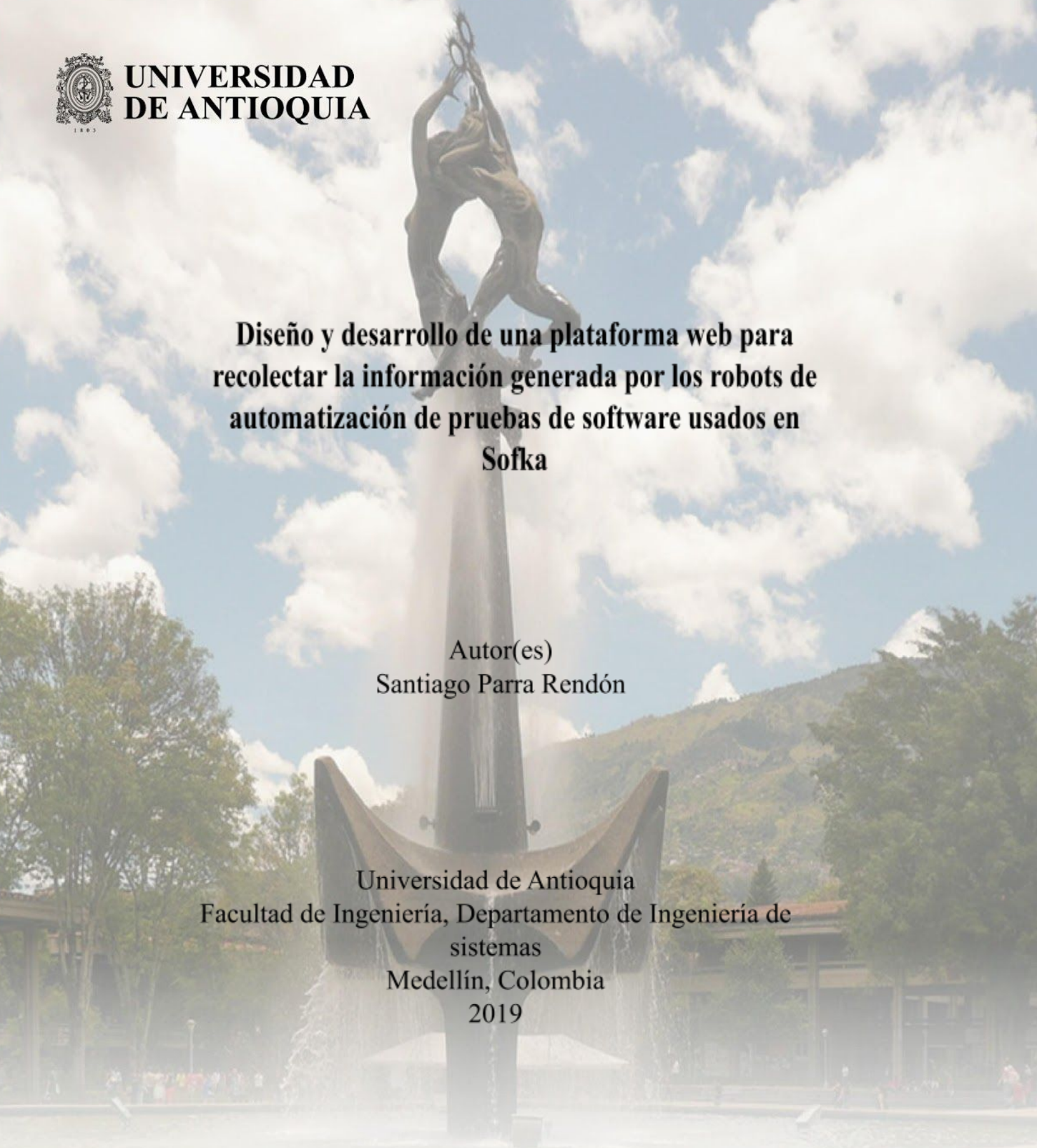

UNIVERSIDAD DE ANTIOQUIA

# Diseño y desarrollo de una plataforma web para recolectar la información generada por los robots de automatización de pruebas de software usados en Sofka


Autor(es)
Santiago Parra Rendón

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería de sistemas
Medellín, Colombia

2019

# Diseño y desarrollo de una plataforma web para recolectar la información generada por los robots de automatización de pruebas de software usados en Sofka

Santiago Parra Rendón

Informe de práctica como requisito para optar al título de:
Ingeniero de Sistemas.

