Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Gustavo Alexander Orozco Tay

Asesorado por el Ing. César Rolando Batz Saquimux

Guatemala, noviembre de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

GUSTAVO ALEXANDER OROZCO TAY
ASESORADO POR EL ING. CÉSAR ROLANDO BATZ SAQUIMUX

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Kevin Vladimir Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Juan Alvaro Díaz Ardavín
EXAMINADOR	Ing. Edgar Josué Gonzalez
EXAMINADOR	Ing. José Ricardo Morales Prado
SECRETARIO	Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 23 de Septiembre de 2020.

Gustavo Alexander Orozco Tay

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERIA EN CIENCIAS Y SISTEMAS

Guatemala 24 de Septiembre de 2021.

MSc. Ing. Carlos Gustavo Alonzo
Director Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería, USAC
Ciudad Universitaria, Guatemala

Reciba un cordial saludo, y por este medio hago constar que yo, Ing. César Rolando Batz Saquimux con No. Colegiado 8549, doy como visto bueno el desarrollo del trabajo final de graduación del estudiante *GUSTAVO ALEXANDER OROZCO TAY*, de la carrera de Ingeniería *en Ciencias y Sistemas* de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, quien se identifica con CUI: 2559 47062 0917 y Registro Académico: 2002-12848, quien realizo su informe final relacionado al proyecto asignado de EPS:

- *Implementación de una Arquitectura de Pruebas Automáticas sobre la Plataforma del Sistema de Control del Proyecto de Desarrollo de Transferencia Tecnológica (Plataforma DTT) de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala /* ***con resultado satisfactorio*** *en cumplimiento de los objetivos planteados y cumpliendo con la entrega de todos los productos comprometidos en el anteproyecto aprobado.*

Dando por concluido el desarrollo e implementación del proyecto planteando las soluciones efectivas para el beneficio de la institución donde se desarrolló la misma.

Doy por concluido de forma eficiente ante mi persona el desarrollo de su trabajo de graduación. Y por lo anterior extiendo la presente carta de Aprobación de su Trabajo de Graduación Final.

Agradeciendo su colaboración y apoyo a esta entidad. Sin otro particular me suscribo.

Atentamente,

Ingeniero en Ciencias y Sistemas
César Rolando Batz Saquimux
No. Colegiado 8549
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

César Rolando Batz Saquimux
Ingeniero en Ciencias y Sistemas
Colegiado No. 8549

Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Unidad de EPS

Guatemala, 05 de octubre de 2021.
REF.EPS.DOC.409.10.2021.

Ing. Oscar Argueta Hernández
Director Unidad de EPS
Facultad de Ingeniería
Presente

Estimado Ingeniero Argueta Hernández:

Por este medio atentamente le informo que como Supervisora de la Práctica del Ejercicio Profesional Supervisado, (E.P.S) del estudiante universitario de la Carrera de Ingeniería en Ciencias y Sistemas, **Gustavo Alexander Orozco Tay, Registro Académico 200212848 y CUI 2559 47062 0917** procedí a revisar el informe final, cuyo título es **IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN UNIVERSIDAD DE SAN CARLOS DE GUATEMALA.**

En tal virtud, **LO DOY POR APROBADO**, solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,

"Id y Enseñad a Todos"

Universidad de San Carlos de Guatemala
COORDINADOR(A) EPS
ÁREA TECNOLOGÍA Y [illegible]
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

Inga. Floriza Felipa Ávila Pesquera de Medinilla
Supervisora de EPS
Área de Ingeniería en Ciencias y Sistemas

FFAPdM/RA

Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Unidad de EPS

Guatemala, 05 de octubre de 2021.
REF.EPS.D.203.10.2021.

Ing. Carlos Gustavo Alonzo
Director Escuela de Ingeniería Ciencias y Sistemas
Facultad de Ingeniería
Presente

Estimado Ingeniero Alonzo:

Por este medio atentamente le envío el informe final correspondiente a la práctica del Ejercicio Profesional Supervisado, (E.P.S) titulado **IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** que fue desarrollado por el estudiante universitario **Gustavo Alexander Orozco Tay, Registro Académico 200212848 y CUI 2559 47062 0917** quien fue debidamente asesorado por el Ing. César Rolando Batz Saquimux y supervisado por la Inga. Floriza Felipa Ávila Pesquera de Medinilla.

Por lo que habiendo cumplido con los objetivos y requisitos de ley del referido trabajo y existiendo la aprobación del mismo por parte del Asesor y la Supervisora de EPS, en mi calidad de Director apruebo su contenido solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,
"Id y Enseñad a Todos"

Universidad de San Carlos de Guatemala
DIRECCIÓN
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

Ing. Oscar Argueta Hernández
Director Unidad de EPS

/ra

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 11 de octubre de 2021

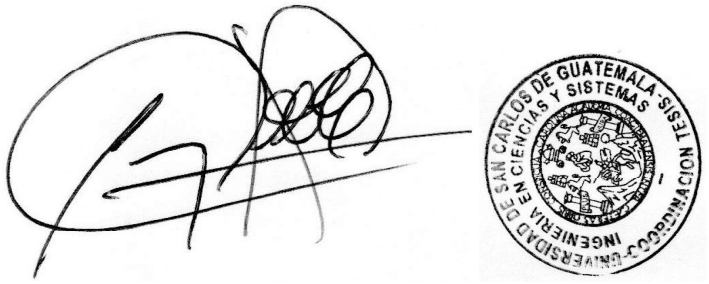
Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación-EPS del estudiante **GUSTAVO ALEXANDER OROZCO TAY** carné **200212848** y CUI **2559 47062 0917,** titulado: **"IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERIA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA",*** *realizado por el estudiante, GUSTAVO ALEXANDER OROZCO TAY aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

Msc. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 16 de noviembre de 2021

USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

Decanato
Facultad de Ingeniería
24189101- 24189102
secretariadecanato@ingenieria.usac.edu.gt

DTG. 672.2021

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** presentado por el estudiante universitario: **Gustavo Alexander Orozco Tay,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Inga. Anabela Cordova Estrada
Decana

Guatemala, noviembre de 2021

AACE/asga

ACTO QUE DEDICO A:

Jehová Dios todo poderoso, por darme el privilegio de vivir y la bendición de alcanzar este sueño, guiarme y protegerme en todo el camino, dándome fuerzas para nunca rendirme y sobre todo ser una persona de bien.

Mi madre Paula Tay Castañón, por su amor y apoyo incondicional durante toda mi vida, siendo un gran ejemplo de superación, dedicación y servicio. Que ha sabido formarme con buenos sentimientos, hábitos y valores.

Mi padre Fredy Jaime Orozco de León, por su amor y apoyo para poder graduarme a nivel diversificado y guiarme durante varios años.

Mis hijos Eidan Gustavo y Ricardo Edelman Orozco Pulido, por ser dos ángeles en mi vida y darme la oportunidad de ser un ejemplo para ellos.

Mis hermanos Edgar Giovany, Marvin Rene y Lilian Celeste Orozco Tay, por el amor y apoyo que siempre me han brindado, con su ayuda y cariño han sido parte fundamental de mi vida.

Mis abuelitos

Anita Castañón y Cirilo Tay, por su gran amor, poderme proteger y guiarme en mi niñez, siendo un ejemplo de alegría y humildad. Y que aun estando lejos los llevo siempre en mi corazón.

Miriam de León y Manuel Orozco, por su amor y apoyo en mi niñez, guiándome siempre por el buen camino para ser una persona de bien.

Mis amigos

Con los que a lo largo de todos estos años he tenido el privilegio de compartir, momentos buenos y malos, pero que siempre están para brindarme su ayuda. Por ser una importante influencia en mi carrera, el apoyo que me han brindado y sobre todo la amistad que se ha formado.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por brindarme la oportunidad de continuar con mis estudios, siempre velar por nuestros intereses como estudiantes. Gracias por formarme como profesional.
Facultad de Ingeniería	Por haberme permitido formarme profesionalmente en sus aulas y todo el apoyo brindado para alcanzar este gran logro.
Escuela de Ingeniería en Ciencias y Sistemas	Por ser parte de esta gran casa de estudios, que nos ha guiado y enseñado por varios años, gracias por su arduo trabajo y dedicación en la formación de profesionales.
María Avelyn Pulido Cardona	Por apoyarme por muchos años, y brindarme la más grande bendición, el de ser Papá. Infinitas gracias.
Inga. Claudia Lorena Cabrera Prado	Por haberme dado la primera oportunidad de un trabajo profesional, creer en mis capacidades y conocimientos cuando aún era un estudiante. Gracias por su gran calidad de persona que me ha demostrado ser con su amistad.

Ing. César Rolando Batz Saquimux	Por toda la colaboración brindada a mi persona para la realización de esta Tesis y su guía durante el proyecto de EPS. Gracias por su gran amistad y apoyo en todo momento.
Ing. Miguel Marín de León	Por brindarme la oportunidad y el apoyo constante para implementar el proyecto de EPS en la Escuela, gracias por su excelente gestión como coordinador de proyectos.
Inga. Floriza Felipa Ávila Pesquera de Medinilla	Gracias por su guía y constante apoyo para realizar el proyecto de EPS, también por ayudarme en el proceso para graduarme y sobre todo por ser una persona siempre amable y muy colaborativa.
Familia Sáenz Urzúa	Por abrirme las puertas de su hogar, guiarme y enseñarme sobre Jehová nuestro Dios, gracias por ayudarme con sus palabras, consejos y acciones para reencontrarme con mi fe.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
PUT	Método de petición HTTP que permite enviar datos para actualizar registros.
POST	Método de petición HTTP que permite enviar datos para crear registros.
DELETE	Método de petición HTTP que permite enviar datos para eliminar registros.
GET	Método de petición HTTP que solo solicita datos.
GB	Un gigabyte es una unidad de almacenamiento de información equivalente a 10^9 (1 000 000 000 -mil millones-) de bytes.
MB	Un megabyte es una unidad de almacenamiento de información equivalente a 10^6 (1 000 000 -un millón-) de bytes.

GLOSARIO

Docker	Es una tecnológica que permite crear imágenes y contenedores. Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y confiable de un entorno informático a otro. Una imagen de contenedor de Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones.
DevOps	Es el acrónimo (en inglés) entre la unión de trabajo de dos equipos, Desarrolladores (Developers) y Operaciones (Operations).
Escuela	Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala.
Frameworks	Conjunto de librerías desarrolladas por otras personas y que están encapsuladas para su transportación permitiendo su utilización (ejecución) en un proyecto de software para la realización de ciertas tareas.

IntelliJ IDE	Es un entorno de desarrollo integrado (IDE) para el desarrollo de programas informáticos.
Jenkins	Es un servidor de automatización de código abierto autónomo que se puede utilizar para automatizar todo tipo de tareas relacionadas con la creación, prueba y entrega o implementación de software.
Plataforma DTT	Plataforma del Sistema de Control del Proyecto de Desarrollo de Transferencia Tecnológica (Plataforma DTT) de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala.
Rest-Assured	Es un DSL (Domain Specific Languages) de Java para simplificar las pruebas de servicios basados en REST construidos sobre peticiones de tipo HTTP. Admite solicitudes POST, GET, PUT, DELETE, OPTIONS, PATCH y HEAD y se puede utilizar para validar y verificar la respuesta de estas solicitudes. Simplemente podemos considerar como DSL cualquier lenguaje que se especialice en modelar o resolver un conjunto específico de problemas. Este conjunto específico de problemas se denomina dominio de aplicación o empresarial.
Scrum	Es un marco de trabajo simple que promueve la colaboración en los equipos para lograr desarrollar productos complejos.

Selenium	Es un proyecto con una gran variedad de herramientas y bibliotecas que permiten la automatización de navegadores web. Proporciona extensiones para emular la interacción del usuario con los navegadores, un servidor de distribución para escalar la asignación del navegador y la infraestructura para implementaciones de la especificación W3C WebDriver que le permite escribir código intercambiable para los principales navegadores web.
TestNG	Es un marco de prueba inspirado en JUnit y NUnit pero que presenta algunas funcionalidades nuevas que lo hacen más poderoso y fácil de usar, como: anotaciones, grupos de ejecución, y otros. TestNG está diseñado para cubrir todas las categorías de pruebas: unitarias, funcionales, end-to-end, integración, entre otros.

RESUMEN

Con la implementación de una arquitectura de pruebas automáticas, se busca reducir principalmente el tiempo, que se utiliza en la fase de control de calidad y mantener los estándares de calidad siempre en alto, siendo este último la principal cara de presentación que se tiene entre los desarrolladores y los clientes.

Utilizamos el lenguaje de programación Java, por la facilidad que nos da al poder integrarse con una gran variedad de frameworks y librerías. Al utilizar Maven nos apoyamos en tener un mejor orden en cuanto a la gestión de dependencias que el proyecto necesita e integrado con TestNG, nos da una gran flexibilidad para realizar pruebas automáticas. Sin dejar a un lado a Selenium que es el framework que nos permite interactuar con los navegadores Chrome y Firefox. Así como otras librerías que nos permiten realizar un mejor desarrollo en el área de programación, utilizando siempre buenas prácticas para su mantenimiento en el futuro.

Nos inclinamos por utilizar Jenkins, para tener un entorno de ejecución de las pruebas automáticas, y por medio de Docker (que es una herramienta que nos permite encapsular todas las dependencias para su funcionamiento) podemos transportar esta imagen entre diferentes equipos de computación.

Finalmente, realizar las primeras pruebas unitarias en el proyecto de la Plataforma DTT, es la base para que los nuevos desarrollos tengan una guía y puedan crear este tipo de pruebas que ayudan en reducir en un gran porcentaje los errores que se puedan generar.

OBJETIVOS

General

Implementar una arquitectura que permita validar de forma automática las principales funcionalidades que actualmente dispone la Plataforma del Sistema de Control del Proyecto de Desarrollo de Transferencia Tecnológica (Plataforma DTT) de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala.

Específicos

1. Configurar una arquitectura para la ejecución, notificación y reportería de pruebas automáticas a la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas.

2. Configurar el entorno de desarrollo de la programación de pruebas automáticas sobre la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas.

3. Crear pruebas automáticas (pruebas de regresión) enfocadas a las principales funcionales de la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas.

4. Implementar pruebas unitarias y de integración en el código fuente desarrollado en la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas y que sirva de base para que los estudiantes puedan desarrollar estas pruebas en futuros proyectos.

INTRODUCCIÓN

Una de las fases más importantes y cruciales en el ciclo del desarrollo de software es la etapa de control de calidad, ya que es aquí, donde se valida que el sistema desarrollado cumpla con todos los requerimientos, tanto funcionales como no funcionales, que el cliente espera.

Por lo cual las personas encargadas de realizar dichas actividades deben de tener un amplio conocimiento del negocio y una buena habilidad en el uso del sistema informático desarrollado. Permitiendo que se pueda validar de la mejor manera los requerimientos solicitados.

Lamentablemente la mayoría de los proyectos, ponen muy poco énfasis en esta etapa y asignan poco recurso y tiempo para completar dichos objetivos, teniendo que hacer grandes sacrificios en el tema de calidad.

Esta fase hace varios años, era realizada de forma manual por un equipo de personas, confiando que estas tengan el conocimiento mínimo del negocio y puedan utilizar las herramientas tecnológicas de la mejor forma. Con el paso del tiempo, se ha mejorado poniendo personas más capacitadas y que tengan un amplio dominio en temas de software y desde un inicio del proyecto estén acompañando en la toma de requerimientos para tener una mejor claridad al momento de realizar las validaciones a la aplicación desarrollada.

Y como era de esperar, inicio también la automatización de pruebas, en otras palabras, desarrollar un sistema que se encargue de realizar la mayor

cantidad de pruebas de forma automática a una aplicación durante su fase de desarrollo.

Esto vino a cambiar los paradigmas sobre la fase de control de calidad en el ciclo del desarrollo de software, entre algunos ejemplos podemos mencionar que las pruebas que se desarrollan se realizan o programan en forma paralela con el desarrollo de las funcionalidades del sistema. También podemos hablar de otro paradigma, que es el crear primero las pruebas automáticas y luego crear o desarrollar las funcionalidades del sistema. Se han creado nuevos lenguajes de desarrollado enfocados solamente para el desarrollo de pruebas automáticas. Así como el uso de herramientas para la ejecución y reportes de estas pruebas, lo que ha permitido mejorar la calidad de las aplicaciones que se desarrollan en la actualidad, incluso reduciendo el tiempo que antes se tomaba esta fase y teniendo un ciclo más rápido de desarrollo de software.

El cambio no es fácil, pero tampoco es imposible. Lo importante es tener la certeza que, al destinar recursos para automatizar la fase de control de calidad, tendremos un producto de buena calidad con la menor cantidad de errores, y una gran satisfacción con los clientes y usuarios teniendo un sistema seguro y confiable, que hoy en día ya no es un valor agregado del producto, sino una obligación por parte de los involucrados al momento de entregar un proyecto.

Durante el desarrollo de este proyecto de automatización de pruebas, podremos conocer diferentes herramientas que nos permitirán de una forma simple, como podemos iniciar con la automatización de un proyecto de software y utilizar buenas prácticas para el mantenimiento de estas pruebas, permitiendo a futuros estudiantes que se interesen en esta área el poder conocer un poco más sobre este paradigma.

1. FASE DE INVESTIGACIÓN

A continuación, se presenta una breve investigación sobre la institución donde se implementó el proyecto desarrollado, así como la descripción del problema que se busca minimizar con la solución realizada.

1.1. Antecedentes de la Institución

El proyecto se implementó en la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, ubicada en el Edificio T-3 Faculta de Ingeniería, en el nivel cero, Escuela de Ingeniería en Ciencias y Sistemas, Universidad de San Carlos de Guatemala, zona 12. Siendo una institución de tipo Publica, y teniendo una demanda de aproximadamente 600 personas que se inscriben en la carrera de Ingeniería en Ciencias y Sistemas al año.

La Facultad de Ingeniería se dedica a la formación de profesionales de prestigio, cuyos conocimientos contribuyen al progreso científico y tecnológico de Guatemala.

1.1.1. Reseña Histórica

> Desde 1676, en sus primeras épocas, la Universidad de San Carlos graduaba teólogos y abogados; posteriormente, a médicos. En el año 1769 se crearon cursos de física y geometría, lo que marcó el inicio de la enseñanza de las ciencias exactas en Guatemala.[1]

[1] Facultad de Ingeniería Universidad de San Carlos de Guatemala. *Antecedentes, Desarrollo CCI.* https://portal.ingenieria.usac.edu.gt/index.php/aspirante/antecedentes.Consulta: 1 de febrero de 2021.

En el año 1873 se fundó la Escuela Politécnica para formar Ingenieros Militares, Topógrafos y de Telégrafos, además de Oficiales Militares. Decretos gubernativos específicos de 1875 son el punto de partida para considerar la creación formal de las carreras de Ingeniería en la recién fundada Escuela Politécnica; carreras que más tarde se incorporaron a la Universidad.[2]

En 1879 se estableció la Escuela de Ingeniería en la Universidad de San Carlos de Guatemala; por decreto del Gobierno, pero en 1882, se tituló como Facultad dentro de esa institución y se separó de la Escuela Politécnica.[3]

De 1908 a 1918 la Facultad tuvo una existencia ficticia. El gobernante Manuel Estrada Cabrera reabrió la Universidad y a la Facultad de Ingeniería se le denominó Facultad de Matemáticas. En 1920 la Facultad reinició sus labores en el edificio que ocupó durante muchos años, frente al parque Morazán; hasta 1930 únicamente ofrecía la carrera de Ingeniero Topógrafo. En 1930 se reestructuraron los estudios y se reestableció la carrera de Ingeniería Civil. Este hecho marcó el inicio de la época "moderna" de esta Facultad.[4]

En 1959 se creó el Centro de Investigaciones de Ingeniería, para fomentar y coordinar la investigación científica con participación de varias instituciones públicas y privadas. En 1965 entró en funcionamiento el Centro de Cálculo Electrónico, dotado de computadoras y del equipo periférico para prestar servicio a catedráticos, investigadores y alumnos, quienes dispusieron de instrumentos para el estudio y aplicación de los métodos modernos de procesamiento de la información. Esto constituyó un logro importante a escala nacional y regional.[5]

En 1970 se creó la carrera de Ingeniería en Ciencias y Sistemas con grado de licenciatura.[6]

1.1.2. Misión

Desarrollar en el alumno las competencias que garanticen el éxito en la construcción del conocimiento a través de los diferentes estilos de aprendizaje y fomente la investigación permanente para permitir una mejor calidad de vida a la comunidad. Teniendo en cuenta las opciones del mercado actual en el país (logística, administración, tecnología de la información, finanzas, contabilidad, comercio, etc.), y también el mercado internacional, hacen que hoy en día exista una alta demanda global y competitividad.[7]

---

[2] Facultad de Ingeniería Universidad de San Carlos de Guatemala. *Antecedentes, Desarrollo CCI.* https://portal.ingenieria.usac.edu.gt/index.php/aspirante/antecedentes. Consulta: 1 de febrero de 2021.

[3] Ibíd.

[4] Ibíd.

[5] Ibíd.

[6] Ibíd.

[7] Escuela de Ciencias y Sistemas. *DTT-ECYS*. https://dtt-ecys.org/about_us. Consulta: 1 de febrero de 2021.

1.1.3. Visión

El estudiante de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala será reconocido como un profesional superior, en base al conocimiento incorporado en el currículo de estudios para capacitar a los estudiantes de manera integral, dándoles las herramientas adecuadas para su desarrollo profesional.[8]

1.1.4. Servicios que realiza

Formar, adecuadamente, los recursos humanos dentro del área técnico-científica que necesita el desarrollo de Guatemala, dentro del ambiente físico natural, social económico, antropológico y cultural del medio que lo rodea, para que pueda servir al país eficiente y eficazmente como profesional de la Ingeniería.[9]

Proporcionar al estudiante de Ingeniería en los diferentes niveles académicos, las facilidades y oportunidades necesarias para que obtenga tanto la formación básica que le sirva de fundamento para cualquier especialización técnico-científica, como conocimiento sobre tecnologías aplicadas al medio y con una mentalidad abierta a cualquier cambio y adaptación futura.[10]

Proporcionar al estudiante la suficiente formación científica general, en el conocimiento y aplicaciones de las ciencias físico-matemáticas y en tecnología moderna; en el sentido más amplio de la ingeniería, como la ciencia y arte de utilizar las propiedades de la materia y las fuentes de energía, para el dominio de la naturaleza, en beneficio del hombre.[11]

Estructurar una programación adecuada que cubra el conocimiento teórico y la aplicación de las disciplinas básicas de la ingeniería.[12]

Proporcionar al estudiante experiencia práctica de las situaciones problemáticas que encontrará en el ejercicio de su profesión.[13]

Capacitar a los profesionales para su auto-educación, una vez egresen de las aulas.[14]

Utilizar métodos de enseñanza-aprendizaje que estén en consonancia con el avance acelerado de la ciencia y la tecnología.[15]

---

[8] Escuela de Ciencias y Sistemas. *DTT-ECYS*. https://dtt-ecys.org/about_us. Consulta: 1 de febrero de 2021.
[9] Ibíd.
[10] Ibíd.
[11] Ibíd.
[12] Ibíd.
[13] Ibíd.
[14] Ibíd.
[15] Ibíd.

Fomentar la investigación y el desarrollo de la tecnología y las ciencias.[16]

Intensificar las relaciones con los sectores externos del país vinculados con las diversas ramas de la Ingeniería, no sólo con el fin de conocer mejor sus necesidades, sino para desarrollar una colaboración de mutuo beneficio.[17]

1.2. Descripción del problema

Actualmente la Escuela de Ingeniería en Ciencias y Sistemas cuenta con el Sistema de Control, del Proyecto de Desarrollo de Transferencia Tecnológica (Plataforma DTT), que le permite llevar la gestión de los estudiantes, catedráticos, cursos que se imparten, asignación de cursos, ingreso de notas, entre otros requerimientos que se han automatizado con el tiempo, para una mejor transparencia por medio de sus catedráticos y personal administrativo.

Aproximadamente en seis años, se han desarrollado diferentes funcionalidades por diversos estudiantes, lo que poco a poco ha ido generando más complejidad a un tema muy importante en la Ingeniería de Software, nos referimos al tema de Control de Calidad, mejor conocido como la fase de pruebas y validaciones, que se realizan a la plataforma previo a su despliegue en el ambiente de producción.

Actualmente estas pruebas se realizan de forma manual y genera un desgaste en los estudiantes que desarrollan las funcionalidades de la plataforma. Luego de validar que todo el sitio web funcione correctamente en su entorno local (equipo de computación donde trabajaron los requerimientos), esta nueva versión de software es publicada en un ambiente (más estable destinado para pruebas, al cual denominamos ambiente de control de calidad) y que junto con otras personas que apoyan se vuelve a validar la plataforma y si todo funciona

[16] Escuela de Ciencias y Sistemas. *DTT-ECYS*. https://dtt-ecys.org/about_us. Consulta: 1 de febrero de 2021.
[17] Ibíd.

correctamente, esta nueva versión se implementará finalmente en el ambiente de producción, esperando que no se haya introducido un bug (error) en funcionalidades ya existentes o en las nuevas que se han realizado.

El tiempo puede ser un factor crítico si no se estima correctamente en los cronogramas de trabajo de cada fase en los proyectos a implementar, ya que en muchas ocasiones es impredecible determinar cuánto se puede llevar validar toda la plataforma por las personas encargadas, ya que depende del conocimiento y tiempo que estas pruebas requieran en su realización. Sin olvidar que muchas veces no hay personal disponible para el apoyo de estas pruebas y en otras ocasiones se deba invertir tiempo en dar inducción a las personas para que conozcan la Plataforma DTT.

Actualmente los esfuerzos, por implementar en el ambiente de producción las nuevas funcionalidades de la plataforma, son muy altos por todos los involucrados, en especial cuando existen requerimientos desarrollados por diferentes estudiantes y que estos se deben de unir en un punto y ser validados por un grupo de personas para su aprobación. O esperar a que se implemente un requerimiento y repetir el ciclo de pruebas para la siguiente funcionalidad a desplegar. Lo que conlleva analizar el esfuerzo a realizar y el tiempo a invertir para lograr el fin deseado.

Gradualmente la complejidad de realizar las pruebas a la Plataforma DTT para las funcionalidades ya existentes y nuevas, va aumentando cada vez más y llegará el punto en que realizar esta gestión manualmente será tan difícil que volverá a la plataforma insostenible y no se le podrá brindar el mantenimiento requerido para su mejor funcionamiento, afectando a la Escuela en sus procesos y administración.

El recurso físico (hardware) que pueda disponer el estudiante al desarrollar puede influir en los tiempos de desarrollo y control de calidad de la plataforma, ya que no es lo mismo montar un sitio web y su base de datos en un servidor dedicado a esta finalidad que en una computadora local. Lo que genera en la mayoría de las veces un alto consumo de recursos del computador provocando que la aplicación sea inestable para las pruebas y el desarrollo de nuevas funcionalidades, llevando al estudiante tener que buscar alternativas adicionales para implementar sus requerimientos desarrollados, esperando que en la fase de control de calidad realizada por otras personas todo funcione bien o en el ambiente de producción no existan problemas después de publicar dichos cambios realizados.

Y nos encontramos que muchas veces el desarrollador necesita validar una funcionalidad en particular y para ello debe realizar todos los pasos necesario para que desde la opción de Ingreso (inicio de sesión) a la plataforma, deba recorrer por todo el menú hasta llegar a la funcionalidad en específico y poder validar en forma general el requerimiento, cuando lo que se requiere es validar un método en particular o bien la funcionalidad de la integración de algunas clases que se utilizan para determinadas tareas que se deben ejecutar, para que el requerimiento que se desarrolla cumpla con los objetivos deseados.

Validar la plataforma DTT de la Escuela, conlleva muchos esfuerzos que son necesarios para evitar que se introduzca algún error. Sabiendo que el soporte de la plataforma no es de 24/7 (24 horas los 7 días a la semana), el estudiante o usuario que utiliza la plataforma debe tener la mejor experiencia de uso, principalmente en cuanto a funcionalidad se refiere y evitar inconvenientes administrativos que le puedan generar más trabajo al personal de la Escuela.

Y por ello se debe enfocar los esfuerzos a mejorar la etapa de control de calidad sobre la plataforma, que permitirá reducir los riesgos de introducir errores en las funcionalidades actuales y futuras. Con ello se promueve una mejor imagen ante los usuarios de una aplicación de buena calidad y la confiabilidad de realizar las gestiones administrativas de forma óptima.

1.2.1. Resumen

La fase de Control de Calidad que se realiza a la Plataforma DTT de la Escuela, es realizada en conjunto por diversas personas de forma manual y depende del tiempo de estas personas y del conocimiento que puedan tener de la plataforma.

Esperando que se puedan cubrir la mayor cantidad de escenarios y funcionalidades, para no introducir un error al momento de implementar en producción los nuevos cambios. Teniendo en cuenta que se puede convertir en un círculo de nunca terminar en estas fases de control de calidad y la de desarrollo, por los cambios a realizar ante los errores que se identifiquen.

El estudiante debe validar que sus cambios funcionan correctamente, lo que conlleva a estar realizando pruebas a la plataforma, para evitar que lo desarrollado no afecte alguna de las funcionalidades ya existentes, generando un desgaste al estudiante por la complejidad y el tiempo que se debe invertir para esta tarea recurrente durante el proyecto. Lo que en muchas ocasiones genera un atraso en el proyecto por no planificar correctamente las etapas de pruebas (control de calidad) del proyecto.

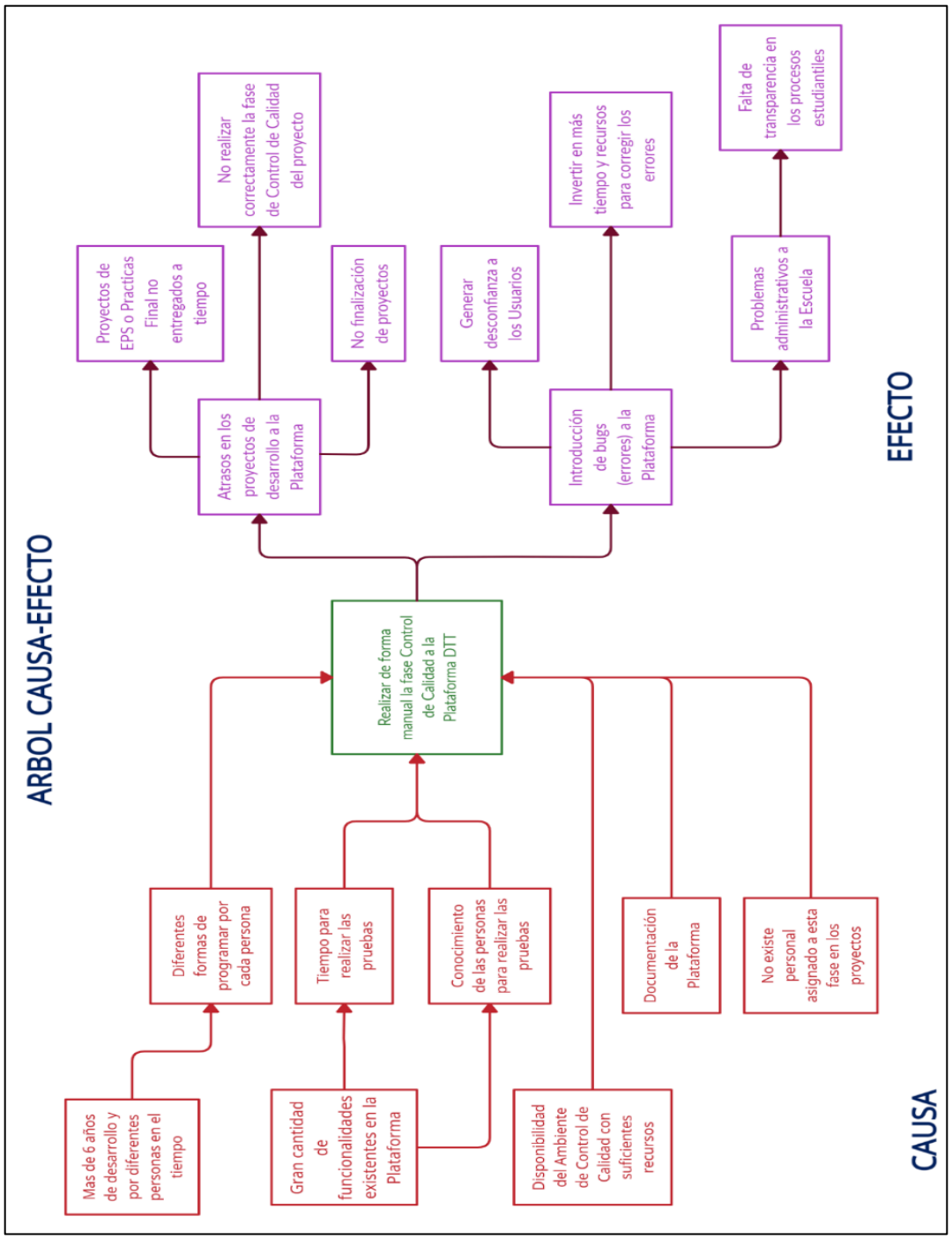
Realizar estas validaciones solo a las funcionalidades conocidas, dejando sin validar otras, por la premura del tiempo que se tiene, incluso llegando a validar

la aplicación como comúnmente se dice “para salir del paso” y no realizar correctamente, son algunas de las medidas que se toman para cumplir con el proyecto, olvidando que lo importante para el usuario, es la confiabilidad que una herramienta le brinda para sus gestiones.

