# Capítulo I: Preliminares

## Portada

Instituto Tecnológico José Mario Molina Pasquel y Henríquez Campus Chapala

Ing. Sistemas Computacionales



QA Automation Initiative

Pedro Solórzano Gómez

Carmen Leticia Salcedo Quevedo

El Salto Jalisco a, a 30 de octubre de 2020.

## Agradecimientos

## Resumen

## Índice

[Capítulo I: Preliminares 1](#_Toc58411767)

[Portada 1](#_Toc58411768)

[Agradecimientos 2](#_Toc58411769)

[Resumen 3](#_Toc58411770)

[Índice 4](#_Toc58411771)

[Capítulo II: Generalidades del proyecto 6](#_Toc58411772)

[Introducción 6](#_Toc58411773)

[Descripción de la empresa u organización y del puesto o área de trabajo del estudiante 7](#_Toc58411774)

[Problemas a resolver. 8](#_Toc58411775)

[Objetivos (Generales y específicos) 9](#_Toc58411776)

[Objetivo General 9](#_Toc58411777)

[Objetivos Específicos 9](#_Toc58411778)

[Justificación 10](#_Toc58411779)

[Capítulo III: Marco teórico 11](#_Toc58411780)

[Marco teórico 11](#_Toc58411781)

[ Integración Continua 11](#_Toc58411782)

[ Jenkins 11](#_Toc58411783)

[ JavaScript 11](#_Toc58411784)

[ Mocha 12](#_Toc58411785)

[ Apiary 12](#_Toc58411786)

[ Dredd 12](#_Toc58411787)

[ Perl 12](#_Toc58411788)

[ Selenium 13](#_Toc58411789)

[ DynamoDB 13](#_Toc58411790)

[ Allure Report 13](#_Toc58411791)

[ Axios 13](#_Toc58411792)

[ Chai 13](#_Toc58411793)

[ Git 14](#_Toc58411794)

[ GitHub 14](#_Toc58411795)

[ Slack 15](#_Toc58411796)

[ JSON Schema 15](#_Toc58411797)

[ Antora 15](#_Toc58411798)

[Capitulo IV: Desarrollo 16](#_Toc58411799)

[Procedimiento y descripción de las actividades realizadas 16](#_Toc58411800)

[ Configuración Prueba Dredd. 16](#_Toc58411801)

[ Implementar prueba Dredd dentro de Jenkins. 16](#_Toc58411802)

[ Mudar trabajo previamente creado a un proyecto tipo Pipeline en Jenkins 16](#_Toc58411803)

[ Implementar un Framework de pruebas para el desarrollo de pruebas de API 17](#_Toc58411804)

[ Implementación de reportes a los resultados de las pruebas antes desarrolladas 17](#_Toc58411805)

[Capítulo V: Resultados 18](#_Toc58411806)

[Resultados, planos, gráficas, prototipos, manuales, programas, análisis estadísticos, modelos matemáticos, simulaciones, normatividades, regulaciones y restricciones, entre otros 18](#_Toc58411807)

[Actividades sociales realizadas en la empresa u organización 19](#_Toc58411808)

[Capítulo VI: Conclusiones 20](#_Toc58411809)

[Conclusiones de proyecto, recomendaciones y experiencia personal profesional adquirida 20](#_Toc58411810)

[Capitulo VII: Competencias desarrolladas 21](#_Toc58411811)

[Competencias desarrolladas y/o aplicadas 21](#_Toc58411812)

[Capitulo VIII: Fuentes de información 22](#_Toc58411813)

[Fuentes de información 22](#_Toc58411814)

[Referencias 22](#_Toc58411815)

[Capitulo IX: Anexos 23](#_Toc58411816)

# Capítulo II: Generalidades del proyecto

## Introducción

## Descripción de la empresa u organización y del puesto o área de trabajo del estudiante

Green Marimba I / O es una agencia de contratación de personal y subcontratación de Tecnologías de la Información centrada en los servicios en la nube de AWS, la automatización de pruebas y todas las áreas de ciberseguridad.

Green Marimba I / O fue fundada por un equipo experimentado de ejecutivos de Tecnologías de la Información que comprenden los desafíos de administrar el personal de TI remoto. Con sede en Boulder, Colorado, y oficinas en Los Ángeles y México.

