Estrategia de aseguramiento de la calidad en plataformas Webs

Juan David Raga Palomeque

Informe sobre la práctica profesional para optar por el título de: **Ingeniero de Sistemas de la Universidad de Antioquia**

Orientadores:
Danny Alexandro Munera Ramirez - Doctor en Informática (Interno)
Lina Cardona - Ingeniera de Sistemas (Externa)

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Departamento de Ingeniería de Sistemas
2023

Dedicatoria

A mi padre, quien ya no se encuentra con nosotros, siempre quise enseñarle este gran logro en mi vida.

Agradecimientos

A mi familia y esposa que siempre creyeron en mis capacidades aun cuando yo no y a cada profesor de la facultad de Ingeniería de la Universidad de Antioquia que aportó en mi crecimiento académico y profesional.

Tabla de contenido

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers
BDD	Behavior Driven Development
MVP	Minimum viable Product
PYME	Pequeñas y medianas empresas
Bugs	Unexpec problem with Software
QA	Quality Assurance
UAT	User Acceptance Testing
Dev	Development
Prod	Production
E2E	End to End
DevOps	Development and Operations
API	Application Programming Interface
CI	Continuous Integration
CD	Continuous Delivery
REST	Representational State Transfer
EIC	Estrategia Integral de Calidad

RESUMEN

La fidelización de clientes mediante un sistema de puntos es una de las estrategias más usadas actualmente y “Plataforma propia” es una plataforma dedicada principalmente a la potenciación de esta modalidad, que no solo proporciona fidelización de clientes sino también de las PYMES aliadas.

La compañía desarrolladora de esta plataforma vio la necesidad de crearla en un tiempo récord porque la necesidad de la misma no daba espera para un mercado creciente. No obstante se presentaban retos avasallantes, uno de estos era garantizar una alta calidad en el producto, esto con el fin de evitar un fracaso en la producción.

Para esta plataforma se desarrolló una estrategia de calidad denominada “Estrategia integral de calidad” con la cual se buscaba la garantía de una alta cobertura en las pruebas. Esta estrategia contiene diferentes apartados como pruebas E2E, funcionales, de regresión, orquestación de pruebas y pruebas de performance todas estas empleando la automatización de pruebas.

Con 13 features y 507 casos de pruebas realizados se obtuvieron 446 casos exitosos y una cobertura del 80% en todo el proyecto.

Finalmente en producción el comportamiento del MVP es estable lo que implica que utilizando la “Estrategia integral de calidad” se pudo certificar la calidad de este producto y también se puede aplicar en otro que contenga características similares.

Palabras clave **— Plataforma Propia, Calidad, Automatización, pruebas, estrategia integral.**

ABSTRACT

Customer loyalty through a points system is one of the most used strategies nowadays and "Plataforma propia" is a platform dedicated mainly to the empowerment of this modality, which builds loyalty not only for customers but also for the allied PYMES.

The company that developed this platform saw the need to create it in a record time because the need for it could not await a growing market. However, there were overwhelming challenges, one of which was to guarantee a high-quality product in order to avoid production failure.

For this platform, a quality strategy called "Integral Quality Strategy" was developed to guarantee high test coverage. This strategy contains different sections, such as E2E, functional, regression, test orchestration and performance tests, all of them using test automation.

With 13 features and 507 test cases performed, 446 successful cases were obtained and a coverage of 80% throughout the project.

Finally, in production, the behavior of the MVP is stable, which implies that using the "Integral Quality Strategy" it was possible to certify the quality of this product and it can also be applied to other products with similar characteristics.

***Keywords* — Plataforma própia, Quality, Automation, testing, integral strategy.**

I. INTRODUCCIÓN

El sistema de puntos de la compañía es el ecosistema de lealtad que nace de una alianza entre varias empresas. Este programa busca reconocer a los clientes a través de la acumulación y/o redención de puntos, todo esto por medio de una gran gama de aliados pertenecientes a diferentes industrias que ofrecen una amplia variedad de servicios y productos. Se puede afirmar que este programa es la evolución de los de los diferentes sistemas de puntos que manejaban diferentes compañías los cuales tuvieron una gran acogida y un gran reconocimiento entre los clientes. El sistema es sencillo para los clientes, consta de realizar una compra de un producto en un establecimiento aliado y, según las condiciones especificadas por este, al cliente se le asigna un número concreto de puntos los cuales puede acumular. Este mismo proceso se puede aplicar en caso de que el cliente quiera hacer una redención, es decir, adquirir un producto mediante el uso de sus puntos, en caso de faltarle puntos puede adicionar el dinero faltante correspondiente a los puntos necesarios para adquirir el servicio o producto.

En los últimos años la acogida de este programa ha aumentado sus beneficios, por lo que la adición de aliados ha creado una problemática para la redención de puntos en las sucursales de los mismos. Actualmente la única forma de hacer redenciones o acumulaciones de puntos es mediante la página web en las secciones de *Tienda Online, bonos y viajes*. Esta es una clara limitante para los aliados porque, al ser demasiados, la página de la compañía no puede contener cada uno de sus productos y servicios que están en constante actualización.

Por esta razón surge la necesidad de crear una plataforma web, a la que se le denomina como proyecto “Plataforma Propia”, en la que un cajero en una sucursal de una compañía aliada, pueda realizar a los clientes los procesos de acumulación y redención de puntos. La compañía que está desarrollando “Plataforma Propia” es uno de los principales clientes que tiene Software Quality Assurance, quien le brinda personas capacitadas y herramientas para complementar los equipos que van a desarrollar la plataforma, esta también cuenta con una versión responsive. Además de permitir redimir y acumular puntos, la plataforma también contará con un componente capaz de realizar reversiones de estas transacciones, un apartado donde se podrán visualizar los movimientos diarios realizados en una sucursal y métricas sobre las transacciones realizadas.

