UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

FACULTAD DE INGENIERÍA

PRÁCTICA PROFESIONAL

HELLO ICONIC

PREVIO A LA OBTENCIÓN DEL TÍTULO

INGENIERO EN SISTEMAS COMPUTACIONALES

PRESENTADO POR:

11741447 NOHELIA CRISTINA EUCEDA FIGUEROA

ASESOR: LIC. TANIA LUCILA MEZA

CAMPUS TEGUCIGALPA; ABRIL, 2021

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

LISTA DE SIGLAS Y GLOSARIO

API Application Programing Interface

BDD Behavior Driven Development

CEO Chief Executive Officer

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

QA Quality Assurance

TDD Test Driven Development

TMR Tiempo Medio de Reparación

UI User Interface

W3C World Wide Web Consortium

RESUMEN EJECUTIVO

El siguiente informe es la muestra del trabajo realizado en la empresa Hello Iconic, donde se tuvo como finalidad añadir calidad, innovación y diseño a dos proyectos: University of Maryland y Neil Young Archives.

Lo que hoy es Hello Iconic comenzó hace muchos años, sumando las mentes de dos personas: Alejandro Corpeño, un creativo, que desde la universidad ya demostraba un lado emprendedor y Jorge García, un visionario que tomó el salto al pasar de docente y director de carrera universitaria a probar suerte en el mundo de las pequeñas empresas de tecnología para así seguir el sueño de tener algo propio. Ahora la visión de la empresa es crear productos digitales que sean escalables desde su etapa inicial hasta ser productos robustos con interfaces usables e integraciones potentes. Se trabaja en equipo todos los días, haciendo, planificando, analizando, diseñando, creando prototipos, lanzando y haciendo crecer productos icónicos.

La contribución de trabajo fue hecha en su totalidad en el departamento de Aseguramiento de la Calidad, donde ésta se encarga de brindar servicios de calidad a todos los proyectos que se desarrollan en la empresa. La meta principal del trabajo realizado fue añadir valor a los productos donde estos estuviesen libres de defectos o haciendo estos defectos mínimos en el producto. Algunas de las asignaciones fueron hacer pruebas exploratorias, creación de matrices de prueba, creación de casos de prueba, realización de pruebas funcionales y no funcionales, diseño y creación de pruebas automatizadas, creación de pruebas para API, entre otras. Utilizando las tecnologías para comprobar la calidad como ser TestRail, WebDriverIO, Appium, Azure DevOps y Postman.

Esta práctica profesional tuvo su duración desde octubre de 2020 hasta abril de 2021 donde se realizó el mayor esfuerzo para ejercitar lo aprendido, investigar lo desconocido y entregar la mejor calidad en el trabajo realizado.

I. INTRODUCCIÓN

Desde la creación del ser humano hasta la actualidad la humanidad se ha visto en un constante enfrentamiento a problemas en su diario vivir, de los cuales debe de encontrar e implementar su debida solución. Uno de los primeros problemas a los que se enfrentó el humano fue la necesidad de encontrar calor donde pronto se dieron cuenta que frotando dos rocas producían una chispa y añadiéndole un poco de paja podían crear una fogata para poder mantener el calor corporal y también podían cocinar su comida. Así que se puede ver que desde tiempos muy antiguos la mujer y el hombre han desarrollado múltiples soluciones para sus necesidades o problemas sencillos y también complejos. Pero cada solución viene de la mano con sus respectivas pruebas para asegurar seguridad, calidad, mantenibilidad y comodidad.

Alva Edison, creador de la bombilla incandescente, se encontró con la necesidad de crear un tipo de candela que fuese duradero y seguro. Para poder realizarlo, Alva tuvo que hacer más de mil diferentes prototipos hasta lograr que los filamentos en su interior funcionaran de la mejor manera y con la mayor seguridad.

Junto con el origen de la Tercera Revolución Industrial llegan los inicios de la tecnología donde se encuentran los primeros ordenadores. El hardware y software son una combinación de soluciones para problemas, necesidades e innovaciones y cada rubro en todas las industrias se ven en la obligación de hacer uso de ellas. Al principio el éxito del funcionamiento apropiado de toda máquina se basó en el método de prueba y error, sin duda alguna sigue siendo un método funcional.

El departamento de calidad en una empresa de desarrollo de software ha tomado importancia cada día más y las competencias de un ingeniero de calidad de software incluyen programación, atención al detalle, habilidades de comunicación y trabajo en equipo.

Dentro de los valores de Hello Iconic está la calidad y su meta siempre es crear y diseñar productos con la mejor calidad posible para sus clientes. Hello Iconic se centra en ofrecer productos de vanguardia como ser aplicaciones móviles, aplicaciones para televisores inteligentes y también

páginas y portales web. Se busca ofrecer soluciones escalables, robustas y con una buena experiencia de usuario.

En el presente informe se detalla información general de la empresa, datos del departamento de Calidad, objetivos generales y específicos, descripción y marco teórico de los trabajos realizados.

II. GENERALIDADES DE LA EMPRESA

2.1 DESCRIPCIÓN DE LA EMPRESA

Hello Iconic nació en el 2012 con la unión del actual Director Ejecutivo (CEO) Alejandro Corpeño y el Director Tecnológico Jorge García con el desarrollo de una primera aplicación móvil creada desde un garaje de casa, pero con 6 millones de descargas. Empezó a ganar popularidad después de su segunda aplicación móvil creada y se expande su equipo a dos oficinas, una en Tegucigalpa, Honduras y otra en California, Estados Unidos.

Hello Iconic es proveedor de servicios de desarrollo de aplicaciones móviles, aplicaciones para televisores inteligentes, páginas web y portales de estudiantes. Brinda soluciones escalables, robustos y con una buena experiencia de usuario. (Elvir, 2019)

2.2 DESCRIPCIÓN DEL DEPARTAMENTO

El departamento de Quality Assurance (QA) es el responsable de asegurar la calidad de los productos creados en Hello Iconic. El equipo está dividido entre ingenieros responsables de diseñar y desarrollar pruebas automatizadas e ingenieros audaces para hacer pruebas no funcionales manuales y también ingenieros que hacen ambos trabajos.

2.3 OBJETIVOS DEL PUESTO

2.3.1 OBJETIVO GENERAL

Monitorear por medio de pruebas funcionales y no funcionales que los requerimientos establecidos por el cliente sean satisfechos dándole seguimiento a los requerimientos técnicos establecidos por el equipo de arquitectura para así poder entregar un producto con eficiencia y calidad a los clientes durante seis meses.

2.3.2 OBJETIVOS ESPECÍFICOS

- Participar en sesiones de entrenamiento para afianzar el conocimiento técnico y comprender el producto y sus características.

- Diseñar y desarrollar pruebas automatizadas de interfaz de usuario en WebDriverIO y pruebas automatizadas de API para monitorear las funcionalidades del producto.
- Crear matrices de prueba al igual que sus casos de prueba con la finalidad de tener una guía del comportamiento del producto en futuras pruebas.
- Realizar pruebas exploratorias, de aceptación, regresión, sanidad y de usabilidad para garantizar la estabilidad del producto en cada una de las fases de desarrollo.
- Monitorear el producto en busca de defectos con el fin de mantener la calidad de la aplicación.