El área en la que estuve realizando mi estancia profesional fue en el área de QA (Control de Calidad), esta área es la encargada de realizar pruebas al código desplegado por parte de los desarrolladores.

En esta área se hace el desarrollo de pruebas automatizadas, desde pruebas de Backend como pueden ser pruebas de API’s hasta pruebas automatizadas de Frontend como lo son pruebas de interfaz de usuario.

## Problemas a resolver.

En el transcurso de la residencia surgieron varios problemas y conforme avanzaba el tiempo y me iba familiarizando más con las herramientas y metodologías que utilizaban pude empezar a resolverlos. Los siguientes problemas fueron los que se me presentaron en la realización del proyecto de residencia:

* Dificultad con el lenguaje de programación utilizado.
* Tecnologías desconocidas para la realización del proyecto.
* Desconocimiento de las diferentes etapas del desarrollo de software.

## Objetivos (Generales y específicos)

## Objetivo General

* Desarrollar un Framework de control de calidad automatizado para los lenguajes Java y JavaScript que soporte la generación de reportes además de la integración de procesos de CI/CD.

## Objetivos Específicos

* Desarrollar una infraestructura en la nube, así como un Framework que soporte los lenguajes de programación: Java y JavaScript.
* Integrar procesos de desarrollo de tipo CI/CD a la infraestructura anteriormente mencionada.
* Automatizar la generación de reportes acorde a los resultados de ejecución de pruebas realizadas.

## Justificación

El presente proyecto permitirá a los usuarios integrar pruebas complejas de Control de Calidad en procesos de CI/CD en dos lenguajes de programación diferentes (Java y JavaScript). Con la posibilidad de generar reportes automatizados con los resultados de dichas pruebas ejecutadas.

Esta herramienta podrá ser implementada en cualquier proyecto con fines de automatizar las pruebas de calidad siempre y cuando cumpla con el tipo de lenguaje anteriormente mencionado.

Integrando desde pruebas de Backend utilizando un framework de pruebas como lo es Mocha. Hasta pruebas de Frontend utilizando Selenium web driver.

# Capítulo III: Marco teórico

## Marco teórico

En esta sección se describirán las herramientas o practicas utilizadas a lo largo del desarrollo de dicho proyecto, se añadirá una pequeña descripción de la herramienta además de una explicación del uso de la herramienta en el proyecto.

### Integración Continua

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización. Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software. (Amazon Web Services, 2020).

### Jenkins

Jenkins es un servidor automatizado de integración continua de código abierto y actualmente uno de los más utilizados por su fácil manejo y extensa integración con diferentes tecnologías. Además de su amplio catálogo de complementos que se pueden descargar directamente de su web.

En el proyecto se utilizó Jenkins como servidor en el proceso de Integración Continua; Se creo un pipeline para integrar desde la obtención del código fuente del proyecto desde el repositorio de control de versiones en GitHub como las pruebas realizadas para Frontend y Backend, hasta el despliegue de la aplicación en AWS.

### JavaScript

JavaScript es un lenguaje de programación de lado del cliente, utilizado mayormente para el desarrollo web, no requiere de compilación ya que los navegadores son los encargados de interpretar el código lo que lo hace multiplataforma.

JavaScript fue el lenguaje que se utilizó para las pruebas automatizadas realizadas, se utilizó este lenguaje debido a su fácil integración con el proyecto y a otras herramientas necesarias para la elaboración de las pruebas, como el framework de pruebas y a sus diferentes estilos de aserción, así como también su sencilla integración con la herramienta utilizada para el reporte de las pruebas.

### Mocha

Mocha es un framework de pruebas para JavaScript ejecutado en Node.js y el navegador, con este framework es posible hacer pruebas asíncronas y síncronas de una manera simple. Además de ofrecer muchas utilidades para la ejecución y reporte de las pruebas.

Este fue el framework de pruebas utilizado para la construcción de las pruebas, se utilizó este framework por su fácil integración con otras librerías de JavaScript además de su capacidad para crear pruebas asíncronas, y su integración con la librería de aserción utilizada.

### Apiary

Apiary es un editor para APIs que permite usar tanto API Blueprint como Swagger, los cuales son lenguajes que permiten describir nuestras APIs, y gracias a ellos se puede realizar algunas tareas como simular una API de forma sencilla y rápida.