Cada una de las funcionalidades descritas representa un gran reto, no solo de desarrollo sino también de alto nivel de certificación, por esta razón a la par de la construcción del software también se desarrolla y se aplica una estrategia integral de calidad. Esta implementa diferentes tipos y niveles de pruebas, así como un procedimiento que podrá ser aplicado a cualquier proyecto que comparte características similares y por consiguiente garantiza un alto nivel en el producto.

En este informe se presenta el proceso de diseño y aplicación de esta estrategia integral de calidad. También se presentan los resultados detallados de su aplicación para la plataforma propia. El resto de este artículo está organizado así: Planteamiento del problema, objetivos, marco teórico, metodología, estrategia integral de calidad, resultados, conclusiones y finalmente las referencias

II. PLANTEAMIENTO DEL PROBLEMA

El proyecto “Plataforma Propia” resuelve la necesidad de acumular y redimir puntos en sucursales físicas de las PYMES afiliadas, al contar con diferentes módulos como: métricas, acumulación, redención y facturación, que están proyectados a ser de gran uso por el usuario final. Se requiere que la calidad de este software sea realmente alta, debido al corto tiempo con el que se cuenta. La implementación de una estrategia estándar de calidad no es una opción; ya que esta ofrece en su gran mayoría pruebas manuales y pruebas en ambientes locales, incrementando la probabilidad de obtener un alto número de bugs y una baja certificación y garantía respecto a las funcionalidades del proyecto. Por lo tanto, aplicar una estrategia de calidad de software que garantice un producto con los más altos estándares de calidad es realmente fundamental en este proyecto.

III. OBJETIVOS

A. Objetivo general

Desarrollar una estrategia para garantizar altos estándares de calidad, seguridad y un excelente rendimiento, en la plataforma web de la compañía.

B. Objetivos específicos

- Realizar un análisis y exploración al proyecto partiendo desde los flujos de funcionamiento básico, con el propósito de entender cómo se integran cada uno de los componentes, datos y recursos utilizados.
- Planear y diseñar una estrategia de evaluación y aseguramiento de la calidad en cada uno de los requisitos funcionales y de rendimiento.
- Ejecutar la estrategia de evaluación y aseguramiento de la calidad.
- Analizar el nivel de calidad que incluye el proyecto, basado en los resultados obtenidos de las pruebas con el fin de sugerir un plan de mejoramiento.

IV. MARCO TEÓRICO

La estrategia a aplicar está especialmente diseñada para una arquitectura basada en microservicios [1], siendo esta la que se aplica en el proyecto con el fin de evitar un alto acoplamiento entre los servicios, para que así los componentes principales de “Plataforma propia” como: usuarios, transacciones (redención y acumulación), términos y condiciones y pasarela de pagos, sean funciones independientes del proyecto.

Las integraciones de cada una de las funcionalidades necesitan estar avaladas mediante una estrategia de calidad que garantice la correcta operatividad y el rendimiento de cada una de ellas. Debido a esto, se opta por diseñar una estrategia basada en pruebas automatizadas [2] en lugar de pruebas manuales. Las pruebas automatizadas ofrecen alta calidad, entrega más rápida, se complementan bien con *continuous integration* (CI) para una colaboración grupal con otros desarrolladores en un mismo pipeline y continuous delivery (CD) para mantener despliegues a producción de forma continua [3], reducción notable de costos, además de pruebas más precisas y reducción de tiempo.

Como elementos claves y principales de la estrategia se realizan diferentes tipos de pruebas las cuales se ejecutan en el momento que el proyecto dependa de la validación de alguna de ellas para avanzar a una siguiente etapa.

Estás pruebas son:

- **Pruebas de humo:** Revisión rápida a la funcionalidad del producto para verificar que se encuentre en perfecto estado, se realizan comúnmente antes de hacer una entrega [4].

- **Pruebas exploratorias:** En estas pruebas se realiza una exploración con una cobertura amplia sobre el producto, se suelen realizar cuando se desconocen las funcionalidades principales y en estas es posible que se tengan hallazgo de bugs [5].

- **Pruebas funcionales:** Incluye un grupo de pruebas que evalúan las funcionalidades que el sistema debe realizar. Los requisitos funcionales pueden estar descritos en productos de trabajo tales como especificaciones de requisitos de negocio, épicas, históricos de usuario, casos de uso o especificaciones funcionales. En algunos casos incluso pueden estar sin documentar, las funciones describen que puede o no hacer el sistema [6].

- **Pruebas de regresión**: Se refiere a la acción de realizar pruebas a funcionalidades o partes que previamente fueron probadas pero posteriormente modificadas, estás pruebas buscan evitar la aparición de bugs o defecto nuevos [7].

- **Orquestación de pruebas:** Se define como una configuración secuencial para la ejecución de los escenarios de pruebas que ya han sido previamente automatizados. Es importante esto, porque ayuda a unir cada una de las actividades o flujos automatizados individualmente en un proceso síncrono [8].

- **Pruebas a APIS**: Se refiere a las pruebas realizadas a nivel de backend a cada una de las funcionalidades realizadas por las APIS lo que implica realizar pruebas a cada uno de los

Endpoints y flujos que concluyen en validaciones con bases de datos o integraciones con otras APIS [9].

- **Pruebas E2E**: El objetivo principal de estas pruebas es garantizar que el flujo de uso de una aplicación tal como lo utilizará el usuario final funcione de principio a fin, por esto también son llamadas pruebas de extremo a extremo. Es uno de los niveles más importantes en la pirámide del testing para la detección de errores en la interfaz de usuario [10].

- **Pruebas de estrés**: Se realiza generalmente para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores a determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere la carga esperada [11].