Asesores (a) o Director(a) o Co- Directores(a).

Jeysson Pérez Gómez
Ingeniero de Sistemas
Especialista en Gerencia Integral

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería de sistemas
Medellín, Colombia
2019.

**Diseño y desarrollo de una plataforma web para recolectar la información generada por los robots de automatización de pruebas de software usados en Sofka**

**Resumen.**

En este informe se muestra de manera práctica la implementación de una arquitectura basada en microservicios con NodeJs y Express teniendo como base el lenguaje de programación Javascript sobre un proyecto real cuyo objetivo es la centralización de la información de pruebas de software en una empresa para incrementar el valor de la misma bajo el marco de trabajo SCRUM. Partiendo de la hipótesis se logra construir un producto conforme a los estándares de calidad planteados por la compañía donde en cada etapa estuvieron involucrados todas las partes para moldear el producto según las necesidades reales donde todos lograron entender la idea del producto y cómo se debían cumplir los objetivos planteados, obteniendo resultados altamente satisfactorios, los cuales se lograron al validar y fallar de forma temprana, montando 4 módulos que permiten recolectar la información en robots basados en el lenguaje Java, una estructura de Back-end compuesta de seis microservicios bajo un ciclo de vida DevOps lo cual permite mantener un flujo claro de todos los procesos involucrados desde la concepción de una idea hasta su materialización al ser implementado en un ambiente productivo y un Front-end desarrollado en Angular con la filosofía de mobile-first cuyo objetivo es acceder a más usuarios.

**Introducción.**

Sofka es una empresa que se dedica al desarrollo de sistemas informáticas, desde su creación, pruebas y soporte, en el año 2017 se funda el área de calidad, con la visión de proporcionar un servicio de testing altamente competitivo y con un alto nivel técnico, siguiendo esta visión el centro de excelencia técnica del área fundado a finales del 2018, inicia un proceso de automatización de procesos internos, identificando que el monitoreo constante a nivel de calidad de los proyectos se definen por resultados recolectados de forma manual, buscando la automatización de procesos internos se toma la decisión de crear una plataforma web centralizada, donde todos los resultados generados por los robots de automatización sean guardados de manera automática y cada persona involucrada en un proyecto pueda ver la información que desea según sus necesidades.

**Objetivos**

- **General:** Diseñar y desarrollar una plataforma web, para recolectar la información generada por los robots de automatización de pruebas de software usados en Sofka, mediante el uso de metodologías ágiles.
- **Específicos**:
    - Desarrollar los 2 módulos de recolección y envío de la información para poder almacenarla y mostrarla a los usuarios.
    - Diseñar la base de datos para poseer persistencia en los datos y soportar la complejidad de las relaciones entre los datos que tiene la información.
    - Desarrollar el microservicio de escritura de datos, para poder escribir la información recibida de los módulos de recolección y guardarla en base de datos.
    - Desarrollar el microservicio de lectura de datos, para que los usuarios de la plataforma puedan acceder a la información recolectada.
    - Diseñar la interfaz responsive de la plataforma para que cada usuario pueda acceder a la información desde cualquier dispositivo con acceso a internet, logrando una buena experiencia de usuario.

**Planteamiento del Problema.**

La información generada por una empresa sobre la calidad de un producto o proyecto elaborado tanto de forma interna como externa se convierten en herramientas que pueden ser usadas para la toma de decisiones o validar el cumplimiento de los acuerdos y objetivos de un equipo, la información generada específicamente por los robots de automatización de pruebas de software generan información entendible en primera instancia para los automatizadores a los cuales les genera un valor agregado al facilitarles el entendimiento de un ¿Qué? y ¿Cómo? se manifestó un fallo y tomar decisiones en base a los requerimientos del proyecto, sin embargo, cuando esta información debe ser escalada a otros roles como los gerentes de proyectos y líderes de calidad, los automatizadores deben generar reportes adicionales generando una carga de trabajo adicional, abriendo adicionalmente una brecha de sesgo de la información que puede ser ocasionados incluso por factores intencionales, finalmente la información generada y almacenada por cada uno de los automatizadores suponía otro problema adicional donde se crea una dependencia en una persona para acceder a la información.

