Implementación de la fase de pruebas para Frontend en una plataforma cognitiva

Angie Castañeda Martínez

Trabajo de grado presentado para optar al título de Ingeniero de Sistemas

Asesores

Leonardo Augusto Pachón Contreras, Doctor (PhD) en Física

Martín Elías Quintero, Ingeniero de Sistemas

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín, Antioquia, Colombia

2021

Cita	(Castañeda Martínez A, 2021)
Referencia **Estilo APA 7 (2020)**	Castañeda Martínez A. (2021). *Implementación de la fase de pruebas para Frontend en una plataforma cognitiva* [Trabajo de grado profesional]. Universidad de Antioquia, Medellín, Colombia.

Tabla de contenido

Lista de figuras

Siglas, acrónimos y abreviaturas

PhD	Doctorado
UdeA	Universidad de Antioquia
E2E	End to end
JS	Javascript
DOM	Document Object Model
API	Application Programming Interface
TDD	Test Driven Development

Resumen

El desarrollo de software es cada vez más demandante en términos de funcionalidades y carga de usuarios; por tanto, es crucial construirlo de forma que garantice una alta calidad, estabilidad y escalabilidad futura. En este contexto, uno de los retos principales es aumentar la automatización y reducir la intervención humana, siendo las pruebas de software uno de los aspectos responsables de ello. Tener una alta cobertura de pruebas en cada artefacto de software ayuda a prevenir muchos errores en la etapa de producción del sistema, además de garantizar la calidad y estabilidad del software.

Hoy en día, cada vez son más las empresas y los productos que requieren la implementación de estas pruebas, como es el caso de guane Enterprises, donde, entre los proyectos que más destacan, se encuentra un asistente cognitivo que centraliza, gestiona y administra diferentes tipos de solicitudes que llegan a varias empresas de logística a través del correo electrónico.

Este trabajo presenta el estudio e implementación de pruebas de software *Front-end* para este sistema cognitivo, esto con el fin de mejorar su calidad en el momento de su puesta en producción. Dentro del alcance de este trabajo se propone la búsqueda de las diferentes tecnologías existentes para realizar pruebas de software que se adapten a este sistema, así como la implementación de una de estas tecnologías y el planteamiento del desarrollo del código de pruebas de software.

Palabras clave: Desarrollo de software, Pruebas de software, Tecnologías *Front-end*, Calidad de software.

Abstract

Software development is becoming increasingly competitive, so building it in a way that guarantees high quality and future stability plays a crucial role in its success. This becomes a challenge since one of the needs of software development is to increase the automation and to reduce human intervention, and software testing is one of the aspects responsible for this. Having a high test coverage in each project, helps to prevent several errors in the production stage of the system, as well as guaranteeing quality and stability.

Nowadays, there are more companies and products that require the implementation of software testing, as is the case of Guane Enterprises, where, among the projects that stand out the most, is the cognitive assistant that centralizes and manages different types of incoming requests to multiple logistics companies through email.

This document presents the study and implementation of *Front-end* software testing for the cognitive assistant, in order to improve its quality for its release date. The scope of this document is the search of different existing technologies for software testing that adjust to the project, the implementation of one of them and the development approach of software testing code.

*Keywords***:** Software development, Software testing, *Front-end* technologies, Software Quality.

Introducción

Las grandes apuestas para la creación de software moderno se centran en aumentar la automatización y reducir la intervención humana en las fases del denominado ciclo de vida del software. La gran ventaja de este enfoque es el aumento de la productividad al delegar en los sistemas la responsabilidad de ejecutar tareas indispensables para llevar los productos de software de la etapa de desarrollo a la de producción. Sin embargo, garantizar la disponibilidad y mantenibilidad de estos sistemas es un reto cuando intervienen tantos componentes diferentes en cada iteración del mismo ciclo.

Entre los pasos más importantes para garantizar la disponibilidad de los servicios se encuentra la responsabilidad de construir artefactos de software de alta calidad y estables en el tiempo. No obstante, dicha responsabilidad también depende del diseño y desarrollo de pruebas de software que definen una importante trazabilidad camino a la calidad (IBM, s.f). Normalmente, las pruebas se ejecutan en función de la temporalidad en la que se encuentra el desarrollo, i.e., inicialmente los desarrolladores construyen pruebas directamente sobre las características del software, que posteriormente se complementan con pruebas de ofuscación robustas, como las pruebas de carga y caos, garantizando así la estabilidad y la resistencia del producto de software. En todos los casos, las pruebas enfocan la atención en la automatización, la disponibilidad de los artefactos y la estandarización en producción. Para realizar estas pruebas de software actualmente existen diferentes tecnologías que nos facilitan su implementación. El mecanismo para elegir la herramienta más adecuada está relacionado con el nivel de complejidad y robustez que el mismo producto o artefacto de software tenga.