III. MARCO TEÓRICO

La adopción de un sistema de gestión de la calidad es una decisión estratégica para una organización que le puede ayudar a mejorar su desempeño global y proporcionar una base sólida para las iniciativas de desarrollo sostenible. El pensamiento basado en riesgos permite a una organización determinar los factores que podrían causar que sus procesos y su sistema de gestión de la calidad se desvíen de los resultados planificados, para poner en marcha controles preventivos para minimizar los efectos negativos y maximizar el uso de las oportunidades a medida que surjan. (ISO 9001:2015 Sistemas de gestión de la calidad, 2015)

El cumplimiento permanente de los requisitos y la consideración constante de las necesidades y expectativas futuras, representa un desafío para las organizaciones en un entorno cada vez más dinámico y complejo. Para lograr estos objetivos, la organización podría considerar necesario adoptar diversas formas de mejora además de la corrección y la mejora continua, tales como el cambio abrupto, la innovación y la reorganización. (ISO 9001:2015 Sistemas de gestión de la calidad, 2015)

La norma ISO 9001:2015 (2015) menciona los principios de la gestión de la calidad:

- Enfoque al cliente
- Liderazgo
- Compromiso de las personas
- Enfoque a procesos
- Mejora
- Toma de decisiones basada en la evidencia
- Gestión de las relaciones

La prueba del sistema es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora. Aunque cada prueba tenga un propósito diferente, todo él funciona para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas. (Pressman, 2010)

Según Pressman (2010) las pruebas se dividen en funcionales y no funcionales, dentro de las pruebas no funcionales se encuentran:

- La recuperación es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada. Si la recuperación es automática (realizada por el sistema en sí), se evalúa el reinicio, los mecanismos de puntos de verificación, la recuperación de datos y la reanudación para correcciones. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación (TMR) para determinar si está dentro de límites aceptables.
- La prueba de seguridad intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia.
- La prueba de esfuerzo ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales.
- La prueba de rendimiento se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado.
- La prueba de despliegue, en ocasiones llamada prueba de configuración, ejercita el software en cada entorno en el que debe operar.

También según Pressman (2015) se encuentran en las pruebas funcionales:

- La prueba de interfaz ejercita los mecanismos de interacción y valida los aspectos estéticos de la interfaz de usuario. La estrategia global para la prueba de interfaz es descubrir errores relacionados con mecanismos de interfaz y descubrir errores en la forma como la interfaz implanta la semántica de navegación, la funcionalidad de la aplicación o el despliegue de contenido Evalúa cuán bien cuida el diseño a los usuarios, ofrece instrucciones claras, entrega retroalimentación y mantiene consistencia de lenguaje y enfoque.
- La prueba de usabilidad es similar a la de semántica de interfaz porque también evalúa el grado en el cual los usuarios pueden interactuar efectivamente con la aplicación y el grado en el que la aplicación guía las acciones del usuario, proporciona retroalimentación significativa y refuerza un enfoque de interacción consistente.
- La prueba de compatibilidad es definir un conjunto de configuraciones de cómputo, y sus variantes, que se encuentran comúnmente en el lado cliente.

- La prueba en el nivel de componente, también llamada prueba de función, se enfoca en un conjunto de pruebas que intentan descubrir errores en funciones de las aplicaciones.
- Las pruebas de navegación se incluyen para asegurarse de que cada interfaz realiza la función que se le ha encargado.

En la Ilustración 1 se muestra una priorización de pruebas y sus tipos.

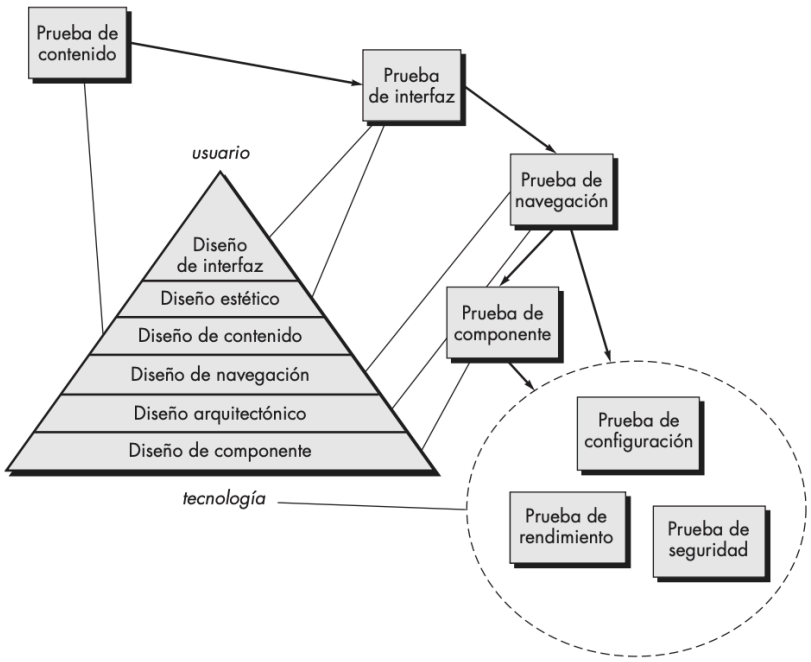


Ilustración 1 – Proceso de pruebas y sus tipos

Fuente: (Pressman, 2010)

De acuerdo con el Estándar del Institute of Electrical and Electronics Engineers (IEEE) 610 (1990) un caso de prueba se define como: "Un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico."

Según el libro Introducción a las Pruebas de Sistemas de Información (Rodríguez, 2014) un caso de prueba debe incluir varios elementos en su definición, y entre los elementos que destacan se encuentran:

- Flujo: secuencia de pasos a ejecutar

- Datos de entrada
- Estado inicial
- Valor de respuesta esperado
- Estado final esperado

Hablar de caso de prueba lleva a pasar a hablar del concepto de oráculo. Básicamente es el mecanismo, ya sea manual o automático, de verificar si el comportamiento del sistema es el deseado o no. Para esto, el oráculo deberá comparar el valor esperado contra valor obtenido, el estado final esperado con el estado final alcanzado, el tiempo de respuesta aceptable con el tiempo de respuesta obtenido, etc. (Rodríguez, 2014)

Muchos casos de prueba están escritos en lenguaje Gherking. Gherking es un lenguaje comprensible por humanos y por ordenadores. Se trata de un lenguaje fácil de leer, fácil de entender y fácil de escribir. (Hernández, 2016)

Hernández (2016) muestra las palabras con las que se construyen las sentencias que describen las funcionalidades:

- Característica: indica el nombre de la funcionalidad que se va a probar. Debe ser un título claro y explícito. Se incluye aquí una descripción en forma de historia de usuario: "Como [rol] quiero [característica] para que [los beneficios]". Sobre esta descripción se comienzan a construir los escenarios de prueba.
- Escenario: describe cada escenario que se prueba.
- Dado que: provee contexto para el escenario en que se va a ejecutar la prueba, tales como punto donde se ejecuta la prueba, o prerrequisitos en los datos. Incluye los pasos necesarios para poner al sistema en el estado que se desea probar.
- Cuando: especifica el conjunto de acciones que lanzan la prueba. La interacción del usuario que acciona la funcionalidad que se desea probar.
- Cuando: especifica el resultado esperado en la prueba. Se observan los cambios en el sistema y se verifica si son los deseados.