**Marco Teórico.**

La realización de pruebas de software es un proceso fundamental para asegurar la calidad de los proyectos de software (Mera, 2016) al reducir la probabilidad de presencia de fallos (Hetzel, 1993), mediante los resultados obtenidos en la ejecución de las pruebas (Bertolino, 2007), estos resultados son usados como evidencia y soporte para la toma de decisiones (ISTQB, 2012) siendo uno de los pasos del proceso de pruebas (Müller, 2013), usado por las compañías que quieren asegurar un nivel de calidad en todos sus proyectos.
Al profundizar un poco más en el mundo de las pruebas de software se pueden encontrar dos formas de hacer pruebas: manual y automática, ambas buscan reducir el tiempo y dinero invertido en pruebas (ISTQB, 2012), sin embargo, las pruebas manuales que deben realizarse constantemente generan altos costos en tiempo, debido a que implican la realización continua de la misma tarea (Esmite, 2007), adicionalmente las metodologías de trabajo tradicionales suponían un gran problema para el aseguramiento de calidad dado que las pruebas se realizaban al final del proceso, violando uno de los principios de pruebas: "Pruebas tempranas" (ISTQB, 2012), es por tal motivo que las metodologías ágiles al trabajar bajo el marco iterativo e incremental, le permiten a los aseguradores de calidad de un producto o proyecto (Testers) trabajar en conjunto con los equipos a lo largo de todo el ciclo de vida (CEU IAM, 2018), al unir el agilismo con la automatización de pruebas, se abre la posibilidad de que los involucrados en un proyecto sepan cómo se comporta el proyecto a nivel de calidad desde su inicio, de forma más rápida y evitando la tarea de ejecutar las pruebas manuales cada que haya un cambio crítico (Bertolino, 2007).

Al usar metodologías ágiles en los proyectos de software la infraestructura, desarrollo y pruebas a usar deben acoplarse a sus principios, por tal motivo la infraestructura orientada a microservicios se acopla fácilmente a estas metodologías debido a la flexibilidad que ofrece cuando se tienen reglas de negocio cambiantes; Los microservicios son pequeñas unidades escalables que cumplen una única función y pueden comunicarse fácilmente entre sí al componerse de pequeñas unidades escalables (Pérez, 2015), cuyo fin es el de cumplir una única función específica, lo cual es ideal para los proyectos que requieran múltiples componentes (Pérez, 2015).

Los datos usados en las pruebas son de gran valor para un producto, esto se debe a que un buen conjunto de datos puede aportar o encontrar comportamientos anormales de los sistemas, sin embargo, si los datos usados son estáticos el Tester estaría probando una y otra vez lo mismo incumpliendo uno de los principios de las pruebas "Paradigma del pesticida" donde se explica que si un sistema se comprueba siempre bajo las mismas condiciones llegará a un punto donde no va a fallar y las pruebas pierden su valor (ISTQB, 2012), este principio fundamenta los datos generados de forma dinámica que buscan someter al sistema a diferentes situaciones y observar su comportamiento en cada una de ellas(Michael, 2001), estos datos deben ser almacenados dado que van a permitir replicar situaciones anormales o en las cuales el sistema manifieste un bug. En el mercado existen actualmente múltiples opciones para el almacenamiento de datos que pueden ser generados en diferentes formatos y depende de cada compañía el diseñar e implementar una estrategia para su uso, en ocasiones las herramientas pueden ser internas para aquellas que busquen tener más control sobre sus datos.

