digital: Por supuesto. Nuestra área se encarga de liderar la implementación de tecnologías innovadoras para optimizar los procesos internos de la empresa. Esto incluye digitalizar operaciones críticas, mejorar la eficiencia operativa mediante herramientas tecnológicas, gestionar grandes volúmenes de datos y garantizar la integración tecnológica entre las diferentes direcciones de la compañía.

Entrevistador: También nos gustaría entender la estructura de la Dirección Técnica. ¿Qué áreas clave están bajo esta dirección?

Jefe de Transformación Digital: La Dirección Técnica está compuesta por cuatro gerencias principales: la Gerencia de Control de Proyectos (PMO), la Gerencia de Sostenibilidad, la Gerencia de Obras y la Gerencia de Estudios y Diseños. Cada gerencia tiene un enfoque específico. Por ejemplo, la Gerencia de Sostenibilidad, donde se desarrolla el objeto de estudio, es responsable de proyectos relacionados con el manejo de recursos y el impacto ambiental de nuestras operaciones.

Entrevistador: ¿Cuál considera que es la mayor problemática digital que enfrenta actualmente el área de Sostenibilidad?

Jefe de Transformación Digital: Una de las principales problemáticas es la lentitud en el procesamiento y la generación de reportes trimestrales en formato PDF. Al trabajar con grandes volúmenes de datos, el tiempo de respuesta en la generación de reportes es crucial para la toma de decisiones. Actualmente, el sistema hecho en Google AppsScript carece de la capacidad para procesar simultáneamente múltiples tareas de forma eficiente, lo que genera cuellos de botella.

Entrevistador: En cuanto al proceso automatizado para la generación de reportes trimestrales en la Gerencia de Sostenibilidad, ¿cuál es su estado actual?

Jefe de Transformación Digital: Actualmente, existe un proceso automatizado para generar los reportes trimestrales en formato PDF, principalmente sobre los fondos propios de los diferentes tipos de alcantarillado. Sin embargo, aunque la automatización ha reducido la carga manual, el sistema es lento y susceptible a errores. En ocasiones, las interrupciones obligan a reiniciar el proceso desde el principio, lo que afecta nuestra eficiencia operativa y aumenta el riesgo de retrasos en la entrega de informes.

Entrevistador: Finalmente, ¿cómo considera que esta solución impactaría en la productividad general de la empresa?

Jefe de Transformación Digital: Estoy convencido de que tendría un impacto muy positivo. No solo optimizaríamos nuestras operaciones internas, sino que también podríamos mejorar la satisfacción de nuestros clientes al cumplir con los tiempos de entrega establecidos. Esto nos permitiría fortalecer nuestra posición como líderes en soluciones medioambientales y afrontar con mayor confianza proyectos más complejos en el futuro.

Anexo 15: Entrevista al Técnico de Tecnología y Desarrollo

Entrevista realizada

• Entrevistador: Kenny Pizarro Luzón

• Entrevistado: Técnico de Desarrollo y Tecnología

• Fecha: lunes, 19 de agosto de 2024

• Duración: Aproximadamente 15 - 20 minutos

Entrevistador: Para comenzar, ¿podría explicarnos cómo se lleva a cabo el proceso actual de generación de reportes trimestrales en formato PDF?

Técnico de Desarrollo y Tecnología: Claro. El proceso comienza con la recopilación y organización de datos en hojas de cálculo, que se realizan manualmente en Google Sheets. Una vez que los datos están listos, utilizamos un script desarrollado en Google Apps Script para automatizar la conversión de estas hojas de cálculo a archivos PDF. Luego, los PDF generados se almacenan en carpetas organizadas en Google Drive, desde donde se revisan y entregan a la gerencia correspondiente.

Entrevistador: ¿Cuáles considera que son las principales limitaciones del proceso actual?

Técnico de Desarrollo y Tecnología: Hay varias limitaciones importantes. Una de las principales es el tiempo de ejecución: Google Apps Script tiene un límite de 30 minutos por proceso, lo cual es un gran problema cuando se manejan grandes volúmenes de archivos. Esto genera interrupciones frecuentes, y en ocasiones, el script se detiene antes de completar el trabajo. Además, los resultados pueden ser inconsistentes, ya que algunos archivos PDF generados pueden salir incompletos o vacíos, lo que afecta la calidad de los reportes finales.

Entrevistador: ¿Qué sucede cuando el proceso se detiene antes de completarse?

Técnico de Desarrollo y Tecnología: Cuando esto ocurre, debo revisar manualmente los archivos generados para identificar dónde se interrumpió el proceso. Luego, tengo que ajustar y reiniciar el script para completar los reportes restantes. Este procedimiento manual consume tiempo y esfuerzo, además de impactar negativamente en la productividad.

Entrevistador: Desde su perspectiva, ¿cómo afecta este proceso a la escalabilidad de la generación de reportes?

Técnico de Desarrollo y Tecnología: El sistema actual no es escalable. Si el volumen de archivos aumenta, el proceso se vuelve mucho más lento y propenso a errores. Por ejemplo, si se generan miles de reportes, el tiempo necesario para completar la tarea se incrementa significativamente, y las interrupciones se vuelven más frecuentes. Esto limita la capacidad del sistema para manejar una mayor carga de trabajo.