Por lo cual, es importante tomar el tiempo necesario y realizar las acciones que correspondan para lograr reducir estas potenciales amenazas. Poniendo énfasis sobre la fase de control de calidad de la plataforma DTT y así mitigar en los futuros proyectos a implementarse estas vulnerabilidades.

La siguiente imagen, nos detalla de mejor forma, los impactos que pueden ocasionar, si la fase de Control de Calidad se mantiene realizando de la misma forma en que se ha venido trabajando y que estamos a tiempo para empezar a realizar cambios en este ciclo de desarrollo de software de la Plataforma DTT.

Figura 1. **Árbol Causa Efecto**



Fuente: elaboración propia.

1.3. Priorización de las necesidades

Para definir de mejor forma las necesidades identificadas, a continuación, se presentan dos tablas sobre el análisis al problema identificado: realización de forma manual la fase Control de Calidad a la Plataforma DTT.

Tabla I. **Análisis de la problemática identificada en tiempo y personas**

Necesidades para solucionar	Factores que los producen
Tiempo para realizar las pruebas.	No se planifica adecuadamente en los cronogramas de los proyectos. Demasiadas funcionalidades para validar y no se tienen métricas de tiempo que se necesita para invertir en cada prueba. En muchas ocasiones, se sufren atrasos en otras fases del proyecto, lo que genera tener que sacrificar y reducir el tiempo asignado a la fase de pruebas.
Conocimiento de las personas para realizar pruebas.	Poco conocimiento sobre la fase de Control de Calidad y los diferentes tipos de pruebas que existen y cuales realizar. El conocimiento sobre la plataforma DTT y todas sus funcionalidades existentes depende de una persona actualmente.
No contar con personas asignadas a esta fase en los proyectos.	No se planifica en los proyectos, el poder contar con personas adicionales para esta fase; y que con anticipación se pueda brindar capacitación tanto del producto como las pruebas a realizar.

Fuente: elaboración propia.

Tabla II. **Análisis de la problemática identificada en otros factores**

Necesidades para solucionar	**Factores que los producen**
Documentación de la Plataforma.	No contar con mucha documentación. Y en otras ocasiones muy compleja y extendida lo que genera que no sea utilizada.
Disponibilidad de un ambiente para Control de Calidad con suficientes recursos.	No contar con recursos adecuados para realizar pruebas a gran escala. Al ser varios proyectos en ejecución, es complejo la publicación de las nuevas funcionalidades.
Diferentes formas de programar por cada estudiante.	Más de 6 años de desarrollo, muchos estudiantes han programado de diferente forma, algunos con buenas prácticas, y otros con criterios mal aplicados. Falta de una guía de estandarización en la fase de programación. No aplicar el uso de buenas prácticas como realizar pruebas unitarias en el código fuente. Actualmente es muy difícil de entender varios módulos (código fuente) y genera mucha expectativa de error si se quiere realizar algún cambio en estos. La base de datos del proyecto es muy compleja de entender.

Fuente: elaboración propia.

Podemos observar que la fase de Control de Calidad en la ejecución de proyectos de la Escuela para la Plataforma DTT, se encuentra actualmente en un punto muy desgastante y con mucha inversión de tiempo, por todos los interesados, lo que genera que se vuelva poco a poco más difícil de mantener, repercutiendo directamente en la implementación de nuevas funcionalidades.

A continuación, presentamos una matriz para identificar de mejor forma, la priorización de las necesidades a cubrir en la implementación de una solución viable y factible.

Tabla III. **Matriz de priorización**

Necesidades	**Factores a**			
	Tiempo	**Conocimiento**	**Recurso Humano**	**Recurso Financiero**
Tiempo para realizar pruebas.	Varios días	En pruebas y la Plataforma DTT.	Varias personas	N/A
Conocimiento de las personas para realizar pruebas.	Varios días	Tipos de pruebas y cantidad de las funcionalidades en la Plataforma.	Varias personas	N/A
Contar con personas asignadas a la fase de pruebas en los proyectos.	Varios días	En pruebas y la Plataforma DTT.	Varias personas	N/A
Disponibilidad de un ambiente para Control de Calidad.	Varios días	Dependencia a una persona para realizarla	1 persona	$18 mensuales Costo del Hosting
Diferentes formas de programar por cada estudiante.	Meses	Según su experiencia.	1 o N personas según el proyecto	N/A

Fuente: elaboración propia.

Con base a esta matriz, podemos determinar que las principales necesidades a solucionar en la problemática están enfocadas en:

- Tiempo que se emplea en la fase de control de calidad.
- Conocimiento que se tiene para realizar pruebas a la Plataforma DTT.
- Cantidad de personas involucradas durante la fase de control de calidad de cada proyecto.
- Disponibilidad de un ambiente para control de calidad en todo momento.
- Cambiar los lineamientos para estandarizar la programación en el desarrollo de la Plataforma DTT.

2. FASE TÉCNICO PROFESIONAL

En este capítulo se desarrolla la solución propuesta, así como los costos del proyecto y beneficios que se obtendrán.

2.1. Descripción del proyecto

Derivado de los problemas identificados anteriormente, se ha realizado un análisis a estos, llegando a identificar diversas soluciones que permitan cumplir con la finalidad de la Escuela, el cual es brindar a los usuarios una plataforma estable, que minimice la mayor cantidad de errores en su funcionamiento, mitigar las potenciales amenazas que impidan un buen desempeño en la gestión administrativa de la Escuela.

2.1.1. Soluciones que se proponen implementar

Implementar una arquitectura de pruebas automáticas sobre Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas, pudiendo contar con una herramienta para la ejecución, notificación y reportería de estas pruebas.

Esta arquitectura por implementar se realizará en un contenedor configurado por medio de archivos de configuración, lo que permitirá que esta se pueda desplegar en cualquier computadora (según los requerimientos mínimos) y pueda ser utilizada por las personas encargadas que la Escuela asigne. Para ello se tiene contemplado que los archivos de configuración estarán almacenados en un repositorio para su versiona miento y fácil acceso para las personas que requieran implementar esta arquitectura.

Así mismo, también se va a configurar un repositorio, para desarrollar las pruebas automáticas de la plataforma DTT de la Escuela. Este entorno contará con las herramientas para el desarrollo y ejecución de las pruebas de una forma local (según los requerimientos mínimos) y será versionado tanto a nivel de cogido de las pruebas a desarrollar, como los archivos de configuración del entorno de trabajo a construir, cada uno en repositorios diferentes, para facilitar el uso de futuros estudiantes que deseen enriquecer más las pruebas automáticas de la Plataforma DTT o bien el mantenimiento que se le brindará a este proyecto.

También se implementará las primeras pruebas unitarias y de integración al proyecto de la Plataforma DTT de la Escuela, en otras palabras, se clonará el proyecto donde está el código fuente de la Plataforma DTT, y se creará una rama (branch) del repositorio de versiona miento, para desarrollar pruebas unitarias y de integración al código ya desarrollado. El objetivo de estas pruebas es ser una plantilla o una guía para los estudiantes que desarrollan en el proyecto de la Plataforma DTT y con ello puedan incorporar este tipo de pruebas al proyecto tanto en las funcionalidades ya existentes, así como las nuevas por desarrollar.

Cabe aclarar que esta arquitectura es el inicio a la automatización de pruebas, y este proyecto contempla ciertas pruebas en principales funcionalidades de la plataforma, pero no cubrirá todas las pruebas que actualmente la plataforma necesita, siendo este proyecto la arquitectura (configuración de herramientas a utilizar) y la guía necesaria para que futuros proyectos contemplen la automatización de más pruebas y con ello ir minimizando las validaciones manuales.

Estás implementaciones se realizarán en contenedores por medio de archivos de configuración y también el uso de repositorios de versiona miento.

Cada uno de estos cuenta con requerimientos mínimos para disponer de un mejor rendimiento, y las herramientas son de licenciamiento libre, permitiendo que la solución a construir no genere costos en su implementación y mantenimiento.

2.1.2. Resumen

La solución está enfocada en realizar cuatro productos:

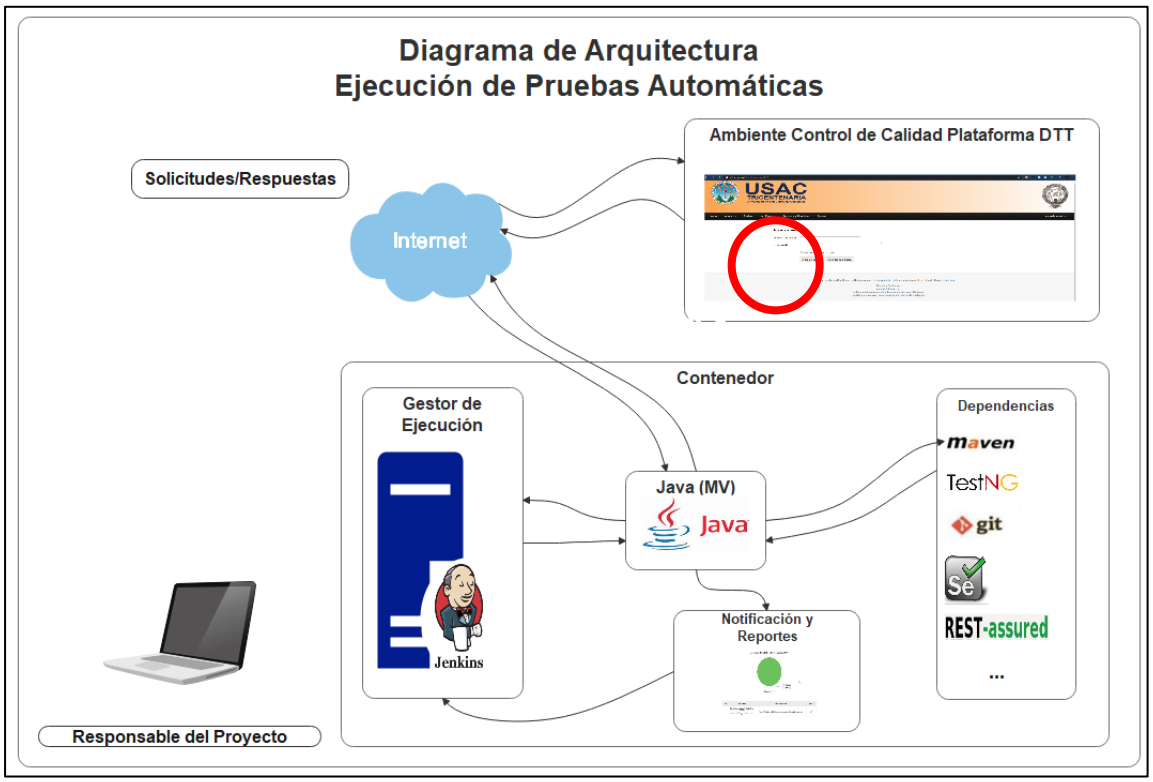
- Un contenedor que se pueda implementar en un equipo de computación (por medio de archivos de configuración), enfocado a la ejecución, notificación y reportería de pruebas automáticas.

- Un repositorio que se pueda implementar en un equipo de computación, enfocado al entorno de desarrollo de las pruebas y su ejecución.

- Construcción del proyecto (en su repositorio de versiona miento), donde se desarrollarán las pruebas automáticas (código fuente) de las diferentes validaciones que se implementarán.

- Un proyecto (rama o branch del repositorio de versiona miento donde se almacena el código fuente proyecto de la Plataforma DTT), donde se desarrollarán las diferentes pruebas unitarias y de integración a las principales clases y métodos del código fuente de la plataforma DTT.

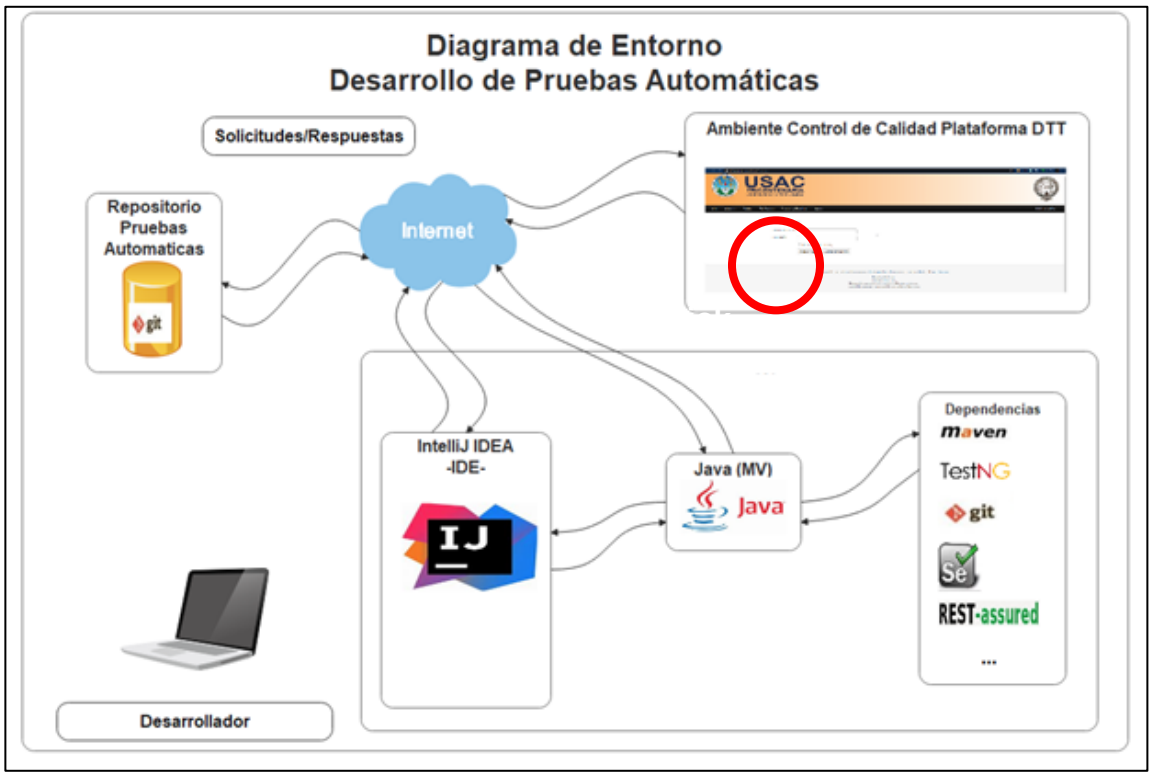
Figura 2. **Diagrama para la ejecución y notificación de pruebas**



Fuente: elaboración propia.

Este diagrama representa el principal objetivo del proyecto, siendo tener un entorno para la ejecución de pruebas automáticas, las cuales serán gestionadas por medio de la herramienta de Jenkins, dicha herramienta será configurada para la ejecución y notificación de las pruebas automáticas a construirse, para la Plataforma DTT de la Escuela. Tomando en cuenta que las pruebas se realizaran al entorno o ambiente de control de calidad.

Figura 3. **Diagrama del entorno de desarrollo de pruebas**



Fuente: elaboración propia.

Este diagrama representa el entorno de desarrollo (local) para la construcción de pruebas automáticas. Este entorno será configurado por medio de algunas herramientas (necesarias en su instalación) y la utilización de un repositorio de versionado de software. Así mismo las pruebas pueden ejecutarse hacia la Plataforma DTT en su ambiente de control de calidad o bien a un ambiente local (siendo en este caso, necesario tener que levantar el ambiente de la Plataforma DTT de forma local).

2.2. Investigación preliminar para la solución del proyecto

Para dar solución al problema detectado, se realizarán diversas acciones con la finalidad de poder implementar una opción viable para la Escuela en cuanto a costo financiero y mantenimiento a largo plazo, siendo estas:

- Como primer análisis, se espera poder implementar estas soluciones en contenedores (utilizando la herramienta de Docker), a través de archivos de configuración (DockerFile), lo que nos permitirá poder implementar esta arquitectura en cualquier computadora (que tenga los requerimientos mínimos para su ejecución).

- Se prevé el uso de GitLab (herramienta en la nube) como repositorio de versionado de software para los diferentes archivos a utilizar, esto nos ayudará a que si existe algún inconveniente en el camino con el equipo de cómputo utilizado para el proyecto, se pueda clonar una copia del proyecto nuevamente, si en determinado caso se desconfiguran los contenedores creados o el sistema operativo utilizado en peor de los casos, por las instalaciones y configuración de las herramientas a utilizar en esta arquitectura a implementar.

- Una de las vulnerabilidades es el acceso a internet, para el uso de la plataforma DTT y la creación de pruebas automáticas, quedando como alternativa el tener que implementar el contenedor y montar todo el entorno de ejecución de la plataforma de forma local en el equipo de cómputo, si fuera necesario y con ello mitigar el acceso a esta por medio de internet.

- El montar la plataforma DTT de forma local, es otra forma de evitar el riesgo de no tener un entorno de la plataforma de forma estable en el

ambiente de control de calidad para las pruebas que se van a automatizar. Derivado a que existen otros proyectos sobre la plataforma, puede darse el escenario que el sitio web esté fuera de línea por un tiempo y para evitar el atraso del proyecto.

- Otro detalle para tomar en cuenta es el lenguaje de desarrollo en el que está implementado la plataforma DTT, ya que actualmente se tiene poca experiencia en este lenguaje, el cual puede generar atrasos en la planificación del proyecto si la comprensión del código fuente se vuelve complejo.

- Las herramientas por utilizar todas son de licencia libre, lo que nos permite no tener costos en el uso de este software.

- Para el desarrollo de las pruebas automáticas, se utilizará el lenguaje de programación Java, este aparte de ser open source, es uno de los lenguajes que más librerías tiene para la gestión, construcción y ejecución de pruebas automáticas, por lo cual nos inclinados para su uso. Dentro de las librerías principales en la construcción de pruebas automáticas esta Selenium y TestNG, estos frameworks en su primer línea base de utilización es con Java, lo que nos da otro plus para poder utilizar dicho lenguaje.

- La otra herramienta importante en su utilización para este proyecto es Jenkins, esta herramienta también open sources, nos ofrece una gran cantidad de plugins con los que puede interactuar, siento entre estos nuevamente Selenium, TestNG, Maven, Java, entre otros. Herramientas que se pueden sincronizar entre ellas, lo que nos permite aprovechar de

forma óptima, las tecnologías actuales para el desarrollo de las pruebas automáticas.

- Actualmente el equipo de desarrollo de la Escuela, tienen ya una base y guía para el desarrollo, siendo utilizar GitLab como repositorio del código fuente, así mismo utilizar Docker para montar la infraestructura requerida en la ejecución de cualquier proyecto, por lo cual nos apegamos a estos lineamientos también.

2.3. Presentación de la solución al proyecto

La solución está enfocada en automatizar las pruebas durante la fase de control de calidad a la Plataforma DTT de la Escuela.

Por medio de la implementación contenedores que se puedan configurar en computadoras que cumplan con los requerimientos mínimos permitirá que las soluciones desarrolladas pueden ser portátiles en diferentes equipos de computación y sistema operativo. Estos archivos de configuración están en un repositorio de versiona miento en GitLab, para su acceso.

Pudiendo desplegar el entorno para la ejecución, notificación y reportería de pruebas automáticas para la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas y para ello se necesita una computadora que cuente con las siguientes características mínimas:

- 4 GB de Memoria RAM.
- 40 GB de Memoria de Disco Duro.
- Sistema Operativo Linux (Ubuntu) o Windows 10.
- Salida a Internet interfaz IPv4.

El contenedor cuenta con las siguientes herramientas configuradas:

- Instalación de Java y las herramientas para realizar pruebas automáticas: TestNG, Selenium, Maven, Chrome, Firefox, entre otras librerías.
- Instalación y configuración de Jenkins como herramienta de integración con los plugin necesarios para su administración.
- Configurar Jenkins para su ejecución por medio de archivos JenkinsFile.
- Instalación y configuración repositorio a utilizar (GitLab).
- Configuración de Jenkins para utilizar GitLab y clonar el código de las pruebas automáticas desarrolladas.
- Instalación y configuración de herramientas para la reportería de los resultados de las pruebas automáticas.
- Configurar de Jenkins para realizar las notificaciones.

En el ambiente local (computadora) con apoyo del repositorio de infraestructura de control de calidad (donde se almacenan las librerías necesarias) para implementar el entorno de trabajo y el desarrollo de las pruebas automáticas de la plataforma (DTT) de la Escuela, se necesita configurar lo siguiente:

Primero se necesita una computadora que cuente con las siguientes características mínimas:

- 4 GB de Memoria RAM.
- 40 GB de Memoria de Disco Duro.
- Sistema Operativo Linux (Ubuntu).
- Salida a Internet interfaz IPv4.

El entorno local por configurar requiere de las siguientes herramientas para su funcionamiento:

- Instalación de Java.
- Instalación de Maven.
- Instalación de IntelliJ Idea como herramienta de desarrollo y ejecución de las pruebas automáticas.
- Instalación de Git y configuración de repositorio a utilizar (GitLab).

Para el desarrollo de las pruebas automáticas, se utilizará GitLab como repositorio de versiona miento para el código a generarse. Se utilizará Java como el lenguaje de programación principal y de Maven para la gestión de librerías que el proyecto necesita para su funcionamiento.

Con relación a la priorización de las pruebas automáticas a desarrollar, estas se enfocan en las funcionalidades más utilizados actualmente y validadas por la Escuela para su planificación en este proyecto.

Las pruebas unitarias y de integración, será un producto por desarrollarse en el repositorio de versionado de software que actualmente está implementado el proyecto de la Plataforma DTT de la Escuela. Se realizará una clonación del proyecto en un equipo de computación y se levantará la infraestructura requerida para ejecutar de forma local la Plataforma DTT. Se desarrollarán las diferentes pruebas requeridas, que serán la plantilla o guía para futuras pruebas de este tipo, esto en una rama (branch) destinada solo para la construcción de este tipo de pruebas (unitarias), facilitando el acceso a este código a los estudiantes que desarrollarán en el proyecto de la plataforma del DTT.

2.4. Descripción de los Productos

Como se describió anteriormente, la solución a implementar se realizará en cuatro productos principales, los cuales describimos a continuación:

2.4.1. Configurar una arquitectura para la ejecución, notificación y reportería de pruebas automáticas a la plataforma DTT de la Escuela

Una arquitectura de pruebas automáticas es un entorno (de software) configurado para la ejecución de diferentes instrucciones en el equipo de computación donde se desplegará dicha infraestructura.

Actualmente, la plataforma DTT de la Escuela, ha venido creciendo en sus funcionalidades, estas desarrolladas por diversos estudiantes, lo que ha generado que cada uno deba validar la aplicación con sus nuevos cambios, y después la plataforma pasa por un proceso de control de calidad por varias personas, dichas pruebas tratando de validar que todo funcione correctamente, esto previo a desplegar en producción las nuevas funcionalidades desarrolladas. Esta tarea de validar la plataforma conlleva el esfuerzo de varias personas, durante varios días y esperando que se puedan cubrir todos los requerimientos existentes bajo el conocimiento de los involucrados.

Con la configuración de esta arquitectura, que se puede implementar en cualquier computador (que cumpla con los requerimientos mínimos), se busca tener un entorno para la ejecución de las diferentes pruebas que se automatizan tanto en este proyecto como a futuro y sea la herramienta principal para la gestión o administración de las pruebas automáticas. Pruebas que validan la plataforma DTT de la Escuela, configurada en su entorno o ambiente de control de calidad.

Enfocada a ser utilizada por las personas encargadas de validar que la plataforma funcione correctamente, previo a realizar el despliegue de los cambios en el ambiente de producción.

Esta herramienta permitirá, notificar los resultados (reportes) de cada prueba automática que se ejecuta, pudiendo visualizar los escenarios que fueron aprobados o bien los que fallaron.

Para facilitar su implementación en otros equipos de computación y sistema operativo, se utilizarán archivos (DockerFile), los cuales estarán almacenados en un repositorio (GitLab) para su versiona miento.

2.4.2. Configurar el entorno de desarrollo para la programación de pruebas automáticas sobre la plataforma DTT de la Escuela

Un entorno de desarrollo es el espacio de trabajo (con las herramientas de software instaladas necesarias) para la programación y ejecución de las pruebas automáticas a realizar.

Se requiere previamente tener instalado Java, Git. Se utilizará GitLab como repositorio para poder descargar la versión de Maven a utilizar, facilitando el acceso para los estudiantes a estos archivos.

En esta infraestructura, será necesario tener un IDE para la de programación y ejecución de las pruebas automáticas, se sugiere IntelliJ Idea.

Así mismo, el entorno estará configurado para trabajar con GitLab y poder gestionar el repositorio de versiona miento donde se almacenará el código fuente del proyecto de pruebas automáticas.

También en este repositorio se desarrollarán, los diferentes archivos (JenkinsFile) que son utilizados por la herramienta de Jenkins, para la ejecución de las pruebas automáticas, siempre sincronizados a su correspondiente repositorio de versiona miento.

Este ambiente local está enfocado al uso de los estudiantes encargados de desarrollar las diferentes funcionalidades de la plataforma DTT de la Escuela, y los estudiantes que le darán mantenimiento a este proyecto de pruebas, y puedan validar que la plataforma funcione correctamente previo al despliegue de las funcionalidades en el ambiente de control de calidad.

2.4.3. Crear diversas pruebas automáticas enfocadas a las principales funcionales de la plataforma DTT de la Escuela

El objetivo principal aquí es que se inicie la automatización de pruebas para la plataforma DTT de la Escuela, lo que permitirá en un futuro que los proyectos que se desarrollan para esta plataforma deban incluir en sus requerimientos, la automatización de las pruebas a dichas funcionalidades a implementar.

Esto permitirá reducir los tiempos de desarrollo, principalmente cuando se realiza control de calidad a la aplicación, previo al despliegue de las funcionalidades en el ambiente de control de calidad y de producción. Así mismo, en cuanto más pruebas manuales se logren automatizar, menor es el riesgo de incluir un bug (error) en la plataforma.

A Continuación, describiremos las pruebas que se automatizará, en orden de prioridad para el desarrollo de estas:

- Rol Administrador
  - Carga masiva de practicantes.
  - Carga masiva de docentes.

- Rol Estudent:
  - Carga masiva de estudiantes.
  - Carga masiva de notas.
  - Generar solvencia de prácticas.

- Rol Academic:
  - Ingreso de solicitudes para hacer prácticas por parte de los estudiantes.

Con las pruebas de carga, el objetivo es validar que las cargas masivas se realicen de forma correcta, desde subir un archivo csv y los diferentes mensajes de respuesta que la plataforma genera (correctas o de error) según las validaciones que se tienen programadas. También lograr validar en base de datos que registros fueron cargados correctamente y la comunicación con los webs servicies que interactúan con estas funcionalidades están respondiendo un estado activo (estatus 200).

En relación con la generación de solvencias de prácticas, esta prueba tendrá un set de datos, para validar que solvencias deben generarse y cuales faltan unos requisitos y se despliegue el mensaje de error correspondiente.

La generación de solicitudes de prácticas, de igual forma tener un set de datos para para estudiantes que puedan generar su solicitud sin ningún inconveniente y otro donde se generen mensajes de error esperados de acuerdo con las validaciones que la plataforma realiza; y lograr validar en base de datos que los registros fueron guardados correctamente.

Para cada una de las pruebas se va a validar que las vistas al usuario respondan correctamente y no se queden en blanco ante cualquier error.

El desarrollo de estas pruebas, contarán con GitLab como repositorio de versiona miento para el código a generar y su fácil acceso por los estudiantes encargados de darle mantenimiento a este proyecto de pruebas.

Estas pruebas serán desarrolladas en el lenguaje de Java, como principal framework de desarrollo, pero contará con el uso de otros frameworks para su ejecución en los entornos descritos anteriormente.

Como indicamos anteriormente, los nuevos desarrollos sobre la plataforma DTT de la escuela, se recomienda que incluya sus pruebas automáticas (utilizando esta arquitectura a implementar) y con las funcionalidades existentes, que este proyecto no tome en cuenta para la automatización de sus pruebas, se recomienda que se cree un proyecto destinado solo para automatizar las pruebas faltantes que se realizan actualmente de forma manual, ya que al contar con esta arquitectura, los esfuerzos se centran en construir más pruebas automáticas.

2.4.4. Crear diversas pruebas unitarias y de integración al código fuente de la plataforma DTT de la Escuela

El objetivo en este producto es crear las guías necesarias para que en el futuro los estudiantes puedan implementar pruebas unitarias y de integración al código que se desarrolla de la plataforma DTT de la Escuela. Para ello se implementarán varias pruebas de tipo unitaria y de integración en componentes desarrollados anteriormente sobre las funcionalidades que la plataforma DTT utiliza actualmente.

Se requiere realizar una clonación del repositorio de versiona miento del proyecto donde está almacenado el código fuente de la plataforma DTT y crear una rama (branch) destinada para implementar este código de pruebas. Siendo el producto esta rama a crear y las diferentes pruebas a desarrollarse.

Las pruebas que se realizarán siempre serán avaladas por la Escuela, enfocándose principalmente en clases y métodos muy utilizados por los diferentes componentes de la plataforma y la complejidad que estos puedan tener.

Las pruebas unitarias por desarrollarse son:

- Prueba unitaria Carga masiva de estudiantes.
- Prueba unitaria Carga masiva de notas (Carné, Nota).
- Prueba unitaria Generar solvencia de prácticas.

Las pruebas unitarias por realizar serán un conjunto de pruebas a los principales métodos de cada clase que son utilizadas para programar la funcionalidad principal.

Las pruebas de integración a desarrollarse son:

- Prueba de Integración Carga masiva de estudiantes.
- Prueba de Integración Carga masiva de notas (Carnet, Nota).
- Prueba de Integración Generar solvencia de prácticas.

Las pruebas de integración a realizar serán un conjunto de pruebas enfocadas a las clases que interactúan entre sí para cumplir con la funcionalidad principal.

En este caso las pruebas a desarrollar (unitarias e integración) serán realizadas en el código fuente que la plataforma tiene y su entorno de ejecución será el IDE donde se programan dichas funcionalidades. Estas pruebas quedan a la vista del programador que desarrolla en la plataforma y cada vez que necesita realizar una validación del código que modificó, tenga a la mano la ejecución de estas pruebas y verificar que todo funciona correctamente ante sus cambios realizados.

2.5. Costos del proyecto

Como sabemos, los costos son aquellos gastos que se tienen en la implementación del proyecto y en esta sección determinamos de forma cuantitativa el uso de los recursos requeridos.

2.5.1. Recurso humano

Este recurso está relacionado a todas las personas que interactúan en el desarrollo del proyecto. A continuación, se detalla los principales actores que se requieren para la ejecución del proyecto y el tiempo de estas personas es cuantificado en el presupuesto del proyecto.

Tabla IV. **Recurso humano**

Recurso	Descripción
Asesores	Un (01) Asesor de EPS de la Escuela de Ingeniería en Ciencias y Sistemas - Ing. César Rolando Batz Saquimux.
	Un (01) Asesor Supervisor de la Institución (Escuela de Ingeniería en Ciencias y Sistemas) - Ing. Miguel Marín De León.
Desarrollo e Implementación	Un (01) estudiante con Pensum Cerrado de la carrera de Ingeniería en Ciencias y Sistemas, para el análisis, diseño y desarrollo de la solución tecnológica del proyecto – Estudiante Gustavo Alexander Orozco Tay.
	Personal que participa en el desarrollo del Proyecto de la Plataforma DTT de la Escuela de Ingeniería en Ciencias y Sistemas y sean asignados para la capacitación del proyecto a implementar. Por lo menos dos (02) personas.

Fuente: elaboración propia.

2.5.2. Recursos materiales

Nos referimos a todos los insumos que se requieren para realizar el proyecto. Para el desarrollo e implementación de la aplicación se requieren de los siguientes materiales, los cuales también son cuantificados en el presupuesto del proyecto.

