Investigación, implementación, ejecución y evaluación de pruebas unitarias en aplicaciones backend de VtexIo

Alejandro Montoya García

Proyecto presentado para optar al título de Ingeniero de Sistemas

Asesor

Sebastián Osorno Zapata, Ingeniero de sistemas

Docente

Robinson Coronado García, Especialista (Esp) en desarrollo de software

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín, Antioquia, Colombia

2024

Tabla de contenido

LISTA DE TABLAS

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

TDD.	Test driven development
BDD.	Behavior driven development
API	Application Programming Interface
CI	Continuous integration
CD	Continuous delivery

RESUMEN

En el mundo del desarrollo de software, la adopción de prácticas para asegurar la calidad del código y el producto final es cada vez más común, entre estas prácticas se encuentra la implementación de pruebas en el desarrollo, como las pruebas unitarias, siendo de las más adoptadas, ya que ayudan a comprobar que las funcionalidades desarrolladas se comporten según lo esperado; sin embargo, aún existen situaciones donde, por diferentes circunstancias, la aplicación de estas pruebas permanece ausente, tal es el caso de los desarrollos sobre la herramienta VtexIo, la plataforma de desarrollo provista por Vtex, una de las principales soluciones de e-commerce en Latinoamérica, la cual permite el desarrollo de aplicaciones alojadas en su infraestructura sin costo adicional.

En VtexIo, especialmente hablando de las aplicaciones de backend, no es común tener pruebas en el código, esto puede ser debido tanto a la falta de documentación para aplicar pruebas en este tipo aplicaciones, como a las restricciones propias de la herramienta, es por ello que el objetivo principal de este proyecto es investigar, implementar y ejecutar exitosamente pruebas unitarias en las aplicaciones backend desarrolladas sobre VtexIo. Este objetivo es desarrollado sobre una metodología iterativa, donde se prueba la implementación de 3 frameworks diferentes de pruebas unitarias, analizando en cada una las particularidades a tener en cuenta para su ejecución y las capacidades que brindan, para finalmente encontrar que es posible hacer uso de cada una aunque con diferentes niveles de dificultad en su implementación.

***Palabras clave* — Vtex, VtexIO, Pruebas Unitarias.**

ABSTRACT

In the world of software development, adoption of practices to ensure the quality of the code and its final product is increasing day by day, among these practices is the implementation of tests in the code, like unit testing, being one of the most accepted, since they help to check that developed functionalities act according as expected, however, there is still cases where, for many reasons, application of those test is absent, such is the case of developments on VtexIo, the development platform given by Vtex, one of the leading e-commerce solutions in latin america, which allows the development of apps hosted in its own infrastructure without additional cost.

In VtexIo, specially in backend applications, there are usually no tests in the code, this could be due the absence of documentation to apply test in this typeof application, as well as tool restrictions, it's because of that that this project main objective is carry out a research, implementation and successful execution of unit testing on backend application developed in VtexIo. This objective is developed on an interactive methodology, where the implementation of 3 different unit testing frameworks is attempted, analyzing the péculiarities to be taken in account and capabilities of each one, to finally find that each one is possible to use, however, with different levels of difficulty for its implementation.

Keywords **— Vtext, VtexIo, Unit Testing.**

I. INTRODUCCIÓN

VtexIo [1] es una plataforma de desarrollo provista por Vtex, una las principales soluciones de e-commerce en latinoamérica, con la cual se pueden desarrollar tanto apps de frontend como de backend sin costo adicional alojadas en la infraestructura de Vtex, además cuenta con varios builders [2], los cuales son instrucciones que habilitan diferentes funcionalidades para las aplicaciones, y estructuras predefinidas para distintas herramientas de desarrollo, como react en el caso de front y de Koa.js [3] en el caso del backend, también posee un cliente y servidor de graphql, sistemas de permisos para interactuar con Vtex, etc; lo que permite crear aplicaciones de forma rápida para su uso en el entorno de Vtex.

Sin embargo VtexIo también cuenta con particularidades propias que pueden dificultar labores que no están totalmente contempladas en la herramienta, algunas de estas es la restricción de las comunicaciones a puertos específicos, el uso de versiones fijas de paquetes que hoy en día pueden carecer de soporte, la imposibilidad en primera instancia de compilar y ejecutar la aplicación en un entorno local o la misma estructura anidada de los proyectos en la cual coexisten múltiples partes de una aplicación.