- **Pruebas de Carga**: Se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones concurrentes (que ingresan al mismo tiempo) esperadas y obtener un tiempo de respuesta [12].

Las pruebas descritas anteriormente requieren de algunas tecnologías y metodologías para su realización, estas son:

- **Azure Devops:** Herramienta que permite la colaboración de distintos procesos en el que trabajan diferentes desarrolladores. Permite administración de proyectos y colaboración para desarrollar software. Las compañías tienen la capacidad de mejorar y trabajar a un ritmo más acelerado gracias a está herramienta que además tiene integración con diferentes tecnologías que pueden ser de gran utilidad [13].

- **Karate Framework**: Marco de trabajo de automatización de pruebas, este es de uso general y de código abierto, principalmente destinado para pruebas a APIS aunque también puede ser usado para pruebas E2E al ser colaborativo con otras herramientas. Este framework soporta varios lenguajes de programación aunque está basado principalmente en la sintaxis BDD y cuenta con la tecnología suficiente para la integración continua [14].

- **Git y GitHub:** Git es un software para el control de versiones de proyectos. GitHub es un servicio que ofrece el alojamiento de los proyectos que son versionados mediante Git, aunque también ofrece otros servicios los repositorios en la nube para el alojamiento de proyectos sigue siendo su principal función [15].

- **Postman:** Cliente REST para el testeo y creación de APIS. Este cliente tiene un funcionamiento muy simple lo cual ha hecho que tenga gran acogida en el mercado, además también agiliza el desarrollo y permite la colaboración entre desarrolladores. Permite el lenguaje de programación JavaScript para realizar pruebas y también se pueden exportar colecciones de Endpoints [16].

- **SQS:** Las colas de mensajes administradas para microservicios. Es uno de los servicios utilizados ofrecidos por AWS. Este servicio puede enviar, almacenar y recibir mensajes entre los diferentes componentes, sin importar el tamaño y no se necesita de la disponibilidad de otros servicios o componentes [17].

- **SNS:** Servicio de notificaciones push mediante suscripciones para microservicios. Es uno de los servicios utilizados y es ofrecido por AWS. Este servicio puede enviar, almacenar y recibir mensajes entre los diferentes componentes, sin importar el tamaño y no se necesita de la disponibilidad de otros servicios o componentes [18].

- **KMS**: Software para crear, administrar y controlar claves criptográficas en aplicaciones. Es uno de los servicios utilizados ofrecidos por AWS. Suele utilizarse para cifrar mensajes como los que circulan en las colas SQS [19].

- **CI** (*Continuous Integration):* la premisa de esta filosofía es sencilla y lo que busca es que el código desarrollado en un proyecto por varios desarrolladores pueda estar en un pipeline de forma rápida y segura donde no hay conflictos y se tiene la más reciente versión de proyecto en todo momento [20].

- **CD** (*Continuous Delivery):* La implementación correcta de CI permite un rápido despliegue a producción lo que garantiza tener incrementos de forma continua, automatizada y sostenible [21].

- **K6:** Marco de desarrollo para pruebas de rendimiento, caracterizado por su capacidad de encapsulamiento para la usabilidad y el rendimiento, cuenta con una interfaz de línea de comando para ejecutar scripts [22].

- **Cypress:** Herramienta de pruebas, principalmente E2E enfocada principalmente en la web moderna, creada principalmente para ingenieros de calidad, la pruebas se realizan mediante el lenguaje de programación JavaScript y se pueden realizar pruebas de E2E, pruebas de componentes, pruebas de integración y test unitarios [23].

- **Agilismo:** Filosofía adoptada principalmente por compañías dedicadas a la creación de software, con el fin de entregar a clientes externos como internos entregas de valor continuas y sostenible. La necesidad de las empresas por adaptarse a los rápidos cambios tecnológicos y de mercado hacen que el agilismo en la actualidad tenga una gran acogida, algunos de los marcos ágiles más utilizados son: Scrum, Cristal, XP y Kanban [24].

V. METODOLOGÍA

Para garantizar la calidad de este proyecto se implementan diversas actividades en tiempos diferentes según el estado del mismo. Principalmente se emprende una exploración, donde se aborda el análisis de cada uno de los componentes técnicos del proyecto, análisis de flujos y servicios en la nube. También, en esta etapa, se establecen o se cambian las tecnologías a usar con el fin de utilizar las más idóneas al proyecto. En la siguiente etapa se aborda el segundo objetivo, el cual para lograrse se enfoca principalmente en la realización actividades referentes a las pruebas como: planeación, estimación de tiempos, definición de recursos, cronograma, análisis de riesgo, alcance, definición de estructuras, diseño de planes, escenarios y casos de pruebas.

La etapa número tres será una fase con un comportamiento cíclico donde se aplicará *scrum* como metodología ágil. Durante este periodo se tomará cada una de las tareas en el *product backlog* y se aplicará en cada una diseño de escenarios y casos de prueba, automatización de estos casos y finalmente certificación de la tarea.

Finalmente, en la última etapa se procede a realizar pruebas de *performance* y la orquestación de las pruebas de los módulos verificados en la etapa anterior. Para alcanzar el objetivo cuatro se parte de los casos y escenarios de prueba exitosos, no exitosos y métricas de desempeño. Una muestra visual de la metodología se muestra en la figura 1.