Un ejemplo de un caso de prueba escrito en lenguaje Gherking sería:

Escenario: un usuario existente con deseo de autenticarse en el portal

Dado: se elige una cuenta válida

Cuando: se selecciona la opción de autenticación

Entonces el usuario ingresa de manera autenticada al portal

Existen distintas técnicas de diseño de casos de prueba, que permiten seleccionar la menor cantidad de casos con mayor probabilidad de encontrar fallas en el sistema. Por este motivo, estos casos se consideran los más interesantes para ejecutar, ya que las pruebas exhaustivas no solo son imposibles de ejecutar en un tiempo acotado, sino que también son muy caras e ineficientes. Es así como se vuelve necesario seleccionar de una forma inteligente los valores de entrada que tengan más posibilidades de descubrir un error. Para esto, el diseño de pruebas se basa en técnicas bien conocidas y utilizadas, tales como particiones de equivalencia, valores límites, combinaciones por pares, tablas de decisión o máquinas de estado. (Rodríguez, 2014)

A continuación, se pueden ver los diferentes tipos de prueba:

Las pruebas de caja negra son aquellos elementos que son estudiados desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. En otras palabras, de una caja negra interesa su forma de interactuar con el medio que le rodea (en ocasiones, otros elementos que también podrían ser cajas negras) entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace. (Navarrete, 2015)

Las pruebas de caja blanca (también conocidas como pruebas de caja de cristal o pruebas estructurales) se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El probador del sistema escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. (Navarrete, 2015)

Un caso de prueba abstracto se caracteriza por no tener determinados los valores para las entradas y salidas esperadas. Se utilizan variables y se describen con operadores lógicos ciertas propiedades que deben cumplir (por ejemplo, "edad > 18" o "nombre válido"). Entonces, la entrada concreta no está determinada. (Rodríguez, 2014)

Un caso de prueba específico (o concreto) es una instancia de un caso de prueba abstracto, en la que se determinan valores específicos para cada variable de entrada y para cada salida esperada. Cada caso de prueba abstracto puede ser instanciado con distintos valores, por lo que tendrá, al momento de ser ejecutado (o diseñado a bajo nivel) un conjunto de casos de prueba específicos, donde se asigna un valor concreto a cada variable (tanto de entrada como de salida) de acuerdo con las propiedades y restricciones lógicas que tiene determinadas. (Rodríguez, 2014)

La prueba dirigida por datos es una técnica para construir casos de prueba basándose en los datos de entrada, separando el flujo que se toma en la aplicación. O sea, por un lado, se representa el flujo (la serie de pasos para ejecutar el caso de prueba) y por otro lado se almacenan los datos de entrada y salida esperados. Esto permite agregar nuevos casos de prueba fácilmente, ingresando simplemente nuevos datos de entrada y de salida esperados, que sirvan para ejecutar el mismo flujo. (Rodríguez, 2014)

Una herramienta para automatizar estos diferentes casos de prueba es WebDriverIO. WebDriverIO permite automatizar cualquier aplicación escrita con marcos web modernos como React, Angular, Polymer o Vue.js, así como aplicaciones móviles nativas para Android e iOS. Viene con estrategias de selección inteligente que pueden buscar los componentes de React por su nombre de componente y filtrarlos por sus propiedades o estados. (WebDriverIO, 2020)

Según Edgar Figueroa (Figueroa, 2019) algunas ventajas de usar WebDriverIO serían:

- La automatización posee control total sobre los navegadores web debido a que es una interfaz personalizada de World Wide Web Consortium (W3C) WebDriver API en lugar de depender de la implementación de WebDriverJS.

- Es completamente extensible y está escrito para ser lo más flexible e independiente del marco posible. Se puede aplicar en cualquier contexto y no solo sirve para realizar pruebas.
- Tiene una interfaz de línea de comandos, "wdio.js", que hace que la configuración de la prueba sea lo más fácil y simple posible para que cualquier persona que no sea un programador pueda configurarla.
- Tiene soporte para la mayoría de los marcos de prueba sobre desarrollo dirigido a comportamiento o en inglés Behavior Driven Development (BDD) y desarrollo dirigido a pruebas o en inglés Test Driven Development (TDD).
- Tiene un buen soporte y una comunidad de desarrolladores y usuarios finales entusiastas.
- Se puede utilizar con "webdriverCSS" para comparar estilos CSS de un elemento en la página web.
- Proporciona una interoperabilidad lo cual permite realizar múltiples pruebas simultáneas en varios navegadores como son: Chrome, Safari, Firefox, etc.
- Admite la realización de pruebas haciendo uso de elementos como carga y descarga de archivos, ventanas emergentes, entre otros elementos que se dan en una sesión normal de uso de un proyecto web.
- Se pueden realizar pruebas para múltiples sistemas operativos entre ellos: Mac, Windows Linux, etc.

El lenguaje utilizado para las pruebas automatizadas es nodejs que es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript). Este entorno de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript. También aporta muchos beneficios y soluciona muchísimos problemas. (Lucas, 2019)

Node.js fue creado por los desarrolladores originales de JavaScript. Lo transformaron de algo que solo podía ejecutarse en el navegador en algo que se podría ejecutar en los ordenadores como si de aplicaciones independientes se tratara. Gracias a Node.js se puede ir un paso más allá en la programación con JavaScript no solo creando sitios web interactivos, sino teniendo la capacidad

de hacer cosas que otros lenguajes de secuencia de comandos como Python pueden crear. (Lucas, 2019)

Las pruebas automatizadas son probadas en un navegador usando Selenium WebDriver. WebDriver controla un navegador de forma nativa, como lo haría un usuario, ya sea localmente o en una máquina remota utilizando el servidor Selenium, marca un salto adelante en términos de automatización de navegadores. (WebDriver, Selenium, 2020)

Selenium WebDriver se refiere tanto a los enlaces de lenguajes como también a las implementaciones individuales del código controlador del navegador. Esto se conoce comúnmente solo como WebDriver. (WebDriver, Selenium, 2020)

- WebDriver está diseñado como una interfaz de programación simple y más concisa.
- WebDriver es una API compacta orientada a objetos.
- Controla el navegador de manera efectiva.

El controlador es específico para el navegador, como es ChromeDriver para Chrome/Chromium de Google, GeckoDriver para Mozilla Firefox, etc. El controlador corre en el mismo sistema que el navegador. Esto puede, o no puede ser, el mismo sistema donde las pruebas se están ejecutando. La comunicación con el navegador puede ser remota a través de Selenium Server o RemoteWebdriver. Éste último corre en el mismo sistema que el controlador y el navegador. (WebDriverIO, 2020)

El WebDriver tiene un trabajo y solo un trabajo: comunicarse con el navegador a través de uno de los métodos nombrados. El WebDriver no tiene que saber nada sobre las pruebas: no sabe cómo comparar cosas, asegurar un acertar o fallar, y ciertamente no sabe nada acerca de reportes o sobre la gramática. Aquí es donde varios frameworks entran en juego. Como mínimo se necesita un framework de pruebas que compare los enlaces de idiomas, por ejemplo, NUnit para .NET, JUnit para Java, RSpec para Ruby, etc. (WebDriverIO, 2020)

En la Ilustración 2 se muestra la comunicación que deben tener los módulos del software para poder levantar un servidor de pruebas y probarlo en un navegador en específico.

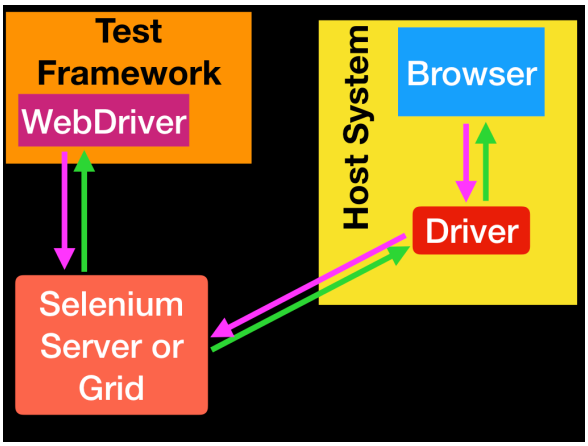


Ilustración 2 - Comunicación entre módulos

Fuente: (WebDriverIO, 2020)

Las pruebas automatizadas son ejecutadas en un ambiente de integración continua. La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software. (¿Qué es la integración continua?, n.d.)