Tabla V. **Recursos materiales**

Recurso	Descripción
Mobiliario y Equipo	Una (01) computadora (Laptop) configurada con las herramientas requeridas para el desarrollo del proyecto. Cabe mencionar que las herramientas a instalar son de licenciamiento libre, el cual no genera un costo para el proyecto.
	Una (01) impresora para la impresión de los diferentes manuales o documentos requeridos durante el desarrollo del proyecto.
	Un (01) escritorio de oficina y una (01) silla de oficina para el uso de la persona a desarrollar el proyecto.
Inmueble	Localidad (oficina de trabajo) donde se resguardarán los insumos, el mobiliario y equipo que se requiere para el desarrollo del proyecto, contando también con los servicios básicos de agua, luz, extracción de basura, servicio telefónico e internet.
Útiles de Oficia	Papel bond (hojas en blanco), engrapadora, grapas, sacabocado, gancho de folders, folders y demás material de oficina para los diferentes documentos requeridos en el desarrollo del proyecto.

Fuente: elaboración propia.

2.5.3. Costo del proyecto (Presupuesto)

A continuación, se presenta una estimación de los gastos del proyecto a desarrollarse durante el tiempo que dura el EPS, tres (03) meses.

Tabla VI. **Presupuesto gastos de inicio**

Recursos	Cantidad	Costo Unitario Q	Subtotal Q
Mobiliario & Equipos			
Laptop	01	7 500,00	7 500,00
Impresora	01	1 600,00	1 600,00
Escritorio oficina	01	600,00	600,00
Silla de oficina	01	300,00	300,00
		Subtotal	**Q 10 000,00**
Subtotal, diez mil quetzales exactos			

Fuente: elaboración propia.

Tabla VII. **Presupuesto gastos mensuales**

Recurso	Costo Mensual Q	Tiempo / Meses	Subtotal Q
Inmueble			
Alquiler Local	1 500,00	03	4 500,00
Consumibles			
Agua	75,00	03	225,00
Energía Eléctrica	200,00	03	600,00
Internet	350,00	03	1 050,00
Telefonía	100,00	03	300,00
Combustible	100,00	03	300,00
Útiles de Oficia	50,00	03	150,00
		Subtotal	**Q 7 125,00**
Subtotal, siete mil ciento veinticinco quetzales exactos			

Fuente: elaboración propia.

Tabla VIII. **Presupuesto recurso humano**

Recurso Humano	*Horas Trabajo	Costo / Hora Q	Subtotal Q
Asesor EPS	60	200,00	12 000,00
Estudiante EPS	252	120,00	30 240,00
		Subtotal	**Q 42 240,00**
Subtotal, cuarenta y dos mil doscientos cuarenta quetzales exactos			

Fuente: elaboración propia.

* Las horas de trabajo se estima en relación con el cronograma de actividades. (Apéndices).

Se estima un porcentaje en gastos imprevistos o no planificados.

Tabla IX. **Presupuesto total**

Gastos Calculados	Estimación Imprevistos	Subtotal Imprevistos	Total Presupuesto
Q 59 365,00	5 %	Q 2 968,25	**Q 62 333,25**
Total, sesenta y dos mil trescientos treinta y tres quetzales con veinticinco centavos.			

Fuente: elaboración propia.

Cabe aclarar que este presupuesto realizado solo es para tener un dato aproximado del valor que un proyecto de este alcance tendría si se contratara los servicios informáticos de un consultor, actualmente este proyecto es realizado sin ningún costo en solicitud al desarrollo como proyecto de Ejercicio Profesional Supervisado (EPS).

2.6. Beneficios del proyecto

La Escuela en Ingeniería en Ciencias y Sistemas ha tenido en los últimos años un rápido crecimiento en su población estudiantil y la demanda de los servicios electrónicos que la Escuela ofrece es de vital importancia para no generar atrasos u otros inconvenientes en su gestión administrativa.

Garantizar la funcionalidad de la Plataforma DTT, es uno de los tantos esfuerzos que la Escuela realiza con su equipo de trabajo, por ello iniciar la automatización de la fase de Control de Calidad de la Plataforma DTT es un objetivo a corto plazo que ayudara mucho en las acciones operativas que se tiene día a día, permitiendo mejorar los tiempos de implementación de nuevas funcionalidades que se requieren a la Plataforma DTT constantemente.

La Escuela de Ingeniería en Ciencias y Sistemas como ente rector de las soluciones tecnológicas, mantiene la iniciativa en implementar las mejores buenas prácticas para desarrollar productos confiables y seguros, siendo una guía para otras Escuelas y Facultades el automatizar la fase de control de calidad de sus proyectos de software, permitiendo a sus estudiantes conocer y llegar a innovar en otras áreas del desarrollo de software.

La Escuela en Ingeniería en Ciencias y Sistemas busca garantizar siempre que sus operaciones administrativas son lo más confiable y transparentes, y la mejor forma es por medio de la automatización de sus procesos.

Esto conlleva un gran esfuerzo de todos los involucrados en estas tareas, y el equipo de desarrollo de la Plataforma DTT es una pieza vital para alcanzar dichos objetivos.

Con la automatización de las pruebas nos permitirá mejorar el ciclo de desarrollo de software del proyecto de la Plataforma DTT, lo que permitirá realizar implementaciones más periódicas, reducir el riesgo de introducir un error a las funcionalidades actuales e implementar buenas prácticas de desarrollo que sirvan de guía a futuros proyectos.

Con la implementación de una arquitectura de pruebas, se busca reducir los tiempos para validar las principales funcionalidades de la Plataforma DTT esperando que estos tiempos se reduzcan en más de un 80 % del tiempo actual sobre estas funcionalidades.

Así mismo mejorar la calidad de productos o nuevas funcionalidades a implementar, permitiendo que el desarrollador se enfoque más en las funcionalidades de la Plataforma y no tanto en realizar pruebas a la misma.

Garantizar que las funcionalidades de la Plataforma DTT, sean confiables y seguras es uno de los principales objetivos del equipo, e implementar metodologías, buenas prácticas y estrategias para alcanzar esta finalidad es una acción que constantemente se realiza en la Escuela, para que sus procesos sean lo más transparente, seguro y confiable a la población estudiantil y autoridades.

Con la implementación de este proyecto, se busca iniciar la automatización de otros procesos en el ciclo de desarrollo de software, principalmente en la etapa de despliegue de la aplicación ya en su entorno de producción. Convirtiendo este proyecto en la guía inicial para alcanzar este fin.

3. ARQUITECTURA DE PRUEBAS AUTOMATIZADAS

A continuación, explicaremos la implementación de cada fase del proyecto, siendo importante mencionar que las herramientas que fueron utilizadas y configuradas, Para el entorno de desarrollo todo fue en un ambiente local (computadora personal), teniendo como finalidad de que esto mismo, pueda ser replicado por otra persona en su equipo de computación. Para el entorno de ejecución de las pruebas, se configuraron las herramientas tanto en un ambiente local como en un servidor proporcionados por la Escuela, para su fácil administración en la ejecución de las pruebas automatizadas.

3.1. Entorno de ejecución y reportería de pruebas automáticas

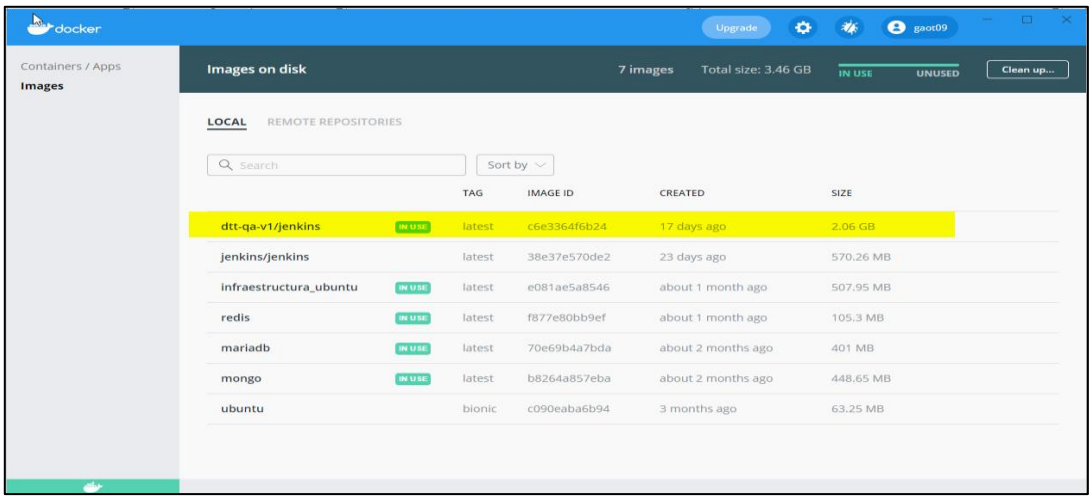
Como primer paso, realizado en el proyecto, fue tener un entorno para la ejecución de las pruebas automáticas. Para ello como requisito indispensable es tener instalado Docker y Git en la computadora.

En los apéndices, se detalla de forma más técnica, la construcción de la imagen y configuración del contenedor que utilizaremos para la ejecución de las pruebas automáticas.

En esta sección, nos enfocaremos a mostrar como quedo implementado Jenkins y la forma en ejecutar las pruebas desde este entorno.

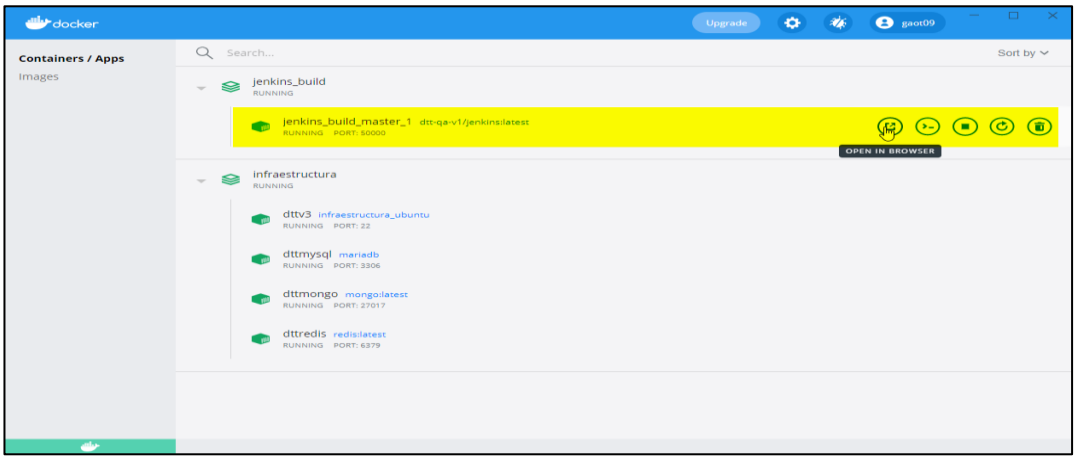
En la siguiente figura podemos visualizar la imagen que se creó para la ejecución de las pruebas automáticas de la Plataforma DTT. A partir de esta imagen, el siguiente paso a realizar es levantar el contenedor para su utilización.

Figura 4. **Visualización de imagen base por la interfaz de Docker**



Fuente: elaboración propia.

Figura 5. **Visualización del contenedor a usar por la interfaz de Docker**



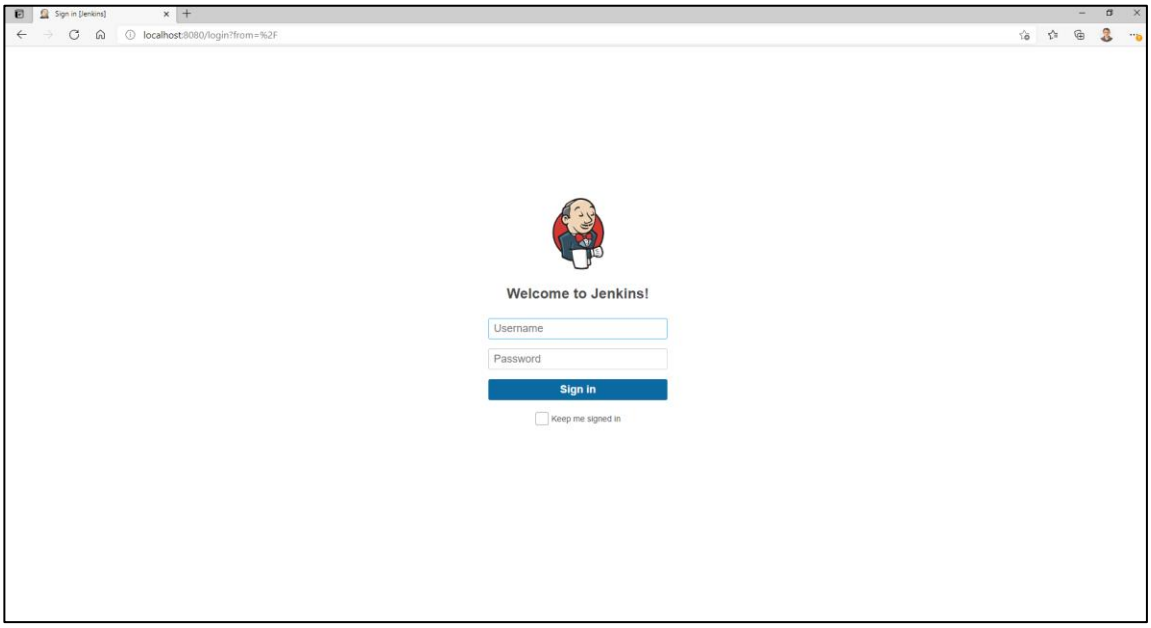
Fuente: elaboración propia.

Como podemos observar, por medio de la interfaz gráfica de Docker, nuestro contenedor está ejecutándose (estatus verde), para poder empezar a

trabajar en el tenemos habilitado el puerto 8080, lo que nos permitirá desde el navegador que tengamos configurado en nuestro equipo de computación poder utilizar Jenkins.

En la figura anterior, se puede observar la opción de abrir por el browser, al seleccionar la opción, se nos despliega en nuestro navegador lo que es Jenkins, este ya está configurado, por lo que nos pide automáticamente credenciales para ingresar. La siguiente figura muestra la pantalla de inicio de Jenkins, el cual ya está configurado, listo para su uso.

Figura 6. **Visualización de Jenkins desde el navegador**

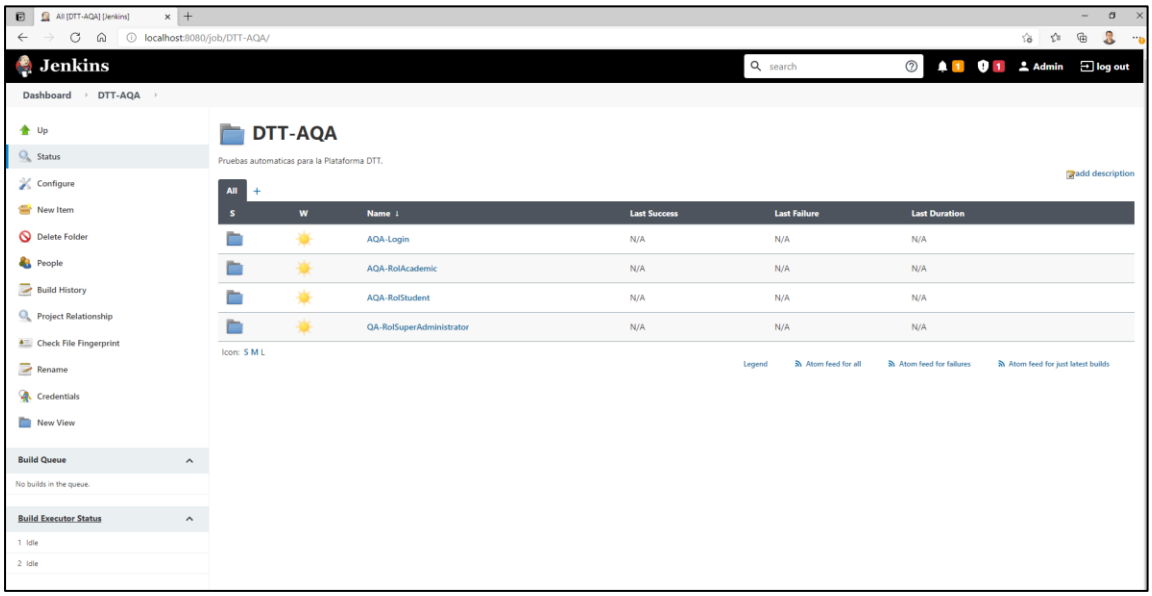


Fuente: elaboración propia.

En la siguiente imagen, podemos observar la configuración de las carpetas por cada rol que tiene la Plataforma DTT, dentro de cada una de estas están sus respectivas pruebas automáticas. Cabe mencionar, que tenemos una carpeta

inicial “AQA-Login”, esta contiene la prueba automática de iniciar sesión en la Plataforma DTT, esta prueba tiene como finalidad, poder validar en cualquier momento que la Plataforma DTT está activa y funcionando, adicional nos ayuda a establecer que tenemos conexión a base de datos al momento de realizar el inicio de sesión, también nos ayuda a comprobar que Jenkins se logra comunicar a la Plataforma y poder realizar las diferentes acciones para iniciar las pruebas automáticas.

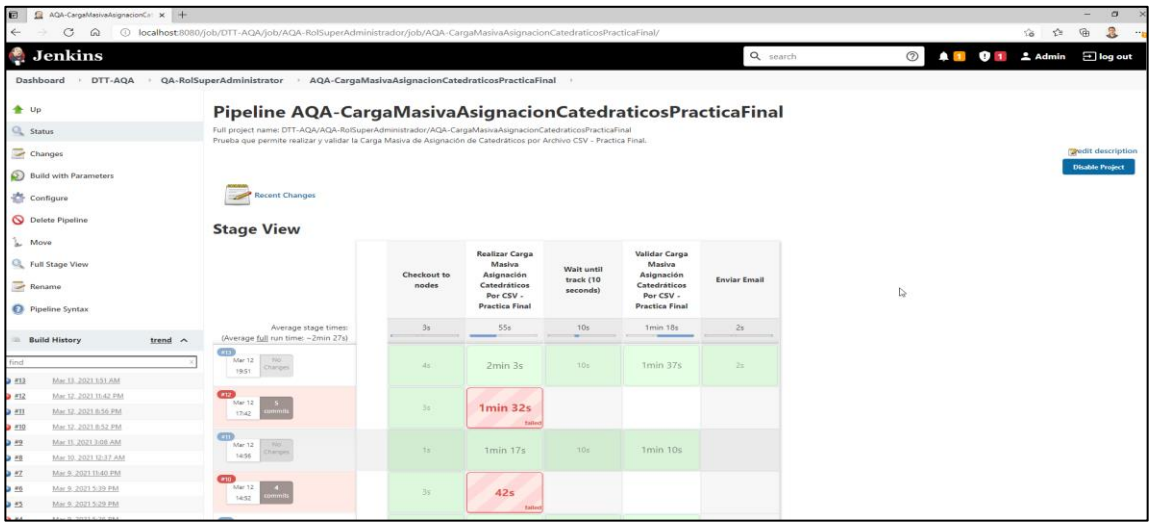
Figura 7. **Jenkins configurado para la ejecución de pruebas**



Fuente: elaboración propia.

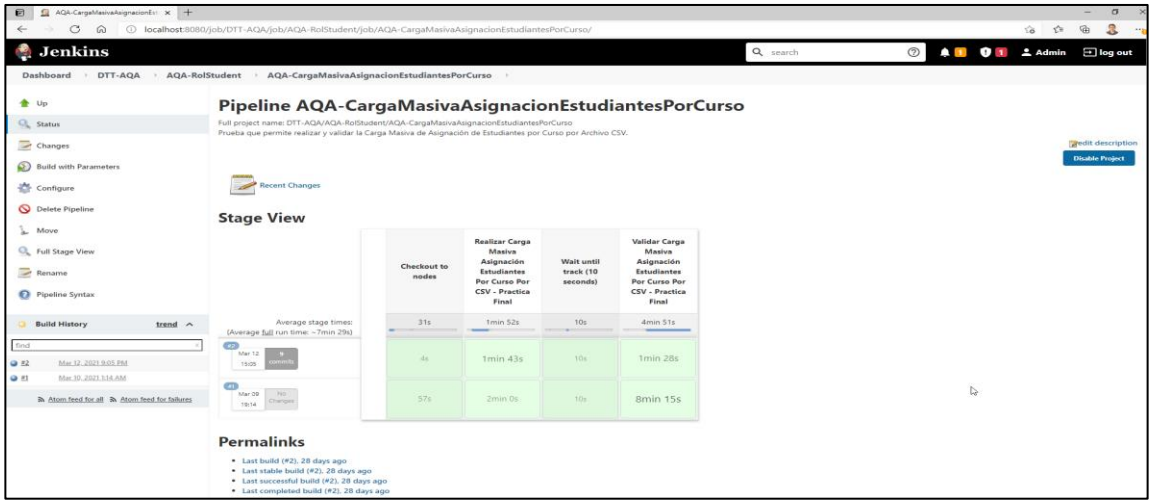
A continuación, se presentan varias imágenes, relacionadas a la ejecución de las pruebas automáticas que tenemos programadas en esta arquitectura.

Figura 8. **Ejemplo prueba automática para Rol Administrador**



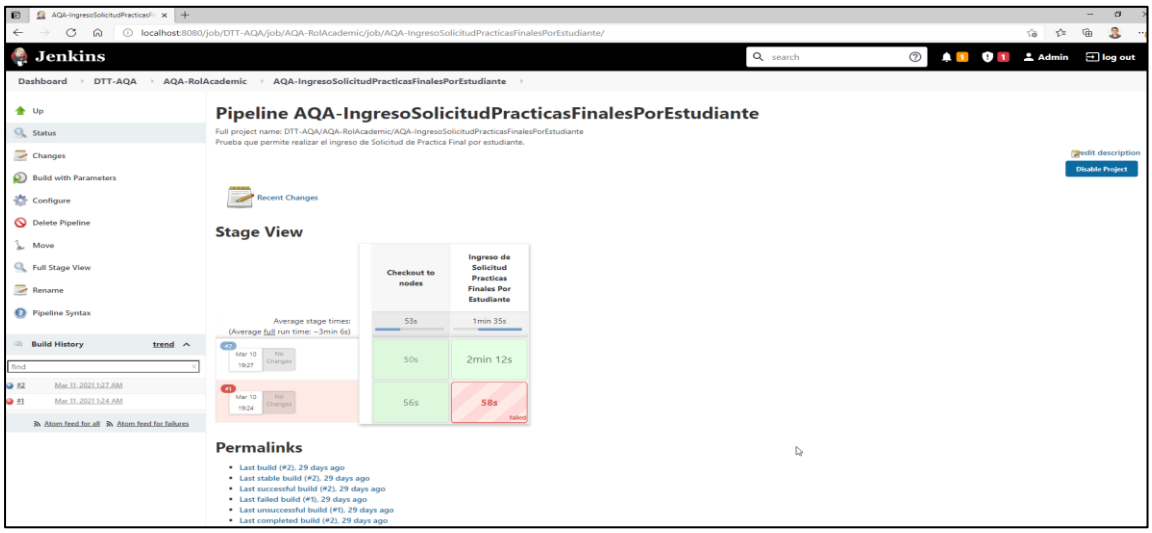
Fuente: elaboración propia.

Figura 9. **Ejemplo prueba automática para Rol Student**



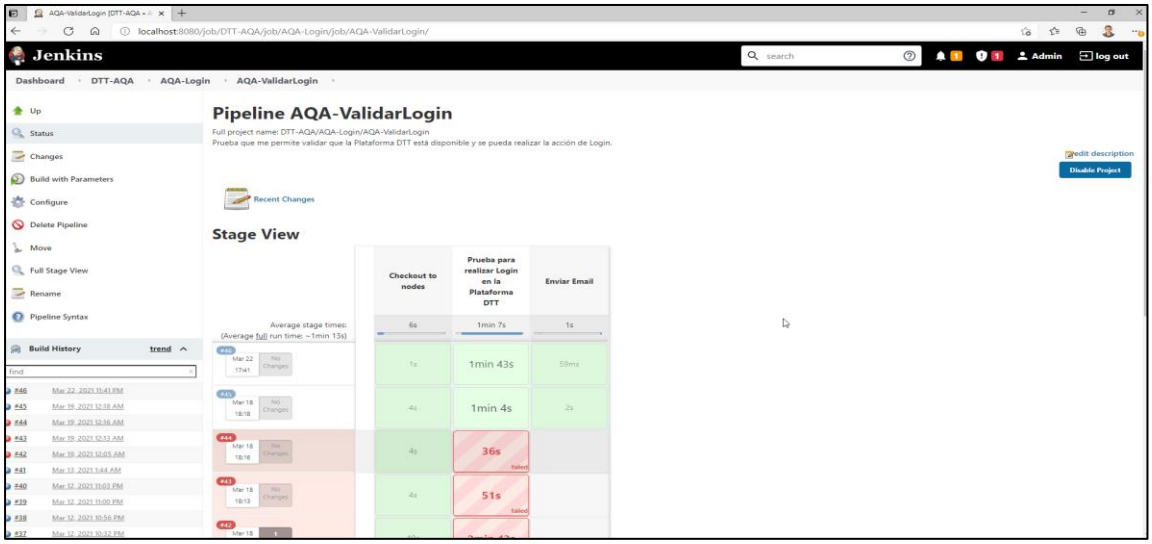
Fuente: elaboración propia.

Figura 10. **Ejemplo prueba automática para Rol Academic**



Fuente: elaboración propia.

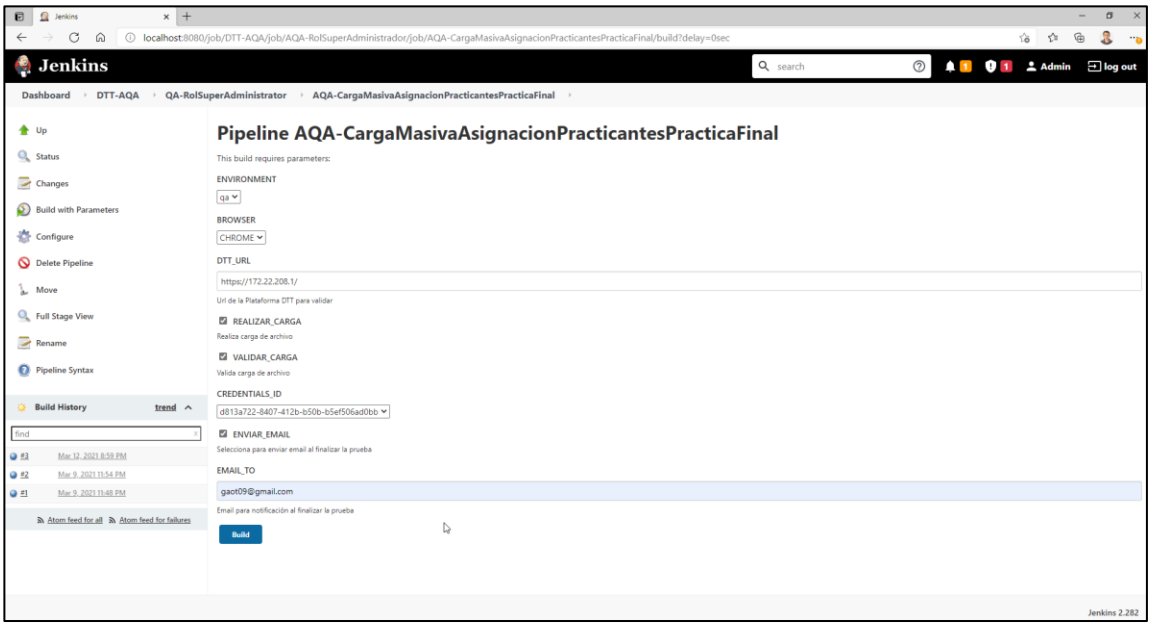
Figura 11. **Ejemplo prueba automática Inicio de Sesión (Login)**



Fuente: elaboración propia.

Como podemos observar en las imágenes, Jenkins nos ofrece un entorno de ejecución bastante practico, al momento de ejecutar cada prueba, se solicitan ciertos parámetros, los cuales nos ayudan a definir cierta configuración en la ejecución de estas pruebas, por ejemplo: la dirección URL al cual se debe apuntar para validar la Plataforma DTT. También si queremos ejecutar cada etapa de la prueba automática o bien solo alguna en particular; que explorador deseamos utilizar para la prueba y finalmente si deseamos recibir correo electrónico.

Figura 12. **Ejemplo de parámetros en la ejecución de una prueba**



Fuente: elaboración propia.

La ejecución de las pruebas automáticas permite enviar correo electrónico, para lo cual se creó el correo electrónico dtt.qa.jenkins@gmail.com. A partir de esta opción, podemos notificar al usuario cuando finaliza la ejecución de la prueba de forma exitosa.

3.2. Entorno de desarrollo de pruebas automáticas

El segundo producto en este proyecto es la configuración de un entorno de desarrollo para la creación y ejecución de pruebas automáticas, en otras palabras, pensando en las personas (desarrolladores) que continuaran con este proyecto o bien le darán mantenimiento, se requiere tener un ambiente montado en el equipo de computación local con el cual se pueda seguir construyendo más pruebas automáticas a la Plataforma DTT.

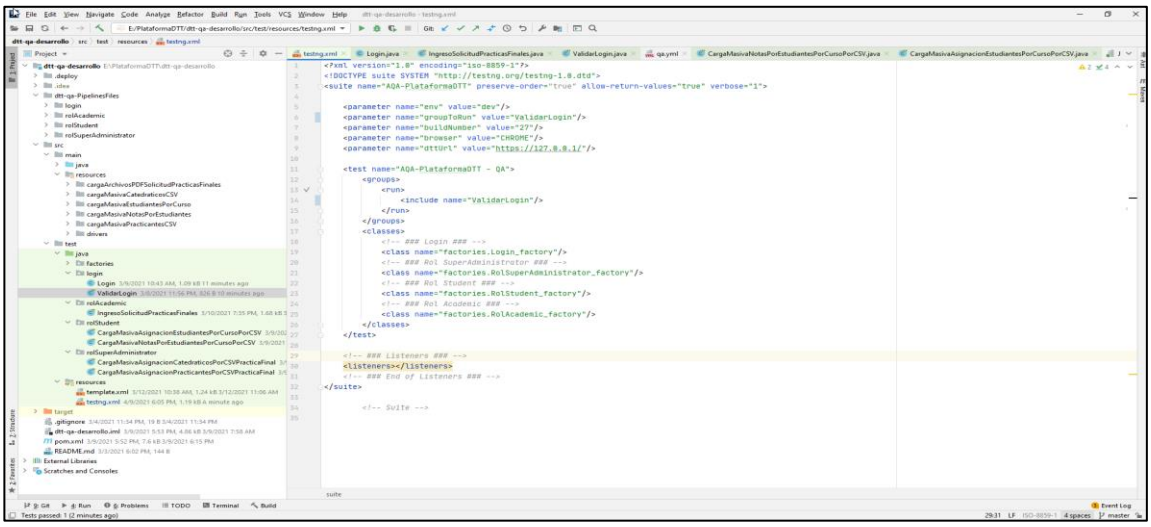
Como mencionamos en el apartado anterior, explicaremos a grandes rasgos como está compuesto este entorno y en el manual técnico se detalla cómo construir este ambiente en el equipo de computación local.

Como previos requisitos, necesitamos instalar Java, Git y Maven. Así mismo si no tenemos disponible el ambiente de control de calidad de la Plataforma DTT, se recomienda configurar y levantar localmente la Plataforma DTT (esta explicación queda fuera del alcance de este proyecto).

También algo importante es la utilización de un IDE para la programación, en este caso nos inclinamos y sugerimos utilizar IntelliJ Idea (versión comunity), ya que su entorno nos facilita hacer operaciones con Git, así como ejecutar las pruebas de una forma gráfica.