Figura 1 - Metodología de aseguramiento de calidad del proyecto

VI. ESTRATEGIA INTEGRAL DE CALIDAD

La propuesta planteada con la que se pretende crear un método que pueda ser posteriormente adaptable para cualquier proyecto que contenga características similares a las descritas en el apartado introductorio; está enfocada principalmente en algunos tipos de pruebas que son indispensables para cualquier proyecto de software que requiera de un tiempo relativamente corto para la obtención de un MVP, que cumpla con altos estándares de calidad.
Si bien, las pruebas aplicadas existen, la diferencia radica principalmente en la forma, tiempo y secuencia en la que son aplicadas. Esta estrategia inicia con la premisa de la estabilización de cada uno de los ambientes que se utilizan en el proyecto, posteriormente se concretan dos reuniones con el equipo de frontend y el de backend para establecer las historias de usuario y los contratos. Estos dos elementos son la base para el diseño de la arquitectura del proyecto de QA, dichos elementos dan paso a la creación de los arquetipos en las tecnologías utilizadas para la creación de suites, escenarios y casos de prueba.
En el transcurso de los sprints se realiza la ejecución de todos los escenarios en ambiente local y en el pipeline, al ejecutar, según sea el caso pruebas de: regresión, orquestación de pruebas, E2E, pruebas de humo y de performance. Finalmente se hace el reporte de los bugs encontrados.

Una muestra visual de este proceso se muestra en la figura 2.

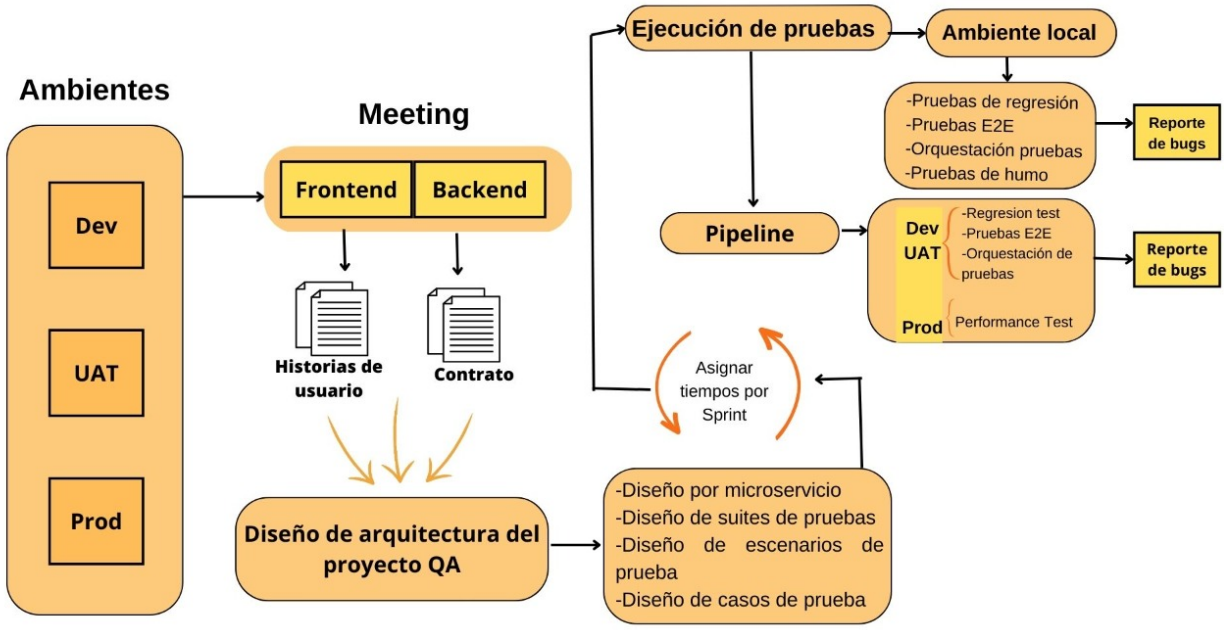

Figura 2 - Estrategia Integral de Calidad

VI.I Antecedentes pre- aplicación de la estrategia

Para la correcta aplicación de esta propuesta se asume que los ambientes básicos: Desarrollo, UAT, ambiente donde se realizan las pruebas de aceptación de usuario final, y Producción, están debidamente estabilizados y no presentan ningún tipo de inconveniente crítico en su totalidad. La concesión por el equipo de trabajo con respecto a la utilización de cada uno de las herramientas que se emplearán en el proyecto con el fin de evitar pérdida de tiempo, también debe estar establecida, porque el empleo oportuno y conciso de este consenso es uno de los pilares mas importantes en el desarrollo de la estrategia.

Cada funcionalidad debe estar debidamente refinada no sólo por los desarrolladores sino por el equipo de QA y la arquitectura general del proyecto debe estar expuesta al pleno conocimiento del equipo, esto con el fin de realizar pruebas de cada nivel y ambiente con un alto grado de aseguramiento de la calidad.

VI.II Inicio de aplicación de la estrategia

Diseño de las pruebas y contrato

FrontEnd: Para el diseño de las pruebas es indispensable contar no solo con los colaboradores de QA, sino también con los colaboradores de diseño, frontend y el líder técnico del equipo. Para el diseño de este módulo, en esta estrategia es indispensable tener a ambos equipos trabajando, no solo se hace el diseño de las pruebas, sino que se inicia desde temprano a identificar bugs en el proyecto.
Un punto de partida claro es el mockup que previamente desarrolló el equipo de diseño en donde se evidencia una primera vista de la meta temprana a alcanzar. Los involucrados analizan y establecen las pautas para proceder y QA se encarga de certificar la calidad y viabilidad en cada una de esas pautas.

BackEnd: Una diferencia notable para este apartado es que no necesitaremos a los colaboradores de diseño sino a los de backend principalmente y por supuesto al líder técnico, el punto central de este apartado es el contrato en el que se definen cada uno de los microservicios.
En este apartado al igual que en el resto de la estrategia, los ejes centrales son, además del compañerismo y el trabajo en equipo, los aportes que cada uno de los involucrados brinda para creación los contratos, donde se describen los microservicios y sus detalladas características, en caso de faltar alguno de estos componentes la aplicación de la estrategia se vería gravemente afectada y la calidad esperada no sería la provista.