Cada vez son más los productos y las empresas que requieren de un correcto diseño, implementación y manejo de pruebas de desarrollo de software. Entre los proyectos llevados a cabo en guane Enterprises, destaca un asistente cognitivo que centraliza, gestiona y administra diferentes tipos de solicitudes que llegan de diversas empresas de logística (*cargo*) a través del correo electrónico. La plataforma hace uso de algoritmos de inteligencia artificial para direccionar, agilizar, extraer y estructurar la información de las solicitudes de los clientes, impactando directamente en sus operaciones.

El presente proyecto es de gran importancia debido a que la inclusión de las fases de prueba en un producto afecta directamente las fases de desarrollo y posteriormente su puesta en

producción, garantizando así su correcto funcionamiento. Particularmente, se hará uso de un tipo de prueba específico llamado E2E usando Cypress que destaca por su facilidad de uso y su rápida implementación (Cypress.Io, s.f).

Planteamiento del problema

Asegurar la calidad de un producto de software se ha convertido en algo necesario para la industria moderna como uno de los principales retos industriales, que implica una necesidad en términos de innovación, estabilidad y garantía. Es bastante común con el avance tecnológico que una aplicación tenga intrínsecamente una arquitectura con muchos componentes que se comunican al tiempo. Esto hace que los procesos de producción sean cada vez más complejos y que exista un error implícito entre el puente de los entornos de desarrollo y producción.

En concreto, dentro del marco de los asistentes cognitivos desarrollados por guane Enterprises se encuentra CharlieBot que es un cliente de correo electrónico para un sector específico. Internamente, CharlieBot tiene una multitud de servicios que están desacoplados y con una única responsabilidad; no obstante, las visualizaciones siguen siendo un solo artefacto. A medida que la complejidad aumenta, desencadena la necesidad de añadir procesos de integración y despliegue continuo; sin embargo, la falta de estrategias de pruebas y *Test Driven Development* (TDD) hace que el sistema tenga una deuda técnica asociada a estas prácticas. Para ello, y antes a la construcción de estas fases intermedias, se debe garantizar el correcto funcionamiento de la aplicación mediante el uso de pruebas de software.

Objetivos

3.1 Objetivo general

Implementar pruebas *front-end* en el sistema de gestión de correo electrónico de la plataforma cognitiva “CharlieBot”.

3.2 Objetivos específicos

3.2.1 Estudiar y seleccionar las mejores herramientas para las pruebas de desarrollo.

3.2.2 Integrar y ejecutar pruebas en las vistas del sistema.

3.2.3 Evaluar la relevancia de los marcos en el equipo de desarrollo de la empresa.

Marco teórico

Organizar y estructurar la lógica en los grandes proyectos de software es un reto que escala rápidamente a con el nivel de complejidad del desarrollo. Entre los enfoques que existen para mitigar esto está el llamado Diseño Dirigido por el Dominio (DDD), que es un enfoque de desarrollo de software adecuado para proyectos donde el dominio del software es complejo. Este enfoque centra el desarrollo en el modelo de dominio y en las reglas de negocio, de forma que haya una alta comprensión de estos y sea más fácil organizar la lógica en la implementación (Fowler, 2020).

Uno de los conceptos introducidos en el DDD es el de Lenguaje Ubicuo, que trata de resolver el problema de la comunicación entre los expertos del dominio y los desarrolladores creando un lenguaje común entre ellos (Henao, 2019). La metodología hace hincapié en la implementación de este vocabulario en el software, para que esté presente durante todo el desarrollo y la vida del producto (Fowler, 2020).

Este lenguaje unificado se estructura continuamente entre los desarrolladores y los equipos de producto, y luego se inicia un proceso de validación para garantizar la calidad del software verificando que cada componente y artefacto está correctamente diseñado. La idea principal de la validación es establecer la dirección de los requisitos funcionales expuestos durante las primeras etapas del desarrollo. Este proceso se denomina fase de pruebas y es independiente del esquema del ciclo de vida del software elegido. Estas pruebas suelen elaborarse por un desarrollador o un equipo especializado en el diseño de pruebas, normalmente conocido como *Tester*. Una de las ventajas de esta fase es la incorporación de prácticas en los proyectos que ayudan a prevenir errores, mejorar el rendimiento y cerrar la brecha de posibles deudas técnicas. Por otro lado, al ser una fase más del proceso de desarrollo, aumenta los tiempos de desarrollo del proyecto, impactando en las estimaciones de los equipos que desarrollan las reglas de negocio y en las opiniones del cliente (IBM, s.f).