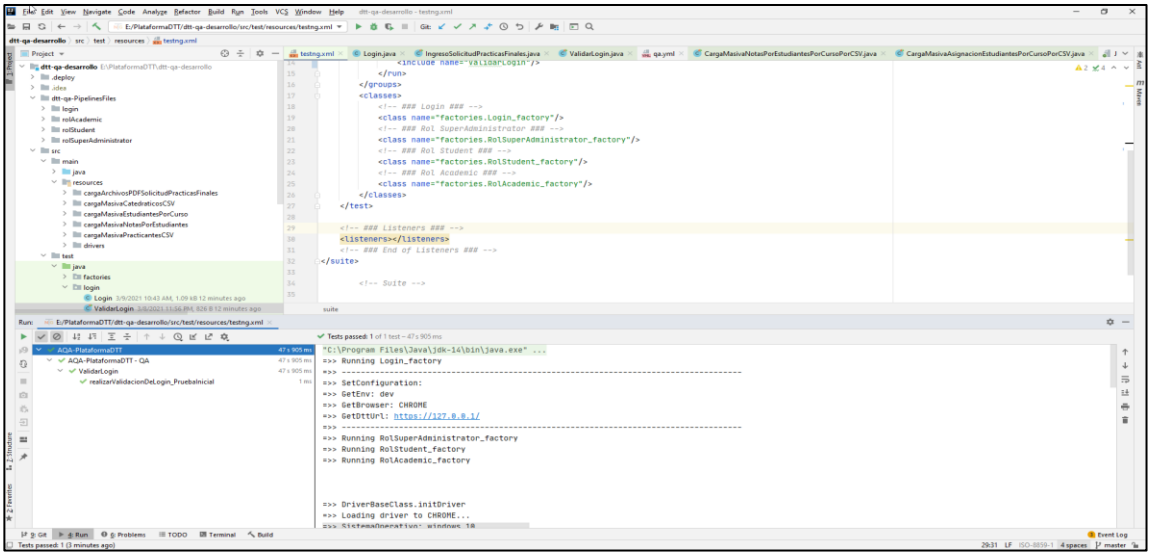
En las siguientes imágenes, podemos observar un ejemplo en la ejecución de una prueba automática, la de validar inicio de sesión (Login). Algo importante es tener también configurado el navegador Chrome o Firefox en su versión correspondiente (según manual técnico), para poder ejecutarse correctamente las pruebas automáticas.

Figura 13. **IDE para la ejecución de pruebas ambiente de desarrollo**



Fuente: elaboración propia.

Figura 14. **Ejecución de pruebas automática por el IDE**



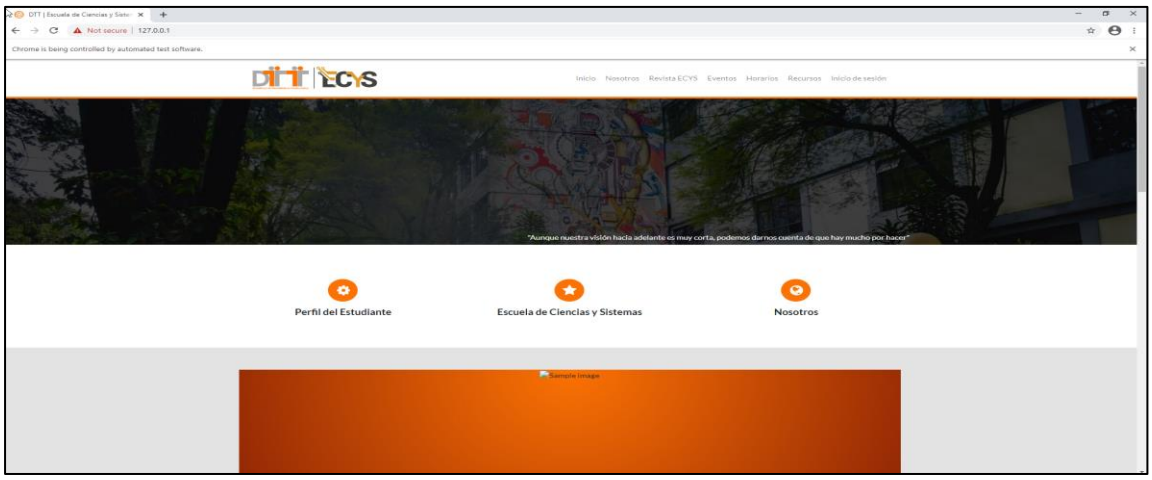
Fuente: elaboración propia.

La prueba automática inicia, levantando la página de la Plataforma DTT (en este ejemplo hicimos que apunte al ambiente local (https://127.0.0.1), por lo cual previamente levantamos la Plataforma en nuestro equipo de computación).

El explorador se abrió de forma automática y se redirecciono a la URL local de la plataforma DTT. Luego de un tiempo prudencial, de forma automática se seleccionó la opción de inicio de sesión y se ingresaron las credenciales correspondientes. Posteriormente es espero a que se lograra ingresar y se validan ciertos elementos que nos indican que se pudo realizar inicio de sesión de forma exitosa.

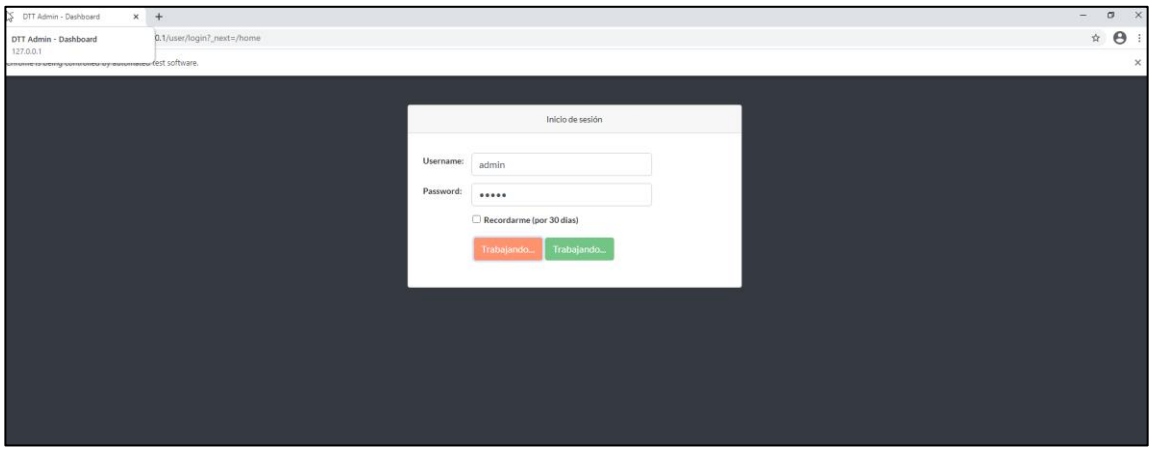
Las siguientes imágenes describen la ejecución de la prueba automática, cabe recalcar que, durante esta ejecución de la prueba, no se tuvo ninguna interacción por alguna persona y todo se realizó automáticamente, atreves de las líneas de código desarrolladas en el proyecto.

Figura 15. **Despliegue automático de la Plataforma DTT local**



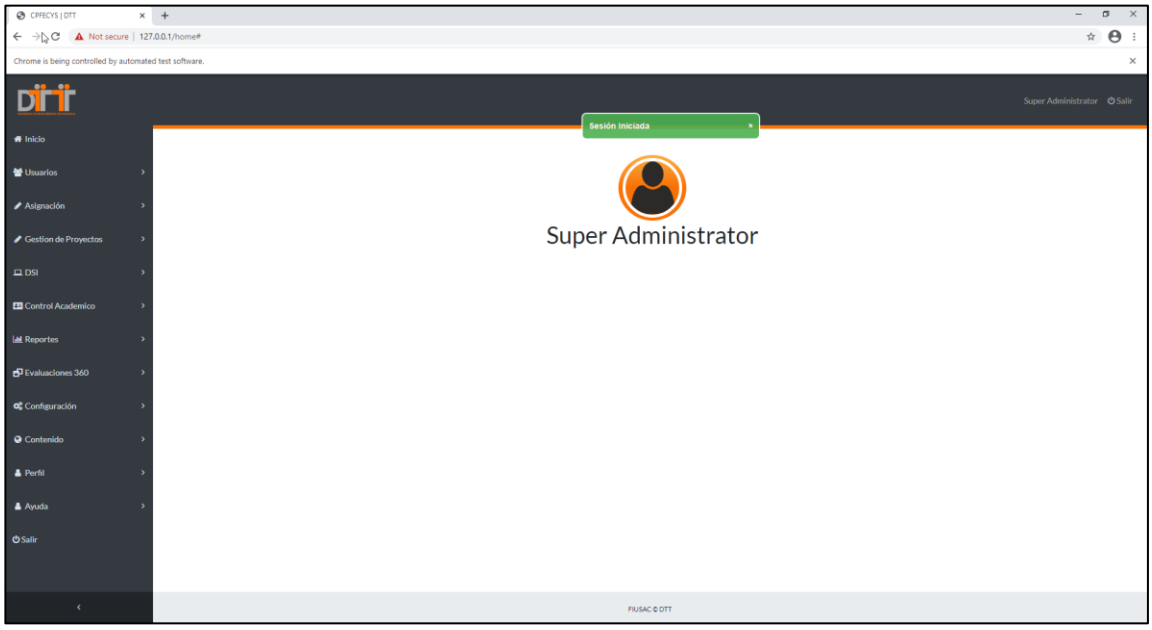
Fuente: elaboración propia.

Figura 16. **Ingreso de credenciales de forma automática**



Fuente: elaboración propia.

Figura 17. **Validación exitosa de inicio de sesión**



Fuente: elaboración propia.

A grandes rasgos, pudimos observar el comportamiento de esta prueba automática desde el entorno de desarrollo; de esta forma es como se ejecutan las demás pruebas, por ejemplo: para validar la carga masiva de practicantes, se realiza primero el proceso de inicio de sesión posterior de forma automática se ubica la opción en el menú y se navega por la página hasta llegar a la opción de la carga masiva a realizar. Luego se selecciona el archivo a cargar (en el entorno de desarrollo, se tiene un folder de recursos, en este se almacenan los diferentes archivos que necesitamos para las pruebas automáticas), posterior de seleccionar el archivo, se selección el botón de cargar y quedamos a esperas de la respuesta del servicio. Desplegamos los mensajes de error si existieran; luego tenemos otra validación que se encarga de revisar que los datos que subimos en el archivo se desplieguen ya en las tablas de información que la Plataforma DTT muestra. Para ello nuevamente se repite el proceso de inicio de sesión, posterior se navega en el menú hasta llegar a la opción correspondiente y luego de desplegarse la vista que solicitamos revisar, esperamos a que se carguen los datos en la tabla que la Plataforma DTT despliega, y luego validamos que se despliegue la información esperada. Al existir algún error, se finaliza la prueba y se muestran en consola los errores o fallos encontrados.

Como indicamos previamente, todo se realiza de forma automática, sin intervención de alguna persona (usuario), ya que la finalidad de estas pruebas es exactamente esto, realizarlas sin la necesidad de alguna persona intervenga en el proceso.

Este entorno de desarrollo está enfocado a que las personas que continuaran con este proyecto puedan ejecutar las pruebas sin necesidad de tener configurado Jenkins, y puedan visualizar gráficamente la interacción de las herramientas para ejecutar estas pruebas.

Las pruebas automáticas desarrolladas en este proyecto tienen como finalidad ser la base para la construcción de más pruebas a la Plataforma DTT.

Por lo cual, se han utilizado diferentes herramientas las cuales describimos brevemente a continuación:

- Selenium, a través de este framework, podemos interactuar con los navegadores (en este caso Chrome o Firefox) los cuales definimos en la construcción de esta arquitectura. Desde hacer que se abra el explorador a partir de una ruta URL, hasta poder ingresar valores a los controles de tipo texto, o bien presionar botones, combos de selección múltiple, buscar archivos, entre otras acciones, son ejemplos de lo que nos ofrece Selenium, permitiéndonos poder interactuar con casi todos los elementos que una página web utiliza. Para poder interactuar con estos controles o elementos, Selenium utiliza ciertas direcciones, la que utilizamos en el proyecto son las direcciones XPath, por medio de estas, podemos localizar cualquier control en la página web, siendo en este caso una dirección única para cada elemento el cual queremos manipular.

- TestNG: con este framework, se realiza la lógica de validación de pruebas, permitiéndonos ejecutar pruebas por grupos o bien por clases. Así mismo definir un mecanismo para la ejecución de cada prueba. También nos permite construir diferentes criterios para validar, entre estos, por ejemplo, validar variables de tipo entero, texto, decimal, entre otros.

- Maven: a través de este framework, gestionamos todas las dependencias o librerías que se requiere en la ejecución de las pruebas automáticas, lo que nos permite no tener que estar cargando cada librería en nuestro

proyecto y también no tener que estar guardando en nuestro repositorio estas librerías, lo que nos permite crear repositorios no tan pesados.

- Archivos JenkinsFile: por medio de estos archivos (que se denominan Pipeline), definimos la comunicación entre las pruebas automáticas (código fuente) y su ejecución en Jenkins (entorno de ejecución definido en la sección anterior). Estos archivos están bajo el lenguaje de Groovy y contiene las instrucciones de clonar desde el repositorio del código de las pruebas, ejecutar las pruebas según el grupo correspondiente y finalmente enviar el correo electrónico si se requiere. Todo esto se ejecuta desde el Job configurado en Jenkins el cual está configurado a que lea estos archivos para su ejecución.

- Java: siendo nuestro lenguaje de programación principal, Java nos ofrece la interacción con un sinfín de librerías, permitiéndonos aprovechar los recursos de mejor manera. Así mismo la programación orientada a objetos nos permite realizar código más claro para su mantenimiento, pudiendo aplicar buenas prácticas en el desarrollo de las pruebas automáticas.

- GitLab: por medio de esta herramienta basada en Git, llevamos los repositorios de versiona miento del proyecto, tanto para el código fuente, así como para las dependencias en la construcción de la arquitectura de las pruebas automáticas.

Como podemos observar, son varios los componentes que interactúan entre sí para la construcción de estas pruebas automáticas y su posterior ejecución. Lo que nos permite que esta arquitectura pueda ir creciendo y actualizándose.

3.3. Automatización de pruebas funcionales de regresión

En la clasificación de pruebas, podemos mencionar que las pruebas funcionales de regresión en este proyecto son todas aquellas que se realizan a la Plataforma DTT de forma actual a través del sitio web, todas las funcionalidades existentes que se validan de forma manual (usuario que interactúa con el sitio para ir probando que cada funcionalidad en el menú funcione correctamente).

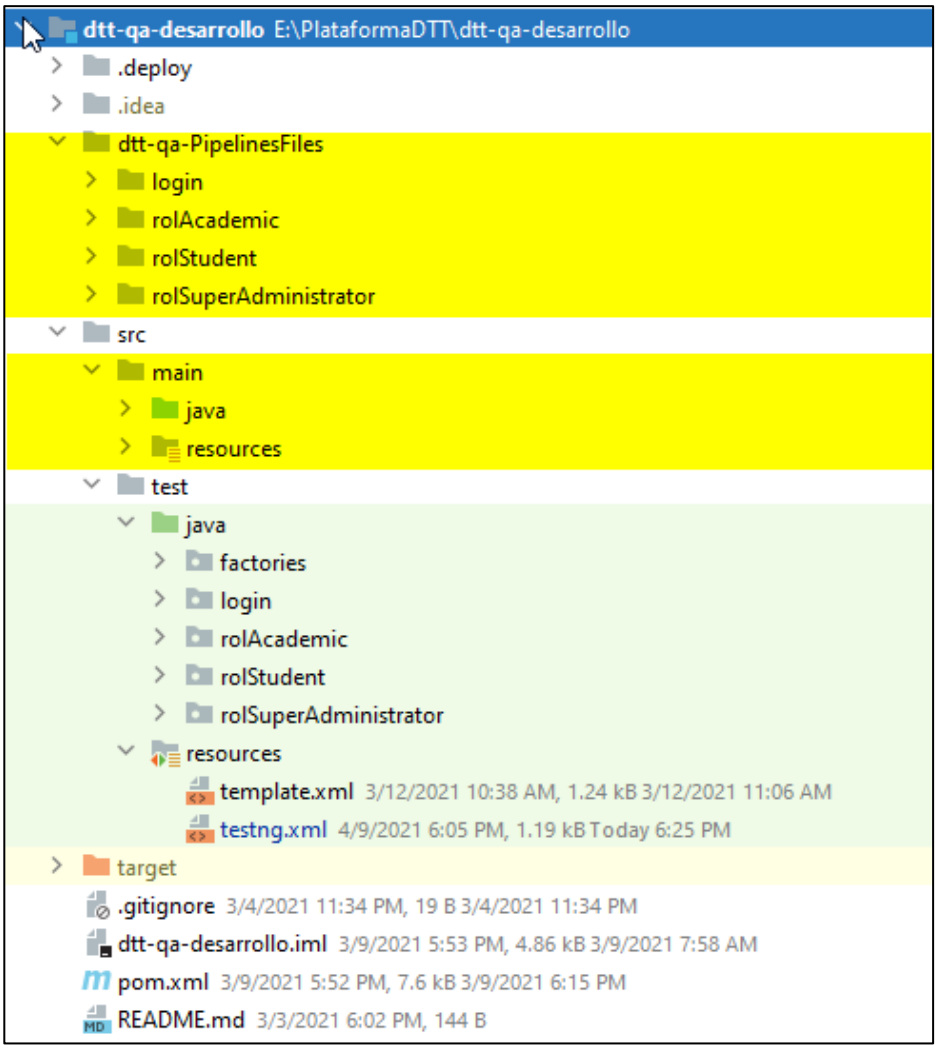
A partir de ello, se definieron algunas pruebas las cuales se automatizaron con la construcción de esta arquitectura, siendo:

- Rol Administrador
    - Carga masiva de practicantes.
    - Carga masiva de docentes.
- Rol Student:
    - Carga masiva de estudiantes.
    - Carga masiva de notas.
    - Generar solvencia de prácticas.
- Rol Academic:
    - Ingreso de solicitudes para hacer prácticas por parte de los estudiantes.

A partir de este listado de pruebas, se procedió a la automatización de cada una, para ello lo primero que se definió fue una estructura de carpetas, clases, archivos y demás recursos necesarios para la codificación de estas pruebas.

La siguiente imagen muestra la estructura de carpetas y archivos, que se implementó en la construcción de estas pruebas.

Figura 18. **Estructura de carpetas y archivos en el proyecto**

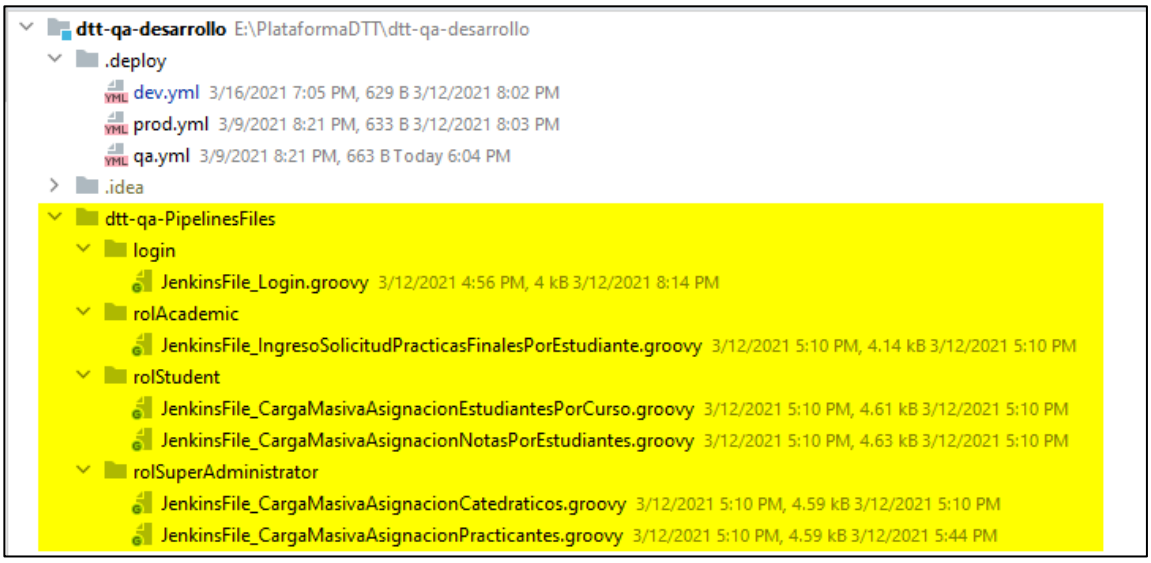


Fuente: elaboración propia.

Como podemos observar (en la figura anterior), el proyecto se divide en tres grandes grupos, el primero es “dtt-qa-PipelineFiles”, el segundo “src/main” y el tercero es “src/tests”. Cada uno cumple una función específica, para mantener una mejor claridad y limpieza en el código fuente.

En la carpeta dtt-qa-PipelineFiles, podemos encontrar las clases construidas, para la ejecución de las pruebas, estas clases fueron creadas bajo el lenguaje de Groovy y contienen las instrucciones para clonar el repositorio de pruebas automáticas, ejecutar cada prueba según el grupo requerido y los parámetros recibidos y poder enviar el correo electrónico de finalización de la prueba. Estas clases, son el medio por el cual los Jobs en Jenkins ejecutan las pruebas automáticas.

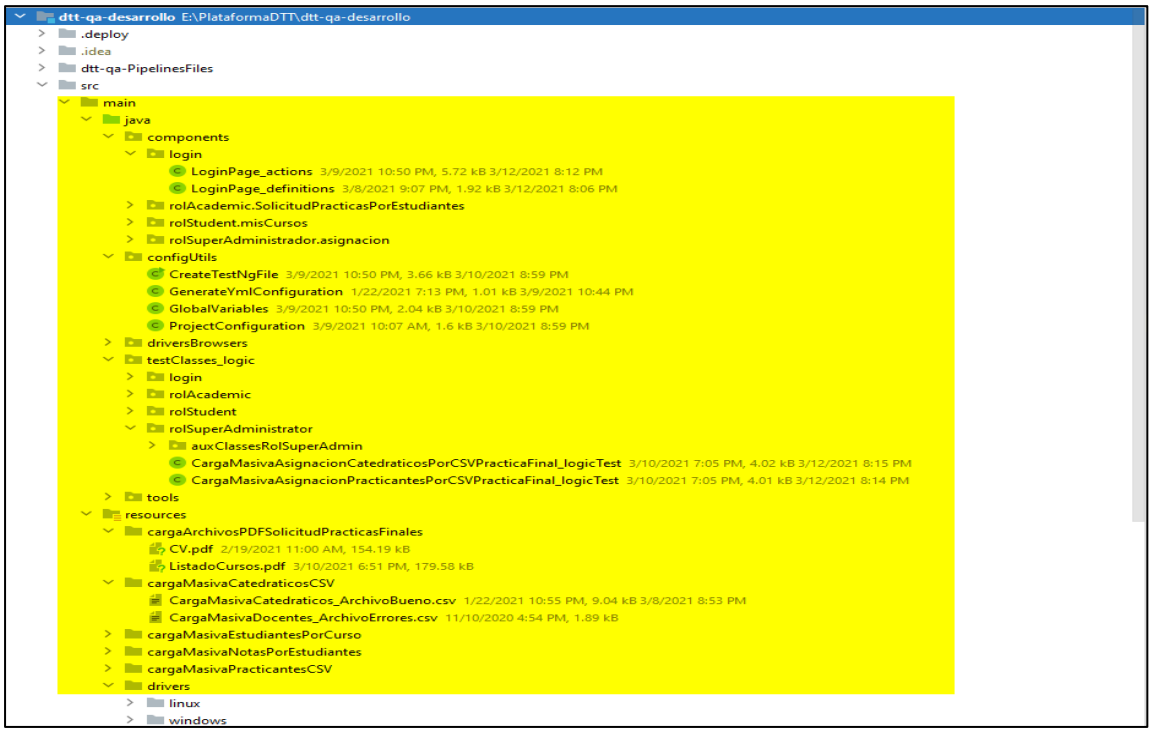
Figura 19. **Estructura de carpetas PipelinesFiles**



Fuente: elaboración propia.

En la carpeta src/main, esta toda la lógica de programación, que se realizó para la construcción de las pruebas automáticas, en esta carpeta podemos encontrar las clases de mapeo, clases con la finalidad de mapear un elemento web con una variable y con ello poder manipular dicho elemento, ya sea darle un click, ingresar un texto, selección el elemento, obtener qué valor tiene, entre otras acciones. Las clases de lógica de negocio, que nos permite realizar la secuencia de instrucciones a realizar para la construcción de la prueba automática también se encuentran en esta carpeta; otras clases de desarrollo para interactuar con nuestra lógica de programación y las acciones que necesitamos realizar, así mismo encontramos otros recursos (como archivos de configuración, archivos de datos) que se requieran para realizar las pruebas.

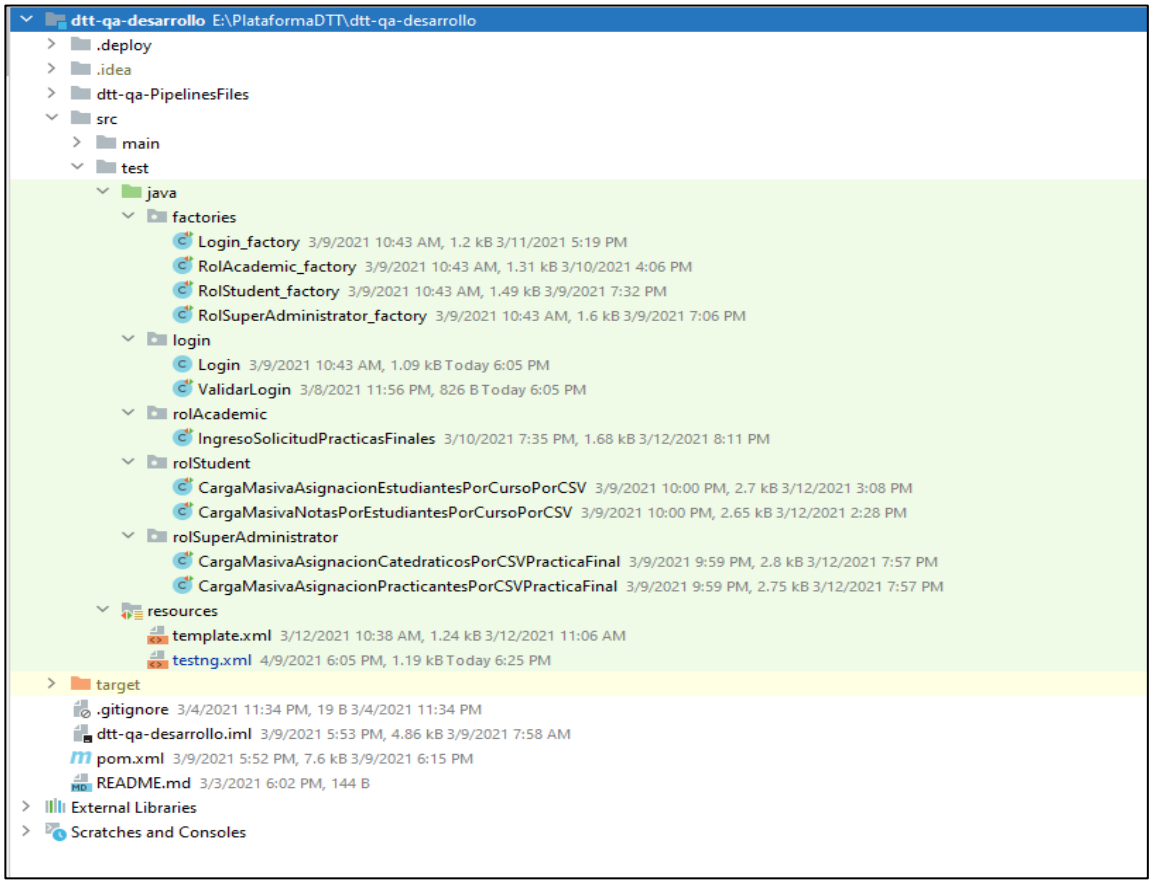
Figura 20. **Estructura de carpetas src/main**



Fuente: elaboración propia.

En la carpeta src/test, están las clases de ejecución de las pruebas. Por medio de estas clases, podemos comunicarnos a las clases de lógica y con ello poder solicitar la ejecución de las pruebas según el grupo que necesitemos, enviando los parámetros necesarios para su funcionamiento. Como podemos observar (en la figura siguiente), también se encuentran en este bloque los archivos xml (template y testng), por medio del archivo testng.xml, es que podemos solicitar la ejecución de las pruebas, este se construye a partir del archivo template.xml y se realiza con ayuda del framework Maven.

Figura 21. **Estructura de carpetas src/test**



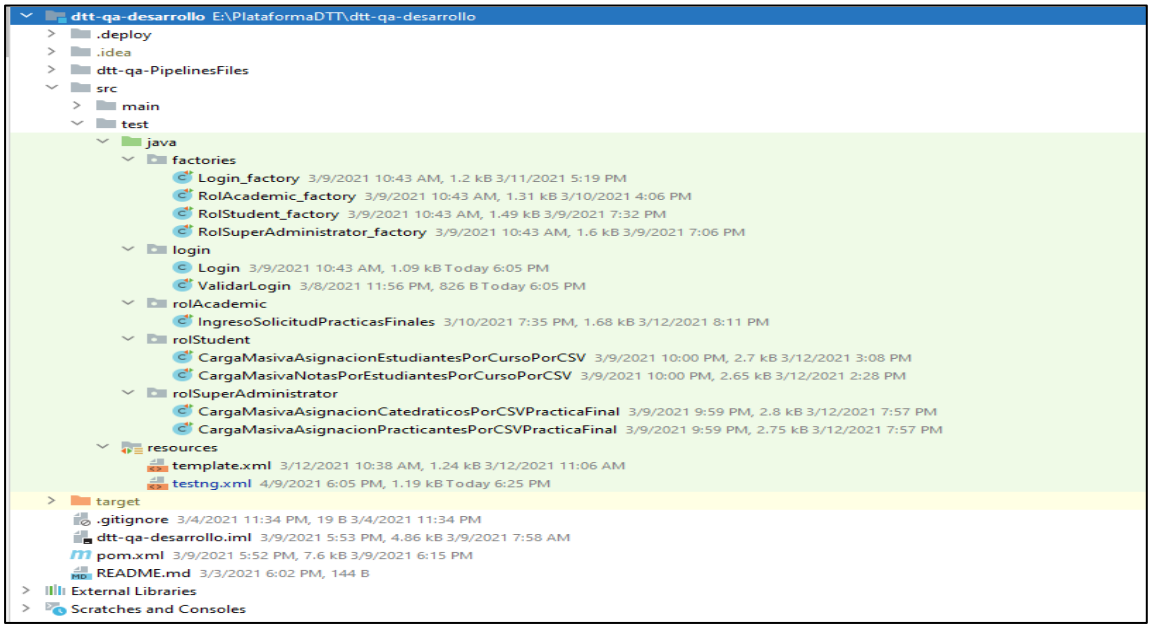
Fuente: elaboración propia.

La automatización de estas pruebas nos requiere una serie de instrucciones, teniendo en cuenta los diferentes frameworks que utilizamos, es necesario tener un orden en la estructura de sus archivos y recursos, por lo cual esta misma estructura es la que se guarda en el repositorio de versionado de software.

Tomando en cuenta, que existe un repositorio para el código fuente y otro para la infraestructura de la arquitectura de pruebas.

En la siguiente figura podemos observar la estructura del segundo repositorio, en donde encontraremos las librerías, instaladores y demás recursos necesarios para levantar el entorno de ejecución de pruebas automáticas de la Plataforma DTT.

Figura 22. **Estructura del repositorio de infraestructura de pruebas**



Fuente: elaboración propia.

3.4. Automatización de pruebas unitarias y de integración

Uno de los objetivos de este proyecto, es iniciar con la automatización de pruebas (unitarias y de integración) a nivel de código, para ello se han creado las plantillas iniciales en el código fuente de la Plataforma DTT.

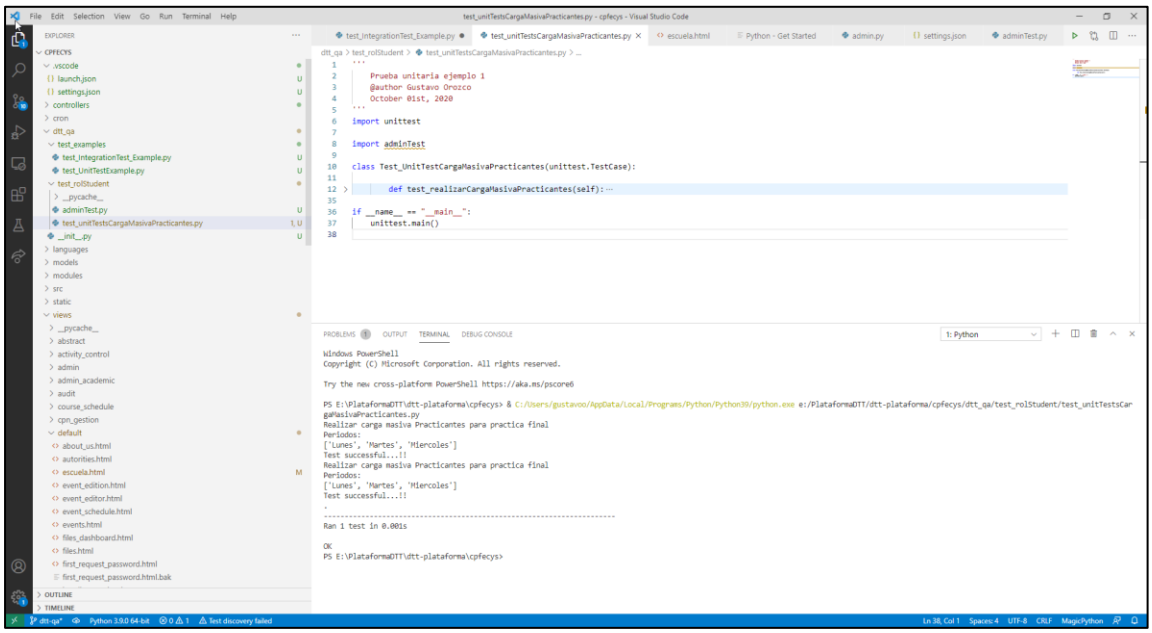
Definimos como pruebas unitarias, todas las pruebas que se puedan ejecutar a métodos puntuales en el código, teniendo como requisito que estos métodos no dependan de otros métodos o clases de programación para su utilización. Con ello aseguramos que, al momento de ejecutar pruebas unitarias, las estamos realizando a métodos que solo tienen una responsabilidad, ejerciendo de alguna forma tener que mantener métodos más claros y puntuales en su ejecución.