Apiary no solo permite editar texto como cualquier otro editor, sino que también ayuda a pasar pruebas a nuestra API empleando Dredd. También analiza las peticiones con su inspector, adema permite previsualizar el documento final y compila constantemente con los últimos cambios dejando ver errores al validarlo todo. (Campos Romero, 2020)

Apiary fue la herramienta que se utilizó para hacer la documentación de la API. Se utilizo esta herramienta por su facilidad para documentar, y por la facilidad que tiene esta para integrarse con otra herramienta de pruebas de documentación de APIs. Otra de las razones por la que se utilizó esta herramienta es por su facilidad de simular el comportamiento de la API sin tener que programarla antes.

### Dredd

Dredd es una herramienta para la validación de documentos de descripción de APIs. Dredd revisa el documento de descripción de la API y paso a paso valida que funcione de la manera en que se especifica en el documento de descripción. Dredd soporta dos formatos de documentos de descripción: API Blueprint y Open API 2.

Se utilizo Dredd como herramienta para verificar el documento de descripción de API por su gran compatibilidad con Apiary y por su fácil ejecución de pruebas, tan solo utilizando unos comandos.

### Perl

Perl es un lenguaje de programación, utilizado mayormente para procesamiento de texto, para escanear archivos de texto y extraer información de ellos.

En la elaboración del proyecto se utilizó Perl para realizar un script, puesto que se necesitaba un pre procesador de texto que inyectara texto obtenido de un fichero en específico dentro de otro fichero.

### Selenium

Selenium es un conjunto de herramientas disponibles en varios lenguajes de programación (Java, Python, C#, Ruby, JavaScript, Kotlin). Estas herramientas sirven para controlar el navegador web, simulando la interacción de un usuario con una página web. Dichas herramientas son utilizadas para la realización de pruebas automatizadas.

Para el desarrollo de las pruebas automatizadas de la Interfaz de Usuario se utilizó Selenium WebDriver, ya que esta herramienta es la más utilizada para dicho propósito, además de su buena integración con otras herramientas necesarias para la correcta ejecución de las pruebas.

### DynamoDB

Amazon DynamoDB es una base de datos de clave-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos duradera de varias regiones y con varios maestros, completamente administrada, que cuenta con copia de seguridad, restauración y seguridad integradas, y almacenamiento de caché en memoria para aplicaciones a escala de Internet. DynamoDB puede gestionar más de 10 billones de solicitudes por día y puede admitir picos de más de 20 millones de solicitudes por segundo. (Amazon Web Services, 2020)

Este fue la base de datos elegida para la elaboración del proyecto puesto que Amazon Web Services es el servicio de computación en la nube preferido de la empresa.

### Allure Report

Allure Report es una herramienta flexible, ligera y multilenguaje para el proceso de elaboración de los reportes de pruebas, ofrece reportes gráficos muy claros sobre la ejecución de las pruebas.

Allure fue la herramienta para la publicación de los reportes de pruebas utilizada durante el proyecto, por su sencilla incorporación con Jenkins el servidor de integración continúa elegido para la ejecución de este proyecto.

### Axios

Axios es una librería JavaScript que puede ejecutarse en el navegador y que nos permite hacer sencillas las operaciones como cliente HTTP basada en promesas, por lo que podremos configurar y realizar solicitudes a un servidor y recibiremos respuestas fáciles de procesar. (Baquero García, 2020).

Se utilizo Axios como la librería específica para la ejecución de peticiones HTTP a los diferentes endpoints de la API a la que se le realizaron las pruebas, se eligió esta librería por su compatibilidad con promesas.

### Chai

Chai.js es una librería de JavaScript que ofrece tres tipos diferentes de aserciones para la captura de resultados en las pruebas, además de ofrecer compatibilidad con cualquier framework de pruebas de JavaScript.

Para las aserciones en los casos de pruebas utilizamos esta librería puesto que ofrece tres diferentes tipos de aserción y una excelente compatibilidad con el framework de pruebas utilizado.

### Git

Git es una herramienta de control de versiones, el propósito de esta herramienta es llevar un registro de los cambios que se realizaron en cada fichero del proyecto y quien realiza dichos cambios.

Para los proyectos de tecnologías de la información es indispensable utilizar una herramienta para llevar a cabo el control de versiones, puesto que es importante tener un control de cada cambio que se realiza en el proyecto. Por esta razón se utilizó Git para el control de versiones además de ser la herramienta más utilizada mundialmente para esta finalidad.