Anteriormente, era común que los desarrolladores de un equipo trabajasen aislados durante un largo periodo de tiempo y solo intentasen combinar los cambios en la versión maestra una vez que habían completado el trabajo. Como consecuencia, la combinación de los cambios en el código resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no se corregían. Estos factores hacían que resultase más difícil proporcionar las actualizaciones a los clientes con rapidez. (¿Qué es la integración continua?, n.d.)

Con la integración continua, los desarrolladores envían los cambios de forma periódica a un repositorio compartido con un sistema de control de versiones como Git. Antes de cada envío, los desarrolladores pueden elegir ejecutar pruebas de unidad local en el código como medida de verificación adicional antes de la integración. Un servicio de integración continua crea y ejecuta automáticamente pruebas de unidad en los nuevos cambios realizados en el código para identificar inmediatamente cualquier error. (¿Qué es la integración continua?, n.d.)

En la Ilustración 3 se muestra el proceso de un archivo configurado con integración continua y sus respectivos lanzamientos a cada etapa del proyecto.

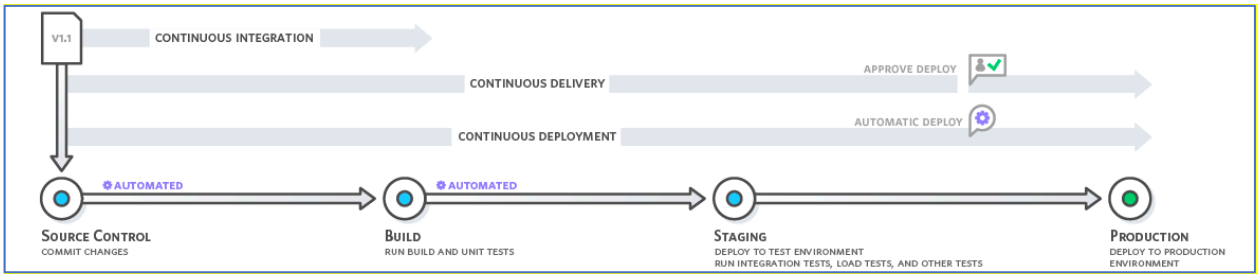


Ilustración 3 - Proceso de integración continua

Fuente: (¿Qué es la integración continua?, n.d.)

La integración continua se refiere a la fase de creación y pruebas de unidad del proceso de publicación de software. Cada revisión enviada activa automáticamente la creación y las pruebas. Con la entrega continua, se crean, prueban y preparan automáticamente los cambios en el código y se entregan para la fase de producción. La entrega continua amplía la integración continua al implementar todos los cambios en el código en un entorno de pruebas y/o de producción después de la fase de creación. (¿Qué es la integración continua?, n.d.)

IV. DESARROLLO

4.1 DESCRIPCIÓN DEL TRABAJO DESARROLLADO

4.1.1 RECORRIDO POR EL PRODUCTO

El recorrido por el producto se lleva a cabo en el proceso de iniciación de cada miembro nuevo del equipo y consiste en mostrar todas las funcionalidades del producto de manera superficial al mismo tiempo que se le comenta al miembro los requerimientos específicos del cliente.

4.1.2 PRUEBAS EXPLORATORIAS Y CREACIÓN DE CASOS DE PRUEBA

La realización de las pruebas exploratorias consiste en verificar el funcionamiento actual del portal, documentar el comportamiento como caso de prueba y documentar algún escenario que falle pero que no tenga relación directa con la funcionalidad en desarrollo. En la Ilustración 4 se muestra un ejemplo de creación de prueba.

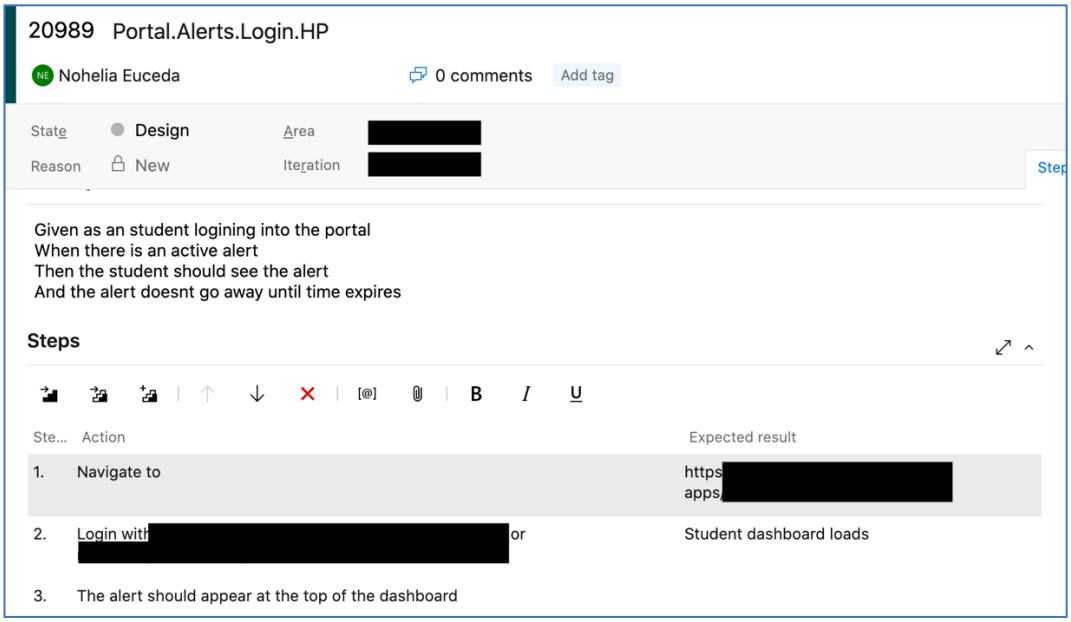
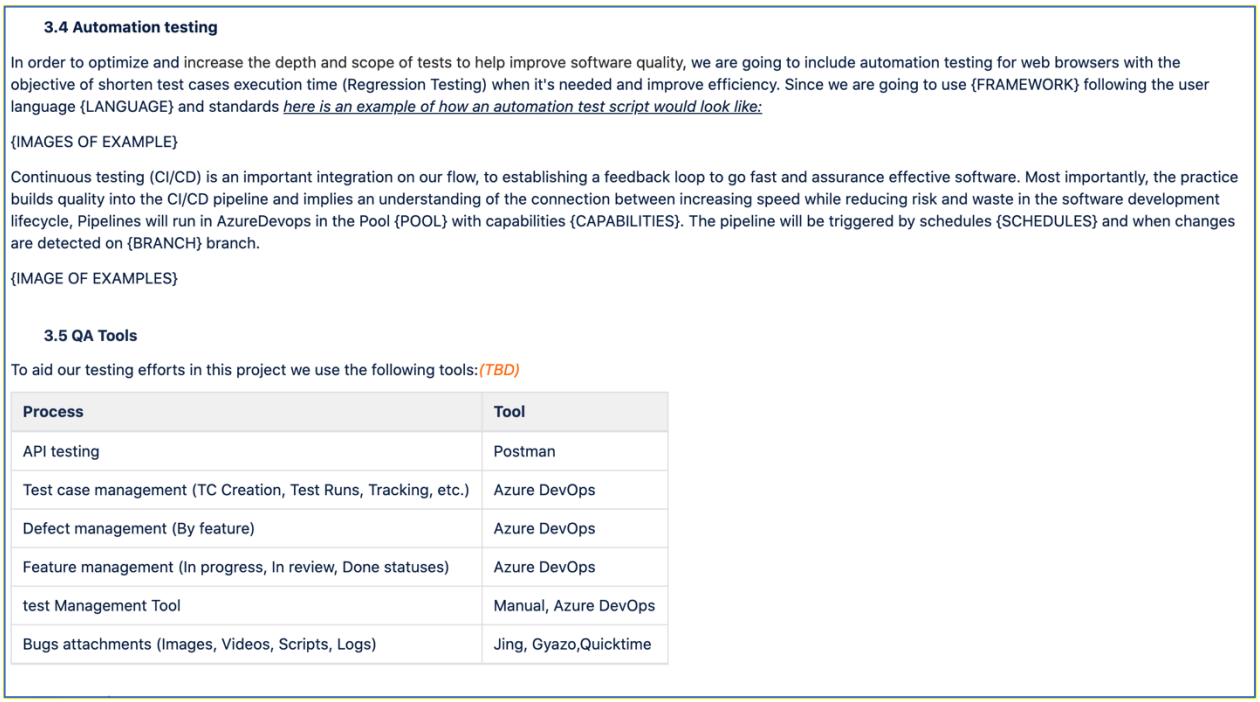