Uno de los hemisferios tecnológicos más importantes y de mayor crecimiento en los últimos años es el de las tecnologías del lado del cliente. Esta área de desarrollo ha sido bautizada como *Front-end*, que es la parte del diseño de software que interactúa cara a cara con los usuarios. Es un elemento muy importante para los productos ya que para el usuario lo que ve en la pantalla, la interfaz de usuario es lo más significativo, por lo que asegurarse de que esta

interfaz gráfica funcione correctamente es realmente esencial para el éxito de la solución (Testim, 2019).

Dentro de las tecnologías actuales para *Front-end* se encuentra Vue.Js que es un *framework* progresivo para crear interfaces de usuario (Vue.js, s.f). Esta tecnología se usa por su simplicidad y adaptabilidad que permite integrar fácilmente con otras tecnologías y otros proyectos existentes (Vue.js, s.f).

Con base en lo anterior, *DDD* y testing, *Test-Driven Development* (*TDD*) es una metodología para el testing de software, en la que primero se piensa en el componente que se quiere desarrollar y se realizan las pruebas para el mismo, para posteriormente desarrollar un software que cumpla con las pruebas previamente realizadas (Fowler, 2005).

Hay muchas herramientas disponibles para las pruebas de *Front-end*, pero también es necesario entender qué conjunto de pruebas de software es el más adecuado para los proyectos. Entre todos los tipos de pruebas que existen, los tres más utilizados son las pruebas unitarias (UT), las pruebas de componentes y las pruebas End-To-End (E2E) (Vue.js, s.f).

El primer tipo de pruebas, las UT, nos permite comprobar fragmentos de código aislados y que son responsabilidad de cada desarrollador ejecutarlos para asegurarse de que su código es estable (Vue.js, s.f). Algunas herramientas que se pueden utilizar y que han ido ganando relevancia son Jest y Mocha, la primera es un *framework* que se centra en la simplicidad, por lo que no hay que hacer una gran configuración para utilizarlo. Además, realiza pruebas aisladas tomando instantáneas que ofrecen un método alternativo de revisión de las mismas (Jest, s.f). Mocha, por su parte, se caracteriza por ser flexible, permitiendo incluir diferentes funcionalidades para evaluar una variedad de características dando la posibilidad de ejecutarse tanto en Node.js como directamente en el navegador (Mocha, s.f).

Las pruebas de componentes se encargan de probar los componentes de la aplicación una vez que se ensamblan en el DOM. Es común en el desarrollo el uso de bibliotecas que ayudan a probar los componentes de la interfaz de usuario de las aplicaciones, centrándose en el usuario (Vue.js, s.f). Los proyectos que proporcionan un ecosistema de este tipo se denominan como *Testing Library*, que es un conjunto de *Application Programming Interfaces* (APIs) que funcionan de manera versátil en muchos *frameworks* web, como Angular, React y Vue. *Testing Library* puede simular un entorno que manipula la API-DOM utilizando Jest, Mocha o un navegador web (Krolick, 2021). Una alternativa a la anteriormente mencionada es Vue-Test-Utils

que es la biblioteca oficial de utilidades de pruebas para Vue.js. Su objetivo principal es proporcionar un conjunto de elementos de uso común para las ejecuciones de validación (Vue-Test-Utils, s.f).

Finalmente, las pruebas E2E, son las más completas, debido a su versatilidad para probar todas las capas de nuestra aplicación una vez que ésta es consumida (Vue.Js, s.f). La tendencia es usar software que se realice sobre infraestructura en la nube a fin de garantizar los recursos suficientes que requieren los comprobadores de código. En ese sentido *Cypress* ofrece un conjunto de herramientas de pruebas modernas, que se ha caracterizado por ser de uso rápido debido a su arquitectura, funcionar con la gran mayoría de los *frameworks* de Javascript y habilitar las pruebas de básicamente cualquier elemento que se ejecute en el navegador. Además, está construido de tal manera que el TDD puede ser implementado fácil y rápidamente (Cypress.Io, s.f). Algunos otros que compiten a la par de Cypress son Nightwatch.js (Nightwatch.js, s.f), Puppeteer (Puppeteer, s.f) y TestCafe (Testcafe, s.f).

Una vez que se ha integrado la fase de pruebas de un proyecto, hay otro proceso muy utilizado hoy en día, que es la integración y el despliegue continuo (CD/CI). Se trata de una serie de pasos que deben ejecutarse cuando se va a lanzar una nueva versión del software (RedHat, s.f). Si un proyecto carece de una fase de pruebas, difícilmente se puede garantizar su estabilidad en un entorno de producción, por lo que las pruebas son un paso fundamental para desarrollar software moderno.

Metodología

La *figura 1* muestra el desarrollo de este proyecto en términos de sus pasos metodológicos.