### GitHub

GitHub es una plataforma de **desarrollo colaborativo de software** para alojar proyectos utilizando el sistema de control de versiones Git.

GitHub aloja tu repositorio de código y te brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto.

En la actualidad, GitHub es mucho más que un servicio de alojamiento de código. Además de éste, se ofrecen varias herramientas útiles para el trabajo en equipo. Entre ellas, caben destacar:

* Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
* Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
* Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio. (Castillo, 2020)

GitHub fue la herramienta que más utilizamos en el proyecto, esta nos ayudó para organizarnos un poco más, separar el trabajo en pequeñas tareas y asignarlas a cada integrante del equipo, además de la función principal de ser el repositorio de código remoto para asegurarse que cada integrante del equipo tenga la versión más actualizada del proyecto y evitar el problema de que un integrante del equipo modifique líneas de código que otro integrante está necesitando en determinado momento.

### Slack

Slack es una plataforma de mensajería basada en canales. Con Slack, las personas pueden trabajar juntas de manera más efectiva, conectar todas sus herramientas y servicios de software y encontrar la información que necesitan para hacer su mejor trabajo, todo dentro de un entorno seguro de nivel empresarial. (Slack Technologies, 2020)

Esta fue la herramienta que utilizamos para contactarnos entre los miembros del equipo, esto debido al problema sanitario a nivel mundial que nos aqueja actualmente por lo cual la empresa tomo la decisión de adoptar la práctica de trabajar desde casa, por esta razón era necesario utilizar una herramienta que permitiera llevar a cabo una comunicación efectiva con cada miembro del equipo.

### JSON Schema

En la actualidad el intercambio de información entre sistemas es muy habitual, es muy común ver que el Backend sea programado con un lenguaje de programación diferente al utilizado en el Frontend por lo que es muy importante él envió de información entre estas dos caras del desarrollo.

Hoy en día se utiliza JSON como formato de intercambio de datos. Por lo que JSON Schema es una excelente herramienta que nos permite definir un esquema donde se especifique la estructura que debería llevar el archivo JSON.

Se utilizo JSON Schema para realizar unas pruebas de Backend, para revisar la respuesta que se obtiene de los diferentes endpoints de la API, se utilizó esta herramienta puesto que es más sencillo utilizar esta herramienta que verificar campo por campo de la respuesta.

### Antora

Antora es un generador de sitios estáticos (static-site) destinado, principalmente, a la documentación de proyectos con múltiples repositorios y/o versiones de desarrollo.

Antora permite centralizar la documentación de múltiples repositorios, así como utilizar diferentes ramas de cada uno de ellos. (Aguilera, 2020)

Se utilizo Antora para la generación de la documentación para el proyecto en el que se estuvieron realizando las pruebas.

### MQTT

MQTT (Message Queue Telemetry Transport), un protocolo usado para la comunicación machine-to-machine en el "Internet of Things". Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos empotrados con pocos recursos (CPU, RAM, …). (Yebenes, 2020)

Se realizo un proyecto donde era necesario la comunicación entre sensores para enviar sus diferentes lecturas a un servicio web. Aquí fue donde se utilizo el protocolo MQTT para la comunicación entre sensor e IoT Core.

# Capitulo IV: Desarrollo

## Procedimiento y descripción de las actividades realizadas

En esta sección se describirá detalladamente las etapas y actividades realizadas durante el proceso de residencia.

### Configuración Prueba Dredd.

Como introducción al proceso de pruebas automatizadas se comenzó realizando esta tarea la cual tiene como objetivo configurar la herramienta Dredd para la realización de pruebas automáticas a la API previamente desarrollada por el equipo de Backend.

La configuración de la prueba Dredd consiste en realizar un archivo de configuración donde se especifique la ubicación del documento de descripción del API, el endpoint donde esta alojada la API entre otros campos.

Después de construir dicho archivo de configuración solo necesitamos cargarlo en la prueba Dredd. La ejecución de esta prueba se realiza mediante la línea de comandos.

### Implementar prueba Dredd dentro de Jenkins.

Una vez que la prueba Dredd fue configurada correctamente de forma local, llega el momento de configurar la ejecución de esta prueba dentro del servidor de integración continua; Jenkins.