Aunque para las aplicaciones front end existe la documentación oficial para realizar distintos tipos de pruebas que permiten mantener distintas buenas prácticas de la industria como lo son los principios S.O.L.I.D [4], que buscan la creación de software mantenible y escalable; no hay mucha información respecto a la implementación y ejecución de pruebas para las aplicaciones de backend, de igual modo para la ejecución y evaluación de estas en entornos para integración continua como los son los pipelines; Con esto en cuenta cabe preguntarse ¿Cómo se puede implementar las pruebas unitarias Backend en VtexIo? ¿Cómo afectan las características de la herramienta en el desarrollo de pruebas unitarias? ¿Se pueden ejecutar las pruebas en un ambiente automatizado como lo es un pipeline de CI/CD?; con esto en mente, a continuación se tiene el objetivo de desarrollar las posibles vías de una implementación de pruebas unitarias con diferentes frameworks para las aplicaciones de backend en node.js desarrolladas sobre VtexIO, que sea replicable y además permita ejecutarse dentro de un pipeline teniendo en cuenta las limitaciones y particularidades que conlleva el desarrollo sobre VtexIO.

II. JUSTIFICACIÓN

En la actualidad la industria del desarrollo del software es cada vez más consciente de la necesidad de asegurar la calidad de las aplicaciones, para cumplir esta necesidad se han creado distintas prácticas en el proceso desarrollo, entre las que se encuentra la implementación de pruebas a diferentes niveles del desarrollo, sin embargo, aún existen muchas situaciones donde no implementan estas pruebas, hay distintos motivos para esta situación, como costumbres, o la falta de ellas, por parte del equipo de desarrollo, desconocimiento del cómo implementar las pruebas, dificultades dadas por el lenguaje o las herramientas con las que se desarrolla, etc. Los desarrollos sobre VtexIo actualmente se encuentra en esta situación, en especial las aplicaciones enfocadas al backend, donde casi no hay referencias para la implementación de pruebas y, presumiblemente, sumado a las particularidades propias de la herramienta, da como resultado que la gran mayoría de proyectos no cuentan con ningún tipo de prueba, o solo implementan las mismas a un nivel superficial. Es por esto que surge la necesidad de investigar cómo implementar las pruebas, en este caso unitarias, en el desarrollo de backend construidas sobre esta herramienta, conocer cuáles retos existen y qué limitaciones se pueden presentar en el proceso.

III. OBJETIVOS

A. Objetivo general

Realizar una investigación, implementación, ejecución y evaluación de pruebas unitarias en las aplicaciones backend de node js desarrolladas sobre VtexIo.

B. Objetivos específicos

- Investigar sobre las herramientas de pruebas unitarias que sean compatibles con los desarrollos backend en VtexIo.
- Realizar una implementación replicable de pruebas unitarias en una aplicación backend de VtexIo.
- Crear un pipeline para la ejecución y evaluación de cobertura de las pruebas unitarias para aplicaciones backend en vtexIo sobre la herramienta Github actions.

IV. MARCO TEÓRICO

A. Vtex

Vtex es una empresa de origen brasilleño fundada en el año 2000 dedicada a las actividades de e-commerce siendo su principal producto su suite de comercio ofrecida bajo el modelo de "software as a service" desde el 2010, su enfoque principal está dirigido a las ventas de minoristas, sin embargo, actualmente también se están potenciando las soluciones para las ventas mayoristas, el mayor atractivo de su suite de comercio es la flexibilidad para la personalización de las tiendas a través de diversas configuraciones en sus múltiples módulos y de la capacidad de tener desarrollos personalizados para necesidades específicas.

Actualmente ofrece sus servicios a grandes marcas como sony, Walmart, Whirlpool, Coca cola entre otros, teniendo más de 3400 tiendas activas en 38 países a través del mundo [5] .

B. VtexIO

Es un kit de herramientas de desarrollo de código abierto integrado con el ecosistema de Vtex, que permite crear soluciones personalizadas para e-commerce [1], ya sea nuevas experiencias de usuario o integraciones que expandan las funcionalidades del sitio donde se instalen.