Ilustración 4 – Creación de caso de prueba

Fuente: Elaboración Propia

4.1.3 CREACIÓN DEL PLAN DE CALIDAD DEL PROYECTO

La creación del plan de QA del producto consiste en definir los objetivos, estrategias, riesgos, procesos del proyecto, así como también los tipos de prueba que se estarán realizando en el desarrollo de este. La participación de la estudiante fue añadir los recursos tecnológicos que se estarán usando en el proyecto. En la Ilustración 5 se muestra una parte del plan de pruebas.



3.4 Automation testing

In order to optimize and increase the depth and scope of tests to help improve software quality, we are going to include automation testing for web browsers with the objective of shorten test cases execution time (Regression Testing) when it's needed and improve efficiency. Since we are going to use {FRAMEWORK} following the user language {LANGUAGE} and standards *here is an example of how an automation test script would look like:*

{IMAGES OF EXAMPLE}

Continuous testing (CI/CD) is an important integration on our flow, to establishing a feedback loop to go fast and assurance effective software. Most importantly, the practice builds quality into the CI/CD pipeline and implies an understanding of the connection between increasing speed while reducing risk and waste in the software development lifecycle, Pipelines will run in AzureDevops in the Pool {POOL} with capabilities {CAPABILITIES}. The pipeline will be triggered by schedules {SCHEDULES} and when changes are detected on {BRANCH} branch.

{IMAGE OF EXAMPLES}

3.5 QA Tools

To aid our testing efforts in this project we use the following tools:*(TBD)*

Process	Tool
API testing	Postman
Test case management (TC Creation, Test Runs, Tracking, etc.)	Azure DevOps
Defect management (By feature)	Azure DevOps
Feature management (In progress, In review, Done statuses)	Azure DevOps
test Management Tool	Manual, Azure DevOps
Bugs attachments (Images, Videos, Scripts, Logs)	Jing, Gyazo,Quicktime

Ilustración 5 - Creación del plan de pruebas

Fuente: Elaboración Propia

4.1.4 DISEÑO Y CREACIÓN DE PRUEBAS AUTOMATIZADAS

Las pruebas automatizadas sirven para monitorear las integraciones y funcionalidades del producto sin necesidad de que una persona esté realizando las mismas pruebas muchas veces en el día. El enfoque de las pruebas automatizadas es verificar las funcionalidades que son vitales para la eficiencia del proyecto, por ejemplo, la autenticación, aceptar una solicitud de amistad, crear un comentario en una fotografía, hacer una compra con tarjeta de crédito, etc. En el proyecto se realizarán primero las pruebas de integración.

En la Ilustración 6 se muestra el archivo donde se realizó la autenticación.

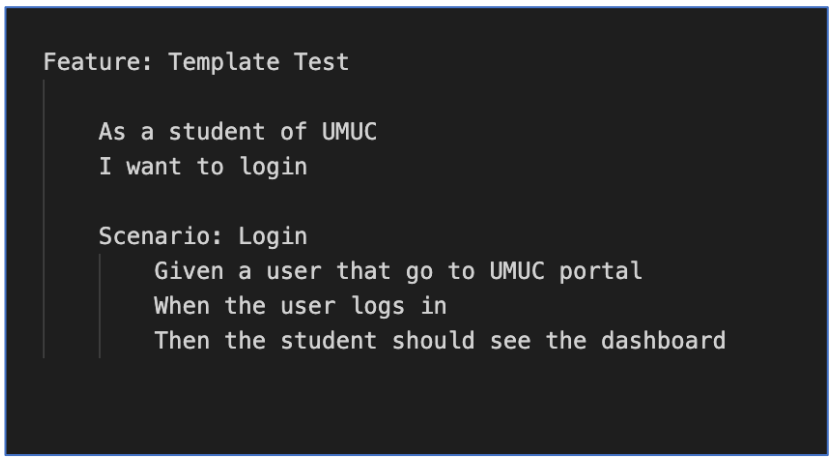


Ilustración 6- Funcionalidad a probar

Fuente: Elaboración Propia

En la Ilustración 7 se puede ver cómo se utilizó un archivo diferente para completar la autenticación

```
const SELECTORS = {
    O365:{
        EMAIL_INPUT: "#i0116",
        SUBMIT_BUTTON: "#idSIButton9",
        PASSWORD_INPUT: "#i0118",
        SIGNIN_BUTTON:"#idSIButton9"

    },

}

class LoginPage {

    get submitButton() {
        return $(SELECTORS.O365.SUBMIT_BUTTON)
    }
    get loginButton() {
        return $(SELECTORS.O365.SIGNIN_BUTTON)
    }
    get emailInput() {
        return $(SELECTORS.O365.EMAIL_INPUT)
    }
    get passwordInput() {
        return $(SELECTORS.O365.PASSWORD_INPUT)
    }

    loginUser() {
        this.submitButton.waitForDisplayed({miliseconds: 1000})
        this.setEmail()
        this.submitButton.click()
        browser.pause(1000)
        this.setPassword()
        this.submitButton.click()
    }
    setEmail() {
        this.emailInput.setValue(USERS.EMAIL)
    }
    setPassword() {
        this.passwordInput.setValue(USERS.PASSWORD)
    }
}
```

Ilustración 7 - Página de Autenticación

Fuente: Elaboración Propia

4.1.5 CREACIÓN Y DISEÑO DE PRUEBAS CONTINUAS

La configuración de las pruebas continuas se realizará en la herramienta de Azure DevOps donde se eligió un servidor de pruebas con capacidades requeridas por el proyecto como ser: sistema operativo Windows, nodejs y Chrome instalados. Las pruebas comienzan a correr automáticamente cuando un usuario hace algún cambio en la rama de Develop. Las pruebas también corren cada cierto tiempo ya configurado por el equipo de calidad y los reportes de cada

prueba son enviados a los correos electrónicos de los encargados del proyecto del lado del cliente y del equipo de desarrollo.