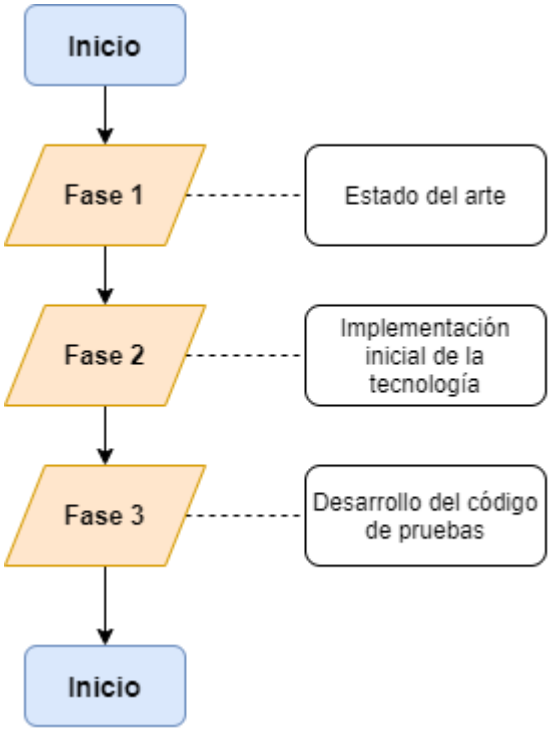

Figura 1. *Flujo de trabajo*

Fase 1: Estado del arte

Para cualquier proyecto es esencial comenzar con una adecuada recopilación de información, de manera que primero se establece el estado del arte para hacer un balance de las tecnologías más apropiadas que deben incluirse en el proyecto. En este estado del arte, se encontraron muchas tecnologías para cada uno de los tipos de pruebas, que son pruebas unitarias, pruebas de componentes, y pruebas E2E. El proyecto CharlieBot, al ser un proyecto de gran envergadura y haber comenzado ya su desarrollo, lo óptimo es incluir una fase de pruebas que sea robusta pero fácil de implementar.

Las pruebas más utilizadas son las unitarias y las E2E, con la diferencia de que las unitarias se utilizan para probar pequeñas partes del código, mientras que las E2E prueban grandes partes de la aplicación, como funcionalidades completas. Otra de las ventajas de las E2E

es que interactúan directamente con la interfaz de usuario y no con el código, lo que garantiza que la interfaz con la que interactúa el usuario funcione correctamente.

Debido a que este proyecto ya está en una fase avanzada, las pruebas unitarias son complejas y llevan mucho tiempo de implementación por tener que realizar pequeñas pruebas para cada pequeña parte del código, por lo que se decidió implementar las pruebas E2E. Una vez seleccionado el tipo de prueba que se va a realizar, se decidió qué tecnología se va a utilizar para ejecutar estas pruebas, en función de la información recopilada inicialmente y de la arquitectura actual del sistema.

De las tecnologías para realizar pruebas E2E, una de las más utilizadas hoy en día es Cypress debido a sus amplias funcionalidad que ayudan al desarrollador a realizar estas pruebas de manera más completa, como lo es el *Time travel*, *Debuggability*, *Automatic Waiting*, entre otros (Cypres.Io, s.f), y al ser una tecnología ampliamente referenciada cuenta con una buena documentación.

Fase 2: Implementación inicial de la tecnología

A partir de la revisión bibliográfica obtenida en la fase anterior, se realiza una primera aproximación al desarrollo de las pruebas, integrando la tecnología seleccionada al sistema. Para iniciar esta fase se utilizó la documentación oficial de Cypress

Una de las ventajas de Cypress es que tiene una buena compatibilidad con los principales *frameworks* de *Front-end*, como lo es Vue.Js, y dispone de documentación oficial para implementarlo, lo que facilita la integración inicial de Cypress con el sistema en producción (Miller, 2021). Además, no es necesario implementarlo directamente en el proyecto principal, sino que se puede instalar tanto un proyecto separado que se centre solo en pruebas, como en el proyecto principal, dependiendo de la necesidad del sistema. Esto es así porque a la hora de realizar las pruebas de Cypress, estas se ejecutan en una URL o dirección web que proporcionamos al principio del código, que por defecto se abrirá en una ventana de Chrome y ejecutará las pruebas de la página que se proporcione.

El proyecto principal se gestiona con un controlador de versiones llamado Git, que facilita la realización de cambios de código en el mismo por diferentes personas (Git, s.f). Por esta razón, las pruebas de Cypress se implementan en una rama del proyecto principal utilizando Git.

Para comenzar la implementación, se instala Cypress dentro de la rama de pruebas designada del proyecto principal utilizando un sistema de gestión de paquetes (Npm, s.f), mediante el comando

npm install --save-dev cypress ,

en una terminal Linux.