VtexIo cuenta con herramientas para desarrollo de tanto de frontend como de backend, por el lado del front end permite el uso de react y frameworks relacionados, mientras que por el backend cuenta con alternativas de desarrollo con .NET y NodeJs [6] el cual toma como base a Koa.js [3], un web framework desarrollado por el equipo original de express, especializado en el desarrollo de aplicaciones ligeras que corren desde un servidor, este fue adaptado para funcionar sobre el lenguaje typescript [7]; así mismo permite exponer tanto api REST como de graphQl de consumo interno, también incluye como herramienta de desarrollo la implementación de policies para conceder permisos a las aplicaciones de interactuar con los módulos de Vtex.

Las aplicaciones desarrolladas en VtexIo son desplegadas en la infraestructura propia de vtex y se pueden publicar de forma privada para la instalación en las cuentas seleccionadas, o de forma pública por medio vtex App store, la tienda de aplicaciones de Vtex para sus tiendas, desde la cual se puede instalar en cualquier ecommerce de vtex las aplicaciones publicadas por la comunidad.

Adicionalmente cabe resaltar que estas aplicaciones están regidas por un sistema de builders [2] que habilita las distintas funcionalidades a las que puede acceder, como el ser reconocida como una aplicación backend de NodeJs o dar acceso al admin para exponer una nueva aplicación con interfaz propia, etc, aparte de habilitar las funcionalidades, los builders también traen definiciones de estas que pueden llegar a ser limitantes como la versión de NodeJs con la que se trabaja, provocando, por ejemplo, que no se pueda hacer uso de versiones recientes de librerías, incompatibilidad entre librerías internas y externas, y puede resultar en comportamientos inesperados.

Finalmente otra particularidad a resaltar al trabajar con VtexIo es que las aplicaciones no pueden ser ejecutadas de forma local, es decir, siempre que se ejecuta la aplicación esta es compilada y desplegada a la infraestructura de vtex donde trabajan bajo una arquitectura de microservicios en la cual cada una está autocontenida y puede funcionar de forma independiente, esto implica que siempre es necesario loguearse en una cuenta de vtex y tener la aplicación asociada a la misma para poder ejecutarla, incluso en entorno de desarrollo, el estar ligado a la infraestructura de vtex también provoca restricciones en las comunicaciones con sistemas externos teniendo solo ciertos puertos habilitados para estas comunicaciones, como el 443 para comunicaciones por medio de https, y dejando por fuera puertos como el 21 y 22 bloqueando la comunicación con servidores ftp, también se exige que las apis externas cuenten con certificados vigentes.

C. Pruebas unitarias

Tim Koomen y Martin Pol definen a las pruebas unitarias como “una prueba, ejecutada por el desarrollador en un entorno de laboratorio, la cual debe demostrar que el programa cumple los requerimientos en la especificación diseñada” [8], o en otras palabras es una prueba que se ejecuta en un entorno controlado, realizada sobre una pieza de software única e independiente bajo unas condiciones predeterminadas, con el objetivo verificar que esta cumpla los requisitos establecidos, además, como establece Robert C. Martin “las pruebas unitarias mantienen el código flexible, mantenible y reusable” [9] esto gracias a que cada cambio que se realice sobre el código está respaldado por una prueba que da la tranquilidad de estropear la funcionalidad actual, por esta razón se hace importante la inclusión de pruebas unitarias en los desarrollos de software.

A. Patrón AAA

El patrón AAA (Arrange, Act, Assert) [10], hace referencia a la estructura para una prueba unitaria, en el cual una prueba se divide en 3 secciones:

- Arrange: Sección donde se configura el entorno sobre el que se va a ejecutar la prueba, es decir, donde se crean los parámetros que van a ser usados en la función y se arreglan las dependencias para que se encuentren en el estado deseado para la prueba.
- Act: Consiste normalmente en la ejecución de la función evaluada y la toma del resultado de esta.
- Assert: corresponde a la sección donde se evalúa el resultado de la función por medio de funciones comparativas entre el resultado y lo esperado.

Este patrón sirve como estándar en la mayoría de pruebas unitarias y ayuda a la legibilidad de las mismas entre desarrolladores, además es ampliamente usado para la práctica de desarrollo dirigido por pruebas, o TDD por sus siglas en inglés, en la cual se escribe primero la prueba antes de desarrollar la lógica de la función.

B. Pruebas en VtexIO

VtexIO se puede encontrar implementaciones para distintos tipos de pruebas, entre ellas están las pruebas A/B [11], las cual se integran al flujo de desarrollo con una instalación de la aplicación correspondiente en la cuenta de vtex del ecommerce, y permite comparar el rendimiento de 2 versiones de un mismo componente, usualmente se toma la versión que se encuentra en productivo en este instante y la versión que se encuentra actualmente en desarrollo.

