

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“ESTUDIO Y EVALUACIÓN DE HERRAMIENTAS
DE TESTING EN EL ÁREA DEL DESARROLLO
WEB”

DIEGO EDUARDO CARVAJAL SOLANO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Cecilia Reyes Covarrubias
Profesor Correferente: Luis Hevia Rodríguez

Marzo - 2022

DEDICATORIA

Dedicada a mis padres, Romy Solano Peña y Roberto Carvajal Caneo, que gracias a su apoyo y esfuerzo estoy donde estoy actualmente.

AGRADECIMIENTOS

Agradezco a mis tíos Dalia Carvajal Caneo y Jorge Moya Washington, que fueron quienes me recibieron durante todo mi proceso académico.

A la profesora Cecilia Reyes por su apoyo, ya que sin ella quizás no hubiese empezado con todo este proceso. Mención especial para todos aquellos que estuvieron ahí en el proceso, ya sea para compartir una cerveza o una buena conversación.

RESUMEN

Resumen— El desarrollo web es, sin lugar a dudas, un área de vital importancia en la actualidad, en especial en el ámbito del comercio electrónico. Empresas de alto calibre en este rubro, cuentan con una gerencia específica para esta área o contratan servicios de otras, para anteponerse a la competencia. Debido al impacto que puede tener en los negocios, es indispensable reducir o eliminar los errores en estas soluciones tecnológicas. Es por lo anterior, que se propone realizar un estudio para evaluar las herramientas de testeo que existen, definiendo así las mejores combinaciones de estas para las distintas tecnologías utilizadas en el desarrollo web.

Palabras Clave— Pruebas, Errores, Revisión

ABSTRACT

Abstract— Web development is, without a doubt, an area of vital importance today, especially in the field of electronic commerce. High-caliber companies in this area have specific management for this area or hire services from others, to get ahead of the competition. Due to the impact it can have on business, it is essential to reduce or eliminate errors in these technological solutions. It is for this reason that it is proposed to carry out a study to evaluate the testing tools that exist, thus defining the best combinations of these for the different technologies used in web development.

Keywords— Tests, Mistakes, Revision

GLOSARIO

FTP: File Transfer Protocol

HU: Historia de Usuario

IDE: Integrated Development Environment

MVC: Modelo Vista Controlador

NPM: Node Packet Manager

POM: Project Object Models

REST: REpresentational State Transfer

SOAP: Simple Object Access Protocol

WAR: Web Application Archive

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	XII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Objetivos	6
1.1.1 Objetivos específicos	6
CAPÍTULO 2: MARCO CONCEPTUAL	7
2.1 Tecnologías de desarrollo	7
2.1.1 Phalcon	7
2.1.2 Java Spring Boot	8
2.1.3 NodeJS	8
2.1.4 VueJs	8
2.1.5 AngularJS	9
2.2 Pruebas de software	10
2.2.1 Tipos de pruebas de Software	10
2.2.2 Ciclo de vida del desarrollo de software	11
2.2.3 Paradigma de desarrollo de software	12
2.2.4 Niveles de prueba	13
2.2.5 Modelo de calidad	14
2.2.6 API de REST	15
2.3 Herramientas de prueba	16
2.3.1 Cucumber	16
2.3.2 Selenium	17
2.3.3 JMeter	17
2.4 Problemáticas durante las pruebas de Software	19
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	20
3.1 Definición del ambiente	20
3.1.1 Controlador	20
3.1.2 Vista	20
3.1.3 Modelo	21
3.1.4 Metodología de Desarrollo	21

3.2	Product Backlog	22
3.2.1	Historias de usuario y criterios de aceptación	22
3.2.2	Tareas a desarrollar	28
3.3	Plan de pruebas	32
3.3.1	Componentes y funcionalidades a evaluar	33
3.3.2	Pruebas unitarias	35
3.3.3	Pruebas de integración	40
3.3.4	Pruebas de sistema	42
3.3.5	Pruebas de rendimiento	44
3.4	Evaluación de herramientas	45
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		47
4.1	Ambiente de pruebas	47
4.2	Pruebas unitarias	47
4.2.1	Escenarios	48
4.2.2	Pasos	49
4.3	Pruebas de integración	50
4.3.1	Escenarios: Método AddToCart	50
4.3.2	Pasos: Método AddToCart	51
4.3.3	Escenarios: Método removeFromCart	52
4.3.4	Pasos: Método removeFromCart	52
4.4	Pruebas de sistema	53
4.4.1	Acceso a catálogo de productos	54
4.4.2	Acceso a ficha de producto	55
4.4.3	Cambio de tipo de producto	57
4.4.4	Filtrado de productos	58
4.4.5	Agregar producto a carro de compras	60
4.4.6	Remover producto del carro de compras	61
4.4.7	Flujo completo de compra	63
4.5	Pruebas de rendimiento	65
4.5.1	Pruebas con múltiples consultas a un mismo endpoint	65
4.5.2	Pruebas con consultas simultáneas a distintos endpoints	69
4.6	Evaluación	73
4.6.1	Resultados	73
4.6.2	Versatilidad	74
4.6.3	Documentación	75
4.6.4	Escalabilidad	76
CAPÍTULO 5: CONCLUSIONES		77
ANEXOS		82
5.1	Enlaces a repositorios utilizados	82
REFERENCIAS BIBLIOGRÁFICAS		83

ÍNDICE DE FIGURAS

1	Software errors during 2016. Fuente: www.dzone.com	2
2	Modelo en cascada. Fuente: www.researchgate.net	3
3	Costo asociado a la corrección de errores en cada etapa del desarrollo. Fuente: www.mat.uson.mx	4
4	Árbol de problema. Fuente: Elaboración propia	5
5	Arquitectura Vue. Fuente: https://avimehenwal.in/blog/vue-front-end-renderer/	9
6	Arquitectura Angular. Fuente: https://www.matridtech.net/	10
7	Ciclo metodología Scrum. Fuente: https://blog.wearedrew.co/ventajas-y-desventajas-de-la-metodologia-scrum	13
8	Niveles de pruebas de software. Fuente: [Umar, 2019]	14
9	Modelo de calidad ISO/IEC 25010. Fuente: ISO/IEC 25010	16
10	Diagrama API REST. Fuente: https://www.redhat.com	16
11	Modelo de datos. Fuente: Elaboración propia	21
12	Historia de Usuario HU-01. Fuente: Elaboración propia	22
13	Historia de Usuario HU-02. Fuente: Elaboración propia	23
14	Historia de Usuario HU-03. Fuente: Elaboración propia	23
15	Historia de Usuario HU-04. Fuente: Elaboración propia	23
16	Historia de Usuario HU-05. Fuente: Elaboración propia	24
17	Historia de Usuario HU-06. Fuente: Elaboración propia	24
18	Historia de Usuario HU-07. Fuente: Elaboración propia	24
19	Historia de Usuario HU-08. Fuente: Elaboración propia	25
20	Historia de Usuario HU-09. Fuente: Elaboración propia	25
21	Historia de Usuario HU-10. Fuente: Elaboración propia	25

22	Historia de Usuario HU-11. Fuente: Elaboración propia	26
23	Historia de Usuario HU-12. Fuente: Elaboración propia	26
24	Historia de Usuario HU-13. Fuente: Elaboración propia	26
25	Historia de Usuario HU-14. Fuente: Elaboración propia	27
26	Historia de Usuario HU-15. Fuente: Elaboración propia	27
27	Historia de Usuario HU-16. Fuente: Elaboración propia	27
28	Estructura pruebas método getCart. Fuente: Elaboración propia	35
29	Estructura pruebas método getCatalog. Fuente: Elaboración propia	35
30	Estructura pruebas método getComponents. Fuente: Elaboración propia	36
31	Estructura pruebas método getCustomer. Fuente: Elaboración propia	36
32	Estructura pruebas método createCustomer. Fuente: Elaboración propia	37
33	Estructura pruebas método createOrder. Fuente: Elaboración propia	37
34	Estructura pruebas método updateOrder. Fuente: Elaboración propia	38
35	Estructura pruebas método getProduct. Fuente: Elaboración propia	38
36	Estructura pruebas método checkProductStock. Fuente: Elaboración propia	39
37	Estructura pruebas método updateProductStock. Fuente: Elaboración propia	39
38	Estructura pruebas método getStore. Fuente: Elaboración propia	40
39	Estructura pruebas método addToCart escenarios con 1 producto. Fuente: Elaboración propia	40
40	Estructura pruebas método addToCart escenarios con 2 productos. Fuente: Elaboración propia	41
41	Estructura pruebas método addToCart escenarios con producto inexistente. Fuente: Elaboración propia	41
42	Estructura pruebas método addToCart escenarios con orden inexistente. Fuente: Elaboración propia	41

43	Estructura pruebas método removeFromCart. Fuente: Elaboración propia	42
44	Estructura pruebas método removeFromCart escenarios con orden inexistente. Fuente: Elaboración propia	42
45	Pruebas unitarias: Resultados generales escenarios. Fuente: Elaboración propia	48
46	Pruebas unitarias: Resultados por escenario. Fuente: Elaboración propia .	49
47	Pruebas unitarias: Resultados generales pasos. Fuente: Elaboración propia	49
48	Pruebas unitarias: Resultados pasos por método. Fuente: Elaboración propia	50
49	Pruebas de integración: Resultados escenarios addToCart. Fuente: Elaboración propia	51
50	Pruebas de integración: Resultados pasos addToCart. Fuente: Elaboración propia	51
51	Pruebas de integración: Resultados escenarios removeFromCart. Fuente: Elaboración propia	52
52	Pruebas de integración: Resultados pasos removeFromCart. Fuente: Elaboración propia	53
53	Gráfico resultados: Acceso a catálogo. Fuente: Elaboración propia	54
54	Gráfico errores: Acceso a catálogo. Fuente: Elaboración propia	55
55	Gráfico tiempos: Acceso a catálogo. Fuente: Elaboración propia	55
56	Gráfico resultados: Acceso a ficha de producto. Fuente: Elaboración propia	56
57	Gráfico errores: Acceso a ficha de producto. Fuente: Elaboración propia .	56
58	Gráfico tiempos: Acceso a ficha de producto. Fuente: Elaboración propia	56
59	Gráfico resultados: Cambio de tipo de producto. Fuente: Elaboración propia	57
60	Gráfico errores: Cambio de tipo de producto. Fuente: Elaboración propia	58
61	Gráfico tiempos: Cambio de tipo de producto. Fuente: Elaboración propia	58
62	Gráfico resultados: Filtrado de productos. Fuente: Elaboración propia . .	59

63	Gráfico errores: Filtrado de productos. Fuente: Elaboración propia	59
64	Gráfico tiempos: Filtrado de productos. Fuente: Elaboración propia . . .	59
65	Gráfico resultados: Agregar al carro. Fuente: Elaboración propia	60
66	Gráfico errores: Agregar al carro. Fuente: Elaboración propia	61
67	Gráfico tiempos: Agregar al carro. Fuente: Elaboración propia	61
68	Gráfico resultados: Remover producto del carro de compras. Fuente: Elaboración propia	62
69	Gráfico errores: Remover producto del carro de compras. Fuente: Elaboración propia	62
70	Gráfico tiempos: Remover producto del carro de compras. Fuente: Elaboración propia	63
71	Gráfico resultados: Flujo completo de compra. Fuente: Elaboración propia	64
72	Gráfico errores: Flujo completo de compra. Fuente: Elaboración propia .	64
73	Gráfico tiempos: Flujo completo de compra. Fuente: Elaboración propia .	64
74	Accesos simultáneos: Gráfico 5 accesos. Fuente: JMeter	66
75	Accesos simultáneos: Tabla resultados 5 accesos. Fuente: JMeter	66
76	Accesos simultáneos: Gráfico 10 accesos. Fuente: JMeter	66
77	Accesos simultáneos: Tabla resultados 10 accesos. Fuente: JMeter	66
78	Accesos simultáneos: Gráfico 20 accesos. Fuente: JMeter	67
79	Accesos simultáneos: Tabla resultados 20 accesos. Fuente: JMeter	67
80	Accesos simultáneos: Gráfico 50 accesos. Fuente: JMeter	67
81	Accesos simultáneos: Tabla resultados 50 accesos. Fuente: JMeter	67
82	Accesos simultáneos: Gráfico 100 accesos. Fuente: JMeter	68
83	Accesos simultáneos: Tabla resultados 100 accesos. Fuente: JMeter	68
84	Accesos simultáneos: Tabla resultados 120 accesos. Fuente: JMeter	69
85	Consultas simultáneas: Gráfico 3 accesos. Fuente: JMeter	69

86	Consultas simultáneas: Tabla resultados 3 accesos. Fuente: JMeter	70
87	Consultas simultáneas: Gráfico 5 accesos. Fuente: JMeter	70
88	Consultas simultáneas: Tabla resultados 5 accesos. Fuente: JMeter	70
89	Consultas simultáneas: Gráfico 8 accesos. Fuente: JMeter	70
90	Consultas simultáneas: Tabla resultados 8 accesos. Fuente: JMeter	71
91	Consultas simultáneas: Gráfico 10 accesos. Fuente: JMeter	71
92	Consultas simultáneas: Tabla resultados 10 accesos. Fuente: JMeter	71
93	Consultas simultáneas: Gráfico 15 accesos. Fuente: JMeter	71
94	Consultas simultáneas: Tabla resultados 15 accesos. Fuente: JMeter	72
95	Consultas simultáneas: Tabla resultados 16 accesos. Fuente: JMeter	72

ÍNDICE DE TABLAS

1	Tareas historia de usuario HU-01. Fuente: Elaboración propia.	28
2	Tareas historia de usuario HU-02. Fuente: Elaboración propia.	28
3	Tareas historia de usuario HU-03. Fuente: Elaboración propia.	28
4	Tareas historia de usuario HU-04. Fuente: Elaboración propia.	29
5	Tareas historia de usuario HU-05. Fuente: Elaboración propia.	29
6	Tareas historia de usuario HU-06. Fuente: Elaboración propia.	29
7	Tareas historia de usuario HU-07. Fuente: Elaboración propia.	29
8	Tareas historia de usuario HU-08. Fuente: Elaboración propia.	30
9	Tareas historia de usuario HU-09. Fuente: Elaboración propia.	30
10	Tareas historia de usuario HU-10. Fuente: Elaboración propia.	30
11	Tareas historia de usuario HU-11. Fuente: Elaboración propia.	31
12	Tareas historia de usuario HU-12. Fuente: Elaboración propia.	31
13	Tareas historia de usuario HU-13. Fuente: Elaboración propia.	31

14	Tareas historia de usuario HU-14. Fuente: Elaboración propia.	32
15	Tareas historia de usuario HU-15. Fuente: Elaboración propia.	32
16	Tareas historia de usuario HU-16. Fuente: Elaboración propia.	32
17	Inputs: Acceso al catálogo de productos. Fuente: Elaboración propia. . .	54
18	Inputs: Acceso a ficha de producto. Fuente: Elaboración propia.	55
19	Inputs: Cambio de tipo de producto. Fuente: Elaboración propia.	57
20	Inputs: Filtrado de productos. Fuente: Elaboración propia.	58
21	Inputs: Agregar producto al carro de compras. Fuente: Elaboración propia.	60
22	Inputs: Remover producto desde el carro de compras. Fuente: Elaboración propia.	62
23	Inputs: Flujo completo de compra. Fuente: Elaboración propia.	63
24	Tabla comparativa: Resultados entregados por cada herramienta. Fuente: Elaboración propia.	73
25	Tabla comparativa: Versatilidad de cada herramienta. Fuente: Elaboración propia.	74
26	Tabla comparativa: Documentación disponible de cada herramienta. Fuente: Elaboración propia.	75
27	Tabla comparativa: Escalabilidad de cada herramienta. Fuente: Elaboración propia.	76

INTRODUCCIÓN

Las pruebas de software surgieron como respuesta a los errores detectados tras la ejecución de un programa o código, desde el conocido caso del insecto encontrado al interior de una de las primeras computadoras, los enfoques del testing fueron evolucionando a medida que transcurría el tiempo. En los comienzos del desarrollo de software no existía una fase de pruebas como tal, tras detectar un error en el producto, el protocolo era realizar un proceso de debugging en el código con el fin de encontrar el motivo del fallo. Con el pasar de los años, se comenzó a trabajar con un proceso de pruebas destinado a demostrar que el producto funcionaba correctamente, empleando casos de prueba sencillos que permitieran cumplir con este propósito, casos que se volvieron más robustos a medida que variaba el enfoque, llegando a lo que hoy en día conocemos. Las pruebas de software fueron tomando casa vez más importancia dentro del desarrollo, volviéndose un indicador importante de la calidad del producto.

Las metodologías de desarrollo de software han experimentado un proceso histórico y evolutivo que inicia en los años 40, entonces no se contaban con parámetros ni estándares, el desarrollo era prácticamente empírico y artesanal, lo que llevó a que una buena parte de los proyectos fallaran por inconformidad del cliente o errores durante su uso. El desarrollo de las pruebas se fue situando cada vez más dentro de las metodologías que surgieron, llegando al punto en que estas se abarcan incluso antes del desarrollo del software mismo.

A lo largo de este documento se abordará la problemática surgida por el desconocimiento y/o mal empleo de herramientas de software durante el desarrollo de soluciones tecnológicas. Como base se definirá el objetivo general y los específicos, a conseguir por medio de esta investigación. En primera instancia se abordará el marco conceptual que se utilizó como base para el estudio. Luego se explicará la propuesta de solución que incluye el desarrollo de un sitio web de Ecommerce, para llevar a cabo un plan de pruebas con cada herramienta de testeo y, por otro lado, la estructura de la evaluación a realizar sobre cada utensilio. A continuación se mostrarán los resultados, obtenidos en cada plan de pruebas, a modo de validación de la propuesta junto con la evaluación de las herramientas empleadas. Finalmente, se procederá a concluir sobre los resultados y comportamientos observados durante el desarrollo del estudio.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

Las pruebas de software a lo largo del desarrollo se han vuelto un proceso indispensable, dado que evalúan una solución tecnológica desde varios aspectos y dictan lo que se debe hacer a continuación. Es por lo anterior y, debido a la inmensa variedad de lenguajes de programación y frameworks de desarrollo, que surge cierta incertidumbre durante la elección de herramientas para llevar a cabo esta tarea.

Cada proyecto de software se construye a partir de diversos requisitos, ya sean funcionales o no funcionales. Dependiendo del área para la cual se desarrolle un software, se pueden presentar requisitos no funcionales establecidos a partir de las necesidades del cliente, un ejemplo de lo anterior son las aplicaciones de shopping y venta de productos, que requieren de un servidor capaz de soportar una gran cantidad de usuarios navegando simultáneamente, mientras que aplicaciones de la banca necesitan que se cumplan complejos protocolos de seguridad.

El hecho de que una aplicación posea dificultades para su uso o tiempos de respuesta prolongados, puede significar que los usuarios pasarán al competidor y, que a su vez, publiquen críticas negativas. Lo anterior, provoca pérdidas monetarias moderadas, dado que se debe optar por realizar una fase de pruebas y codificación nuevamente. La seguridad es otro aspecto importante que se puede ver afectado por errores, dando incluso el pase a la pérdida de información comprometedoras para la empresa. Los errores de software fueron responsables de la mayoría de los desarrollos fallidos en el gobierno, el entretenimiento, las finanzas, el comercio minorista, los servicios y el transporte en 2016 [Spaven, 2017], en la figura 1 se muestra un resumen de los distintos errores y problemas detectados en software durante este año:

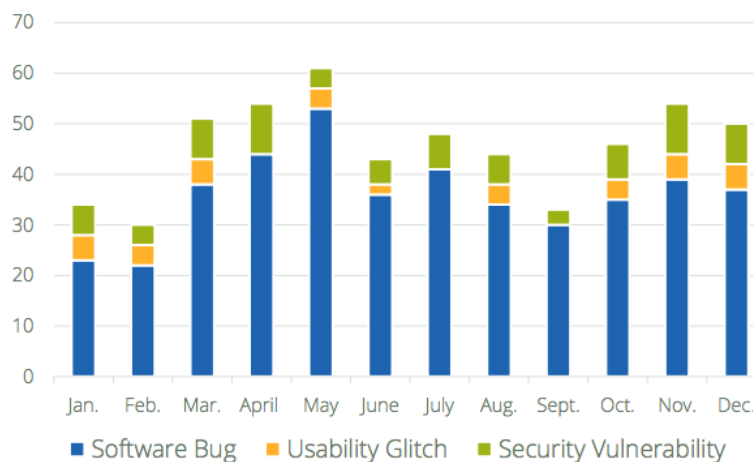


Figura 1: Software errors during 2016. Fuente: www.dzone.com

Durante el proceso de desarrollo del producto, se deben construir pruebas con distintos alcances, dentro de las cuales se encuentran las pruebas unitarias, de integración, de sistema, de rendimiento y seguridad. Para lo anterior, se debe considerar también el modelo de desarrollo y la metodología utilizada. A modo de ejemplo, en la figura 2 se muestra el modelo de desarrollo en cascada:

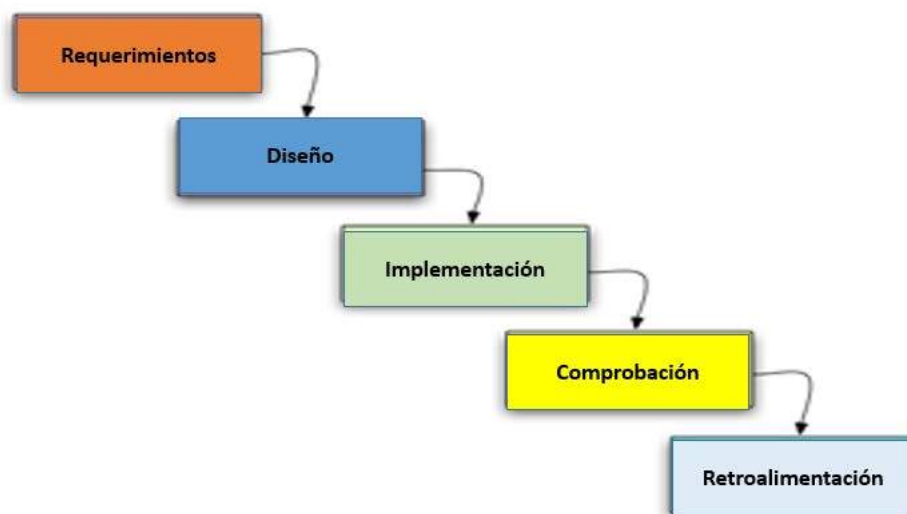


Figura 2: Modelo en cascada. Fuente: www.researchgate.net

Es importante mencionar que el monto de las pérdidas depende de diversas variables, algunas de ellas son el tipo de error/falla, la cantidad de clientes que posea la empresa y el tiempo de demora en detectar el error. Un ejemplo del tiempo de demora puede darse para el caso de una tienda de ventas en línea que posea una falla en su sistema de pagos, dando el pase a los clientes aún cuando estos no hayan realizado el procedimiento, esto puede significar el surgimiento de la necesidad de requerir un ejecutivo que revise cada orden para comprobar los pagos o incluso la pérdida total de un producto dado que el sistema retorna que el pago fue realizado con éxito y, por ende, se concreta el despacho del producto.

Un software desplegado en un ambiente productivo conlleva un largo periodo de desarrollo, la detección de una falla al momento de la puesta en marcha significa una pérdida considerable para la empresa, dado que se requiere de un nuevo periodo de implementación y existe un costo hundido debido a que no se puede recuperar la inversión. Un estudio llevado a cabo por Tricentis, una empresa de pruebas de software, refleja que en el año 2017 hubo un estimado de 1.7 billones de dólares en pérdidas debido a errores de software [Tricentis, 2018], este reporte fue llevado a cabo abarcando 606 fallos de software en 314 compañías distintas, entregando también como resultado la cifra de 3.6 billones de personas afectadas.

Los errores de software pueden tardar un tiempo en ser detectados, aún cuando la

aplicación ya se encuentra al alcance de los usuarios. CGTechnology, una empresa de apuestas deportivas, tuvo que pagar una multa de 1.5 millones de dólares y despedir a su CEO luego de ignorar una falla en su aplicación que provocó un pago menor al que le correspondía a los apostantes ganadores. En este ámbito es de vital importancia conocer las herramientas que se deben emplear y el alcance que posean estas para abordar todos los casos, pero también se debe tener presente que un gran porcentaje de este trabajo recae en la planificación y definición de los procedimientos de prueba que deben realizarse, por lo que no debe descartarse una inversión en un profesional con conocimientos sólidos en calidad de software y/o testing.

Dadas las variables mencionadas anteriormente, es aquí donde surge la problemática de qué tecnologías o bibliotecas utilizar para desarrollar a completitud el plan de pruebas definido, junto con la incertidumbre de cuál va a ser la que mayor seguridad entregue al momento de finalizar esta fase, debido a que no existe software capaz de detectar de manera automática todos los casos de uso posibles y testearlos según corresponda.

Los errores de software pueden ser detectados en cualquier etapa del desarrollo del producto, pero estas etapas poseen una diferencia relevante entre sí, que recae en el costo que tiene el corregir un error dentro de ellas, en la figura 3 se explica, a escala, el costo asociado a la corrección de fallas en cada fase [Mireles, 2018].