Una vez instalado, los componentes de Cypress se inicializan utilizando el comando

./node_modules/.bin/cypress open,

que realiza automáticamente la configuración básica necesaria para las pruebas. Una vez creada, la estructura inicial de los archivos de Cypress se genera como se muestra en la Figura 2.

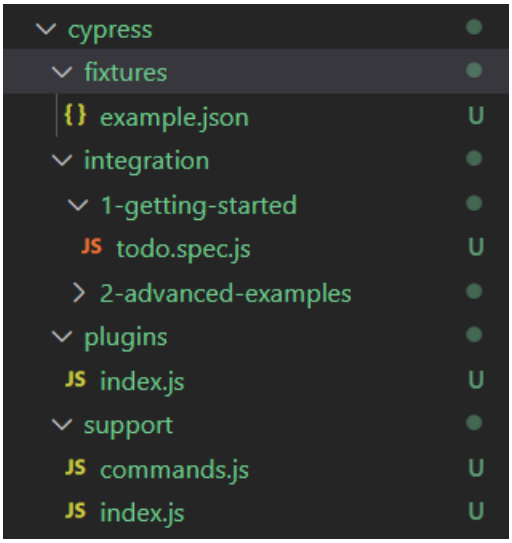

Figura 2. *Distribución de archivos iniciales*

Finalmente, una vez configurado Cypress en el proyecto, se deciden los módulos del sistema que se van a probar inicialmente y se realiza un primer diseño de la distribución de archivos para estas pruebas. En esta primera aproximación, los archivos de prueba se ven como en la Figura 3 y Figura 4.

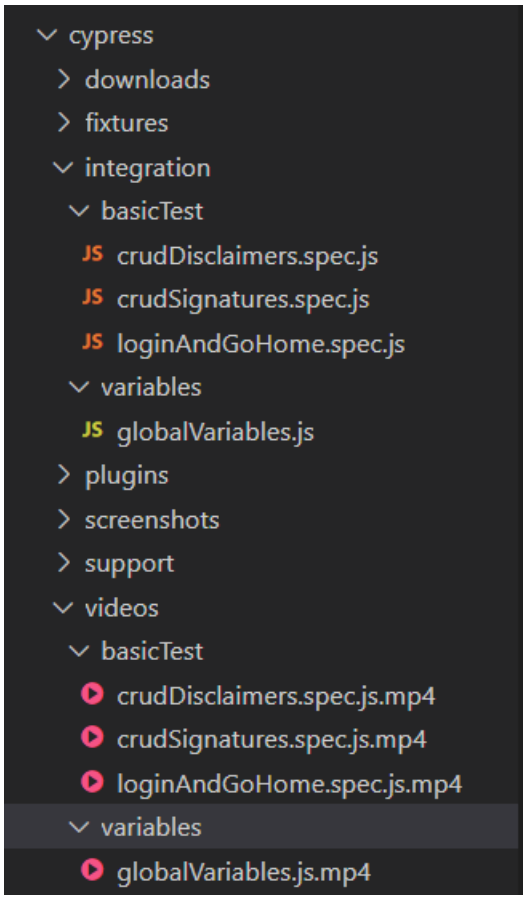

Figura 3. *Distribución de archivos para las pruebas*

```
import pageUrl from "../variables/globalVariables";
/// <reference types="cypress" />

context('Window', () => {
  it('Open app and login', () => {
    Cypress.on('uncaught:exception', (err, runnable) => {
      // returning false here prevents Cypress from
      // failing the test
      return false
    })
    cy.visit(pageUrl)
    cy.get('[id^=username]').type('user1')
    cy.get('[id^=password]').type('password1')
    cy.get('.login').click()
    cy.wait(10000)
    cy.get('.content').click().should('be.visible')
    cy.get('.sidebar').click().should('be.visible')
    cy.get('.emails').click().should('be.visible')
  })
})
```

Figura 4. *Código inicial de las pruebas*

Fase 3: Desarrollo del código de pruebas

La última fase de las pruebas es el desarrollo del código de prueba, específicamente para los módulos y funcionalidades previamente definidos. Con el fin de realizar las pruebas por funcionalidad, se creó un archivo para cada una de ellas. Para una primera aproximación, se crearon pruebas para 3 funcionalidades específicas, *Signatures, Disclaimers y Login*. Además de estos 3 archivos donde se desarrolló el código de prueba para estas funcionalidades, existe un archivo donde se incluirán las variables a utilizar en las pruebas, como la URL del sitio a probar. Una de las ventajas de Cypress es que almacena capturas de pantalla y videos de las pruebas que se ejecutan desde el corredor de pruebas de Cypress, lo que ayuda a tener un historial del desempeño de estas pruebas. Esto se almacena en dos carpetas por defecto llamadas capturas de pantalla y vídeos.