Además existe la librería VTEX Test Tools [12] en la cual se encuentran varias herramientas para realizar diferentes tipos de pruebas, la mayoría enfocados a los aspectos de front end, entre ellos están pruebas para el comportamiento de componentes y hooks de react, pruebas de integración para queries de graphQl, pruebas End to End para evaluar el flujo de la aplicación en escenarios predefinidos, los cuales simulan una interacción del usuario con la aplicación, haciendo uso de la librería cypress [13].

V. METODOLOGÍA

Para la realización de este proyecto se escoge la metodología design science [14], esta es una metodología de 6 etapas de carácter iterativo centrada en desarrollo investigativo, las etapas son identificación del problema y la motivación, la definición de los objetivos de la solución, el diseño y desarrollo de la solución, la demostración de lo desarrollado, la evaluación de resultados y finalmente la comunicación de los resultados, proponiendo que en las dos últimas etapas se pueda iterar a partir de la definición de objetivos o del diseño.

En este caso con las primeras 2 etapas ya definidas anteriormente, queda por desarrollar las etapas 3, 4 y 5, en las cuales primero se tomará un conjunto de herramientas para pruebas unitarias, en este caso se tomarán Jest [15], Mocha [16] y Jasmine [17] , y con cada una se intenta realizar una implementación de esta un proyecto de VtexIo bajo el patrón AAA y se analizan los resultados obtenidos teniendo en cuenta las siguientes medidas de carácter cualitativo: la facilidad de implementación, el soporte para mocks y el soporte de análisis de cobertura, y de carácter cuantitativo el tiempo de ejecución de las pruebas, tanto sin análisis de cobertura como con el mismo, y la cobertura lograda con cada herramienta bajo la misma suite de pruebas.

El proceso anterior será repetido hasta abarcar las 3 herramientas, tras lo cual se analizan los resultados encontrando las ventajas y desventajas de cada una para finalmente construir un pipeline de CI/CD en github actions [18] que permita ejecutar estas pruebas, esta decisión se toma teniendo en cuenta que github se encuentra el mayor contenido relacionado a desarrollos con VtexIo y existen antecedentes para esta actividad.

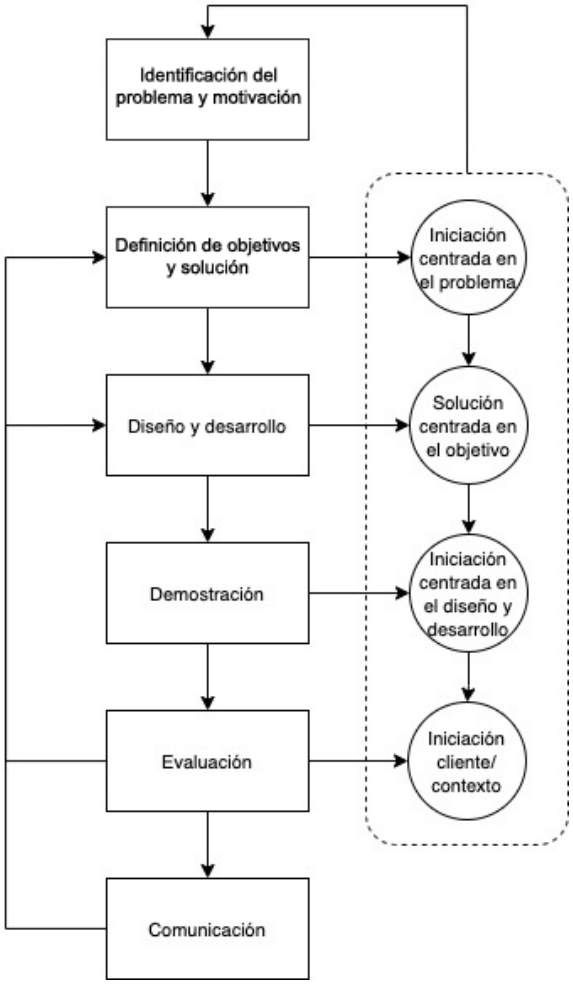

Fig 1 Metodología design science

VI. RESULTADOS

A continuación se presentan los resultados con cada una de las herramientas seleccionadas junto con hallazgos compartidos que aplican para todos los casos.