Cada una de estas reuniones ofrece al equipo de QA historias de usuario de las cuales se crean los suite, escenarios y casos de pruebas.

VI.III Creación arquitectura de modulo QA desde el contrato

Cada funcionalidad está ligada a un microservicio y cada microservicio debe estar discriminado por servicio como se muestra en la Figura 1. Estos servicios están plasmados en cada uno de los features y también asociados los endpoints necesarios. Estos deben tener mínimo tres escenarios que dependen principalmente del protocolo HTTP definido ya que son los códigos de status 200 (201), 400 y 404 los que hacen referencia a recursos creados o consultados exitosamente, errores

de sintaxis en la solicitud realizada y recurso no encontrado, respectivamente. En la Figura 2 se muestra un ejemplo de esto.

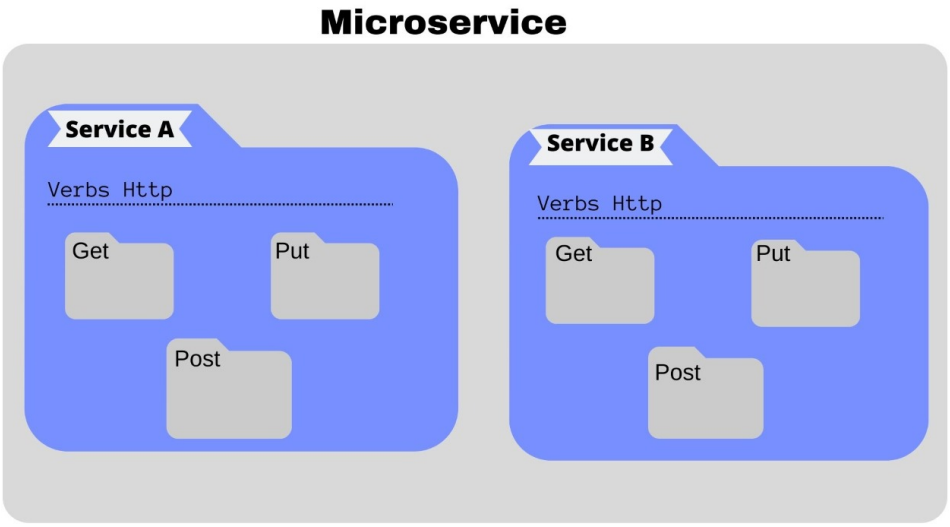

Figura 1 - Diagrama de paquetes de microservicios

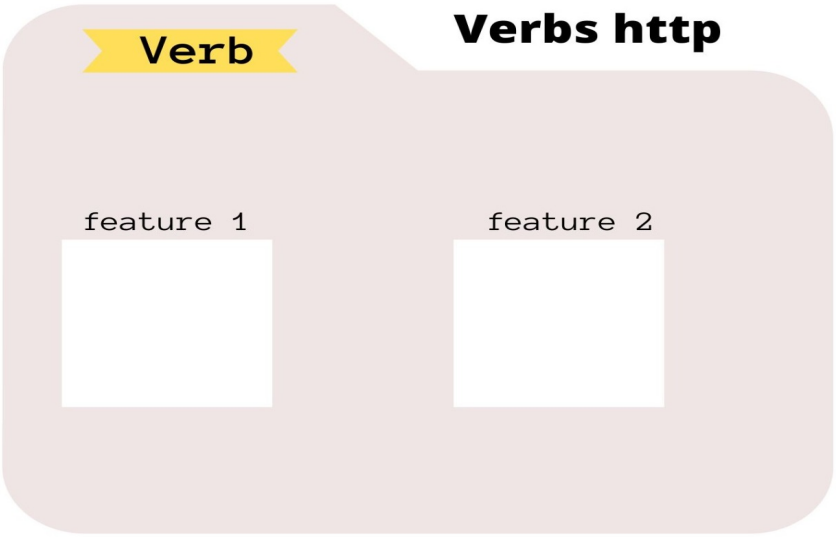

Figura 2 - Features por cada servicio de una API

Estos tres códigos asociados a los escenarios resumen en su totalidad todos los códigos HTTP conocidos en cada uno de sus rangos. Es pertinente aclarar que cada uno de ellos puede tener tantos casos de prueba como sea necesario, lo que implica que el status code 500, así como otros tipos de códigos HTTP fueron evaluados implícitamente en las pruebas. Una orientación visual de esto se muestra en la figura 3.