Para implementar la prueba en el servidor fue necesario crear un trabajo dentro de Jenkins y configurar un nuevo trabajo. Se configura el repositorio de GitHub donde se descargará todo el proyecto para trabajarlo dentro de Jenkins y se configura las credenciales correctas para poder tener acceso al repositorio.

Se instala la herramienta Dredd para poder realizar la prueba y justo después se configuran los pasos que dicho trabajo realizara para poder llevar a cabo la ejecución de la prueba Dredd dentro de nuestro servidor en Jenkins.

### Mudar trabajo previamente creado a un proyecto tipo Pipeline en Jenkins

Para esta actividad se debió de crear un nuevo trabajo en Jenkins, pero este debía ser de tipo pipeline. Hoy en día se utiliza mucho el concepto de pipeline para los procesos de integración continua, pues estos permiten automatizar el proceso ya que se pueden agregar las diferentes etapas del ciclo de vida del software y este proceso no avanzara a la siguiente etapa hasta completar con éxito la etapa anterior, lo que permite agregar pruebas antes del proceso de despliegue asegurándonos que el software siempre completa correctamente la etapa de pruebas antes de que este sea lanzado, y así automatizar este proceso.

La configuración de este tipo de trabajo fue diferente a la configuración realizada en el trabajo anterior, ya que para este tipo de trabajo las acciones a realizar se dividen en etapas y pasos, estos se especifican en un archivo llamado Jenkinsfile.

Este archivo debe existir en el repositorio pues el trabajo de Jenkins ira a buscar el archivo Jenkinsfile dentro del contenido del repositorio y comenzara a realizar las acciones que se especifiquen dentro de él.

### Implementar un Framework de pruebas para el desarrollo de pruebas de API

Una vez que las pruebas de Dredd ya estaban listas y se ejecutaban correctamente en el servidor de integración continua, se procedió a desarrollar pruebas específicas al API desarrollada en el Backend, para el desarrollo de estas pruebas se utilizo Mocha como Framework de pruebas y Chai como el método de aserción para estas pruebas.

Para esta tarea se crearon pruebas que verifiquen cada uno de los endpoints de la API haciendo peticiones a cada endpoint de una manera que se reciba una respuesta de aceptación, y se crea otra petición al mismo endpoint, pero con algún campo incompleto para que la respuesta sea de error.

Cualquiera de las dos respuestas es capturada y pasa por el proceso de comparación de acuerdo al tipo de prueba ejecutado. Estas pueden ser pruebas negativas; estas esperan una respuesta negativa por parte del servicio a probar y las pruebas positivas, que esperan una respuesta de aceptación.

### Implementación de reportes a los resultados de las pruebas antes desarrolladas

La siguiente tarea a realizar fue implementar una herramienta que nos permita convertir los resultados de las pruebas creadas en las anteriores tareas en reportes para ser más fácil la visualización de resultados.

Para esta actividad se utilizo la herramienta Allure Report, puesto que tiene gran integración con el servidor de integración continúa utilizado, Jenkins.

La integración de la prueba con los reportes de Allure fue algo complicado puesto que para una correcta integración de los reportes se necesitó integrar una librería al proyecto que hace la función de adaptador. El adaptador se encarga de producir un archivo con los resultados de la prueba en un formato que sea admitido por Allure, una vez integrado el adaptador se necesita especificar la ruta donde se generara el archivo con los resultados de las pruebas para posteriormente agregar una nueva etapa al pipeline donde se publicaran los resultados de las pruebas en el formato grafico que entrega Allure.

### Incorporación de prueba MQTT.

En esta tarea se adicionaron mas casos de pruebas, en este caso, pruebas mqtt. Esta prueba fue necesaria para verificar el correcto funcionamiento de la comunicación entre el dispositivo que obtiene las lecturas de los sensores y el IoT Core, uno de los servicios que ofrece Amazon Web Services.