A. Jest

Jest es un framework de pruebas soportado por META, compatible con javascript y diferentes lenguajes frameworks basados en el mismo como NodeJs, typescript, babel, etc.

Jest permite la ejecución de pruebas en paralelo y en un proceso separado de la ejecución principal del programa, además cuenta con análisis de cobertura de código integrado, herramientas para mocking, soporte para llamados asíncronos y demás características para realización de pruebas unitarias.

Para su instalación no hay ningún conflicto con usar la última versión a las fecha, 29.7.0, por la cual al momento basta con instalar el paquete y sus respectivos tipos para typescript, adicional a esto hay que realizar la configuración para soporte de pruebas escritas en typescript, jest presenta 2 alternativas para esto, la librería ts-jest [19] y la librería babel [20], ts-jest suele ser la primera opción por ser la más completa y la que mejor soporte tiene en typescript, sin embargo, esta librería tiene un requerimiento mínimo de la versión typescript usada en el proyecto y es la versión 4.3, pero VtexIo usa la versión 3.9.7 y no es posible actualizarla por lo cual se desiste del uso de esta opción.

Por el lado de babel se puede usar la última versión, 7.23.7, sin inconvenientes por lo cual solo es instalar en el proyecto junto con el preset de typescript y el plugin de jest, para finalizar la configuración inicial sólo es ejecutar el comando jest o asignarlo a un script para ejecutar las pruebas.

En cuanto a la implementación de mocks se puede hacer uso de funcionalidad incorporada de jest, la mayor consideración es el uso de la propiedad _esModule para hacer mocks de clases de typescript.

Para el análisis de cobertura Jest cuenta con una versión de la librería istanbul [21] incorporada, lo que le da funcionalidades de cobertura de código en diferentes aspectos como número de líneas, ramificaciones, cantidad de funciones y declaraciones; esta funcionalidad

trabaja sin errores bajo el ecosistema de VtexIo por lo que se puede utilizar libremente añadiendo la opción –coverage al script de ejecución.

B. Mocha

Mocha es una librería para ejecución de pruebas unitarias que corre tanto sobre NodeJs como el navegador, también soporta la ejecución de tests en paralelo y llamados asíncronos, junto con otras características, sin embargo no cuenta con funciones de aserciones incluidas, por lo cual se requiere de una librería adicional que provea estas herramientas. Aquí entra Chai [22], una librería de aserciones para javascript enfocada al desarrollo con TDD o BDD compatible con la mayoría de frameworks para pruebas de javascript, esta cuenta con varios estilos de aserciones como should, expect y assert con diferentes sintaxis dependiendo del caso de uso o las preferencias del desarrollador.

Con la instalación de mocha se puede recurrir a la última versión estable, 10.2.0, con sus respectivos tipos, en cuanto a la instalación de chai hay que instalar concretamente la versión 4.3.9 o menor ya que después de esta versión chai utiliza una nueva declaración de módulos de ecmascript que no es reconocida por los proyectos en VtexIo, por lo cual toca recurrir a esta versión específica.

Otra consideración a tener en cuenta es el soporte para typescript, el cual se logra con el uso de ts-node [23], además de necesitar esm [24] para reconocer los módulos, esto conlleva a que ambos paquetes tengan que incluirse en el script de ejecución junto con un comando para configurar en variable de ambiente el uso de commonJs, el tipo de módulo base para typescript, para poder ejecutar las pruebas escritas sobre este lenguaje.

La última configuración a tener en cuenta, esta vez para no tener errores para la ejecución de la aplicación, es la actualización de la configuración del archivo tsconfig para añadir el tipo “mocha” y “node” a la lista de types.

En cuanto a la implementación de mocks la librería tampoco cuenta con esta funcionalidad integrada, gracias a esto hay que recurrir de nueva cuenta a una librería adicional, en este caso se recurre a sinon.js [25], librería especializada en mocks con distintas funcionalidades como fakes para modificar instancias de clase, stubs para modificar funcionalidades, spies para hacer seguimiento de los llamados y los mismos mock que funcionan como una combinación de los dos

últimos; actualmente no hay ningún conflicto con el uso de su última versión, 17.0.1, por lo cual se puede instalar y usar sin problemas.

Para el análisis de cobertura de nueva cuenta hay que recurrir a otra librería ya que no viene integrado en mocha, para mantener la consistencia se recurre a istanbul, misma librería que usa jest de base para su análisis de cobertura, esta librería puede ser usada en VtexIo en su última versión, 15.1.0, por lo cual basta con instalar la respectiva librería y agregar nyc al script de ejecución de pruebas.