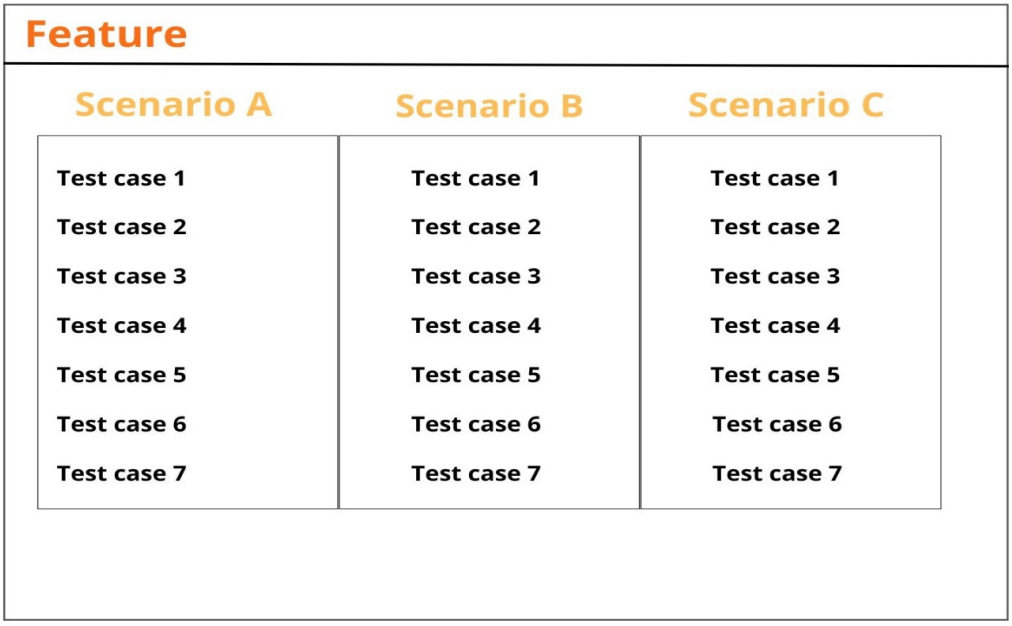

Figura 3 - Diseño general de escenarios de prueba

VI.IV Historias de usuario y reporte de bugs

Cada feature de pruebas seleccionado para desarrollar está directamente relacionado con una historia de usuario y, por consiguiente, estas features tendrán específicamente los tres escenarios de prueba descritos anteriormente. En cada escenario se tendrán tantos casos de pruebas como sea necesario, no obstante en algunas historias específicas este lineamiento puede no cumplirse. Aun así se debe realizar el desarrollo de pruebas guiado por comportamiento (Behavior Driven Development), es de esperarse la aparición de bugs, estos deben ser asignados al desarrollador a cargo de la historia de usuario el cual debe realizar el ajuste y posteriormente notificar al analista de pruebas para su verificación. Es de suma importancia abordar los bugs encontrados ya que estos están reportados en la herramienta de azure devops de acuerdo a su grado de criticidad, evitar la acumulación excesiva de bugs contribuye al mantener el agilismo en el equipo.

Finalizado cada sprint se deben generar gráficas en las que se evidencie el avance que se ha tenido en la solvencia de los bugs.

VI.V Pruebas de regresión por microservicio

Si bien las pruebas de regresión están más orientadas a realizarse cuando se ejecuta un cambio en alguna funcionalidad que ya ha sido certificada, en esta estrategia también se utilizan las pruebas de regresión para la verificación del correcto funcionamiento de un flujo o microservicio completamente desarrollado. El analista de calidad debe asegurarse cada vez que un servicio en

un microservicio esté listo, realizando las pruebas pertinentes, esto con el fin de evitar la aparición de bugs que ya hayan sido solucionados anteriormente.

Esta estrategia está diseñada específicamente para llevarse a cabo por analistas de calidad que tengan un conocimiento intermedio en programación, de no ser así varios de los objetivos no podrían lograrse; esto porque al ser un proyecto desarrollado con tecnologías modernas requiere un alto conocimiento en lógica de programación y fundamentos de ingeniería de software.

VI.VI Orquestación de pruebas

Este proceso síncrono se realiza antes de crear el pipeline en el ambiente de producción, por consiguiente, es en el ambiente de pruebas de aceptación de usuario donde se debe consolidar esta orquestación la cual debe ser completamente síncrona y cada una de las funcionalidades integradas en la automatización de la misma debe compartir con el ambiente de producción el mayor número de similitudes posibles.

La orquestación se inicia, principalmente, haciendo una llamada tipo instancia a cada uno de los endpoints de los microservicios creados y pasando los parámetros necesarios para lograr el éxito en este apartado. Si por ejemplo, este proyecto se tratase de un CRUD sencillo, la automatización de la orquestación de pruebas haría un llamado al Create inicialmente, luego al Read y seguiría de este modo hasta completar el flujo. La diferencia directa de este proceso con las pruebas de regresión, es que aquí los datos o parámetros ingresados en el Créate son los mismos que luego se leen, actualizan y finalmente se borran de una forma completamente automatizada y cercana a un proceso que puede ser realizado por el usuario final. Además todo este proceso se realiza de forma automatizada ya que el Analista de Calidad solo tuvo que enviar los parámetros y ejecutar el flujo.

Una ventaja clara de esta parte de la estrategia es la simplicidad y posteriormente robustez que esto representa para CI y CD, ya que con solo proceder a las orquestación de pruebas no solo se está ejecutando cada una de las funcionalidades del microservicio si no que se está garantizando la funcionalidad integral del mismo.

Para este proyecto puntualmente la orquestación de pruebas es realizada para el “camino feliz” pero es completamente factible realizar la orquestación de pruebas para cualquier escenario que se requiera garantizar.

VI.VII Pruebas de rendimiento

Este es el paso final de la estrategia, nuestro producto está a las puertas de Producción y las pruebas de rendimiento son la puerta de entrada a este ambiente. Es de suma importancia que el lenguaje de programación que utiliza la tecnología con la que se realizan estas pruebas sea el lenguaje más usado durante todo el proyecto, en este caso puntual fue el lenguaje Javascript.

En el caso puntual de un MVP las pruebas de rendimiento mínimas son las de estrés y de carga, ya que estas garantizan una estabilidad apropiada para esta etapa del proyecto. Como requisito fundamental en la mantenibilidad de esta estrategia, es importante que se realice la orquestación

de pruebas cada vez que se haga una migración a ambientes nuevos, (UAT, Dev o Prod) incluso en ambientes locales en caso de ser necesario.

VII RESULTADOS

Después de rigurosos esfuerzos por mantener y aplicar la estrategia integral de calidad, se logra que esta finalmente sea aplicada en cada módulo del MVP. Esto se logra en un transcurso total de 6 sprints de una semana y 10 sprint de dos semanas, alcanzando así cada uno de los objetivos propuestos. En cada uno de estos módulos se realizó la exploración necesaria con el fin de encontrar la viabilidad en la aplicación de la estrategia integral de pruebas. En los siguientes apartados se muestra con detalle los hallazgos y aciertos teniendo como punto de partida la creación de contratos, el diseño y desarrollo de test suite con la implementación de features, escenarios y casos de pruebas.