Resultados

Luego de tener todo el ambiente de Cypress configurado, así como haber desarrollado el código de las pruebas iniciales del sistema cognitivo, se pueden visualizar estas a través del test runner de Cypress en la Figura 5.

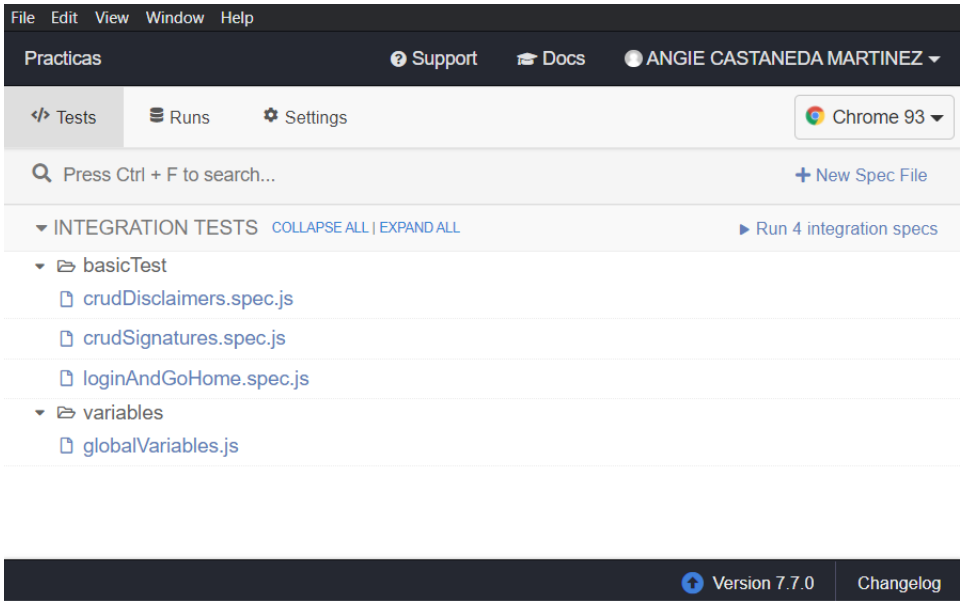

Figura 5. *Test runner de Cypress*

Estas pruebas iniciales comprueban todo el funcionamiento del CRUD (Create, Read, Update, Delete) de los disclaimers y las signatures, además del login, de forma que, al desplegar una actualización del sistema en producción, se ejecutan las pruebas para confirmar que no hay ningún error en estas funcionalidades.

Adicionalmente, Cypress ofrece un dashboard en su página web, en el que, al entrar desde el test runner, podemos guardar una grabación de las pruebas, así como abrir la página web del dashboard de Cypress y ver las estadísticas del rendimiento de dichas pruebas. Esto se puede ver en la Figura 6.

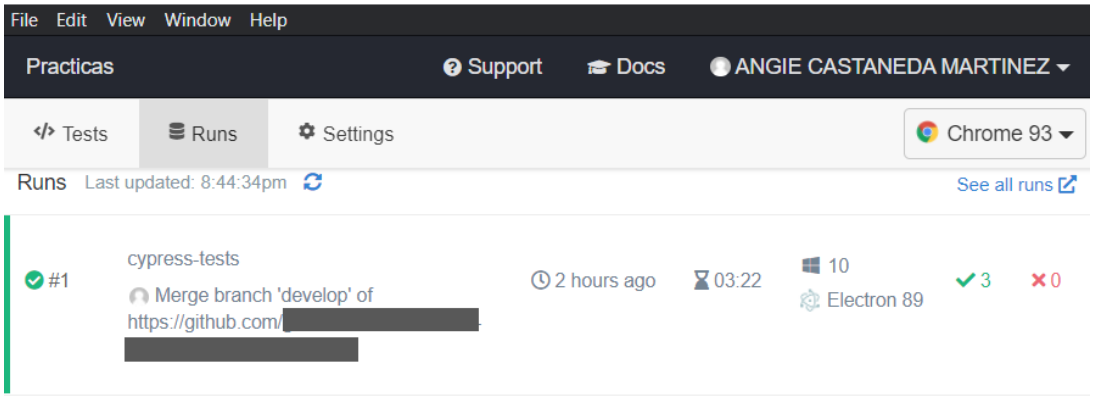

Figura 6. *Pruebas guardadas en el test runner*

En las Figuras 7 y 8 podemos ver un resumen de las pruebas, así como las estadísticas de rendimiento de las mismas, que, para las pruebas inicialmente implementadas, fueron todas exitosas.