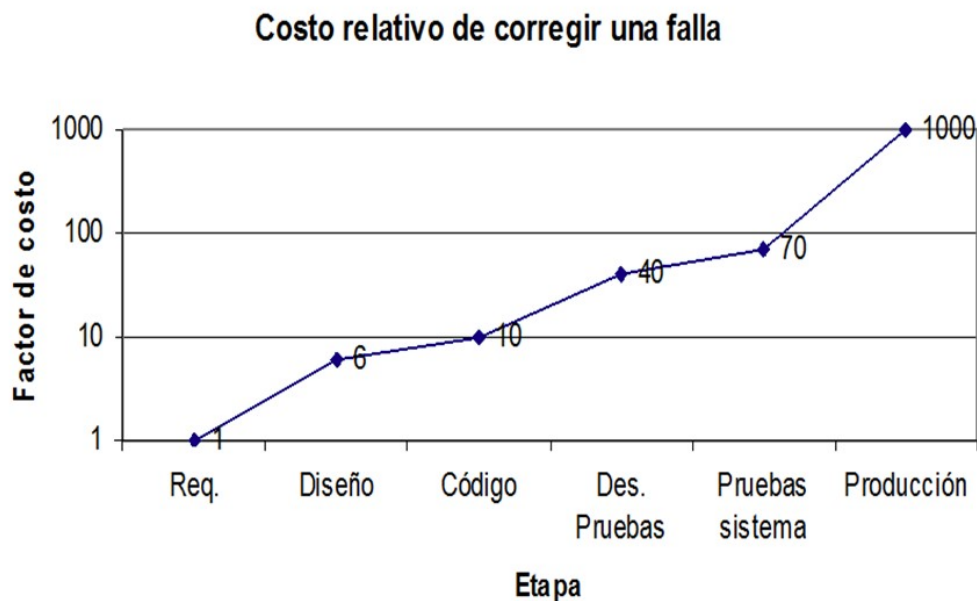


Figura 3: Costo asociado a la corrección de errores en cada etapa del desarrollo.

Fuente: www.mat.uson.mx

Analizando el gráfico, en primera instancia se extrae que la pérdida generada por corregir un error en una etapa avanzada del desarrollo es significativamente mayor al que conlleva el realizar esta acción durante la toma de requisitos, esto se debe a que en las

siguientes fases se debe añadir un proceso de apoyo como consecuencia de la falla, los factores que influyen en esta pérdida son variados, puede deberse al fin del contrato con los profesionales o la extensión de este, por otro lado en algunos casos se puede dar que se deban repetir pasos anteriores.

En este sentido, se deben evaluar diversas opciones de herramientas de testing para construir un modelo de pruebas, tomando en cuenta el conjunto de herramientas que se utilice y los requisitos que debe cumplir la aplicación. Lo anterior se puede volver un evento complejo, si se da el caso de que se deba implementar una aplicación que no posea semejanzas con desarrollos previos.

En la figura 4 se grafica el árbol del problema que sintetiza lo expuesto a lo largo de esta sección:

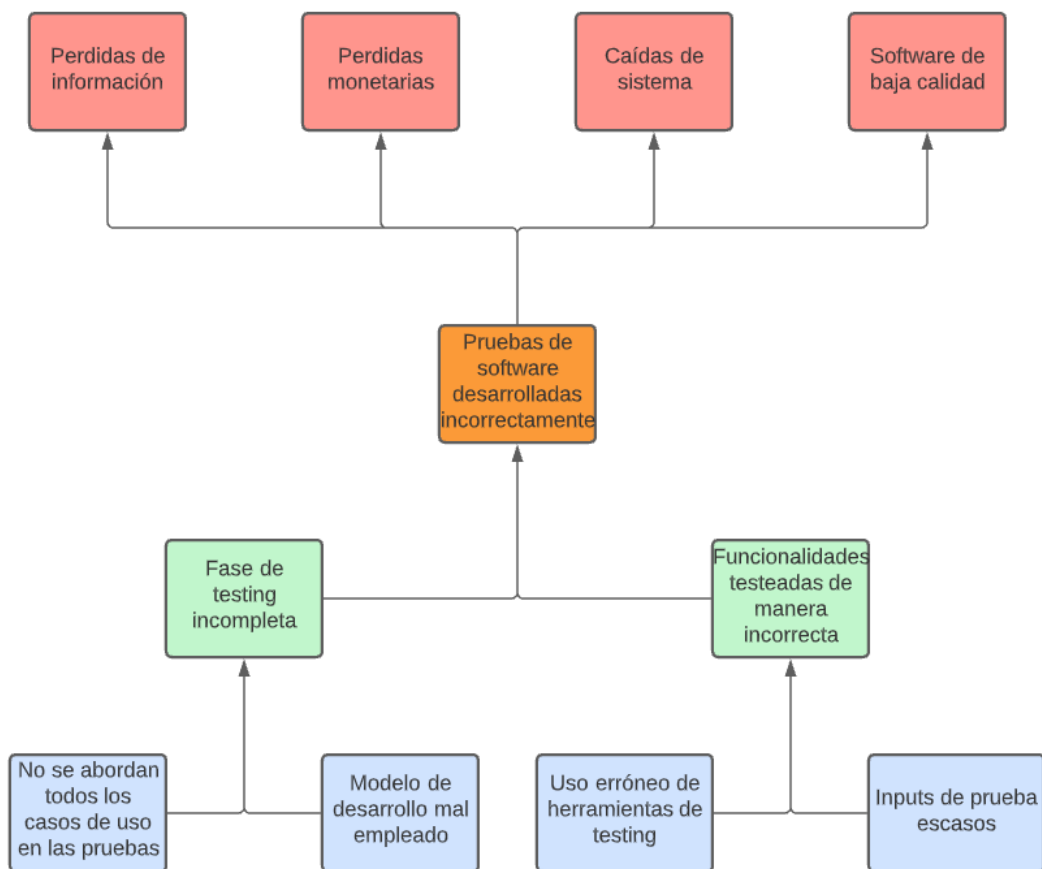


Figura 4: Árbol de problema. Fuente: Elaboración propia

1.1. Objetivos

Objetivo general: Evaluar las principales herramientas de testeo que existen en el mercado, estableciendo las instancias puntuales en las que deben ser utilizadas y los respectivos usos que se les puede dar para cada caso, según los requisitos que deba cumplir la aplicación web, de tal manera de asegurar su calidad y disminuir costos asociados al tiempo, beneficiando a empresas pertenecientes al rubro del comercio electrónico.

1.1.1. Objetivos específicos

- Proponer un modelo de elección de herramientas de testeo, que cumpla con lo solicitado para desplegar o continuar con el desarrollo de una aplicación web.
- Definir, por medio de una evaluación técnica, las herramientas de testeo que mejor se desempeñan en cada conjunto de tecnologías.
- Relacionar los requerimientos de las aplicaciones web a las herramientas de testeo que posean el mejor desempeño para cada caso.
- Evaluar la compatibilidad entre herramientas de testeo, estableciendo combinaciones que optimicen los resultados.
- Estimar la calidad de los resultados entregados por las herramientas de testeo al utilizarlas sobre una aplicación web.

CAPÍTULO 2

MARCO CONCEPTUAL

A continuación, se procederá a describir el marco conceptual que actuará como fundamento teórico para el estudio y evaluación de las principales herramientas de testeo. Se compondrá de los siguientes ejes:

- Tecnologías de desarrollo
- Pruebas de software
- Herramientas de prueba
- Problemáticas durante las pruebas de Software

2.1. Tecnologías de desarrollo

A lo largo de este eje se abordaran las principales tecnologías empleadas actualmente en el mercado de desarrollo de software, dando énfasis en las utilizadas para el desarrollo de las aplicaciones web. Se elegirá un conjunto de estas para llevar a cabo el desarrollo de una aplicación de prueba que servirá como medio de evaluación de las herramientas de testeo.

2.1.1. Phalcon

Phalcon es un Framework de código abierto de pila completa para *PHP*, escrito como una extensión del lenguaje *C*. *Phalcon* está optimizado para un alto rendimiento. Su arquitectura única permite que el marco sea siempre residente en la memoria, ofreciendo su funcionalidad siempre que sea necesario, sin estadísticas de archivos costosas y lecturas de archivos que emplean los Frameworks de *PHP* tradicionales.

Los desarrolladores no necesitan conocer el lenguaje *C* para usar el Framework. *Phalcon* está acoplado de manera flexible, lo que permite a los desarrolladores usar solo los objetos que necesitan como componentes de pegamento en función de las necesidades de sus aplicaciones [Phalcon, 2021].

2.1.2. Java Spring Boot

Spring Boot [Spring, 2021] es una solución para el framework Spring de *Java* que sigue el principio de “convención sobre configuración” y reduce la complejidad del desarrollo de nuevos proyectos basados en *Spring*. Para ello, *Spring Boot* proporciona la estructura básica configurada del proyecto, que incluye las pautas para usar el marco y todas las bibliotecas de terceros relevantes para la aplicación. De esta manera se simplifica mucho la creación de aplicaciones independientes y reproducibles, por lo que la mayoría de las nuevas aplicaciones basadas en Spring se desarrollan con *Spring Boot*. Las principales características de este se presentan a continuación:

- Incorporación directa de aplicaciones de servidores web y/o contenedores como *Apache Tomcat* o *Jetty*, eliminando la necesidad de incluir archivos WAR (Archivos Web).
- Simplificación de la configuración de *Maven* gracias a los POM iniciales.
- Configuración automática de Spring.
- Características no funcionales, como métricas o configuraciones externalizadas.

2.1.3. NodeJS

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación *JavaScript*, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, *NodeJS* está diseñado para crear aplicaciones web escalables.

2.1.4. VueJs

Es un framework progresivo para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, *Vue* está diseñado desde cero para ser utilizado incrementalmente. La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, *Vue* también es perfectamente capaz de impulsar sofisticadas aplicaciones de una página cuando se utiliza en combinación con herramientas modernas y librerías de apoyo.

En [codingpotions, 2019] se describen ciertas características del framework, a continuación se presentan las más destacables:

- **Modularidad:** Es una herramienta completamente modular, es decir, entrega al usuario los componentes básicos y le da libre albedrío para elegir que otras herramientas utilizar para complementar.
- **Reactividad:** Si cambia una variable en una parte de la vista de la página, *Vue* actualizará su nuevo valor sin necesidad hacerlo manualmente.
- **Componentes web:** *Vue* se basa en componentes web. Un componente web es una parte de una web que puede ser reutilizada, que normalmente tiene estilos y funcionalidad aislada.
- **Virtual DOM:** En caso de requerir un cambio en la vista, en lugar de sustituir directamente los nuevos valores en esta, *Vue* crea una réplica del DOM, es decir, de los elementos de la página web.

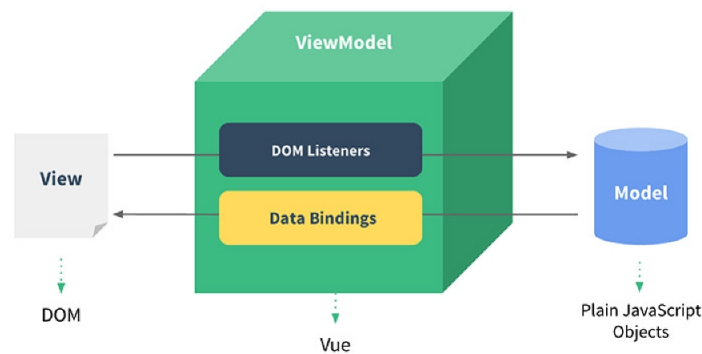


Figura 5: Arquitectura Vue. Fuente: <https://avimehenwal.in/blog/vue-frontend-renderer/>

2.1.5. AngularJS

Es un framework de *JavaScript* de código abierto que extiende el HTML, DOM con atributos adicionales. Este framework ayuda a hacer que los elementos sean escalables y receptivos a las acciones del usuario, también a usar HTML como un lenguaje de plantilla, ofreciendo a los desarrolladores una aplicación del lado del cliente. Básicamente, la intención detrás del desarrollo de este framework es simplificar el proceso de desarrollo web [Argos, 2020]. El framework también ofrece varias directivas para personalizar las funciones para que se adapten al flujo de trabajo de desarrollo de una empresa y a las necesidades de funciones. Las principales características del framework son:

- **Enlace de datos bidireccional:** El enlace de datos bidireccional permitió a los desarrolladores separar la lógica de la aplicación del modelo de objetos de documento; por lo tanto, facilita el proceso de prueba realizado por el desarrollador.

- Manejo fluido de HTML: Los diseñadores y desarrolladores pueden trabajar en sincronía, ya que HTML se utiliza para crear plantillas. Además, las extensiones en *AngularJS* a HTML se realizan mediante ng-directives (como ng-bind, ng-controller, ng-class, etc.).
- Framework apto para pruebas: Excelente preparación para las pruebas unitarias, *AngularJS* hace posible la comunicación basada en el contexto.
- Código reutilizable: Adopta el estándar de componentes web. Mediante un conjunto de APIs es factible generar etiquetas HTML personalizadas y reutilizar código.

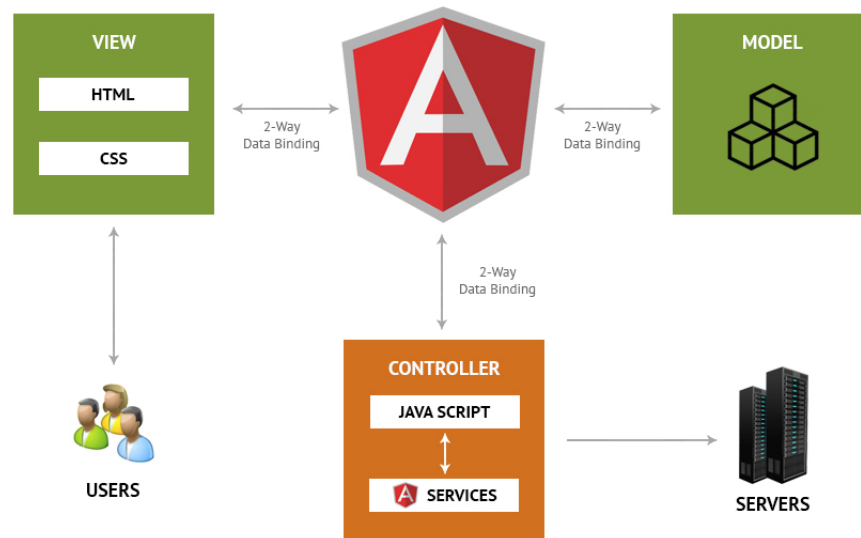


Figura 6: Arquitectura Angular. Fuente: <https://www.matridtech.net/>

2.2. Pruebas de software

Corresponden a un conjunto de procesos con los que se pretende probar el sistema o aplicación en diferentes momentos para comprobar su correcto funcionamiento. Este tipo de pruebas abarca cualquier estado del desarrollo del sistema, desde su creación hasta su puesta en el ambiente de producción, se estima que cerca del 50 % del tiempo y esfuerzo del proceso de desarrollo de software es aplicado en las pruebas [Umar, 2019]. A lo largo de la investigación se trabajará con pruebas de software automatizadas, para determinar la efectividad de estas sobre un sistema predefinido.

2.2.1. Tipos de pruebas de Software

Las pruebas de software se pueden dividir en dos grupos diferentes: pruebas funcionales y no funcionales. Diferentes aspectos de una aplicación de software requieren diferentes

tipos de pruebas, como pruebas de rendimiento, pruebas de escalabilidad, pruebas de integración, pruebas unitarias y muchas más.

- Pruebas funcionales: Se llevan a cabo para comprobar las características críticas para el negocio, la funcionalidad y la usabilidad. Las pruebas funcionales garantizan que las características y funcionalidades del software se comportan según lo esperado, sin ningún problema. Valida principalmente toda la aplicación con respecto a las especificaciones mencionadas en el documento *SRS* (Software Requirement Specification).
- Pruebas no funcionales: Son similares a las funcionales, sin embargo, la principal diferencia es que esas funciones se prueban bajo carga para el rendimiento de los observadores, fiabilidad, usabilidad, escalabilidad, etc. Las pruebas no funcionales, como las pruebas de carga y esfuerzo, normalmente se llevan a cabo mediante herramientas y soluciones de automatización.

2.2.2. Ciclo de vida del desarrollo de software

El ciclo de vida del desarrollo de software (*SDLC – Systems Development Life Cycle*) es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de software, abarcando la vida completa del sistema, desde la definición de los requisitos hasta la finalización de su uso. Dentro del ciclo de desarrollo se encuentran las siguientes fases:

- Comunicación: Instancia en la que un cliente solicita un producto de software determinado.
- Planificación y análisis: El desarrollo de software comienza con una fase inicial de planificación, incluyendo un análisis de los requisitos presentados por el cliente. En esta fase se indaga en profundidad y se hacen demostraciones prácticas incluyendo a los usuarios clave.
- Estudio de viabilidad: Después de la recolección de requisitos, se idea un plan para procesar el software. Se analiza que parte del software cubre los requisitos de cada usuario.
- Análisis del sistema: En este paso se asignan los recursos y se planifica el tiempo de duración del proyecto.
- Diseño Software: El siguiente paso es diseñar el producto software con la ayuda de toda la información recogida sobre requisitos y análisis. Los inputs de los usuarios y los resultados de la recogida de información hecha en la fase anterior serán las aportaciones base de la fase actual.

- **Codificación:** La implementación del diseño de software empieza con el lenguaje de programación más conveniente, y desarrollando programas ejecutables y sin errores de manera eficiente.
- **Pruebas:** Las pruebas de Software se hacen mientras se codifica y suelen hacerlas los desarrolladores y otros expertos evaluadores a varios niveles. Esto incluye evaluación de módulos, evaluación del programa, evaluación del producto, evaluación interna y finalmente evaluación con el consumidor final.
- **Integración:** El Software puede necesitar estar integrado con bibliotecas, Bases de datos o con otro u otros programas. Esta fase del *SDLC* se focaliza en la integración del software con las entidades del mundo exterior.
- **Implementación:** Se realiza la instalación del software en el ambiente entregado por el cliente.
- **Formación:** Fase de capacitación para el usuario del producto de software desarrollado.
- **Mantenimiento y Funcionamiento:** Esta fase confirma el funcionamiento del software en términos de más eficiencia y menos errores.

El ciclo puede variar según la empresa y/o las herramientas empleadas para el desarrollo de la solución tecnológica.

2.2.3. Paradigma de desarrollo de software

El paradigma de desarrollo de Software corresponde a la estrategia para llevar a cabo la construcción del producto. El paradigma tiene su propio conjunto de herramientas, métodos y procedimientos, los cuales son expresados de forma clara. A continuación se presentan algunos paradigmas o modelos de proceso:

- **Modelo de Cascada:** Modelo más simple en el desarrollo de software, sigue una estructura en que las fases del ciclo funcionan en secuencia lineal, de manera que se sigue en orden para ejecutarlas una a una.
- **Modelo repetitivo:** Proyecta el proceso de desarrollo de forma cíclica repitiendo cada paso después de cada ciclo en el proceso de *SDLC*.
- **Modelo en espiral:** Corresponde a una mezcla de ambos modelos anteriormente mencionados. Se puede ver como si se combina un modelo de *SDLC* combinado con un proceso cíclico (modelo repetitivo).

- Modelo V: Modelo que aporta opciones de evaluación del software en cada etapa de manera inversa. En cada etapa, se crea la planificación de las pruebas y los casos de pruebas para verificar y validar el producto según los requisitos.
- Scrum: Metodología de desarrollo ágil de software que ayuda a los equipos a desarrollar productos en periodos cortos, permitiendo obtener de forma rápida feedback por parte del cliente, adaptaciones y una mejora continuada [eniun, 2021].

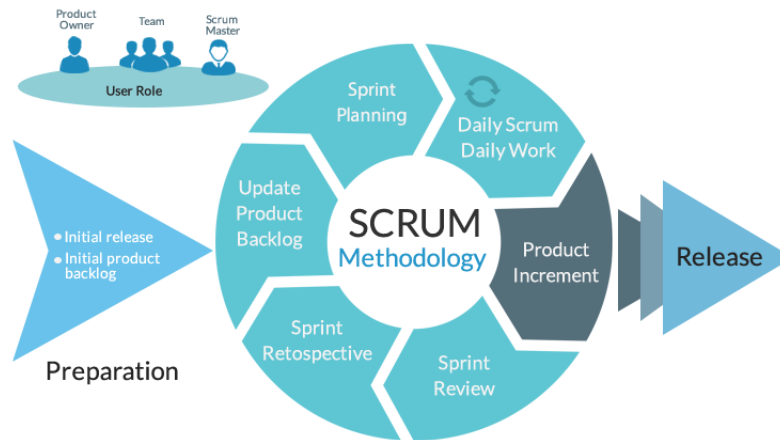


Figura 7: Ciclo metodología Scrum. Fuente:
<https://blog.wearedrew.co/ventajas-y-desventajas-de-la-metodologia-scrum>

2.2.4. Niveles de prueba

Al momento de aplicar pruebas a un software, se tienen en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en diferentes etapas del proceso de desarrollo. En [Umar, 2019] se definen los siguientes niveles de prueba:

- Pruebas unitarias: Corresponden a las pruebas aplicadas sobre las porciones más pequeñas del software, por ejemplo como módulos o funciones, con el objetivo de verificar su funcionamiento según su especificación. La unidad puede ser un constructor o su contra parte, a nivel de clases en un ambiente orientado a objetos. Algunos ejemplos de pruebas unitarias son las de control de flujo y las de flujo de datos.
- Pruebas de integración: Envuelven a dos o más unidades combinadas que deben trabajar en conjunto de manera que se asegure un flujo de control, o de datos, sin

errores. Una interfaz de usuario, interacciones o casos de uso son algunos ejemplos de pruebas de integración.

- Pruebas de sistema: Son las pruebas aplicadas sobre un software completamente ensamblado, de tal manera que se verifique el cumplimiento de sus requerimientos. Estas pruebas validan las interacciones entre componentes para asegurar el funcionamiento unánime de todos los módulos y programas sin errores.
- Pruebas de aceptación: Estas pruebas se realizan para validar el software frente a los requisitos del cliente. Se realizan para garantizar que el software haga lo que el cliente quiere que haga y verificar la aceptabilidad del sistema.

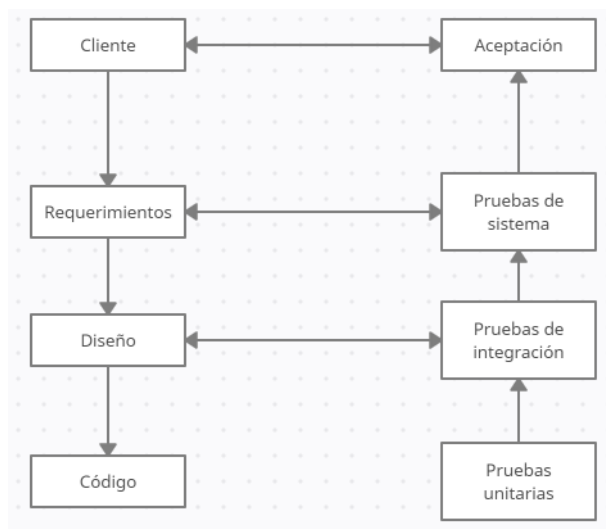


Figura 8: Niveles de pruebas de software. Fuente: [Umar, 2019]

2.2.5. Modelo de calidad

La calidad de un producto de software se ha definido de manera formal dentro de la familia de normas ISO/IEC 25000, en las cuales se establece un marco de trabajo común para la evaluación de calidad, siendo la principal base para el área de calidad de la Ingeniería de Software [Rodríguez, 2015]. En la norma ISO/IEC 25010 se definen algunos modelos de calidad y se determinan características o factores que se pueden evaluar referentes a la calidad del software, tales como:

- Adecuación funcional: hace referencia a la capacidad que presenta un producto de software para contar con funcionalidades que cumplan con las necesidades del cliente, en las condiciones específicas de uso estimadas.

- Rendimiento: Es una característica representativa al desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones, como el caso del tiempo de respuesta o la capacidad de un elemento de software para cumplir con su requerimiento al límite máximo.
- Compatibilidad: Se define entre dos o más sistemas o componentes, al compartir el mismo entorno de hardware o de software pueden intercambiar información y cumplir con funciones requeridas. La coexistencia e interoperabilidad es necesaria para que un software comparta sus recursos e información con otro sistema, en un entorno común, sin ponerse en riesgo el uno al otro.
- Usabilidad: Capacidad del software para ser entendido, aprendido, usado y resultar atractivo para el usuario en determinadas condiciones. También hace referencia a la capacidad de uso, de aprendizaje del producto por parte del usuario, su nivel de protección o recuperación de errores provocados por el usuario, su interfaz y accesibilidad.
- Fiabilidad: El sistema cumple con su objetivo en condiciones normales, este operativo y accesible cuando se le requiera, continúe con su operación pese a fallos que se presenten en hardware o software recuperando datos afectados y restableciendo a un estado deseado.
- Seguridad: Representa la capacidad de mantener bajo protección la información del sistema, de tal manera que los datos de sí mismo y de los usuarios involucrados no puedan ser leídos o modificados por personas no autorizadas. La seguridad hace referencia a la confidencialidad, integridad, no repudio, responsabilidad y autenticidad.
- Portabilidad: Capacidad del producto software para ser transferido de forma efectiva y eficiente de un entorno a otro, presentando adaptabilidad, facilidad para ser instalado y reemplazado por otro producto software con el mismo propósito y en el mismo entorno.
- Mantenibilidad: Capacidad del producto software para ser modificado de manera efectiva y eficiente, ya sea por necesidades evolutivas, correctivas o perfectivas.