C. Jasmine

Jasmine es otra alternativa para la ejecución de pruebas unitarias, su principal atractivo es estar construida 100% sobre javascript sin ninguna dependencia externa, maximizando su compatibilidad, cuenta con aserciones incluidas y ciertas funcionalidades para mocks, además es la librería de pruebas más usada con el framework angular.

La librería puede ser instalada en la última versión estable, 5.1.0, junto con sus tipos, y al igual que con Mocha, es necesario instalar ts-node para el reconocimiento de typescript y añadirlo al sript de ejecución junto con la configuración de variable de entorno para usar commonJs.

Otra similitud con mocha es la necesidad explícita de agregar los tipos “jasmine” y “node” a la propiedad types del archivo tsconfig.

Para terminar de configurar la ejecución de jasmine hay que crear un archivo de configuración de tipo json con la configuración para typescript y referenciarlo en el script de ejecución.

Para la integración de mocks, Jasmine cuenta con una funcionalidad para este objetivo, los espíes, estos permiten tanto hacer seguimiento de los llamados como modificar las funcionalidades, en este caso para su uso no hay que tener consideraciones adicionales y se pueden usar según lo sugiere la documentación oficial.

Respecto al análisis de cobertura Jasmine no cuenta con esta funcionalidad integrada, de modo que es necesario recurrir a una librería externa de nueva cuenta, dado que istanbul también es compatible con esta herramienta de pruebas se recurre nuevamente a esta opción con la seguridad que no presenta conflictos con VtexIo, por lo mismo el proceso de integración es el mismo, instalar la librería y añadir nyc al script de ejecución.

D. Hallazgos comunes

Al intentar realizar las pruebas en VtexIo surgen ciertos retos propios de la estructura que se maneja par las aplicaciones backend y se resumen en los siguientes puntos:

- Simular el contexto: Como ya se había mencionado, el framework sobre el que se basa el desarrollo de vtexio en aplicaciones backend es koa.js, el cual recibe como parámetros en sus servicios un objeto de tipo contexto, el cual también es implementado en vtex io pero extendido con otras funcionalidades, al usar typescript, los métodos que hacen uso del contexto esperan recibir el tipo específico, sin embargo, este objeto de contexto no es creado explícitamente en el código no hay forma de extraerlo para pasarle una instancia de ejemplo al método en el test y debido a su extensión se dificulta el crearlo manualmente, por ello, para poder proveer este contexto a los métodos que hacen uso de él, es necesario crear un objeto con todas las propiedades que serán usadas en el método a evaluar, y asignar el tipo “any” para que sea aceptado por el método sin importar la restricción de tipo.
- Uso de clientes: del punto anterior se derivan 2 retos adicionales, el primero es el uso de clientes, los cuales básicamente son los encargados de realizar llamados a sistemas externos, ya que estos también hacen uso del contexto pero de propiedades que no son triviales de instanciar para que sean simuladas, además, dada su característica de interactuar con sistemas externos, las funciones declaradas en los clientes no deberían efectuarse de verdad en las pruebas unitarias. Por ello se pasa a ser necesario que la herramienta de pruebas provea o soporte librerías para crear servicios mock, de esta forma es posible enviar la instancia del cliente al contexto que recibe el método a probar sin preocuparse por cómo se tiene que ejecutar, ya que esta será reemplazada por el mock generado en las pruebas.
    - Simulación del cuerpo en servicios REST: el segundo reto derivado es el envió del cuerpo de una petición a los métodos utilizados para los servicios REST, ya que estos son codificados de la petición en un objeto de tipo incomingMessage [26] y decodificados en el método, para poder simular la entrada de este objeto se crea una clase nueva que funcionará como sustituto de este nuevo tipo, esta clase se crea extendiendo de la clase

transform de NodeJs [27], la cual comparte propiedades con incomingMessage, y se complementa agregando las propiedades propias de una petición http.