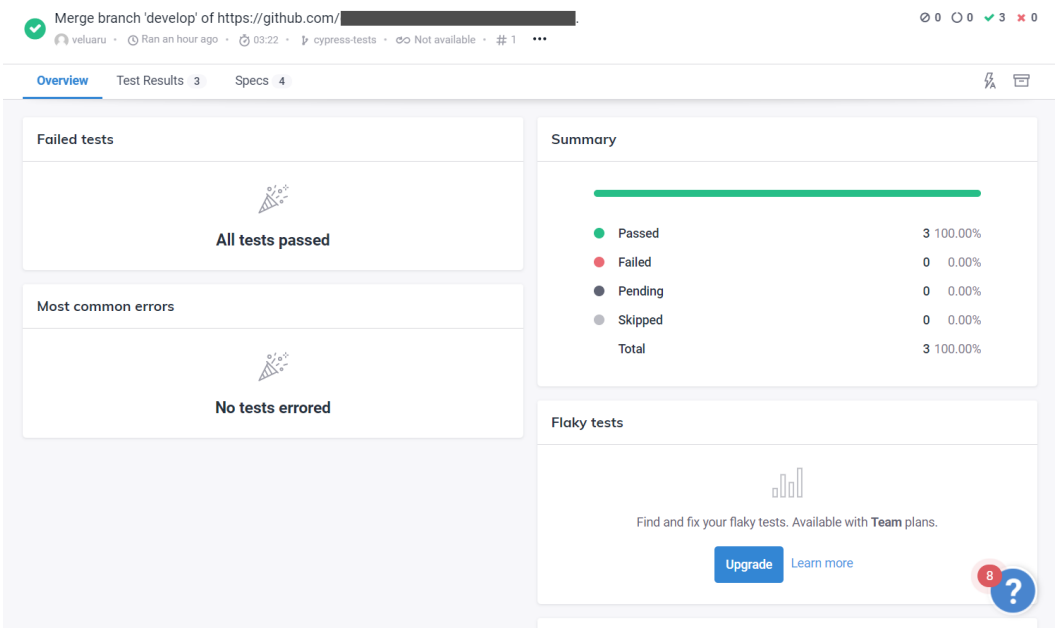

Figura 7. *Resumen de pruebas dashboard de Cypress*

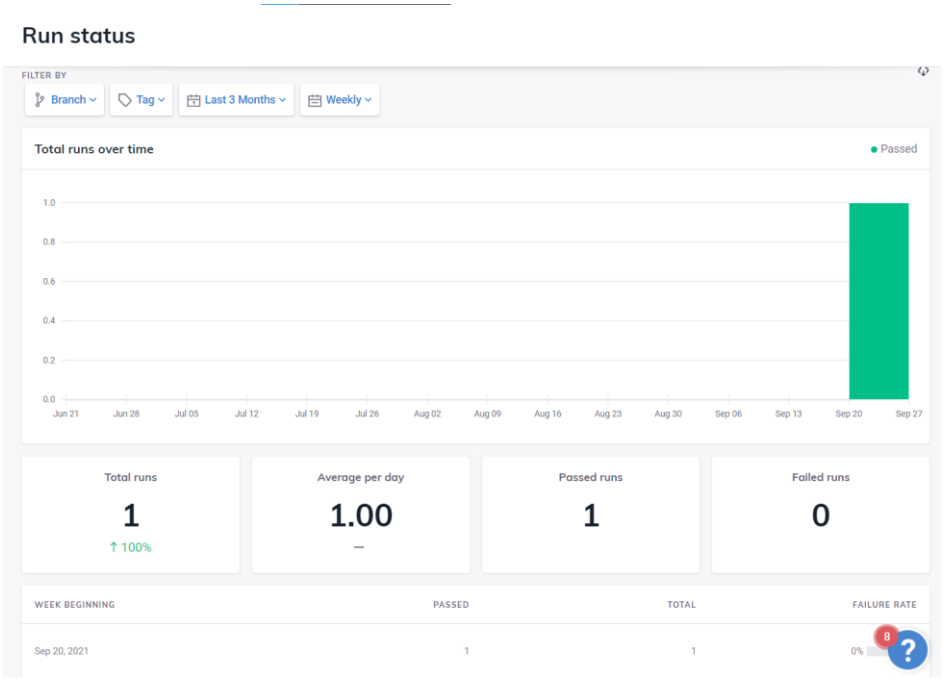

Figura 8. *Estadísticas dashboard de Cypress*

Conclusiones

Hoy en día, con las grandes aplicaciones es cada vez más necesario implementar fases de pruebas de software, con el fin de garantizar un producto final de mayor calidad. En concreto, las pruebas End to End son muy útiles porque prueban cada una de las funcionalidades completas del sistema, para asegurar que el flujo de trabajo del usuario funciona correctamente. Cypress es una herramienta que nos proporciona una forma muy fácil y rápida de implementar este tipo de pruebas ofrecen muchas funcionalidades que facilitan la visualización de las estadísticas de estas pruebas. Además, hoy en día, con las grandes aplicaciones se hace cada vez más necesario implementar este tipo de pruebas de software, de manera que podamos asegurar un producto final de mayor calidad.