4.1.6 CREACIÓN DE PRUEBAS AUTOMATIZADAS DE PANTALLA DE INICIO Y DISPOSITIVOS MÓVILES PARA NAVEGADORES WEB

Se crearon las pruebas automatizadas para verificar las pantallas de inicio y del perfil del usuario y se hicieron modificaciones para hacer las mismas verificaciones en dispositivos móviles. En la Ilustración 8 se muestra cómo se hace la configuración extra para correr las pruebas en dispositivos móviles. Las etiquetas dirigidas por arrobas definen qué pruebas serán ejecutadas en dispositivos Android o iOS, en este caso las pruebas serán ejecutadas en ambos dispositivos.

```
Feature: Login
    As a student of UMUC
    I want to login


  @androidBrowser @iosBrowser
  Scenario: Login
    Given a user that go to UMUC portal
    When the user logs in
    Then the student should see the dashboard

  @androidBrowser @iosBrowser
  Scenario: Login Header
    Given a user logged in the UMGC portal
    When the user opens the sidebar
    Then should see your name and student id
```

Ilustración 8 - Configuración de pruebas en dispositivo móvil

Fuente: Elaboración Propia

En la Ilustración 9 se puede observar el punto de la automatización donde ya se inició sesión y se verifica la pantalla de inicio. De igual manera se realiza el mismo procedimiento para los dispositivos iOS.

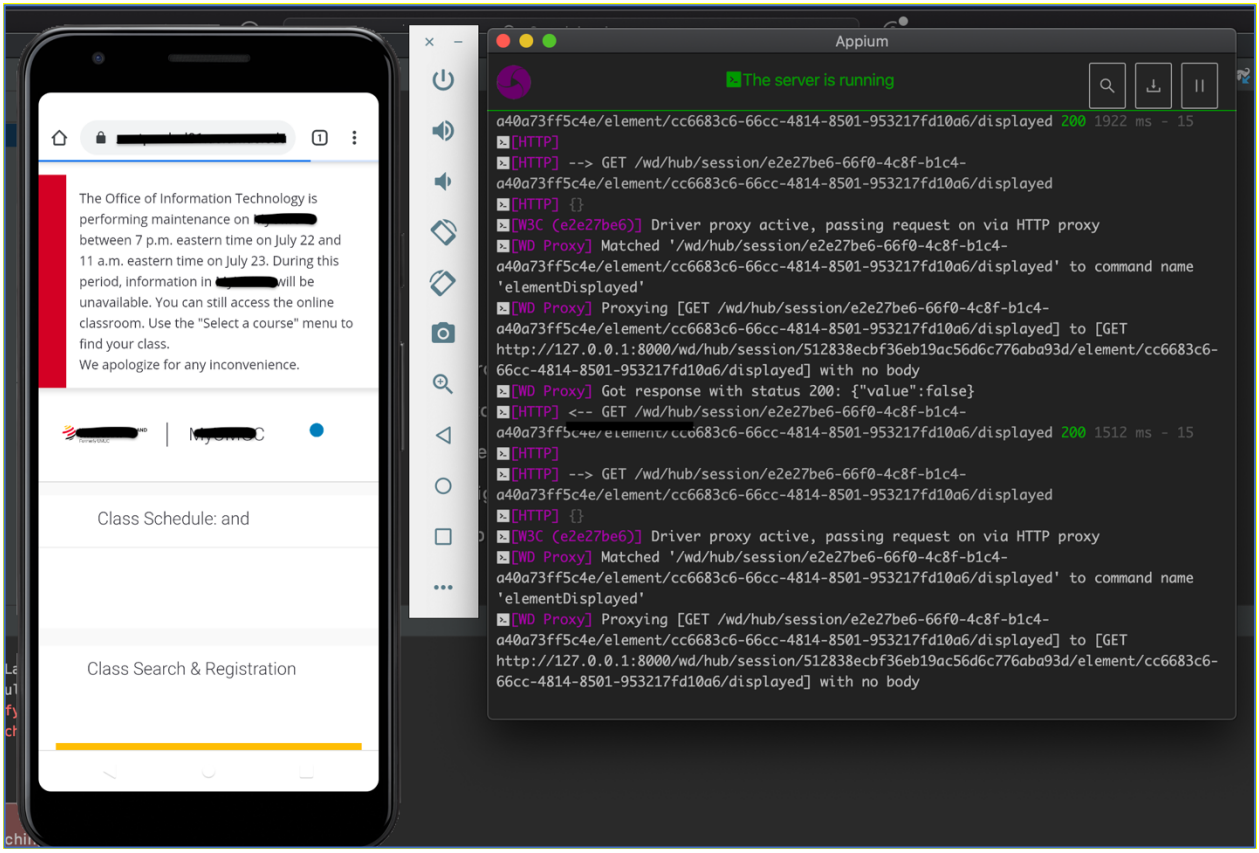


Ilustración 9 - Ejecución de prueba en dispositivo Android

Fuente: Elaboración Propia

4.1.7 CREACIÓN DE PRUEBAS AUTOMATIZADAS DE VERIFICACIÓN DE PERFIL PARA DISPOSITIVOS MÓVILES Y NAVEGADORES WEB

Se crearon las pruebas automatizadas para la verificación del perfil del usuario haciendo llamados a la API correspondiente y comparando los valores retornados por la API con los valores mostrados en el perfil del estudiante, por ejemplo: nombre, fecha de nacimiento, identificación del estudiante, entre otros.

4.1.8 CONFIGURACIÓN DE INTEGRACIÓN CONTINUA DE LAS PRUEBAS AUTOMATIZADAS

Las pruebas automatizadas fueron configuradas en un ambiente de integración continua en un servidor alojado en las instalaciones del cliente, en la Ilustración 10 se muestran los pasos que se ejecutan dentro del servidor.