Las herramientas de software son de gran ayuda en las pruebas, lo que implica que una buena selección e implementación de las mismas influye de forma directa(ISTQB, 2012) en los resultados obtenidos al verificar la calidad de un producto, los proyectos tiene diferentes necesidades que se pueden plantear desde su concepción al plantear los objetivos, limitaciones, presupuesto y generar todo un ecosistema de ideas donde debe quedar de forma clara y concisa el ¿Qué? y ¿Cómo?, cada uno de estos factores influyen en la selección de las herramientas a usar (ISTQB, 2012), sin embargo, al plantear las herramientas a usar la selección no depende únicamente de las necesidades del proyecto, a su vez existe también una dependencia con el conocimiento y habilidades que posee el equipo encargado del proyecto sobre las herramientas, es importante tener presente que el mal uso de estas herramientas por falta de conocimiento puede ocasionar que los objetivos no sean cumplidos de forma óptima y efectiva.

Las herramientas de recolección de información no deben afectar el resultado de las pruebas, fundamentados en que dichas herramientas no son el foco objetivo de las pruebas (ISTQB, 2012), por ende al implementar una herramienta se debe asegurar un principio de no intervención y alta disponibilidad, ambas características se deben satisfacer de manera funcional y no funcional (Hetzel, 1993). Cuando el nicho objetivo de las herramientas es considerablemente alto el Performance es un factor clave que puede influir en una recolección exitosa o desastrosa de la información(Khare, 2003).

El conocimiento de los integrantes de un proyecto puede ocasionar que la selección del mismo se base en la experiencia que tengan sobre las tecnologías a usar, e inclusive se creen espacios para que las puedan conocer y aprender, un claro ejemplo de esta última son los spike en el marco de trabajo SCRUM (CEU, 2018), el cual es un espacio generado en el proyecto para que todos los integrantes se puedan familiarizar con las herramientas y despejar las dudas que posean antes de emplearlas de forma real sobre el proyecto, asegurando una correcta implementación de las herramientas seleccionadas, sin embargo, estas herramientas no solo benefician a quienes se encargan de desarrollar los productos, aquellos que se encargan de velar por el cumpliemiento de buenos estandares de calidad también pueden hacer uso de ellas, un ejemplo claro es Cucumber y los *Domain Specific Language* DSL, los cuales buscan llevar el planteamiento de los flujos de comportamiento de los aplicativos a un lenguaje que todos los involucrados en un proyecto puedan entender, evitando pérdida o una mala interpretación de los flujos, adicionalmente le permite a los Tester poder transmitir sus preocupaciones a los equipos de desarrollo de forma clara y concisa al tener una base sólida la cual fue definida al planear el funcionamiento del producto.

## Metodología

Se realizaron todas las ceremonias usadas en el marco de trabajo SCRUM, donde se implementaron sprints con una duración de 4 semanas, las cuales tuvieron como objetivo cada uno de los objetivos específicos planteados, cada Sprint estuvo compuesto por:

- Planning: Se realizaron al inicio de cada sprint donde se dio respuesta a ¿Qué se realizará en el sprint?
- Daily: Se realizaron al inicio de cada dia laboral con el propósito de informar al equipo de trabajo sobre los avances y/o dificultades encontradas el día anterior.
- Review: Su objetivo era el permitirle al equipo organizar los avances y presentarlos a los inversionistas al finalizar el sprint.
- Retrospective: Se realizaron al final de cada sprint y tenían como objetivo reunir al equipo de trabajo, hablar y definir puntos a mejorar para el siguiente sprint.
- Refinamiento: Se realizaron en el transcurso de un sprint cuando algunos de los objetivos definidos en el planning quedaban inconclusos.

## Resultados y análisis.

La concepción de una idea que tiene como objetivo darle valor a las evidencias generadas por los robots de automatización, en un contexto que puedan entender todos los involucrados en un proyecto de software empezó a cobrar forma al realizar una reunión donde diferentes roles y cargos (Quality Head, Technical leader, Project manager) pudieron aportar sus opiniones y fortalecer la idea planteada al materializar sobre papel, al tener claro el camino a seguir se empezó la búsqueda de fondos y selección de las personas que estarían involucradas las cuales debían poseer una visión de calidad y entender en su totalidad lo que se quería lograr, lo cual se logró conseguir por medio de diferentes actividades como la creación de un Elevator Pitch, caja del producto, entre otras.