También nos enfocamos a pruebas de integración y en este contexto definimos estas pruebas a realizarse sobre los métodos principales de programación, donde se integran (valga la redundancia) varias clases o métodos entre sí; teniendo como objetivo poder validar aquellos métodos principales donde la lógica de programación se une.

Como observación importante, estas pruebas son enfocadas a los métodos y clases de lógica de negocio, donde se desarrolla la lógica de programación del proyecto, no a las clases, métodos y archivos de presentación (frontend), es decir, archivos html, css, javascript, y otros que se utilicen para el diseño del sitio web.

En la siguiente figura, podemos observar la herramienta utilizada para la ejecución de pruebas unitarias: Visual Studio Code en su versión community.

Figura 23. **IDE para ejecutar pruebas unitarias y de integración**



Fuente: elaboración propia.

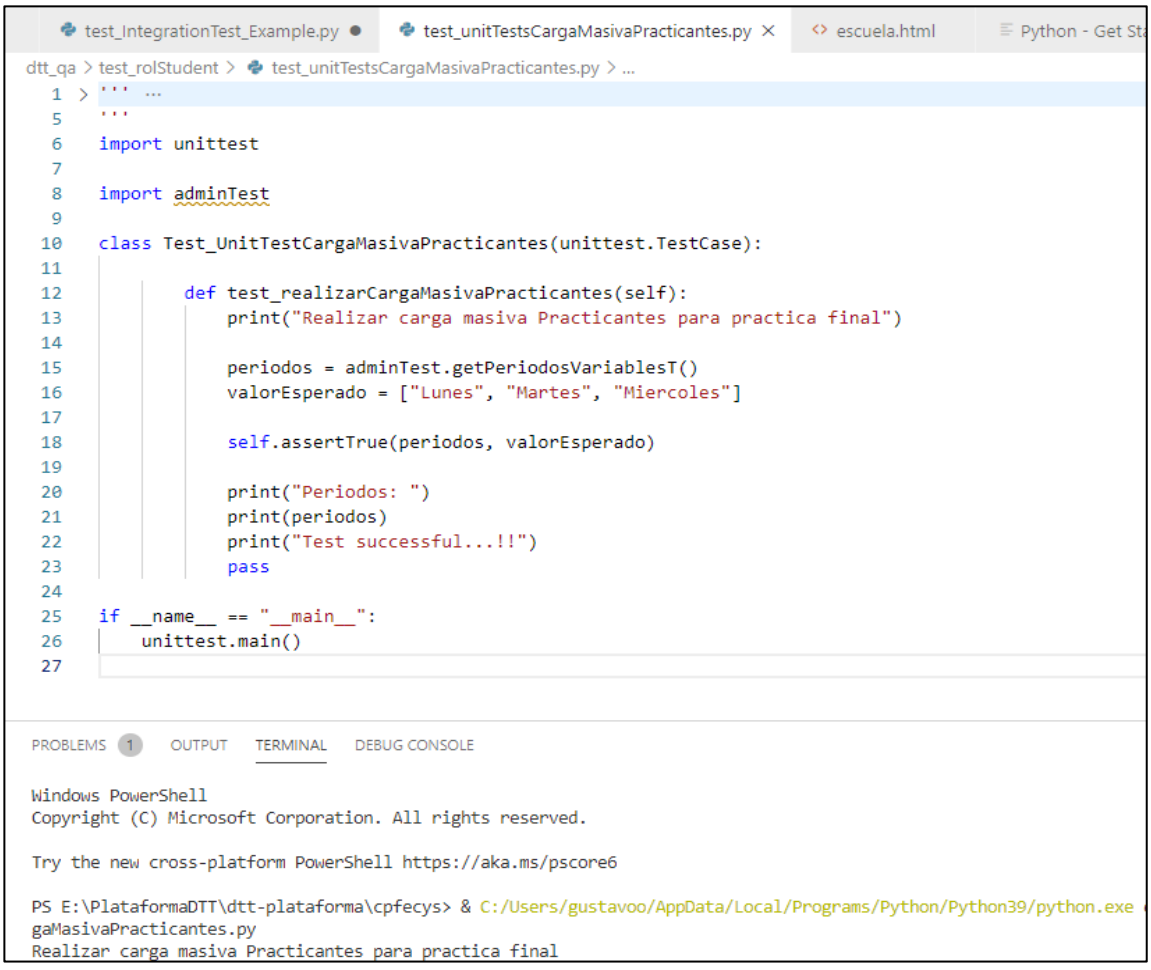
Como el proyecto de la Plataforma DTT está desarrollado principalmente en el lenguaje de Python, se utilizaron las librerías de UniTest para este lenguaje de programación. Y su suite de TestCases. Con este framework, se pueden programar métodos con el objetivo de validar métodos del código fuente de la Plataforma DTT.

Siendo necesario instanciar el método que necesitamos validar (enviando parámetros para su procesamiento si fuera necesario) y tener claro que valores nos debería de responder para realizar la validación.

En este pequeño ejemplo lo que se valida es que el método “getPeriodosVariables” devuelva siempre periodos que nosotros mismos sabemos que deben existir. Los valores con los que se va a comparar la

respuesta de este método, es un valor quemado en código, que sabemos que no pueden ser diferente. Al momento de tener valores aleatorios que sabemos que pueden variar según los parámetros, es necesario tener un archivo o bien una variable con el valor de estas diferentes respuestas, pero si los valores de respuesta no son conocidos y son muy cambiantes, lo ideal es tener un rango de valores que consideramos que debería estar en la respuesta. Con ello logramos saber que el método responde dentro del alcance considerado.

Figura 24. **Ejemplo prueba unitaria**



Fuente: elaboración propia.

Como podemos observar, en la figura anterior, el código de una prueba unitaria debería ser muy simple, tener la instancia del método a validar, tener la respuesta esperada que nos debería procesar siempre el método a validar y luego realizar la comparación de ambos datos y si todo está correcto la prueba se ejecuta con éxito, de lo contrario genera un mensaje de error.

Las pruebas unitarias o de integración, deberían ser pruebas fijas y no deberían estar cambiando, a menos que el método a validar sufra cambios considerables, de lo contrario si el método no ha sido manipulado durante el desarrollo, sabemos que debería mantener su comportamiento inicial, por ello cuando un nuevo desarrollador viene y modifica alguna clase o método, debe ejecutar las pruebas unitarias y de integración primero, antes de cualquier cambio que tenga que realizar al código, así garantiza que el código es funcional antes de realizar sus cambios. Posterior de realizar sus cambios debería de ejecutar las pruebas nuevamente, para asegurarse que no afecto ninguna funcionalidad existente de los métodos ya existentes. Y como indicamos anteriormente, las pruebas solo se modificarán si el método que se valida sufrió algún cambio, de lo contrario se deberían de ejecutar exitosamente todas las pruebas y si alguna falla, es porque existe alguna inconsistencia con sus cambios y lo que existe actualmente.

A partir de las pruebas unitarias y de integración, podemos mitigar de gran medida la posible introducción de un error (bug) en el código de la Plataforma DTT, generando más confianza al desarrollador para que sus cambios implementados se desplieguen más rápidamente en el ambiente de Producción.

El objetivo de este proyecto es dejar la línea base para la elaboración de pruebas unitarias y de integración, realizar la investigación inicial para saber que

framework utilizar y dejar la plantilla de ejemplo, para que los nuevos desarrollos puedan incluir ya este tipo de pruebas en la Plataforma DTT.

Para el desarrollo de estas pruebas (unitarias y de integración), algo importante es poder tener claro la finalidad de los métodos y clases que se desarrollan ya en el proyecto de la Plataforma DTT; para no repetir esfuerzos, se aconseja que se enfoquen a métodos principales que sabemos son esenciales para la ejecución de las funcionalidades. Tomemos en cuenta que las pruebas de regresión funcionales prueban la funcionalidad como una caja negra, donde se ingresan datos y se obtiene un resultado final, aquí en las pruebas unitarias y de integración, estamos validando los engranajes principales que componen esta caja negra y muchas clases o métodos pueden ser utilizados en diversas partes del código, por lo cual entre más claro son los nombres (de las clases, métodos y variables) mejor será la utilización y definición de las pruebas a realizar.

Finalmente, podemos decir que una gran ventaja en el desarrollo de las pruebas unitarias y de integración, es que se puede llegar a conocer de mejor forma las reglas del negocio y todas las validaciones que se realizan en la Plataforma DTT, ya que al conocer que validaciones se realizan y como se hacen, se tiene un mejor concepto en cuanto a las funcionalidades existentes y las futuras a desarrollarse, permitiendo tener una mejor claridad al programador para realizar mejores métodos y sobre todo, un código más claro para su futuro mantenimiento.

La construcción de las pruebas unitarias y de integración, es una de las principales bases que se requiere, para que se desarrolle el despliegue continúo automatizado para la compilación e implementación de artefactos en el ambiente de producción.

4. FASE DE APRENDIZAJE ENSEÑANZA

A continuación, se presentan dos temas muy importantes, para muchas empresas que necesitan mejorar su fase de control de calidad y en la automatización de sus procesos de desarrollo de software. Tomemos en cuenta que un proceso de automatización es un proyecto a largo plazo, y que en muchas ocasiones observar resultados a corto plazo es muy difícil, por lo cual la empresa debe tener en cuenta que al iniciar este proyecto es para fortalecer las fases operativas y mejorar sus procesos internos de desarrollo, con el objetivo principal de mejorar la calidad de los productos desarrollados y que a mediano/largo plazo se buscara reducir el tiempo en estos.

4.1. Guía para iniciar la automatización de pruebas

Antes de dar una guía para la automatización de pruebas, nos enfocaremos a tener en mente un par de conceptos y tener con ello la claridad de lo que buscamos. Una de las primeras preguntas que nos podemos hacer es:

4.1.1. ¿Qué es la automatización de pruebas en la fase de control de calidad del desarrollo de software?

En respuesta a esta pregunta, podemos resumir que la automatización de pruebas es tener como objetivo una herramienta (generalmente desarrollada a la medida) que nos permita controlar y configurar todo el entorno de los escenarios de prueba, que permita ejecutar las pruebas, validar los resultados obtenidos con los esperados y nos pueda notificar al momento de finalizar las pruebas. Y que

se pueda realizar desde un entorno amigable al usuario encargado de realizar estas pruebas.

4.1.2. ¿En qué nos ayuda la automatización de pruebas y cuáles son sus beneficios?

Dentro de los principales beneficios podemos mencionar:

- Reduce el costo del proyecto a largo plazo.
- Reduce el tiempo de las pruebas a mediano/largo plazo.
- Con el tiempo, se obtienen mayor cobertura de las pruebas.
- Se obtiene mejor consistencia en las pruebas automatizas.
- No se genera dependencia a personas para validar funcionalidades complejas.
- Se pueden programar para su ejecución a determinados periodos y posterior a ciertas tareas.

Estos son algunos de los beneficios, principalmente la automatización de pruebas nos ayuda en mejorar la calidad de los productos desarrollados.

4.1.3. ¿Se puede automatizar todas las pruebas en un proyecto?

Debemos tener claro que en un proyecto de software no todas las pruebas son automatizables y es de las primeras cosas a tomar en cuenta, no debemos enfocarnos en automatizar todo, las pruebas automatizadas no sustituyen las pruebas manuales. Para entender correctamente cómo funciona el sistema, que pasos se requieren para validar las funcionalidades o bien la secuencia en que se deben ejecutar es necesario realizar de primero pruebas manuales, esto nos

permitirá tener un mejor contexto sobre los escenarios o casos de pruebas que se desean automatizar.

4.1.4. Criterios para la automatización de pruebas

Para iniciar con la automatización de pruebas necesitamos identificar las tareas o casos de pruebas con ciertas características, tales como:

- Las que se ejecutan varias veces o son repetitivas.
- Donde se centran las áreas de riesgo de la aplicación.
- Las que necesitan ser ejecutadas con diferentes conjuntos de datos o tengan que usar alta carga de datos.
- Las que representan las rutas críticas de la aplicación, es decir las que son las prioridades para el negocio.
- Funcionalidades que representen un alto grado de error durante las pruebas manuales.
- Pruebas que se requieran realizar en varios sistemas operativos, dispositivos móviles y navegadores.

Ahora bien, también es importante saber identificar cuando no es prudente automatizar las pruebas en el desarrollo de un proyecto de software:

- Cuando existan funcionalidades inestables.
- Funcionalidades con resultados no predecibles, por ejemplo, los captchas.
- Cuando no se tiene el tiempo y el recurso para su asignación.
- No se tenga definido claramente el alcance del proyecto, y este bajo constante cambio.

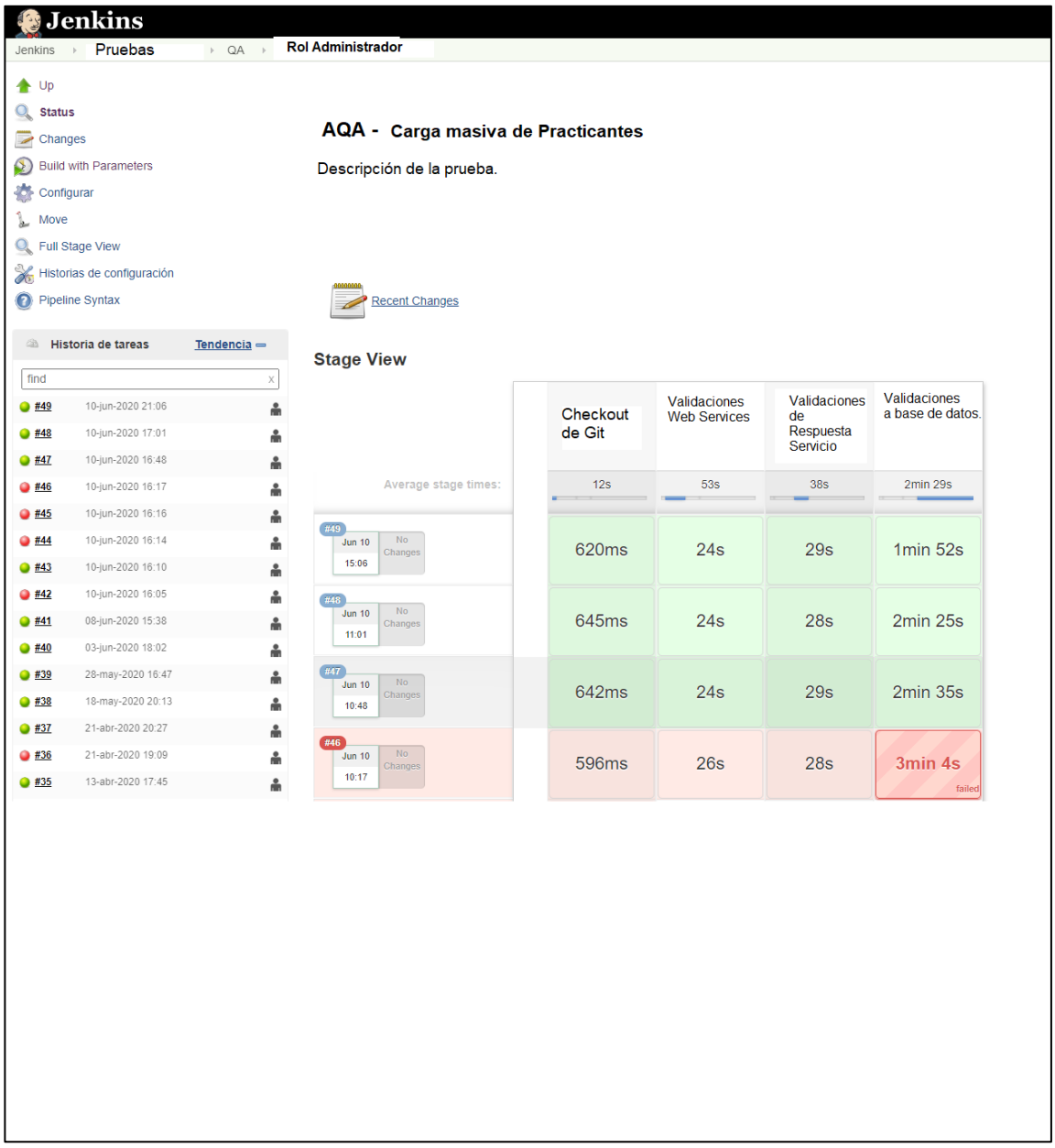
Cabe aclarar que a pesar de que no se puedan automatizar las pruebas, siempre se deben realizar pruebas manuales al proyecto.

4.1.5. Algunos elementos importantes en la automatización de pruebas

Para finalizar, es importante tomar en cuantos ciertos elementos que nos ayudaran a tener un proyecto de automatización de pruebas mantenible con el tiempo y pueda ser fácil de darle mantenimiento por diversas personas:

- Incluir en todas las etapas del proyecto, al equipo de control de calidad.
- Identificar el lenguaje de programación a utilizar, que permita integrarse a la mayoría de las librerías o frameworks existentes para realizar pruebas automáticas.
- Utilizar repositorio de versiona miento para el código a desarrollar en las pruebas automáticas.
- Definir una herramienta para la ejecución de pruebas a desarrollar, que permita ejecutar las pruebas de forma amigable por cualquier integrante del proyecto.
- Definir una nomenclatura y buenas prácticas en el desarrollo del código, así como revisiones al mismo.
- Documentar los escenarios a validar, para tener una referencia de que se está probando y bajo qué criterios.
- Y principalmente, definir tiempo tanto para el desarrollo de pruebas automáticas, como para la etapa de mantenimiento del proyecto, ya que, si cambia una funcionalidad, la prueba debe actualizarse también.

Figura 25. **Ejemplo de la ejecución de una prueba automática**



Fuente: elaboración propia.

4.2. Cultura DevOps

Nuevamente, antes de hablar un poco de este tema, es importante conocer algunos conceptos.

4.2.1. ¿A que nos referimos con DevOps?

Podemos decir que dos departamentos que han trabajado independiente deben de trabajar juntos ahora. En otras palabras, DevOps es el acrónimo entre la unión de trabajo de dos equipos, Desarrolladores y Operaciones.

Por un lado, están los desarrolladores (Developers) que son los programadores o desarrolladores de software, que trabajan bajo su propia jerarquía, metodologías y herramientas. Y por otro lado están los Operadores (Operations) que son los encargados de sistemas y hardware de la empresa, también con su propia jerarquía, metodologías y herramientas.

DevOps tiene como objetivo, unir estos dos equipos de trabajo que permitirá un mejor beneficio en la gestión de los proyectos.

Podemos entonces decir que DevOps es una filosofía de buenas prácticas para trabajar desde un principio en un mismo proyecto como un único equipo.

4.2.2. Los pilares de DevOPS

Se basa en tres pilares principalmente:

- Generar una cultura en la organización que lleve a la colaboración.

- Se basa en una gestión ágil, utilizando diferentes metodologías agiles de gestión de proyectos y de servicios.
- Y por último la automatización, implementar una cadena de herramientas, que se ejecutan y nos permitan llevar el control para artefacto realizado por el desarrollador, validado por control de calidad y todo el proceso que se requiera para su despliegue en el ambiente de producción listo para ser utilizado por los usuarios.

4.2.3. ¿Para qué sirve DevOps?

Hoy en día, la demanda por realizar proyectos de software es bastante alta, y en un corto tiempo han aparecido muchas empresas que tienen como meta cubrir gran parte de esta demanda. O bien empresas que desarrollan internamente para su propia gestión, saben que la demanda de automatizar procesos cada vez es más exigente. Ante esto los tiempos de desarrollo y entrega de productos han tomado un punto crucial, y se busca tener productos de software en el menor tiempo y con la mejor calidad posible.

Lamentablemente los diferentes equipos (desarrolladores, control de calidad, sistemas) que requiere un proyecto, trabajan de forma independiente.

Para buscar un trabajo en equipo, se deben trabajar en ciertos aspectos importantes:

La desigualdad en velocidad de entrega, nos referimos a que ambos equipos deben estar conscientes que llegada la fecha de entrega, ambos deben tener listo sus entregables, por un lado el desarrollador con el artefacto realizado, debe estar validado correctamente y por el otro lado las personas de operaciones

deben de proveer la infraestructura y recursos necesarios para poder implementar este artefacto en producción.

Diferencias entre métricas e incentivos, tomar en cuenta que en la mayoría de las empresas el área de desarrollo se mide por tiempos de entregas y costos, operaciones se mide por disponibilidad y estabilidad. Lo que implica que cada área es diferente y no se puede medir por igual a todos.

Trabajar en el muro entre los departamentos, para definir una mejor colaboración entre ambos equipos durante toda la gestión del proyecto.

Unificar los requerimientos, tanto funcionales como no funcionales, no solo que sean fáciles de realizar si no también, fáciles de manejar, fáciles de implementar, monitorear y tengan buen rendimiento.

También se debe trabajar el hecho que diferentes ambientes tienen diferentes resultados.

Y por último se debe trabajar también en poder compartir los activos, es decir compartir la información generada en cada área y el conocimiento generado, planes de pruebas, técnicas de monitoreo y seguridad.

4.2.4. ¿Cómo unir estos mundos para tener DevOps?

Para iniciar la implementación de DevOps en la empresa es importante tomar en cuenta:

Primero la calidad, es decir tener procesos que permitan realizar entregas continuas, hacer cambios rápidos, fáciles de probar y dar seguimiento.

Segundo es la automatización, aquí busca acelerar las liberaciones de los cambios, reducción de operaciones manuales y por ende de errores.

Y por último la colaboración, buscando tener métricas en común entre todos los equipos, mejor visibilidad del trabajo realizado y compartir los activos generados.

Figura 26. **DevOps un cambio cultural**

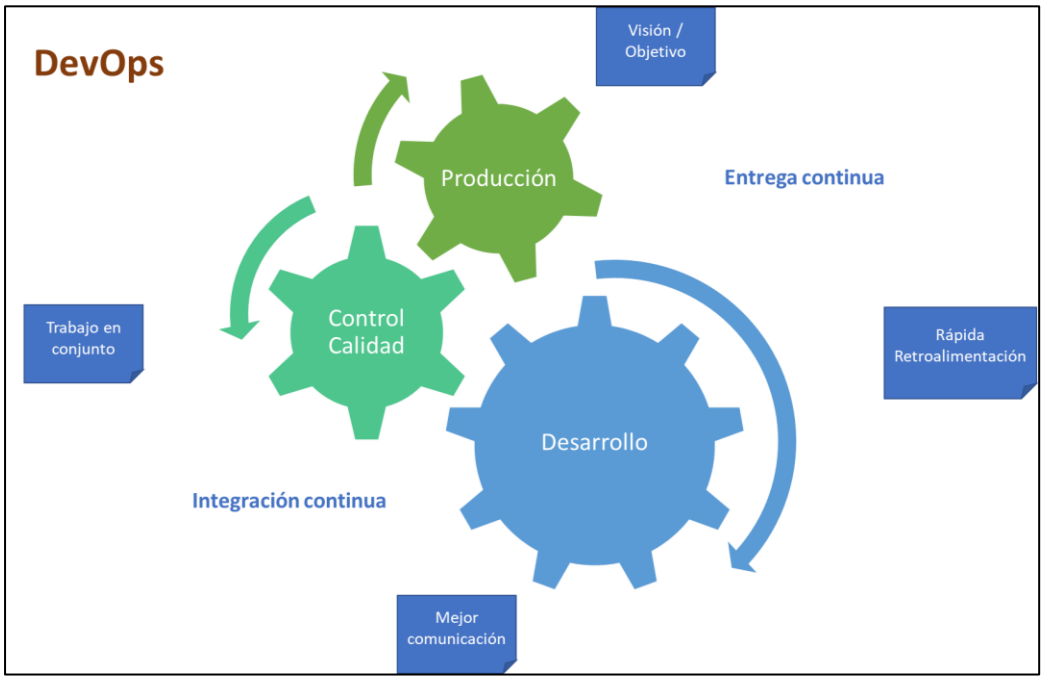
Fuente: elaboración propia.

DevOps no solo es un producto, si no que se trata también de procesos, de involucrar a la gerencia y los departamentos, a los clientes, en otras palabras, es interactuar con la gente que tiene reuniones periódicamente. A fin de cuentas, DevOps se trata de un cambio cultural, una reestructuración de la relación entre

desarrollo y operaciones, para buscar el bien común del negocio y generar valor al cliente.

Con un modelo ágil la retroalimentación del usuario en las diferentes áreas es mucho más rápido, además como los periodos de evolución son más cortos, los problemas de comunicación entre los departamentos se dan antes y con más frecuencia. Los problemas son compartidos y se fortalece el trabajo en equipo, con estos antecedentes formamos lo que es el movimiento DevOps que permite eliminar las barreras entre las áreas de sistemas y desarrollo y con el esfuerzo de todos, se busca generar la sinergia suficiente para responder ante los continuos cambios del negocio y entregar productos de calidad y de satisfacción al cliente.

Figura 27. **Movimiento DevOps**



Fuente: elaboración propia.

CONCLUSIONES

1. Utilizar Docker como herramienta para el entorno tanto de ejecución y desarrollo nos permite unificar e integrar en un solo lugar todas las dependencias (*frameworks,* librerías, entre otros) que nuestro proyecto utiliza, facilitando la construcción y despliegue en los ambientes de control de calidad y producción la implementación de nuevos cambios al proyecto de pruebas automatizadas. También nos facilita la instalación y configuración de los ambientes en los equipos locales de los desarrolladores que necesitan usar el proyecto de pruebas.

2. La principal ventaja de utilizar Java como herramienta de desarrollo se debe a los *frameworks* que nos ofrece este lenguaje (integración con *TestNG, Selenium, RestAssured,* entre otros), ya que nos permite controlar de mejor forma la ejecución de pruebas y poder tener el control en todo momento de las pruebas y con ello poder determinar en qué momento una validación no cumple con los criterios de aceptación.

3. Nuestro principal motivo de utilizar Jenkins como herramienta de ejecución de pruebas se deriva a que esta herramienta permite construir Jobs (espacios de tareas o trabajos) que permitirán más adelante poder integrar nuevas tareas enfocadas a compilar y desplegar en cada ambiente los diferentes artefactos desarrollados, actividad que actualmente se realiza de forma manual y poder tener una herramienta que administre todo este proceso de actualización de la Plataforma DTT en los ambientes correspondientes.

4. La Plataforma cuenta con varias funcionalidades a las cuales se le debe desarrollar sus pruebas automáticas, el proyecto cubrió un porcentaje mínimo de estas pruebas a realizar, ya que el principal objetivo del proyecto fue implementar una arquitectura para el desarrollo, ejecución y notificación de estas pruebas. Ser una guía para la elaboración de pruebas con las diferentes herramientas que se configuraron en los entornos de trabajo del proyecto, así como dar los lineamientos y directrices generales para la elaboración de las pruebas automáticas.

5. También el proyecto tuvo como finalidad dar a conocer el tema de las pruebas unitarias, elaborando una plantilla para la creación de estas pruebas en una branch del repositorio donde está el código fuente de la Plataforma DTT, con ello se busca que los desarrolladores que realizan cambios al código fuente, desarrollen también sus pruebas unitarias lo que permitirá tener un mejor código de desarrollo y sobre todo no se introduzca un bug (error de código) al momento de realizar cambios en el proyecto.

6. Realizar un proyecto de automatización de pruebas significa que se ha identificado un déficit durante la fase de control de calidad en el desarrollo de software y que se debe solventar lo antes posible, en otras palabras, debe tener la importancia como un proyecto de desarrollo de software propio, incluso dependiendo del avance de este se debe de priorizar sobre ciertas tareas que el proyecto de la Plataforma DTT requiera.

7. Para finalizar, queremos concluir que, para iniciar un proceso de automatización de pruebas en el desarrollo de software, es importante identificar dos puntos importantes, primero los procesos que la empresa tiene en la fase de control de calidad, estos deben ser acorde a los objetivos que la empresa tiene en la gestión y aseguramiento de calidad en los productos desarrollados; segundo es la comunicación que se pueda tener entre los equipos de control de calidad y los demás equipos involucrados en el proyecto. Es importante mencionar que dicha comunicación debe ser la más asertiva y oportuna entre todos los integrantes. Teniendo como finalidad el cumplimiento de los objetivos en la etapa de control de calidad del producto desarrollado.

RECOMENDACIONES

1. Utilizar archivos *DockerFile* para configurar Docker con las dependencias correspondientes, estos archivos son scripts de comandos que permiten de mejor forma generar los ambientes de trabajo. En estos archivos se especifican todas las dependencias a configurarse en la imagen de *Docker* y se ejecutan desde consola, lo que permite que se puedan ejecutar desde Windows o Linux y que es portable (los archivos de configuración), actualmente el proyecto se dejó todo configurado a partir de estos archivos, lo que permitirá a futuro un mejor mantenimiento al momento de actualizar los entornos de trabajo.

2. Integrar con otras herramientas, ya que Java ofrece diversidad de librerías como, por ejemplo, Cucumber que nos permite realizar historias de usuario para los criterios de aceptación de una prueba y los diferentes escenarios a validar, y con ello generar las validaciones correspondientes y tener un mejor orden en cuanto a las pruebas automáticas. Vale la pena realizar un análisis a esta herramienta y poder utilizarla en algunos casos de prueba para identificar de mejor forma el alcance de las pruebas. También en cuanto a la herramienta es recomendable ir actualizando los *frameworks* y versiones de estas, ya que cada nueva versión nos ofrece mejores resultados de rendimiento y sobre todo mejores criterios para realizar las validaciones.

3. Utilizar archivos JenkinsFile en la creación de Jobs en Jenkins (o también archivos Pipeline). Estos archivos permiten llevar el control y programación de las tareas de ejecución, permitiendo un mejor

mantenimiento a los procesos de ejecución de pruebas automáticas. Jenkins permite por medio de su interfaz gráfica crear Jobs con los cuales se ejecutan las pruebas automáticas, pero a largo plazo estar configurando cada uno de estos se vuelve complejo. Por lo cual tener archivos de configuración que puedan ser actualizados es mucho más sencillo y permite que un cambio se realice y refleje en Jenkins rápidamente. Cabe recalcar que estos archivos también dan la ventaja de poder versionarse y ser alojados en un repositorio para un mejor acceso y actualización. El proyecto se desarrolló con estos archivos, por lo cual queda la facilidad para que el estudiante pueda guiarse e implementar en las nuevas pruebas a realizar el mantener usando estos archivos de configuración.

4. Analizar y tener uno o dos estudiantes para continuar con la automatización de pruebas. Se debe de cubrir las pruebas de las funcionalidades que existen actualmente, lo que permitirá en un futuro a mediano plazo tener pruebas automatizadas de regresión sobre la Plataforma DTT pudiendo ver un cambio significativo en los tiempos de la fase de control de calidad. También en los próximos proyectos de desarrollo de la Escuela, se recomienda que se solicite entre los productos a entregar por parte de los estudiantes, las pruebas automatizadas de aceptación para cada uno de los artefactos o módulos a desarrollarse. Lo que permitirá mantener un proyecto de pruebas actualizado y no depender solo de proyectos enfocados a pruebas para su mantenimiento.

5. Incluir en los próximos desarrollos por parte de la Escuela con relación a la Plataforma DTT, que al momento de aprobar un proyecto de EPS los estudiantes tengan en sus productos el desarrollar las pruebas unitarias en el código fuente de los entregables a realizar. Con ello se busca tener

un código fuente de mejor calidad y que pueda por medio de estas pruebas mitigar posibles bugs. Otra gran ventaja de las pruebas unitarias es que permiten crear pruebas a diferentes escenarios lo que ayuda a tener un mejor contexto de lo que una clase o método debe de tener como alcance permitiendo segmentar de mejor forma el código fuente.