2.2.6. API de REST

Una API de transferencia de estado representacional es una interfaz de programación de aplicaciones creada por el informático Roy Fielding[RedHat, 2020], la cual se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el



Figura 9: Modelo de calidad ISO/IEC 25010. Fuente: ISO/IEC 25010

proveedor de información y el usuario, donde se establece el contenido que se necesita del consumidor (la llamada) y el que requiere el producto (la respuesta). Una de las ventajas de uso es que no se necesita saber cómo se recibe el recurso ni de dónde proviene.

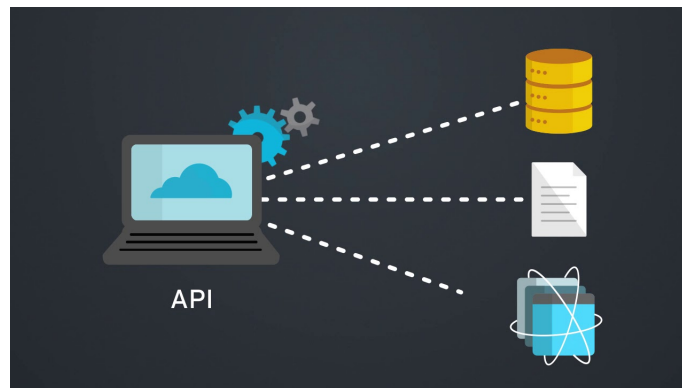


Figura 10: Diagrama API REST. Fuente: <https://www.redhat.com>

2.3. Herramientas de prueba

2.3.1. Cucumber

Cucumber es una herramienta que apoya el desarrollo impulsado por el comportamiento (BDD), es decir, lee las especificaciones ejecutables escritas en texto plano, valida que el software hace lo que dicen esas especificaciones y genera un informe que indica éxito o fracaso para cada escenario. Las especificaciones pueden constar de varios ejemplos o escenarios. Cada escenario es una lista de pasos que debe seguir Cucumber.

Para que Cucumber comprenda los escenarios, se deben seguir algunas reglas de sintaxis básicas, llamadas Gherkin, las cuales corresponden a un conjunto de reglas gramaticales que hacen que el texto sin formato esté lo suficientemente estructurado para que

Cucumber lo entienda. Este conjunto de reglas tiene múltiples propósitos:

- Especificación ejecutable inequívoca
- Pruebas automatizadas
- Documentar cómo se comporta realmente el sistema

La gramática de Gherkin existe en diferentes estilos para muchos idiomas hablados, lo anterior permite a los equipos de trabajo utilizar el más adecuado según corresponda.

La conexión entre la especificación y la implementación se denomina como definición de paso. Las definiciones de pasos conectan los pasos de *Gherkin* con el código de programación, cuando Cucumber ejecuta un paso *Gherkin* en un escenario, buscará una definición de paso coincidente para ejecutar. Por ejemplo, en el caso de *Java*, una definición de paso es un método con una expresión que lo vincula a uno o más pasos de *Gherkin*.

2.3.2. Selenium

Selenium es un software de prueba automatizado de código abierto, que se utiliza para validar aplicaciones web en diferentes navegadores y plataformas. Se pueden utilizar varios lenguajes de programación como Java, C#, Python, entre otros, para crear scripts de prueba. Las pruebas realizadas con la herramienta Selenium generalmente se denominan como Selenium Testing. Dentro de las funcionalidades que facilita Selenium, se pueden destacar:

- Compatibilidad con gran variedad de navegadores
- Ingreso automatizado de datos e interacciones con la interfaz
- Selección de elementos del sitio por medio de parámetros de este
- Obtención de información desde el sitio para utilizar durante las pruebas

2.3.3. JMeter

Apache JMeter es un software de código abierto desarrollado utilizando el lenguaje de programación Java, permite al usuario realizar pruebas sobre su aplicación web por medio de una gran variedad de utensilios.

JMeter se puede utilizar para probar el rendimiento tanto en recursos estáticos como dinámicos en aplicaciones web dinámicas. Dentro de sus funcionalidades se encuentra la

de simular una carga pesada en un servidor, conjunto de servidores, red u objeto para probar su capacidad de respuesta o para analizar el rendimiento general bajo diferentes tipos de carga.

Las características incluidas dentro de JMeter son:

- Capacidad de cargar y probar el rendimiento de distintos tipos de desarrollos. Por ejemplo aplicaciones Web, servicios web SOAP (Simple Object Access Protocol) o REST (Representational State Transfer), servicios de transferencia de archivos (FTP), entre otros.
- Ambiente de desarrollo integrado (IDE) de pruebas que dispone de todas las funciones que permiten la grabación rápida del plan de prueba, ya sea desde navegadores o aplicaciones nativas.
- Modo línea de comandos (CLI) para cargar la prueba desde cualquier sistema operativo compatible con Java (Linux, Windows, MAX OSX, etc).
- Informe en HTML dinámico y preparado para su uso.
- Generar de manera sencilla una correlación al extraer datos desde los distintos formatos de respuesta más utilizados actualmente (HTML, JSON, XML u otro formato textual).
- Portabilidad debido a que se encuentra desarrollado en Java.
- El marco completo de subprocessos múltiples permite el muestreo simultáneo de muchos subprocessos y el muestreo simultáneo de diferentes funciones por grupos de subprocessos separados.
- Almacenamiento en caché y análisis o reproducción fuera de línea de los resultados de las pruebas.
- Posee un núcleo altamente extensible:
 - Los muestreadores conectables permiten capacidades de prueba ilimitadas.
 - Casos de prueba programables.
 - Permite elegir entre varias estadísticas de carga con temporizadores enlazables.
 - Los complementos de análisis y visualización de datos permiten una gran expansibilidad y personalización.
 - Las funciones se pueden aplicar para proporcionar entradas dinámicas en las pruebas.

2.4. Problemáticas durante las pruebas de Software

El proceso de desarrollar casos de prueba y realizar pruebas manuales o automatizadas es complejo. Los desarrolladores a menudo se encuentran con muchos desafíos a lo largo del ciclo de vida de las pruebas que hacen que todo el proceso tome un tiempo incluso superior al que se tarda en desarrollar una funcionalidad. En [channele2e, 2021] se definen problemáticas que se presentan durante este ciclo, a continuación se hace mención a algunas de ellas:

1. Estándares de calidad indefinidos: Existe una gran complejidad al momento de validar si los estándares de calidad están definidos correctamente, lo que hace que las pruebas sean menos eficaces. Los estándares de calidad indefinidos o mal definidos significan que no hay claridad en los requisitos, especificaciones, pautas o características de las pruebas. En ausencia de los estándares adecuados, se vuelve extremadamente difícil para los desarrolladores satisfacer las necesidades del cliente, cumplir con los requisitos de calidad y cumplir con las regulaciones.
2. Prueba de duplicación del entorno: Al probar un producto de software, la mayor parte de los equipos limitan sus esfuerzos a pocos casos de prueba. Sin embargo, esto no es una representación real de los desafíos que el producto podría enfrentar en un entorno del mundo real. No utilizar un entorno de prueba realista significa que los evaluadores solo encontrarán problemas que se reflejan en circunstancias muy específicas, lo que reduce la posibilidad de descubrir errores del mundo real o difíciles de encontrar.
3. Ambiente inestable: Otra problemática con respecto a las pruebas de software es la presencia de entornos de prueba inestables. Los entornos inestables tienden a interrumpir potencialmente el proceso de lanzamiento general. Dado que se gestionan de forma deficiente, generan conflictos y retrasos en la programación, a largo plazo, pueden afectar la calidad, la disponibilidad y la eficiencia de los entornos de prueba al tiempo que afectan los plazos y los costos de comercialización.
4. Recopilación de requisitos insuficientes: Un análisis de requisitos inadecuado o insuficiente tiene varias consecuencias sobre la calidad del software en desarrollo, dado que puede llevar a una funcionalidad inadecuada o incluso al descubrimiento de nuevos requisitos al final del ciclo de vida del desarrollo. Demasiados requisitos omitidos ejercen una enorme presión sobre los cronogramas del proyecto y obligan a los evaluadores a omitir los casos de prueba para ahorrar tiempo.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

En esta sección se procederá a describir la propuesta de solución a la problemática identificada, la cual consistirá en la elaboración y aplicación de un plan de pruebas sobre un ambiente elaborado específicamente para este estudio. A partir del análisis de los resultados se realizará una evaluación de las herramientas de testeo utilizadas. Cabe destacar que las pruebas serán llevadas a cabo durante el desarrollo de este ambiente, el cual será detallado a continuación.

3.1. Definición del ambiente

El ambiente de pruebas, de ahora en adelante conocido como APE (Ambiente de Pruebas Ecommerce), consistirá en una aplicación web basada en una tienda virtual de artículos tecnológicos tales como celulares, computadoras, entre otros. Lo anterior será construido siguiendo un patrón de arquitectura MVC, utilizando la tecnología VueJS para la parte visual (FrontEnd), los Frameworks Swagger y Express para el controlador (BackEnd) y PostgreSQL como motor de base de datos.

3.1.1. Controlador

Para el controlador del aplicativo se utilizará el Framework Swagger debido a su facilidad de uso, su escalabilidad reflejada al momento de añadir nuevos endpoints (URLs) y a las herramientas que ofrece para el desarrollo de una API REST. Por otro lado, se acompañará al framework anterior con Express para la lógica necesaria, cuya curva de aprendizaje refleja buenos resultados en poco tiempo.

3.1.2. Vista

La parte visual será implementada utilizando el Framework Vue en su versión número 3, el cuál posee una gran cantidad de librerías útiles para el desarrollo y puesta en marcha de la aplicación. Además, se cuenta con experiencia previa utilizando esta herramienta, lo que facilitará y reducirá los tiempos de desarrollo, a diferencia del uso de otras tecnologías.

3.1.3. Modelo

El modelo de datos será desplegado utilizando el motor de bases de datos PostgreSQL, esto debido a que es una herramienta gratuita (open source) y entrega estabilidad para la base de datos. Además posee una herramienta gráfica llamada pgAdmin, que permite administrar una base de datos de forma fácil e intuitiva, permitiendo también insertar datos directamente en las tablas.

En el siguiente diagrama se presenta el modelo de datos que será utilizado en el ambiente:

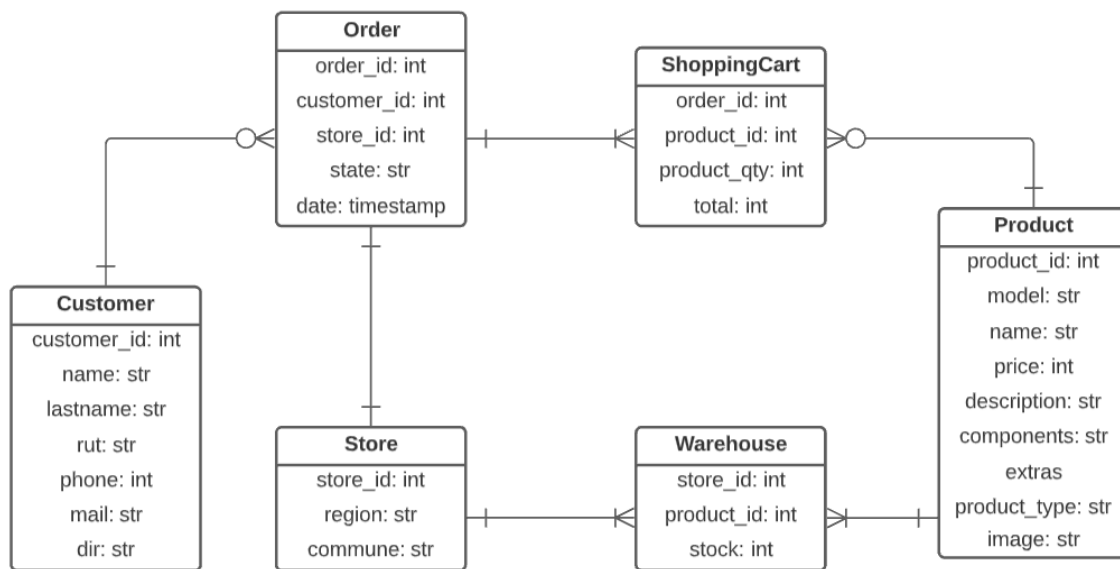


Figura 11: Modelo de datos. Fuente: Elaboración propia

3.1.4. Metodología de Desarrollo

Para el desarrollo de este estudio se utilizara la metodología ágil SCRUM, ya que se enfoca en la entrega de segmentos funcionales del producto y trabaja en base a un backlog establecido previamente, lo cual es conveniente para el desarrollo de este ambiente y su fase de pruebas. A lo anterior se adiciona la experiencia que se posee trabajando con este modelo, teniendo así conocimiento de cada componente de este.

Se efectuaran Sprints de 2 semanas, en los cuales se asignará una cantidad similar de historias, en función de su complejidad. Esta última se estimará tomando en cuenta el número y la dificultad que conlleve cada tarea, evaluando el desarrollo que se deba llevar a cabo en cada una de ellas.

3.2. Product Backlog

3.2.1. Historias de usuario y criterios de aceptación

A continuación, se definen las historias de usuario que serán abordadas durante el desarrollo de APE, las cuáles serán abordadas a completitud en cada Sprint y poseerán una etapa de pruebas como parte de su ciclo de desarrollo. Para confeccionar la estructura de las historias de usuario, se utilizará el rol de negocio como solicitante y se les asignará la sigla HU-XX, donde XX corresponde a la enumeración de cada una partiendo desde el 01. A lo anterior se suman los criterios de aceptación correspondientes por cada HU que serán listados en cada figura con la sigla COS-XX, donde XX cumple la misma función anterior.

HU-01	Barra de navegación
Como Negocio quiero una barra de navegación para escoger el tipo de producto y redirigir el sitio con el tipo seleccionado	
Criterios de aceptación	
COS-01: La barra de navegación permite intercambiar entre 3 tipos de producto	
COS-02: La barra de navegación resalta el tipo de producto actual	
COS-03: Al seleccionar un nuevo tipo de producto se actualiza la URL del sitio	

Figura 12: Historia de Usuario HU-01. Fuente: Elaboración propia

HU-02	Catálogo de productos
Como Negocio quiero visualizar los productos en un catálogo cuadrulado para incentivar la compra de estos por parte de los clientes	
Criterios de aceptación	
COS-01: El catálogo muestra los datos base de cada producto (marca, nombre y precio)	
COS-02: Al presionar sobre la cuadrícula de un producto se debe redireccionar a la ficha de este	
COS-03: Cada fila del catálogo esta compuesta por 3 productos	
COS-04: El diseño del catálogo no interfiere con los otros componentes del sitio	

Figura 13: Historia de Usuario HU-02. Fuente: Elaboración propia

HU-03	Navegación en catálogo de productos
Como Negocio quiero dividir el catalogo en grupos de 9 productos para reducir la saturación de información y mejorar la experiencia del cliente	
Criterios de aceptación	
COS-01: El catálogo se muestra en grupos de 9 productos	
COS-02: Dependiendo de la cantidad de productos, se despliega una barra numérica para navegar entre los grupos de estos	
COS-03: La barra numérica bloquea el número del grupo actual	

Figura 14: Historia de Usuario HU-03. Fuente: Elaboración propia

HU-04	Panel de filtros
Como Negocio quiero un panel de filtros para organizar los distintos filtros que tendrá el aplicativo	
Criterios de aceptación	
COS-01: El diseño del panel se ajusta con el resto de elementos	
COS-02: El panel se despliega al lado izquierdo del sitio	

Figura 15: Historia de Usuario HU-04. Fuente: Elaboración propia

HU-05	Filtro de precios
Como Negocio quiero un filtro de precios para filtrar los productos del catálogo por rangos de precios	
Criterios de aceptación	
COS-01: Los productos se filtran al añadir una cota inferior de precio	
COS-02: Los productos se filtran al añadir una cota superior de precio	
COS-03: Los productos se filtran al añadir ambas cotas de precio	
COS-04: Los productos mostrados cumplen con las cotas indicadas	
COS-05: Los filtros seleccionados deben listarse con la opción de quitarlos	

Figura 16: Historia de Usuario HU-05. Fuente: Elaboración propia

HU-06	Filtro de marcas
Como Negocio quiero un filtro de marca para filtrar los productos del catálogo por marca	
Criterios de aceptación	
COS-01: En el panel del filtro se indican las marcas disponibles en el catálogo	
COS-02: Los productos se filtran al seleccionar una marca	
COS-03: Los filtros seleccionados deben listarse con la opción de quitarlos	

Figura 17: Historia de Usuario HU-06. Fuente: Elaboración propia

HU-07	Filtro por especificaciones
Como Negocio quiero un filtro de especificaciones para filtrar los productos según sus especificaciones	
Criterios de aceptación	
COS-01: Se debe listar cada especificación de los productos, tales como RAM, memoria o cámara	
COS-02: Al seleccionar una especificación se deben filtrar los productos	
COS-03: Los filtros seleccionados deben listarse con la opción de quitarlos	

Figura 18: Historia de Usuario HU-07. Fuente: Elaboración propia

HU-08	Carro de compras
Como Negocio quiero un carro de compras para almacenar los productos que se deseen comprar	
Criterios de aceptación	
COS-01: Los productos añadidos se almacenan en el carro de compras	
COS-02: Se integra un ícono de carro de compras en todas las vistas del sitio	
COS-03: Al hacer click en el ícono se redirige al carro de compras	

Figura 19: Historia de Usuario HU-08. Fuente: Elaboración propia

HU-09	Visualizar carro de compras
Como Negocio quiero visualizar el carro de compras en una nueva página para que el cliente vea los productos que añadió	
Criterios de aceptación	
COS-01: Se listan todos los productos añadidos	
COS-02: Se muestra el precio y cantidad de cada producto	
COS-03: El diseño no afecta a los otros componentes de la vista del carro de compras	

Figura 20: Historia de Usuario HU-09. Fuente: Elaboración propia

HU-10	Ficha de producto
Como Negocio quiero que los productos tengan su ficha de especificaciones y detalles para que el cliente tenga acceso a su información	
Criterios de aceptación	
COS-01: Se despliegan las especificaciones técnicas del producto	
COS-02: Se incluye boton para añadir producto al carro de compras	
COS-03: Se puede seleccionar la cantidad del producto a añadir al carro	

Figura 21: Historia de Usuario HU-10. Fuente: Elaboración propia

HU-11	Añadir productos al carro de compras
Como Negocio quiero que un producto pueda ser añadido al carro de compras desde su ficha para su futura compra	
Criterios de aceptación	
COS-01: Se actualiza el carro de compras	
COS-02: Se actualizan los totales	
COS-03: Se actualiza el stock del producto	

Figura 22: Historia de Usuario HU-11. Fuente: Elaboración propia

HU-12	Remover productos del carro de compras
Como Negocio quiero que un producto pueda ser removido del carro de compras para permitir al cliente modificar su pedido	
Criterios de aceptación	
COS-01: Se incluye botón para eliminar productos en la vista del carro de compras	
COS-02: Se actualiza el carro de compras	
COS-03: Se actualizan los totales	
COS-04: Se actualiza el stock del producto	

Figura 23: Historia de Usuario HU-12. Fuente: Elaboración propia

HU-13	Totales en vista de carro de compras
Como Negocio quiero un panel de totales en la vista del carro de compras para entregar información sobre el pedido actual	
Criterios de aceptación	
COS-01: Se visualiza el precio total de la compra	
COS-02: El panel no genera conflictos con el resto de componentes de la vista del carro de compras	
COS-03: El panel se actualiza cada vez que se añada un nuevo producto al carro de compras	

Figura 24: Historia de Usuario HU-13. Fuente: Elaboración propia

HU-14	Formulario de compra
Como Negocio quiero un formulario de compra para registrar los datos del cliente	
Criterios de aceptación	
COS-01: Se incluyen los campos de nombre, apellido, RUT, email, dirección, nro de domicilio y nro de dpto.	
COS-02: Se incluye botón para finalizar orden	
COS-03: Al finalizar la orden se debe limpiar el carro de compras	
COS-04: Al finalizar la orden se despliega un modal de éxito	

Figura 25: Historia de Usuario HU-14. Fuente: Elaboración propia

HU-15	Registro de clientes
Como Negocio quiero un registro de los clientes para enviar ofertas personalizadas	
Criterios de aceptación	
COS-01: Existe una tabla específica para los clientes del sitio	
COS-02: Cada vez que se realice un pedido se registran los datos del cliente	
COS-03: Cuando un RUT se encuentre en base de datos se actualizará el registro con los datos ingresados en el último pedido	

Figura 26: Historia de Usuario HU-15. Fuente: Elaboración propia

HU-16	Registro de pedidos
Como Negocio quiero un registro de pedidos para gestionar el estado de estos	
Criterios de aceptación	
COS-01: Existe una tabla específica para los pedidos realizados	
COS-02: Los pedidos quedan registrados en base de datos	
COS-03: Los pedidos incluyen fecha y hora de ingreso	

Figura 27: Historia de Usuario HU-16. Fuente: Elaboración propia

3.2.2. Tareas a desarrollar

A partir de las historias de usuario mencionadas previamente, se definen las tareas asociadas a cada una de ellas. Para hacer una división óptima de lo que se debe desarrollar, se establecerán tareas para ambos ambientes de la aplicación web. Se utilizará la sigla T-XX para identificarlas, donde XX corresponde a un número desde el 01 en adelante.

1. HU-01: Barra de navegación

HU-01	Barra de navegación
FrontEnd	
T-01: Desarrollo de código HTML y estilos CSS	
T-02: Agregar URL a botones	

Tabla 1: Tareas historia de usuario HU-01. Fuente: Elaboración propia.

2. HU-02: Catálogo de productos

HU-02	Catálogo de productos
FrontEnd	
T-01: Desarrollo de filas del catálogo	
T-02: Desarrollo de cuadrículas para productos	
T-03: Lógica para conexión con backend	
BackEnd	
T-04: Creación de Endpoint para envío de productos	
T-05: Consulta a base de datos y envío de respuesta a frontend	

Tabla 2: Tareas historia de usuario HU-02. Fuente: Elaboración propia.

3. HU-03: Navegación en catálogo de productos

HU-03	Navegación en catálogo de productos
FrontEnd	
T-01: Aplicar límite de catálogo a grupos de 9 productos	
T-02: Desarrollo de barra numérica	
T-03: Lógica para barra numérica	

Tabla 3: Tareas historia de usuario HU-03. Fuente: Elaboración propia.

4. HU-04: Panel de filtros

HU-04	Panel de filtros
FrontEnd	
T-01: Desarrollo de panel de filtros	

Tabla 4: Tareas historia de usuario HU-04. Fuente: Elaboración propia.

5. HU-05: Filtro de precios

HU-05	Filtro de precios
FrontEnd	
T-01: Desarrollo de sección del filtro	
T-02: Implementación de inputs para cota inferior y superior	
T-03: Lógica de filtrado para ambas cotas	

Tabla 5: Tareas historia de usuario HU-05. Fuente: Elaboración propia.

6. HU-06: Filtro de marcas

HU-06	Filtro de marcas
FrontEnd	
T-01: Desarrollo de sección de filtro	
T-02: Despliegue de marcas disponibles	
T-03: Lógica de filtrado	

Tabla 6: Tareas historia de usuario HU-06. Fuente: Elaboración propia.

7. HU-07: Filtro por especificaciones

HU-07	Filtro por especificaciones
FrontEnd	
T-01: Desarrollo de sección de filtro	
T-02: Despliegue de especificaciones disponibles	
T-03: Implementación de inputs y barras deslizables para cada especificación	
T-04: Lógica de filtrado	

Tabla 7: Tareas historia de usuario HU-07. Fuente: Elaboración propia.

8. HU-08: Carro de compras

HU-08	Carro de compras
FrontEnd	
T-01: Creación de estructura en variables de sesión para almacenar carro	
T-02: Lógica para insertar producto en estructura del carro	
T-03: Implementación de icono de carro de compras	
T-04: Lógica para conexión con Backend	
BackEnd	
T-05: Creación de Endpoint	
T-06: Inserción en base de datos y envío de respuesta a frontend	

Tabla 8: Tareas historia de usuario HU-08. Fuente: Elaboración propia.

9. HU-09: Visualizar carro de compras

HU-09	Visualizar carro de compras
FrontEnd	
T-01: Desarrollo de vista carro de compras	
T-02: Lógica para desplegar productos en el carro	
T-03: Lógica para desplegar datos de cada producto	

Tabla 9: Tareas historia de usuario HU-09. Fuente: Elaboración propia.