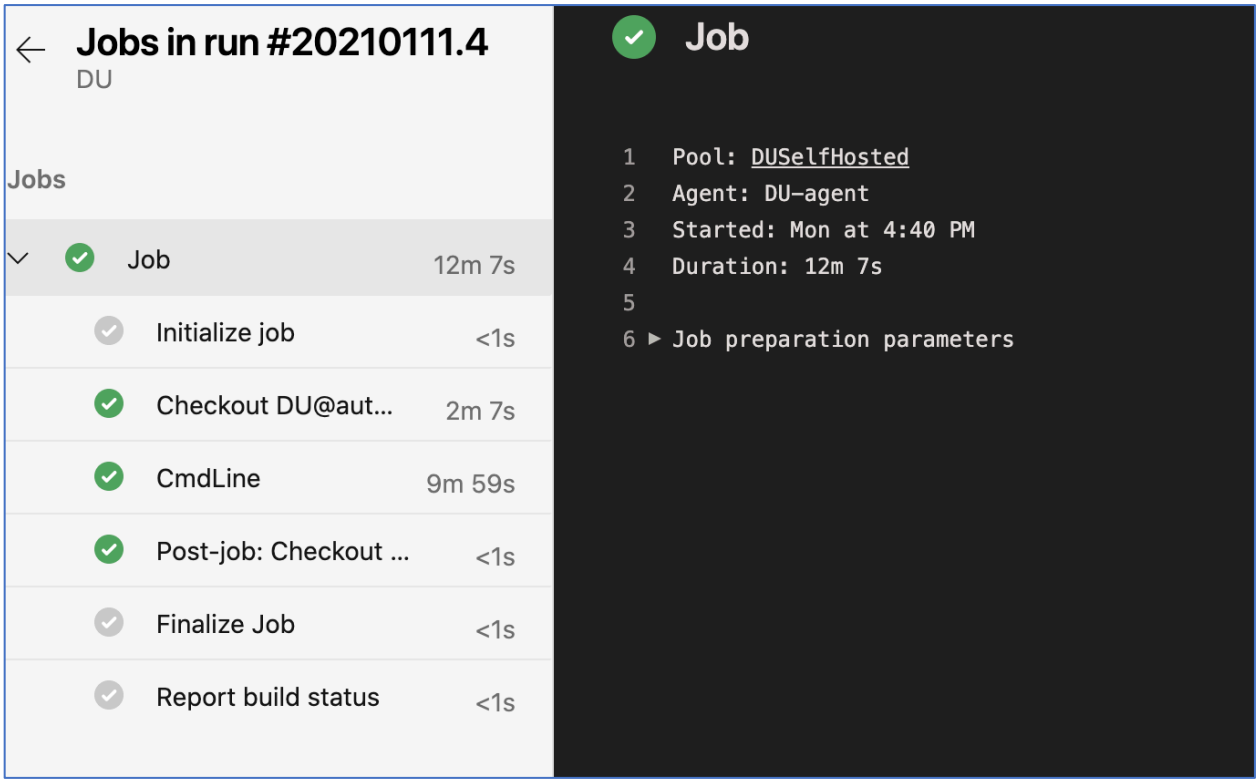


Ilustración 10 - Proceso de Integración Continua de Pruebas

Fuente: Elaboración Propia

4.1.9 CREACIÓN DE PRUEBAS AUTOMATIZADAS PARA SIMULACIÓN DE AUTENTICACIÓN POR MEDIO DE API

Con el fin de reducir el tiempo de ejecución de la matriz de pruebas automatizadas se diseñó el código para simular la autenticación de un usuario usando llamados de API sin necesidad de ejecutar la prueba gráfica de autenticación. El tiempo de autenticación se redujo de 10 segundos a 4 segundos, ya que esta prueba debía ejecutarse en 19 distintas pruebas equivalía a 190 segundos lo cual se redujo a 76 segundos, una reducción del 60%.

En la Ilustración 11 se ejemplifica la manera anterior de realizar el proceso de autenticación.

```
HomePage.OpenMenuSettings()
LoginPage.NavigateToLoginScreen()
LoginPage.logincredentials(Utils.Credentials.validEmail, Utils.Credentials.password)
LoginPage.Auth0LoginButtonClick()
LoginPage.LoginSuccessLabel()
browser.url(`/album?id=${Utils.albumID}`)
```

Ilustración 11- Autenticación utilizando interfaz gráfica

Fuente: Elaboración Propia

La mejora se realizó utilizando llamados a API y colocando la información necesaria en el almacenamiento del navegador, se muestra en la Ilustración 12.

```
const api = new Api(browser.config.baseUrl.includes('[redacted]') ? utils.Auth0ProdDomain : utils.Auth0DevDomain);
let testlogin = browser.call(() => {
return api.loginUserOnAuth0(utils.Credentials.validEmail, utils.Credentials.password)


console.log('nonce: ',utils.Auth0LoginNonce)
console.log('login ticket: ',utils.Auth0LoginTicket)
let authorizeUser = browser.call(() => {
    return api.AuthorizeUserOnAuth0();
})
browser.call(() => {
    return GlobalFunctions.RegexToken(authorizeUser);
})
let userInfo = browser.call(() => {
    return api.UserInfoOnAuth0();
})
const api2 = new Api(browser.config.baseUrl.includes('[redacted]') ? utils.Auth0ManagementProdDomain: utils.Auth0ManagementDevDomain);
let orastreamToken = browser.call(() => {
    return api2.UserSteamInfoOnAuth0();
})
browser.execute(function (orastreamToken, id_token, user_info, user_email) {
    this.localStorage.setItem("ade:orastream-token", orastreamToken)
    this.localStorage.setItem("ade:id-token", id_token)
    this.localStorage.setItem("ade:user-info", JSON.stringify(user_info))
    this.localStorage.setItem("ade:user-email", user_email)
}, orastreamToken.orastream_jwt, utils.Auth0IDToken, userInfo, utils.Credentials.validEmail)
```

Ilustración 12 - Autenticación utilizando API

Fuente: Elaboración Propia

4.1.10 Creación de manual sobre configuración de integración continua

Al haber creado y probado las configuraciones de los pasos a seguir en el servidor de integración continua, se solicitó crear un manual cronológico detallado para poder mantener un registro y así poder reproducir la idea en el futuro o en otros proyectos de la empresa.

4.1.11 PRUEBAS AUTOMATIZADAS DE API

Las pruebas automatizadas de API se configuraron en la herramienta Postman con el fin de revisar el retorno de datos del directorio consultado. Las pruebas consisten en asegurar que el tiempo de la respuesta sea prudente, que el estado de la respuesta sea correcto y que retorne las llaves correspondientes. En la Ilustración 13 se ejemplifica.

```
var time = pm.response.responseTime
let jsonData = pm.response.json()

pm.test(`Status code is 200 - Response time: ${time} ms`, function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 4000ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(4000);
});
pm.test(`Data assertion`, () => {
    pm.expect(jsonData.Grades_Data.UC_DU_VIEW_GRADE_DOC).to.be.an("array");
    _.each(jsonData.Grades_Data.UC_DU_VIEW_GRADE_DOC, (item) => {
        pm.expect(item).to.be.an("object")
        pm.expect(item.Grade_details).to.be.an('array')
        _.each(item.Grade_details.Class_tbl_se_vw, (item) => {
            pm.expect(item).to.be.an("object")
            pm.expect(item).to.have.keys('CLASS_NBR', 'SUBJECT', 'CATALOG_NBR','CLASS_SECTION','COMPONENT','DESCR','UNT_TAKEN',
                'CLASS_TYPE','CRSE_GRADE_OFF','GRADE_POINTS')
                pm.expect(item.Acad_stdng_actn).to.be.an('array')
                _.each(item.Acad_stdng_actn, (item) => {
                pm.expect(item).to.be.an("object")
                pm.expect(item).to.have.keys('EFFDT', 'EFFSEQ', 'ACAD_STNDNG_ACTN','ACAD_PROG','ACAD_STNDNG_STAT','ACTION_DT',
                    'OPRID')

            })
        })
    })
});
```

Ilustración 13 - Pruebas Automatizadas de API en Postman

Fuente: Elaboración Propia

4.1.12 ENTRENAMIENTO Y PREPARACIÓN DE CHARLA SOBRE PRUEBAS DE ACCESIBILIDAD

Se designó preparar una charla sobre pruebas de accesibilidad en entorno web. Donde se tomó un curso de este tipo de pruebas y se compartió a los demás miembros del equipo sobre los hallazgos.

4.2 CRONOGRAMA DE ACTIVIDADES

En la Ilustración 14 se muestra el cronograma con las actividades más relevantes desarrolladas durante la práctica profesional.