El procedimiento en esta prueba fue el siguiente: primero se necesita crear un registro del dispositivo a activar en la base de datos, para esto se hace una petición al servicio web que se encarga de crear el registro, para realizar esta acción es importante especificar los campos requeridos por esta actividad en el cuerpo de la petición al servicio. Posteriormente se verifica que el registro persista en la base de datos, esto se realiza con otra petición al servicio web especificando el dispositivo que se necesita revisar, con esta petición se revisa que el dispositivo exista y que el dispositivo se encuentre desactivado puesto que aun no se procede mandar el mensaje mqtt que activa el dispositivo. Una vez que se verifique que el dispositivo se encuentra en la base de datos y se verifica que este esta desactivado se procede a enviar el mensaje mqtt que activa el dispositivo, este mensaje cuenta como parámetros el dispositivo a activar y un ID de transacción, esto para obtener un seguimiento de las transacciones que se realizan en el dispositivo. Acto seguido del mensaje mqtt se procede a realizar otra petición al servicio web, especificando de nueva cuenta el dispositivo que se especifico en el mensaje mqtt y se verifica que este ahora se encuentre activo de igual manera se verifica que el ID de transacción sea el especificado en la comunicación mqtt previamente realizada.

### Incorporación de pruebas de Schema

H

# Capítulo V: Resultados

## Resultados, planos, gráficas, prototipos, manuales, programas, análisis estadísticos, modelos matemáticos, simulaciones, normatividades, regulaciones y restricciones, entre otros

## Actividades sociales realizadas en la empresa u organización

# Capítulo VI: Conclusiones

## Conclusiones de proyecto, recomendaciones y experiencia personal profesional adquirida

# Capitulo VII: Competencias desarrolladas

## Competencias desarrolladas y/o aplicadas

# Capitulo VIII: Fuentes de información

## Fuentes de información

### Referencias

Aguilera, J. (8 de Diciembre de 2020). *Soy Jorge Aguilera*. Obtenido de Soy Jorge Aguilera: https://jorge.aguilera.soy/blog/2020/intro-antora.html

Allure. (2 de Diciembre de 2020). *AllureQaTools*. Obtenido de AllureQaTools: http://allure.qatools.ru/

Alvarez Corredera, P. (22 de Mayo de 2020). *ciberninjas*. Obtenido de ciberninjas: https://ciberninjas.com/jenkins/

Alvarez, C. (8 de Diciembre de 2020). *genbeta*. Obtenido de genbeta: https://www.genbeta.com/desarrollo/utilizando-json-schema

Amazon Web Services. (2 de Diciembre de 2020). *AWS*. Obtenido de AWS: https://aws.amazon.com/es/dynamodb/

Amazon Web Services. (23 de Noviembre de 2020). *aws.amazon*. Obtenido de aws.amazon: https://aws.amazon.com/es/devops/continuous-integration/

Baquero García, J. (2 de Diciembre de 2020). *arsys*. Obtenido de arsys: https://www.arsys.es/blog/programacion/axios/

Campos Romero, H. (2 de Diciembre de 2020). *adictosaltrabajo*. Obtenido de adictosaltrabajo: https://www.adictosaltrabajo.com/2017/08/17/documenta-tu-api-con-apiary/

Castillo, L. (3 de Diciembre de 2020). *Conociendo GitHub*. Obtenido de Conociendo GitHub: https://conociendogithub.readthedocs.io/en/latest/data/introduccion/

Chaijs. (2 de Diciembre de 2020). *Chaijs*. Obtenido de Chaijs: https://www.chaijs.com/

Dredd. (2 de Diciembre de 2020). *Dredd*. Obtenido de Dredd: https://dredd.org/en/latest/

Garzas, J. (9 de Mayo de 2014). *JavierGarzas*. Obtenido de JavierGarzas: https://www.javiergarzas.com/2014/05/jenkins.html

janpoloy. (2 de Diciembre de 2020). *@janpoloy*. Obtenido de @janpoloy: https://janpoloy.medium.com/qu%C3%A9-es-y-para-que-sirve-git-3fd106e6e137

Mocha. (2 de Diciembre de 2020). *Mocha*. Obtenido de Mocha: https://mochajs.org/

Perl. (2 de Diciembre de 2020). *PerlDoc*. Obtenido de PerlDoc: https://perldoc.perl.org/perl

Selenium. (2 de Diciembre de 2020). *Selenium*. Obtenido de Selenium: https://www.selenium.dev/documentation/

Slack Technologies. (3 de Diciembre de 2020). *Centro de ayuda: Slack*. Obtenido de Slack: https://slack.com/intl/es-mx/help/articles/115004071768-%C2%BFQu%C3%A9-es-Slack-

Yebenes, J. A. (9 de Diciembre de 2020). *geekytheory*. Obtenido de geekytheory: https://geekytheory.com/que-es-mqtt

# Capitulo IX: Anexos