Adicional a esto, aún queda el proceso de ejecución en un pipeline, para ello se toma como base el github action Vtex IO Test Action [28], el cual cuenta con la capacidad de correr el script básico para pruebas unitarias sin verse afectado por la estructura de carpetas propia de Vtex IO, sin embargo cuando se intenta ejecutar pruebas con análisis de cobertura dispara un error, por lo que es necesario cambiar la aproximación que tiene esta implementación, para ello se crea una nueva versión cambiando la base de ejecución de docker a JavaScript, y agregando la posibilidad de indicar cuál es el script que se desea ejecutar, con esta nueva versión es posible ejecutar las pruebas con las 3 herramientas tanto con cómo sin análisis de cobertura y para asegurar un mínimo de cobertura es necesario agregar la instrucción para esta tarea en la configuración de istanbul o jest con el mínimo porcentaje esperado.

E. Ponderación de resultados

En la tabla I se enseña un resumen de los resultados según los criterios definidos en la sección metodología, los tiempos promedios son el resultado promedio de 10 ejecuciones de la misma cantidad de pruebas sobre la misma máquina.

TABLA I RESULTADOS DE LA IMPLEMENTACIÓN

Criterio	Jest	Mocha	Jasmine
Proceso de implementación	● Requiere de la instalación de babel. ● Necesita un archivo de configuración para babel.	● Requiere de la instalación de ts-node y esm. ● Necesita de un archivo de configuración. ● Requiere de una configuración adicional de typescript.	● Requiere de la instalación de ts-node. ● Necesita de un archivo de configuración.

		• Necesita de una librería adicional para las aserciones.	
Soporte de Mocks	Posee la funcionalidad integrada.	No posee la funcionalidad integrada, soporta librerías adicionales para esta función.	Posee la funcionalidad integrada.
Soporte de análisis de cobertura	Posee la funcionalidad integrada.	No posee la funcionalidad integrada, soporta librerías adicionales para esta función.	Posee la funcionalidad integrada.
Tiempo promedio de ejecución de pruebas en segundos	24.72	56.54	50.27
Tiempo promedio de ejecución de pruebas con cobertura en segundos	37.72	72.18	58.70
Porcentaje de cobertura: • Lines • Branch • Functions • Statements	90 58.33 100 90	97.91 80.76 100 97.91	97.91 92.3 100 97.91

VII. DISCUSIÓN

Al analizar los resultados de implementar las herramientas para pruebas se puede decir en primera instancia que Jest es la mejor opción de las 3 gracias a que tiene la menor cantidad de pasos para su implementación junto con Jasmine, trae integrado las funcionalidades de mocks y de análisis de cobertura y además tiene el menor tiempo de ejecución en los 2 escenarios, no obstante, la medición de cobertura deja ver que el uso de las otras opciones sigue siendo válido según el caso, por ejemplo, aunque en un principio mocha luzca como la peor opción al ser la más difícil de implementar y tener menos funcionalidades integradas, posee el índice de cobertura más alto de los 3, debido justamente a la necesidad de librerías externas, lo cual le da acceso a funcionalidades avanzadas de esas librerías especiales, como la posibilidad de tener mocks más dinámicos que ayudan a probar más escenarios.

Otro punto de interés son consideraciones que fueron necesarias para realizar la implementación, aunque efectivamente surgieron restricciones a causa del uso de VtexIo como la imposibilidad de usar ts-jest en lugar de babel, o la necesidad de usar una versión anterior de chai, aun así, es cierto que la mayoría, en principio, son heredados del uso de typescript o del framework base koa.js.

Para finalizar el análisis el reto para ejecutar las pruebas unitarias en una ambiente automatizado con el pipeline de CI/CD no radica tanto en el uso de una herramienta de pruebas u otra, si no en la estructura de carpetas del proyecto y en el pipeline en sí.

VIII. CONCLUSIONES

Fue posible implementar el uso de pruebas unitarias en desarrollos backend de VtexIo con múltiples herramientas, aunque con diferentes niveles de dificultades para lograrlo. De igual forma también fue posible ejecutar las pruebas en un pipeline de github actions siendo independiente de la herramienta de pruebas.

Las restricciones que se esperaban de usar de VtexIo parecieron, pero cabe resaltar que varias son heredadas de las tecnologías de las que deriva este entorno de desarrollo.

La posibilidad de implementar pruebas unitarias en las aplicaciones Backend de VtexIo desde el primer nivel donde se reciben las peticiones permite tener más confianza en el código construido.

Si bien Vtex no provee mucha documentación ni ejemplos de implementación de pruebas para aplicaciones de backend, la existencia de herramientas como Vtex IO Test Action refleja la intención de abordar este tema, pero aún falta profundización.