Al tener un equipo sólido se comienza el proceso de materialización de los diferentes objetivos y desarrollarlos, el primer paso a realizar era definir una estructura, infraestructura y tecnologías a usar, la cual fue definida de la siguiente manera:

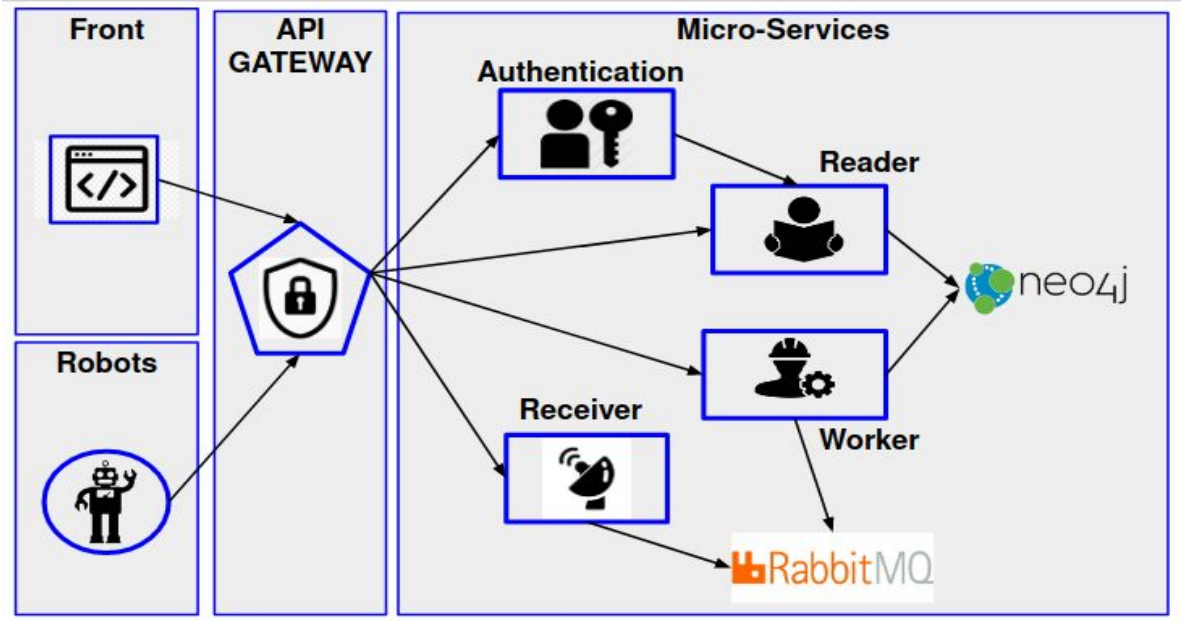


**Imagen 1. Infraestructura base del proyecto.**

Al tener una guía clara del comportamiento en general del sistema se dió inicio al proceso de definición de tareas para 3 capas definidas de la siguiente manera:

1. API para la recolección de la información en diferentes versiones de Cucumber (Tecnología objetivo del proyecto para los robot de automatización).
2. Back-end encargado de procesar todas las solicitudes de los clientes y de las API.
3. Front-end encargado de mostrar los resultados a los usuarios.

Cada capa se compone a su vez de subtareas la cuales fueron asignadas a cada integrante del equipo según preferencias y conocimientos de los mismos. La primera etapa requería que las ejecuciones realizadas por los automatizadores del área de calidad de la empresa fueran capaz de recolectar toda la información generada sin afectar el performance ni los resultados de las ejecuciones, para lograr este objetivo se definió entonces la creación de un Listener cuyo objetivo es recolectar información sin interferir en la ejecución, posteriormente se procedió a analizar las versiones de Cucumber que entraron en el nicho al analizar su uso dentro de la compañía, de lo cual se obtuvo:

| | Cucumber 1 | Cucumber 2 | Cucumber 3 | Cucumber 4 |
|---|---|---|---|---|
| Uso(%) | 67% | 9% | 0% | 24% |