6. Definir uno o dos proyectos a futuro para automatizar las pruebas de las funcionalidades ya existentes, lo que permitirá que la Plataforma DTT tenga un gran porcentaje cubierto de las pruebas que necesite. Estos proyectos serian enfocados únicamente para automatizar pruebas para tener la misma prioridad que otros proyectos de desarrollo. En cuanto a nuevas funcionalidades se debe analizar en los próximos proyectos de EPS que los productos a entregar por el estudiante, esté el desarrollo de sus propias pruebas.

7. Incluir la fase de control de calidad durante todo el ciclo de desarrollo de software, es decir, tener un equipo de control de calidad que pueda iniciar en conjunto con el equipo de desarrollo (como proyecto de EPS). Esto permitirá que desde un inicio se pueda tener un mejor contexto, alcance y sobre todo entendimiento de las necesidades que la Escuela necesita en el tema de la automatización de sus procesos. También que ambos equipos estén siempre involucrados en los mismos productos y poder tener como mínimo dos reuniones semanales para retroalimentación en cuanto a los avances, inconvenientes y objetivos que cada producto a entregar se lleva.

BIBLIOGRAFÍA

1. BEUST, Cédric. *TestNG Documentation*. [en línea]. <https://testng.org/doc/documentation-main.html>. [Consulta: 17 de mayo de 2021].

2. Centro de Cálculo e Investigación Educativa. *Carreras de la Facultad de Ingeniería*. [en línea]. <http://eciencias.ingenieria.usac.edu.gt/index.php/carreras>. [Consulta: 20 de noviembre de 2020].

3. _____. *EPS Formatos*. [en línea]. <https://eps.ingenieria.usac.edu.gt/>. [Consulta: 20 de noviembre de 2020].

4. Ciberninjas. *Automatización de Pruebas*. [en línea]. <https://ciberninjas.com/10-mejores-herramientas-pruebas-ui/>. [Consulta: 01 de febrero de 2021].

5. Docker Incorporation. *Docker Documentation*. [en línea]. <https://www.docker.com>. [Consulta: 01 de febrero de 2021].

6. Escuela de Ciencias y Sistemas. *Plataforma DTT*-ECYS. [en línea]. <https://dtt-ecys.org/>. [Consulta: 20 de noviembre de 2020].

7. Facultad de Ingeniería Universidad de San Carlos de Guatemala. *Desarrollo CCI Ingeniería*. [en línea]. <https://portal.ingenieria.usac.edu.gt/index.php/aspirante/antecedentes>. [Consulta: 20 de noviembre de 2020].

8. International Organization for Standardization. *Manual Para Redactar Citas Bibliográficas Según Norma ISO 690 y 690-2.* [en línea]. <*https://biblioteca.intec.edu.do/downloads/documents/files/formatos-bibliograficos/manual-citas-bibliograficas-iso-690.pdf*>. *[Consulta: 04 de noviembre de 2021].*

9. Jenkins Incorporation. *User Documentation Home*. [en línea]. <https://www.jenkins.io/doc/>. [Consulta: 27 de julio de 2021].

10. Johan Haleby. *RestAssured Documentation*. [en línea]. < https://rest-assured.io/>. [Consulta: 17 de mayo de 2021].

11. Oracle Corporation. *Java para Windows*. [en línea]. <https://www.java.com/es>. [Consulta: 20 de noviembre de 2020].

12. RedHat Incorporation. *El concepto de DevOps*. [en línea]. <https://www.redhat.com/es/topics/devops>. [Consulta: 17 de mayo de 2021].

13. Selenium Incorporation. *The Selenium Browser Automation Project*. [en línea]. <https://www.selenium.dev/documentation/>. [Consulta: 17 de mayo de 2021].

14. Tecnología Informática. *Sistema Informático*. [en línea]. <https://www.tecnologia-informatica.com/que-es-sistema-informatico/>. [Consulta: 17 de mayo de 2021].

15. WALTON, Alex. *Fundamentos Java desde Cero*. [en línea]. <https://javadesdecero.es/>. [Consulta: 01 de febrero de 2021].

APÉNDICES

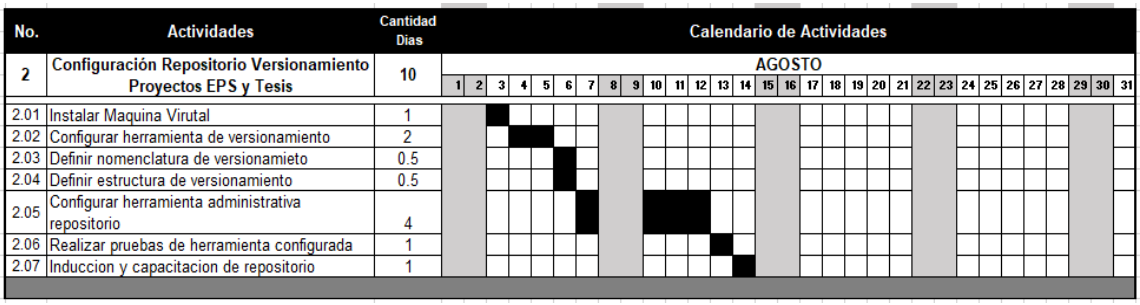
Cronograma de actividades del proyecto, que nos permite establecer la cantidad de horas a utilizar. Y que nos sirve de base para estimar el costo de los servicios de consultoría a realizarse.

Apéndice 1. **Cronograma de actividades fase 1**

No.	Actividades	Cantidad Dias	Calendario de Actividades
1	Configuración Ambientes de Desarrollo y Control de Calidad + Clúster	23	JULIO
			1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1.01	Instalar Maquina Virutal (Ambiente Desarrollo)	1	
1.02	Configurar Servidor de Aplicaciones	1	
1.03	Configurar Servidor de Base de datos	2	
1.04	Configurar repositorio de versionamiento	1	
1.05	Montar plataforma DTT en servidor	1	
1.06	BackUp de base de datos de Prod y clonar en bases de datos del ambiente	1	
1.07	Configurar Cluster de Base de Datos	7	
1.08	Realizar pruebas de funcionamiento de la plataforma en ambiente desarrollo	1	
1.09	Instalar Maquina Virutal (Ambiente Control de Calidad)	0.5	
1.10	Configurar Servidor de Aplicaciones	0.25	
1.11	Configurar Servidor de Cache	0.25	
1.12	Configurar Servidor de Base de datos	1	
1.13	Configurar repositorio de versionamiento	0.5	
1.14	Montar plataforma DTT en servidor	0.5	
1.15	BackUp de base de datos de Prod y clonar en bases de datos del ambiente	1	
1.16	Configurar Cluster de Base de Datos	2	
1.17	Realizar pruebas de funcionamiento de la plataforma en ambiente control de calidad	1	
1.18	Inducción y capacitación en ambientes	1	

Fuente: elaboración propia.

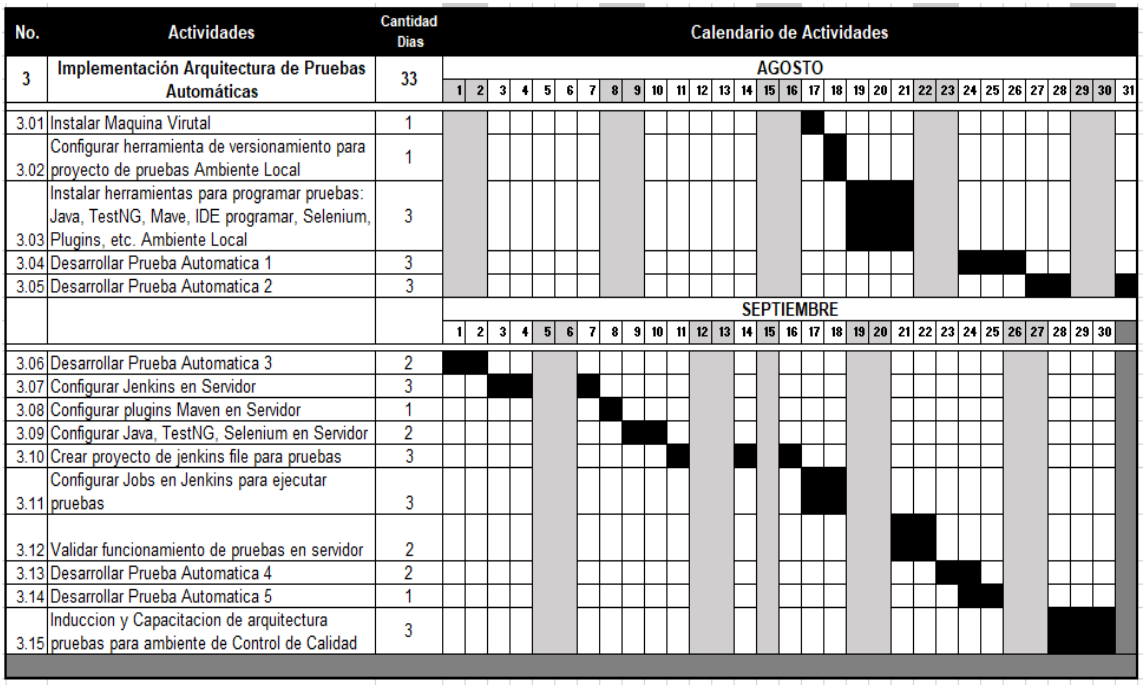
Apéndice 2. **Cronograma de actividades fase 2**



No.	Actividades	Cantidad Dias	Calendario de Actividades
2	Configuración Repositorio Versionamiento Proyectos EPS y Tesis	10	AGOSTO
2.01	Instalar Maquina Virutal	1	
2.02	Configurar herramienta de versionamiento	2	
2.03	Definir nomenclatura de versionamieto	0.5	
2.04	Definir estructura de versionamiento	0.5	
2.05	Configurar herramienta administrativa repositorio	4	
2.06	Realizar pruebas de herramienta configurada	1	
2.07	Induccion y capacitacion de repositorio	1	

Fuente: elaboración propia.

Apéndice 3. **Cronograma de actividades fase 3**



No.	Actividades	Cantidad Dias	Calendario de Actividades
3	Implementación Arquitectura de Pruebas Automáticas	33	AGOSTO
3.01	Instalar Maquina Virutal	1	
3.02	Configurar herramienta de versionamiento para proyecto de pruebas Ambiente Local	1	
3.03	Instalar herramientas para programar pruebas: Java, TestNG, Mave, IDE programar, Selenium, Plugins, etc. Ambiente Local	3	
3.04	Desarrollar Prueba Automatica 1	3	
3.05	Desarrollar Prueba Automatica 2	3	
			SEPTIEMBRE
3.06	Desarrollar Prueba Automatica 3	2	
3.07	Configurar Jenkins en Servidor	3	
3.08	Configurar plugins Maven en Servidor	1	
3.09	Configurar Java, TestNG, Selenium en Servidor	2	
3.10	Crear proyecto de jenkins file para pruebas	3	
3.11	Configurar Jobs en Jenkins para ejecutar pruebas	3	
3.12	Validar funcionamiento de pruebas en servidor	2	
3.13	Desarrollar Prueba Automatica 4	2	
3.14	Desarrollar Prueba Automatica 5	1	
3.15	Induccion y Capacitacion de arquitectura pruebas para ambiente de Control de Calidad	3	

Fuente: elaboración propia.

Breve resumen del presupuesto del proyecto, que nos permite tener un dato estimado y podamos valorar la implementación de este proyecto en términos financieros.

Apéndice 4. **Presupuesto del Proyecto**

Gastos de inicio			
Recursos	Cantidad	Costo Unitario Q	Subtotal Q
Mobiliario & Equipos			
Laptop	01	7 500,00	7 500,00
Impresora	01	1 600,00	1 600,00
Escritorio oficina	01	600,00	600,00
Silla de oficina	01	300,00	300,00
		Subtotal	**Q 10 000,00**
Gastos mensuales durante la ejecución del proyecto - 03 meses -			
Recurso	Costo Mensual Q	Tiempo / Meses	Subtotal Q
Inmueble			
Alquiler Local	1 500,00	03	4 500,00
Consumibles			
Agua	75,00	03	225,00
Energía Eléctrica	200,00	03	600,00
Internet	350,00	03	1 050,00
Telefonía	100,00	03	300,00
Combustible	100,00	03	300,00
Útiles de Oficia	50,00	03	150,00
		Subtotal	**Q 7 125,00**
Recurso Humano	*Horas Trabajo	Costo / Hora Q	Subtotal Q
Asesor EPS	60	200,00	12 000,00
Estudiante EPS	252	120,00	30 240,00
		Subtotal	**Q 42 240,00**
Gastos No Planificados			
	Gastos Calculados Q	Estimación	Subtotal Q
Se estima un porcentaje en gastos imprevistos o no planificados	59 365,00	5 %	2 968,25
		Subtotal	**Q 2 968,25**
		Total	**Q 62 333,25**
Total, sesenta y dos mil trescientos treinta y tres quetzales con veinticinco centavos.			

Fuente: elaboración propia

Apéndice 5. **Documento Manual Técnico**

A continuación como apéndice al documento, incluimos el manual técnico, dicho manual tiene como objetivo principal ser una herramienta de apoyo al estudiante para implementar la arquitectura de pruebas automáticas desarrollada en este proyecto en un entorno local (computadora con los requerimientos mínimos), en otras palabras, es una guía de los diferentes pasos que se requiere para desplegar la infraestructura de software a utilizarse para la ejecución y notificación de las pruebas automáticas desarrolladas.

IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Continuación apéndice 5.

INTRODUCCIÓN

El presente documento, tiene como finalidad dar a conocer la forma de configurar la herramienta Jenkins para la ejecución de las pruebas automáticas desarrolladas en el proyecto de EPS “IMPLEMENTACIÓN DE UNA ARQUITECTURA DE PRUEBAS AUTOMÁTICAS SOBRE LA PLATAFORMA DEL SISTEMA DE CONTROL DEL PROYECTO DE DESARROLLO DE TRANSFERENCIA TECNOLÓGICA (PLATAFORMA DTT) DE LA ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA”.

Así también dar a conocer las principales características del código fuente realizado y que sea una guía para los desarrolladores que pretendan dar continuidad o mantenimiento a este proyecto de automatización de pruebas.

La instalación de Docker, Git, Java y Maven (que son herramientas requeridas para la creación de las pruebas automáticas) queda fuera del alcance de este manual, enfocándonos únicamente a los artefactos o productos construidos durante el desarrollo de este proyecto de automatización.

Continuación apéndice 5.

ÍNDICE MANUAL TÉCNICO

Continuación apéndice 5.

Continuación apéndice 5.

1. CONFIGURACIÓN DE IMAGEN Y CONTENEDOR EN DOCKER PARA LA EJECUCIÓN DE PRUEBAS AUTOMÁTICAS

A continuación, describimos los pasos para crear un contenedor a partir de su imagen base en Docker, para la ejecución de pruebas automáticas desde Jenkins.

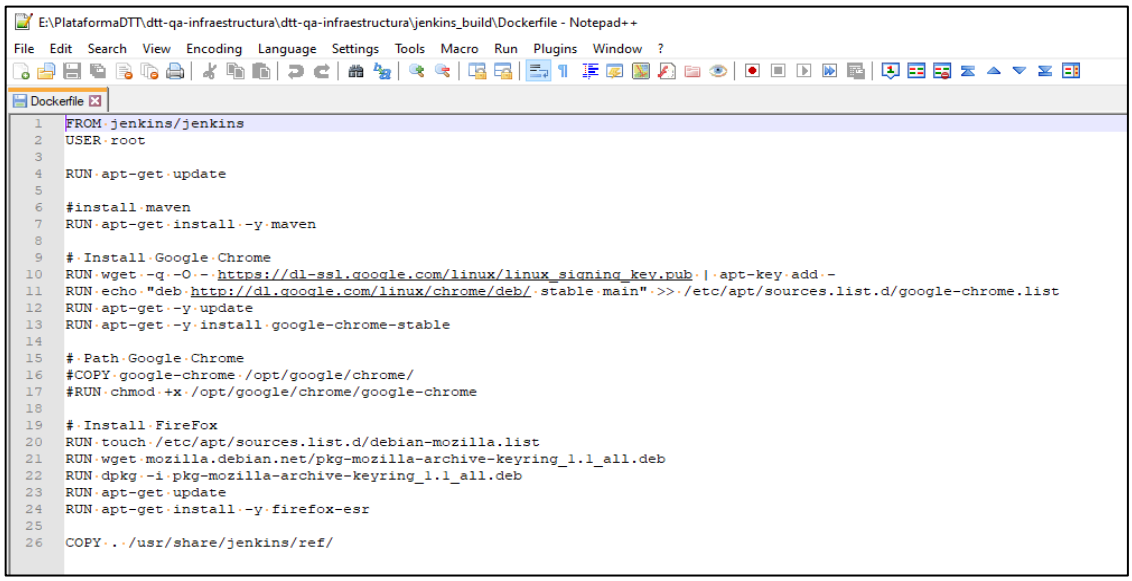
En la siguiente figura, se muestra el archivo “Dockerfile” (así se debe de nombrar), el cual fue el primero que utilizamos para montar nuestra arquitectura base, el cual posteriormente actualizaremos y configuraremos con las diferentes herramientas que necesitamos y que finalmente será nuestro primer producto entregable en este proyecto. Una imagen de Docker, para la ejecución, notificación y reportería de las pruebas automáticas sobre la plataforma DTT de la escuela.

Continuación apéndice 5.

1.1. Archivo DockerFile para construir imagen base del proyecto

Con este archivo, el cual contiene las peticiones necesarias, iniciamos la configuración de nuestro entorno de ejecución de pruebas automáticas. Las peticiones que solicitamos en este archivo son, tener una imagen a partir de Jenkins, es decir que nuestra imagen va a tener instalado Jenkins, adicional se solicita que actualice el sistema operativo (maquina Linux como base en la imagen Docker) y posteriormente que nos instale Maven, Chrome y Firefox.

Figura 1. **Diagrama para la ejecución y notificación de pruebas**



Fuente: elaboración propia.

Algo importante en este archivo de instrucciones, es la línea de COPY, el cual indica que todo lo relacionado a esta imagen en construcción se copie en la dirección que se indica.

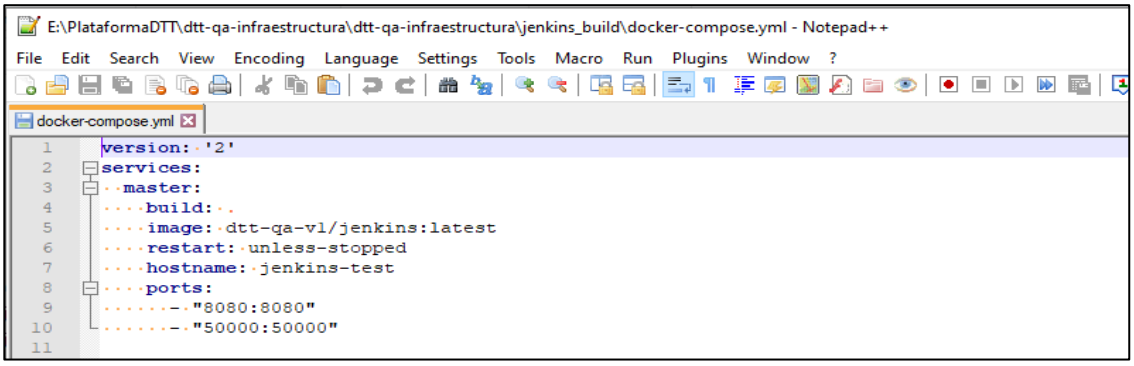
Continuación apéndice 5.

Para construir nuestra imagen, se requiere de un segundo archivo “docker-compose.yml” (el cual se debe de nombrar de esta forma), con el cual necesitamos configurar algunos criterios. La siguiente imagen muestra la estructura del archivo.

1.2. Archivo docker-compose.yml para construir la imagen base

En este archivo, definimos el nombre de nuestra imagen a construir, en este caso “dtt-qa-v1/Jenkins:latest” y habilitar los puertos que necesitamos para la comunicación al momento de la configuración de Jenkins por medio del browser que utilizaremos en nuestro equipo local.

Figura 2. **Archivo docker-compose.yml para construir la imagen base**



Fuente: elaboración propia.

Para ejecutar este archivo, necesitamos utilizar la terminal de comandos, al estar trabajando en Windows, utilizaremos el Command Pront -cmd- (de modo administrador), o bien si es en Linux la terminal de comandos con permisos Sudo, esto para ejecutar la siguiente instrucción:

Continuación apéndice 5.

1.3. Comando para construir la imagen base con Docker

El comado “docker-compose build”, se debe ejecutar en la terminal, teniendo está apuntando en la carpeta donde se encuentra el archivo DockerFile y docker-compose.yml, tal como podemos observar en la imagen.

Figura 3. **Comando para construir la imagen base con Docker**

Fuente: elaboración propia.

1.4. Construcción de la imagen base por medio de Docker

Luego de tener La siguiente imagen, nos muestra al momento que Docker inicia con la instalación de las dependencias y paquetes necesarios para construir la imagen base que solicitamos.

Figura 4. **Construcción de la imagen base por medio de Docker**

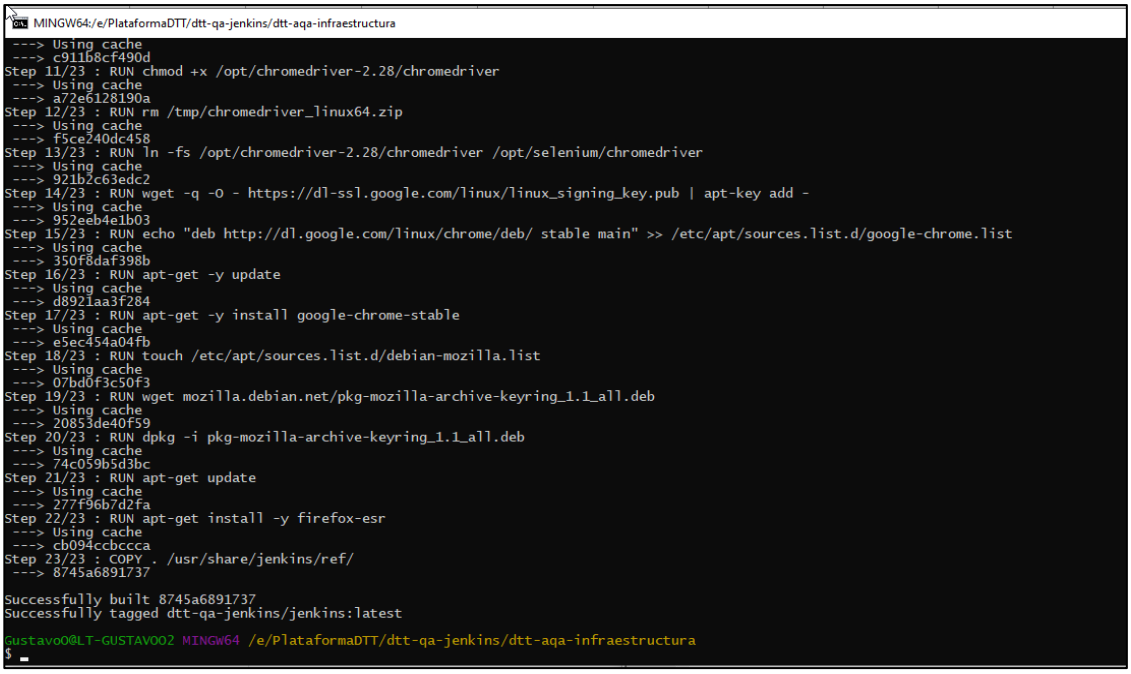
Fuente: elaboración propia.

Continuación apéndice 5.

1.5. Construcción final de la imagen base por medio de Docker

Finalmente, Docker realiza la instalación de las dependencias y con ello tenemos en nuestro equipo local la imagen a trabajar, tal como podemos observar en la imagen, se instalaron cada una de estas dependencias configuradas.

Figura 5. **Construcción final de la imagen base por medio de Docker**



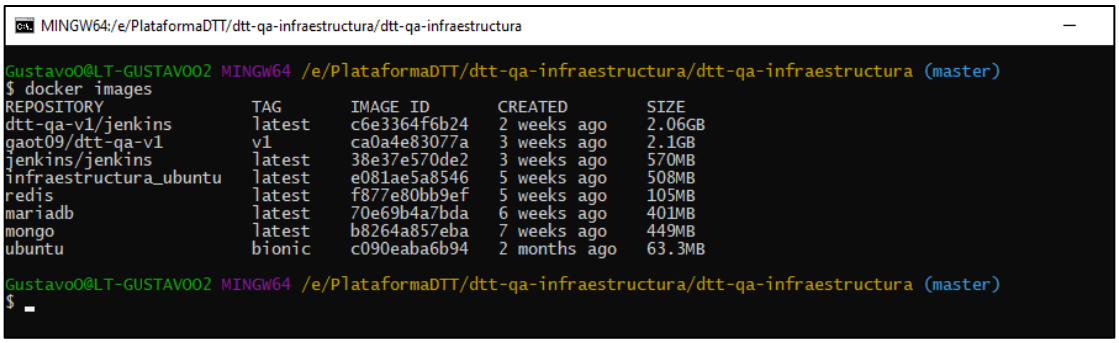
Fuente: elaboración propia.

Continuación apéndice 5.

1.6. Visualización de la imagen base por medio de consola

Para poder visualizar la imagen construida, podemos hacerlo por dos medios, el primero por medio siempre de consola ejecutando los comandos de “docker ps“, o bien de modo gráfico, visualizando en la interface de Docker, en el cual nos debe desplegar la nueva imagen creada.

Figura 6. **Visualización de la imagen base por medio de consola**



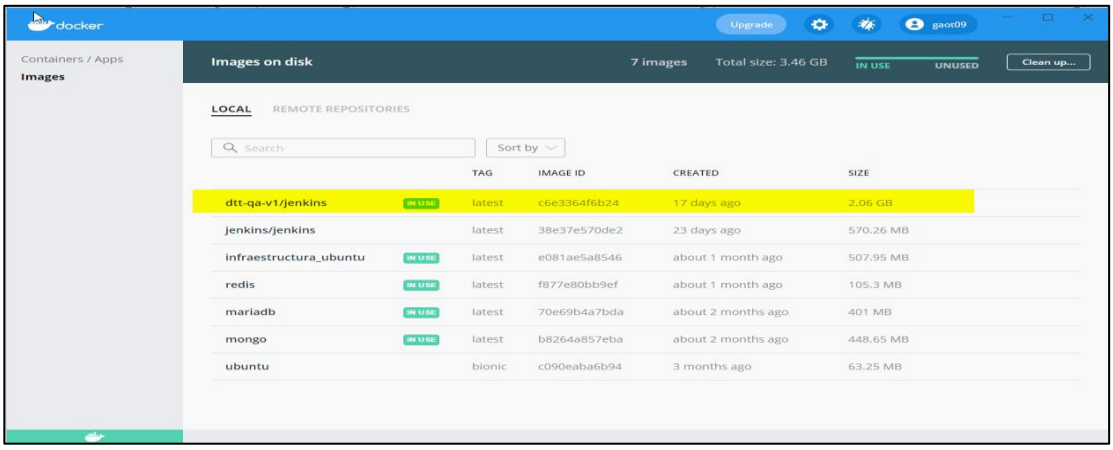
Fuente: elaboración propia.

1.7. Visualización de imagen base por medio del interfaz Docker

A partir de tener creada la imagen base para nuestro proyecto, procedemos a construir el contenedor en Docker. Para ello nos basaremos en la utilización de otro archivo.

Continuación apéndice 5.

Figura 7. **Visualización de imagen base por medio del interfaz Docker**



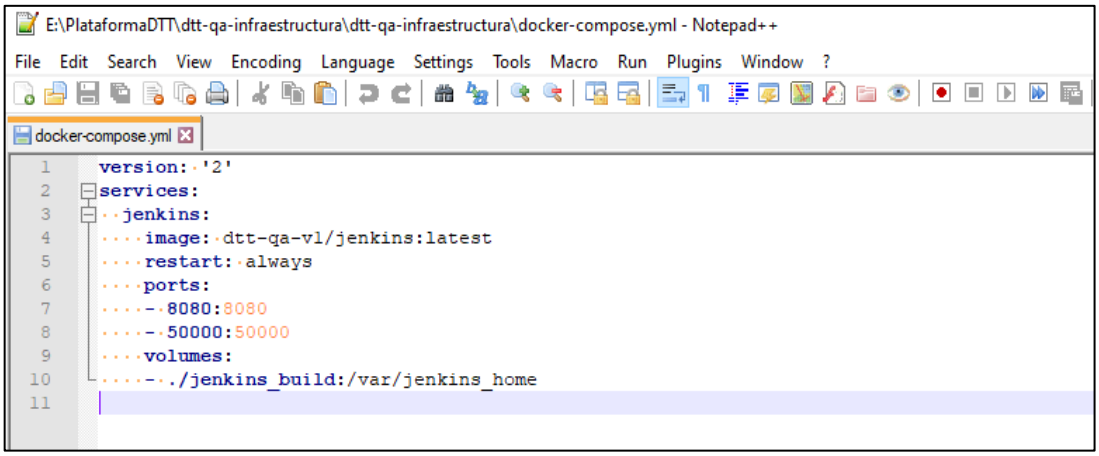
Fuente: elaboración propia.

1.8. Archivo docker-compose.yml para construir el contenedor

Este archivo, tiene algunos aspectos importantes a tomar en cuenta: El primero, consiste en el nombre que le vamos a poner al contenedor, en este caso “dtt-qa-v1/Jenkins:latest”. El segundo punto es habilitar siempre los puertos para que por medio del navegador (browser) en nuestro equipo local, podamos interactuar con Jenkins y así poder configurarlo. Y el tercer punto (el más importante) es definir la etiqueta de “Volumes”, el cual nos va permitir mapear todo lo que está en la carpeta jenkins_home de nuestro contenedor a una carpeta de nuestro equipo local, permitiendo que al modificar algún archivo en la carpeta de nuestro equipo local, el contendor pueda replicar automáticamente los cambios, o bien podamos transportar este contenedor como una imagen posteriormente.

Continuación apéndice 5.

Figura 8. **Archivo docker-compose.yml para construir el contenedor**



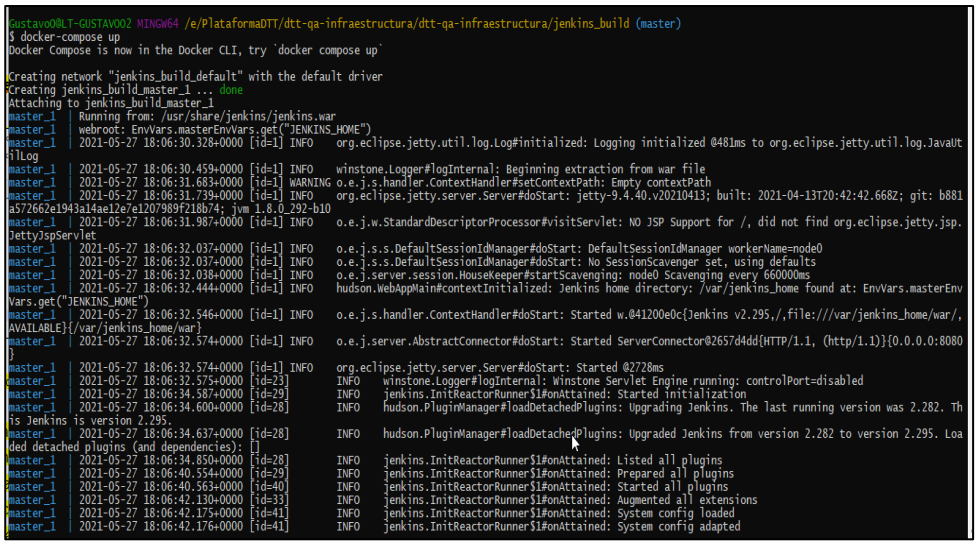
Fuente: elaboración propia.

1.9. Construcción del contenedor Docker por medio de archivo YML

A continuación, visualizaremos el comando para levantar un contenedor a partir del archivo de configuración YML. Teniendo como principal requisito, la imagen base activa dentro de nuestro Docker local. Es importante exponer los puertos 8080 para poder acceder desde nuestro navegador (requisito importante tener instalado un navegador) para continuar con la configuración y posterior utilización de la herramienta Jenkins.

Continuación apéndice 5.

Figura 9. **Construcción del contenedor Docker por medio de archivo YML**



Fuente: elaboración propia.