1. **Análisis y exploración (Diseño de contrato y suite test y casos de prueba)**

   Con un total de 7 contratos los cuales contienen la documentación de cada una de los microservicios y que dan paso a las 39 historias de usuario. Se crearon los planes de pruebas que fueron un total de 15, cada uno con los siguientes apartados: planeación y diseño, ejecución de pruebas y certificación.

2. **Estrategia integral**

   a. **Ejecución de pruebas a servicios**

      Se realizaron pruebas automatizadas a los servicios contenidos en los módulos de: redención, acumulación, pasarela de pagos, facturación, métricas, y finalmente términos y condiciones. Cada módulo se abordó partiendo de su respectiva historia de usuario que contenía, severidad, criticidad y criterios de aceptación. El principal objetivo de la realización de estas pruebas se basó en la obtención de bugs desde una edad temprana en el proyecto, en total se realizaron 36 escenarios de prueba. Finalmente, de los 507 casos de prueba que se planearon durante cada uno de los sprints, 446 casos fueron exitosos, es decir el 88% de los casos fueron aprobados y ninguno de los no exitosos fue de alta criticidad.

Los casos fallidos se reportan como bugs a los desarrolladores, se realiza en sesiones donde se socializa cada uno de ellos para afianzar la procedencia y veracidad de cada bug.

b. Ejecución de pruebas E2E

Se llevaron a cabo la realización de pruebas automatizadas la cuales afianzaron los servicios consumidos desde el Frontend y que ya habían sido probados anteriormente, esto garantiza la funcionalidad de cada uno de los servicios, mejoró la experiencia de usuario y aportó a la detención de temprana de bugs.

c. Ejecución de pruebas de regresión

Debido al tiempo con el que se contó para la realización del proyecto los cambios para garantizar una mejor calidad eran inevitables, los módulos como: Redención, acumulación y pasarela de pagos. La cobertura automatizada para estos módulos fue del 100% por lo que se podían aplicar pruebas de regresión una vez por semana, no obstante, en caso de encontrarse cambios significativos, se tomaban 3 días del sprint para realizar los cambios en el código y posteriormente se realizaba de nuevo la ejecución.

d. Orquestación de pruebas

Se realizó la orquestación de pruebas que implicó la utilización de los servicios cuyos casos de prueba ya estaban en “estado exitoso” en las pruebas realizadas a los servicios de manera individual, E2E y de regresión; como se esperaba no se encontraron bugs para su reporte. El objetivo principal de estas pruebas fue analizar el comportamiento en la integración de cada uno de los servicios que componen el proyecto.

3. Reporte y resolución de Bugs

Cada una de las pruebas realizadas que presentan estado “no exitoso”, da como resultado automáticamente el reporte de un bug. Esto se pudo establecer de esta forma ya que los casos de pruebas no aplicables en pro del MVP no eran tenidos en cuenta. Los bugs se reportaron mediante azure devops y estaban directamente asociados al item de test plan también creado en azure devops y se les asociaba de acuerdo a su resultado pos ejecución, un estado como se muestra en la figura 4. Se reportó un total de 49 bugs.

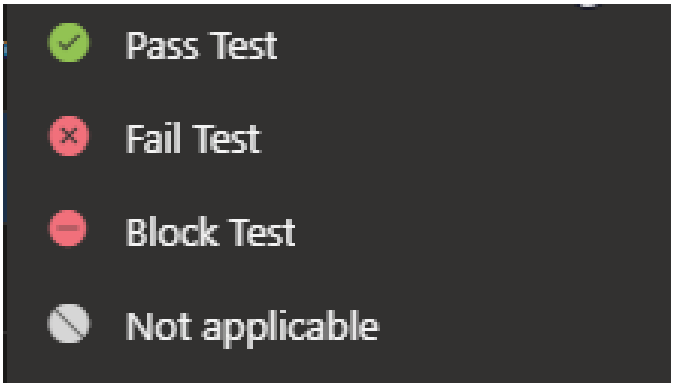

Figura 4 - Estado de los bugs

4. Plan de Mejoramiento

Si bien la Estrategia Integral de Calidad Aplicada en el desarrollo del MVP fue exitosa, algunos factores implicaron pequeños retrasos y formas de trabajar atípicas que en su momento puntual dieron pie a un sobreesfuerzo de trabajo y también a la detención o aplazamiento de algunas tareas. La estabilización de los ambientes de Desarrollo, UAT y Producción son sumamente importantes para la garantía de la calidad de un producto en cualquier compañía. De no contar con ambientes estables, no se puede garantizar que los microservicios que operan sobre estos funcionen de la mejor manera. Esto puede crear gran incertidumbre sobre el funcionamiento de un producto e incluso durante su despliegue al ambiente de producción.

Un punto de partida para la solución del inconveniente mencionado es contar con un equipo de DevOps experimentado para la implementación de estos ambientes, porque de forma transversal e implícita el equipo de QA también realiza pruebas al ambiente mientras evalúa la calidad de los microservicios que corren en el mismo .

Las tecnologías que permitieron obtener el MVP en producción fueron de las más modernas y en su mayoría requerían de credenciales de acceso. La tardanza de proveer a cada uno de los colaboradores con sus propias credenciales implicó en su momento un

retraso que, si bien no afectó los tiempos finales de entrega y despliegue, si afecto metas parciales que estuvieron cambiando de forma dinámica por las demoras en los accesos y credenciales.

Finalmente la demora en la elección de un software para la realización de las pruebas de rendimiento, fue claro ejemplo de que no se debía dejar para el final, ya que fueron realizadas después de desplegar el MVP en producción .