**Tabla 1. Relación de uso por cada versión de Cucumber en la compañía.**

Al hacer el análisis de uso se decidió tener como objetivo principal los módulos de recolección (Listener) para las versiones 1 y 4 de Cucumber, sin embargo, se identificó que el desarrollo y mantenimiento de ambos módulos sería bastante complejo en el transcurso del tiempo, razón de peso que llevó al equipo involucrado a analizar las diferentes posibilidades, donde se concluyó que lo más óptimo era encontrar similitudes entre las versiones para generar artefactos con la menor cantidad de diferencias posibles.
Al analizar la estructura de recolección de datos generados por cucumber se encuentra que existe una relación entre diferentes versiones que puede ser representada por:

| Similitud | Cucumber 1 | Cucumber 2 | Cucumber 3 | Cucumber 4 |
|---|---|---|---|---|
| Cucumber 1 | 100% | 0% | 0% | 0% |
| Cucumber 2 | 0% | 100% | 100% | 99% |
| Cucumber 3 | 0% | 100% | 100% | 99% |
| Cucumber 4 | 0% | 99% | 99% | 100% |

**Tabla 2. Relación de líneas de código usables entre las diferentes versiones de Cucumber**

Estos valores parten de que las versiones 2, 3 y 4 son absolutamente diferentes en su arquitectura en comparación con la versión 1, adicionalmente se logra conceptualizar que las versiones 2 y 3 son iguales en su estructura e implementación, por último se identifica que la última versión (4) hace uso un disparador de eventos diferente a las versiones 2 y 3 el cual incluye nuevas funcionalidades a nivel de estructura, sin embargo, en su base sigue siendo la misma que las versiones anteriores, estos datos llevan a la conclusión del planteamiento de versiones padre, conocidas como versiones base que pueden ser reutilizadas más adelante, de este proceso surgen dos conocidas como v1 y basada en eventos:

- V1: Librería de recolección de información para aquellas tecnologías que no recolectan información de las pruebas en base a eventos, para el caso puntual la versión 1 de Cucumber.

- Basada en eventos: Librería de recolección de la información para aquellas tecnologías que basen la obtención de los datos en disparadores de eventos, para el caso puntual las versiones 2, 3 y 4 de Cucumber.

Tras la implementación de esta idea se logra implementar la V1 y se logra una homogeneidad para las versiones 2 y 3, adicionalmente usando la base de la versión 2 de Cucumber se implementa la v4 al modificar una línea de código, esto se logra haciendo uso del disparador de eventos nuevo para dicha versión pero sin hacer uso de ninguna nueva funcionalidad ofrecida por la misma, al concluir este objetivo se tienen dos versiones globales que pueden ser reutilizables para diferentes tecnologías y versiones de las mismas, adicionalmente se logra cumplir el objetivo de obtener dos módulos de recolección que incluso fueron ampliados a 4 sin esfuerzo adicional del equipo de trabajo, mejorando tiempos de corrección de bugs y tiempo empleado en mantenimiento dando por culminada la primera capa del proyecto y primer objetivo planteado en el proyecto.

La segunda capa del proyecto requiere recibir y moldear los datos generados en una estructura de relación que genere valor y mantenga una jerarquía, es decir, abordar el segundo objetivo planteado al definir la estructura de la base datos que se debe usar, esta tarea es un factor crucial en un proyecto, en el cual se debe conocer con exactitud la arquitectura y los objetivos que posee, esto impulsó al equipo a que el diseño de la misma se hiciese en conjunto con todos los involucrados en el proyecto en un lenguaje de comportamiento que todos pudiesen entender, del cual los desarrolladores podían extraer los elementos y relaciones. El uso de tal metodología fue de gran impacto, pues terminó siendo clave para posteriormente escoger el tipo y base de datos, lo cual nos llevó a identificar que el proyecto requería de una base de datos que pudiese soportar relaciones altamente complejas y diferentes para cada usuario, motivo que por sí solo nos llevó a escoger Neo4j como base de datos principal, al ser una base de datos orientada a grafos permite manejar de forma sencilla esa complejidad que requería el proyecto, sin embargo, el equipo de desarrollo tomó la decisión de validar a nivel técnico la estructura definida como un filtro adicional donde se buscó ayuda de personal adicional de la compañía con el objetivo de validar los elementos que componen los datos y sus relaciones definidas, al finalizar la reunión la estructura quedó definida por los siguientes elementos:

- Cliente

- Parámetro
- Preferencia
- Producto
- Proyecto
- Equipo
- Sprint
- Usuario
- Canal
- Notificación
- Plugin

Al tener una estructura clara de los elementos la capa del Back-end pudo entrar a analizar el proceso de lectura y escritura de los datos recibidos en la base de datos, afrontando de esta manera los objetivos que tenían como propósito la creación de los microservicios que cumplieran estos procesos conocidos como *Worker* encargado de la escritura y *Reader* encargado de la lectura, la concepción y definición de ambos microservicios se llevó a cabo de forma paralela donde en primera instancia era indispensable analizar todas las tecnologías viables y determinar cuáles usar para su creación, de lo cual se plantearon dos grandes opciones: Spring Boot y Express, para determinar cual de estos usar como base se creó una tabla cuyo objetivo era analizar la viabilidad de implementación de cada una al preguntarle a los involucrados en el conocimiento que poseían sobre diferentes aristas necesarias para lograr una correcta implementación:

| | Spring Boot | Express |
|---|---|---|
| Conocimiento del equipo | 10% | 80% |
| Conocimiento de integración con las otras tecnologías a usar | 20% | 90% |
| Conocimiento de pruebas | 70% | 40% |
| Conocimiento del lenguaje base | 80% | 90% |
| Viabilidad de disminuir las | 30% | 71% |

| | | |
|---|---|---|
| falencias de conocimiento en un Spike de dos semanas | | |

**Tabla 2. Relación de de viabilidad de implementación de cada tecnología.**

El análisis de la **Tabla 2** llevó al equipo a escoger a Express como base, al tener claro la tecnología a usar se dió inicio a la búsqueda de librerías que permitieran establecer conexión con la base de datos Neo4j, encontrando la librería que la Neo4j ofrece para Javascript, lenguaje base usado en ambos microservicios; Tras la primera implementación se estableció un flujo de promesas y sesiones las cuales permitían recibir una solicitud en forma de promesa, abrir una sesión en la base de datos de escritura o lectura según el microservicio usado y realizar la tarea deseada para finalmente resolver o rechazar la promesa dependiendo si la tarea ejecutada fue exitosa hubo un error asociado a la misma, sin embargo tras realizar pruebas de performance se encontró que la implementación no era eficaz para más de 30 usuarios realizando solicitudes de escritura al *Worker* de forma simultánea, esto llevó al equipo a plantear una nueva estrategia que pudiese soportar altos flujos transaccionales con un enfoque para ambos microservicios, para lo cual se hizo uso de las transacciones atómicas de Neo4j, manteniendo una sesión abierta para todos los usuarios, pero segmentando cada solicitud como una transacción independiente, tras la segunda implementación el resultado arrojado por las pruebas de performance estableció un límite de 10000 transacciones simultáneas soportadas en la escritura de datos y 20000 transacciones de lectura, para finalizar la construcción se implementó un verificador de solicitudes en ambos microservicios, cuyo propósito era analizar cada solicitud y pasarla por un filtro para verificar que poseen una estructura y los datos necesarios para realizar la solicitud evitando que las solicitudes sean rechazadas por errores del tipo estructural.