10. HU-10: Ficha de producto

HU-10	Ficha de producto
FrontEnd	
T-01: Desarrollo de vista ficha de producto	
T-02: Lógica para desplegar datos de producto	
T-03: Desarrollo de botón para añadir producto al carro	
T-04: Lógica para conexión con Backend	
BackEnd	
T-05: Creación de Endpoint	
T-06: Consulta a base de datos y envío de respuesta a frontend	

Tabla 10: Tareas historia de usuario HU-10. Fuente: Elaboración propia.

11. HU-11: Añadir productos al carro de compras

HU-11	Añadir productos al carro de compras
FrontEnd	
T-01: Insertar lógica para añadir producto al carro en botón de ficha	
T-02: Lógica para actualizar totales de cada producto	
T-03: Lógica de conexión con Backend para actualizar stock	
BackEnd	
T-04: Creación de Endpoint	
T-05: Inserción en base de datos y envío de respuesta a frontend	

Tabla 11: Tareas historia de usuario HU-11. Fuente: Elaboración propia.

12. HU-12: Remover productos del carro de compras

HU-12	Remover productos del carro de compras
FrontEnd	
T-01: Desarrollo de botón en vista de carro de compras	
T-02: Lógica para actualizar totales de cada producto	
T-03: Lógica de conexión con Backend para actualizar stock	
BackEnd	
T-04: Creación de Endpoint	
T-05: Inserción en base de datos y envío de respuesta a frontend	

Tabla 12: Tareas historia de usuario HU-12. Fuente: Elaboración propia.

13. HU-13: Totales en vista de carro de compras

HU-13	Totales en vista de carro de compras
FrontEnd	
T-01: Insertar lógica para añadir producto al carro en botón de ficha	
T-02: Lógica para actualizar totales de cada producto	
T-03: Lógica de conexión con Backend para actualizar stock	
BackEnd	
T-04: Creación de Endpoint	
T-05: Inserción en base de datos y envío de respuesta a frontend	

Tabla 13: Tareas historia de usuario HU-13. Fuente: Elaboración propia.

14. HU-14: Formulario de compra

HU-14	Formulario de compra
FrontEnd	
T-01: Desarrollo de vista de formulario	
T-02: Lógica para botón de finalizar orden	
T-03: Lógica para limpiar carro de compras	
BackEnd	
T-04: Creación de Endpoint	
T-05: Actualización en base de datos y envío de respuesta a frontend	

Tabla 14: Tareas historia de usuario HU-14. Fuente: Elaboración propia.

15. HU-15: Registro de clientes

HU-15	Registro de clientes
BackEnd	
T-01: Implementar lógica para creación de cliente	
T-02: Implementar lógica para actualizar datos de cliente	

Tabla 15: Tareas historia de usuario HU-15. Fuente: Elaboración propia.

16. HU-16: Registro de pedidos

HU-16	Registro de pedidos
BackEnd	
T-01: Creación de modelo de pedidos y lógica de inserción	

Tabla 16: Tareas historia de usuario HU-16. Fuente: Elaboración propia.

3.3. Plan de pruebas

En la presente sección, se establecerá el plan de pruebas a realizar sobre las historias de usuario descritas anteriormente. Para cada caso en particular se realizarán pruebas funcionales y no funcionales, más específicamente pruebas unitarias, de integración y pruebas de rendimiento, estas últimas para evaluar la estabilidad del sistema frente a distintos tipos de interacciones. Para el caso de las pruebas funcionales, estas se realizarán utilizando módulos de *NodeJS* de las herramientas de pruebas (Selenium y Cucumber) y serán aplicadas de manera programática, por el lado de las pruebas de

rendimiento, éstas serán llevadas a cabo utilizando JMeter utilizando la interfaz que dispone la herramienta.

3.3.1. Componentes y funcionalidades a evaluar

Debido a que se busca obtener una eficacia óptima de las herramientas de prueba, para efectos de esta investigación se decidió aplicar pruebas sobre todas las historias de usuario que tengan o utilicen funcionalidades del ambiente BackEnd, como por ejemplo el traer los productos desde el catálogo. Los métodos a evaluar mediante pruebas unitarias son:

- `getCart(idOrden)`: Método que obtiene el carro de compras mediante la id de una orden.
- `getCatalog(tipo, ram, core, camera, storage, has4g, has5g, priceBottom, priceTop)`: Método encargado de retornar el catálogo de productos para cada tipo de producto, permite filtrar por medio de sus características y/o precio.
- `getComponents(tipo)`: Método que retorna todos los tipos de componentes relacionados a un tipo de producto.
- `getCustomer(rut)`: Función que retorna los datos de un cliente utilizando su rut como parámetro de búsqueda.
- `createCustomer(data)`: Método encargado de crear un cliente en base de datos, en caso de existir un registro con el mismo rut, se retorna la id de este, mismo comportamiento para el caso contrario tras su creación.
- `createOrder(store_id)`: Crea una orden de compra asociada a la id de la tienda seleccionada, el estado queda ingresado como pendiente.
- `updateOrder(order_id, customer_id)`: Método que actualiza la orden, asociando un cliente a esta y cambiando su estado.
- `getProduct(id)`: Método utilizado en la ficha de un producto, entrega todos los datos relacionados a este (marca, precio, componentes, descripción, etc
- `checkProductStock(id, id_store)`: Función encargada de retornar el stock actual en bodega del producto indicado.
- `updateProductStock(id, id_store, qty, operation)`: Método responsabilizado de actualizar el stock de los productos en bodega, permite agregar o desagregar cantidad según corresponda.
- `getStore()`: Método encargado de retornar la información de las tiendas disponibles.

Para el caso de las pruebas de integración se evaluarán los siguiente métodos:

- `addToCart(idOrden, idProduct, id_store)`: Método que agrega un producto al carro de compras, consultando y actualizando su stock en bodega. Conlleva al uso de los métodos `checkProductStock` y `updateProductStock`.
- `removeFromCart(idOrden, idProduct, id_store, qty)`: Implementación cuyo fin es el de eliminar un producto del carro de compras, en caso de que la cantidad llegue a 0 elimina el registro desde la base de datos. Se compone de los métodos `checkProductStock` y `updateProductStock`.

Las pruebas de sistema serán implementadas en las siguientes interacciones con el ambiente FrontEnd:

- Acceso al catálogo de productos
- Filtrado de productos
- Acceso a la ficha de un producto
- Agregar producto al carro
- Acceder a vista de carro de compras
- Eliminar producto desde el carro de compras
- Envío de formulario de datos personales

3.3.2. Pruebas unitarias

Las pruebas unitarias, como se mencionó anteriormente, serán implementadas con Cucumber. Para ello, se realizará el desarrollo de un código en lenguaje Python para poblar la base de datos y esto se utilizará como datos de entrada en cada método mencionado. Se desarrollaron un total de 11 planes de prueba para los métodos y, según cada uno de ellos, se aplicaron distintos escenarios descritos a continuación.

- Método getCart: Para esta prueba se utilizarán como datos de entrada ID's de ordenes existentes en base de datos, llevando a cabo su validación a partir de los datos que retorne el método, verificando que estos datos de salida correspondan a los que se establecen en el modelo.

```
Feature: Obtener carro de compras

Scenario Outline: Pruebas con distintas ordenes
  Given Dada la <id> de una orden
  When Se obtienen los productos del carro asociados a esa orden
  Then La cantidad es de <cantidad> y uno de los productos posee la id <producto>

Examples:


| id | cantidad | producto |
|----|----------|----------|
| 1  | 11       | 1        |
| 2  | 5        | 3        |
| 3  | 6        | 52       |
| 4  | 4        | 23       |
| 5  | 4        | 76       |
| 6  | 3        | 67       |
| 11 | 4        | 6        |
| 14 | 1        | 2        |


```

Figura 28: Estructura pruebas método getCart. Fuente: Elaboración propia

- Método getCatalog: Debido a que el sitio de pruebas solo posee 3 tipos de producto, se utilizaron estos para realizar las pruebas, verificando que la respuesta entregada contenga la cantidad de productos correspondiente.

```
Feature: Obtener catalogo de productos

Scenario Outline: Pruebas getCatalog
  Given Al ingresar al catalog con el <tipo> de producto
  When Se obtienen los productos asociados
  Then La cantidad de productos es <cantidad>

Examples:


| tipo         | cantidad |
|--------------|----------|
| 'mobile'     | 610      |
| 'laptop'     | 4        |
| 'accesorios' | 0        |


```

Figura 29: Estructura pruebas método getCatalog. Fuente: Elaboración propia

- Método getComponents: Para este caso, al igual que en el anterior, se utilizó como input de prueba los tipos de producto (mobile, laptop y accesorios), validando que la respuesta posea la cantidad indicada de datos en cada tipo de componente, para el caso de RAM, núcleos y almacenamiento.

Feature: Obtener conjunto de componentes

Scenario Outline: Pruebas de getComponents

Given Dado el tipo de producto <tipo>

When Se obtienen el conjunto de componentes

Then Existen por lo menos <ram> variantes de ram, <cores> de nucleos y <storage> de almacenamiento

Examples:

tipo	ram	cores	storage
'mobile'	6	5	3
'laptop'	2	2	3
'accesorios'	0	0	0

Figura 30: Estructura pruebas método getComponents. Fuente: Elaboración propia

- Método getCustomer: Dado el input RUT en cada instancia, esta prueba apunta a obtener los datos del cliente asociado a este dato y validar que el nombre, teléfono y dirección retornados correspondan a los indicados en el modelo definido.

Feature: Obtener datos de cliente

Scenario Outline: Pruebas de getCustomer

Given Dado el <rut> de un cliente

When Se obtienen, si existen, los datos de este

Then Se comprueba el nombre del cliente <nombre>, su telefono <telefono> y direccion <direccion>

Examples:

rut	nombre	telefono	direccion
'19423509-7'	'Diego'	'933040415'	'La cordillera'
'10223251-8'	'Roberto'	'999999999'	'Pasaje sofia'
'11610572-1'	'Romy'	'999999998'	'Pasaje sofia'
'19247020-k'	'Yesnia'	'999999997'	'Obrien'
'12404031-0'	'Pedro'	'999999996'	'Prueba01'
'20026802-4'	'Juan'	'999999995'	'Prueba02'
'16904670-0'	'Stuart'	'999999994'	'Prueba03'
'15329256-6'	'Alonso'	'999999993'	'Prueba04'

Figura 31: Estructura pruebas método getCustomer. Fuente: Elaboración propia

- Método createCustomer: A partir de la información de un cliente, se ejecuta el método para almacenar sus datos en el sistema. Finalmente se valida que el registro exista por medio del RUT indicado.

Feature: Crea cliente en el sistema

Scenario Outline: Pruebas de createCustomer

Given Dados los datos <nombre>, <apellido>, <rut>, <telefono>, <correo> y <direccion>

When Se crea el registro en base de datos

Then Se comprueba que el cliente exista por medio de su <rut>

Examples:

rut	nombre	apellido	telefono	correo	direccion
'11111111-1'	'Test'	'1'	'999999991'	'Tes1'	'Test'
'11111112-1'	'Test'	'2'	'999999992'	'Tes2'	'Test'
'11111113-1'	'Test'	'3'	'999999993'	'Tes3'	'Test'
'11111114-1'	'Test'	'4'	'999999994'	'Tes4'	'Test'

Figura 32: Estructura pruebas método createCustomer. Fuente: Elaboración propia

- Método createOrder: Utilizando la id de una tienda como información de entrada, se valida el funcionamiento de la creación de ordenes, para esto se ejecuta el método y luego se valida comprobando si quedo registro en base de datos.

Feature: Crea orden en el sistema

Scenario Outline: Pruebas de createOrder

Given Dada la <id> de una tienda

When Se crea la orden en base de datos

Then Se comprueba que la orden existe en base de datos

Examples:

id
1
2
3
4
5
6
7
7
4
2

Figura 33: Estructura pruebas método createOrder. Fuente: Elaboración propia

- Método `updateOrder`: Para evaluar este método se utilizará un cliente previamente registrado en base de datos y la id de la orden a modificar, para comprobar que se registró un cambio se utilizará el id de la orden comprobar el estado.

```
Feature: Actualizar estado de orden

Scenario Outline: Pruebas de updateOrder
  Given Dada la <id1> de una orden y la <id2> de un cliente
  When Se actualiza el estado de la orden
  Then Se comprueba el nuevo estado de la orden

Examples:


| id1 | id2 |
|-----|-----|
| 17  | 1   |
| 18  | 2   |
| 19  | 3   |
| 20  | 6   |
| 21  | 7   |
| 22  | 8   |
| 23  | 9   |
| 24  | 10  |


```

Figura 34: Estructura pruebas método `updateOrder`. Fuente: Elaboración propia

- Método `getProduct`: Por medio de la id de un producto, se ejecutará el método para traer la información desde la base de datos, para luego validar que la columna indicada posea el dato requerido en el plan de pruebas.

```
Feature: Obtener datos de producto

Scenario Outline: Pruebas de getProduct
  Given La id de un producto <var>
  When Obtengo la columna <column>
  Then Que tiene como valor <result>

Examples:


| var | column  | result         |
|-----|---------|----------------|
| 1   | 'model' | 'Samsung'      |
| 2   | 'name'  | 'S21'          |
| 3   | 'price' | '1299000'      |
| 4   | 'model' | 'Xiaomi'       |
| 5   | 'model' | 'Xiaomi'       |
| 6   | 'name'  | 'Redmi Note 9' |
| 7   | 'model' | 'Xiaomi'       |
| 8   | 'model' | 'Xiaomi'       |


```

Figura 35: Estructura pruebas método `getProduct`. Fuente: Elaboración propia

- Método `checkProductStock`: Para ejecutar el plan de pruebas sobre este método se utilizarán como inputs, en cada iteración, las ids de un producto y una tienda, una vez ejecutado se validará la respuesta utilizando la cantidad esperada de stock.

```
Feature: Revisa stock de un producto

Scenario Outline: Pruebas de checkProductStock
  Given Dada la <id1> de un producto y la <id2> de una tienda
  When Se consulta el stock disponible
  Then Se valida que el stock sea igual a <qty>

Examples:
  | id1 | id2 | qty |
  | 301 | 1   | 61  |
  | 302 | 2   | 82  |
  | 303 | 5   | 73  |
  | 304 | 1   | 112 |
  | 305 | 5   | 128 |
  | 306 | 3   | 10  |
  | 307 | 7   | 29  |
  | 308 | 6   | 140 |
  | 309 | 3   | 155 |
  | 300 | 2   | 118 |
```

Figura 36: Estructura pruebas método checkProductStock. Fuente: Elaboración propia

- Método updateProductStock: Dada la id de un producto y la id de una tienda, en esta prueba se utilizará el método para aumentar y disminuir el stock de un producto. Para validar que la operación se realizó correctamente, se registra el stock del producto previo a la modificación, para luego validar si aumento o disminuyó según corresponda.

```
Feature: Actualizar stock de un producto

Scenario Outline: Pruebas de updateProductStock
  Given Dada la <id1> del producto y la <id2> de una tienda, se consulta el stock actual del producto
  When Se realiza la operacion <operation>
  Then Se comprueba el nuevo stock del producto segun la operacion <operation>

Examples:
  | id1 | id2 | operation |
  | 17  | 1   | 'remove'  |
  | 18  | 5   | 'remove'  |
  | 19  | 3   | 'remove'  |
  | 20  | 6   | 'add'     |
  | 21  | 7   | 'add'     |
  | 22  | 6   | 'add'     |
  | 23  | 7   | 'remove'  |
  | 24  | 2   | 'add'     |
```

Figura 37: Estructura pruebas método updateProductStock. Fuente: Elaboración propia

- Método `getStore`: Dada un ingreso simulado al sitio de pruebas, se valida que la respuesta del método incluya la id indicada en cada instancia.

```
Feature: Obtiene tiendas disponibles

Scenario Outline: Pruebas de getStore
  Given Dado un ingreso al sitio
  When Se obtienen las tiendas disponibles
  Then Se comprueba que la tienda <id> se encuentre en la respuesta

Examples:
| id |
| 1  |
| 2  |
| 3  |
| 4  |
| 5  |
| 6  |
| 7  |
```

Figura 38: Estructura pruebas método `getStore`. Fuente: Elaboración propia

3.3.3. Pruebas de integración

Haciendo uso de la herramienta en el ambiente Backend de la aplicación, se desarrollaron los siguientes conjuntos de pruebas de integración:

- Método `addToCart`: A partir de la id de un producto, la id de una orden y la id de una tienda, se ejecuta el método para ingresar el producto al carro de compras relacionado con la orden, actualizando el stock del producto según la tienda indicada. Para validar el funcionamiento, se comprueba que el producto actualice su stock y que se encuentre en el carro de compras. Se evaluarán 4 tipos de escenarios:

```
Feature: Prueba de agregar producto al carro

Scenario Outline: Pruebas con 1 producto
  Given Dado el producto <idProducto>, la orden numero <idOrden> y la tienda <idTienda>
  Then Se obtiene el stock actual del producto
  When Se agrega producto al carro
  Then Se comprueba el nuevo stock del producto tras agregarlo al carro
  Then Se valida que exista el producto en el carro

Examples:
| idProducto | idOrden | idTienda |
| 910        | 1       | 3        |
| 921        | 2       | 2        |
| 932        | 11      | 3        |
| 943        | 15      | 5        |
| 954        | 3       | 2        |
| 965        | 16      | 4        |
| 976        | 5       | 7        |
| 987        | 6       | 2        |
```

Figura 39: Estructura pruebas método `addToCart` escenarios con 1 producto.
Fuente: Elaboración propia

Scenario Outline: Pruebas con 2 productos
Given Dado el producto <idProducto> y el producto <idProducto2>, la orden numero <idOrden> y la tienda <idTienda>
Then Se obtiene el stock actual de los productos
When Se agregan productos al carro
Then Se comprueba el nuevo stock de los productos
Then Se valida que existan los productos en el carro

Examples:

idProducto	idProducto2	idOrden	idTienda
910	911	1	3
921	922	2	2
932	933	11	3
943	944	15	5
954	955	3	2
965	966	16	4
976	977	5	7
987	988	6	2

Figura 40: Estructura pruebas método addToCart escenarios con 2 productos.
Fuente: Elaboración propia

Scenario Outline: Pruebas con producto inexistente
Given Dado el producto <idProducto>, la orden numero <idOrden> y la tienda <idTienda>
Then Se obtiene el stock actual del producto
When Se agrega producto al carro
Then Se comprueba el nuevo stock del producto tras agregarlo al carro
Then Se valida que exista el producto en el carro

Examples:

idProducto	idOrden	idTienda
9910	1	3
9921	2	2
9932	11	3
9943	15	5
9954	3	2
9965	16	4
9976	5	7
9987	6	2

Figura 41: Estructura pruebas método addToCart escenarios con producto inexistente.
Fuente: Elaboración propia

Scenario Outline: Pruebas con orden inexistente
Given Dado el producto <idProducto>, la orden numero <idOrden> y la tienda <idTienda>
Then Se obtiene el stock actual del producto
When Se agrega producto al carro
Then Se comprueba el nuevo stock del producto tras agregarlo al carro
Then Se valida que exista el producto en el carro

Examples:

idProducto	idOrden	idTienda
910	9991	3
921	9992	2
932	9993	3
943	9994	5
954	9995	2
965	9996	4
976	9997	7
987	9998	2

Figura 42: Estructura pruebas método addToCart escenarios con orden inexistente.
Fuente: Elaboración propia

- Método removeFromCart: Realizando el proceso inverso del método anterior, para este conjunto de pruebas se remueve el producto indicado desde el carro de compras, tras esto se valida que no exista registro del producto y que el stock se haya actualizado acorde a esto. Se evaluarán 2 tipos de escenario:

Feature: Remover producto del carro

Scenario Outline: Pruebas de removeFromCart

Given Dada la <idProducto> de un product, la <idOrden> de una orden y la <idTienda> de una tienda

Then Se obtiene el stock actual del producto

When Se remueve el producto del carro de compras

Then Se comprueba el nuevo stock del producto

Then Se comprueba que el producto ya no este en el carro de compras

Examples:

idProducto	idOrden	idTienda
900	1	3
901	2	2
902	11	3
903	15	5
904	3	2
905	16	4
906	5	7
907	6	2

Figura 43: Estructura pruebas método removeFromCart. Fuente: Elaboración propia

Scenario Outline: Pruebas con orden inexistente

Given Dada la <idProducto> de un product, la <idOrden> de una orden y la <idTienda> de una tienda

Then Se obtiene el stock actual del producto

When Se remueve el producto del carro de compras

Then Se comprueba el nuevo stock del producto

Then Se comprueba que el producto ya no este en el carro de compras

Examples:

idProducto	idOrden	idTienda
900	9991	3
901	9992	2
902	9993	3
903	9994	5
904	9995	2
905	9996	4
906	9997	7
907	9998	2

Figura 44: Estructura pruebas método removeFromCart escenarios con orden inexistente. Fuente: Elaboración propia

3.3.4. Pruebas de sistema

Utilizando Selenium se realizarán pruebas automatizadas para evaluar las interacciones con el ambiente mencionadas previamente, estas pruebas automatizadas serán generadas a partir de las siguientes acciones específicas, con el fin abordar completamente estas interacciones del usuario:

- Acceso a catálogo de productos: Esta prueba automatizada iniciará accediendo al catálogo de productos del sitio de pruebas, una vez finalizada la carga de los productos, se procederá a navegar utilizando la barra inferior, esto de manera aleatoria y registrando los tiempos que tarda el sistema en realizar cada prueba, desde que se accede al sitio hasta cuando termina la carga de la nueva pestaña de productos.
- Acceso a ficha de producto: Para este caso, la prueba comenzará cargando la vista de catálogo del sitio de pruebas, dentro de esta se realizarán movimientos aleatorios utilizando la barra de navegación inferior, una vez cargado el sitio se accederá a un producto aleatorio de la opción actual de la barra de navegación.

Finalmente, se evaluará que la ficha cargada corresponda al producto seleccionado en el catálogo y el tiempo de procesamiento que tardó el sistema.

- Cambio de tipo de producto: Al igual que en las pruebas anteriores, se comenzará desde la vista de catálogo del sistema, dentro de esta se realizarán clicks aleatorios entre cada tipo de producto, se evaluará el tiempo de carga y los posibles errores que puedan surgir.
- Filtrado de productos: Utilizando el catálogo de productos como punto de inicio, se realizarán pruebas de filtrado aleatorias. Durante esta prueba se dejará registro de los filtros utilizados y de los resultados obtenidos, para validar su correcto funcionamiento se establecerán los resultados esperados para cada filtro.
- Acceso a vista de carro de compras: Esta prueba será llevada a cabo insertando 10 productos aleatorios al carro de compras, una vez realizado lo anterior se procederá a tomar el tiempo de computo que tarda el sistema en acceder desde el catálogo de productos al carro de compras.
- Agregar producto al carro de compras: Utilizando valores aleatorios, accediendo desde el catálogo de productos, se seleccionará una opción de la barra de navegación y un producto perteneciente a esta, una vez en la ficha del producto se añadirá este al carro de compras, almacenando el nombre de este como valor para la validación. Finalmente, se ingresará al carro de compras y se validará la existencia de este producto en la vista.
- Remover producto del carro de compras: Se imitará el comportamiento de la prueba anterior para preparar el carro de compras para su testeo, una vez terminado el proceso de añadir productos se procederá a tomar el tiempo de ejecución para las pruebas de esta funcionalidad, para validar el funcionamiento del sistema se confirmará si no queda registro de un producto después de haber sido removido del carro de compras por medio de la opción presente en este último.
- Flujo completo de compras: Por medio de los procedimientos mencionados previamente, para este caso se iniciará la prueba desde el catálogo de productos, se seleccionará de manera aleatoria una cantidad fija de productos para agregar al carro de compras. Una vez finalizado lo anterior, se accederá a la vista del carro de compras, en este punto se validaran los productos que fueron añadidos. Finalmente, se accederá al proceso de checkout, donde el usuario debe ingresar sus datos y enviar su orden, los datos serán insertados de manera aleatoria a partir de un conjunto de inputs de prueba, para validar esto último, se esperará a que se despliegue la alerta correspondiente al éxito del procedimiento.

La cantidad de iteraciones y otros inputs de cada prueba serán definidos en el siguiente capítulo que corresponde al de validación de la solución. En la sección de anexos se incluye el código desarrollado para llevar a cabo las pruebas anteriormente mencionadas.

3.3.5. Pruebas de rendimiento

Las pruebas de rendimiento serán realizadas sobre el ambiente de backend debido a que Jmeter no es una herramienta que posea la capacidad de cargar un sitio web visualmente, pero lo anterior se abarcará en las pruebas automatizadas con Selenium. Para efectuar el testeó, se realizará una evaluación de todos los puntos de acceso disponibles en la API (Backend), los cuáles se listan a continuación:

- GET /catalog: Endpoint que ejecuta el método *getCatalog*, enviando como respuesta los productos para mostrar en el catálogo.
- GET /product/id: Endpoint que utiliza el método *getProduct*, envía la información del producto con la id indicada.
- GET /catalog/components: Endpoint que utiliza el método *getComponents*, retorna los datos para armar el panel de filtros.
- GET /cart: Endpoint que utiliza el método *getCart*, responde con la información del carro asociado a la id de orden enviada como parámetro.
- GET /store: Endpoint que ejecuta el método *getStore*, envía como respuesta las tiendas existentes en base de datos.
- POST /cart: Endpoint que usa el método *addToCart*, deja un registro en base de datos del producto añadido al carro y responde con un mensaje de estado, según el resultado de la petición.
- POST /order: Endpoint que emplea el método *createOrder*, utilizando la id de tienda enviada como parámetro, se crea una orden nueva en base de datos y se retorna su id como respuesta.
- POST /cart/remove: Endpoint que utiliza el método *removeFromCart*, elimina el registro asociado al producto indicado, actualizando también su stock, retorna un mensaje de resultado de la petición.
- PUT /order: Endpoint que ejecuta el método *updateOrder*, en caso de no existir, se crea un registro para el cliente y luego se asocia a la orden, cambiando el estado de esta y retornando un mensaje de éxito o error, según corresponda.