En concreto, las pruebas End to End son muy útiles porque prueban cada una de las funcionalidades completas del sistema, para asegurar que el flujo de trabajo del usuario funciona correctamente. Cypress es una herramienta que nos proporciona una forma muy fácil y rápida de implementar este tipo de pruebas, además de ofrecer muchas características que facilitan la visualización de las estadísticas de estas pruebas. Además, al acortar el tiempo dedicado a la fase de implementación y generación de estadísticas, se puede dedicar más tiempo a la fase de desarrollo de código de las pruebas para que sean más completas y cubran más rápidamente la mayor parte de la funcionalidad del sistema. Al acortar el tiempo dedicado a la fase de implementación y a la generación de estadísticas, se puede dedicar más tiempo a la fase de desarrollo del código de las pruebas para que sean más completas y cubran más rápidamente la mayor parte de la funcionalidad del sistema.

Trabajo Futuro

Las pruebas de software han demostrado ser una de las mejores para garantizar la calidad de los artefactos digitales. En ese sentido, se propone como trabajo futuro desarrollar todo tipo de pruebas orientadas al desarrollo utilizando la técnica TDD y DDD. Posteriormente, se busca incluir toda esta metodología en los respectivos flujos de trabajo para la integración y el despliegue continuos, así como las pruebas automatizadas guiadas por inteligencia artificial.

Referencias

Fowler, M. (2020, 22 abril). *DomainDrivenDesign*. martinfowler.com. https://martinfowler.com/bliki/DomainDrivenDesign.html

Henao, W. B. (2019, 26 junio). *Conoce el diseño de Software, orientado al modelo de negocio*. pragma.com.co. https://www.pragma.com.co/academia/lecciones/conoce-el-diseno-de-software-orientado-al-modelo-de-negocio

What is software testing? (s. f.). IBM. Recuperado 13 de junio de 2021, de https://www.ibm.com/topics/software-testing

Front End Testing: A Complete Conceptual Overview. (2019, 11 diciembre). Testim. https://www.testim.io/blog/front-end-testing-complete-overview/

Introduction a Vue.js. (s. f.). Vue.Js. Recuperado 2 de octubre de 2021, de https://vuejs.org/v2/guide/

Fowler, M. (2005, 5 marzo). *TestDrivenDevelopment*. martinfowler.com. https://martinfowler.com/bliki/TestDrivenDevelopment.html

Testing Vue.js. (s. f.). Vuejs. Recuperado 13 de junio de 2021, de https://vuejs.org/v2/guide/testing.html

Jest. (s. f.). Jest. Recuperado 13 de junio de 2021, de https://jestjs.io/

Mocha - the fun, simple, flexible JavaScript test framework. (s. f.). Mocha Js. Recuperado 13 de junio de 2021, de https://mochajs.org/

Krolick, A. (2021, 13 enero). *Introduction | Testing Library*. Testing-Library. https://testing-library.com/docs/

Introduction | Vue Test Utils. (s. f.). Vue-Test-Utils. Recuperado 14 de junio de 2021, de https://vue-test-utils.vuejs.org/

End to End Testing Framework. (s. f.). Cypress.Io. Recuperado 14 de junio de 2021, de https://www.cypress.io/how-it-works/

Node.js powered End-to-End testing framework. (s. f.). Nightwatch.js. Recuperado 14 de junio de 2021, de https://nightwatchjs.org/

Puppeteer. (s. f.). Puppeteer. Recuperado 13 de junio de 2021, de https://pptr.dev/

Why TestCafe? (s. f.). Testcafe. Recuperado 13 de junio de 2021, de https://testcafe.io/documentation/402631/why-testcafe

¿Qué es un canal de CI/CD? (s. f.). RedHat. Recuperado 13 de junio de 2021, de https://www.redhat.com/es/topics/devops/what-cicd-pipeline

Why Cypress? (2021). Cypres.Io. https://docs.cypress.io/guides/overview/why-cypress#Our-mission

Miller, L. (2021, 6 abril). *Getting Started with Cypress Component Testing (Vue 2/3)*. Cypress Blog. https://www.cypress.io/blog/2021/04/06/getting-start-with-cypress-component-testing-vue-2-3/

Git--everything-is-local. (s. f.). Git. Recuperado 1 de octubre de 2021, de https://git-scm.com/

npm. (s. f.). Npm. Recuperado 1 de octubre de 2021, de https://www.npmjs.com/