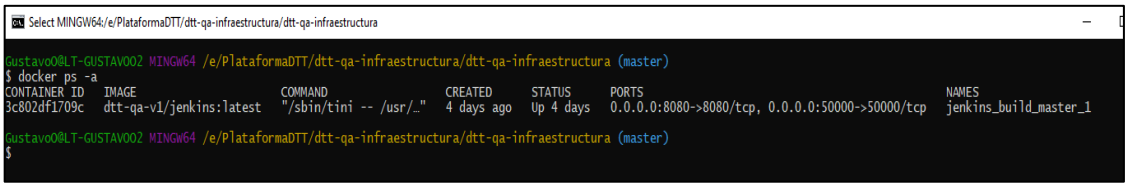
1.10. Visualización del contenedor Docker por medio de consola

Con el comando: “docker-compose up” Se indica a Docker que se construya el contenedor a partir de la imagen descrita en el archivo, para ello es importante ejecutar este comando en la carpeta donde se encuentre el archivo docker-compose.yml con la estructura como vimos anteriormente.

Luego de que termine la construcción del contenedor, lo podemos visualizar por medio de la interfaz gráfica o bien la consola con el comando “docker ps -a “.

Continuación apéndice 5.

Figura 10. **Visualización del contenedor Docker por medio de consola**



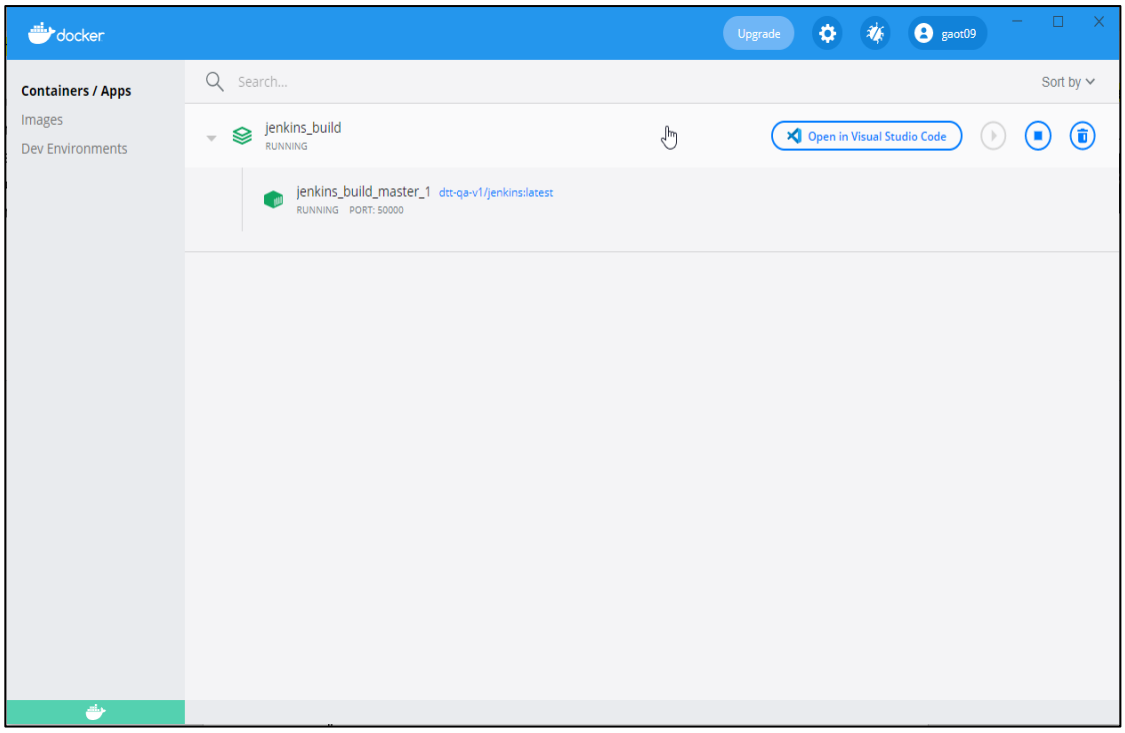
Fuente: elaboración propia.

1.11. Visualización del contenedor Docker por medio de interfaz gráfica

A partir de aquí, ya tenemos configurado el contendor (a partir de una imagen base con Jenkins) para su utilización. Lo siguiente que nos corresponde es configurar la herramienta Jenkins, así como construir los Jobs para su ejecución.

Continuación apéndice 5.

Figura 11. **Visualización del contenedor Docker por medio de interfaz gráfica**



Fuente: elaboración propia.

Continuación apéndice 5.

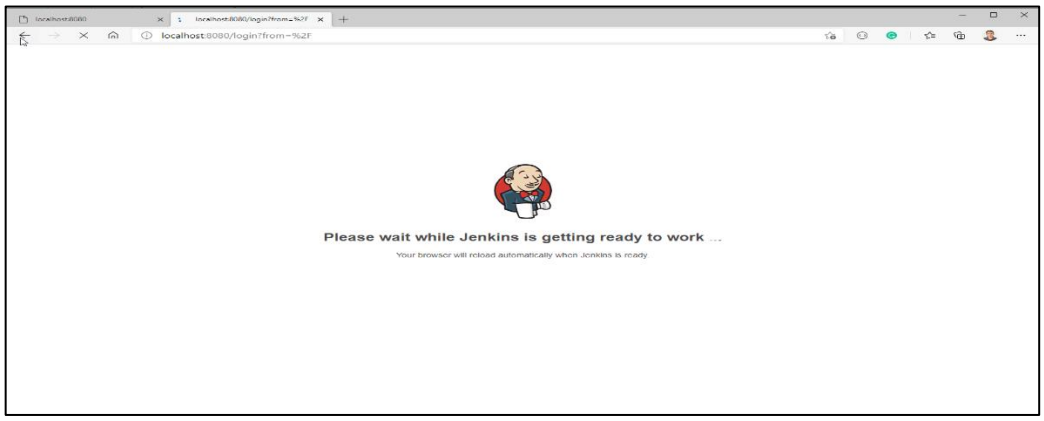
2. CONFIGURACIÓN DE JENKINS, PARA LA EJECUCIÓN DE PRUEBAS AUTOMÁTICAS

Luego de haber configurado nuestro contenedor, con el puerto 8080 habilitado, procedemos a ingresar en nuestro navegador la ip (127.0.0.1:8080) el cual nos permitirá acceder a nuestra herramienta de Jenkins que está instalada en nuestro contenedor.

2.1. Jenkins desde nuestro navegador

Tener en cuenta que para poder configurar Jenkins debemos tener instalado en nuestro equipo de computación un navegador por ejemplo Chrome o Firefox.

Figura 12. **Jenkins desde nuestro navegador**



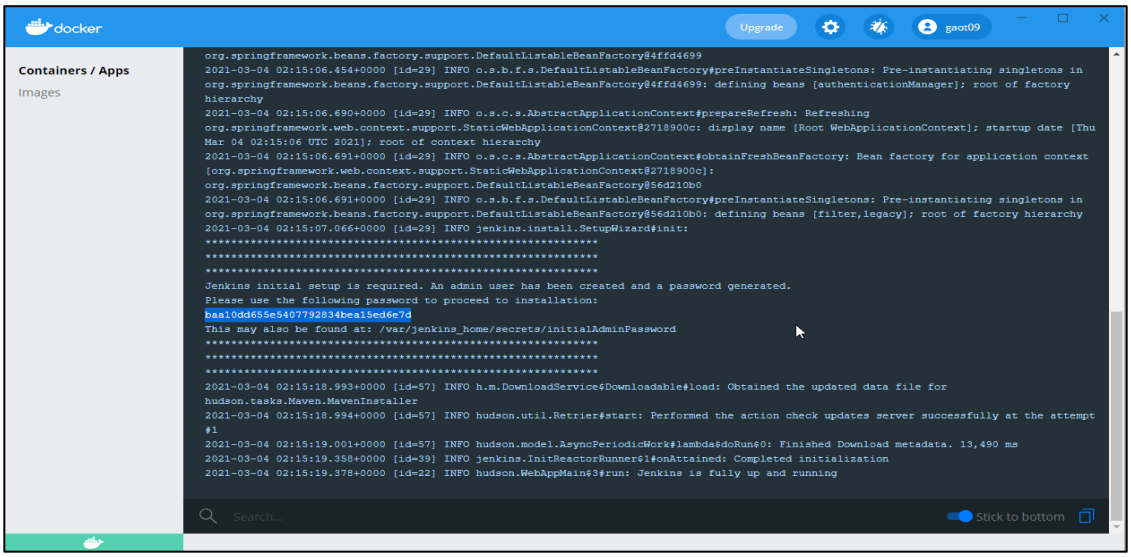
Fuente: elaboración propia.

Continuación apéndice 5.

2.2. Búsqueda de credenciales temporales

Luego de cargar la página, nos pedirá credenciales de autenticación para su ingreso, cabe mencionar que estas credenciales son temporales y se pueden obtener de la consola de ejecución del contenedor, como podemos observar en las siguientes imágenes.

Figura 13. **Búsqueda de credenciales temporales**



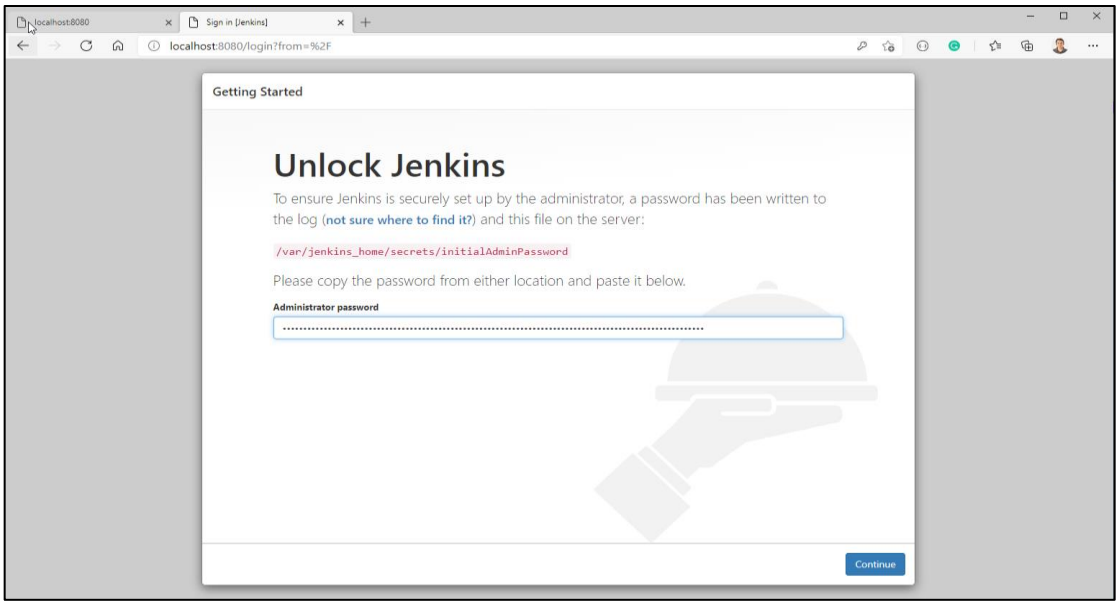
Fuente: elaboración propia.

Continuación apéndice 5.

2.3. Ingresar a partir de las credenciales temporales

Se nos despliega la ventana con la contraseña encriptada y se empieza a cargar la siguiente sección para su configuración, es importante tener en cuenta que al escribir mal la contraseña no se podrá avanzar a la siguiente vista.

Figura 14. **Ingresar a partir de las credenciales temporales**



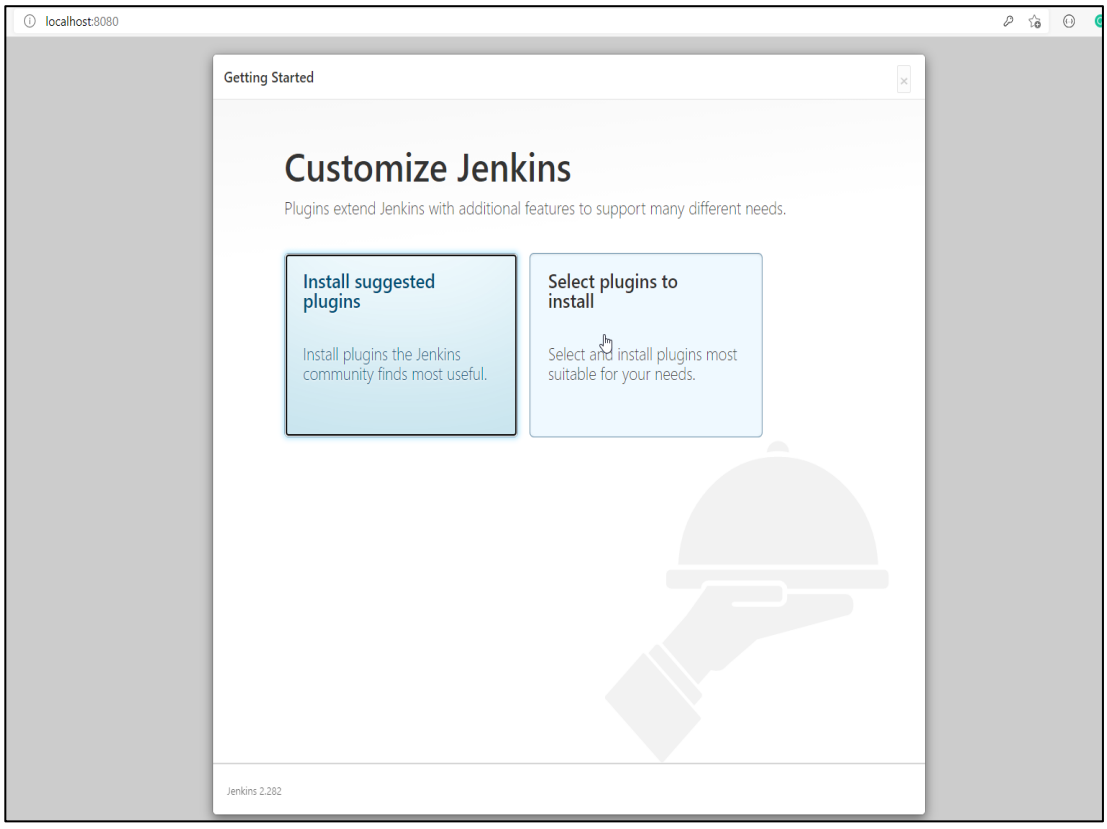
Fuente: elaboración propia.

Continuación apéndice 5.

2.4. Vista plugins a instalar

Posterior de la ventana de autenticación, para iniciar la configuración de Jenkins necesitamos seleccionar la opción de Instalar plugins, esta opción nos permitirá que Jenkins automáticamente busque e instale los plugins necesarios para su funcionamiento.

Figura 15. **Vista plugins a instalar**



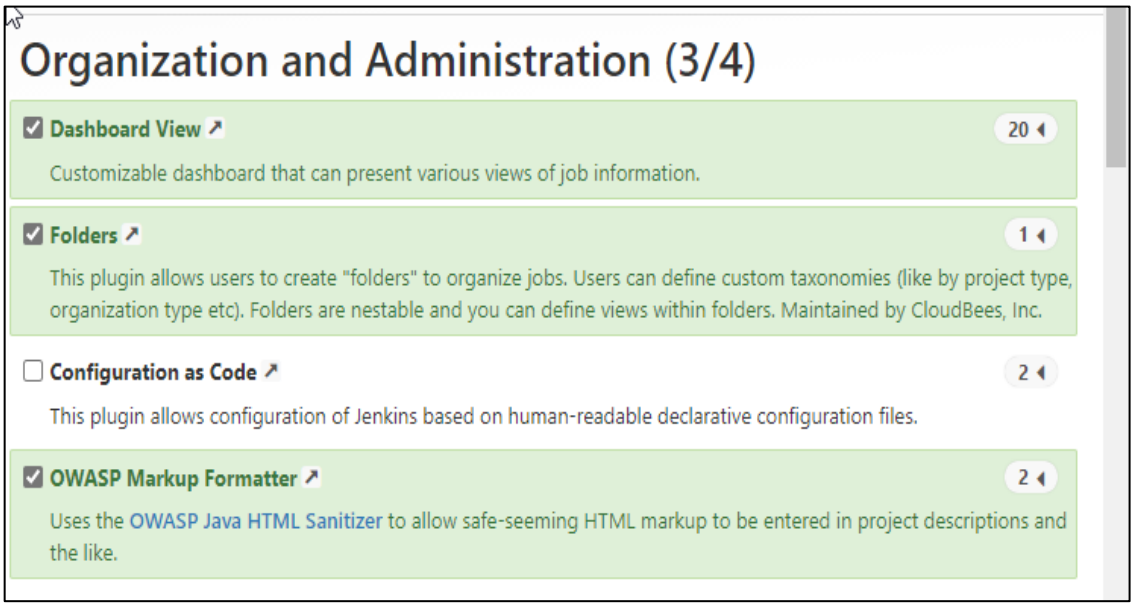
Fuente: elaboración propia.

Continuación apéndice 5.

2.5. Selección de plugins vista administración

En las imágenes a continuación se muestran que plugins necesitamos que se seleccionen para su instalación y un funcionamiento básico de Jenkins y con ello se puedan ejecutar las pruebas automáticas sin inconvenientes. Queta a criterio siempre de cada uno el seleccionar otros plugins que permitan una mejor administración y notificación al momento de ejecutar los Jobs y la notificación del resultado de estos.

Figura 16. **Selección de plugins vista administración**



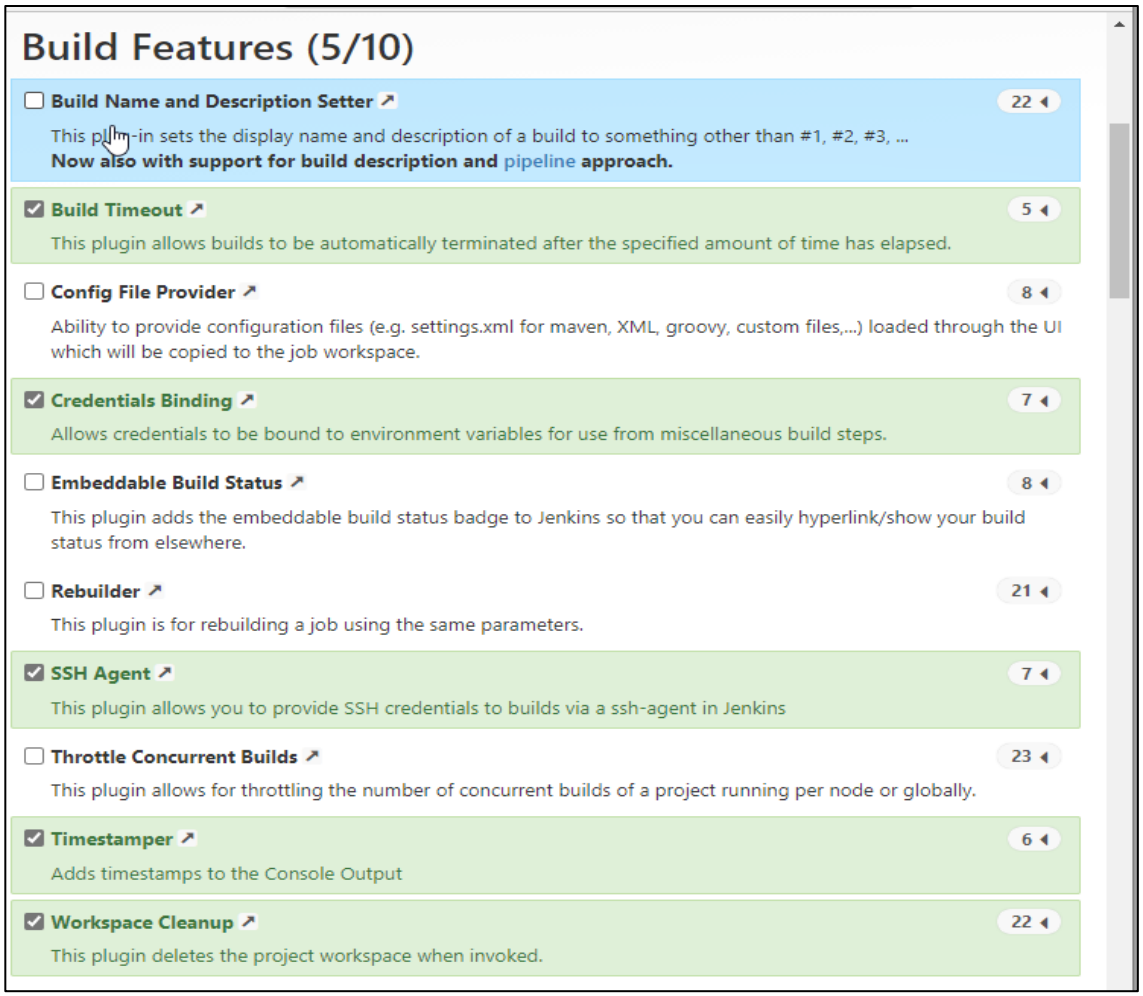
Fuente: elaboración propia.

Continuación apéndice 5.

2.6. Selección de plugins vista construcción

Esta vista nos ofrece varias herramientas a tener en cuenta para la instalación en Jenkins, algunas de estas ya la herramienta las selecciona como sugeridas para un funcionamiento básico de Jenkins.

Figura 17. **Selección de plugins vista construcción**



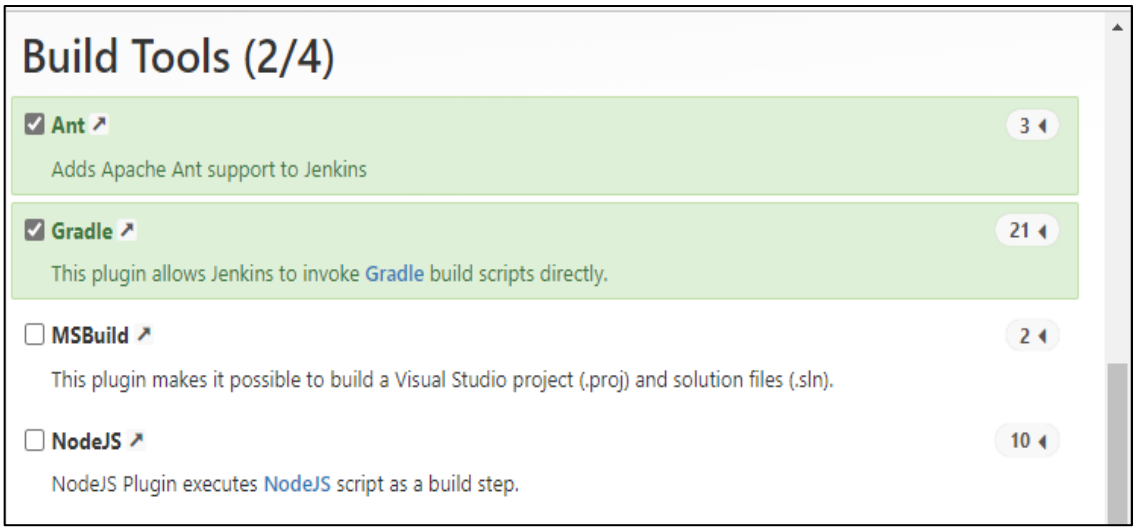
Fuente: elaboración propia.

Continuación apéndice 5.

2.7. Selección de plugins vista herramientas

Aquí seleccionamos principalmente Gradle que es el lenguaje de codificación que utilizan Jenkins en los archivos JenkinsFile para su programación y ejecución.

Figura 18. **Selección de plugins vista herramientas**



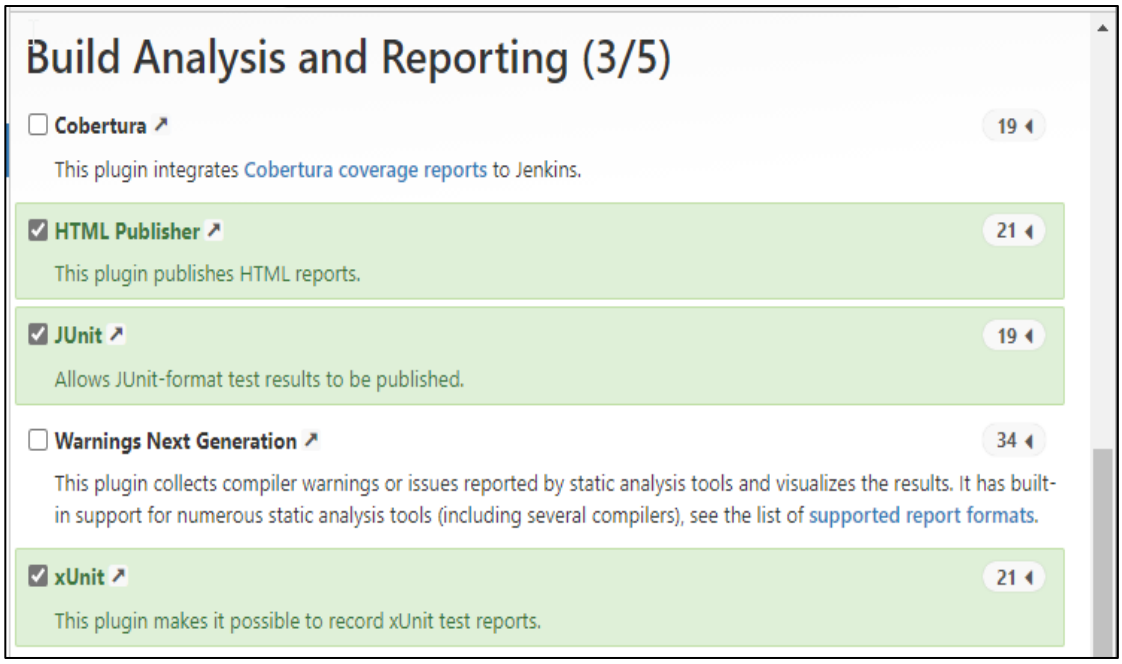
Fuente: elaboración propia.

Continuación apéndice 5.

2.8. Selección de plugins vista reportes

Se seleccionan algunas herramientas para los reportes a generar, principalmente HTML por la utilización del explorador en la utilización de Jenkins como principal despliegue o salida de datos.

Figura 19. **Selección de plugins vista reportes**



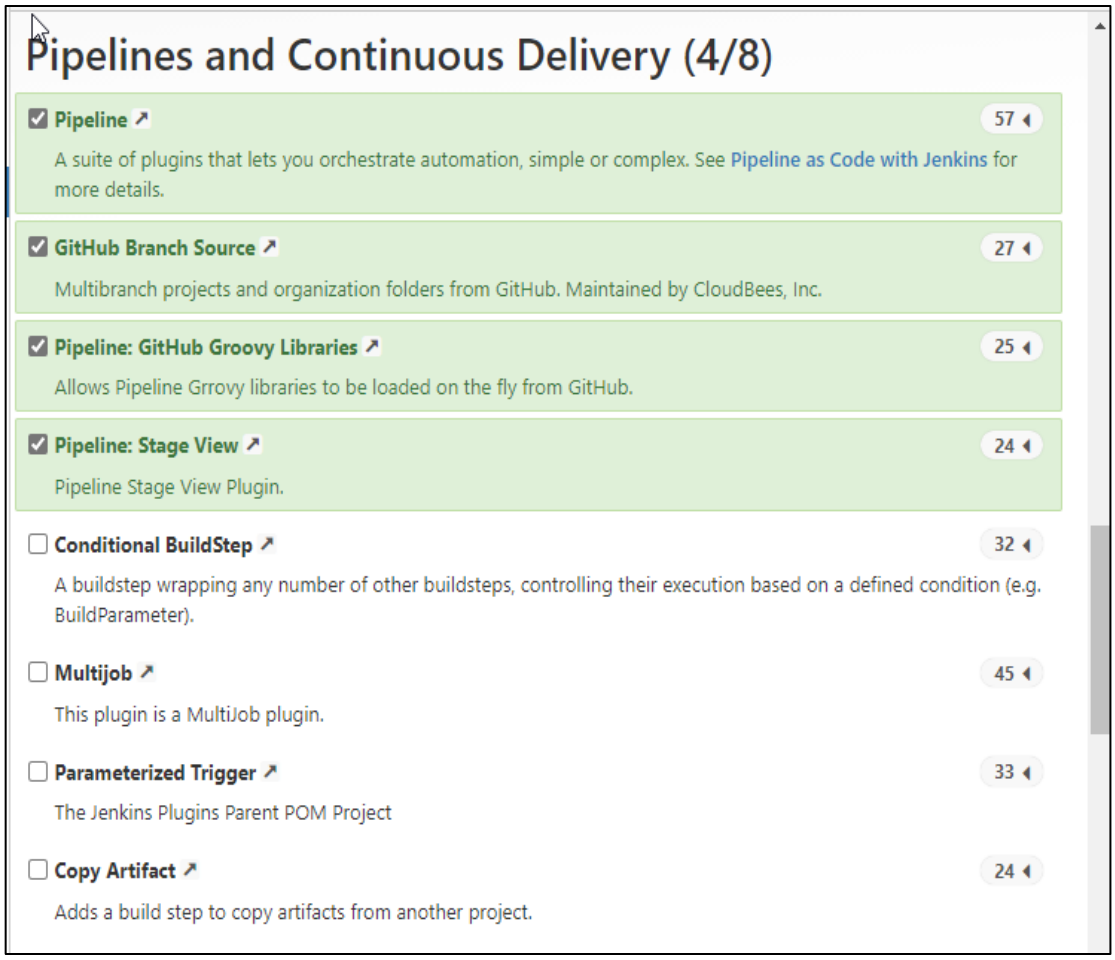
Fuente: elaboración propia.

Continuación apéndice 5.

2.9. Selección de Plugins vista despliegue

En esta vista, configuramos que los Jobs que se ejecutan en Jenkins se realizaran principalmente por archivos de configuración (JenkinsFiles) por ello dejamos seleccionado los principales plugins que podamos utilizar.

Figura 20. **Selección de Plugins vista despliegue**



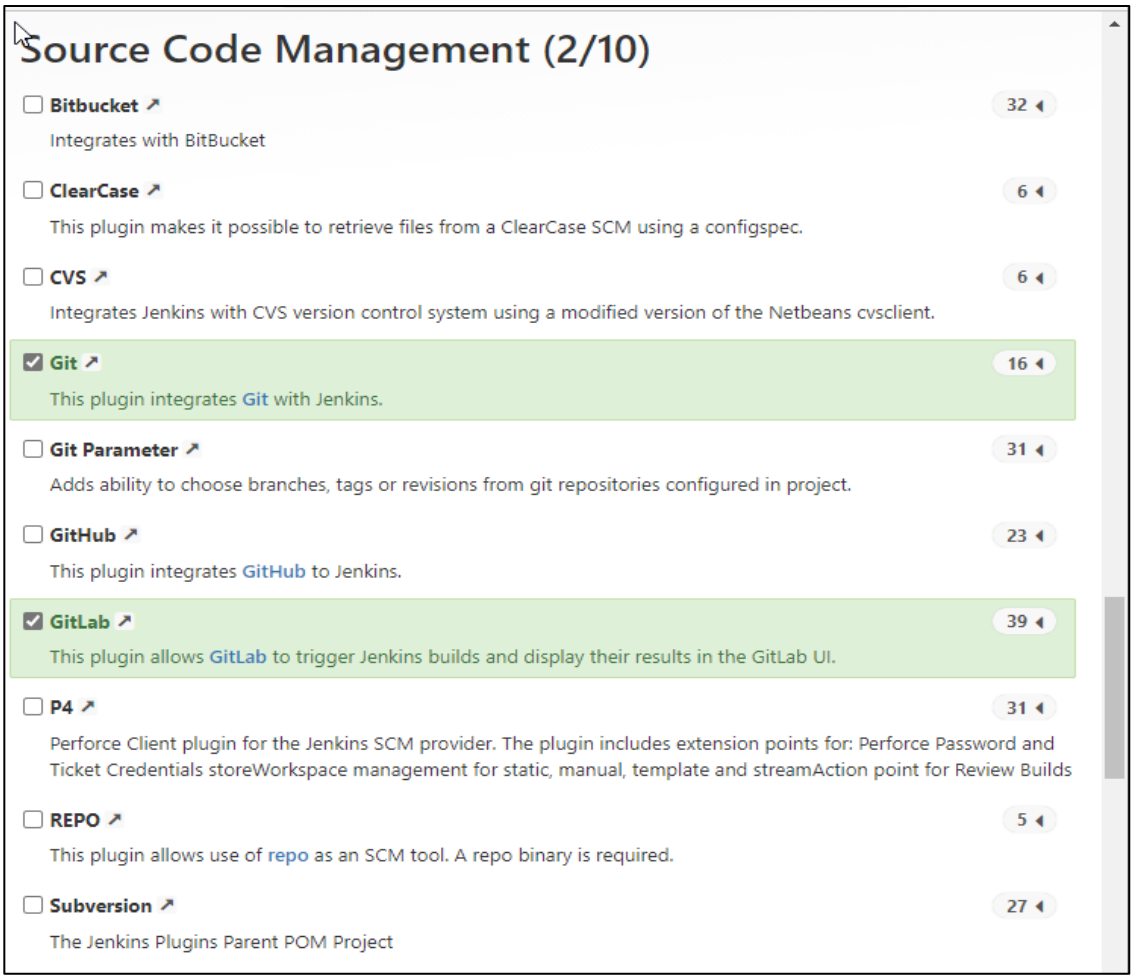
Fuente: elaboración propia.