El plan de pruebas evaluará:

- Tiempos de respuesta por endpoint
- Estabilidad del sistema frente a múltiples consultas
- Estabilidad del sistema frente a consultas simultáneas a distintos endpoints

- Rendimiento del sistema frente a múltiples consultas
- Rendimiento del sistema frente a consultas simultáneas a distintos endpoints

Lo anterior será medido utilizando las funcionalidades con la que dispone Jmeter para estos fines.

3.4. Evaluación de herramientas

Para efectos de esta investigación, las herramientas de testeo se evaluarán a partir de los resultados obtenidos en cada instancia. A continuación, se mencionan los aspectos a evaluar en cada herramienta:

- Resultados: Una vez finalizadas las pruebas, a partir de las observaciones que se puedan obtener de los resultados generales en cada herramienta, se espera evaluar la utilidad que tuvieron estos. Se evaluarán los siguientes aspectos de estos resultados:
 1. Estado de las pruebas: Se refiere al como la herramienta informa sobre los resultados durante o después de la ejecución de las pruebas.
 2. Manejo de errores: Este punto corresponde a como la herramienta actúa al detectar un error y como este puede afectar al resultado.
 3. Visualización de resultados: Apunta a como se muestran los resultados, más allá de números, que visualizaciones nos entrega para entender que ocurrió en las pruebas.
 4. Formato de resultados: Se evaluará también si la herramienta entrega un output que permita un procesamiento externo, es decir, si entrega los resultados de manera que puedan ser tomados por otra herramienta o software.
- Versatilidad: Un factor relevante en el uso de herramientas de testeo, son los distintos usos que se les puede dar, las diferentes formas de abarcar escenarios. Se tomarán en cuenta los siguientes escenarios:
 1. Pruebas Backend (Resultados)
 2. Pruebas Frontend (Resultados)
 3. Pruebas Backend (Rendimiento)
 4. Pruebas Frontend (Rendimiento)

A partir del análisis de los resultados se definirá en que escenarios la herramienta puede desenvolverse.

- Escalabilidad: Este punto será evaluado a partir de lo experimentado durante la implementación de las pruebas, dado que se espera aplicar mejoras continuas a cada test, añadiendo más validaciones o funcionalidades en cada proceso. Para este estudio, se evaluará la posibilidad de añadir nuevas validaciones y escenarios en cada prueba.
- Documentación: Dentro del uso de una nueva herramienta, en cualquier área, es relevante la información y manuales que pueda poseer para utilizarla de manera correcta. A continuación, se entregan los aspectos a evaluar:
 1. Fuentes de información
 2. Calidad de la información
 3. Registro de actualizaciones
 4. Ejemplos de uso

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

A lo largo de la presente sección se procederá a plasmar y validar los resultados obtenidos a partir de la solución propuesta anteriormente. A modo de resumen, en primera instancia se describirá el ambiente utilizado para las pruebas descritas, luego se expondrán y explicarán los resultados obtenidos dando énfasis a los comportamientos observados. Finalmente se procederá a evaluar las herramientas a partir de los puntos mencionados en la sección anterior.

4.1. Ambiente de pruebas

En este punto se detallarán los componentes del ambiente de pruebas utilizado para este estudio, esto debido a que los resultados obtenidos se ven directamente afectados por las capacidades de la computadora.

- Procesador: Intel Core i7-9750H 2.60GHz 6 núcleos 12 hilos
- Almacenamiento: Intel Optane 512GB
- RAM: 24,0 GB 2666Mhz
- Sistema operativo: Windows 10

Es importante mencionar que el ancho de banda también afecta en los tiempos de carga del sitio, pero para efectos de este estudio se comprobó que se cumple con lo necesario para que esto no sea un factor de variación en los resultados de las pruebas.

4.2. Pruebas unitarias

En la sección anterior se presentó y explico el modelo de pruebas para cada método, junto con la descripción de cada uno de ellos. El código desarrollado para trabajar con la herramienta se complementó con el modulo *assert* de *Node* para realizar las validaciones, por otro lado, cada prueba presenta una lógica distinta de validación, asociada a los pasos indicados en el modelo, los cuales están tras una de las siguientes instrucciones *GIVEN*, *WHEN* y *THEN*. Lo anteriormente mencionado aplica también para las pruebas de integración.

Los resultados de las pruebas unitarias serán presentados en gráficos en conjunto, evaluándose 2 tipos de información, la primera de ellas corresponde a los resultados por

escenario, que equivalen al flujo completo de información desde que se inserta la data de prueba hasta que se entrega un output en la instrucción *THEN*. Un escenario puede fallar si se detecta algún error en alguno de sus pasos o si la información de salida es distinta a lo establecido en el modelo. Por otro lado, se visualizaran los resultados por paso, para esto se tendrán en cuenta los posibles estados que se ven reflejados en el reporte entregado por Cucumber, los cuales son:

- Passed: El paso acabó su ejecución sin errores ni validaciones fallidas.
- Failed: Ocurrió un error o una validación incorrecta durante o tras finalizar su ejecución.
- Skipped: Tras el fallo de un paso en la ejecución de un escenario, se omiten las instrucciones que continúen en dicho escenario.

Para obtener mayor variación en los resultados y validar el correcto funcionamiento del código para casos que puedan generar conflictos, se desarrollaron escenarios con validaciones contrarias al original, es decir, se espera que un resultado sea incorrecto. Junto con lo anterior, también se implementaron escenarios que presentaran errores durante la ejecución de sus pasos, por lo que se pueden dar los casos de omisión de instrucciones.

4.2.1. Escenarios

Se ejecutó el script de Cucumber sobre todos los modelos de prueba descritos, los cuales se encuentran en un directorio distinto al de las pruebas de integración. A partir del reporte generado por la herramienta, se evaluó un total de 142 escenarios, de ellos 98 finalizaron correctamente y 44 fallaron durante su ejecución. Los resultados anteriores se encuentran en el siguiente gráfico:



Figura 45: Pruebas unitarias: Resultados generales escenarios. Fuente: Elaboración propia

Haciendo uso de lo mencionado en el punto anterior, con respecto a los casos con errores, la ejecución del modelo de pruebas para cada método finalizó con los siguientes resultados:

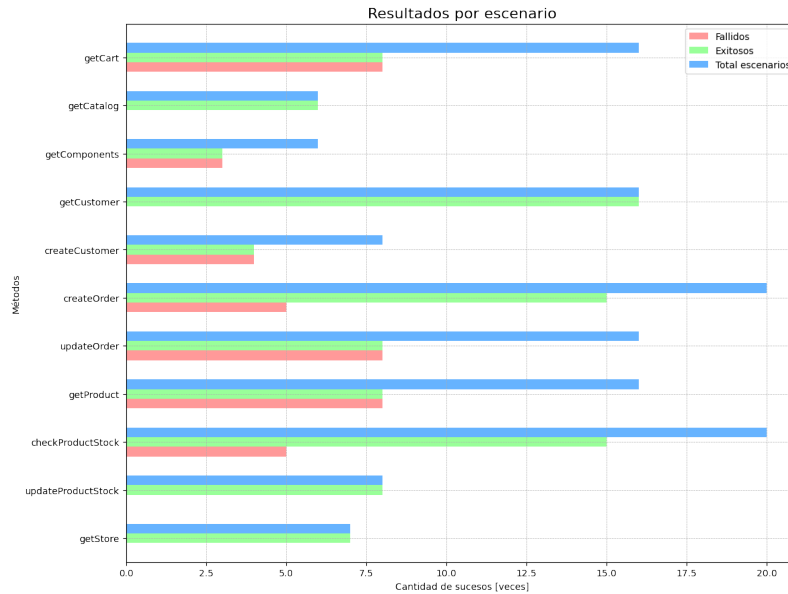


Figura 46: Pruebas unitarias: Resultados por escenario. Fuente: Elaboración propia

4.2.2. Pasos

Por el lado de los pasos realizados en cada escenario, se ejecutaron un total de 426 instrucciones, de las cuales 360 finalizaron su ejecución correctamente, 44 fallaron y 22 fueron omitidas tras un fallo, el tiempo de ejecución fue de 12.77[s]. Los resultados anteriores pueden ser observados en el siguiente gráfico:

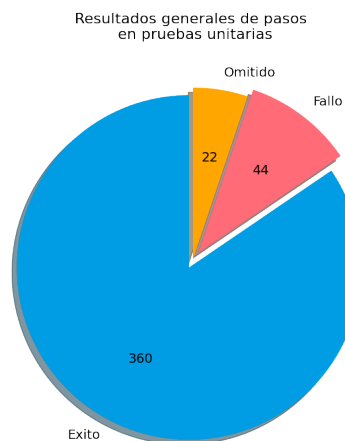


Figura 47: Pruebas unitarias: Resultados generales pasos. Fuente: Elaboración propia

A continuación, se entregan los resultados obtenidos por cada de método:

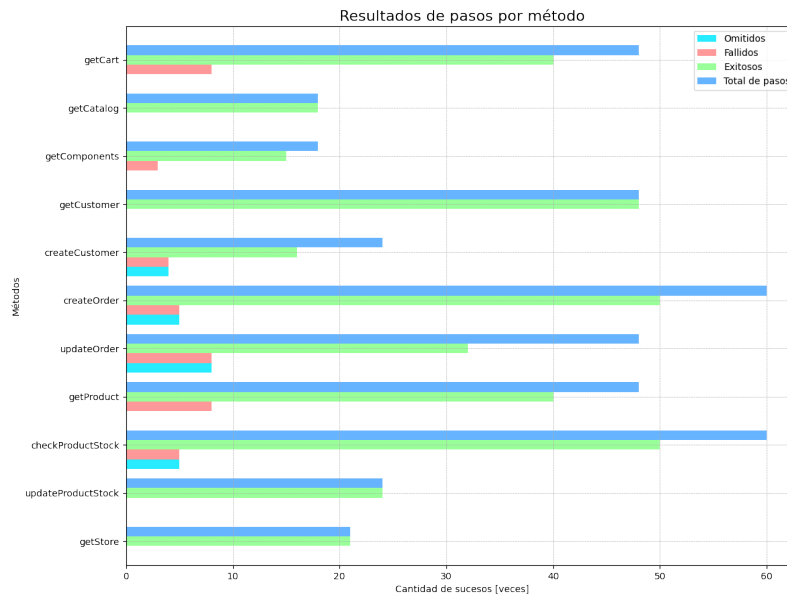


Figura 48: Pruebas unitarias: Resultados pasos por método. Fuente: Elaboración propia

Tras realizar un análisis por método de los resultados obtenidos en las pruebas unitarias, fue posible observar que todas las pruebas cumplieron acorde a lo especificado en sus modelos, los escenarios que estaban descritos para funcionar correctamente y los que se definieron para generar conflictos tuvieron el comportamiento esperado. Por otro lado, la herramienta entregó todos los errores que surgieron y en que paso se generaron.

4.3. Pruebas de integración

Las pruebas de integración presentan una mayor cantidad de escenarios y pasos para cada método, esto implica un mayor número de validaciones dado que se debe confirmar el correcto funcionamiento de todas las funciones ejecutadas durante cada escenario. Por lo anterior, se decidió realizar un análisis por método en las secciones subsiguientes.

4.3.1. Escenarios: Método AddToCart

Se ejecutaron un total de 32 escenarios distintos, de los cuales 16 completaron su ejecución con éxito y 16 presentaron un error durante su ejecución.

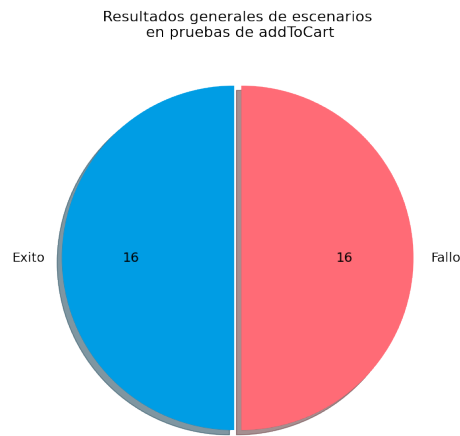


Figura 49: Pruebas de integración: Resultados escenarios addToCart. Fuente: Elaboración propia

Los resultados presentes en el gráfico reflejan lo definido en el modelo de pruebas, de los escenarios establecidos la mitad de ellos presentaba un valor no existente en base de datos, por lo que al ejecutar dichos escenarios se retorna un mensaje de error desde el método y esto es recibido en la herramienta, siendo reflejado como un escenario fallido.

4.3.2. Pasos: Método AddToCart

Se ejecutaron un total de 160 pasos, de los cuales 104 fueron exitosos, 16 fallaron y, como consecuencia de esto último, se omitieron 40.

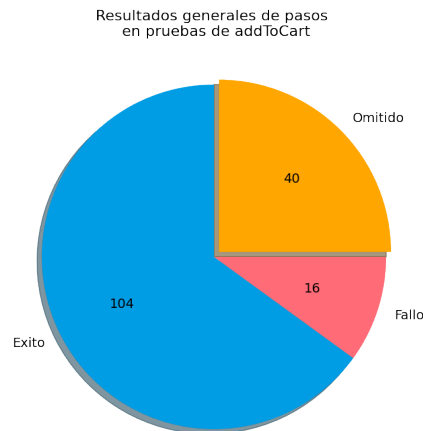


Figura 50: Pruebas de integración: Resultados pasos addToCart. Fuente: Elaboración propia

El tiempo de computo que tomó el plan de pruebas fue de 16.425[s], el cual se reduce por los pasos omitidos durante la ejecución. Los datos obtenidos en el reporte indican

que el método tuvo el comportamiento esperado para los casos con datos erróneos, dado que este retorna un mensaje de error que puede ser reconocido desde el ambiente Frontend y reflejarlo al usuario según se estime conveniente. Por otro lado, se valida que el método presenta un desarrollo que permite continuar con su ejecución pese a un error, de manera que no provoca la caída del sistema. Esto último es de vital importancia para una aplicación web dado que los usuarios pueden acceder al código desde el navegador y modificar las consultas al ambiente Backend, dando la posibilidad de que se den estos casos abarcados.

4.3.3. Escenarios: Método `removeFromCart`

Se evaluaron un total de 19 escenarios, para este plan de pruebas se dio que hubo 9 exitosos y 10 fallidos.

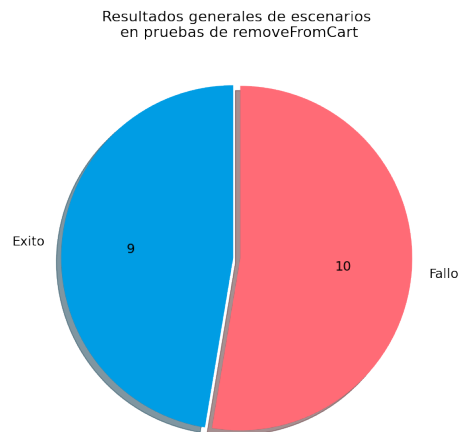


Figura 51: Pruebas de integración: Resultados escenarios `removeFromCart`. Fuente: Elaboración propia

Los escenarios fallidos se deben a lo definido en el modelo de pruebas, dado que se establecieron 2 tipos de escenario donde los datos de prueba presentaban valores que no están registrados en la base de datos. Esto último se refleja en los métodos llamados dentro de `removeFromCart`, `checkProductStock` y `updateProductStock` más específicamente, los cuales son el origen de los errores obtenidos.

4.3.4. Pasos: Método `removeFromCart`

El plan de pruebas recorrió un total de 95 pasos, el reporte indica que 60 fueron exitosos, 10 fallaron y 25 fueron omitidos tras un fallo. A continuación se dispone una visualización de lo anterior mencionado:

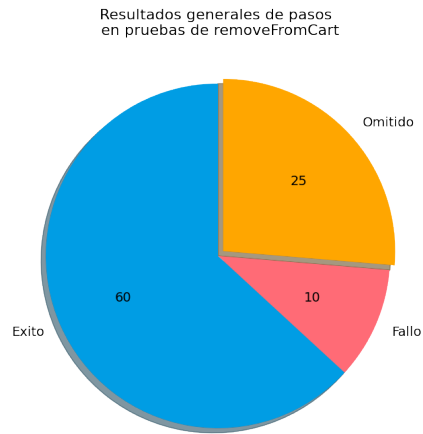


Figura 52: Pruebas de integración: Resultados pasos removeFromCart. Fuente: Elaboración propia

Dados los datos resultantes de las pruebas, se puede abstraer que el método, al igual que el anterior, cumple con un desarrollo con procesamiento de errores, evitando caídas del sistema. Analizando la herramienta para este caso y el anterior, es importante mencionar que la posibilidad de simular la caída o detención del método en subprocesos que se ejecuten dentro de este permite abarcar, desde otro foco, la evaluación del método.

4.4. Pruebas de sistema

Para el caso de cada prueba se mencionaran los datos utilizados para realizar cada iteración y los resultados obtenidos. Los inputs en común de todas las pruebas son: el número de interacciones que corresponde a la cantidad de veces que se realizará la acción en cuestión y el tiempo(ms) de espera límite para la carga de los componentes del sitio.

Los resultados obtenidos en cada prueba serán procesados previamente para visualizar los promedios correspondientes a cada iteración. Una iteración corresponde a realizar las n interacciones, donde se registrará si la prueba termino exitosamente o con errores, junto con el tiempo de computo en caso de éxito. El proceso anterior se describe a continuación:

- **Outputs:** Para facilitar el procesamiento de los resultados obtenidos en cada prueba, se decidió registrar los datos en un archivo de texto para cada prueba, siguiendo el siguiente formato:

$$t_{\text{espera}} : (\text{exito}/\text{fallo})/\text{tipo}_{\text{error}}, \text{error}, t_{\text{computo}} - \dots$$

Por ejemplo '100:true,false,1.5-Timeout,true,0' corresponde a dos pruebas con

tiempo de espera 100[ms], una exitosa con tiempo de computo de 1.5[s] y otra que tuvo un error por tiempo de espera durante su ejecución.

- **Procesamiento de resultados:** Para trabajar con la información obtenida se emplearon librerías de Python para procesar y visualizar los outputs de cada prueba.

Para efectos de este estudio, llámese 'prueba' al conjunto de repeticiones de una acción a testear, es decir, una prueba corresponde a las n repeticiones de una acción manteniendo el navegador de prueba abierto, una vez finalizado todo este proceso se registra el resultado en el archivo de texto correspondiente. Para poseer información más precisa se decidió repetir el proceso anterior (n repeticiones por cada tiempo de espera) 5 veces y trabajar en función del promedio de los resultados.

4.4.1. Acceso a catálogo de productos

- **Datos de entrada:** Para llevar a cabo las pruebas de esta funcionalidad, se emplearon los siguientes datos:

Inputs	
Número de interacciones	20
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Barra de navegación	[1, 2, ..., 50]

Tabla 17: Inputs: Acceso al catálogo de productos. Fuente: Elaboración propia.

Las interacciones de esta prueba corresponden al acceso al catálogo y la navegación utilizando la barra inferior. El input de barra de navegación es generado de manera aleatoria en cada prueba, manteniéndose dentro de los rangos indicados en la tabla.

- **Resultados:**

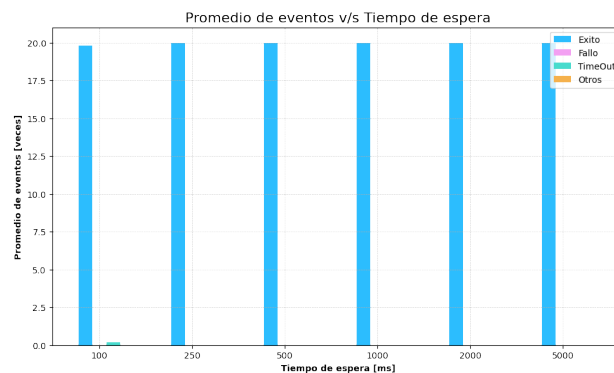


Figura 53: Gráfico resultados: Acceso a catálogo. Fuente: Elaboración propia

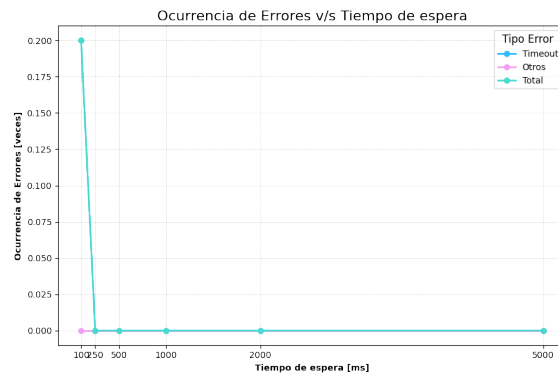


Figura 54: Gráfico errores: Acceso a catálogo. Fuente: Elaboración propia

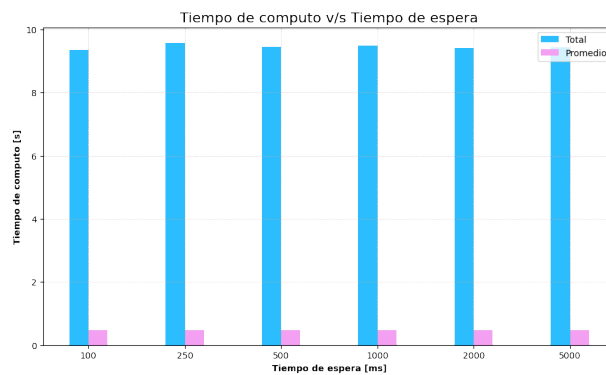


Figura 55: Gráfico tiempos: Acceso a catálogo. Fuente: Elaboración propia

- **Interpretación de resultados:** Los resultados obtenidos y visualizados en los gráficos apuntan a que la funcionalidad está implementada correctamente, pudiendo también afirmar que funcionará casi bajo cualquier escenario. Así mismo, la experiencia del usuario será óptima cuando este interactúe con el catálogo de productos.

4.4.2. Acceso a ficha de producto

- **Datos de entrada:** En esta verificación de la funcionalidad se definieron los siguientes valores para cada ejecución:

Inputs	
Número de interacciones	20
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Barra de navegación	[1, 2, ..., 50]
Número de producto	[1, 2, ..., 9]

Tabla 18: Inputs: Acceso a ficha de producto. Fuente: Elaboración propia.

El input de barra de navegación es generado de la misma manera que en la prueba anterior, para este caso se realiza el mismo procedimiento para el número de producto.

■ Resultados:

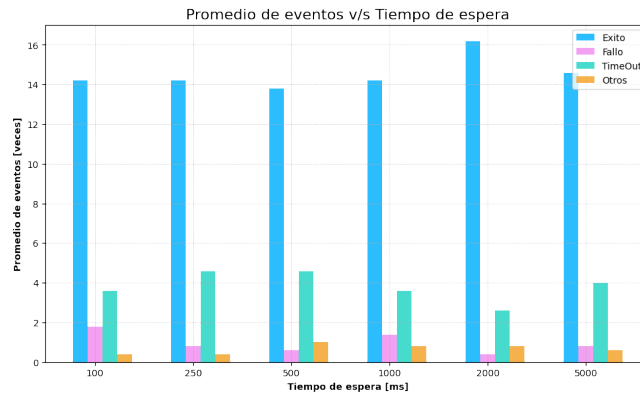


Figura 56: Gráfico resultados: Acceso a ficha de producto. Fuente: Elaboración propia

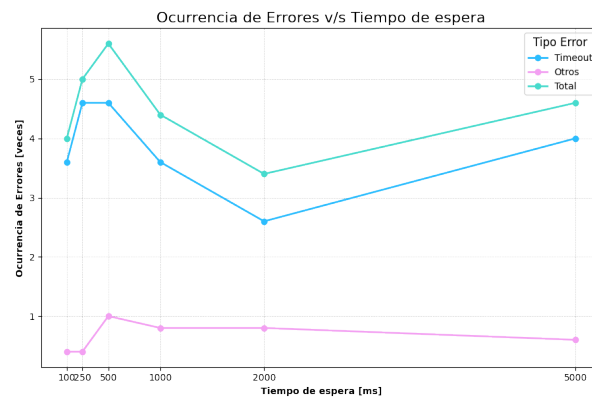


Figura 57: Gráfico errores: Acceso a ficha de producto. Fuente: Elaboración propia

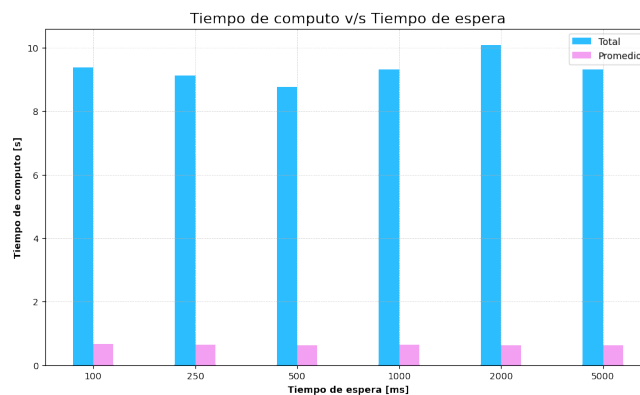


Figura 58: Gráfico tiempos: Acceso a ficha de producto. Fuente: Elaboración propia

- **Interpretación de resultados:** El comportamiento observado en las pruebas realizadas refleja, en primera instancia, que el sitio responde considerablemente rápido a la petición del usuario, llegando a un tiempo de espera límite de 2 segundos para cargar todo lo necesario. Por el lado de ambiente, se noto una reiterada demora en la funcionalidad que entrega los datos del producto, lo cual se visualiza en los errores de tiempo de espera. A partir de lo anterior y, considerando que el desarrollo de este sitio pudiese continuar a futuro, serán necesarios ajustes en el ambiente Backend.