Al completar los objetivos de la segunda capa el equipo pudo enfocar sus esfuerzos en analizar y cumplir los objetivos de la tercera capa, afrontando el objetivo de diseñar de la interfaz responsive en Angular, dado que la experiencia de usuario es muy importante en especial en un proyecto donde se trabaja bajo la filosofía de que cada usuario vea la información según sus necesidades de una forma clara y concisa, para el cual se hizo uso de Bootstrap dado que los integrantes del equipo ya estaban familiarizados con esta tecnología usando una estrategia de pantallas, la cual consiste en desarrollar componentes de una pantalla o vista con ayuda de los diseñadores de la empresa y de forma inmediata validar

con personas que iban a tener los diferentes roles del aplicativo para comprobar de manera temprana que la información si fuese mostrada de forma entendible, este proceso tuvo gran impacto pues la mayoría de veces se realizaron cambios inclusive del 100% dado que la retroalimentación temprana arrojaba resultados poco satisfactorios, estas validaciones también se hacían para las vistas móviles dado que se trabajó con Mobile first, es decir, la vista móvil era la primera en implementarse y luego se procedía con las demás, esto mejoró los tiempos de desarrollo pues la abstracción de las vistas cada vez fue más sencilla al ir conociendo los intereses de los involucrados.

**Conclusiones.**

Se presentan en forma concreta y lógica los resultados del trabajo. Las conclusiones deben ser la respuesta a los objetivos o propósitos.

- La implementación de la estructura de microservicios basados en imágenes de Docker alojadas en la nube le permite al proyecto tener actualizaciones de manera sencilla y automática, entregando software de valor a los clientes de forma temprana y continua, además crea un entorno amigable de despliegue que permite crear auditoría de todo el proceso que posee un cambio desde su concepción hasta su implementación para todos los involucrados en el proyecto.
- El diseño del Front-end requiere de la implementación de experiencia de usuario (UX), esto se evidenció al analizar la lectura de la información y las preguntas generadas por los clientes en las pantallas que fueron diseñadas bajo este concepto contra aquellas que no lo tenían.
- La implementación de un marco de trabajo SCRUM y la ideología del agilismo le permite a los equipos obtener conocimiento sobre las herramientas a usar, fallar de forma temprana y validar constantemente la idea y los objetivos del proyecto, generando funcionalidades de verdadero valor para los usuarios, adicionalmente plantea un cronograma entendible que le permite a los miembros del equipo interactuar constantemente permitiendo identificar falencias o problemas en cadas reunión, de cara al proyecto y sus integrantes.
- La oportunidad de aprender y aplicar los conocimientos adquiridos a lo largo de del ciclo de vida de un proyecto desde la concepción de su idea es una oportunidad única donde se puede interactuar en cada etapa y conocer de forma directa la visión

que posee cada integrante y cómo aporta cada uno en su desarrollo obteniendo conocimientos que expanden las fronteras tecnológicas donde pude observar la etapa de planteamiento de negocio y mercadeo del producto.

**Referencias Bibliográficas.**


A. Bertolino. (2007). “Software Testing Research: Achievements, Challenges, Dreams”, en Future of Software Engineering, Pág 85-103.

W. Hetzel. (1993). The Complete Guide to Software Testing, Estados Unidos: John Wiley & Sons.

C. Pérez. (2015). Arquitecturas basadas en microservicios

CEU IAM. (2018). ¿Por qué todos hablan de metodologías ágiles?. Recuperado de: https://www.ceuiam.com/business-school/por-que-todos-hablan-de-metodologias-agiles--post

I. Esmite, M. Farías, N. Farías, B. Pérez. (2007). Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source

ISTQB (2012). Standard glossary of terms used in software testing. International Software Testing Qualifications Board. ISTQB: Munich.

J. A. Mera, P. (2016). “Análisis del proceso de pruebas de calidad de software”, Ingeniería Solidaria en vol. 12, no. 20.

T. Müller, D. Friedenberg, ISTQB WG Foundation Level. (2013). Libro Programa de Estudio de Nivel Básico Para Probador Certificado, istqb, Pág. 14.

B. Korel. (1990). Automated software test data generation, Software Engineering IEEE Transactions en vol. 6, Pág. 870-879.

C.C. Michael, G. McGraw, M.A. Schatz. (2001). "Generating software test data by evolution", Software Engineering IEEE Transactions en vol. 27, no. 12, Pág. 1085-1110.

V. Khare. X. Yao. K. Deb. (2003). Performance Scaling of Multi-objective Evolutionary Algorithms.