Continuación apéndice 5.

2.10. Selección de Plugins vista repositorios

Como parte de las buenas prácticas es el uso de repositorios de versionado de software, por lo cual para conectar Jenkins con estos seleccionamos los correspondientes que utilizamos en el proyecto.

Figura 21. **Selección de Plugins vista repositorios**



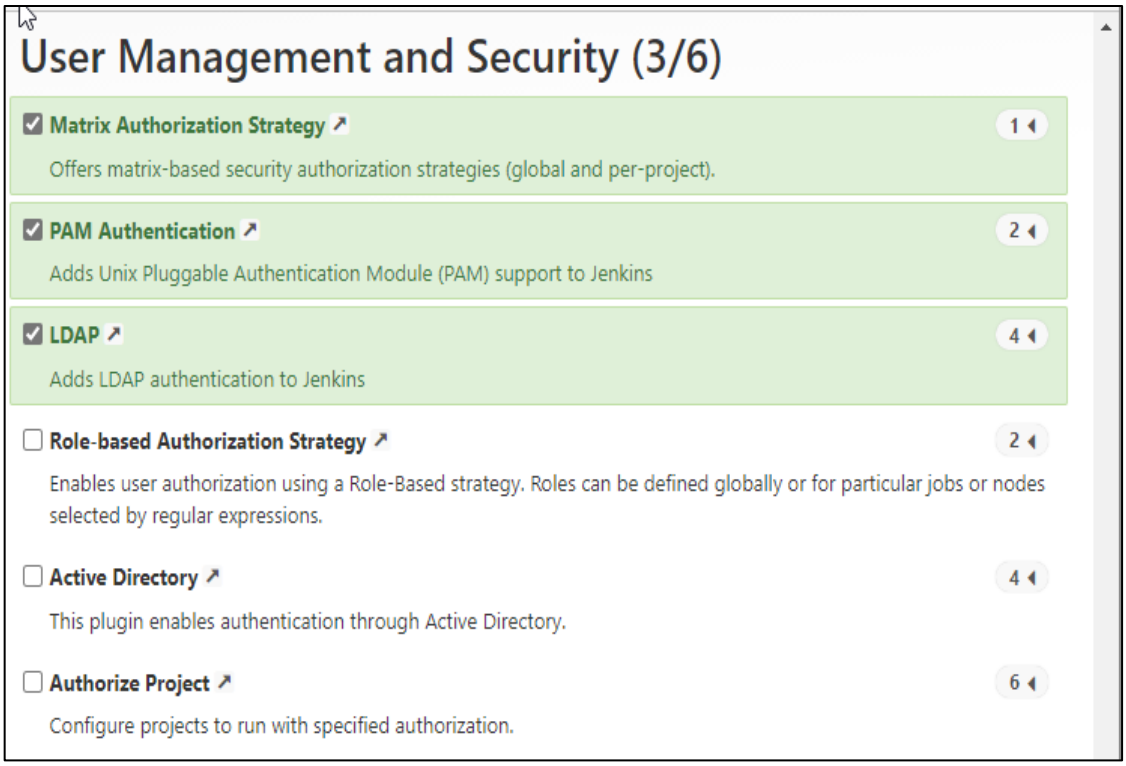
Fuente: elaboración propia.

Continuación apéndice 5.

2.11. Selección de Plugins vista seguridad

Siempre buscando mantener seguro nuestra codificación y el uso de esta herramienta, Jenkins nos proporciona herramientas para nuestra seguridad, por default se dejan preseleccionados los sugeridos.

Figura 22. **Selección de Plugins vista seguridad**



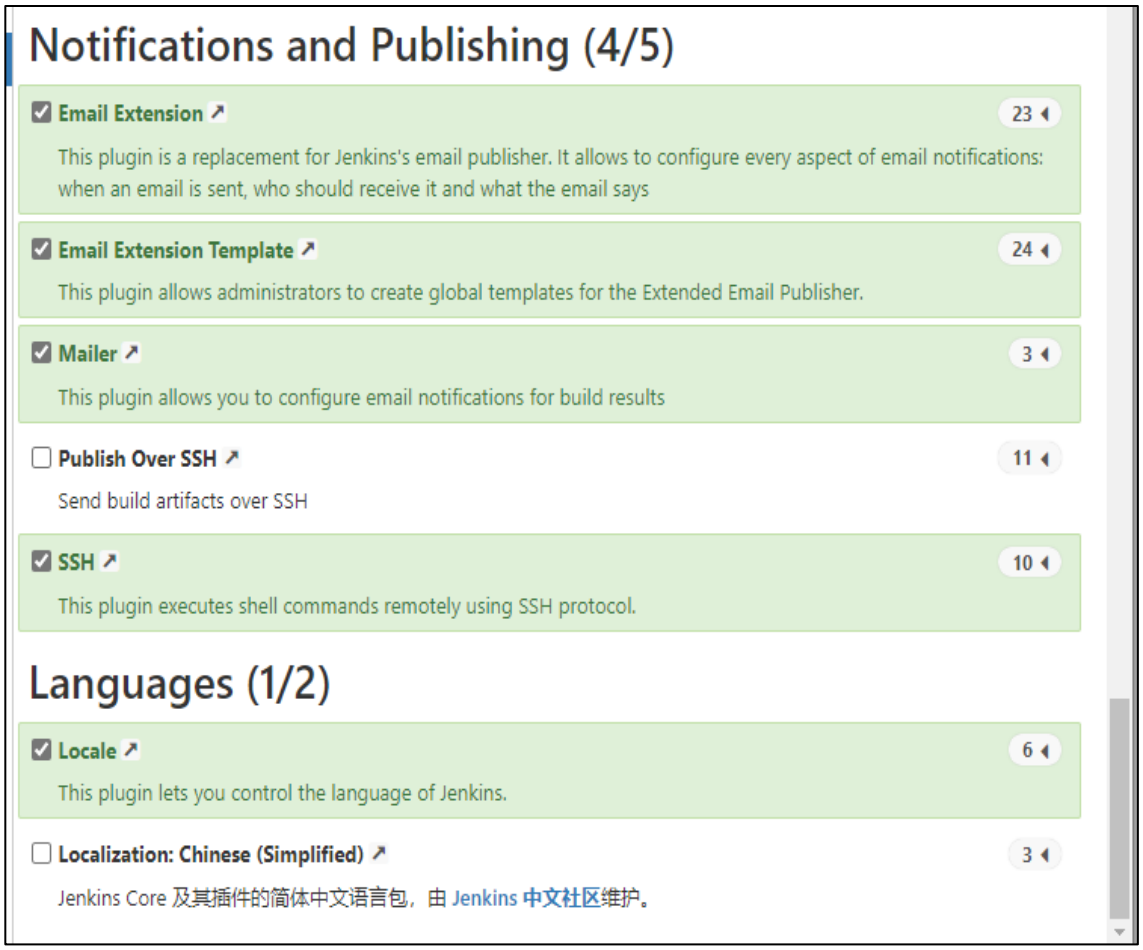
Fuente: elaboración propia.

Continuación apéndice 5.

2.12. Selección de Plugins vista notificaciones

Para el tema de motivación seleccionamos los plugins relacionados a Email o correos electrónicos, para más adelante poder configurar esta herramienta de notificación.

Figura 23. **Selección de Plugins vista notificaciones**



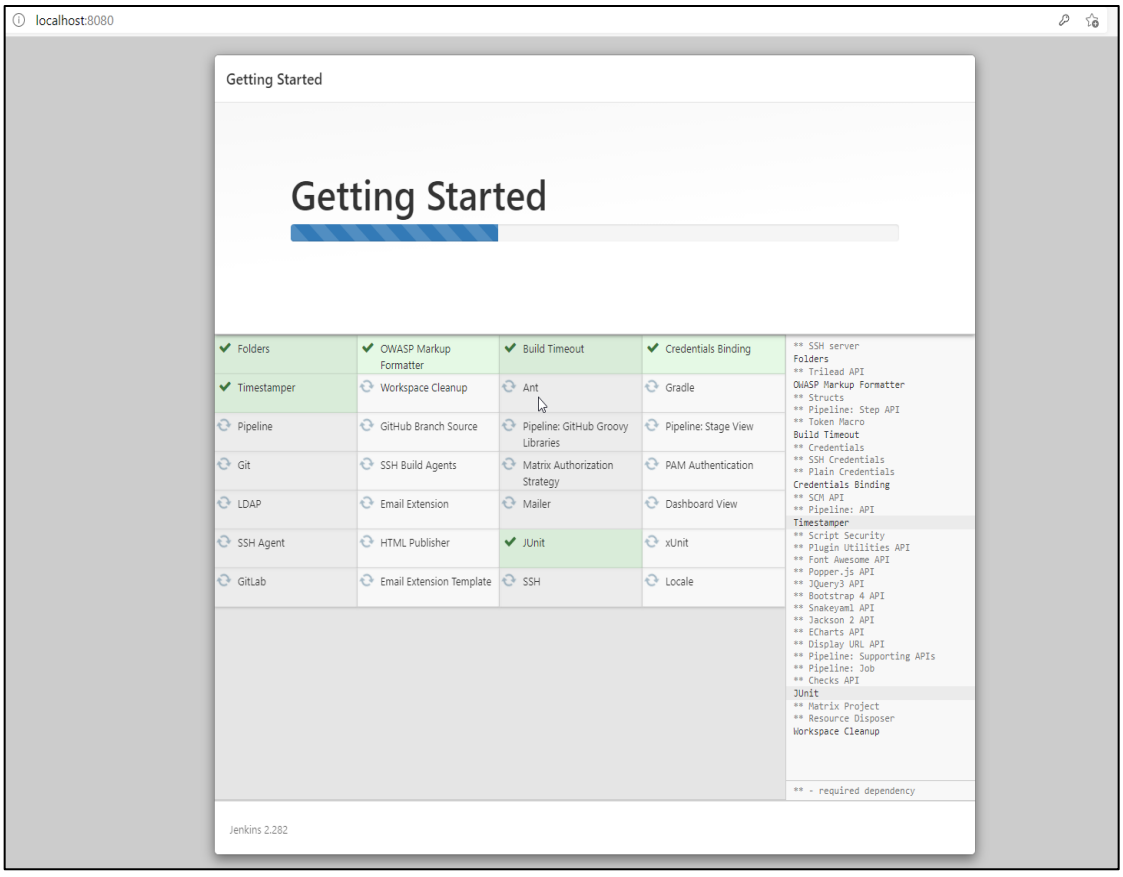
Fuente: elaboración propia.

Continuación apéndice 5.

2.13. Instalando plugins para Jenkins

Luego de haber seleccionado los diferentes plugins a instalarse, Jenkins realiza la acción de instalarlos, desplegando una vista donde se puede ir observando cada uno de los plugins instalados.

Figura 24. **Instalando plugins para Jenkins**



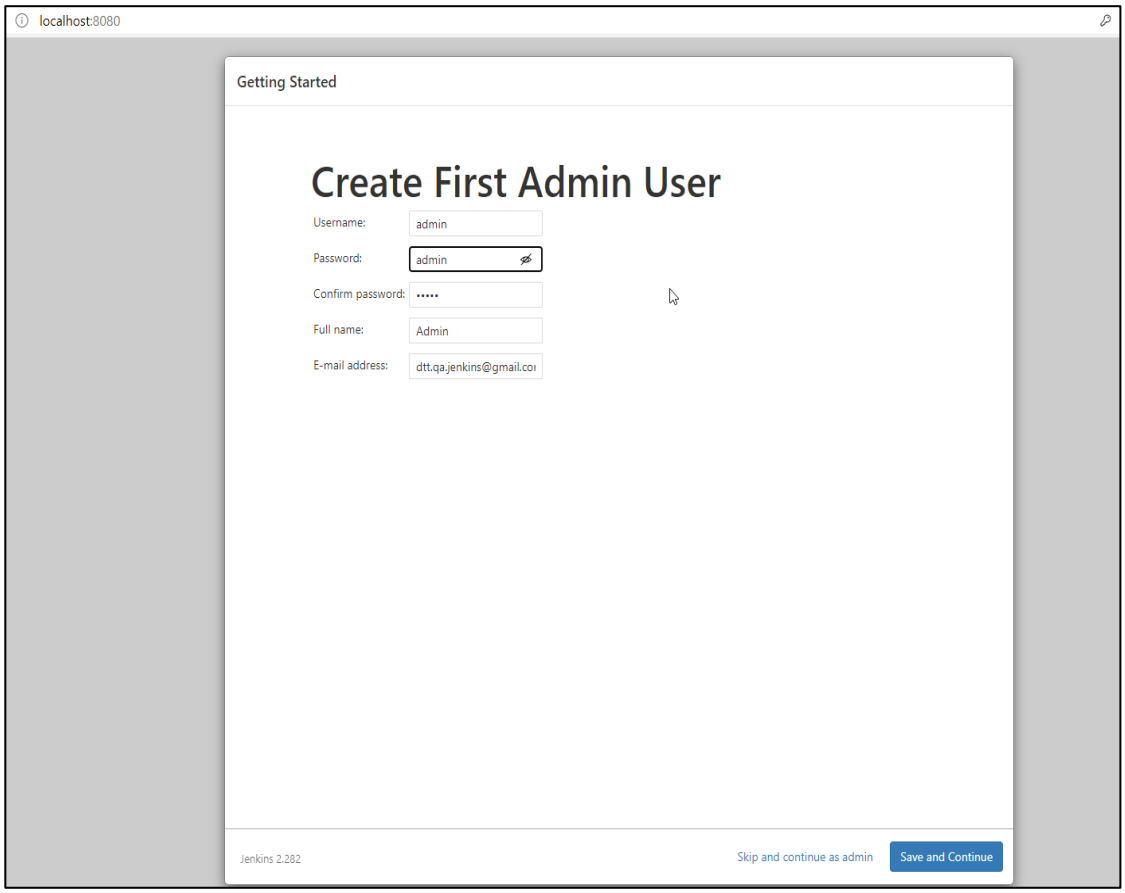
Fuente: elaboración propia.

Continuación apéndice 5.

2.14. Crear nuevas credenciales para Jenkins

Posterior a la instalación de los plugins, la siguiente vista que se despliega es la de ingresar las nuevas credenciales de autenticación (este usuario es el usuario administrador) que Jenkins necesita para su ingreso por primera vez en la herramienta para la creación y ejecución de Jobs.

Figura 25. **Crear nuevas credenciales para Jenkins**



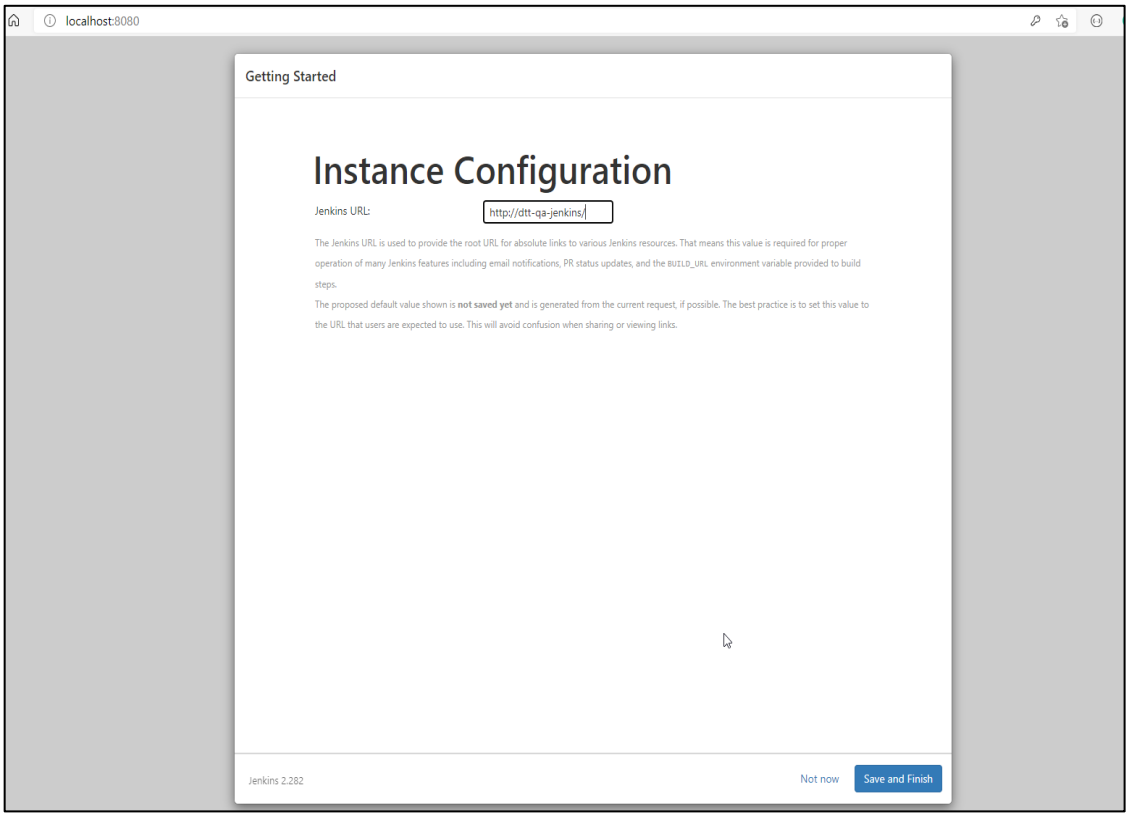
Fuente: elaboración propia.

Continuación apéndice 5.

2.15. Configuración de dominio para Jenkins local

La siguiente vista que se nos presenta es poder configurar el dominio local (computadora) para su ejecución de Jenkins, y no estar ingresando por ip.

Figura 26. **Configuración de dominio para Jenkins local**



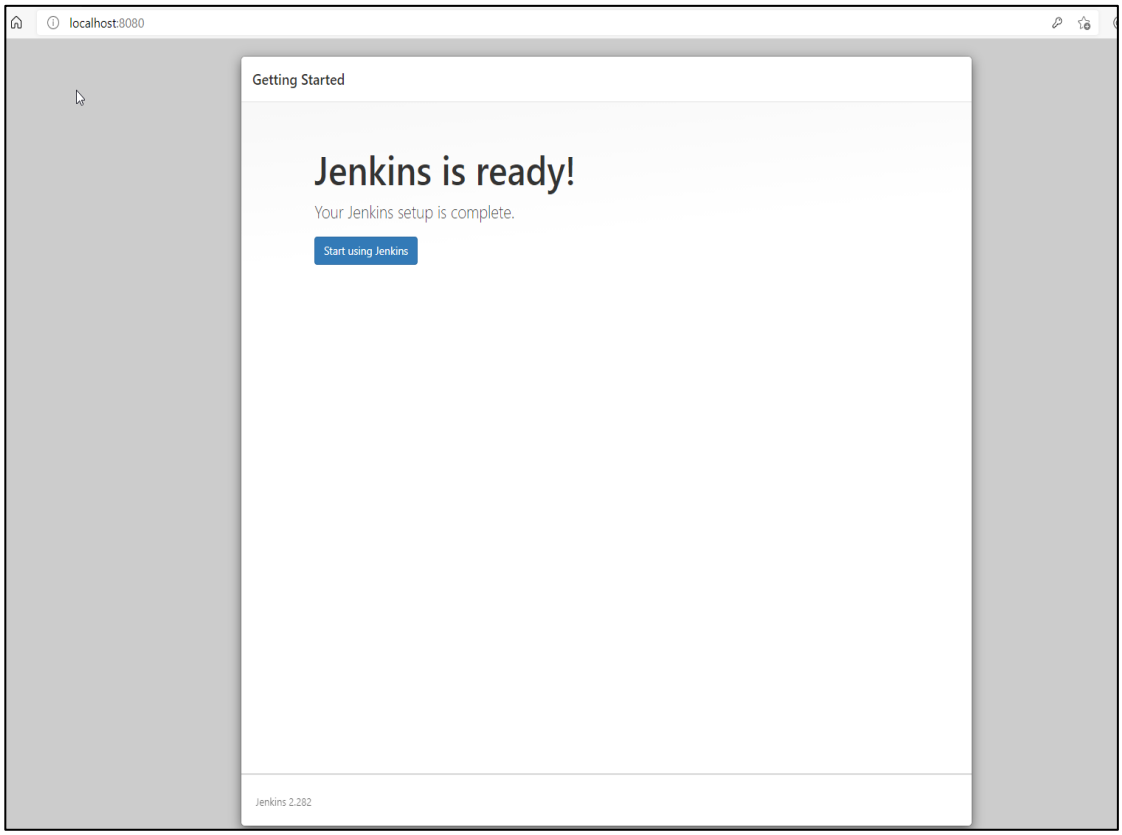
Fuente: elaboración propia.

Continuación apéndice 5.

2.16. Jenkins configurado correctamente

Y si todo esta correctamente bien configurado e instalado, se nos despliega la vista de finalización exitosa.

Figura 27. **Jenkins configurado correctamente**



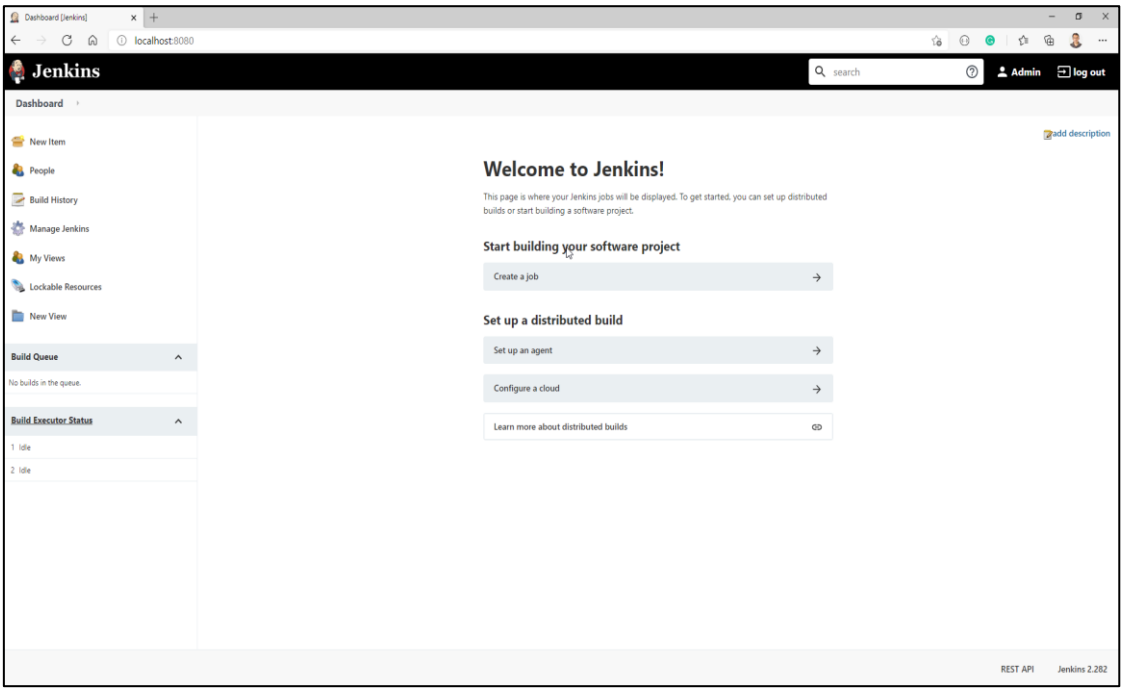
Fuente: elaboración propia.

Continuación apéndice 5.

2.17. Ingresando a Jenkins por primera vez

Con las credenciales configuradas, se realiza el ingreso por primera vez ya a Jenkins el cual mostrara una vista (como lo muestra la siguiente imagen), en esta vista el usuario puede realizar diversas acciones, como lo es crear un Job, una carpeta, o configurar la herramienta Jenkins.

Figura 28. **Ingresando a Jenkins por primera vez**



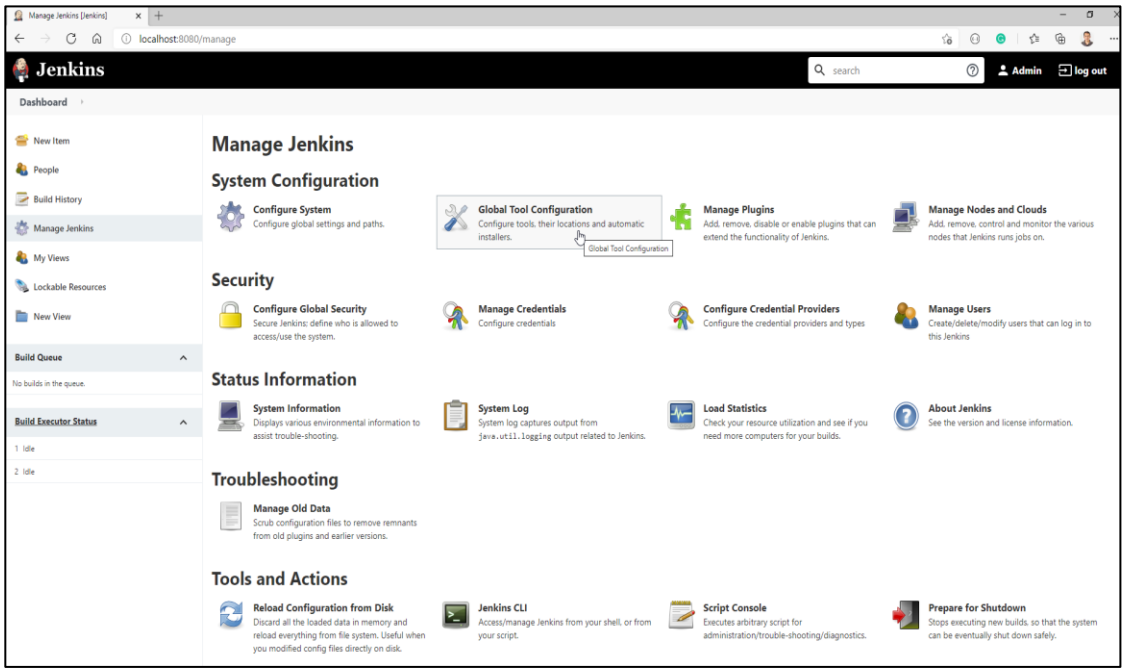
Fuente: elaboración propia.

Continuación apéndice 5.

2.18. Jenkins vista de configuración

Ya para finalizar, lo que nos falta para que Jenkins funcione correctamente en nuestro ambiente local, es configurar Java, Maven y el envió de correos electrónicos, para ello buscamos la opción de Administración o Manage Jenkins y luego en la opción de Configuración de Herramientas o Gobla Tool Configurations.

Figura 29. **Jenkins vista de configuración**



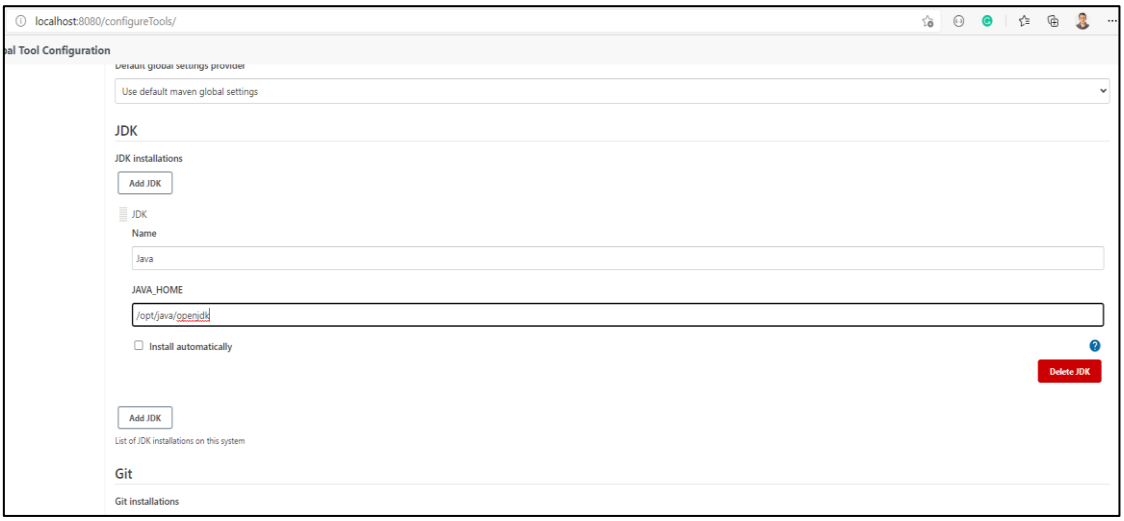
Fuente: elaboración propia.

Continuación apéndice 5.

2.19. Configurando Java en Jenkins

En la opción de herramientas de configuración, buscamos la parte de configurar JDK, aquí colocamos la ruta donde está instalado Java en nuestro equipo local (dependiendo del sistema operativo se coloca la misma ruta que usamos para configurar nuestra variable de entorno JAVA_HOME).

Figura 30. **Configurando Java en Jenkins**



Fuente: elaboración propia.

Continuación apéndice 5.

2.20. Configurando Maven en Jenkins

También buscamos la opción de Maven, al haber seleccionado la instalación del plugin de Maven, esta opción se nos despliega y posterior configuramos la ruta (similar a la ruta de nuestra variable de entorno MAVEN_HOME).

Figura 31. **Configurando Maven en Jenkins**

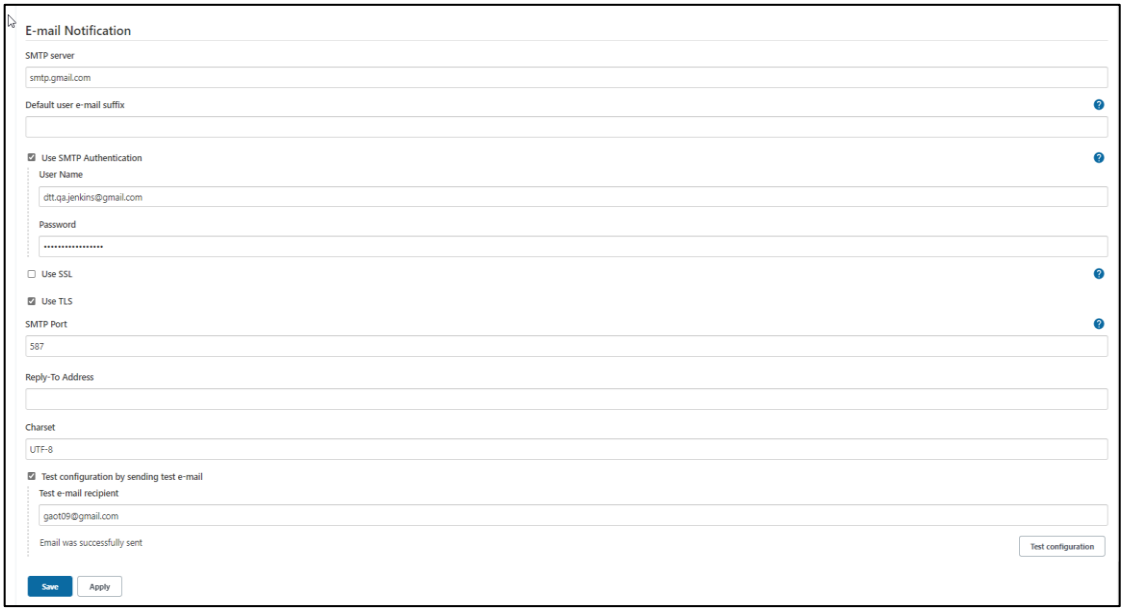
Fuente: elaboración propia.

Continuación apéndice 5.

2.21. Configurando correo electrónico en Jenkins

Para finalizar la configuración, en la parte de notificación, debemos ingresar nuestro servidor SMTP (para fines de practica usamos el de Google que es gratuito), llenamos los demás campos requeridos y con ello realizamos una prueba para enviar correos y tendremos listo nuestra configuración.

Figura 32. **Configurando correo electrónico en Jenkins**



Fuente: elaboración propia.

Aquí finaliza el presente manual, el cual tiene como finalidad ayudar al estudiante a poder configurar de forma local (computadora) su entorno de ejecución de pruebas automáticas. Permitiendo crear una imagen de Docker y poder ejecutar su contenedor correspondiente, para posteriormente configurar Jenkins como herramienta principal para fines de este proyecto de automatización.

Fuente: elaboración propia.