4.4.3. Cambio de tipo de producto

- **Datos de entrada utilizados:** Para llevar a cabo las pruebas de esta funcionalidad, se emplearon los siguientes datos:

Inputs	
Número de interacciones	20
Tiempo de espera	[100, 250, 500, 1000, 2000]
Tipo de producto	[mobile, laptop, accesorios]

Tabla 19: Inputs: Cambio de tipo de producto. Fuente: Elaboración propia.

El tipo de producto es seleccionado aleatoriamente en cada iteración, afectando en la validación que se debe realizar al final de estas.

- **Resultados:**

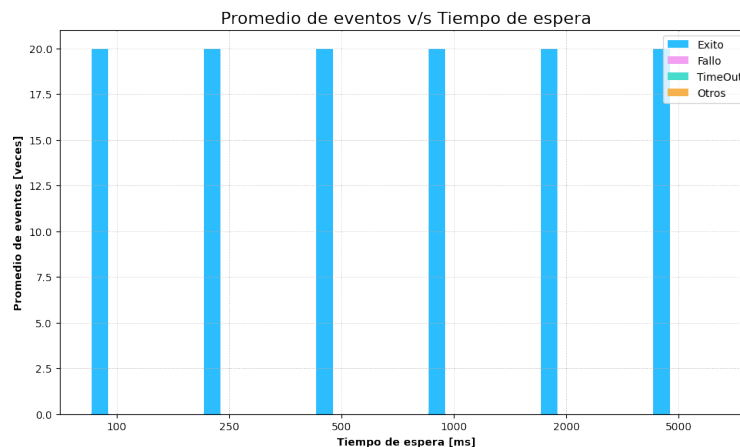


Figura 59: Gráfico resultados: Cambio de tipo de producto. Fuente: Elaboración propia

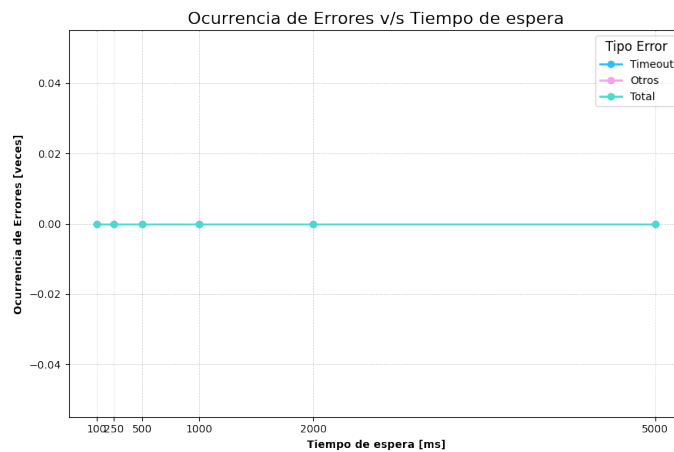


Figura 60: Gráfico errores: Cambio de tipo de producto. Fuente: Elaboración propia

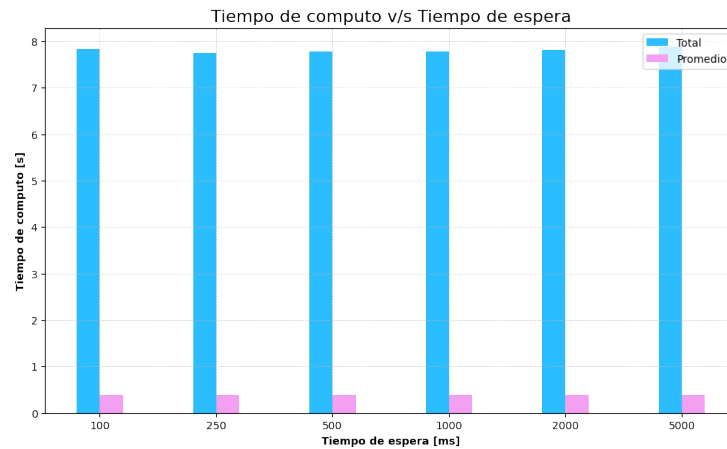


Figura 61: Gráfico tiempos: Cambio de tipo de producto. Fuente: Elaboración propia

- **Interpretación de resultados:** Al igual que en la funcionalidad de acceso al catálogo de productos, comparando los resultados, se determina que esta interacción del usuario presenta un desarrollo óptimo para la experiencia de este.

4.4.4. Filtrado de productos

- **Datos de entrada utilizados:**

Inputs	
Número de interacciones	25
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Filtros	[priceBottom, priceTop, ram, ..., has5g]

Tabla 20: Inputs: Filtrado de productos. Fuente: Elaboración propia.

A partir del input de filtros, en cada iteración se obtiene un valor aleatorio desde este arreglo, el cual es utilizado para aplicar el filtro correspondiente y realizar las validación pertinentes.

■ Resultados:

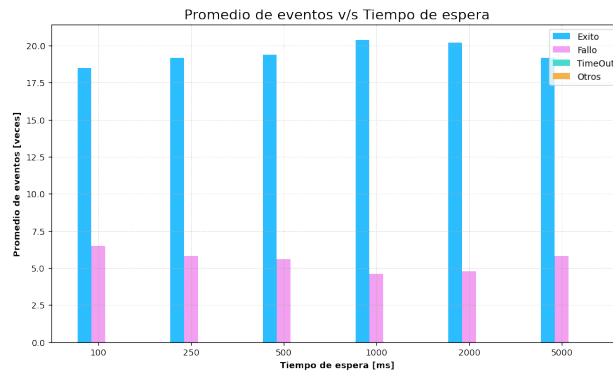


Figura 62: Gráfico resultados: Filtrado de productos. Fuente: Elaboración propia

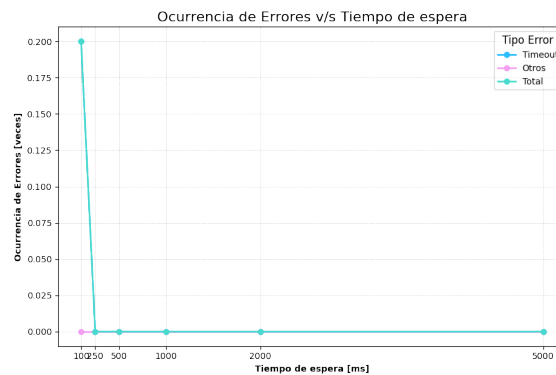


Figura 63: Gráfico errores: Filtrado de productos. Fuente: Elaboración propia

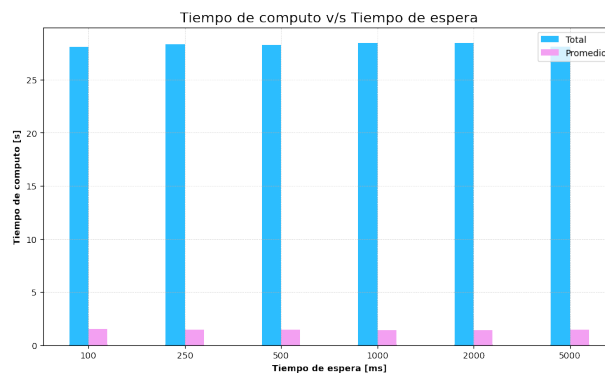


Figura 64: Gráfico tiempos: Filtrado de productos. Fuente: Elaboración propia

- **Interpretación de resultados:** Analizando la carga que ejerce esta funcionalidad sobre el sistema, se determinó que se encuentra bien implementada y entrega resultados correctos en la mayor parte de los casos. Los fallos visualizados, se revisaron a nivel de código, llegando a que se generan porque el navegador no se sincroniza totalmente con las pruebas, dado que la URL del sitio no se actualiza a tiempo para las validaciones realizadas en cada iteración.

4.4.5. Agregar producto a carro de compras

- **Datos de entrada utilizados:**

Inputs	
Número de interacciones	20
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Barra de navegación	[1, 2, ..., 50]
Número de producto	[1, 2, ..., 9]

Tabla 21: Inputs: Agregar producto al carro de compras. Fuente: Elaboración propia.

El número de interacciones corresponde a la cantidad de productos que serán añadidos al carro y, a su vez, a la cantidad de validaciones que se realizarán. El input de barra de navegación y el número de producto son generados de manera aleatoria en cada prueba, manteniéndose dentro de los rangos indicados en la tabla.

- **Resultados:**

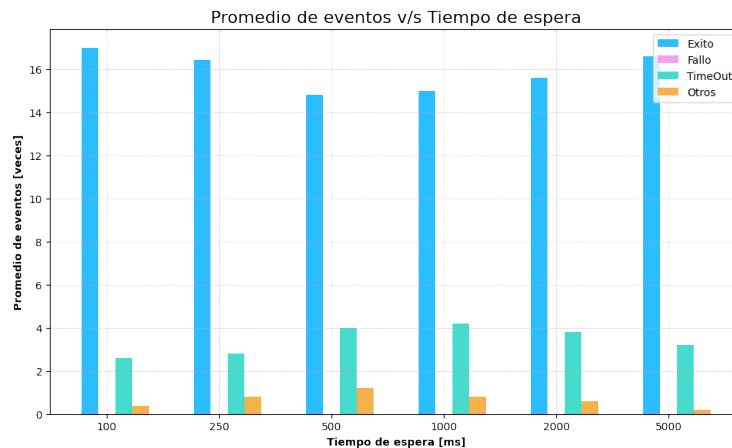


Figura 65: Gráfico resultados: Agregar al carro. Fuente: Elaboración propia

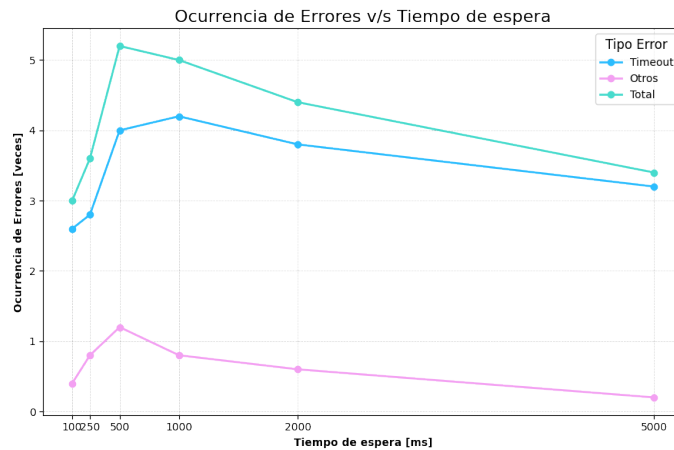


Figura 66: Gráfico errores: Agregar al carro. Fuente: Elaboración propia

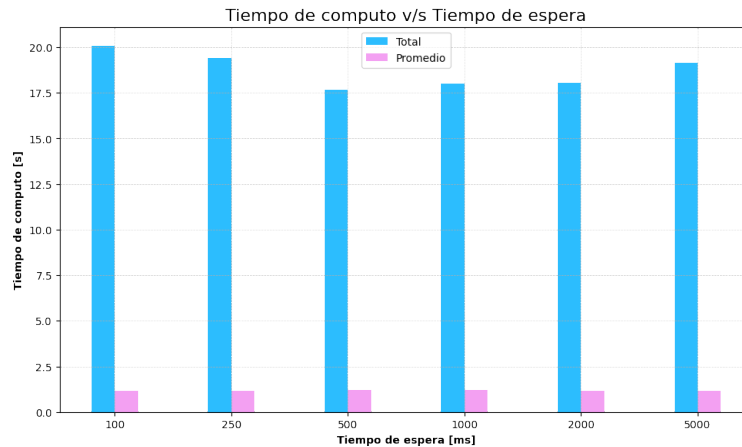


Figura 67: Gráfico tiempos: Agregar al carro. Fuente: Elaboración propia

- **Interpretación de resultados:** La funcionalidad presenta inestabilidad al trabajar bajo los valores intermedios del intervalo de tiempos de espera, los errores por Timeout se presentan más frecuentemente en estos datos y se reducen a medida que aumentan. Los tiempos de computo se mantienen estables a pesar de la variación del tiempo de espera. De lo anterior se puede obtener como requerimiento el optimizar el procesamiento en el ambiente Backend, dado que de tratarse de un ambiente productivo, los tiempos intermedios de espera serían más frecuentados con un tráfico normal de clientes.

4.4.6. Remover producto del carro de compras

- **Datos de entrada utilizados:** En esta prueba el número de interacciones corresponde a la cantidad de productos que serán añadidos y removidos del carro

de compras.

Inputs	
Número de interacciones	10
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Barra de navegación	[1, 2, ..., 50]
Número de producto	[1, 2, ..., 9]

Tabla 22: Inputs: Remover producto desde el carro de compras.
Fuente: Elaboración propia.

El input de barra de navegación y el número de producto son generados de manera aleatoria en cada prueba, manteniéndose dentro de los rangos indicados en la tabla.

■ Resultados:

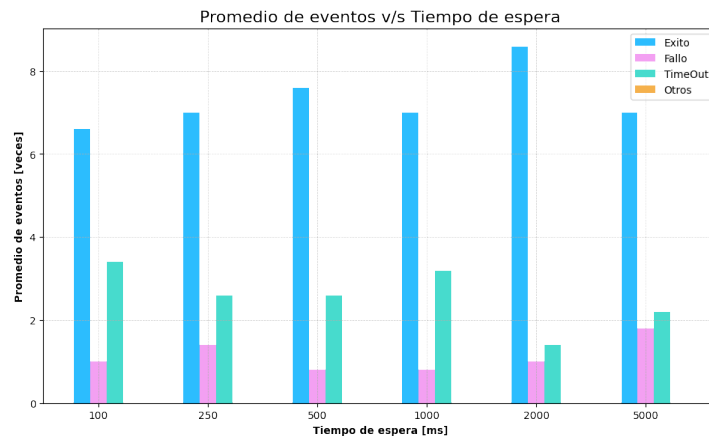


Figura 68: Gráfico resultados: Remover producto del carro de compras.
Fuente: Elaboración propia

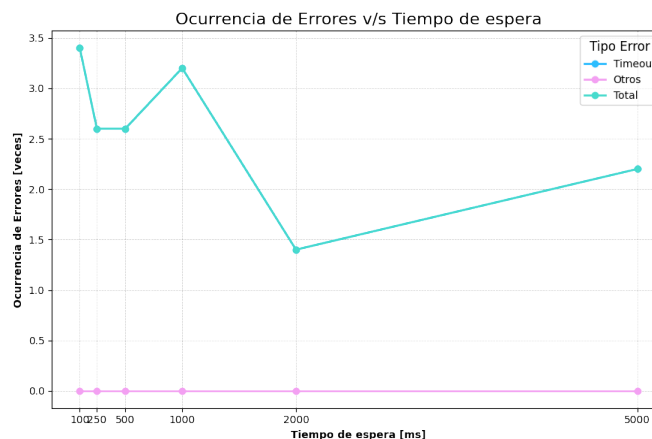


Figura 69: Gráfico errores: Remover producto del carro de compras.
Fuente: Elaboración propia

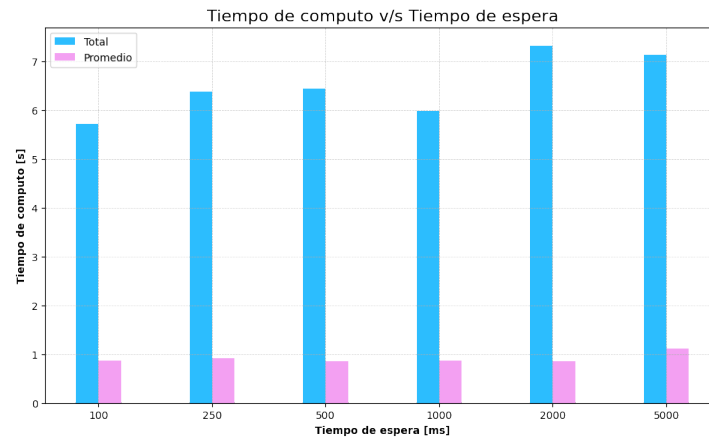


Figura 70: Gráfico tiempos: Remover producto del carro de compras.
Fuente: Elaboración propia

- **Interpretación de resultados:** Debido al parecido lógico que posee esta interacción con la anterior, el resultado obtenido es similar, esto se ve reflejado en la inestabilidad que se observa al variar los tiempos de espera. Abordando los resultados de esta prueba desde el punto de vista de negocio, se debe realizar una revisión y, de ser necesario, modificar la funcionalidad de tratarse de un problema de optimización del código. De no deberse a un problema en el desarrollo, los datos obtenidos apuntan a un problema con los recursos que posee la aplicación web para mantenerse en línea, por lo que se debería realizar un análisis y evaluación de infraestructura.

4.4.7. Flujo completo de compra

- **Datos de entrada utilizados:** En esta prueba el número de interacciones corresponde a la cantidad de flujos que se realizarán. Como consideración, para esta prueba, por conflictos con las variables de sesión utilizadas en el sitio, se decidió cerrar y abrir nuevamente el navegador de prueba al final de cada flujo. La canti-

Inputs	
Número de interacciones	10
Tiempo de espera	[100, 250, 500, 1000, 2000, 5000]
Barra de navegación	[1, 2, ..., 50]
Número de producto	[1, 2, ..., 9]
Cantidad de productos	5

Tabla 23: Inputs: Flujo completo de compra. Fuente: Elaboración propia.

dad de productos corresponde a la cantidad que será añadida al carro de compras para efectuar el flujo completo. El input de barra de navegación y el número de producto son generados de manera aleatoria en cada prueba, manteniéndose dentro de los rangos indicados en la tabla.

■ Resultados:

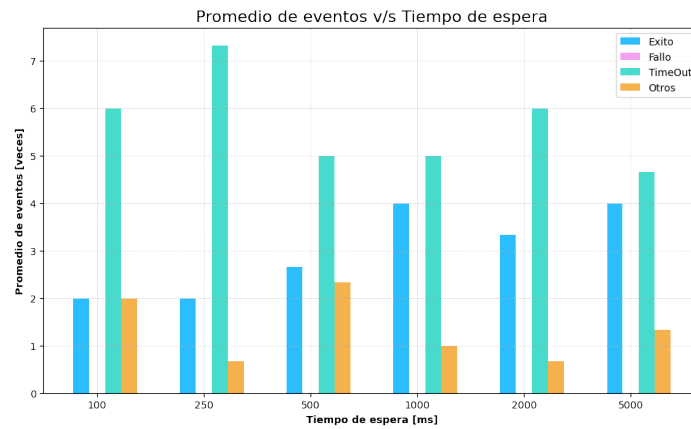


Figura 71: Gráfico resultados: Flujo completo de compra. Fuente: Elaboración propia

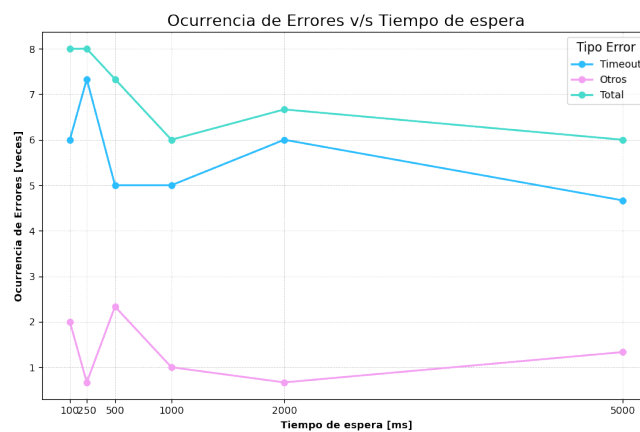


Figura 72: Gráfico errores: Flujo completo de compra. Fuente: Elaboración propia

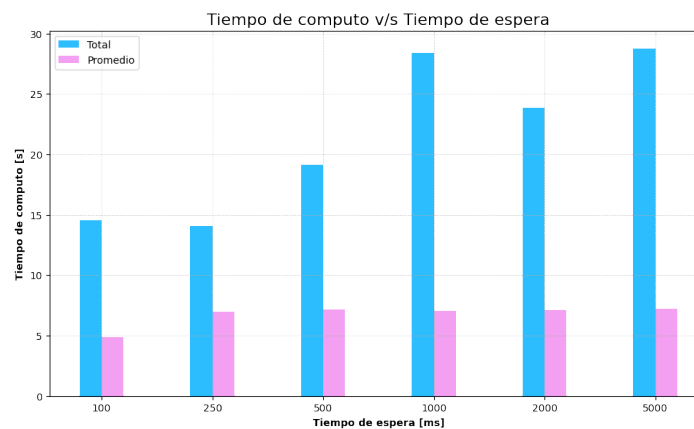


Figura 73: Gráfico tiempos: Flujo completo de compra. Fuente: Elaboración propia

- **Interpretación de resultados:** En comparación con los resultados anteriores, esta prueba del flujo de compras presenta una mejora significativa en sus resultados en tiempos de espera igual o superiores a 1[s], reduciendo los errores por timeout y teniendo una mayor tasa de éxito en cada iteración. Lo anterior refleja que el sitio, con los componentes del dispositivo utilizado como servidor, entrega una experiencia de usuario conveniente, dado que ofrece al usuario respuestas visuales en tiempos aceptables.

4.5. Pruebas de rendimiento

En esta sección se presentarán los resultados obtenidos utilizando Jmeter, junto con la estructura y los datos utilizados para llevar a cabo cada escenario. Finalmente, se realizará un análisis general de los resultados obtenidos a partir de los casos abordados.

Se utilizarán los receptores de información de Jmeter para visualizar y analizar la información de salida en cada prueba, más específicamente el reporte resumen, árbol de resultados y gráfico. Se realizará un gráfico con todos los puntos de acceso y sus tiempos de espera para cada caso de prueba, los valores a presentar en estos gráficos serán la media aritmética, cota inferior y cota superior obtenida, esto último se debe a que es información relevante para realizar un análisis del comportamiento del sistema.

4.5.1. Pruebas con múltiples consultas a un mismo endpoint

En esta sección se espera evaluar el comportamiento del ambiente Backend al trabajar bajo múltiples consultas sobre un punto de acceso. Un ejemplo real de este tipo de interacción se da cuando a un sitio de ventas accede una gran cantidad de clientes, como se da cuando hay ofertas especiales.

Este escenario fue abordado estableciendo un plan de pruebas con todos los puntos de accesos mencionados en la sección anterior. Se realizaron iteraciones variando el número de accesos simultáneos, acorde al siguiente intervalo [5, 10, 20, 50, 100], para obtener información más precisa de cada caso se realizaron 10 iteraciones para cada prueba, es decir, se repitió cada prueba con un mismo acceso simultáneo esa cantidad de veces.

Los resultados presentados a continuación corresponden a las pruebas que no generaron conflictos en el ambiente. Se realizaron pruebas con cantidades superiores a las nombradas, pero el sitio presentó caídas reiteradas durante la ejecución de la primera iteración de consultas, trayendo como consecuencia que todas las conexiones subsiguientes a la caída retornaran un error. Se analizarán los casos especiales más adelante.