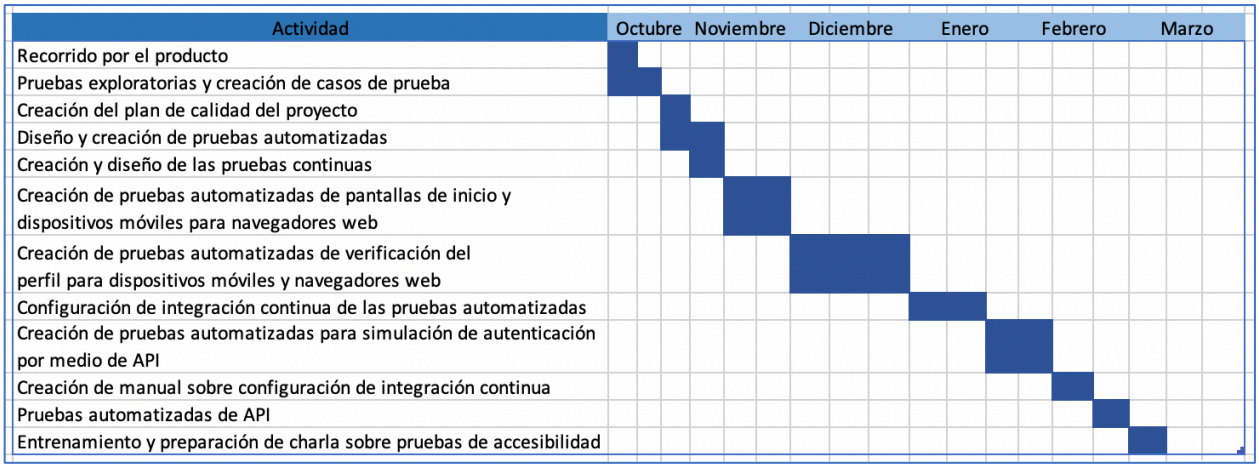


Ilustración 14 - Diagrama de Gantt

Fuente: Elaboración Propia

V. CONCLUSIONES

1. En este documento se mostró la evidencia que por medio de pruebas funcionales y no funcionales se cumplieron los requerimientos establecidos por el cliente, así mismo se aseguró la calidad y eficiencia del producto verificando los requerimientos establecidos por el equipo de arquitectura. Las tareas realizadas fueron revisadas por los empleados con más experiencia en el área de QA mostrando agrado a lo terminado.
2. Se participó en sesiones de entrenamiento con los clientes y dueños de producto para obtener conocimiento técnico y comprender el producto y sus características, para poder realizar pruebas críticas sobre la aplicación.
3. Se diseñaron y desarrollaron pruebas automatizadas de interfaz de usuario en WebDriverIO y pruebas automatizadas de API en Postman para monitorear las funcionalidades de la aplicación y así alertar cuando algún módulo del producto no cumpliera con la funcionalidad esperada.
4. Se crearon matrices de prueba con sus distintos casos seccionados por módulos, para poder tener con exactitud una pauta del comportamiento del producto establecido por el cliente.
5. Se realizararon pruebas exploratorias al inicio del proyecto para obtener experiencia en la aplicación. Durante la ejecución del proyecto se efectuaron pruebas de aceptación, sanidad y de usabilidad para garantizar la estabilidad del producto en cada entregable realizado por el equipo de desarrollo. Con cada nueva versión lista para el ambiente de producción se realizaron pruebas de regresión para dar garantía de la mejor condición del portal.
6. Durante la duración del proyecto se monitoreó el portal en busca de defectos y errores con el objetivo de mejorar el estándar de calidad en los servicios. La herramienta de WebDriverIO y Postman fueron esenciales para lograr esta meta permitiendo un fácil desarrollo de este tipo de pruebas automatizadas.

VI. RECOMENDACIONES

1. Se debe agilizar la autorización de permisos sobre usuarios, el producto y servidores del cliente para realizar las tareas definidas en el tiempo estimado por el equipo.
2. Se debe delegar a alguien cercano a los equipos de desarrollo y calidad el trabajo de la creación de usuarios de prueba, para así poder crear distintos usuarios específicos y hacer pruebas puntuales con ellos.
3. Al cliente de la Universidad de Maryland se le recomienda migrar su arquitectura de pruebas automatizadas de Selenium Webdriver Java a un lenguaje de programación más conocido, con menor curva de aprendizaje y con mejor documentación existente como lo es NodeJs o Python.
4. A la universidad se le recomienda enseñar más a los estudiantes sobre el área de calidad y la importancia que tiene en un producto en el mercado actual.
5. A la academia se le recomienda mejorar los sílabos de las clases para así poder impartir diferentes recursos y temas que tienen un mayor auge como lo son productos en la nube y en demanda.
6. A los estudiantes se les recomienda ser autodidactas y proactivos en emprender experiencias no enseñadas en la universidad ya que el rubro tiende a mejorar las tecnologías usadas aceleradamente.

BIBLIOGRAFÍA

1. Angular. (s.f de s.f de s.f). *Introduction to the Angular Docs*. Recuperado el 21 de octubre de 2020, de https://angular.io/docs
2. Bhamra, P. K. (22 de enero de 2020). *Selenium Tutorial – Learn Selenium from Experts*. Recuperado el 1 de noviembre de 2020, de intellipaat.com: https://intellipaat.com/blog/tutorial/selenium-tutorial/
3. BrowserStack. (s.f de s.f de s.f). *Selenium*. Recuperado el 31 de octubre de 2020, de BrowserStack: https://www.browserstack.com/selenium
4. Leonardo Da Vinci project. (S.F de S.F de 2014). *Definición del aseguramiento de la calidad*. Recuperado el 1 de noviembre de 2020, de Eurocytology project: https://www.eurocytology.eu/es/course/795
5. Elvir, M. (June de 2019). *Iconic History*. Recuperado el 12 de noviembre de 2020, de Hello ICONIC Confluence: https://icomsco.atlassian.net/wiki/spaces/ICONIC/pages/650608677/Iconic+History+Espa+ol
6. *Estandar IEEE 610*. (1990). Recuperado el 12 de noviembre de 2020 de https://ieeexplore.ieee.org/document/159342
7. Figueroa, E. (2019). *How to Choose an Automation Framework*. Recuperado el 12 de noviembre de 2020 de Wizeline: https://www.wizeline.com/automation-framework/
8. Hernández, R. (2016). *BDD, Cucumber y Gherkin. Desarrollo dirigido por comportamiento*. Recuperado el 12 de noviembre de 2020 de Genbeta: https://www.genbeta.com/desarrollo/bdd-cucumber-y-gherkin-desarrollo-dirigido-por-comportamiento

9. *ISO 9001:2015 Sistemas de gestión de la calidad*. (2015). Recuperado el 12 de noviembre de 2020 de ISO 9001:2015, Sistemas de gestión de la calidad: https://www.iso.org/obp/ui/#iso:std:iso:9001:ed-5:v1:es

10. Lucas, J. (4 de Septiembre de 2019). *Qué es NodeJS y para qué sirve*. Recuperado el 12 de noviembre de 2020 de Open Webinars: https://openwebinars.net/blog/que-es-nodejs/

11. Mares, G. S. (s.f de s.f de s.f). *Selenium WebDriver en un Ambiente de Pruebas Continuas*. Recuperado el 31 de octubre de 2020, de Software Guru: https://sg.com.mx/revista/54/selenium-webdriver-un-ambiente-pruebas-continuas

12. Mozilla. (11 de mayo de 2019). *What is JavaScript?* Recuperado el 20 de noviembre de 2020, de Developer Mozilla: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

13. Navarrete, J. E. (2015). *REFINAMIENTO DE SOFTWARE*.

14. Pressman, R. S. (2010). *Ingeniería del software*. New York: McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V.

15. *¿Qué es la integración continua?* (s.f.). Recuperado el 12 de noviembre de 2020 de Amazon Web Services: https://aws.amazon.com/es/devops/continuous-integration/

16. Rivera, L. L. (2010). *Migración de una aplicación distribuida a un entorno Web*. Cholula, Puebla, México.

17. Software Freedom Conservancy. (s.f de s.f de s.f). *Selenium History*. Recuperado el 1 de noviembre de 2020, de Selenium: https://selenium.dev/history/

18. Rodríguez, F. T. (2014). *INTRODUCCIÓN A LAS PRUEBAS DE SISTEMAS DE INFORMACIÓN*. Montevideo: Abstracta.

19. *WebDriverIO*. (2020). Recuperado el 12 de noviembre de 2020 de WebDriverIO: https://webdriver.io/

20. Tolfsen, A. (2020). *W3c/webdriver* (2.1.0) [HTML]. Recuperado el 12 de noviembre de 2020 World Wide Web Consortium. https://github.com/w3c/webdriver