Este proyecto puede tomar como punto de partida para mejorar las prácticas en los desarrollos sobre el entorno de VtexIo, cómo implementar pruebas de integración, despliegue continuo por medio de un pipeline, etc.

IX. TRABAJOS FUTUROS

A partir de este proyecto queda pendiente evaluar el impacto que tendrá la aplicación de las pruebas unitarias en la calidad del software desarrollado y la tasa de errores reportados en un futuro.

Además se pueden desprender varias líneas de trabajo futuras, la primera es la implementación de otros tipos de pruebas, como las end to end, para tener más alcance al asegurar la calidad del código.

Otra oportunidad se encuentra en los pipelines de CI/CD, donde se puede continuar a partir de lo construido de este proyecto y agregar procesos de despliegue continuo, evaluación de otros tipos de pruebas, análisis estático de código, entre otros.

REFERENCIAS

[1] Vtext team, «Crea aplicaciones de comercio personalizadas con la plataforma ... - VTEX,» Vtex, [En línea]. Available: https://vtex.com/co-es/vtex-io/.

[2] Vtex Team, «Builders,» Vtex, [En línea]. Available: https://developers.vtex.com/docs/guides/vtex-io-documentation-builders.

[3] KoaJs, «Koa - next generation web framework for node.js,» [En línea]. Available: https://koajs.com/.

[4] M. Noback, Principles of package design: Creating reusable software components, 2018.

[5] Vtex Team, «Somos expertos en comercio digital,» [En línea]. Available: https://vtex.com/co-es/about-us/.

[6] Node.js, «About,» [En línea]. Available: https://nodejs.org/en/about.

[7] Typescript, «TypeScript is JavaScript with syntax for types,» Microsoft, [En línea]. Available: https://www.typescriptlang.org.

[8] T. Koomen y M. Pol, Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing, 1999.

[9] R. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2009.

[10] V. Khorikov, «Using the AAA pattern,» de *Unit testing: Principles, practices, and patterns*, Manning, 2020, pp. 42-43.

[11] Vtex Team, «Running A/B tests,» Vtex, [En línea]. Available: https://developers.vtex.com/docs/guides/vtex-io-documentation-running-native-ab-testing.

[12] l. v, «Vtex/test-tools: Add tests to your VTEX Io app in an instant,» GitHub, [En línea]. Available: https://github.com/vtex/test-tools. [Último acceso: 21 09 2023].

[13] «JavaScript component testing and E2E testing framework,» Cypress, [En línea]. Available: https://www.cypress.io/. [Último acceso: 24 09 2023].

[14] J. V. Brocke, A. Hevner y A. Maedche, de *Introduction to Design Science Research*, 2020, pp. 1-13.

[15] Jest, «Jest · Delightful JavaScript Testing,» Meta, [En línea]. Available: https://jestjs.io.

[16] Mocha, «Mocha - the fun, simple, flexible JavaScript test framework,» [En línea]. Available: https://mochajs.org.

[17] Jasmine, «Jasmine Documentation,» [En línea]. Available: https://jasmine.github.io.

[18] «Features • github actions,» Github, [En línea]. Available: https://github.com/features/actions.

[19] K. Kabra, G. Wengel y H. Gandon , «ts-jest,» [En línea]. Available: https://www.npmjs.com/package/ts-jest.

[20] «Babel · Babel,» Babel, [En línea]. Available: https://babeljs.io.

[21] «Istanbul, a JavaScript test coverage tool.,» Istanbul, [En línea]. Available: https://istanbul.js.org.

[22] «Chai Assertion Library,» Chai, [En línea]. Available: https://www.chaijs.com.

[23] «ts-node,» ts-node, [En línea]. Available: https://typestrong.org/ts-node/.

[24] J. D. Dalton, «esm,» [En línea]. Available: https://www.npmjs.com/package/esm.

[25] Sinon committers, «Standalone test spies, stubs and mocks for JavaScript.,» [En línea]. Available: https://sinonjs.org.

[26] «HTTP | Node.js v21.6.1 documentation,» Node.js, [En línea]. Available: https://github.com/vtex-apps/broadcaster.

[27] «Stream | Node.js v21.6.1 documentation,» Node.js, [En línea]. Available: https://nodejs.org/api/stream.html#duplex-and-transform-streams.

[28] «Vtex IO Test Action,» Vtex, [En línea]. Available: https://github.com/marketplace/actions/vtex-io-test-action.