Caso 1: Cinco conexiones

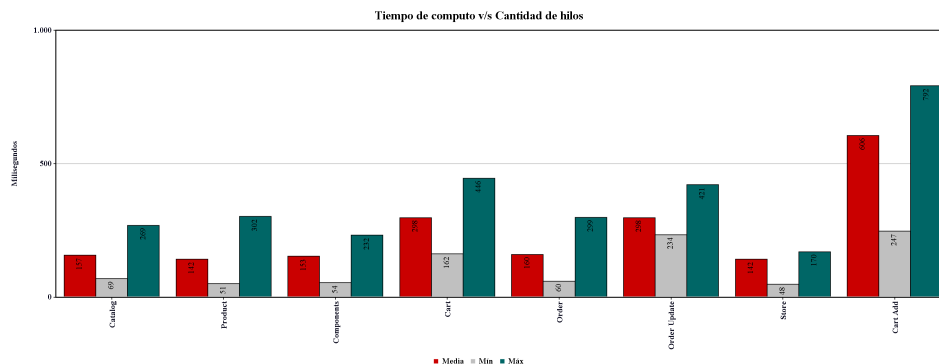


Figura 74: Accesos simultáneos: Gráfico 5 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	50	157	69	269	31,21	0,00%	2,58933	1510,06	0,48	597181
Product	50	142	51	302	39,73	0,00%	2,58371	0,57	0,41	224
Components	50	153	54	232	24	0,00%	2,57374	1,01	0,46	402
Cart	50	298	162	446	46,18	0,00%	2,54803	1,79	0,41	717,7
Order	50	160	60	299	34,15	0,00%	2,55624	0,72	0,5	289
Order Update	50	298	234	421	32,35	0,00%	2,52717	0,71	0,84	286
Store	50	142	48	170	22,64	0,00%	2,54959	1,21	0,39	485
Cart Add	50	606	247	792	94,23	0,00%	2,53344	0,68	0,6	274
Total	400	245	48	792	157,27	0,00%	19,5141	1428,92	3,91	74982,3

Figura 75: Accesos simultáneos: Tabla resultados 5 accesos. Fuente: JMeter

Caso 2: Diez conexiones

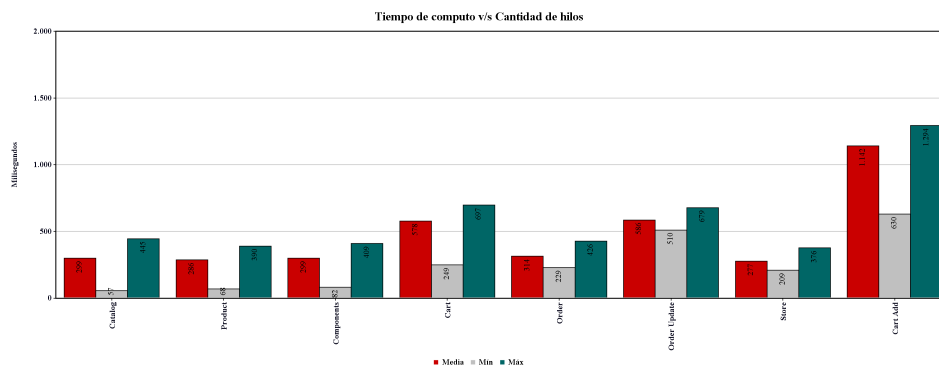


Figura 76: Accesos simultáneos: Gráfico 10 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	100	299	57	445	53,81	0,00%	2,78381	1623,47	0,52	597181
Product	100	286	68	390	44,15	0,00%	2,76426	0,6	0,44	224
Components	100	299	82	409	39,11	0,00%	2,75923	1,08	0,49	402
Cart	100	578	249	697	53,53	0,00%	2,72428	1,91	0,44	718
Order	100	314	229	426	32,57	0,00%	2,71747	0,77	0,53	289
Order Update	100	586	510	679	32,54	0,00%	2,69651	0,75	0,9	286
Store	100	277	209	376	22,03	0,00%	2,7274	1,29	0,42	485
Cart Add	100	1142	630	1294	114,81	0,00%	2,69876	0,72	0,64	274
Total	800	473	57	1294	286,22	0,00%	20,64197	1511,51	4,13	74982,4

Figura 77: Accesos simultáneos: Tabla resultados 10 accesos. Fuente: JMeter

Caso 3: Veinte conexiones

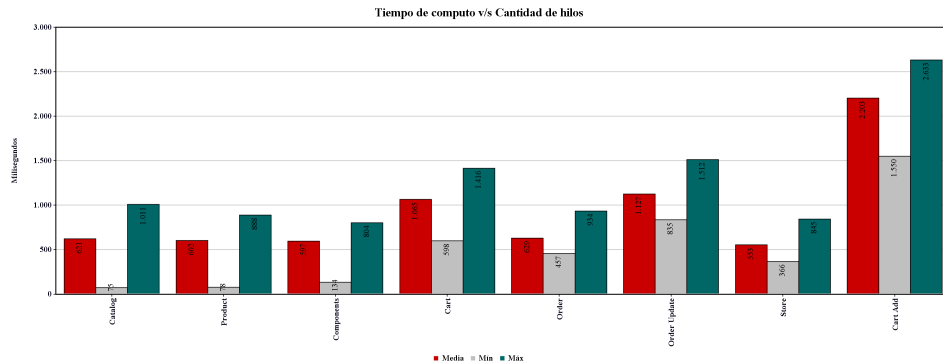


Figura 78: Accesos simultáneos: Gráfico 20 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	200	621	75	1011	119,5	0,00%	2,9146	1699,75	0,54	597181
Product	200	602	78	888	92,09	0,00%	2,89448	0,63	0,46	224
Components	200	597	134	804	76,69	0,00%	2,87216	1,13	0,51	402
Cart	200	1065	598	1416	94,55	0,00%	2,83443	1,99	0,45	718
Order	200	629	457	934	89,82	0,00%	2,8362	0,8	0,55	289
Order Update	200	1127	835	1512	92,44	0,00%	2,81666	0,79	0,94	286
Store	200	553	366	845	51,84	0,00%	2,83881	1,34	0,44	485
Cart Add	200	2203	1550	2633	123,66	0,00%	2,8003	0,75	0,66	274
Total	1600	925	75	2633	535,63	0,00%	21,39838	1566,9	4,28	74982,4

Figura 79: Accesos simultáneos: Tabla resultados 20 accesos. Fuente: JMeter

Caso 4: Cincuenta conexiones

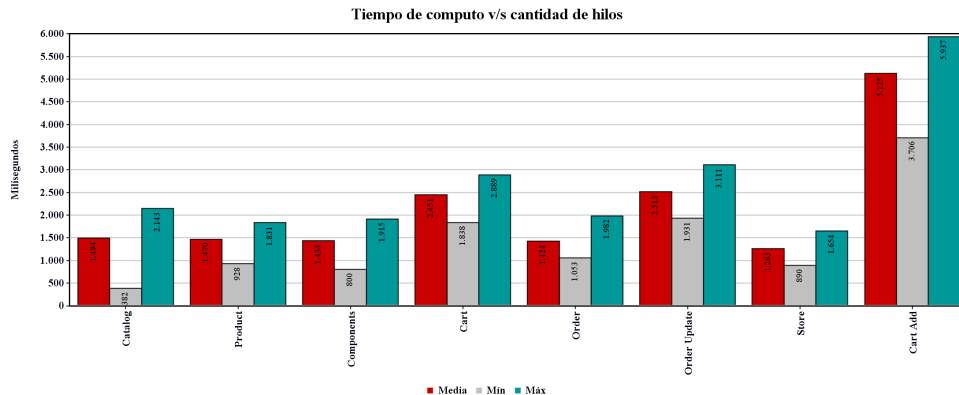


Figura 80: Accesos simultáneos: Gráfico 50 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	500	1494	382	2143	225,36	0,00%	3,14523	1834,25	0,58	597181
Product	500	1470	928	1831	126,73	0,00%	3,12737	0,68	0,5	224
Components	500	1434	800	1915	128,23	0,00%	3,12262	1,23	0,55	402
Cart	500	2451	1838	2889	176,04	0,00%	3,1059	2,18	0,5	720
Order	500	1424	1053	1982	148,73	0,00%	3,13109	0,88	0,61	289
Order Update	500	2513	1931	3111	186,74	0,00%	3,11181	0,87	1,04	286
Store	500	1263	890	1654	129,14	0,00%	3,13228	1,48	0,48	485
Cart Add	500	5125	3706	5937	293,88	0,00%	3,08907	0,83	0,73	274
Total	4000	2147	382	5937	1228,16	0,00%	23,14266	1694,63	4,63	74982,6

Figura 81: Accesos simultáneos: Tabla resultados 50 accesos. Fuente: JMeter

Caso 5: Cien conexiones

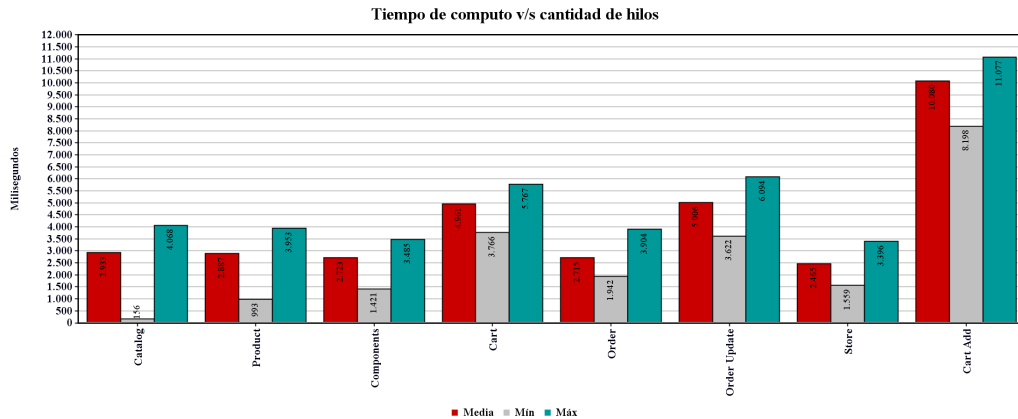


Figura 82: Accesos simultáneos: Gráfico 100 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	700	2933	156	4068	527,22	0,00%	3,33297	1943,74	0,62	597181
Product	700	2887	993	3953	302,77	0,00%	3,29818	0,72	0,53	224
Components	700	2723	1421	3485	248,82	0,00%	3,26471	1,28	0,58	402
Cart	700	4961	3766	5767	243,62	0,00%	3,24178	2,28	0,52	720
Order	700	2715	1942	3904	289,22	0,00%	3,28401	0,93	0,64	289
Order Update	700	5006	3622	6094	330,43	0,00%	3,25058	0,91	1,08	286
Store	700	2465	1559	3396	253,89	0,00%	3,28341	1,56	0,5	485
Cart Add	700	10080	8198	11077	437,77	0,00%	3,19537	0,86	0,76	274
Total	5600	4221	156	11077	2435,79	0,00%	23,52664	1722,74	4,71	74982,6

Figura 83: Accesos simultáneos: Tabla resultados 100 accesos. Fuente: JMeter

Para el intervalo de conexiones simultáneas [5, 20] los tiempos de computo son relativamente estables, salvo para el caso del punto de acceso *Cart Add*, dado que este presenta una considerable variación en estos tiempos en todos los casos, a partir de las 50 conexiones se refleja un alto aumento en el retardo de respuesta y en la variación de este. El rendimiento del sistema es similar en cada escenario, indicando que, en promedio, se resuelven 3 peticiones por segundo.

Descartando la opción de realizar cambios en el código del sistema, estas pruebas señalan un rendimiento regular para una tienda de bajo alcance (por ejemplo una pyme), obteniendo una experiencia de usuario regular para un flujo de 5 a 50 clientes. En caso de tener un flujo fuera del rango indicado, las interacciones de los clientes tardan más en dar responder y, considerando el tiempo de carga de la vista, este retardo aumenta.

Las pruebas llevadas a cabo con más de 100 accesos dieron como resultado caídas del sistema, por consiguiente debe limitarse la cantidad de conexiones permitidas o realizar mejoras a nivel de infraestructura. En la presente tabla se entregan los resultados obtenidos en 2 iteraciones con 120 conexiones simultáneas:

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	240	2976	199	4518	1203,4	0,00%	2,198	1281,84	0,37	597181
Product	240	3353	1517	4066	537,51	5,00%	2,1475	0,48	0,34	229,7
Components	240	3130	2331	3955	363,17	32,50%	2,13462	2,22	0,26	1064,7
Cart	240	4036	12	6749	2129,77	52,92%	2,17145	3,59	0,17	1694,6
Order	240	4944	2026	82494	13455,99	52,92%	2,25678	3,45	0,21	1563,6
Order Update	233	6584	2025	82494	14987,31	55,37%	2,2302	3,53	0,33	1620
Store	224	2289	2023	3200	295,39	53,57%	2,20334	3,6	0,16	1674,8
Cart Add	224	5868	2024	11153	4126,92	53,57%	2,21879	3,42	0,24	1576,9

Figura 84: Accesos simultáneos: Tabla resultados 120 accesos. Fuente: JMeter

Tal como se mencionó previamente, en la tabla se puede observar que la tasa de errores aumentó considerablemente a medida que se ejecutan iteraciones con esta cantidad de consultas. Se detuvo su ejecución a las 2 iteraciones debido a que los resultados posteriores a la caída del sistema son errores.

4.5.2. Pruebas con consultas simultáneas a distintos endpoints

Para este escenario se busca evaluar como se comporta el sitio bajo múltiples usos simultáneos de distintas funcionalidades. Lo anterior es equivalente al caso que se da cuando un grupo de usuarios se encuentra interactuando en distintas vistas del sitio de manera sincrónica, provocando así un flujo moderado de consultas al ambiente Backend.

Estas pruebas se abordaron utilizando el número de hilos como una variable global que será establecida en cada grupo de hilos, cada punto de acceso será asignado a un grupo distinto. Se utilizó el intervalo [3, 5, 8, 10, 15] para las consultas simultáneas a cada endpoint del ambiente. Cabe mencionar que se realizaron la misma cantidad de iteraciones para cada valor. Se abordarán los casos conflictivos más adelante.

Caso 1: Tres conexiones

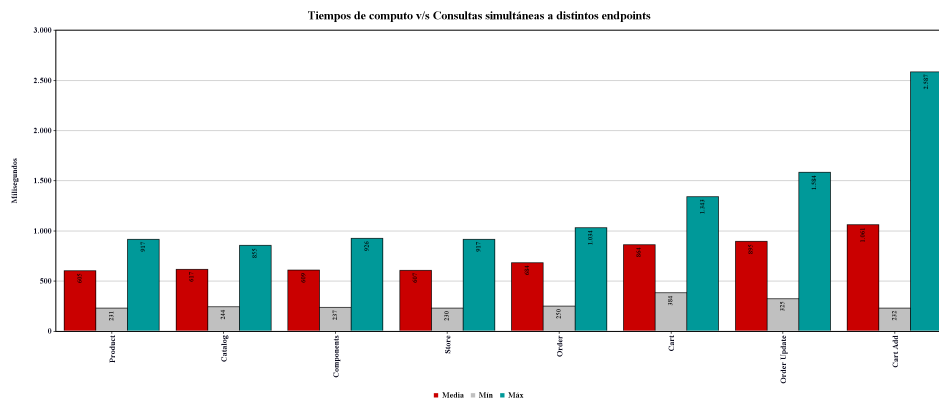


Figura 85: Consultas simultáneas: Gráfico 3 accesos. Fuente: JMeter

ESTUDIO Y EVALUACIÓN DE HERRAMIENTAS DE TESTING EN EL ÁREA DEL DESARROLLO WEB

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Product	30	605	231	917	130,11	0,00%	4,25532	0,93	0,68	224
Catalog	30	617	244	855	134,12	0,00%	4,28694	2500,08	0,8	597181
Components	30	609	237	926	129,33	0,00%	4,25532	1,67	0,76	402
Store	30	607	230	917	130,76	0,00%	4,25955	2,02	0,65	485
Order	30	684	250	1034	189,52	0,00%	3,89813	1,1	0,76	289
Cart	30	864	384	1343	358,58	0,00%	3,24535	2,55	0,52	805
Order Update	30	895	325	1584	414,7	0,00%	3,18979	0,89	1,06	286
Cart Add	30	1061	232	2587	821,22	0,00%	2,71174	0,73	0,64	274
Total	240	743	230	2587	402,11	0,00%	21,68609	1588,19	4,34	74993,2

Figura 86: Consultas simultáneas: Tabla resultados 3 accesos. Fuente: JMeter

Caso 2: Cinco conexiones

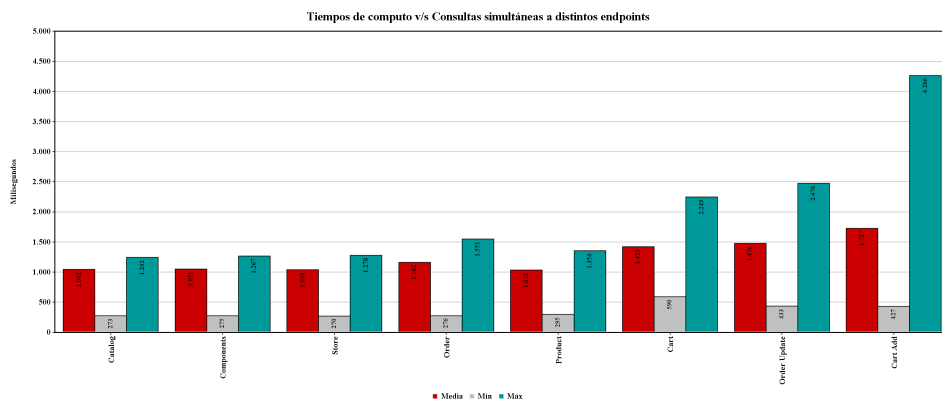


Figura 87: Consultas simultáneas: Gráfico 5 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	50	1042	273	1242	192,13	0,00%	4,31742	2517,85	0,8	597181
Components	50	1051	275	1267	196,14	0,00%	4,28449	1,68	0,76	402
Store	50	1038	270	1278	187,12	0,00%	4,32938	2,05	0,66	485
Order	50	1162	276	1551	313,86	0,00%	3,96385	1,12	0,77	289
Product	50	1032	295	1354	199,72	0,00%	4,30589	0,94	0,69	224
Cart	50	1423	590	2249	616,26	0,00%	3,34247	2,63	0,54	805
Order Update	50	1476	433	2476	720,69	0,00%	3,2514	0,91	1,08	286
Cart Add	50	1727	427	4266	1407,54	0,00%	2,81468	0,75	0,67	274
Total	400	1244	270	4266	672,72	0,00%	22,50225	1647,97	4,5	74993,2

Figura 88: Consultas simultáneas: Tabla resultados 5 accesos. Fuente: JMeter

Caso 3: Ocho conexiones

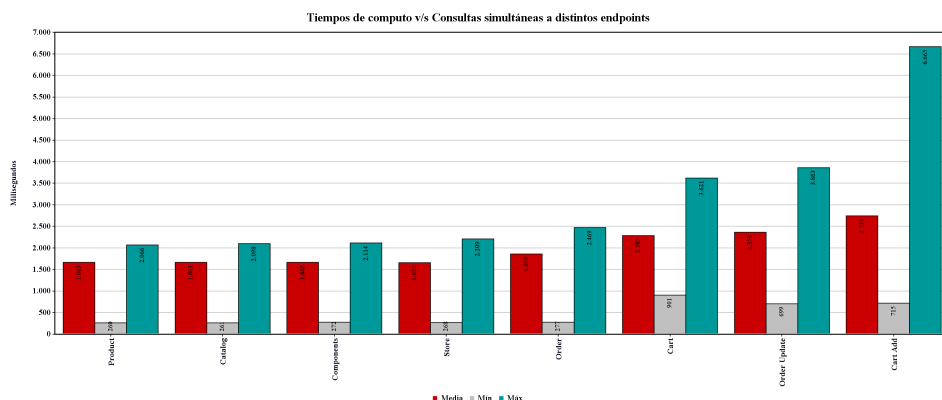


Figura 89: Consultas simultáneas: Gráfico 8 accesos. Fuente: JMeter

ESTUDIO Y EVALUACIÓN DE HERRAMIENTAS DE TESTING EN EL ÁREA DEL DESARROLLO WEB

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Product	80	1663	260	2066	271,34	0,00%	4,46254	0,98	0,71	224
Catalog	80	1663	261	2098	275	0,00%	4,41623	2575,48	0,82	597181
Components	80	1665	272	2114	283,56	0,00%	4,42258	1,74	0,79	402
Store	80	1655	268	2209	285,75	0,00%	4,47302	2,12	0,69	485
Order	80	1859	277	2469	478,98	0,00%	4,11734	1,16	0,8	289
Cart	80	2285	901	3621	950,45	0,00%	3,38581	2,66	0,54	805
Order Update	80	2359	699	3863	1102,09	0,00%	3,28812	0,92	1,09	286
Cart Add	80	2739	715	6665	2233,11	0,00%	2,8681	0,77	0,68	274
Total	640	1986	260	6665	1053,49	0,00%	22,93249	1679,47	4,59	74993,2

Figura 90: Consultas simultáneas: Tabla resultados 8 accesos. Fuente: JMeter

Caso 4: Diez conexiones

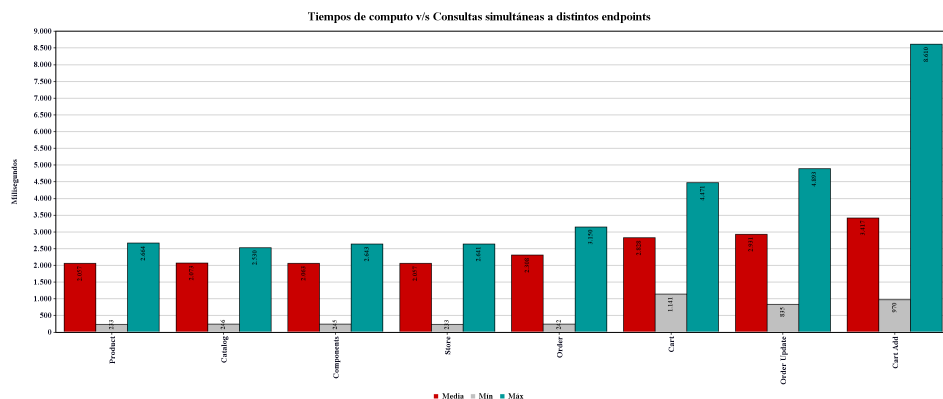


Figura 91: Consultas simultáneas: Gráfico 10 accesos. Fuente: JMeter

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Product	100	2057	233	2664	357,5	0,00%	4,53186	0,99	0,72	224
Catalog	100	2073	246	2530	347,82	0,00%	4,45295	2596,89	0,83	597181
Components	100	2063	245	2643	351,33	0,00%	4,47768	1,76	0,8	402
Store	100	2057	233	2641	349,79	0,00%	4,48571	2,12	0,69	485
Order	100	2308	242	3150	633,48	0,00%	4,09651	1,16	0,8	289
Cart	100	2828	1141	4471	1151,88	0,00%	3,427	2,69	0,55	805
Order Update	100	2931	835	4893	1385,86	0,00%	3,32591	0,93	1,11	286
Cart Add	100	3417	970	8610	2777,05	0,00%	2,88509	0,77	0,68	274
Total	800	2467	233	8610	1312,64	0,00%	23,06672	1689,31	4,62	74993,2

Figura 92: Consultas simultáneas: Tabla resultados 10 accesos. Fuente: JMeter

Caso 5: Quince conexiones

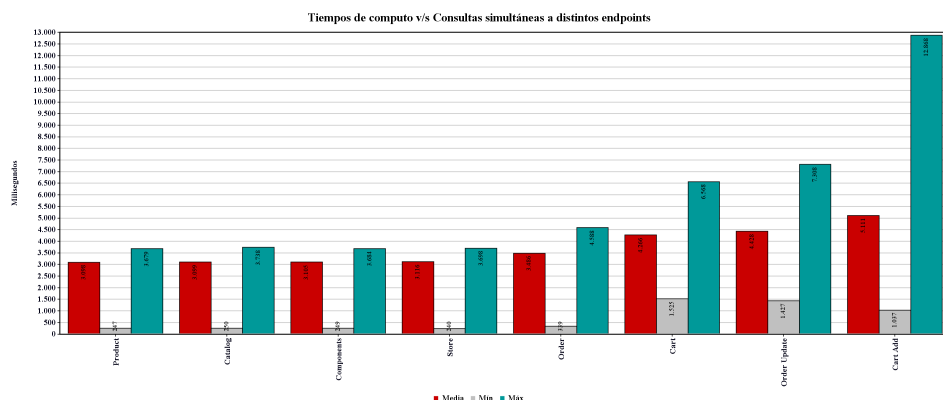


Figura 93: Consultas simultáneas: Gráfico 15 accesos. Fuente: JMeter

ESTUDIO Y EVALUACIÓN DE HERRAMIENTAS DE TESTING EN EL ÁREA DEL DESARROLLO WEB

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Product	150	3098	247	3679	483,34	0,00%	4,5159	0,99	0,72	224
Catalog	150	3099	250	3738	505,41	0,00%	4,51155	2631,07	0,84	597181
Components	150	3105	249	3684	483,37	0,00%	4,51753	1,77	0,8	402
Store	150	3116	240	3698	461,12	0,00%	4,58057	2,17	0,7	485
Order	150	3486	339	4588	835,02	0,00%	4,12859	1,17	0,81	289
Cart	150	4266	1525	6568	1723,11	0,00%	3,44084	2,7	0,55	805
Order Update	150	4428	1427	7308	2046,12	0,00%	3,31712	0,93	1,1	286
Cart Add	150	5111	1037	12868	4206,08	0,00%	2,8976	0,78	0,69	274
Total	1200	3714	240	12868	1961,59	0,00%	23,16871	1696,77	4,64	74993,2

Figura 94: Consultas simultáneas: Tabla resultados 15 accesos. Fuente: JMeter

A nivel de peticiones, el primer caso de este escenario de pruebas se asimila al de 20 conexiones en el anterior, dado que este realizan hasta 24 peticiones simultáneas. Comparando resultados se puede observar que los **tiempos de computo promedio** para todos los servicios **son menores y rodean los 1000[ms]**. Lo anterior es un indicador de buen rendimiento para un flujo de 24 usuarios, manteniendo la estabilidad del sistema.