VIII. CONCLUSIONES

Durante el tiempo que duró la pasantía se realizaron exploraciones a cada uno de los servicios del proyecto, trabajando principalmente en el equipo de QA como Analista de Automatización de pruebas, se llevaron a cabo tareas de planeación, estimación de tiempos, definición de recursos, cronograma, análisis de riesgo, alcance, definición de estructuras, diseño de planes, escenarios y casos de pruebas. Todo esto trabajando en conjunto con cada uno de los colaboradores del equipo.

La modificación a la metodología ágil scrum de una semana y posteriormente de dos semanas, en los últimos sprint, en el marco de trabajo fue de gran ayuda para el desarrollo del proyecto, esto porque al tener un tiempo reducido para la entrega y exposición de un incremento, motivó al equipo a trabajar con mayor cohesión con la intención de tener éxito en cada sprint.

Esto dio paso a contar con los recursos y el tiempo necesario para diseñar una Estrategia Integral de Calidad (EIC), que cumplió con cada uno de los temas requeridos en cuanto a la garantía y aseguramiento de la calidad. Durante la ejecución de esta estrategia, se hizo uso de diferentes tecnologías que en un principio eran desconocidas pero que poco a poco se volvieron cruciales e indispensables para cumplir cada una de las metas. El conocimiento adquirido por la academia definitivamente fue un punto central en la adaptación a las nuevas tecnologías, si bien, las herramientas evolucionan, la lógica de programación siempre es la misma.

Finalmente mediante el gran trabajo en equipo y la aplicación de la Estrategia Integral de Calidad se obtuvo una gran cobertura de pruebas en todo el proyecto, se redujo considerablemente los tiempos de prueba, se garantizó el correcto funcionamiento de cada microservicio y arrojó resultados destacables en el MVP.

REFERENCIAS

[1] "What are microservices?" microservices.io. https://microservices.io .

[2] "What is automation testing? Test tutorial". Guru99. https://www.guru99.com/automation-testing.html.

[3] "What is CI/CD? Continuous integration and continuous delivery explained". InfoWorld. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html.

[4] "¿Qué es la prueba de humo? Su función y beneficios". Cynoteck. https://cynoteck.com/es/blog-post/what-is-smoke-testing/.

[5] "What is exploratory testing?" Guru99. https://www.guru99.com/exploratory-testing.html.

[6] "What is functional testing? Types & examples". Guru99. https://www.guru99.com/functional-testing.html.

[7] "What is regression testing? Test cases (example)". Guru99. https://www.guru99.com/regression-testing.html.

[8] "What is test orchestration? Benefits, tools & strategy". katalon.com. https://katalon.com/resources-center/blog/test-orchestration.

[9] A. S. Gillis y K. Brush. "What is API testing and How to Use It?" App Architecture. https://www.techtarget.com/searchapparchitecture/definition/API-testing.

[10] Katalon. "What is end-to-end testing? | E2E testing tools | katalon". katalon.com. https://katalon.com/resources-center/blog/end-to-end-e2e-testing.

[11] "Stress testing". Load testing for engineering teams | Grafana k6. https://k6.io/docs/es/tipos-de-prueba/stress-testing/#:~:text=Las%20pruebas%20de%20estrés%20(stress,sistema%20bajo%20una%20carga%20pesada.

[12] "¿Qué es la prueba de carga? | Pruebas de carga por LoadView ? 2023 Guía de expertos". LoadView. https://www.loadview-testing.com/es/pruebas-de-carga/.

[13] "What is Azure DevOps? - Azure DevOps". Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops.

[14] "Karate framework: Testeo de apis de impacto | apiumhub". Apiumhub. https://apiumhub.com/es/tech-blog-barcelona/karate-framework-testeo-apis/.

[15] "Git vs github: What's the difference and how to get started with both". Kinsta®. https://kinsta.com/knowledgebase/git-vs-github/.

[16] "What is postman?" Postman. https://www.postman.com.

[17] "Colas de mensajes completamente administradas - amazon simple queue service – amazon

web services". Amazon Web Services, Inc. https://aws.amazon.com/es/sqs/.

[18] "AWS | servicio de notificaciones push (SNS)". Amazon Web Services, Inc. https://aws.amazon.com/es/sns/.

[19] "Uso de claves - AWS key management service - amazon web services". Amazon Web Services, Inc. https://aws.amazon.com/es/kms/.

[20] "Continuous integration, continuous delivery". https://www.nokia.com/networks/core/ci-cd/?did=d00000000608&gclid=CjwKCAiAv9ucBhBXEiwA6N8nYPV6mHM6VcBZl1eUIcxjaQ_3_F7ZJRPfaX6dF1JLVPelEy585OeklxoC1eUQAvD_BwE.

[21] "Continuous integration, continuous delivery". https://www.nokia.com/networks/core/ci-cd/?did=d00000000608&gclid=CjwKCAiAv9ucBhBXEiwA6N8nYPV6mHM6VcBZl1eUIcxjaQ_3_F7ZJRPfaX6dF1JLVPelEy585OeklxoC1eUQAvD_BwE.

[22] Mwaura, W. (s.f.). *API performance testing with k6*. CircleCI. https://circleci.com/blog/api-performance-testing-with-k6/#:~:text=k6%20is%20an%20open%20source,monitor%20the%20test%20run%20results.

[23] *Why cypress?* | *cypress documentation*. (s.f.). Why Cypress? | Cypress Documentation. https://docs.cypress.io/guides/overview/why-cypress#In-a-nutshell.

[24] *¿Qué es agilismo y cuáles son sus beneficios?* (s.f.). Prehome. http://www.pragma.com.co/blog/que-es-el-agilismo-y-cuales-son-sus-beneficios.