Lo mencionado previamente se replica para todas las pruebas realizadas, dado que los tiempos de computo promedio son significativamente menores que los casos de carga en un solo endpoint. El último caso, que representa **una carga de hasta 120 conexiones simultáneas** al sistema, asegura que **el sitio soporta esta cantidad de clientes**, pero los tiempos de computo se disparan, llegando a los 5 segundos promedio para realizar la lógica de agregar un producto al carro de compras. Siendo este un caso más realista de como se comporta un sitio de Ecommerce, lo obtenido implica que el sistema soporta un flujo de hasta 120 clientes, pero para entregar un rendimiento que no afecte la experiencia del usuario de deben limitar los accesos a 100 conexiones.

Por otro lado, se efectuó una instancia de prueba con 16 conexiones simultáneas en cada servicio, provocando una caída del sistema y posterior falla de todas las consultas realizadas tras ocurrido esto. Tanto este caso como el que se entregó en la sección anterior generan como requerimiento urgente el desarrollo de una funcionalidad que permita recuperar el ambiente tras caídas por demasiadas conexiones, de manera que se detecten estos casos y realizar un reinicio del sistema de manera automatizada. Los resultados de lo anterior mencionado se presentan a continuación:

Endpoint	# Muestras	Media	Min	Max	Desv. Estándar	% Error	Rendimiento	Kb/sec	Resp KB/sec	Media de Bytes
Catalog	32	1950	181	3049	973,3	65,63%	5,64872	1141,51	0,36	206932,6
Components	32	1883	21	3211	1072,83	62,50%	5,65571	9,49	0,38	1717,8
Store	32	1874	13	3074	1033,79	65,63%	5,67275	9,66	0,33	1744,3
Product	32	1892	18	3082	1038,31	65,63%	5,65072	9,49	0,31	1720,3
Order	32	2267	182	3295	831,89	84,38%	5,66372	12,15	0,17	2195,8
Cart	32	2524	374	3553	685,8	96,88%	5,65871	13,89	0,03	2514,1
Order Update	32	2582	2028	3605	586,94	100,00%	5,66773	14,24	0	2573,5
Cart Add	32	2579	2030	3598	583,54	100,00%	5,67477	14,26	0	2573,5
Total	256	2194	13	3605	925,66	80,08%	45,18976	1224,47	1,57	27746,5

Figura 95: Consultas simultáneas: Tabla resultados 16 accesos. Fuente: JMeter

4.6. Evaluación

4.6.1. Resultados

	Estado de las pruebas	Manejo de errores
Cucumber	Al finalizar las pruebas, entrega el resultado y estado de cada escenario recorrido, junto con los estados de los pasos ejecutados. En caso de algún error durante la ejecución, notifica, por medio de la consola, sobre el error ocurrido, especificando el paso y el escenario en el que se produjo. Estos estados poseen un color propio y se ve también en el reporte que genera la herramienta.	Finaliza el escenario en ejecución, dando información sobre el paso donde ocurrió. A nivel de código es posible implementar una funcionalidad que reciba los errores y modificar el mensaje que será desplegado en consola cuando el paso falle, esto se debe desarrollar en cada paso.
Selenium	Se debe desarrollar código para extraer datos para luego trabajarlos. Permite visualizar en tiempo real la interacción que se realice con el sistema.	Se debe incluir manualmente el manejo de errores o la ejecución del plan de pruebas finaliza.
JMeter	Posee la opción de visualizar en tiempo real los resultados de cada hilo ejecutado. Por otro lado, también permite registrar en una tabla los resultados de cada prueba.	Da la opción de elegir el como proceder, se pueden detener o continuar las pruebas. En caso de no seleccionar alguna opción, JMeter continua con la ejecución del resto del plan. Los errores pueden ser registrados y graficados en tiempo real, permitiendo identificar comportamientos del sitio.
	Visualización de resultados	Formato de resultados
Cucumber	Genera un reporte contabilizando el estado de todos los escenarios evaluados, a su vez, incluye la opción de visualizar cada modelo con sus escenarios y pasos ejecutados. En cada paso, de haber ocurrido, incluye el mensaje de error. Lo anterior es subido al sitio oficial de Cucumber, permitiendo el acceso de otros usuarios a los resultados de las pruebas, este reporte se mantiene en línea durante 24 horas y permite editar el modelo de pruebas por medio del navegador.	No permite el trabajo directo con los datos, dado que el reporte generado no da la opción de extraer la información para procesarla. Se debe hacer un paso a paso en cada escenario para extraer datos de interés.
Selenium	La herramienta no incluye funcionalidades para visualizar los datos. Es necesario implementar e insertar dentro del código un registro de la información, para luego procesar y trabajarla, tal como se realizó en las pruebas vistas anteriormente.	Dado que se debe implementar un registro para almacenar los resultados, el formato queda a elección del usuario, pero la herramienta no incluye esta extensión.
JMeter	Incluye utensilios que permiten generar tablas y gráficos, dando la factibilidad de personalizarlos seleccionando que datos plasmar, etiquetas, nombres de los ejes, etc.	Facilita la opción de exportar los resultados obtenidos a formato CSV, permitiendo así su rápido procesamiento, así también para los gráficos, generando un archivo PNG.

Tabla 24: Tabla comparativa: Resultados entregados por cada herramienta.

Fuente: Elaboración propia.

4.6.2. Versatilidad

	Pruebas BackEnd (Resultados)	Pruebas FrontEnd (Resultados)
Cucumber	Las pruebas unitarias y de integración que se implementaron utilizando Cucumber demuestran que esta herramienta posee la capacidad de realizar validaciones e imponer condiciones a los resultados. Con esto, cuenta con la facultad de llevar a cabo esta clase de pruebas.	Si bien no se realizaron pruebas con esta herramienta en la parte visual del sistema, se evaluó la posibilidad de llevarla a este ambiente. Como resultado de la evaluación se comprobó que Cucumber y Selenium pueden coexistir en un modelo de pruebas y obtener resultados, ampliando la cantidad de funcionalidades disponibles en una prueba, permitiendo así interactuar y realizar pruebas.
Selenium	Si bien es posible realizar llamadas a puntos de acceso del ambiente, Selenium es una herramienta cuyo fuerte recae en la automatización de interacciones de un posible usuario. Por lo que no posee la capacidad de testear este ambiente.	Permite replicar cualquier tipo de interacción del usuario y repetirla la cantidad de veces requerida. Junto con lo anterior, es posible validar que estas acciones generen una respuesta visual acorde a lo realizado en la vista donde se encuentre. Es por lo anterior, que se afirma su aptitud para estas pruebas.
JMeter	Permite utilizar todos los métodos disponibles para consultar a un endpoint, validar y poner condiciones a la respuesta obtenida, por lo que JMeter posee todo lo necesario para realizar pruebas de este tipo.	La herramienta no cuenta con la capacidad de cargar visualmente un sitio, por lo que no permite realizar estas pruebas.
	Pruebas BackEnd (Rendimiento)	Pruebas FrontEnd (Rendimiento)
Cucumber	No posee la capacidad de realizar este tipo de testeo, dado que las pruebas se realizan utilizando un hilo y no generan carga para el ambiente.	Por los mismos motivos dados en el ambiente BackEnd, no posee la facultad de realizar pruebas de rendimiento en este.
Selenium	Como se explico más arriba, el utensilio no posee la capacidad de realizar pruebas de rendimiento en el ambiente Backend.	Para pruebas de rendimiento, la herramienta permite visualizar el sitio, junto con las funcionalidades necesarias para validar la correcta carga de este. Por esto, si posee la capacidad de realizar pruebas de rendimiento.
JMeter	Posee todo lo necesario para realizar este tipo de acción, tal como se observo en las pruebas realizadas.	La herramienta no cuenta con la capacidad de cargar visualmente un sitio, por lo que no permite realizar pruebas en FrontEnd.

Tabla 25: Tabla comparativa: Versatilidad de cada herramienta.
Fuente: Elaboración propia.

4.6.3. Documentación

	Fuentes de información	Calidad de la información
Cucumber	En su sitio oficial posee una sección específica a la herramienta, según el lenguaje utilizado. También es posible encontrar información en <i>Github</i> y en el sitio del gestor de paquetes de <i>NodeJS</i> (npmjs). Por otro lado existe una gran variedad de cursos gratuitos y certificaciones disponibles en sitios como <i>Udemy</i> .	Dentro del sitio oficial la información es casi nula, pero entrega un enlace a la web de NPM que posee más datos del modulo de <i>NodeJS</i> . La información de cada método se encuentra dispersa.
Selenium	Posee un sitio oficial con documentación relacionada a cada lenguaje. También es posible encontrar información dispersa en otros sitios, junto con certificaciones y cursos gratuitos.	La información entregada posee todos los métodos disponibles, explicados con un formato claro, indicando los parámetros de entrada y salida para cada uno de ellos.
JMeter	Posee un sitio del servidor Apache destinado a la herramienta. La información también abunda en otros sitios y cursos para todo nivel de conocimiento de la herramienta.	Se entrega información sobre todas las herramientas, dando casos de uso y buenas prácticas. La información se va entregando por pasos, divididos en secciones que incluyen las descripciones necesarias para cada funcionalidad disponible.
	Registro de actualizaciones	Ejemplos de uso
Cucumber	La sección del modulo en <i>NPM</i> entrega toda la información sobre actualizaciones y versiones, permite acceder a los repositorios de versiones antiguas.	En el repositorio del modulo se encuentran algunos ejemplos de uso, pero solo 1 para explicar su uso en <i>NodeJS</i> .
Selenium	Las actualizaciones del modulo poseen una sección específica, donde se indican las mejoras y cambios realizados en cada versión.	Los ejemplos entregados en el sitio son escasos con respecto a la cantidad de métodos que posee la herramienta, por lo que, en la mayoría de ellos, se debe buscar ejemplos de uso en fuentes externas.
JMeter	Posee una sección con todos los lanzamientos, incluyendo las notas de cada versión y los problemas corregidos.	Se explica el uso de todas las herramientas, incluye imágenes para facilitar el entendimiento y estructura. Lo anterior se lleva a cabo con imágenes y explicando los parámetros.

Tabla 26: Tabla comparativa: Documentación disponible de cada herramienta.

Fuente: Elaboración propia.

4.6.4. Escalabilidad

	Validaciones	Escenarios
Cucumber	Es posible añadir más validaciones dentro de cada paso, así como también implementar un nuevo paso que incluya una validación extra, el cual puede ser incluido dentro de cualquier escenario, permitiendo así incluir más validaciones sin necesidad de rehacer el escenario completo.	Los escenarios de prueba, como se detallaron anteriormente, pueden ser incluidos en el mismo modelo de pruebas de algún método, pudiendo también utilizar los pasos de otros escenarios según se estime conveniente.
Selenium	Facilita el añadir más validaciones en cualquier instancia del flujo de prueba, permitiendo verificar otros elementos del sitio. Cabe mencionar que las validaciones deben ser añadidas acorde al flujo, dado que si consulta por un elemento que pertenecía a otra vista, la prueba fallará en todas sus iteraciones.	Los archivos de prueba de Selenium no permiten abordar otros escenarios, a menos que se codifique al finalizar el anterior, pero esto reduce el entendimiento del código original.
JMeter	Permite añadir validaciones ilimitadas en cada prueba, pudiendo destinarlas a un paso en específico o que se aplique para cada proceso.	Los escenarios establecidos en JMeter no son escalables, dado que no permiten evaluar otros casos sin modificar su estructura interna, por lo que si se desea evaluar otro caso con una mínima diferencia se debe modificar todo el grupo de hilos o crear uno nuevo.

Tabla 27: Tabla comparativa: Escalabilidad de cada herramienta.
Fuente: Elaboración propia.

CAPÍTULO 5

CONCLUSIONES

A lo largo de esta sección, se expondrán las conclusiones surgidas tras el desarrollo, puesta en marcha de la propuesta de solución, resultados obtenidos y comportamientos identificados.

A pesar de la carencia de experiencia previa utilizando las herramientas evaluadas en este estudio, aplicar sus funcionalidades e implementar los escenarios de prueba ejecutados fue un proceso ágil que pudo ir de la mano con el desarrollo del sitio web. Cabe mencionar que la tarea de establecer los parámetros de entrada para cada prueba se asemeja en tiempo al dedicado en el desarrollo de la parte lógica de estas, dado que se buscó generar distintos resultados para detectar posibles fallos.

Uno de los aspectos importantes observado durante el proceso de construcción y puesta en marcha de las pruebas de software, fue la aplicación del concepto de mejora continua, dado que en cualquiera de sus etapas se fueron visualizando e incluyendo más características para evaluar, lo cual produjo que se realizaran más iteraciones en cada punto, efectuando ejecuciones con cada prototipo de prueba, llegando a finalizar este proceso con la obtención de un incremento en la calidad de las pruebas, abordando más a completitud el testing de cada funcionalidad. Por lo anterior, es importante destacar que aún cuando el plan de pruebas ya se encuentre definido, iterar sobre este durante su desarrollo o ejecución, permite visualizar puntos de mejora dentro de este.

Como consecuencia de la ejecución de planes de pruebas, fue posible vislumbrar las falencias de la aplicación desarrollada, por un lado se pudieron aplicar mejoras al código para mitigar errores que se detectaron en instancias iniciales de la implementación, mejorando la calidad y estabilidad del sistema. Así mismo, se realizaron cambios para optimizar el tiempo de respuesta de los métodos pertenecientes a la parte lógica del sistema. Por otro parte, las pruebas entregaron datos que permitieron identificar patrones de comportamientos acorde a los parámetros utilizados, gracias a esto fue factible realizar un análisis a más profundidad sobre las causales que provocaban ciertos resultados erráticos, tales como las caídas de sistema o los tiempos excesivos de cómputo.

De esto modo, se pudo indagar más en la extracción de información desde los resultados de las pruebas, permitiendo así otorgar mayor prioridad al procesamiento de los datos obtenidos, de manera que al procesarlo o darles una visualización, se pueda sacar el mayor provecho.

A partir del análisis realizado a los resultados obtenidos en las distintas pruebas, se puede concluir lo siguiente de los tipos de prueba implementados:

- Pruebas unitarias: La información entregada al finalizar el proceso de desarrollo y pruebas, no tiene un impacto mayor para el desarrollador, pero estas pruebas

si facilitan la detección de errores durante la implementación de los métodos, agregando valor al tiempo destinado a esta actividad, debido a que detectar este tipo de errores al trabajar con módulos acoplados puede tardar mucho más tiempo. Es por lo anterior que las pruebas unitarias añaden mucho más valor al ser desarrolladas y ejecutadas durante el proceso de desarrollo de funcionalidades.

- **Pruebas de integración:** Una vez realizadas las pruebas unitarias, se facilitó bastante la detección de errores, dado que solo podían surgir del método que estaba siendo instanciado en la ejecución de la prueba y no en los que eran ejecutados dentro de este. Estas pruebas permiten dar por finalizado el desarrollo de un punto de acceso, agilizando la puesta en marcha de este.
- **Pruebas de sistema:** Si bien los resultados de estas pruebas en un principio eran solo visuales, fue posible ahorrar bastante tiempo al trabajar en conjunto con el desarrollo del frontend, dado que solo bastaba con ejecutar un script que permitía preparar y probar cualquier cambio que se realizará, de manera que no era necesario llevar a cabo manualmente otras acciones para visualizar esto. Por otro lado, en una fase más avanzada de estas pruebas, fue factible observar y registrar más comportamientos del ambiente al realizar interacciones, dado que se implementó un registro que almacenaba todos los resultados como consecuencia de que se visualizaron comportamientos erráticos que no son posibles replicar manualmente dado que equivalen a la interacción de varios usuarios simultáneamente.
- **Pruebas de rendimiento:** Estas pruebas se volvieron indispensables para conocer los límites del sitio, ya que permiten agregar una barrera de seguridad para prevenir caídas del sistema.

Evaluando las herramientas por la dificultad que presentan para comprender su funcionamiento, es posible concluir, a partir de lo experimentado, que Cucumber es la opción más amigable para realizar pruebas en aplicaciones web. Esto se debe a que posee una estructura intuitiva que permite describir lo que ocurrirá en las pruebas, permitiendo insertar lógica a estas instrucciones.

Por otro lado, aunque no permite realizar pruebas con interacciones en el ambiente frontend, posee la factibilidad de testear todos los métodos disponibles, tanto para pruebas unitarias como para pruebas de integración, incluyendo también la posibilidad de simular pruebas de sistema hasta antes de mostrar la respuesta visual al usuario, validando la información de salida de cada interacción. Lo anterior mencionado quedó fuera del alcance de esta memoria, pero puede ser parte de futuras investigaciones y evaluaciones de herramientas de testeo enfocadas en un ambiente en particular.

JMeter presenta las mejores funcionalidades para evaluar el ambiente backend, dado que dispone de diversas utilidades que permiten inspeccionar cada aspecto necesario, pero presenta una dificultad al momento de realizar validaciones para pruebas unitarias, dado que no dispone de la capacidad de acceder al código, solo a los endpoints dispuestos por la aplicación. Para el caso de las pruebas de integración puede aplicar validaciones

de todo tipo a la respuesta, por ejemplo de código de estado, tiempo de espera, datos en la respuesta, entre otras.

Por lo mencionado anteriormente, JMeter es una herramienta que permite realizar tanto pruebas de rendimiento como pruebas de integración, pero entrega mejores resultados al momento de evaluar el rendimiento dado que posee un amplio catálogo de utilitarios para generar carga en el sistema y visualizar la información obtenida.

Las pruebas automatizadas facilitan el trabajo y testeo de una aplicación web, debido a que permiten la definición de un flujo de interacciones y la visualización en tiempo real de lo que ocurre con el sitio en cada una de ellas. Selenium es una herramienta eficaz para llevar a cabo estas pruebas, dado que permite simular toda interacción que pueda tener un cliente con la vista.

Por otro lado, se debe tener en cuenta que no se entregan datos de salida al realizar estos flujos automatizados, es por ello que al momento de evaluar si utilizar o no Selenium, se debe tener en cuenta lo siguiente:

- Estructura de la aplicación web: La aplicación o framework utilizado puede tener una reportería o registro que almacene los resultados de cada interacción. Por lo que, como se realiza en sitios de Ecommerce, las pruebas automatizadas se desarrollan solo con las interacciones necesarias y sin almacenar los resultados obtenidos.
- Finalidad de las pruebas: Se pueden evaluar distintos aspectos con pruebas automatizadas, estos definen el cómo se realizarán y que se espera de ellas. Por un lado, se puede requerir la evaluación de flujos de compra o de otras interacciones, en estos casos Selenium es óptimo porque solo se requiere finalizar el flujo, los resultados pueden ser visualizados por la consola o en el registro del sitio. Sin embargo, si se requiere almacenar resultados durante la ejecución o realizar pruebas de rendimiento, no es recomendable emplear la herramienta, dado que se debe realizar un desarrollo aparte solo para generar y almacenar la información, salvo para los casos que se tenga contabilizado el tiempo que tardaría este desarrollo.
- Tipo de respuesta esperada: El sitio puede responder de distintas maneras frente a una interacción del usuario, pero a grandes rasgos se pueden dividir en 2 grupos: respuestas visuales y respuestas lógicas. La primera de ellas corresponde a lo que el cliente puede ver al realizar una interacción, cambios de vista, despliegue de modal o alerta, etc. La otra corresponde a lo que ocurre por el lado de código, el almacenamiento de una variable, la llamada a un servicio, entre otras. Este factor es importante debido a que si se busca testear la respuesta lógica de una interacción, la herramienta no permite analizar el cambio que se pueda provocar, a menos claro que se almacene en una variable de sesión del navegador, siendo un punto a tener en cuenta al momento de elegir Selenium.

Para el caso de la evaluación de versatilidad de las herramientas se dio un foco apuntado

a la capacidad de realizar distintos tipos de pruebas en ambos ambientes. Este foco fue seleccionado debido a que permitía realizar una comparación más clara y directa de donde podían ser aplicadas estas herramientas. Se tuvo en consideración la posibilidad de revisar que tal versátiles eran las funcionalidades de cada una, pero se estimó que podía limitar lo que se iba a ver en los otros aspectos. Es por lo anterior, que el estudio se vio limitado en este aspecto y existe la posibilidad de profundizar mucho más, llegando a nivel de funcionalidades.

Como se presentó anteriormente, se estimó que la documentación de las herramientas añadía valor a estas, dado que la curva de aprendizaje depende en gran parte de la información y la calidad de esta, siendo máspreciada si se entrega con un formato entendible y con ejemplos que expliquen de manera correcta el como emplearlas.

Con respecto a los objetivos de este estudio, la evaluación de las herramientas de testeo fue llevada a cabo a completitud, permitiendo así observar su funcionamiento y los resultados que entregan, para evaluarlas bajo los distintos aspectos mencionados en la sección de validación de solución. Continuando con lo anterior, y a partir de las conclusiones ya entregadas, se puede afirmar que se cumplió el objetivo de proponer un modelo de selección, dado que se identificaron los parámetros a evaluar para elegir la herramienta más útil en cada instancia.

En cuanto al objetivo de definir en qué conjunto de tecnologías se desempeñan mejor las herramientas de testeo, considerando que el comportamiento y funcionalidades se mantienen aún para otros lenguajes y frameworks de desarrollo, es posible extrapolar que los aspectos analizados en la última parte de la sección de validación de solución permiten conocer el cómo se desempeñarán según las condiciones del sistema en el que se apliquen.

Este estudio se limitó a evaluar las herramientas elegidas sin realizar pruebas mezclando sus funcionalidades, si bien JMeter no es compatible con el resto, Cucumber y Selenium pueden ser utilizados en conjunto ampliando el espectro de validaciones y funcionalidades disponibles, como se mencionó en el cuadro comparativo dentro de la sección de validación. Por consiguiente, si fue posible cumplir en parte con el objetivo de evaluar la compatibilidad entre herramientas. Esto último puede ser objeto de otra investigación que continúe con lo realizado en este documento, evaluando la factibilidad de expandir los usos de las herramientas al complementarlas con otras.

Como se pudo visualizar en la evaluación de las herramientas, fue posible establecer aspectos para evaluar la calidad de las respuestas entregadas por cada una, comparando y estimando en que escenarios era posible emplearlas. De esta forma, fue posible cumplir en gran parte con el objetivo destinado a estimar la calidad de los resultados. Por otro lado, se evaluó la posibilidad de establecer nuevos puntos para entregar una estimación más precisa de cada herramienta, pero esto se vio limitado debido a que se evaluaron otras características.

Finalmente, con respecto a lo conseguido en este estudio, se puede concluir que se obtu-

vieron resultados con peso para la elección de herramientas, a la vez que se adquirieron conocimientos que serán útiles para emplear al momento de trabajar en desarrollo web. Así mismo, lo revisado a lo largo de este documento abre las puertas para continuar con la investigación, tanto para evaluar las herramientas como para realizar un estudio del impacto económico que puede llegar a tener el emplear correctamente estos planes de prueba.

ANEXOS

5.1. Enlaces a repositorios utilizados

A continuación se entregan los enlaces para acceder a los repositorios donde se almacenó todo lo desarrollado para llevar a cabo este estudio. Dentro de cada ambiente se incluyen también los archivos utilizados para las pruebas unitarias, de integración y de sistema, según corresponda.

- FrontEnd:
<https://gitlab.inf.utfsm.cl/dcarvaja/memoria-test-de-herramientas/-/tree/master>
- BackEnd:
<https://gitlab.inf.utfsm.cl/dcarvaja/memoria-test-de-herramientas-backend/-/tree/master>

REFERENCIAS BIBLIOGRÁFICAS

- [Argos, 2020] Argos (2020). Reasons behind the popularity of angularjs framework. *www.argosinfotech.com*.
- [channele2e, 2021] channele2e (2021). 5 key challenges of software testing. *www.channele2e.com*.
- [codingpotions, 2019] codingpotions (2019). Introducción a vue js. *codingpotions.com*.
- [eniun, 2021] eniun (2021). Metodología scrum para el desarrollo de software ágil. *www.eniun.com*.
- [Mireles, 2018] Mireles, G. A. G. (2018). Gestión de la calidad del software. *mat.uson.mx*, p. 1.
- [Phalcon, 2021] Phalcon (2021). Phalcon introduction. *docs.phalcon.io/*.
- [RedHat, 2020] RedHat (2020). ¿qué es una api de rest? *Integración*, 1(1).
- [Rodríguez, 2015] Rodríguez, Pedreira, F. (2015). Certificación de la mantenibilidad del producto software: Un caso práctico. *Revista Latinoamericana de Ingeniería de Software*, 3(3):127–134.
- [Spaven, 2017] Spaven, F. (2017). How much could software errors be costing your company? *DZone*, p. 1.
- [Spring, 2021] Spring (2021). Why spring? *spring.io/*.
- [Tricentis, 2018] Tricentis (2018). Software fail watch. *www.tricentis.com*, p. 1.
- [Umar, 2019] Umar, M. A. (2019). Comprehensive study of software testing: Categories, levels, techniques, and types. 5:32–40.